інформатика

Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина

ДИСКРЕТНА МАТЕМАТИКА

ПІДРУЧНИК

ДЛЯ ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ

bhv ПИТЕР

О. В. НІКОЛЬСЬКИЙ, В. В. ПАСІЧНИК, Ю. М. ЩЕРБИНА

ДИСКРЕТНА МАТЕМАТИКА

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, які навчаються за напрямами «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика та управління», «Комп'ютерна інженерія», «Прикладна математика»

Серія «ІНФОРМАТИКА»
За загальною редакцією академіка НАН України
М. З. Згуровського

Київ
Видавнича група BHV
2007

Рецензенти: *Приймак М. В.*, завідувач кафедри комп'ютерних наук Тернопільського державного технічного університету ім. Івана Пулюя, доктор технічних наук, професор.
Цегелик Г. Г., завідувач кафедри математичного моделювання соціально-економічних процесів Львівського національного університету ім. Івана Франка, доктор фізико-математичних наук, професор.

Гриф надано Міністерством освіти і науки України, лист № 14/18.2-333 від 13.02.2006 р.

Нікольський Ю. В., Пасічник В. В., Щербина Ю. М.

Н64 Дискретна математика. — К.: Видавнича група BHV, 2007. — 368 с.: іл.
ISBN 966-552-201-9

У підручнику в логічній послідовності викладено основні поняття та методи дискретної математики. Окрім таких розділів, як теорія множин і математична логіка, теорія графів, основи теорії кодування, теорія булевих функцій, теорія алгоритмів та формальних мов, які традиційно входять до базового курсу дисципліни, розглянуто також основи теорії складності обчислень та деякі застосування дискретної математики у штучному інтелекті.

За змістом та обсягом підручник відповідає навчальним планам дисципліни «Дискретна математика» для студентів базових напрямів «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика та управління», «Комп'ютерна інженерія» та «Прикладна математика».

ББК 22.176я73

Серія підручників «Інформатика» виходить у світ за підтримки видавництв «Издательский дом „Питер"», м. Санкт-Петербург, та «Освітня книга», м. Київ.

ISBN 966-552-201-9

Стислий зміст

Зміст

Передмова

Підручник „Дискретна математика", який пропонуємо до уваги студентів і викладачів вищих навчальних закладів III та IV рівнів акредитації базових напрямів "Комп'ютерні науки", "Комп'ютеризовані системи, автоматика і управління", "Комп'ютерна інженерія" та "Прикладна математика", увібрав багаторічний досвід викладання зазначеної дисципліни в провідних університетах та інститутах Західноукраїнського регіону. Він призначений для підготовки сучасного висококваліфікованого фахівця в галузі комп'ютерних наук із застосуванням теоретичного й алгоритмічного багатства дискретної математики.

У підручнику, який складається з 10 розділів, послідовно розкрито базові теми дисципліни. Усі розділи книги логічно взаємопов'язані. Багато питань із розглянутих у підручнику недостатньо висвітлено в сучасній навчальній літературі. Зокрема, це стосується нових інформаційних технологій, які ґрунтуються на наведених у підручнику алгоритмах. Значну увагу автори приділили термінологічній базі дисципліни, застосуванню уніфікованого підходу до формування системи позначень.

Автори дотримувалися певної схеми викладення матеріалу, яка дає змогу побудувати вивчення предмету за зростанням складності. Усі твердження супроводжено ілюстративними прикладами. Кожний розділ закінчується набором ретельно підібраних контрольних запитань та завдань. Практичний матеріал значний за обсягом (становить близько 20 % загального обсягу підручника). Він двох типів: вправи для закріплення й поглиблення розуміння теоретичних положень і завдання на виконання комп'ютерних проектів. Значна їх частина розширює введені поняття, дає змогу глибше зрозуміти обґрунтування чи надати уявлення про практичне застосування матеріалу в різних галузях знань, зокрема штучному інтелекті, математичній лінгвістиці, програмуванні, дослідженні операцій, системному аналізі, теорії прийняття рішень тощо. На думку авторів, це суттєво сприятиме поліпшенню організації самостійної роботи студентів.

У підручнику зібрано як загальновідомий матеріал, який уже став класикою дискретної математики, так і розроблений останніми роками. Цю проблематику висвітлено здебільшого в небагатьох доступних для фахівців спеціалізованих виданнях і недостатньо методично опрацьовано в україномовних підручниках. Автори намагалися поєднати строгість викладення матеріалу з багатством застосувань. З одного боку, у книзі докладно подано багато алгоритмів дискретної математики – від класичних (генерування комбінаторних об'єктів, найкоротший шлях у графі, мінімізація булевих функцій тощо) до сучасних алгоритмів штучного інтелекту (алгоритм побудови дерева прийняття рішень, метод резолюцій тощо). Це робить підручник привабливим для практичних застосувань і хорошою основою майбутніх курсів із сучасних інформаційних технологій. З іншого боку, автори намагалися не поступатись математичною строгістю подання матеріалу: у книзі багато теорем із докладними доведеннями. Це дуже

Розділ 1

Основи: логіка та методи доведення, множини

- Логіка висловлювань
- Логіка першого ступеня
- Логічне виведення в логіці висловлювань
- Метод резолюцій
- Правила виведення в численні предикатів
- Методи доведення теорем
- Множина. Кортеж. Декартів добуток
- Операції над множинами. Доведення рівностей із множинами
- Комп'ютерне подання множин

Логіку як науку, створену Арістотелем (384–322 до н. е.), упродовж століть використовували для розвитку багатьох галузей знань, зокрема філософії та математики. По сутті, логіка — це наука про міркування, яка дає змогу визначити істинність або хибність математичного твердження, виходячи з первинних припущень, які називають аксіомами. Логіку також застосовують в інформатиці для побудови комп'ютерних програм і доведення їх коректності. Поняття, методи й засоби логіки покладено в основу сучасних інформаційних технологій.

Теорія множин і математична логіка — універсальна мова подальших розділів книги.

1.1. Логіка висловлювань

Висловлюванням називають розповідне речення, про яке можна сказати, що воно чи істинне, чи хибне, але не одне й інше водночас. Розділ логіки, який вивчає висловлювання та їхні властивості, називають *пропозиційною логікою* або *логікою висловлювань* [28, 52].

Приклад 1.1. Наведемо приклади речень.

1. Сніг білий.
2. Київ — столиця України.

3. $x+1=3$.
4. Котра година?
5. Читай уважно!

Два перші речення — висловлювання, решта три — ні, бо третє речення набуває істинного чи хибного значення залежно від значення змінної x, четверте та п'яте речення — не розповідні.

Значення „істина" чи „хибність", яких набуває висловлювання, називають його *значенням істинності*. Значення „істина" позначають буквою T (від англ. „truth"), а „хибність" — буквою F (від „false"). Для позначення висловлювань використовують малі латинські букви з індексами чи без них. Символи, використовувані для позначення висловлювань, називають *атомарними формулами* чи *атомами*.

Приклад 1.2. Наведемо приклади висловлювань.

1. p: „Сніг білий".
2. q: „Київ — столиця України".

Тут символи p, q — атомарні формули.

Багато речень утворюють об'єднанням одного чи декількох висловлювань. Отримане висловлювання називають *складним*. Побудову складних висловлювань уперше розглянуто 1845 р. в книзі англійського математика Дж. Буля (G. Boole) „The Laws of Truth". Складне висловлювання утворюють із наявних висловлювань за допомогою *логічних операцій*. У логіці висловлювань використовують п'ять логічних операцій: *заперечення* (читають „не" та позначають знаком „$\neg$"), *кон'юнкцію* (читають „і (та)" й позначають знаком „$\wedge$"), *диз'юнкцію* (читають „або (чи)" та позначають знаком „$\vee$"), *імплікацію* (читають "якщо..., то" та позначають знаком „$\rightarrow$"), *еквівалентність* (читають „тоді й лише тоді" та позначають знаком „$\sim$").

Приклад 1.3. Наведемо приклади складних висловлювань.

1. Сніг білий, і небо теж біле.
2. Якщо погода хороша, то ми їдемо відпочивати.

У наведених прикладах логічні операції — це „і" та „якщо..., то".

Приклад 1.4. Розглянемо такі висловлювання: p: „Вологість велика", q: „Температура висока", r: „Ми почуваємо себе добре". Тоді речення „Якщо вологість велика та температура висока, то ми не почуваємо себе добре" можна записати складним висловлюванням $((p \wedge q) \rightarrow (\neg r))$.

У логіці висловлювань атом p чи складне висловлювання називають *правильно побудованою формулою* або *формулою*. Вивчаючи формули, розглядають два аспекти — синтаксис і семантику.

Синтаксис — це сукупність правил, які дають змогу будувати формули та розпізнавати правильні формули серед послідовностей символів. *Формули* в логіці висловлювань означають за такими правилами:

- атом — це формула;
- якщо p — формула, то $(\neg p)$ — також формула;

- якщо p та q — формули, то $(p \wedge q)$, $(p \vee q)$, $(p \to q)$, $(p \sim q)$ — формули;
- формули можуть бути породжені тільки скінченною кількістю застосувань указаних правил.

Формули, як і атоми, позначають малими латинськими буквами з індексами чи без них.

Приклад 1.5. Вирази $(p\to)$, $(p\wedge)$, $(p\neg)$, $(\vee q)$ — не формули.

Часто заперечення висловлювання p позначають також $\bar{p}$. Такий спосіб запису заперечення не потребує дужок. Якщо не виникає непорозумінь, то зовнішні дужки у формулах можна випускати.

Приклад 1.6. Формули $(p \vee q)$, $(p \to q)$ та $((p \wedge q) \to (\neg r))$ можна записати відповідно у вигляді $p \vee q$, $p \to q$ та $(p \wedge q) \to \bar{r}$.

Семантика — це сукупність правил, за якими формулам надають значення істинності. Нехай p та q — формули. Тоді значення істинності формул $(\neg p)$, $(p \wedge q)$, $(p \vee q)$, $(p \to q)$ та $(p \sim q)$ так пов'язані зі значеннями істинності формул p та q.

1. Формула $(\neg p)$ істинна, коли p хибна, і хибна, коли формула p істинна. Формулу $(\neg p)$ читають „не p" чи „це не так, що p" та називають *запереченням формули* p.
2. Формула $(p \wedge q)$ істинна, якщо p та q водночас істинні. У всіх інших випадках формула $(p \wedge q)$ хибна. Формулу $(p \wedge q)$ читають „p і q" й називають *кон'юнкцією* формул p та q.
3. Формула $(p \vee q)$ хибна, якщо p та q водночас хибні. У всіх інших випадках $(p \vee q)$ істинна. Формулу $(p \vee q)$ читають „p або q" й називають *диз'юнкцією* формул p та q.
4. Формула $(p \to q)$ хибна, якщо формула p істинна, а q — хибна. У всіх інших випадках вона істинна. Формулу $(p \to q)$ називають *імплікацією*, атом p — *припущенням імплікації*, а q — її *висновком*. Оскільки імплікацію використовують у багатьох математичних міркуваннях, то існує багато термінологічних варіантів для формули $(p \to q)$. Ось деякі з них: „якщо p, то q", „з p випливає q", „p лише тоді, коли q", „p достатнє для q", „q, якщо p", „q необхідне для p".
5. Формула $(p \sim q)$ істинна, якщо p та q мають однакові значення істинності. У всіх інших випадках формула $(p \sim q)$ хибна. Формулу $(p \sim q)$ читають „p тоді й лише тоді, коли q" чи „p еквівалентне q" та називають *еквівалентністю* формул p та q.

Семантику логічних операцій зручно задавати за допомогою таблиць, які містять значення істинності формул залежно від значень істинності їх атомів. Такі таблиці називають *таблицями істинності*. Семантику введених операцій у формі таблиць істинності наведено в табл. 1.1.

Таблиця 1.1

p	q	$(\neg p)$	$(p \wedge q)$	$(p \vee q)$	$(p \to q)$	$(p \sim q)$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Приклад 1.7. Знайдемо заперечення висловлювання „Сьогодні п'ятниця". Воно має вигляд „Це не так, що сьогодні п'ятниця". Це речення також можна сформулювати як „Сьогодні не п'ятниця" чи „П'ятниця не сьогодні". Зазначимо, що речення, пов'язані з часовою змінною, — не висловлювання доти, доки не визначено момент часу. Це стосується й змінних у реченнях, які характеризують місце чи особу. Ці речення — не висловлювання, якщо не зазначено відповідного місця чи конкретної особи.

Приклад 1.8. Знайдемо кон'юнкцію висловлювань p та q, де p — висловлювання „Сьогодні п'ятниця", а q — „Сьогодні падає дощ". Кон'юнкція цих висловлювань — „Сьогодні п'ятниця, і сьогодні падає дощ". Воно істинне в дощову п'ятницю й хибне в інший день або в недощову п'ятницю.

Приклад 1.9. Що являє собою диз'юнкція висловлювань p та q з прикладу 1.8? Диз'юнкція висловлювань p та q — висловлювання „Сьогодні п'ятниця чи сьогодні падає дощ". Воно істинне в будь-яку п'ятницю чи в будь-який дощовий день (зокрема, у дощову п'ятницю) і хибне тільки в недощові „не п'ятниці".

Логічна операція „диз'юнкція" відповідає одному з двох способів уживання слова „чи (або)" в українській мові. Диз'юнкція істинна, якщо істинне принаймні одне з двох висловлювань. Розглянемо речення „Лекції з логіки можуть відвідувати студенти, які прослухали курси математичного аналізу чи дискретної математики". Його зміст полягає в тому, що лекції можуть відвідувати як студенти, які прослухали обидва курси, так і ті, хто прослухав тільки один із них. Але є й інше, *альтернативне „чи (або)"*. Розглянемо речення „Лекції з логіки мають відвідувати студенти, які прослухали тільки один із двох курсів — математичного аналізу чи дискретної математики". Зміст цього речення полягає в тому, що студенти, які прослухали обидва ці курси, уже не повинні слухати лекції з логіки. Аналогічно, якщо в меню зазначено „Закуску чи салат подають із першою стравою", то це майже завжди означає, що з першою стравою буде подано чи закуску, чи салат, а не обидві страви. В останніх двох реченнях використано альтернативне „чи (або)"; його позначають знаком „$\oplus$". Значення істинності цієї операції наведено в табл. 1.2.

Таблиця 1.2

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Імплікацію як логічну операцію називають також *умовним реченням*. Щоб зрозуміти, чому імплікація набуває таких значень істинності, слід сприймати її як зв'язок обов'язкового й очікуваного. Наприклад, розглянемо звернення, адресоване студентам: „Якщо ви виконаєте всі завдання, то отримаєте відмінну оцінку". Це означає, що в разі виконання студентами всіх завдань вони одержать відмінну оцінку. Якщо ж студенти не виконають усіх завдань, то вони можуть отримати оцінку „відмінно", а можуть і не отримати її залежно від інших обставин. Однак якщо студенти зробили всі завдання, а викладач не поставив оцінку „відмінно", то студенти відчуватимуть себе ображеними. Це відповідає ситуації, коли в імплікації $p \to q$ припущення p „Ви виконаєте всі завдання" істинне, а її висновок q „Ви отримаєте відмінну оцінку" хибний.

Розуміння імплікації в логіці дещо відрізняється від його розуміння в природній мові. Наприклад, „Якщо буде сонячно, то ми підемо на пляж" — умовне речення, уживане у звичайній мові. Воно залишається істинним до того моменту, коли настане сонячний день, а ми не підемо на пляж. За означенням імплікації умовне речення „Якщо сьогодні п'ятниця, то $2 + 3 = 5$" істинне, бо висновок імплікації істинний. При цьому значення істинності припущення в імплікації тут не має відношення до висновку. Імплікація „Якщо сьогодні п'ятниця, то $2 + 3 = 6$" істинна щодня, крім п'ятниці, хоча висловлювання $2 + 3 = 6$ хибне. Останні дві імплікації ми не вживаємо в природній мові (хіба що як жарт), оскільки у кожному з відповідних умовних речень немає змістовного зв'язку між припущенням і висновком.

Конструкція „якщо p, то q", використовувана у вигляді „if p then q" в алгоритмічних мовах, відрізняється за змістом від імплікації в логіці. Тут p — висловлювання, а q — програмний сегмент, який складається з одного чи багатьох операторів. Програмний сегмент q виконується, якщо висловлювання p істинне, і не виконується, якщо воно хибне.

Для знаходження значення істинності складного висловлювання потрібно надати значення істинності всім атомам, які містить відповідна формула. Набір значень істинності всіх атомів формули називають її *інтерпретацією*. Для обчислення значень істинності формули, яка зображає складне висловлювання, потрібно знаходити значення логічних операцій, визначених табл. 1.1. Послідовність обчислень задають парами дужок. Якщо формула має n атомів, то є 2^n способів надати значення істинності її атомам, тобто така формула має 2^n інтерпретацій, а всі її значення можна звести в таблицю істинності з 2^n рядками. Формулу, яка містить n атомів, називають *n-місною*.

Формулу f називають *виконанною*, якщо існує принаймні одна інтерпретація, у якій f набуває значення T. У такому разі говорять, що формула f *виконується* в цій інтерпретації.

Приклад 1.10. Розглянемо формулу $(p \wedge q) \to (p \sim \bar{r})$. Позаяк кожному з атомів p, q й r можна надати два значення — F або T, то задана формула має $2^3 = 8$ інтерпретацій. Обчислимо значення істинності заданої формули для значень істинності атомів p, q та r, що відповідно дорівнюють T, T й F. Цим задано одну з інтерпретацій формули. Тоді формула $(p \wedge q)$ має значення T; $\bar{r}$ має значення T, позаяк r хибне; формула $(p \sim \bar{r})$ істинна, бо p та $\bar{r}$ істинні; нарешті, формула $((p \wedge q) \to (p \sim \bar{r}))$ істинна, оскільки $(p \wedge q)$ та $(p \sim \bar{r})$

істинні. Отже, задана формула виконується в цій інтерпретації, позаяк вона набуває значення Т. Значення істинності формули $(p \wedge q) \to (p \sim \bar{r})$ у всіх її інтерпретаціях наведено в табл. 1.3.

Таблиця 1.3

p	q	r	$\bar{r}$	$(p \wedge q)$	$(p \sim \bar{r})$	$(p \wedge q) \to (p \sim \bar{r})$
T	T	T	F	T	F	F
T	T	F	T	T	T	T
T	F	T	F	F	F	T
T	F	F	T	F	T	T
F	T	T	F	F	T	T
F	T	F	T	F	F	T
F	F	T	F	F	T	T
F	F	F	T	F	F	T

Формулу f логіки висловлювань називають *загальнозначущою* чи *тавтологією*, якщо вона виконується в усіх інтерпретаціях. Якщо формула f — тавтологія, то використовують позначення $\models f$. Формулу, хибну в усіх інтерпретаціях, називають *заперечуваною*, *невиконанною* чи *суперечністю*.

Оскільки кожна формула логіки висловлювань має скінченну кількість інтерпретацій, то завжди можна перевірити її загальнозначущість або заперечуваність, знайшовши значення істинності в усіх можливих інтерпретаціях.

Приклад 1.11. Розглянемо формулу $((p \to q) \wedge p) \to q$. Її атоми — p та q. Формула має $2^2 = 4$ інтерпретації. Значення істинності наведено в табл. 1.4. Ця формула істинна в усіх інтерпретаціях, тобто являє собою тавтологію.

Таблиця 1.4

p	q	$(p \to q)$	$(p \to q) \wedge p$	$((p \to q) \wedge p) \to q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Приклад 1.12. Розглянемо формулу $(p \to q) \wedge (p \wedge \bar{q})$. З табл. 1.5 доходимо висновку, що вона хибна в усіх інтерпретаціях, тобто заперечувана.

Таблиця 1.5

p	q	$(p \to q)$	$\bar{q}$	$p \wedge \bar{q}$	$(p \to q) \wedge (p \wedge \bar{q})$
T	T	T	F	F	F
T	F	F	T	T	F
F	T	T	F	F	F
F	F	T	T	F	F

Комп'ютери відображають інформацію за допомогою бітів. *Біт* має два можливі значення — 0 (нуль) і 1 (одиниця). Його можна використовувати для подання значень істинності T й F. Зазвичай 1 використовують для зображення T й 0 — для зображення F. Змінну називають *булевою*, якщо її значення — T чи F. Отже, булеві змінні можна подати за допомогою бітів.

Комп'ютерні операції над бітами відповідають логічним операціям. Замінивши T на 1, а F — на 0 у таблицях істинності для логічних операцій $\vee$, $\wedge$ та $\oplus$, отримаємо таблиці відповідних операцій над бітами. Ми будемо також використовувати нотацію OR, AND і XOR відповідно для логічних операцій $\vee$, $\wedge$ та $\oplus$, як у багатьох мовах програмування. Значення операцій OR, AND та XOR над бітами наведено в табл. 1.6.

Таблиця 1.6

x	y	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Бітовим рядком називають скінченну послідовність бітів. Розглядають також рядок, який не містить жодного біта (*порожній рядок*). *Довжиною* бітового рядка називають кількість бітів у ньому. Наприклад, 110010100 — бітовий рядок довжиною дев'ять.

Операції над бітами можна узагальнити на бітові рядки. Означимо *порозрядне* OR, *порозрядне* AND і *порозрядне* XOR двох бітових рядків з однією довжиною як бітовий рядок, який має таку саму довжину, а його біти — відповідно результати операцій OR, AND і XOR над відповідними бітами цих рядків.

Приклад 1.13. Знайдемо результати операцій порозрядного OR, порозрядного AND і порозрядного XOR бітових рядків 1011000011 і 1101010101. Одержимо

1011000011
1101010101
1111010111 — порозрядне OR
1001000001 — порозрядне AND
0110010110 — порозрядне XOR

1.2. Закони логіки висловлювань

Формули f і g називають *еквівалентними*, *рівносильними* чи *тотожними* (позначають $f = g$), якщо значення їх істинності збігаються в усіх інтерпретаціях цих формул. Властивість еквівалентності формул f і g можна сформулювати у вигляді такого твердження.

ТЕОРЕМА 1.1. Формули f і g еквівалентні тоді й лише тоді, коли формула $(f \sim g)$ загальнозначуща, тобто $\models f \sim g$.

Приклад 1.14. За допомогою таблиці істинності доведемо, що $p \to q = \bar{p} \vee q$. Результати розв'язування задачі наведено в табл. 1.7.

Таблиця 1.7

p	q	$p \to q$	$\bar{p}$	$\bar{p} \vee q$	$(p \to q) \sim (\bar{p} \vee q)$
T	T	T	F	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

Приклад 1.15. За допомогою таблиці істинності доведемо, що $p \to q \neq q \to p$. Результати розв'язування задачі наведено в табл. 1.8.

Таблиця 1.8

p	q	$p \to q$	$q \to p$	$(p \to q) \sim (q \to p)$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

Розглянемо еквівалентні формули, які задають правила перетворень. Такі еквівалентності називають *законами логіки висловлювань*. Перетворення виконують заміною якоїсь формули в складі іншої формули на еквівалентну їй формулу. Цю процедуру повторюють доти, доки не буде отримано формулу в потрібній формі. Основні закони логіки висловлювань наведено в табл. 1.9.

Таблиця 1.9

Назва закону	Формулювання закону
1. Закони комутативності	*а*) $p \vee q = q \vee p$; *б*) $p \wedge q = q \wedge p$
2. Закони асоціативності	*а*) $(p \vee q) \vee r = p \vee (q \vee r)$
	б) $(p \wedge q) \wedge r = p \wedge (q \wedge r)$
3. Закони дистрибутивності	*а*) $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
	б) $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$
4. Закон суперечності	$p \wedge \bar{p} = \mathrm{F}$
5. Закон виключеного третього	$p \vee \bar{p} = \mathrm{T}$
6. Закон подвійного заперечення	$\bar{\bar{p}} = p$
7. Закони ідемпотентності	*а*) $p \vee p = p$; *б*) $p \wedge p = p$
8. Закони де Моргана	*а*) $\overline{p \vee q} = \bar{p} \wedge \bar{q}$; *б*) $\overline{p \wedge q} = \bar{p} \vee \bar{q}$
9. Закони поглинання	*а*) $(p \vee q) \wedge p = p$; *б*) $(p \wedge q) \vee p = p$
10. Співвідношення для сталих	*а*) $p \vee \mathrm{T} = \mathrm{T}$; *б*) $p \wedge \mathrm{T} = p$
	в) $p \vee \mathrm{F} = p$; *г*) $p \wedge \mathrm{F} = \mathrm{F}$

Закони асоціативності дають змогу записувати багатомісні диз'юнкції та кон'юнкції без дужок.

За допомогою правил $p \to q = \bar{p} \vee q$ та $p \sim q = (p \to q) \wedge (q \to p)$ можна усувати логічні операції імплікації й еквівалентності з формул. Ці правила можна використовувати також для *введення* імплікації й еквівалентності.

Наведені еквівалентності можна перевірити, побудувавши таблиці істинності. Зокрема, у прикладі 1.14 доведено еквівалентність $p \to q = \bar{p} \vee q$, а приклад 1.15 показує, що імплікація не комутативна. Покажемо, як можна застосувати закони логіки висловлювань для доведення еквівалентності формул.

Приклад 1.16. Застосувавши закони логіки висловлювань, доведемо еквівалентність формул $p \to (q \wedge r)$ і $(p \to q) \wedge (p \to r)$. Запишемо послідовність перетворень і назви використаних законів і правил:

$$p \to (q \wedge r) = \bar{p} \vee (q \wedge r)$$
(за правилом усунення імплікації)
$$= (\bar{p} \vee q) \wedge (\bar{p} \vee r)$$
(за законом дистрибутивності 3*а*)
$$= (p \to q) \wedge (p \to r)$$
(за правилом уведення імплікації).

Приклад 1.17. За допомогою законів логіки висловлювань доведемо еквівалентність формул $p \to q$ та $\bar{q} \to \bar{p}$. Цю еквівалентність називають *правилом контрапозиції*.

$$p \to q = \bar{p} \vee q$$
(за правилом усунення імплікації)
$$= q \vee \bar{p}$$
(за законом комутативності 1*а*)
$$= \bar{\bar{q}} \vee \bar{p}$$
(за законом подвійного заперечення 6)
$$= \bar{q} \to \bar{p}$$
(за правилом уведення імплікації).

1.3. Нормальні форми логіки висловлювань

Літералом називають атом або його заперечення. Приклади літералів — p, $\bar{q}$, r. Літерал називають *позитивним*, якщо він не має знака заперечення, і *негативним*, якщо має. Пару літералів $\{p, \bar{p}\}$ називають *контрарною*.

Говорять, що формулу f записано в *кон'юнктивній нормальній формі* (КНФ), якщо вона має вигляд $f = f_1 \wedge f_2 \wedge \ldots \wedge f_n$ $(n \geqslant 1)$, де кожна з формул $f_1, f_2, \ldots, f_n$ — літерал або диз'юнкція літералів і всі формули f_i $(i = 1, 2, \ldots, n)$ різні.

Приклад 1.18. Нехай p, q й r — атоми. Тоді $f = (p \vee \bar{q} \vee \bar{r}) \wedge (\bar{p} \vee q)$ — формула, записана в КНФ. У ній $f_1 = (p \vee \bar{q} \vee \bar{r})$ і $f_2 = (\bar{p} \vee q)$, тобто f_1 — диз'юнкція літералів p, $\bar{q}$ й r, а f_2 — диз'юнкція літералів $\bar{p}$ та q.

Говорять, що формулу f записано в *диз'юнктивній нормальній формі* (ДНФ), якщо вона має вигляд $f = f_1 \vee f_2 \vee \ldots \vee f_n$ $(n \geqslant 1)$, де кожна з формул $f_1, f_2, \ldots, f_n$ — літерал або кон'юнкція літералів і всі f_i $(i = 1, 2, \ldots, n)$ різні.

Приклад 1.19. Нехай p, q й r — атоми. Тоді $f = (\bar{p} \wedge q) \vee (p \wedge \bar{q} \wedge \bar{r})$ — формула, записана в ДНФ. У ній $f_1 = (\bar{p} \wedge q)$ й $f_2 = (p \wedge \bar{q} \wedge \bar{r})$; f_1 — кон'юнкція літералів $\bar{p}$ та q, а f_2 — кон'юнкція літералів p, $\bar{q}$ й $\bar{r}$.

Довільну формулу можна перетворити в одну з нормальних форм, застосувавши закони логіки висловлювань. Для побудови нормальних форм потрібно виконати таку послідовність еквівалентних перетворень.

Крок 1. Застосувати правила $f \to g = \overline{f} \vee g$ й $f \sim g = (f \to g) \wedge (g \to f)$ (див. підрозділ 1.2) для усунення логічних операцій „$\to$" та „$\sim$".

Крок 2. Застосувати закон подвійного заперечення та закони де Моргана для перенесення знака заперечення безпосередньо до атомів.

Крок 3. Застосувати відповідні закони дистрибутивності для побудови нормальної форми. Щоб побудувати КНФ, потрібно використати дистрибутивний закон для диз'юнкції щодо кон'юнкції (закон 3*а* з табл. 1.9). Для побудови ДНФ слід застосувати дистрибутивний закон для кон'юнкції щодо диз'юнкції (закон 3*б* з табл. 1.9).

Приклад 1.20. Побудуємо ДНФ формули $((p \vee \overline{q}) \to r) \wedge (\overline{r} \to s)$. Наведемо послідовність кроків та зазначимо застосовані закони логіки висловлювань.

$$((p \vee \overline{q}) \to r) \wedge (\overline{r} \to s) = \overline{((p \vee \overline{q}) \vee r)} \wedge (\overline{\overline{r}} \vee s)$$
(усунення логічної операції „$\to$")
$$= ((\overline{p} \wedge \overline{\overline{q}}) \vee r) \wedge (r \vee s)$$
(закон де Моргана 8*а*)
$$= ((\overline{p} \wedge q) \vee r) \wedge (r \vee s)$$
(закон подвійного заперечення)
$$= ((\overline{p} \wedge q) \wedge (r \vee s)) \vee (r \wedge (r \vee s))$$
(закон дистрибутивності 3*б*)
$$= ((\overline{p} \wedge q \wedge r) \vee (\overline{p} \wedge q \wedge s)) \vee ((r \wedge r) \vee (r \wedge s))$$
(закон дистрибутивності 3*б*)
$$= (\overline{p} \wedge q \wedge r) \vee (\overline{p} \wedge q \wedge s) \vee (r \wedge r) \vee (r \wedge s)$$
(закон асоціативності 2*а*)
$$= (\overline{p} \wedge q \wedge r) \vee (\overline{p} \wedge q \wedge s) \vee r \vee (r \wedge s)$$
(закон ідемпотентності 7*б*).

Ми одержали ДНФ. Її можна спростити, якщо двічі використати закон поглинання 9*б*: диз'юнктивний член r поглинає члени $(\overline{p} \wedge q \wedge r)$ і $(r \wedge s)$. Отже, $(\overline{p} \wedge q \wedge s) \vee r$ — інша ДНФ заданої формули. Останні міркування свідчать, що ДНФ, загалом кажучи, не єдина.

Приклад 1.21. Побудуємо КНФ формули $(p \wedge (q \to r)) \to s$. Наведемо послідовність кроків і застосовані закони:

$$(p \wedge (q \to r)) \to s = \overline{(p \wedge (\overline{q} \vee r))} \vee s$$
(усунення логічної операції „$\to$")
$$= (\overline{p} \vee \overline{(\overline{q} \vee r)}) \vee s$$
(за законом де Моргана 8*б*)
$$= \overline{p} \vee (\overline{\overline{q} \vee r}) \vee s$$
(за законом асоціативності 2*а*)
$$= \overline{p} \vee (\overline{\overline{q}} \wedge \overline{r}) \vee s$$
(за законом де Моргана 8*а*)
$$= \overline{p} \vee (q \wedge \overline{r}) \vee s$$
(за законом подвійного заперечення 6)
$$= \overline{p} \vee s \vee (q \wedge \overline{r})$$
(за законом комутативності 1*а*)
$$= (\overline{p} \vee s) \vee (q \wedge \overline{r})$$
(за законом асоціативності 2*а*)
$$= (\overline{p} \vee q \vee s) \wedge (\overline{p} \vee \overline{r} \vee s)$$
(за законом дистрибутивності 3*а*).

Ми одержали шукану КНФ.

1.4. Логіка першого ступеня

Як зазначено в розділі 1.1, існують речення, які не являють собою висловлювання та містять змінні. Було наведено приклад такого речення – „$x + 1 = 3$". Речення зі змінними – це не висловлювання, але вони перетворюються на висловлювання, якщо надати змінним певних значень. Речення зі змінними дуже поширені. Вони містяться в математичних формулах і комп'ютерних програмах. Зокрема, у мовах програмування є оператори у вигляді „Повторити цикл доти, доки змінні x та y не стануть рівними, або припинити обчислення після 100 повторень". Позначивши як i лічильник повторень, умову закінчення програми можна задавати виразом „$(x = y) \vee (i > 100)$". Тоді оператор циклу набирає вигляду „повторювати, якщо $(\neg((x = y) \vee (i > 100)))$".

Приклад 1.22. Речення „$x > 3$", „$x = y + 3$", „$x + y = z$" містять змінні. Вони не істинні й не хибні доти, доки змінним не буде надано якихось значень.

У наведеному прикладі речення „$x > 3$", або, в іншому вигляді, „x більше 3", складається з двох частин: першу, змінну x, називають *предметом*, а другу – „більше 3", – яка показує властивість предмета, – *предикатом*. Часто предикатом називають усе речення.

Означимо *логіку першого ступеня* (*логіку предикатів*), у якій до понять логіки висловлювань додано нові поняття [28, 34]. Для формулювання складних думок у логіці висловлювань використовують атоми як основні елементи формул. Атом розглядають як неподільне ціле – його структуру та склад не аналізують. Позаяк багато міркувань неможливо описати лише за допомогою висловлювань, уведемо поняття атома в логіці першого ступеня. Для запису атомів логіки першого ступеня використовують такі типи символів:

- *індивідні символи*, або *сталі* – це імена об'єктів, які починаються з великої букви, та сталі, наприклад: Іван, Марія, Дискретна_математика, T, F, 2, 5;
- *предметні символи*, *предметні змінні*, або просто *змінні* – імена, якими позначають змінні та які записують малими буквами (можливо, з індексами), наприклад: x, y, z, v_i, w_j;
- *предикатні символи* – імена, якими позначають предикати та які записують великими буквами (наприклад, P, Q, R) або змістовними словами, які записують великими буквами (наприклад, *БІЛЬШЕ*, *ЛЮБИТЬ*).

Приклад 1.23. Позначимо речення „x більше 3" як $P(x)$, де предикатний символ P позначає предикат „більше 3", а x – предметна змінна. Вираз $P(x)$ у цілому теж називають предикатом. Щоб записати твердження „x більше 3" як предикат, можна зробити інакше – означити предикат *БІЛЬШЕ*(x, y), який позначає „x більше y". Тоді речення „x більше 3" можна записати як предикат *БІЛЬШЕ*$(x, 3)$.

Загалом, предикат, який містить n предметних змінних $x_1, x_2, ..., x_n$, записують $P(x_1, x_2, ..., x_n)$ і називають *n-місним*. *Предметною областю змінної* x_i називають множину D_i її значень, а символ P – *n-місним предикатним символом*. Замість поняття „предикат" іноді використовують термін *„пропозиційна функція"*.

Атом логіки першого ступеня має вигляд $P(x_1, x_2, ..., x_n)$, де P – предикатний символ, а $x_1, x_2, ..., x_n$ – предметні чи індивідні символи.

Щойно змінна x набуває якогось значення з предметної області, предикат $P(x)$ набуває значення T чи F і перетворюється на висловлювання. Аналогічно, якщо всі змінні багатомісного предиката набувають якихось значень, то він набуває значення істинності та перетворюється на висловлювання.

Приклад 1.24. Нехай вираз „$x + y = 2$" задано предикатом $Q(x, y)$. Тоді $Q(1, 2)$ — хибне висловлювання, а $Q(2, 0)$ — істинне. Позначимо це так: $Q(1, 2) = \text{F}$, $Q(2, 0) = \text{T}$.
Задамо речення „x любить y" предикатом *ЛЮБИТЬ*(x, y). Тоді істинне речення „Іван любить Марію" можна подати істинним висловлюванням *ЛЮБИТЬ*(Іван, Марія).

Приклад 1.25. Якщо *БІЛЬШЕ*(x, y) — предикат, означений у прикладі 1.23, то *БІЛЬШЕ*(5, 3) — істинне висловлювання, а *БІЛЬШЕ*(1, 3) — хибне.

Є й інший спосіб перетворення предиката у висловлювання — *квантифікація*. Нехай $P(x)$ — предикат, D — задана предметна область, $x \in D$. Використовують два спеціальні символи $\forall$ та $\exists$, які називають відповідно *кванторами загальності* й *існування*. Якщо x — предметна змінна, то вираз $(\forall x)$ читають „для всіх x", „для кожного x" або „для будь-якого x". Запис $(\forall x)P(x)$ означає „предикат $P(x)$ істинний для всіх значень x із предметної області"; його читають як „$P(x)$ для всіх x". Вираз $(\exists x)$ читають „існує x", „для деяких x" або „принаймні для одного x". Запис $(\exists x)P(x)$ має зміст „в області D існує таке x, що предикат $P(x)$ істинний", або „в області D існує принаймні одне x таке, що предикат $P(x)$ істинний", або „предикат $P(x)$ істинний для якогось x з області D". Дужки біля квантора можна випускати, тобто замість $(\forall x)$ та $(\exists x)$ писати відповідно $\forall x$ та $\exists x$.

Перехід від $P(x)$ до $\forall xP(x)$ або $\exists xP(x)$ називають *зв'язуванням* предметної змінної x, а саму змінну x — *зв'язаною*. Незв'язану змінну називають *вільною*. Говорять, що у виразах $\forall xP(x)$ та $\exists xP(x)$ предикат $P(x)$ належить *області дії* відповідного квантора.

Приклад 1.26. У виразі $\exists xP(x, y)$ змінна x зв'язана, а змінна y — вільна, бо предикат $P(x, y)$ не належить області дії квантора зі змінною y.

Для пошуку значення істинності висловлювання, отриманого з пропозиційної функції зв'язуванням її змінних кванторами, потрібно знати предметну область.

Приклад 1.27. Нехай предикат $P(x)$ відповідає реченню „$x \geqslant 1$", а предметна область складається з усіх дійсних чисел. Тоді висловлювання $\forall xP(x)$ хибне: $\forall xP(x) = \text{F}$. Якщо ж предметна область складається з усіх натуральних чисел, то висловлювання $\forall xP(x)$ істинне: $\forall xP(x) = \text{T}$.

Правильно побудовані формули логіки першого ступеня, або *формули логіки першого ступеня*, означають так:

- атом — це формула;
- якщо H і G — формули, то $(\neg H)$, $(H \vee G)$, $(H \to G)$ та $(H \sim G)$ — формули;
- якщо H — формула, а x — вільна змінна у формулі H, то $\forall xH$ та $\exists xH$ — формули;
- формули можна породити тільки скінченною кількістю застосувань попередніх трьох правил.

Зазначимо, що замість $(\neg H)$ можна писати $\overline{H}$.

Наведемо приклади висловлювань, одержаних згідно з означенням формули логіки першого ступеня.

Приклад 1.28. Позначимо речення „x — просте число" як $P(x)$, „x — раціональне число" — як $Q(x)$, „x — дійсне число" — як $R(x)$ та „x менше y" — як $МЕНШЕ(x, y)$. Розглянемо такі істинні твердження.

1. Кожне раціональне число дійсне.
2. Існує просте число.
3. Для кожного числа x існує таке число y, що $x < y$.

Наведені речення можна записати такими формулами.

1. $\forall x\,(Q(x) \to R(x))$.
2. $\exists x\,P(x)$.
3. $\forall x\,\exists y\;МЕНШЕ(x, y)$.

Приклад 1.29. У формулі $\forall x\,\exists y\;МЕНШЕ(x, y)$ формула $МЕНШЕ(x, y)$ належить області дії квантора існування, а формула $\exists y\;МЕНШЕ(x, y)$ — області дії квантора загальності. Формула $(Q(x) \to R(x))$ належить області дії квантора загальності у формулі $\forall x\,(Q(x) \to R(x))$.

Зв'язування частини змінних багатомісного предиката перетворює його на предикат із меншою кількістю змінних. Зміст зв'язаних і вільних змінних у предикатах різний. Вільні змінні — це звичайні змінні, які можуть набувати різних значень із предметної області D; вираз $P(x)$ змінний, і його значення залежить від значення змінної x. Формули $\exists x P(x)$ та $\forall x P(x)$ не залежать від змінної x і для певних P та D мають конкретні значення. Це, зокрема, означає, що перейменування зв'язаних змінних, а саме, заміна $\forall x P(x)$ на $\forall y P(y)$, не змінює істинності формули.

Розглянемо приклади перекладу речень, записаних українською мовою, на мову предикатів і кванторів. Головна проблема такого перекладу полягає в правильному використанні кванторів. Кожне речення можна подати декількома способами, і не існує загального алгоритму, який дає змогу будувати його крок за кроком.

Приклад 1.30. Запишемо речення „Кожний студент групи вивчав дискретну математику" за допомогою предикатів і кванторів. Спочатку перепишемо речення так, щоб було зрозуміло, як краще розставити квантори: „Про кожного студента групи відомо, що цей студент вивчав дискретну математику". Тепер уведемо змінну x, і речення набере вигляду: „Про кожного студента x групи відомо, що x вивчав дискретну математику". Уведемо предикат $C(x)$: „x вивчав дискретну математику". Якщо предметна область змінної x — усі студенти групи, то можна записати задане речення як $\forall x C(x)$. Є й інші коректні подання з різними предметними областями та предикатами. Зокрема, можна вважати, що нас цікавлять інші групи людей, окрім тих, які вчаться в одній академічній групі. Узявши як предметну область усіх людей, можна записати задане речення так: „Для кожної особи x, якщо ця особа x — студент групи, то x вивчав дискретну математику". Якщо предикат $S(x)$ має вигляд „Особа x учиться в групі", то задане речення треба записати у вигляді $\forall x(S(x) \to C(x))$. Зауважимо, що задане речення не можна записати як $\forall x(S(x) \wedge C(x))$, бо тоді це означало б, що всі особи з предметної області вчаться в групі та вивчали дискретну математику.

Ще один спосіб записати задане речення — це ввести двомісний предикат $Q(x, y)$: „Студент x вивчає дисципліну y". Тоді можна замінити $C(x)$ на $Q(x,$ Дискретна_математика),

що дасть можливість переписати наведені формули у вигляді $\forall x Q(x,$ Дискретна_математика) чи $\forall x(S(x) \to Q(x,$ Дискретна_математика)).

Приклад 1.31. Запишемо речення „Деякі студенти групи відвідали Париж" за допомогою предикатів і кванторів. Це речення означає „У групі є такий студент, що цей студент відвідав Париж". Якщо ввести змінну x, то задане речення можна переписати так: „У групі є такий студент x, що x відвідав Париж". Уведемо предикат $M(x)$, який відповідає реченню „x відвідав Париж". Якщо предметна область змінної x складається тільки зі студентів певної групи, то можна записати це речення як $\exists x M(x)$.

Якщо ж нас цікавлять інші особи, окрім студентів зазначеної групи, то перше із запропонованих речень матиме інший вигляд: „Є така особа x, що x — студент групи й x відвідав Париж". У такому разі предметна область складається з усіх можливих людей. Нехай $S(x)$: „x — студент групи". Тоді речення має такий вигляд: $\exists x(S(x) \wedge M(x))$, бо воно містить повідомлення про те, що хтось — студент групи та відвідав Париж. Це речення не можна подати формулою $\exists x(S(x) \to M(x))$, оскільки вона істинна навіть тоді, коли особа x — не студент групи.

Приклад 1.32. Запишемо речення „Кожний студент групи відвідав Париж або Рим" за допомогою предикатів і кванторів. Задане речення можна переписати так: „Для кожного x із групи відомо, що x відвідав Париж або x відвідав Рим". Позначимо як $C(x)$ речення „x відвідав Париж", а $M(x)$ — „x відвідав Рим". Припустивши, що предметна область складається зі студентів певної групи, задане речення можна записати у вигляді $\forall x(C(x) \vee M(x))$. Якщо ж предметна область складається з усіх людей, то речення набере такого вигляду: „Для кожної особи x за умови, що x — студент групи, відомо, що особа x відвідала Париж або особа x відвідала Рим". Це речення можна записати як $\forall x(S(x) \to (C(x) \vee M(x)))$.

Розглянемо приклади, які ілюструють подання речень природної мови та містять вкладені квантори, тобто в області дії одних кванторів є інші квантори.

Приклад 1.33. Нехай предметна область змінних x та y — усі дійсні числа. Речення $\forall x \forall y(x + y = y + x)$ означає, що рівність $x + y = y + x$ — правило комутативності для суми дійсних чисел — правильна для всіх дійсних чисел x та y. Формула ж $\forall x \exists y(x + y = 0)$ означає, що для кожного дійсного числа x існує таке дійсне число y, що $x + y = 0$. Формула $\forall x \forall y \forall z\ (x + (y + z) = (x + y) + z)$ — це запис правила асоціативності для суми дійсних чисел.

Приклад 1.34. Перекладемо на українську мову формулу $\forall x \forall y(((x > 0) \wedge (y < 0)) \to (xy < 0))$, істинну для дійсних чисел x та y. Ця формула означає, що для дійсних чисел x та y таких, що x — додатне число, а y — від'ємне, їх добуток xy — від'ємне число. Це можна записати реченням „Добуток додатного та від'ємного дійсних чисел — від'ємне число".

Переклад формул із вкладеними кванторами на українську мову може бути доволі складним. Він полягає у виписуванні змісту предикатів і кванторів, після чого потрібно переписати задану формулу реченням української мови.

Приклад 1.35. Подамо формулу $\forall x(C(x) \vee \exists y(C(y) \wedge F(x, y)))$ українською мовою, якщо $C(x)$ означає „x має комп'ютер", $F(x, y)$ — „x та y — друзі", а предметна область для x і для y — усі студенти певного курсу. Зміст формули можна так подати українською мовою: „Кожний студент курсу має комп'ютер або друга, у якого є комп'ютер".

Приклад 1.36. Подамо формулу

$$\exists x \forall y \forall z((F(x, y) \wedge F(x, z) \wedge (y \neq z)) \to \overline{F}(y, z))$$

українською мовою, якщо $F(a, b)$ означає, що a та b — друзі, а предметна область змінних x, y і z — усі студенти університету. Спочатку проаналізуємо формулу $((F(x, y) \wedge F(x, z) \wedge (y \neq z)) \rightarrow \overline{F}(y, z))$. Вона означає, що якщо x та y — друзі, x і z — друзі, а y і z — різні особи, то y і z — не друзі. У формулі кванторами зв'язано всі змінні. З урахуванням змісту кванторів задану формулу можна прочитати так: „Є студент, який дружить із двома групами студентів, котрі не дружать між собою". Така ситуація можлива для студента, який перейшов з одного факультету на інший.

Приклад 1.37. Подамо речення „Якщо певний чоловік — один із батьків, то він — тато" у вигляді формули логіки предикатів. Предметна область кожної змінної — усі люди. Переформулюємо задане речення так: „Для кожної особи x за умови, що x — чоловік та x — один із батьків, існує така особа y, що x — тато y". Увівши предикати $M(x)$: „x — особа чоловічої статі", $P(x)$: „x — один із батьків", $F(x, y)$: „x — батько y", можна записати задане речення так: $\forall x(((M(x) \wedge P(x)) \rightarrow \exists y F(x, y))$.

Приклад 1.38. Подамо речення „Кожна людна має одного найкращого друга" формулою, яка містить предикатні символи, квантифіковані змінні, логічні операції, а предметна область складається з усіх людей. Задане речення переформулюємо так: „Для кожної особи x правильно, що особа x має точно одного найкращого друга". Якщо особа x має точно одного найкращого друга, то це означає, що існує єдина особа y, яка є найкращим другом x. Крім того, кожна особа z, відмінна від y, — не найкращий друг особи x. Уведемо предикат $B(x, y)$: „y — найкращий друг x". Тоді формулу, зміст якої полягає в тому, що людина x має точно одного найкращого друга, можна записати як $\exists y(B(x, y) \wedge \forall z((z \neq y) \rightarrow \rightarrow \overline{B}(x, z)))$, а задане речення — як $\forall x \exists y (B(x, y) \wedge \forall z((z \neq y) \rightarrow \overline{B}(x, z)))$.

Приклад 1.39. Запишемо формулою логіки предикатів речення „Сума двох додатних чисел — додатне число". Спочатку перепишемо це речення так: „Два довільні додатні числа дають у сумі додатне число". Уведемо змінні x та y і отримаємо речення „Будь-які додатні числа x та y утворюють суму $x + y$, яка являє собою додатне число". Запишемо його формулою $\forall x \forall y (((x > 0) \wedge (y > 0)) \rightarrow (x + y > 0))$. Тут предметна область кожної змінної — усі дійсні числа.

Приклад 1.40. Запишемо речення „Кожне дійсне число, окрім нуля, має обернене" у вигляді логічної формули. Спочатку перепишемо це речення так: „Якщо число x дійсне та відмінне від нуля, то число x має обернене", — а потім — так: „Для кожного дійсного числа x, відмінного від нуля, існує таке дійсне число y, що $xy = 1$". Останнє речення можна записати логічною формулою $\forall x ((x \neq 0) \rightarrow \exists y (xy = 1))$.

1.5. Закони логіки першого ступеня

Еквівалентні формули логіки висловлювань залишаються такими й у логіці першого ступеня. Однак у логіці першого ступеня є еквівалентності (закони), пов'язані зі специфікою означення об'єктів логіки першого ступеня.

Дві формули логіки предикатів називають *еквівалентними*, якщо вони набувають однакових значень істинності для довільних значень вільних змінних. Зокрема, якщо формули P та Q еквівалентні, то формула $P \sim Q$ — тавтологія, і навпаки. Еквівалентність формул P та Q записують як $P = Q$. Проблема побудови законів логіки першого ступеня полягає в доведенні еквівалентності формул P та Q. У логіці висловлювань еквівалентність можна перевірити, побудувавши відповіді

таблиці істинності. У логіці першого ступеня аналогічна процедура загалом неможлива, бо предметні змінні можуть мати нескінченні предметні області, і повний перебір усіх їх значень неможливий.

Нижче наведено *основні закони логіки першого ступеня*. Зауважимо, що у формулах показано лише зв'язані змінні.

1. $\overline{\forall x P(x)} = \exists x \overline{P}(x)$.
2. $\overline{\exists x P(x)} = \forall x \overline{P}(x)$.
3. $\forall x (P(x) \wedge Q(x)) = \forall x P(x) \wedge \forall x Q(x)$.
4. $\exists x (P(x) \vee Q(x)) = \exists x P(x) \vee \exists x Q(x)$.
5. $\forall x (P(x) \wedge Q) = \forall x P(x) \wedge Q$.
6. $\forall x (P(x) \vee Q) = \forall x P(x) \vee Q$.
7. $\exists x (P(x) \wedge Q) = \exists x P(x) \wedge Q$.
8. $\exists x (P(x) \vee Q) = \exists x P(x) \vee Q$.
9. $\forall x \forall y P(x, y) = \forall y \forall x P(x, y)$.
10. $\exists x \exists y P(x, y) = \exists y \exists x P(x, y)$.

Для доведення цих законів потрібні спеціальні методи. Проілюструємо це на прикладі доведення еквівалентності $\overline{\exists x P(x)} = \forall x \overline{P}(x)$. Нехай для якогось предикатного символу P та предметної області D ліва частина еквівалентності істинна. Тоді не існує такого $a \in D$, для якого $P(a)$ істинне. Отже, для будь-якого a $P(a)$ хибне, а $\overline{P}(a)$ істинне; тому істинна права частина еквівалентності. Якщо ліва частина еквівалентності хибна, то існує таке $a \in D$, для якого $P(a)$ істинне, тобто права частина хибна. Аналогічно можна довести й закон $\overline{\forall x P(x)} = \exists x \overline{P}(x)$.

Приклад 1.41. Проілюструємо еквівалентність $\overline{\forall x P(x)} = \exists x \overline{P}(x)$. Розглянемо заперечення речення „Кожний студент університету вивчає математичний аналіз". Це речення можна записати за допомогою квантора загальності як $\forall x P(x)$, де $P(x)$ — речення „x вивчає математичний аналіз". Заперечення заданого речення — „Це не так, що кожний студент університету вивчає математичний аналіз"; воно еквівалентне реченню „Існує такий студент університету, який не вивчає математичного аналізу".

Приклад 1.42. Розглянемо речення „В університеті є студент, який вивчає математичний аналіз". Його можна записати як $\exists x P(x)$, де $P(x)$ — речення „x вивчає математичний аналіз". Заперечення заданого речення — „Це не так, що в університеті є студент, який вивчає математичний аналіз"; воно еквівалентне реченню „Кожний студент університету не вивчає математичного аналізу". Останнє отримують застосуванням квантора загальності до заперечення заданого речення, тобто $\forall x \overline{P}(x)$. Це ілюструє еквівалентність $\overline{\exists x P(x)} = \forall x \overline{P}(x)$.

Доведемо закон $\forall x (P(x) \wedge Q(x)) = \forall x P(x) \wedge \forall x Q(x)$. Нехай ліва частина істинна для якихось P та Q, тобто для довільного $a \in D$ істинне $P(a) \wedge Q(a)$. Тому $P(a)$ та $Q(a)$ водночас істинні для довільного a, тобто $\forall x P(x) \wedge \forall x Q(x)$ істинне. Якщо ж ліва частин хибна, то для якогось $a \in D$ хибне принаймні одне з висловлювань $P(a)$ чи $Q(a)$. Це означає, що хибне принаймні одне з висловлювань $\forall x P(x)$ або $\forall x Q(x)$, тобто хибна й права частина.

Аналогічно можна довести еквівалентність $\exists x(P(x) \vee Q(x)) = \exists x P(x) \vee \exists x Q(x)$.

Зазначимо, що в законах 9 і 10 змінні в предикатах зв'язані однаковими кванторами, тому можна переставляти їх без порушення еквівалентності. Якщо квантори різні, подібна еквівалентність виконується не завжди, тобто загалом $\forall x \exists y P(x, y) \neq \exists y \forall x P(x, y)$. Наведемо приклад, який ілюструє це зауваження.

Приклад 1.43. Розглянемо двомісний предикат $P(x, y)$: „$x \geqslant y$" на різних предметних областях. Формула $\exists x \forall y P(x, y)$ означає, що в предметній області існує максимальний елемент. Ця формула істинна на предметній області, що являє собою будь-яку скінченну підмножину множини цілих чисел, але хибна, наприклад, на множині $\{1/2, 2/3, 3/4, ..., n/(n+1), ...\}$. Висловлювання $\forall y \exists x P(x, y)$ означає, що для довільного елемента y існує елемент x, не менший від y. Таке висловлювання істинне на довільній непорожній множині. Отже, переставлення кванторів існування та загальності може змінити зміст висловлювання та значення його істинності.

Якщо $D = \{a_1, a_2, ..., a_n\}$ — скінченна предметна область змінної x у предикаті $P(x)$, то можна скористатися логічними еквівалентностями $\forall x P(x) = P(a_1) \wedge P(a_2) \wedge ... \wedge P(a_n)$ та $\exists x P(x) = P(a_1) \vee P(a_2) \vee ... \vee P(a_n)$. У такому разі заперечення квантифікованої формули дає той самий результат, що й застосування відповідного закону де Моргана. Це випливає з того, що

$$\overline{\forall x P(x)} = \overline{P(a_1) \wedge P(a_2) \wedge ... \wedge P(a_n)} = \\ = \overline{P}(a_1) \vee \overline{P}(a_2) \vee ... \vee \overline{P}(a_n);$$

остання формула еквівалентна $\exists x \overline{P}(x)$.

Аналогічно,

$$\overline{\exists x P(x)} = \overline{P(a_1) \vee P(a_2) \vee ... \vee P(a_n)} = \\ = \overline{P}(a_1) \wedge \overline{P}(a_2) \wedge ... \wedge \overline{P}(a_n),$$

що еквівалентно $\forall x \overline{P}(x)$.

1.6. Випереджена нормальна форма

Говорять, що формулу логіки першого ступеня записано у *випередженій нормальній формі*, якщо вона має вигляд $Q_1 x_1 Q_2 x_2 ... Q_n x_n M$, де кожне $Q_i x_i$ ($i = 1, 2, ..., n$) — це $\forall x_i$ або $\exists x_i$, а формула M не містить кванторів. Вираз $Q_1 x_1 ... Q_n x_n$ називають *префіксом*, а M — *матрицею* формули, записаної у випередженій нормальній формі [34, 48].

Приклад 1.44. Наведемо приклади формул, записаних у випередженій нормальній формі.

1. $\forall x \forall y (P(x, y) \wedge Q(y))$.
2. $\forall x \exists y (\overline{P}(x) \vee Q(y))$.
3. $\forall x \forall y \exists z (Q(x, y) \wedge R(z))$.
4. $\forall x \exists y \forall z ((P(x, y) \vee Q(x, z)) \wedge \overline{R}(y, z))$.
5. $\forall x \forall y \forall z \exists u (\overline{P}(x, z) \vee \overline{P}(y, z) \vee Q(x, y, u))$.

Наведемо алгоритм зведення довільної формули логіки першого ступеня до випередженої нормальної форми.

Крок 1. Усунути з формули логічні операції „~" та „→" застосуванням еквівалентних формул $P \sim Q = (P \to Q) \wedge (Q \to P)$ та $P \to Q = \overline{P} \vee Q$.

Крок 2. Унести знак заперечення всередину формули безпосередньо до атома за допомогою таких законів:

- подвійного заперечення $\overline{\overline{P}} = P$;
- де Моргана $\overline{P \vee Q} = \overline{P} \wedge \overline{Q}$ та $\overline{P \wedge Q} = \overline{P} \vee \overline{Q}$;
- $\overline{\forall x P(x)} = \exists x \overline{P}(x)$ та $\overline{\exists x P(x)} = \forall x \overline{P}(x)$.

Перейменувати зв'язані змінні, якщо це потрібно,

Крок 3. Винести квантори в префікс, скориставшись законами 3–8 з підрозділу 1.5.

Приклад 1.45. Зведемо формулу $\forall x P(x) \to \exists y Q(y)$ до випередженої нормальної форми за умови, що предикати $P(x)$ і $Q(y)$ не містять вільних змінних. Наведемо послідовність формул, отриманих у процесі побудови випередженої нормальної форми:

$$\forall x P(x) \to \exists y Q(y) = \overline{\forall x P(x)} \vee \exists y Q(y)$$

(виключено логічну операцію „→")

$$= \exists x \overline{P}(x) \vee \exists y Q(y)$$

(застосовано закон $\overline{\forall x P(x)} = \exists x \overline{P}(x)$)

$$= \exists x \exists y (\overline{P}(x) \vee Q(y))$$

(квантори існування винесено у префікс за законом 8 з підрозділа 1.5).

Приклад 1.46. Побудуємо випереджену нормальну форму для формули $\forall x \forall y (\exists z P(x, z) \wedge \wedge P(y, z)) \to \exists u Q(x, y, u)$. Нижче подано процес побудови формули:

$$\forall x \forall y (\exists z (P(x, z) \wedge P(y, z)) \to \exists u Q(x, y, u)) = \forall x \forall y (\overline{\exists z (P(x, z) \wedge P(y, z))} \vee \exists u Q(x, y, u))$$

(виключено логічну операцію „→")

$$= \forall x \forall y \left(\overline{\forall z (P(x,z) \wedge P(y,z))} \vee \exists u Q(x,y,u) \right)$$

(застосовано закон $\overline{\exists x P(x)} = \forall x \overline{P}(x)$)

$$= \forall x \forall y (\forall z (\overline{P}(x,z) \vee \overline{P}(y,z)) \vee \exists u Q(x,y,u))$$

(застосовано закон де Моргана)

$$= \forall x \forall y \forall z \exists u (\overline{P}(x,z) \vee \overline{P}(y,z) \vee Q(x,y,u))$$

(за законами 6 та 8 підрозділа 1.5 у префікс формули винесено $\forall z$ та $\exists u$).

1.7. Логічне виведення в логіці висловлювань

Говорять, що формула g — *логічний наслідок* формул $f_1, f_2, ..., f_n$, або що g *логічно випливає* з $f_1, f_2, ..., f_n$, якщо в кожній інтерпретації, у якій виконується формула $f_1 \wedge f_2 \wedge ... \wedge f_n$, формула g також виконується. Формули $f_1, f_2, ..., f_n$ називають *гіпотезами* (*аксіомами*, *постулатами* чи *засновками*) формули g. Той факт, що формула g логічно випливає з $f_1, f_2, ..., f_n$, позначають $f_1, f_2, ..., f_n \vdash g$.

ТЕОРЕМА 1.2. Формула g — логічний наслідок формул $f_1, f_2, ..., f_n$ тоді й лише тоді, коли формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ загальнозначуща.

Доведення. Необхідність. Нехай g — логічний наслідок формул $f_1, f_2, ..., f_n$ та I — довільна їх інтерпретація. Якщо формули $f_1, f_2, ..., f_n$ істинні в інтерпретації I, то за означенням логічного наслідку формула g також істинна в I. Звідси випливає, що формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ істинна в інтерпретації I. З іншого боку, якщо не всі формули з $f_1, f_2, ..., f_n$ істинні в інтерпретації I, тобто принаймні одна з них хибна в I, то формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ істинна в I. Отже, формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ істинна в довільній інтерпретації, або $\models ((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$.

Достатність. Припустимо, що формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ загальнозначуща. Тоді якщо формула $(f_1 \wedge f_2 \wedge ... \wedge f_n)$ істинна в якійсь інтерпретації, то й формула g має бути істинною в цій інтерпретації, тобто g — логічний наслідок формул $f_1, f_2, ..., f_n$.

Якщо g — логічний наслідок формул $f_1, f_2, ..., f_n$, то формулу $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ називають *логічною теоремою*, а g — її висновком. У такому разі говорять, що формулу g можна *вивести* з формул $f_1, f_2, ..., f_n$ і g — *вивідна* формула. Вираз $f_1, f_2, ..., f_n \vdash g$ називають *правилом виведення*. Тут гіпотези записано зліва від знака $\vdash$, а висновок — справа; сам знак $\vdash$ має зміст „отже".

ТЕОРЕМА 1.3 (принцип прямої дедукції). Формула g — логічний наслідок формул $f_1, f_2, ..., f_n$ тоді й лише тоді, коли $(f_1 \wedge f_2 \wedge ... \wedge f_n \wedge \bar{g})$ — суперечність.

Доведення. За теоремою 1.2 формула g — логічний наслідок формул $f_1, f_2, ..., f_n$ тоді й лише тоді, коли формула $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ загальнозначуща. Отже, g — логічний наслідок формул $f_1, f_2, ..., f_n$ тоді й лише тоді, коли заперечення формули $((f_1 \wedge f_2 \wedge ... \wedge f_n) \to g)$ — суперечність. Справді,

$$\overline{(f_1 \wedge f_2 \wedge ... \wedge f_n) \to g} = \overline{\overline{(f_1 \wedge f_2 \wedge ... \wedge f_n)} \vee g} =$$
$$= \overline{\bar{f}_1 \vee \bar{f}_2 \vee ... \vee \bar{f}_n \vee g} = f_1 \wedge f_2 \wedge ... \wedge f_n \wedge \bar{g}.$$

Приклад 1.47. Розглянемо формули $f_1 = (p \to q)$, $f_2 = \bar{q}$, $g = \bar{p}$. Доведемо, що формула g — логічний наслідок формул f_1 і f_2.

Спосіб 1. Скористаємося таблицями істинності, щоб показати, що формула g виконується в кожній інтерпретації, у якій виконується формула $(p \to q) \wedge \bar{q}$. Із табл. 1.10 видно, що є лише одна інтерпретація, у якій $(p \to q) \wedge \bar{q}$ виконується, а саме $p = \mathrm{F}$, $q = \mathrm{F}$; у цій інтерпретації формула $\bar{p}$ також виконується. Отже, за означенням формула $\bar{p}$ — логічний наслідок формул $p \to q$ та $\bar{q}$.

Таблиця 1.10

p	q	$p \to q$	$\bar{q}$	$(p \to q) \wedge \bar{q}$	$\bar{p}$
T	T	T	F	F	F
T	F	F	T	F	F
F	T	T	F	F	T
F	F	T	T	T	T

Спосіб 2. Скористаємося теоремою 1.2. Доведемо, що формула $(f_1 \wedge f_2) \to g$ загальнозначуща. Для цього побудуємо табл. 1.11 для формули $(f_1 \wedge f_2) \to g = ((p \to q) \wedge \bar{q}) \to \bar{p}$.

Таблиця 1.11

p	q	$((p \to q) \wedge \bar{q}) \to \bar{p}$
T	T	T
T	F	T
F	T	T
F	F	T

Оскільки формула $((p \to q) \wedge \bar{q}) \to \bar{p}$ загальнозначуща, то формула $\bar{p}$ – логічний наслідок формул $p \to q$ та $\bar{q}$.

Спосіб 3. Скористаємося теоремою 1.3 та доведемо, що формула

$$(f_1 \wedge f_2) \to \bar{g} = ((p \to q) \wedge \bar{q}) \wedge \bar{\bar{p}} = ((p \to q) \wedge \bar{q}) \wedge p$$

заперечувана. Побудуємо таблицю істинності для цієї формули. Із табл. 1.12 видно, що формула $(p \to q) \wedge \bar{q} \wedge p$ хибна в будь-якій інтерпретації.

Таблиця 1.12

p	q	$p \to q$	$\bar{q}$	$(p \to q) \wedge \bar{q}$	$((p \to q) \wedge \bar{q}) \wedge p$
T	T	T	F	F	F
T	F	F	T	F	F
F	T	T	F	F	F
F	F	T	T	T	F

Тепер відповідно до теореми 1.3 можна дійти висновку, що формула $\bar{p}$ логічно випливає з формул $p \to q$ та $\bar{q}$.

1.8. Застосування правил виведення в логіці висловлювань

Розглянемо правила виведення та їх застосування в логіці висловлювань. Ці правила обґрунтовують кроки *доведення логічних теорем*, яке полягає в перевірці того, що висновок являє собою логічний наслідок множини гіпотез.

Деякі важливі правила виведення та відповідні їм тавтології наведено в табл. 1.13. Для прикладу, правило виведення modus ponens має вигляд $p, p \to q \vdash q$. Згідно з теоремою 1.2, воно ґрунтується на тавтології $(p \wedge (p \to q)) \to q$.

Таблиця 1.13

Правило виведення	Тавтологія	Назва правила виведення
$p \vdash p \vee q$	$p \to (p \vee q)$	Уведення диз'юнкції
$p \wedge q \vdash p$	$(p \wedge q) \to p$	Виключення кон'юнкції
$p, q \vdash p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Уведення кон'юнкції
$p, p \to q \vdash q$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\bar{q}, p \to q \vdash \bar{p}$	$(\bar{q} \wedge (p \to q)) \to \bar{p}$	Modus tollens
$p \to q, q \to r \vdash p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Гіпотетичний силогізм

Правило виведення	Тавтологія	Назва правила виведення
$p \vee q, \bar{p} \vdash q$	$((p \vee q) \wedge \bar{p}) \to q$	Диз'юнктивний силогізм
$p \vee q, \bar{p} \vee r \vdash q \vee r$	$((p \vee q) \wedge (\bar{p} \vee r)) \to (q \vee r)$	Резолюція

Приклад 1.48. Припустимо, що імплікація „Якщо падає сніг, то ми катаємося на лижах" і її гіпотеза „Падає сніг" істинні. Тоді за правилом modus ponens висновок імплікації „Ми катаємося на лижах" також істинний.

Приклад 1.49. Нехай істинна імплікація: „Якщо $n > 3$, то $n^2 > 9$". Отже, якщо $n > 3$, то за правилом modus ponens висновок $n^2 > 9$ правильний для цього n.

Далі наведено декілька прикладів міркувань із використанням правил виведення, наведених у табл. 1.13.

Приклад 1.50. З'ясуємо, яке правило виведення використано в такому міркуванні: „Похолоднішало. Отже, похолоднішало чи почав падати дощ."

Нехай p — висловлювання „Похолоднішало", а q — висловлювання „Почав падати дощ". Тоді це твердження можна записати у вигляді правила *введення диз'юнкції* $p \vdash p \vee q$.

Приклад 1.51. З'ясуємо, яке правило виведення використано в такому міркуванні: „Похолоднішало та почав падати дощ. Отже, похолоднішало."

Нехай p: „Похолоднішало", а q: „Почав падати дощ". Тоді це твердження можна записати у вигляді правила *виключення кон'юнкції* $p \wedge q \vdash p$.

Приклад 1.52. З'ясуємо, яке правило виведення використано в такому міркуванні. „Якщо сьогодні падатиме дощ, то сьогодні ми не поїдемо на пікнік. Якщо ми не поїдемо на пікнік сьогодні, то поїдемо на пікнік завтра. Отже, якщо сьогодні падатиме дощ, то ми поїдемо на пікнік завтра".

Нехай p: „Сьогодні падатиме дощ", q: „Сьогодні ми не поїдемо на пікнік", а r: „Ми поїдемо на пікнік завтра". Тоді це твердження можна записати у вигляді правила гіпотетичного силогізму $p \to q, q \to r \vdash p \to r$.

Якщо твердження містить багато гіпотез, то потрібно застосувати декілька правил виведення для того, щоб довести істинність висновку. Проілюструємо це такими прикладами.

Приклад 1.53. Доведемо, що з гіпотез „Сьогодні не сонячний день і холодніше, ніж учора", „Ми підемо купатися лише якщо сьогодні сонячний день", „Якщо ми не підемо купатись, то поїдемо плавати на човні" та „Якщо ми поїдемо плавати на човні, то повернемося пізно ввечері" випливає висновок „Ми повернемося пізно ввечері".

Нехай p: „Сьогодні сонячний день", q: „Сьогодні холодніше, ніж учора", r: „Ми підемо купатися", s: „Ми поїдемо плавати на човні", t: „Ми повернемося пізно ввечері". Тут гіпотези — $\bar{p} \wedge q$, $r \to p$, $\bar{r} \to s$ і $s \to t$, а висновок — t.
Нижче наведено послідовність кроків отримання висновку із заданої множини гіпотез і зазначено застосовані правила виведення.

1. $\bar{p} \wedge q$ — гіпотеза.
2. $\bar{p}$ — правило виключення кон'юнкції до 1.
3. $r \to p$ — гіпотеза.
4. $\bar{r}$ — modus tollens до 2 та 3.
5. $\bar{r} \to s$ — гіпотеза.

6. s — modus ponens до 4 та 5.
7. $s \to t$ — гіпотеза.
8. t — modus ponens до 6 та 7.

Висновок доведено.

Приклад 1.54. Доведемо, що з гіпотез „Якщо ти надішлеш мені повідомлення електронною поштою, то я закінчу писати програму", „Якщо ти не надішлеш мені повідомлення електронною поштою, то я рано піду спати" та „Якщо я рано піду спати, то прокинуся бадьорим" випливає висновок „Якщо я не закінчу писати програму, то я прокинуся бадьорим".

Уведемо такі позначення: p: „Ти надішлеш мені повідомлення електронною поштою", q: „Я закінчу писати програму", r: „Я рано піду спати", s: „Я прокинуся бадьорим". Гіпотези можна записати у вигляді $p \to q$, $\bar{p} \to r$, $r \to s$. Потрібно обґрунтувати висновок $\bar{q} \to s$.

Нижче наведено послідовність кроків для отримання висновку із заданої множини гіпотез. Застосовані правила виведення записано справа.

1. $p \to q$ — гіпотеза.
2. $\bar{q} \to \bar{p}$ — контрапозиція (див. приклад 1.17).
3. $\bar{p} \to r$ — гіпотеза.
4. $\bar{q} \to r$ — гіпотетичний силогізм до 2 та 3.
5. $r \to s$ — гіпотеза.
6. $\bar{q} \to s$ — гіпотетичний силогізм до 4 та 5.

Висновок доведено.

1.9. Метод резолюцій

Існують комп'ютерні програми, котрі розроблено для автоматизації міркувань, виконуваних за допомогою доведення логічних теорем. У багатьох із цих програм використано правило виведення, відоме як резолюція. *Правило резолюції* записують у вигляді $p \vee q, \bar{q} \vee r \vdash p \vee r$. На основі цього правила Дж. Робінсон (G. Robinson) 1965 р. запропонував *метод резолюцій* автоматичного доведення логічних теорем.

Нехай формулу f записано в КНФ (див. підрозділ 1.3)

$$f = d_1 \wedge d_2 \wedge \ldots \wedge d_m.$$

Тут кожна з формул d_i ($i = 1, 2, \ldots, m$) — літерал або диз'юнкція літералів. Формулу d_i називають *елементарною диз'юнкцією*, *диз'юнктом* або *клаузою*. Кількість літералів у формулі d_i називають *рангом* елементарної диз'юнкції. Розглядають також елементарну диз'юнкцію з рангом 0, яку позначають □; така диз'юнкція не містить жодного літералу. За означенням елементарну диз'юнкцію з рангом 0 уважають такою, що дорівнює F.

За принципом прямої дедукції формулу g можна вивести з формул $f_1, f_2, \ldots, f_n$ тоді й лише тоді, коли $f_1 \wedge f_2 \wedge \ldots \wedge f_n \wedge \bar{g}$ — суперечність. Так буде, якщо одна з формул $f_1, f_2, \ldots, f_n$ хибна чи формула g істинна. Нехай формулу $f = f_1 \wedge f_2 \wedge \ldots \wedge f_n \wedge \bar{g}$ записано в КНФ $f = d_1 \wedge d_2 \wedge \ldots \wedge d_m$, де $d_1, d_2, \ldots, d_m$ — її елементарні диз'юнкції. Очевидно, що цю КНФ можна записати у вигляді множини її елементарних диз'юнкцій $S = \{d_1, d_2, \ldots, d_m\}$. Множину S називають *невиконанною*, якщо формула f невиконанна.

Припустимо, що елементарні диз'юнкції d_1 і d_2 такі, що d_1 містить літерал l_1, контрарний до літералу l_2 з d_2. Викреслимо літерал l_1 із d_1 та l_2 з d_2; побудуємо диз'юнкцію решти літералів цих елементарних диз'юнкцій. Отриману елементарну диз'юнкцію називають *резольвентою*.

Приклад 1.55. Побудуємо резольвенту пари елементарних диз'юнкцій $d_1 = p \vee r$ і $d_2 = \bar{p} \vee q$. Ці елементарні диз'юнкції містять контрарну пару літералів p та $\bar{p}$, які можна викреслити з d_1 і d_2. Утворимо диз'юнкцію літералів, що залишилися. Одержимо резольвенту $r \vee q$.

ТЕОРЕМА 1.4. Резольвента d елементарних диз'юнкцій d_1 і d_2 — логічний наслідок диз'юнкцій d_1 і d_2.

Доведення. Нехай $d_1 = l \vee d_1'$, $d_2 = \bar{l} \vee d_2'$, $d = d_1' \vee d_2'$, де d_1' і d_2' — диз'юнкції літералів, відмінних від l і $\bar{l}$. Нехай диз'юнкції d_1 і d_2 істинні в інтерпретації I. Доведемо, що резольвента d елементарних диз'юнкцій d_1 і d_2 також істинна в цій інтерпретації. Зазначимо, що один із літералів l або $\bar{l}$ хибний в I. Нехай це літерал l. Тоді d_1 не може бути елементарною диз'юнкцією з рангом 1, бо тоді вона була б хибною в інтерпретації I. Це означає, що диз'юнкція d_1' істинна в інтерпретації I, а отже, і резольвента $d = d_1' \vee d_2'$ істинна в I. Аналогічно доведемо, що якщо літерал $\bar{l}$ хибний в інтерпретації I, то диз'юнкція d_2' істинна в I. Звідси одержимо, що диз'юнкція $d = d_1' \vee d_2'$ істинна в інтерпретації I.

Приклад 1.56. Нехай $f_1 = x \vee z$, $f_2 = y \vee \bar{z}$, $g = x \vee y$. Доведемо, що $f_1, f_2 \vdash g$. Формулу g отримано вилученням контрарної пари літералів $\{z, \bar{z}\}$ із диз'юнкції формул f_1 і f_2. Це резольвента формул f_1 і f_2, тому $f_1, f_2 \vdash g$.

Виведення формули d з елементарних диз'юнкцій множини S за методом резолюцій полягає в побудові такої скінченної послідовності елементарних диз'юнкцій $d_1', d_2', ..., d_k'$, що кожна d_i' $(i = 1, 2, ..., k)$ являє собою або елемент множини S, або резольвенту елементарних диз'юнкцій, які передують d_i', причому $d = d_k'$.

Елементарну диз'юнкцію d можна *вивести* з множини S, якщо існує виведення d з S.

Головну ідею методу резолюцій формулюють так: перевірити, чи містить множина елементарних диз'юнкцій S диз'юнкцію $\square$. Якщо S містить $\square$, то множина S невиконанна. Якщо S не містить $\square$, то перевіряють, чи можна вивести $\square$ із множини S. Виведення $\square$ з S називають *доведенням невиконанності множини* S або *спростуванням* S.

Алгоритм методу резолюцій

Задано множину гіпотез $f_1, f_2, ..., f_n$ і висновок g. Алгоритм дає змогу визначити, чи являє собою формула g логічний наслідок множини гіпотез.

Крок 1. Побудувати кон'юнкцію множини гіпотез $f_1, f_2, ..., f_n$ і заперечення висновку $\bar{g}$ у вигляді $f_1 \wedge f_2 \wedge ... \wedge f_n \wedge \bar{g}$. Звести отриману формулу до КНФ і записати множину її елементарних диз'юнкцій S.

Крок 2. Записати кожну елементарну диз'юнкцію множини S в окремому рядку.

Крок 3. Вибрати дві елементарні диз'юнкції, які містять контрарну пару літералів, і побудувати їх резольвенту. Записати одержану резольвенту в новому рядку, якщо в попередніх рядках іще немає такої елементарної диз'юнкції.

Крок 4. Крок 3 виконувати до отримання диз'юнкції з рангом 0. Одержання елементарної диз'юнкції з рангом 0 свідчить про те, що формулу g можна вивести з $f_1, f_2, ..., f_n$. Якщо неможливо отримати резольвенту, відмінну від елементів множини S і вже побудованих резольвент, то множина S неспростовна. Кінець.

Приклад 1.57. Покажемо невиконанність множини $S = \{\bar{p} \vee q, \bar{q}, p\}$ за допомогою методу резолюцій. Запишемо кожну елементарну диз'юнкцію в окремому рядку та перенумеруємо їх:

(1) $\bar{p} \vee q$;
(2) $\bar{q}$;
(3) p;
(4) $\bar{p}$.

Резольвенту (4) отримано з елементарних диз'юнкцій (1) і (2). Далі,

(5) $\square$.

Резольвента (5) — диз'юнкція з рангом 0; її одержано з елементарної диз'юнкції (3) та резольвенти (4). Виведення диз'юнкції з рангом 0 із множини S спростовує цю множину.

Приклад 1.58. Доведемо логічний наслідок формули $p \vee q, p \to r, q \to s \vdash r \vee s$ методом резолюцій. За принципом прямої дедукції побудуємо формулу, невиконанність якої потрібно довести. Вона має вигляд $(p \vee q) \wedge (p \to r) \wedge (q \to s) \wedge \overline{(r \vee s)}$. Запишемо гіпотези у вигляді елементарних диз'юнкцій і випишемо кожну з них в окремому рядку:

(1) $p \vee q$;
(2) $\bar{p} \vee r$;
(3) $\bar{q} \vee s$.

Оскільки заперечення висновку $\overline{r \vee s} = \bar{r} \wedge \bar{s}$ являє собою дві елементарні диз'юнкції $\bar{r}$ і $\bar{s}$ із рангом 1, то їх теж випишемо в окремих рядках:

(4) $\bar{r}$;
(5) $\bar{s}$.

Послідовно побудувавши всі можливі резольвенти методом резолюцій, виведемо диз'юнкцію з рангом 0. Біля кожної резольвенти випишемо номери елементарних диз'юнкцій, з яких її отримано:

(6) $\bar{p}$ (2), (4);
(7) q (6), (1);
(8) $\bar{q}$ (3), (5);
(9) $\square$ (7), (8).

Одержання диз'юнкції з рангом 0 доводить теорему.

1.10. Правила виведення в численні предикатів

Розглянемо деякі важливі правила виведення для формул із кванторами.

Універсальна конкретизація — це правило виведення того, що $P(c)$ істинне для довільного елемента c з предметної області за умови, що формула $\forall x P(x)$ істинна. Наприклад, універсальну конкретизацію можна використати тоді, коли з твер-

дження „Всі люди смертні" потрібно дійти висновку „Сократ — смертний". Тут Сократ — елемент предметної області, яка складається з усіх людей.

Універсальне узагальнення — це правило виведення, згідно з яким $\forall x P(x)$ істинне, якщо істинне $P(c)$ для довільного c з предметної області. Це правило використовують тоді, коли на підставі істинності $P(c)$ для кожного елемента c з предметної області твердять, що $\forall x P(x)$ істинне. Вибраний елемент c має бути довільним і не конкретизованим. Універсальне узагальнення неявно застосовують у багатьох математичних доведеннях і рідко згадують явно.

Екзистенційна конкретизація — це правило, яке дає змогу дійти висновку про те, що на підставі істинності $\exists x P(x)$ можна твердити, що в предметній області є елемент c, для якого $P(c)$ істинне. Зазвичай про елемент c відомо тільки те, що він існує. Із цього випливає, що можна позначити його та продовжувати міркування.

Екзистенційне узагальнення — це правило виведення, використовуване для того, щоб на підставі істинності $P(c)$ на якомусь елементі c з предметної області дійти висновку, що $\exists x P(x)$ істинне.

Правила виведення в численні предикатів зазначено в табл. 1.14.

Таблиця 1.14

Правило виведення	Назва
1. $\forall x P(x) \vdash P(c)$	Універсальна конкретизація
2. $P(c) \vdash \forall x P(x)$	Універсальне узагальнення
3. $\exists x P(x) \vdash P(c)$	Екзистенційна конкретизація
4. $P(c) \vdash \exists x P(x)$	Екзистенційне узагальнення

У правилах 1 і 2 елемент c предметної області довільний, а в правилах 3 та 4 в предметній області має бути принаймні один такий елемент.

Приклад 1.59. Доведемо, що гіпотези „Кожний, хто вивчає комп'ютерні науки, слухає курс дискретної математики" та „Марія вивчає комп'ютерні науки" дають змогу сформулювати висновок „Марія слухає курс дискретної математики".

Нехай $D(x)$: „x вивчає комп'ютерні науки", $C(x)$: „x слухає курс дискретної математики". Тоді гіпотези — це формули $\forall x(D(x) \to C(x))$ і D(Марія), а висновок — C(Марія). Доведення висновку для введеної множини гіпотез виконаємо в такій послідовності.

1. $\forall x(D(x) \to C(x))$ — гіпотеза.
2. D(Марія) $\to C$(Марія) — універсальна конкретизація до 1.
3. D(Марія) — гіпотеза.
4. C(Марія) — modus ponens до 2 та 3.

Приклад 1.60. Доведемо, що з гіпотез „У групі є студент, який не читав підручника" та „Всі студенти групи склали іспит" можна сформулювати висновок „Дехто з тих, хто склав іспит, не читав підручника".

Нехай $C(x)$: „x учиться в групі", $B(x)$: „x читав підручник" і $P(x)$: „x склав іспит". Гіпотези — це $\exists x(C(x) \wedge \overline{B}(x))$ і $\forall x(C(x) \to P(x))$, а висновок — $\exists x(P(x) \wedge \overline{B}(x))$. Доведення — це така послідовність кроків.

1. $\exists x(C(x) \wedge \overline{B}(x))$ — гіпотеза.
2. $C(a) \wedge \overline{B}(a)$ — екзистенційна конкретизація до 1.
3. $C(a)$ — виключення кон'юнкції до 2.

4. $\forall x(C(x) \to P(x))$ — гіпотеза.
5. $C(a) \to P(a)$ — універсальна конкретизація до 4.
6. $P(a)$ — modus ponens до 3 та 5.
7. $\overline{B}(a)$ — виключення кон'юнкції до 2.
8. $P(a) \wedge \overline{B}(a)$ — уведення кон'юнкції до 6 і 7.
9. $\exists x(P(x) \wedge \overline{B}(x))$ — екзистенційне узагальнення до 8.

1.11. Методи доведення теорем

Доведення теорем може бути доволі складним. Розглянемо різні методи доведення. Оскільки багато теорем мають вигляд імплікації, потрібно вміти доводити тавтологічність імплікації. Повторимо, що $p \to q$ істинне, окрім випадку, коли p істинне, а q — хибне. Розглянемо найзагальніші методи доведення.

Пряме доведення

Тавтологічність імплікації $p \to q$ можна довести, переконавшись, що коли припущення імплікації p істинне, то й висновок q також істинний.

Доведення від протилежного

Можна довести (див. задачу 5), що імплікація $p \to q$ еквівалентна кожній із формул $\overline{q} \to \overline{p}$, $(p \wedge \overline{q}) \to \overline{p}$, $(p \wedge \overline{q}) \to q$, $(p \wedge \overline{q}) \to$ F, де F — значення „хибність". Тому замість доведення тавтологічності $p \to q$ можна довести тавтологічність однієї з чотирьох наведених формул. Розглянемо, наприклад, імплікацію $\overline{q} \to \overline{p}$. За умови істинності $\overline{q}$ потрібно довести істинність $\overline{p}$. Це найпростіший спосіб доведення теореми $p \to q$ від протилежного: ми припускаємо протилежне до того, що потрібно довести, й одержуємо суперечність із тим, що дано.

У разі доведення на основі решти трьох формул ми беремо до уваги водночас і те, що дано (p), і протилежне до того, що потрібно довести ($\overline{q}$), тобто $(p \wedge \overline{q})$. Тоді для доведення теореми $p \to q$ достатньо отримати суперечність із тим, що дано ($\overline{p}$), або вивести те, що потрібно довести (q), або, нарешті, одержати суперечність $F = r \wedge \overline{r}$. Отже, в останньому випадку з $(p \wedge \overline{q})$ достатньо вивести якесь висловлювання r і його заперечення $\overline{r}$ (бо тоді мало б бути істинним висловлювання $r \wedge \overline{r}$, що неможливо). Останній спосіб доведення від протилежного в певному розумінні найзагальніший.

Доведення аналізом випадків

Іноді для доведення тавтологічності імплікації $p \to q$ зручно використати замість p диз'юнкцію $(p_1 \vee p_2 \vee \ldots \vee p_n)$ як припущення імплікації, якщо p та $(p_1 \vee p_2 \vee \ldots \vee p_n)$ еквівалентні.

На основі логічної еквівалентності

$$(p_1 \vee p_2 \vee \ldots \vee p_n) \to q = (p_1 \to q) \wedge (p_2 \to q) \wedge \ldots \wedge (p_n \to q)$$

доведення тавтологічності імплікації

$$(p_1 \vee p_2 \vee \ldots \vee p_n) \to q$$

можна замінити доведенням тавтологічності кожної з n імплікацій $p_i \to q$, $i = 1, 2, \ldots, n$ окремо.

1.12. Множина. Кортеж. Декартів добуток

Теорію множин, основи якої викладено в цьому розділі, у математиці називають наївною [17]. Є й інші варіанти побудови теорії множин, наприклад конструктивний і формалістський, у яких поняття множини вводять інакше (у конструктивній теорії множин — означають). У нас поняття множини первісне, тобто неозначуване. Опишемо це поняття так: *множиною* називають будь-який набір певних відмінних один від одного об'єктів нашої інтуїції чи інтелекту, розглядуваних як єдине ціле. Відповідно до цього опису вивчають не окремі об'єкти, а їх сукупності як певні утворення.

У математиці застосовують і такі синоніми терміна „множина": система, клас, область, сукупність. Використовують також поняття „сім'я", але ми вживатимемо його в іншому значенні (див. розділ 3).

Об'єкти, які утворюють множину, називають її *елементами*. Про множину говорять, що вона містить ці елементи. Якщо об'єкт a — елемент множини A, то пишемо $a \in A$, а ні, то $a \notin A$.

Множину можна задати, навівши її елементи у фігурних дужках. Наприклад, множина $A = \{a, e, i, o, u\}$ містить елементи a, e, i, o, u й лише ці елементи. Множина не може містити двох однакових елементів, а порядок її елементів не фіксують.

Для часто використовуваних множин є спеціальні позначення:

- $\varnothing$ — *порожня* множина, яка не містить жодного елемента;
- Z — множина *цілих чисел*, $Z = \{..., -2, -1, 0, 1, 2, ...\}$;
- R — множина *дійсних чисел*;
- N — множина *натуральних чисел*, $N = \{1, 2, ...\}$;
- N_0 — множина *натуральних чисел із числом* 0, $N_0 = \{0, 1, 2, ...\}$.

Можна задати множину, зазначивши спільну властивість всіх її елементів. Тоді множину A задають за допомогою позначення $A = \{x \mid P(x)\}$, яке читають так: „A — це множина об'єктів x, які мають властивість $P(x)$". Наприклад, $A = \{x \mid x \in N_0, x < 7\}$ — це множина $\{0, 1, 2, 3, 4, 5, 6\}$. Іноді замість вертикальної риски використовують дві крапки, тобто $A = \{x : x \in N_0, x < 7\}$.

Дві множини A та B називають *рівними*, якщо вони складаються з одних і тих самих елементів. Рівність множин A та B записують як $A = B$.

Множину A називають *підмножиною* множини B, якщо кожний елемент множини A належить B. У такому разі пишуть $A \subset B$, причому може бути $A = B$. Якщо $A = B$ чи $A = \varnothing$, то A називають *невласною* підмножиною множини B, а ні, то *власною*. Для будь-якої множини A правдиве включення $\varnothing \subset A$.

Множини бувають скінченними й нескінченними. *Скінченною* називають множину, для якої існує натуральне число, що дорівнює кількості її елементів. Множину, яка не є скінченною, називають *нескінченною*. Кількість елементів скінченної множини A позначають як $|A|$ і називають *потужністю*. Поняття потужності вводять і для нескінченних множин, але ми не будемо розглядати його.

Часто всі досліджувані множини являють собою підмножини якоїсь множини, називаної *універсальною множиною*, чи *універсумом*. Універсальну множину позначають як U.

Множини можна зображати графічно за допомогою *діаграм Венна*, які запровадив 1881 р. англійський математик Дж. Венн (J. Venn). Універсальну множину позначають прямокутником, а всі інші множини — кругами в ньому.

Для заданої множини A можна розглянути множину всіх її підмножин, зокрема порожню множину $\varnothing$ і саму множину A. Цю множину позначають 2^A чи $P(A)$ й називають *множиною-степенем*, чи *булеаном* множини A. Для скінченної множини A множина 2^A містить $2^{|A|}$ елементів.

Приклад 1.61. Нехай $A = \{0, 1, 2\}$. Тоді $2^A = \{\varnothing, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$. Ця множина містить $2^3 = 8$ елементів.

Кортеж — це впорядкований набір елементів. Це не означення кортежу, бо не пояснено, що таке впорядкований набір. Уважатимемо поняття „кортеж" (*вектор, рядок, ланцюжок*), як і поняття множини, первісним, неозначуваним. Елементи, що утворюють кортеж, називають його *компонентами*. Компоненти нумерують, кількість компонент називають *довжиною* (*розмірністю*) кортежу. Нескінченні кортежі не розглядатимемо.

На відміну від елементів множини, компоненти кортежу можуть повторюватись. Кортеж записують у круглих дужках, наприклад (a, b, c, a, d) — кортеж довжиною 5. Іноді дужки й навіть коми не пишуть, наприклад 011001. Кортежі довжиною 2 часто називають *парами*, довжиною 3 — *трійками*, довжиною n — *n-ками* („*енками*").

Два кортежі рівні, якщо вони мають однакову довжину та відповідні їх компоненти рівні. Інакше кажучи, кортежі $(a_1, ..., a_m)$ і $(b_1, ..., b_n)$ рівні, якщо $m = n$ та $a_1 = b_1$, $a_2 = b_2$, ..., $a_m = b_n$.

Декартовим добутком множин A та B (позначають $A \times B$) називають множину всіх таких пар (a, b), що $a \in A$, $b \in B$. Зокрема, якщо $A = B$, то обидві компоненти належать A. Такий добуток позначають як A^2 та називають *декартовим квадратом* множини A. Аналогічно, декартовим добутком n множин A_1, ..., A_n (позначають $A_1 \times ... \times A_n$) називають множину всіх таких кортежів $(a_1, ..., a_n)$ довжиною n, що $a_1 \in A_1$, ..., $a_n \in A_n$. Частковий випадок $A \times ... \times A$ позначають як A^n і називають *n-м степенем* множини A.

Приклад 1.62. Нехай $A = \{1, 2\}$, $B = \{a, b, c\}$. Тоді

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\},$$
$$B \times A = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}.$$

Зрозуміло, що загалом $A \times B \neq B \times A$.

Для скінченних множин потужність (кількість елементів) декартового добутку дорівнює добутку потужностей цих множин: $|A \times B| = |A| \cdot |B|$.

1.13. Операції над множинами. Доведення рівностей з множинами

Будемо вважати, що всі розглядувані множини — підмножини якось універсуму U. Для довільних множин A та B можна побудувати нові множини за допомогою *теоретико-множинних операцій*:

- об'єднанням множин A та B називають множину $A \cup B = \{x \mid (x \in A) \vee (x \in B)\}$;
- перетином множин A та B називають множину $A \cap B = \{x \mid (x \in A) \wedge (x \in B)\}$;
- різницею множин A та B називають множину $A \setminus B = \{x \mid (x \in A) \wedge (x \notin B)\}$;
- доповненням множини A називають множину $\overline{A} = U \setminus A$, де U — універсальна множина.

На рис. 1.1 подано діаграми Венна, які ілюструють операції над множинами.

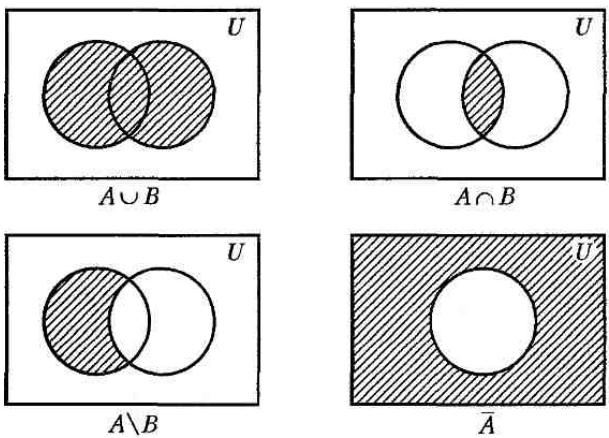


Рис. 1.1

Теорстико-множинні операції задовольняють законам, наведеним у табл. 1.15.

Таблиця 1.15

Назва закону	Формулювання закону
1. Закони комутативності	*а*) $A \cup B = B \cup A$
	б) $A \cap B = B \cap A$
2. Закони асоціативності	*а*) $A \cup (B \cup C) = (A \cup B) \cup C$
	б) $A \cap (B \cap C) = (A \cap B) \cap C$
3. Закони дистрибутивності	*а*) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
	б) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
4. Закон подвійного доповнення	$\overline{(\overline{A})} = A$
5. Закони ідемпотентності	*а*) $A \cap A = A$
	б) $A \cup A = A$
6. Закони де Моргана	*а*) $\overline{A \cup B} = \overline{A} \cap \overline{B}$
	б) $\overline{A \cap B} = \overline{A} \cup \overline{B}$

Таблиця 1.15 (*продовження*)

Назва закону	Формулювання закону
7. Закони поглинання	*а*) $A \cap (A \cup B) = A$
	б) $A \cup (A \cap B) = A$
8. Закони тотожності	*а*) $A \cup \varnothing = A$
	б) $A \cap U = A$
9. Закони домінування	*а*) $A \cup U = U$
	б) $A \cap \varnothing = \varnothing$

Говорять, що дві множини A та B *не перетинаються*, якщо вони не мають спільних елементів, тобто якщо $A \cap B = \varnothing$.

Для будь-яких скінченних множин A та B правдива рівність $|A \cup B| = |A| + |B| - |A \cap B|$ — частинний випадок принципу включення-виключення, докладно розгляненого в розділі 2.

Систему $S = \{A_i\}$ ($i \in I$, де I — множина індексів) підмножин множини A називають *розбиттям* множини A за таких умов.

1. $A_i \neq \varnothing$ для $i \in I$.
2. $A_i \cap A_j = \varnothing$, $i \neq j$.
3. $\bigcup\limits_{i \in I} A_i = A$.

Інакше кажучи, система S непорожніх підмножин множини A являє собою розбиття цієї множини, якщо будь-який елемент $a \in A$ належить точно одній множині A_i із системи S.

Доводити рівності з множинами можна різними способами. Нижче наведено приклади, що ілюструють способи доведення.

Спосіб 1. Цей спосіб ґрунтується на такій теоремі.

ТЕОРЕМА 1.5. Множини A та B рівні тоді й лише тоді, коли $A \subset B$ та $B \subset A$.

Приклад 1.63. Доведемо рівність множин, яка являє собою формулювання закону де Моргана $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Припустимо, що $x \in \overline{A \cap B}$. Тоді $x \notin A \cap B$, звідки випливає, що $x \notin A$ чи $x \notin B$. Отже, $x \in \overline{A}$ чи $x \in \overline{B}$, а це означає, що $x \in \overline{A} \cup \overline{B}$. Ми довели, що $\overline{A \cap B} \subset \overline{A} \cup \overline{B}$. Навпаки, нехай $x \in \overline{A} \cup \overline{B}$. Тоді $x \in \overline{A}$ чи $x \in \overline{B}$, звідки випливає, що $x \notin A$ чи $x \notin B$. Це означає, що $x \notin A \cap B$, тобто $x \in \overline{A \cap B}$. Отже, $\overline{A} \cup \overline{B} \subset \overline{A \cap B}$.

Спосіб 2. Доведення рівності множин за допомогою законів логіки.

Приклад 1.64. Доведемо рівність $\overline{A \cap B} = \overline{A} \cup \overline{B}$. Послідовно перевіримо рівності

$$\overline{A \cap B} = \{x \mid x \notin A \cap B\} = \{x \mid \neg (x \in A \cap B)\} =$$

$$= \{x \mid \neg ((x \in A) \wedge (x \in B))\} = \{x \mid (x \notin A) \vee (x \notin B)\} =$$

$$= \{x \mid (x \in \overline{A}) \vee (x \in \overline{B})\} = \{x \mid x \in \overline{A} \cup \overline{B}\} = \overline{A} \cup \overline{B}.$$

Спосіб 3. Доведення рівності множин за допомогою *таблиць належності*, які містять усі можливі комбінації належності елементів множинам (1 — елемент належить множині, 0 — не належить).

Приклад 1.65. Доведемо цим способом рівність $\overline{A \cap B} = \overline{A} \cup \overline{B}$. Доведення подано в табл. 1.16.

Таблиця 1.16

A	B	$A \cap B$	$\overline{A \cap B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cup \overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Стовпчики, які в табл. 1.16 відповідають значенням $\overline{A \cap B}$ та $\overline{A} \cup \overline{B}$, однакові, отже $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Спосіб 4. Доведення рівності множин за допомогою основних законів теорії множин (див. табл. 1.15).

Приклад 1.66. Довести тотожність $\overline{A \cup (B \cap C)} = (\overline{C} \cup \overline{B}) \cap \overline{A}$. Використовуючи закони де Моргана та комутативні (див. табл. 1.15), можна записати таку послідовність рівних множин:

$$\overline{A \cup (B \cap C)} = \overline{A} \cap \overline{(B \cap C)}$$

за законом де Моргана 6*а*

$$= \overline{A} \cap (\overline{B} \cup \overline{C})$$

за законом де Моргана 6*б*

$$= (\overline{B} \cup \overline{C}) \cap \overline{A}$$

за законом комутативності 1*б*.

$$= (\overline{C} \cup \overline{B}) \cap \overline{A}$$

за законом комутативності 1*а*.

1.14. Комп'ютерне подання множин

У комп'ютері можна подавати множини різними способами. Один зі способів — зберігати невпорядковані елементи множини. Проте в такому разі операції з множинами займатимуть багато часу через те, що потрібно щоразу переглядати елементи. Тому розглянемо інші способи.

Одним із найпоширеніших і найпростіших способів — подання множин за допомогою бітових рядків. Упорядкуємо довільним способом елементи універсальної множини. Нехай універсальна множина U містить n елементів, тоді $U = \{a_1, a_2, a_3, ..., a_{n-1}, a_n\}$.

Множину $A \subset U$ подають у комп'ютері рядком із 0 і 1 довжиною n так: якщо $a_i \in A$, то i-й біт дорівнює 1, а ні, то 0.

Приклад 1.67. Нехай $U = \{a, b, c, d, e, f, m, n, p, q\}$, $A = \{b, m, n, q\}$, $B = \{a, b, f, m, q\}$. Тоді множину A подають рядком 0100001101, а множину B — рядком 1100011001.

Тепер на комп'ютері легко виконати операції над множинами A та B. Неважко переконатись, що об'єднанню множин відповідає порозрядне OR над бітовими рядками, які подають множини A та B, а перетину множин — порозрядне AND над відповідними бітовими рядками.

Приклад 1.68. Використаємо бітові рядки, які подають множини A та B з прикладу 1.67. Бітовий рядок, який відповідає об'єднанню цих множин $A \cup B = \{a, b, f, m, n, q\}$, знаходимо як результат виконання операції порозрядного OR:
0100001101
1100011001
1100011101.
Бітовий рядок, який відповідає перетину множин $A \cap B = \{b, m, q\}$, знаходимо як результат виконання операції порозрядного AND:
0100001101
1100011001
0100001001.

Якщо універсальна множина U має велику потужність, а її підмножини не дуже потужні, то подання за допомогою бітових рядків неефективне щодо витрат пам'яті. У такому разі доцільно використовувати інші структури даних — зазвичай зв'язані списки та хеш-таблиці [2]. У певних задачах потрібні спеціальні методи подання множин, які ґрунтуються на використанні дерев [2, 23].

Контрольні запитання та завдання

1. Задано висловлювання p: „Завтра буде холодно" та q: „Падатиме сніг". Записати наведені нижче висловлювання за допомогою p, q та логічних операцій:
   а) завтра буде холодно й падатиме сніг;
   б) завтра буде холодно, але сніг не падатиме;
   в) завтра не буде холодно й не падатиме сніг;
   г) завтра падатиме сніг або буде холодно (або одне й друге);
   д) якщо завтра буде холодно, то падатиме сніг;
   е) завтра буде холодно чи падатиме сніг, але не падатиме сніг, якщо буде холодно;
   ж) для того, щоб завтра було холодно, необхідно й достатньо, щоб падав сніг.
2. Визначити кількість різних двомісних логічних операцій.
3. Визначити кількість різних двомісних логічних операцій, які набувають значення T щонайбільше в трьох інтерпретаціях.
4. Побудувати таблиці істинності для кожного з висловлювань:
   а) $p \to q$;
   б) $\bar{p} \sim q$;

в) $(p \to q) \vee (\bar{p} \to q)$;

г) $(p \to q) \wedge (\bar{p} \to q)$;

д) $(p \sim q) \vee (\bar{p} \sim q)$;

е) $(\bar{p} \sim \bar{q}) \sim (p \sim q)$.

5. Довести, що формули $\bar{q} \to \bar{p}$, $(p \wedge \bar{q}) \to \bar{p}$, $(p \wedge \bar{q}) \to q$, $(p \wedge \bar{q}) \to$ F, де F — значення „хибність", мають ту саму таблицю істинності, що й формула $p \to q$.

6. Довести, що формули $p \sim q$ та $(p \to q) \wedge (q \to p)$ мають однакові таблиці істинності.

7. Знайти значення змінної x після обчислення кожного з операторів, наведених нижче, якщо перед їх виконанням $x = 1$:

а) `if (1 + 2 = 3) then` x`:=` x `+ 1;`

б) `if (1 + 1 = 3) OR (2 + 2 = 3) then` x `:=` x `+ 1;`

в) `if (2 + 3 = 5) AND (3 + 4 = 7) then` x `:=` x `+ 1;`

г) `if (1 + 1 = 2) XOR (1 + 2 = 3) then` x `:=` x `+ 1;`

д) `if` x `< 2 then` x `:=` x `+ 1.`

8. Застосувавши таблиці істинності, довести закони дистрибутивності.

9. Застосувавши таблиці істинності, довести закони де Моргана.

10. Застосувавши таблиці істинності, довести правило контрапозиції $p \to q = \bar{q} \to \bar{p}$.

11. Довести закони поглинання $p \vee (p \wedge q) = p$ та $p \wedge (p \vee q) = p$.

12. Довести закони виключеного третього $p \vee \bar{p} =$ T та суперечності $p \wedge \bar{p} =$ F.

13. На основі властивості імплікації (без використання таблиць істинності та еквівалентних перетворень) довести, що формули наведені нижче тавтології:

а) $(p \wedge q) \to p$; б) $p \to (p \vee q)$; в) $\bar{p} \to (p \to q)$;

г) $(p \wedge q) \to (p \to q)$; д) $(\neg(p \to q)) \to p$; е) $(\neg(p \to q)) \to \bar{q}$.

14. Який із законів дистрибутивності $p * (q \blacklozenge r) = (p * q) \blacklozenge (p * r)$ та $(p \blacklozenge q) * r = (p * r) \blacklozenge (q * r)$ виконується, якщо логічні операції, позначені символами „$\blacklozenge$" та „$*$", задано в наступній таблиці.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$*$	$\wedge$	$\wedge$	$\wedge$	$\vee$	$\vee$	$\vee$	$\oplus$	$\oplus$	$\oplus$
$\blacklozenge$	$\oplus$	$\to$	$\sim$	$\oplus$	$\to$	$\sim$	$\wedge$	$\vee$	$\to$
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
$*$	$\oplus$	$\to$	$\to$	$\to$	$\to$	$\sim$	$\sim$	$\sim$	$\sim$
$\blacklozenge$	$\sim$	$\wedge$	$\vee$	$\oplus$	$\sim$	$\wedge$	$\vee$	$\oplus$	$\to$

15. *Двоїстим* до складного висловлювання, яке містить лише логічні операції $\vee$, $\wedge$, $\neg$, називають висловлювання, одержане заміною $\vee$ на $\wedge$, $\wedge$ на $\vee$, T на F та F на T. Знайти висловлювання, двоїсті до висловлювань:

а) $p \wedge q \wedge r$; б) $(p \wedge q \wedge r) \vee s$; в) $(p \vee$ F$) \wedge (q \vee$ T$)$.

16. Нехай складне висловлювання задано такою таблицею істинності.

p	q	r	f_1	f_2	f_3	f_4	f_5
F	F	F	T	F	F	T	T
F	F	T	F	T	F	T	F
F	T	F	F	T	T	T	T
F	T	T	T	F	T	F	T
T	F	F	F	T	F	T	F
T	F	T	T	F	T	F	F
T	T	F	T	F	F	F	F
T	T	T	F	T	F	T	F

Виберемо інтерпретації, у яких значення істинності висловлювання дорівнює Т. Для кожної такої інтерпретації побудуємо кон'юнкцію атомів або їх заперечень: якщо значення атома хибне, то воно входить до кон'юнкції із запереченням, а ні, то без заперечення. Запишемо диз'юнкцію отриманих кон'юнкцій. Побудовану формулу називають *досконалою диз'юнктивною нормальною формою* (ДДНФ). Наприклад, формула $(\overline{p} \wedge \overline{q} \wedge \overline{r}) \vee (\overline{p} \wedge q \wedge \overline{r}) \vee (\overline{p} \wedge q \wedge r)$ — ДДНФ висловлювання f_5. Побудувати ДДНФ висловлювань $f_1 - f_4$. Довести, що ДДНФ задає складне висловлювання.

17. Нехай складне висловлювання задано таблицею істинності. Виберемо інтерпретації, у яких висловлювання хибне. Для кожної такої інтерпретації побудуємо диз'юнкцію атомів або їх заперечень: якщо значення атома хибне, то він уходить до диз'юнкції без заперечення, а ні, то із запереченням. Запишемо кон'юнкцію отриманих диз'юнкцій. Побудовану формулу називають *досконалою кон'юнктивною нормальною формою* (ДКНФ). Наприклад, формула $(p \vee \overline{q} \vee \overline{r}) \wedge (\overline{p} \vee q \vee \overline{r}) \wedge (\overline{p} \vee \overline{q} \vee r)$ — ДКНФ висловлювання f_4. Побудувати ДКНФ висловлювань f_1, f_2, f_3, f_5 із задачі 16. Довести, що ДКНФ задає складне висловлювання.

18. Побудувати складне висловлювання, яке складається з простих висловлювань p, q, r та набуває значення Т тоді й лише тоді, коли:

а) p та q істинні, r хибне;

б) точно два з трьох висловлювань p, q, r істинні.

19. Побудувавши таблиці істинності, виявити, чи є висловлювання, наведені нижче, тавтологіями:

а) $((p \to q) \wedge (q \to r)) \sim (p \to r)$; б) $((p \to q) \wedge (\overline{p} \to q)) \sim (\overline{p} \to \overline{q})$;

в) $((p \to q) \wedge (\overline{p} \to \overline{q})) \sim \overline{p}$; г) $((p \to q) \wedge (\overline{p} \to q)) \sim q$;

д) $(((\overline{p} \to \overline{q}) \to r) \wedge ((\neg(p \to q)) \to r)) \sim (p \to q)$.

20. За допомогою еквівалентних перетворень перевірити, чи є наведені нижче висловлювання тавтологіями:

а) $((p \to q) \wedge (\overline{p} \to q)) \sim q$; б) $((p \to q) \wedge (p \to \overline{q})) \sim \overline{p}$;

в) $(p \to q) \vee (\overline{p} \to q)$; г) $((p \to q) \wedge (\overline{p} \to q)) \vee \overline{q}$;

д) $(p \wedge q \wedge (p \sim q)) \sim (p \wedge q)$.

21. За означенням імплікації (без побудови таблиць істинності та виконання еквівалентних перетворень), перевірити, чи є наведені нижче висловлювання тавтологіями:

а) $((p \vee q) \wedge (p \to q) \wedge (q \to r)) \to r$;

б) $((p \vee q \vee s) \vee (p \to r) \wedge (q \to r) \wedge (s \to r)) \to r$;

в) $(((p \wedge \overline{q}) \to \overline{p}) \wedge ((q \wedge \overline{r}) \to r)) \to (p \to r)$;

г) $((p \vee q) \wedge (p \to r) \wedge (q \to s)) \to (r \vee s)$.

22. Довести, що наведені нижче формули еквівалентні:

а) $p \wedge (q \oplus r)$ і $(p \wedge q) \oplus (p \wedge r)$;

б) $p \to (q \sim r)$ і $(p \wedge q) \sim (p \wedge r)$;

в) $p \to (q \vee r)$ і $(p \to q) \vee (p \to r)$.

23. Перевірити, чи еквівалентні наведені нижче формули:

а) $p \oplus (q \wedge r)$ і $(p \oplus q) \wedge (p \oplus r)$;

б) $(p \to q) \to r$ і $p \to (q \to r)$;

в) $p \to (q \wedge r)$ і $(p \to q) \wedge (p \to r)$.

24. Подати висловлювання $p \sim q$ формулою, яка містить лише логічні операції $\neg$ та $\vee$.

25. Подати висловлювання $p \vee q$ формулою, яка містить лише логічну операцію $\to$.

26. Нехай $P(x)$: „$x = x^2$", а предметна область змінної x — множина цілих чисел. Знайти значення істинності висловлювань:

а) $P(0)$; б) $P(1)$; в) $P(-1)$;

г) $\exists x P(x)$; д) $\forall x P(x)$.

27. Нехай $Q(x, y)$: „$x + y = x - y$", а предметна область кожної змінної — множина цілих чисел. Визначити значення істинності висловлювань:

а) $Q(1, 1)$; б) $Q(2, 0)$; в) $\forall y Q(1, y)$;

г) $\exists x Q(x, 2)$; д) $\exists x \exists y Q(x, y)$; е) $\forall x \exists y Q(x, y)$;

ж) $\forall x \forall y Q(x, y)$; и) $\exists x \forall y Q(x, y)$; к) $\forall y \exists x Q(x, y)$; л) $\exists y \forall x Q(x, y)$.

28. Предметна область кожної змінної предиката $P(x, y)$ — множина {1, 2, 3}. Записати наведені нижче висловлювання за допомогою логічних операцій кон'юнкції та диз'юнкції:

а) $\exists x P(x, 3)$; б) $\forall y P(1, y)$;

в) $\forall x \forall y P(x, y)$; г) $\exists x \exists y P(x, y)$;

д) $\exists x \forall y P(x, y)$; е) $\forall y \exists x P(x, y)$.

29. Побудувати випереджену нормальну форму формул:

а) $\forall x(\neg(\exists y(P(x) \to Q(y))))$;

б) $\exists x(\neg(\forall y P(x, y, z) \to \exists u Q(x, u))) \wedge \forall t(\neg(\forall v(A(t) \vee B(v))))$.

в) $\forall x(P(x) \to \exists y Q(x, y))$;

г) $\exists x(\neg((\exists y P(x, y)) \to (\exists z Q(z) \to R(x))))$;

д) $\exists x(\neg((\exists y P(x,y)) \to (\exists z Q(z) \to R(x))))$;

е) $\forall x \forall y (\exists z P(x,y,z) \wedge (\exists u Q(x,u) \to \exists v Q(y,v)))$.

30. Записати заперечення наведених нижче висловлювань формулами логіки першого ступеня та реченнями української мови:

а) кожний студент групи любить математику;

б) у групі є студент, який ніколи не бачив комп'ютера;

в) у групі є студент, який прослухав усі запропоновані математичні курси;

г) у групі є студент, який відвідав принаймні одну аудиторію кожного з навчальних корпусів університету.

31. Довести, що формули, наведені нижче, мають однакові значення істинності.

а) $\overline{\exists x \forall y P(x,y)}$ і $\forall x \exists y \overline{P}(x,y)$;

б) $\forall x(P(x) \wedge Q(x))$ і $\forall x P(x) \wedge \forall x Q(x)$;

в) $\exists x(P(x) \vee Q(x))$ і $\exists x P(x) \vee \exists x Q(x)$.

32. Позначення $\exists! x P(x)$ відповідає реченню „У предметній області існує таке єдине x, що $P(x)$ істинне". Нехай множина цілих чисел — предметна область змінної x. Знайти значення істинності формул:

а) $\exists! x(x > 1)$; б) $\exists! x(x^2 = 1)$; в) $\exists! x(x + 3 = 2x)$; г) $\exists! x(x = x + 1)$;

д) $\exists! x P(x) \to \exists x P(x)$; е) $\forall x P(x) \to \exists! x P(x)$; ж) $\exists! x \overline{P}(x) \to \forall x \overline{P}(x)$.

33. Задано предметну область $M = \{1, 2, 3\}$ змінної x. Записати висловлювання $\exists! x P(x)$ за допомогою заперечення, кон'юнкції та диз'юнкції.

34. Які правила виведення використано у наступних твердженнях?

а) Якщо падатиме дощ, то басейн буде зачинено. Падає дощ. Отже, басейн зачинено.

б) Якщо падатиме сніг, то університет буде зачинено. Університет не зачинено. Отже, сніг не падає.

в) Якщо я піду плавати, то довго буду на сонці. Якщо я довго буду на сонці, то засмагну. Отже, якщо я піду плавати, то засмагну.

г) Кенгуру живуть в Австралії, мають сильні нижні кінцівки, роблять довгі стрибки, і вони сумчасті. Отже, кенгуру — сумчасті.

д) На вулиці спека більше 40 градусів або небезпечне забруднення повітря. Сьогодні менше 40 градусів; отже, забруднення повітря небезпечне.

35. Дано гіпотези: „Іван тяжко працює", „Якщо Іван тяжко працює, то він пасивний хлопець" і „Якщо Іван — пасивний хлопець, то він не знайде кращої роботи". Використати правила виведення для обґрунтування такого висновку з цих гіпотез: „Іван не знайде кращої роботи".

36. Дано такі гіпотези: „Логіка складна чи небагато студентів люблять логіку" та „Якщо математика складна, то логіка не складна". Визначити, чи можна з цих гіпотез отримати висновок.

а) Математика не легка, якщо багато студентів люблять логіку.

б) Небагато студентів люблять логіку, якщо математика не складна.

в) Математика не легка чи логіка складна.

г) Логіка не складна чи математика не легка.

д) Якщо небагато студентів люблять логіку, то чи математика не легка, чи логіка не складна.

37. Довести, що в наведених нижче прикладах висновки можна вивести з наведених гіпотез.

а) Гіпотези: „Усі леви — жорстокі істоти", „Деякі леви не п'ють кави".
Висновок: „Деякі жорстокі істоти не п'ють кави".

б) Гіпотези: „Усі колібрі мають яскраве пір'я", „Жодний великий птах не їсть меду та не має яскравого пір'я".
Висновок: „Колібрі — маленькі птахи".

в) Гіпотези: „Кожний атлет сильний", „Кожний, хто сильний і розумний, досягне успіху", „Петро — атлет", „Петро — розумний".
Висновок: „Петро досягне успіху".

38. Для кожного з логічних виведень, наведених нижче, визначити коректність висновку та зробити потрібні пояснення.

а) Усі студенти цієї групи розуміють логіку. Дмитро — студент цієї групи. Отже, Дмитро розуміє логіку.

б) Кожний студент, який вивчає комп'ютерні науки та є студентом старшого курсу, прослухав курс дискретної математики. Наталка прослухала курс дискретної математики. Отже, Наталка — студентка старшого курсу та вивчає комп'ютерні науки.

в) Кожний папуга схожий на фрукт. Моя пташка не папуга. Отже, моя пташка не схожа на фрукт.

г) Роман любить дивитися бойовики. Роман любить фільм „Третій зайвий". Отже, фільм „Третій зайвий" — бойовик.

д) Кожний студент університету має жити в гуртожитку. Михайло не живе в гуртожитку. Отже, Михайло не студент університету.

е) Якщо геометрична фігура — квадрат, то її діагоналі взаємно перпендикулярні та в точці перетину діляться навпіл. Ця фігура не квадрат. Отже, її діагоналі не перпендикулярні та не діляться навпіл.

ж) Якщо число має дільник 6, то воно має дільниками числа 2 та 3. Якщо число має дільниками числа 2 та 3, то воно має дільник 6. Отже, число має дільник 6 тоді й лише тоді, коли воно має дільниками числа 2 та 3.

и) Якщо Петро поїде до Харкова, то Іван поїде до Києва. Петро поїде чи до Харкова, чи до Львова. Якщо Петро поїде до Львова, то Ольга залишиться у Полтаві. Ольга не залишилась у Полтаві. Отже, Іван поїхав до Києва.

39. Довести логічні теореми:

а) $\overline{p} \vee q,\ \overline{p} \vee r,\ \overline{q} \vee \overline{r} \vdash \overline{p}$;

б) $\overline{h},\ \overline{h} \to (p \vee q),\ p \to c,\ q \to c \vdash c$.

40. Побудувати резольвенту елементарних диз'юнкцій d_1 та d_2:

а) $d_1 = p \vee r$, $d_2 = \overline{p} \vee q$; б) $d_1 = \overline{p} \vee q \vee r$, $d_2 = \overline{q} \vee s$.

41. Перевірити наведені нижче виведення, використовуючи метод резолюцій:

а) $p \vee q, p \to r, q \to r \vdash r$; б) $p \vee q \vee s, p \to r, q \to r, s \to r \vdash r$;

в) $(p \wedge \overline{q}) \to \overline{p}, (q \wedge \overline{r}) \to r \vdash p \to r$; г) $p \vee q, p \to r, q \to s \vdash r \vee s$;

д) $p \to r, \overline{p} \to s, r \to q, s \to \overline{t}, t \vdash q$; е) $p \to q, \overline{p} \to r, q \to s \vdash r \vee s$.

42. Зобразити на координатній площині декартовий добуток множин $A \times B$ та $B \times A$. У задачі *б* записати також усі елементи декартових добутків:

а) $A = \{x \mid x \in R, 3 \leqslant x \leqslant 5\}$; $B = \{x \mid x \in R, 3 \leqslant x \leqslant 6\}$;

б) $A = \{x \mid x \in R, 3 \leqslant x \leqslant 5\}$; $B = \{x \mid x \in R, 3 \leqslant x \leqslant 6\}$.

43. Задано множини $A = \{a, b, c\}$, $B = \{x, y\}$, $C = \{0, 1\}$. Побудувати декартові добутки *а–г*:

а) $A \times B \times C$; б) $C \times B \times A$;

в) $C \times A \times B$; г) $B \times B \times B$.

44. Задано множини $A = \{a, b, c, d, e\}$ та $B = \{a, b, c, d, e, f, g, h\}$. Побудувати множини:

а) $A \cap B$; б) $A \cup B$;

в) $A \setminus B$; г) $B \setminus A$.

45. Знайти множини A та B, якщо $A \setminus B = \{1, 5, 7, 8\}$, $B \setminus A = \{2, 10\}$, і $A \cap B = \{3, 6, 9\}$.

46. Довести рівність множин $A \setminus B = A \cap \overline{B}$.

47. Довести рівності множин:

а) $A \cup (B \cup C) = (A \cup B) \cup C$; б) $A \cap (B \cap C) = (A \cap B) \cap C$;

в) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$; г) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

48. Задано множини A, B та C. Довести рівність $(A \setminus B) \setminus C = (A \setminus C) \setminus (B \setminus C)$.

49. Задано множини $A = \{0, 2, 4, 6, 8, 10\}$, $B = \{0, 1, 2, 3, 4, 5, 6\}$ та $C = \{4, 5, 6, 7, 8, 9, 10\}$. Побудувати множини:

а) $A \cap B \cap C$; б) $A \cup B \cup C$;

в) $(A \cup B) \cap C$; г) $(A \cap B) \cup C$.

50. Симетричною різницею множин A та B називається множина, задана рівністю $A \oplus B = (A \cup B) \setminus (A \cap B)$:

а) зобразити діаграму Венна множини $A \oplus B$;

б) задано множини $A = \{1, 3, 5\}$ та $B = \{1, 2, 3\}$, побудувати множину $A \oplus B$;

в) нехай A та B — довільні множини. Довести рівність $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

51. Чи можна твердити, що $A = B$, якщо для множин A, B та C виконуються рівності:

а) $A \cup C = B \cup C$; б) $A \cap C = B \cap C$; в) $A \oplus C = B \oplus C$.

52. Нехай A, B та C — довільні множини. Довести, що
$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$.
53. Задано універсальну множину $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$:
а) подати бітовими рядками множини $\{3, 4, 5\}$, $\{1, 3, 6, 10\}$, $\{2, 3, 4, 7, 8, 9\}$;
б) відновити множини за бітовими рядками 0101111100, 1000000001, 1111111111.
54. Показати, як можна використати операції над бітовими рядками для знаходження значень виразів:
а) $(A \cup B) \cap C$; б) $(A \cap B) \cup C$;
в) $(A \cap D) \cup (B \cap C)$; г) $A \cup B \cup C \cup D$.
Тут універсальна множина U – латинський алфавіт, який складається з 26 букв, а множини A, B, C та D такі: $A = \{a, b, c, d, e\}$, $B = \{b, c, d, g, p, t, v\}$, $C = \{c, e, i, o, u, x, y, z\}$, $D = \{d, e, h, i, n, o, t, u, x, y\}$.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами

1. Задано значення істинності висловлювань p та q. Знайти значення істинності кон'юнкції, диз'юнкції, альтернативного „або", імплікації й еквівалентності цих висловлювань.
2. Складне висловлювання задано таблицею істинності. Побудувати ДКНФ (див. задачу 17).
3. Задано два бітові рядки довжиною n. Знайти результати виконання порозрядних операцій OR, AND і XOR цих рядків.
4. Задано множину елементарних диз'юнкцій логіки висловлювань. Методом резолюцій визначити, чи невиконанна ця множина.
5. Дано множини A та B, які являють собою підмножини n-елементного універсуму. За допомогою бітових рядків визначити $\overline{A}$, $A \cap B$, $A \cup B$, $A \setminus B$, $A \oplus B$.
6. Дано скінченні множини A, B, C, D. Визначити всі пари цих множин, у яких перша множина — підмножина другої.
7. Дано множини E й F, кожна з яких — результат теоретико-множинних операцій над множинами A, B, C, D. За допомогою таблиці належності довести чи спростувати рівність $E = F$.
8. Дано скінченні множини A, B, C. Побудувати множини $A \times B \times C$, $C \times B \times A$, $C \times A \times B$, $C \times C \times B$, $A \times B \times A$, $C \times B \times C$.

Розділ 2
Комбінаторний аналіз

- Основні правила комбінаторного аналізу. Розміщення та сполучення
- Перестановки
- Біном Ньютона
- Поліноміальна теорема
- Задача про цілочислові розв'язки
- Числа Стірлінга другого роду та числа Белла
- Генерування перестановок, сполучень, розбиттів множини
- Рекурентні рівняння
- Принцип коробок Діріхле
- Принцип включення-виключення
- Твірні функції

У комбінаторному аналізі (комбінаториці) вивчають об'єкти зі скінченної множини $A = \{a_1, a_2, ..., a_n\}$ та їх властивості, а також визначають кількість об'єктів із певними властивостями. Розглядають також твердження (принципи), використовувані в різних задачах [19, 45]. На них ґрунтуються важливі методи математичного доведення, широко застосовувані в теорії скінченних автоматів [6] та інших розділах.

2.1. Основні правила комбінаторного аналізу. Розміщення та сполучення

Почнемо з формулювання двох основних правил комбінаторики: *правила суми* та *правила добутку*.

Правило суми. Якщо об'єкт x можна вибрати n_1 способами, а інший об'єкт y — n_2 способами, то можна вибрати або x, або y $n_1 + n_2$ способами.

Приклад 2.1. Студент має вибрати тему курсової роботи зі списку, розміщеного на трьох аркушах. Аркуші містять відповідно 20, 15 і 17 тем. З якої кількості можливих тем студент робить свій вибір?

За правилом суми кількість тем для вибору становить $20 + 15 + 17 = 52$.

Правило добутку. Якщо об'єкт x можна вибрати n_1 способами та після кожного такого вибору об'єкт y можна вибрати n_2 способами, то пару об'єктів (x, y) у зазначеному порядку можна вибрати $n_1 n_2$ способами. Це правило можна пояснити інакше. Нехай якусь процедуру можна виконати розв'язавши два завдання. Якщо є n_1 способів розв'язати перше завдання та n_2 способів розв'язати після цього друге завдання, то всю процедуру можна виконати $n_1 n_2$ способами.

Приклад 2.2. В одній із версій мови БЕЙСІК ім'я змінної — це рядок з одного чи двох символів, якими можуть бути 26 букв латинського алфавіту та 10 цифр. Першим символом має бути буква. Крім того, не можна використовувати п'ять двосимвольних рядків, які зарезервовані для спеціального використання. Знайдемо, скільки різних імен змінних є в цій версії мови БЕЙСІК.

Нехай V — величина, яку потрібно обчислити, V_1 — кількість односимвольних імен, V_2 — двосимвольних. За правилом суми всього імен $V = V_1 + V_2$. Очевидно, що $V_1 = 26$; за правилом добутку $V_2 = 26 \cdot 36 - 5 = 931$. Отже, $V = 26 + 931 = 957$.

Розглянемо основні комбінаторні об'єкти — розміщення та сполучення [19], — попередньо означивши важливе поняття вибірки [17].

Нехай задано скінченну непорожню множину $A = \{a_1, a_2, ..., a_n\}$ і виконано r таких кроків.

Крок 1. Із множини A вибирають якийсь елемент a_{i_1}.

Крок 2. Із множини A чи з $A \setminus \{a_{i_1}\}$ вибирають якийсь елемент a_{i_2}.

... ...

Крок r. Якщо $a_{i_1}, a_{i_2}, ..., a_{i_{r-1}}$ — елементи, які вибрані на перших $r-1$ кроках $(r \geqslant 3)$, то на цьому кроці вибирають якийсь елемент a_{i_r} із множини A чи $A \setminus \bigcup_{k=1}^{r-1} \{a_{i_k}\}$. Тоді елементи $a_{i_1}, a_{i_2}, ..., a_{i_r}$ утворюють *вибірку обсягом r*, або *r-вибірку*, із множини A.

Вибірку називають *упорядкованою*, якщо задано порядок її елементів, а ні — то *невпорядкованою*. Зрозуміло, що впорядкована r-вибірка — це кортеж (вектор) з r компонентами, і тому її позначають $(b_1, b_2, ..., b_r)$, $b_i \in A$, $i = 1, ..., r$. Невпорядковану r-вибірку позначатимемо як $[b_1, b_2, ..., b_r]$, $b_i \in A$, $i = 1, ..., r$.

Упорядковані r-вибірки з n-елементної множини називають *розміщеннями з n елементів по r*, а невпорядковані — *сполученнями з n елементів по r*. Використовують також поняття r-розміщення й r-сполучення. Розглянемо два способи вибору елементів.

Згідно з першим способом вибору на кожному кроці вибирають елемент з усієї множини A. Отже, один і той самий елемент із множини A може зустрітись у вибірці декілька разів. Такі вибірки називаються *вибірками з повтореннями*.

У разі застосування другого способу вибраний елемент вилучають із множини A. Це означає, що на кожному j-му кроці $(1 < j \leqslant k)$ вибирають елемент із множини $A \setminus \bigcup_{k=1}^{j-1} \{a_{i_k}\}$, і вибірка не містить однакових елементів. Такі вибірки називають *вибірками без повторень*.

Приклад 2.3. Задано множину $A = \{a, b, c\}$, тобто $n = 3$.
Наведемо розміщення без повторень із трьох елементів по два, тобто $r = 2$:

$$(a, b), (a, c), (b, c), (b, a), (c, a), (c, b);$$

розміщення з повтореннями з трьох елементів по два:

$$(a, b), (a, c), (b, c), (b, a)\ (c, a), (c, b), (a, a), (b, b), (c, c);$$

сполучення без повторень із трьох елементів по два:

$$[a, b], [a, c], [b, c];$$

сполучення з повтореннями із трьох елементів по два:

$$[a, b], [a, c], [b, c], [a, a], [b, b], [c, c].$$

Зазначимо, що сполучення без повторень з n елементів по r — це просто r-елементні підмножини множини з n елементів; отже, їх можна записати так: $\{a, b\}, \{a, c\}, \{b, c\}$.
Сполучення з повтореннями — це, узагалі кажучи, не множина у звичайному розумінні: її елементи можуть повторюватись, тобто зустрічатися більше одного разу.

2.2. Обчислення кількості розміщень і сполучень

Кількість усіх розміщень без повторень з n елементів по r позначають як A_n^r або $A(n, r)$, де r і n — невід'ємні цілі числа, причому $r \leqslant n$. Кількість різних розміщень із повтореннями з n елементів по r позначають як $\tilde{A}_n^r$ або $\tilde{A}(n,r)$. Тут r і n — будь-які невід'ємні цілі числа. Кількість усіх сполучень без повторень з n елементів по r позначають як C_n^r, $C(n, r)$ або $\binom{n}{r}$, де r і n — невід'ємні цілі числа, причому $r \leqslant n$. Кількість усіх сполучень із повтореннями з n елементів по r позначимо як H_n^r або $H(n, r)$, де r і n — будь-які невід'ємні цілі числа. Числа C_n^r називають *біноміальними коефіцієнтами.*
Доведемо, що

$$A_n^r = n(n-1)\ldots(n-r+1) = \frac{n!}{(n-r)!}, \tag{2.1}$$

$$\tilde{A}_n^r = n^r, \tag{2.2}$$

$$C_n^r = \frac{A_n^r}{r!} = \frac{n!}{r!\,(n-r)!}, \tag{2.3}$$

$$H_n^r = C_{n+r-1}^r. \tag{2.4}$$

Доведемо рівність (2.1). Розглянемо якесь розміщення $(b_1, b_2, \ldots, b_r)$ без повторень з n елементів по r. Ми можемо взяти як b_1 будь-який з n елементів, як b_2 — будь-який з $(n-1)$ елементів, що залишились, і продовжити цей процес. Отже,

для b_r залишається $(n+r-1)$ можливостей вибору. Використавши правило добутку, переконуємось у тому, що рівність (2.1) правильна.

Рівність (2.2) також справджується, бо в розміщенні з повтореннями $(b_1, b_2, ..., b_r)$ для кожного елемента b_i, $i = 1, 2, ..., r$, є n незалежних можливостей вибору.

Доведемо рівність (2.3). Розглянемо якесь сполучення $[b_1, b_2, ..., b_r]$ без повторень з n елементів по r. Виявимо, скільки можна отримати різних розміщень без повторень з r елементів по r із цього сполучення як з r-елементної множини. За формулою (2.1) дістанемо $A_r^r = r(r-1)...2 \cdot 1 = r!$. Очевидно, що в разі $[b_1, ..., b_r] \neq [c_1, ..., c_r]$ із двох сполучень без повторень $[b_1, ..., b_r]$ і $[c_1, ..., c_r]$ не можна одержати однакових розміщень без повторень з r елементів по r. Отже, $A_n^r = r! \cdot C_n^r$, і рівність (2.3) доведено.

Нарешті, доведемо рівність (2.4). Замість n-елементної множини $A = \{a_1, a_2, ..., a_n\}$ розглянемо множину $A' = \{1, 2, ..., n\}$ також з n елементів. Кожну невпорядковану r-вибірку з множини A' можна записати у вигляді $[m_1, m_2, ..., m_r]$, де $m_1 \leqslant m_2 \leqslant \leqslant ... \leqslant m_r$, оскільки порядок елементів не суттєвий. Тоді $[m_1 + 0, m_2 + 1, ..., m_r + r - 1]$ — сполучення без повторень з $n + r - 1$ елементів по r.

Розглянемо відображення Γ множини всіх сполучень із повтореннями з n елементів по r на множину всіх сполучень без повторень з $n + r - 1$ елементів по r: $\Gamma([m_1, m_2, ..., m_r]) = [m_1 + 0, m_2 + 1, ..., m_r + r - 1]$. Два сполучення з повтореннями *рівні*, якщо вони складаються з однакових елементів і кратності цих елементів збігаються. Якщо $[m_1, m_2, ..., m_r] \neq [m_1', m_2', ..., m_r']$, то й $[m_1 + 0, m_2 + 1, ..., m_r + r - 1] \neq \neq [m_1' + 0, m_2' + 1, ..., m_r' + r - 1]$. Більше того, якщо $[n_1, n_2, ..., n_r]$ — сполучення без повторень з $n + r - 1$ елементів по r, де $n_1 < n_2 < ... < n_r$, то $[n_1, n_2 - 1, ..., n_r - r + 1]$ — елемент множини сполучень із повтореннями з n елементів по r. Отже, Γ — бієктивне відображення множини всіх сполучень із повтореннями з n елементів по r на множину всіх сполучень без повторень з $n + r - 1$ елементів по r. Рівність (2.4) доведено.

2.3. Перестановки

Перестановка з n елементів — це особливий випадок розміщення без повторень з n елементів, коли в розміщення входять усі елементи. Перестановки з n елементів називають також *n-перестановками*. Окремі n-перестановки різняться лише порядком елементів. Кількість таких перестановок позначають як P_n. Формулу для P_n одержують із формули (2.1) для кількості розміщень без повторень: $P_n = A_n^n = n!$.

Розглянемо тепер задачу про перестановки n елементів за умови, що не всі елементи різні (перестановки з повтореннями). Точніше, нехай є n елементів k різних типів, а число n_j $(j = \overline{1, ..., k})$ — кількість елементів j-го типу. Очевидно, що $n_1 + n_2 + ... + n_k = n$. Перестановки з n елементів за такої умови називають *перестановками з повтореннями*. Кількість таких перестановок позначають як $P_n(n_1, n_2, ..., n_k)$. Щоб знайти явний вираз для $P_n(n_1, n_2, ..., n_k)$, візьмемо окрему

перестановку та замінимо в ній усі однакові елементи різними. Тоді кількість різних перестановок, котрі можна отримати з узятої однієї перестановки, дорівнює $n_1!n_2!\ldots n_k!$. Якщо зробити це для кожної перестановки, то одержимо $n!$ перестановок. Отже, $P_n(n_1, n_2, \ldots, n_k)n_1!n_2!\ldots n_k! = n!$, звідки

$$P_n(n_1, n_2, \ldots, n_k) = \frac{n!}{n_1!n_2!\ldots n_k!}.$$

Приклад 2.4. Знайдемо кількість слів (рядків), які можна утворити, переставляючи букви слова PRODUCT. Оскільки жодна буква тут не повторюється, то можна утворити $P_7 = 7! = 5040$ слів.

Приклад 2.5. Знайдемо, скільки слів можна утворити, переставляючи букви слова SUCCESS. У цьому слові є повторні входження букв, тому скористаємося формулою для перестановок із повтореннями:

$$P_7(3, 2, 1, 1) = \frac{7!}{3!\,2!\,1!\,1!} = 420 \text{ слів}.$$

Розглянемо задачу розкладання в ящики. Загальне формулювання цієї задачі таке. Дано n різних предметів і k ящиків. Потрібно покласти в перший ящик n_1 предметів, у другий $-n_2$ предметів, ..., k-й $-n_k$ предметів, де $n_1 + n_2 + \ldots + n_k = n$. Тут n_1, n_2, ..., n_k — фіксовані числа. Скількома способами можна зробити це?

Можна розкласти предмети так. Серед n предметів візьмемо довільну n_1-підмножину й покладемо її в перший ящик (це можна зробити $C_n^{n_1}$ способами). Серед $n - n_1$ предметів, що залишились, візьмемо n_2-підмножину й покладемо її в другий ящик (це можна зробити $C_{n-n_1}^{n_2}$ способами) і продовжимо цей процес. За правилом добутку загальна кількість розкладань дорівнює

$$C_n^{n_1} C_{n-n_1}^{n_2} \ldots C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} =$$

$$= \frac{n!}{n_1!(n-n_1)!} \frac{(n-n_1)!}{n_2!(n-n_1-n_2)!} \ldots \frac{(n-n_1-\ldots-n_{k-1})!}{n_k!(n-n_1-\ldots-n_k)!} =$$

$$= \frac{n!}{n_1!n_2!\ldots n_k!}.$$

Отже, розкладань у ящики стільки, скільки перестановок із повтореннями. Розглянемо зв'язок між цими двома задачами. Для цього занумеруємо всі n місць, які можуть займати предмети. Кожній перестановці відповідає розподіл номерів місць на k класів: в i-й клас потрапляють номери тих місць, на які покладено предмети i-го типу. Отже, знайдено відповідність між перестановками з повторенням та розкладанням номерів місць у ящики. Тому формули розв'язання обох задач збігаються.

2.4. Біном Ньютона

Нагадаємо, що біноміальними коефіцієнтами називають числа $C_n^r = n!/[r!(n-r)!]$ — кількість сполучень з n елементів по r. Розглянемо деякі властивості біноміальних коефіцієнтів.

1. Нехай n і r — невід'ємні цілі числа, $n \geqslant r$. Тоді $C_n^r = C_n^{n-r}$. Справді,

$$C_n^{n-r} = \frac{n!}{(n-r)![n-(n-r)]!} = \frac{n!}{(n-r)!\,r!} = \frac{n!}{r!(n-r)!} = C_n^r.$$

2. Рівність Паскаля: $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$.

   Позначимо як $S_{n,k}$ множину всіх сполучень з елементів $\{a_1, ..., a_{n-1}, a_n\}$ по k елементів; як $S_{n-1,k}$ — відповідно з елементів $\{a_1, ..., a_{n-1}\}$ по k; $S_{n-1,k-1}$ — з елементів $\{a_1, ..., a_{n-1}\}$ по $k-1$. Кожному сполученню з $S_{n,k}$, яке містить елемент a_n, відповідає сполучення з $S_{n-1,k-1}$. Якщо ж сполучення з $S_{n,k}$ не містить a_n, то йому відповідає сполучення з $S_{n-1,k}$. Отже, існує бієкція між множинами $S_{n,k}$ й $S_{n-1,k} \cup S_{n-1,k-1}$. Оскільки очевидно, що $S_{n-1,k} \cap S_{n-1,k-1} = \varnothing$, то $|S_{n,k}| = |S_{n-1,k}| + |S_{n-1,k-1}|$, тобто $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$.

   Отримане рекурентне співвідношення дає змогу побудувати таблицю для чисел C_n^k, яку називають *трикутником Паскаля* (табл. 2.1).

Таблиця 2.1

n \ k	0	1	2	3	4	5	6	7	8	9	...
0	1										...
1	1	1									...
2	1	2	1								...
3	1	3	3	1							...
4	1	4	6	4	1						...
5	1	5	10	10	5	1					...
6	1	6	15	20	15	6	1				...
7	1	7	21	35	35	21	7	1			...
8	1	8	28	56	70	56	28	8	1		...
9	1	9	36	84	126	126	84	36	9	1	...
...	...	...	...	...	...	...	...	...	...	...	...

3. Послідовність (p_n) дійсних чисел називають *унімодальною*, якщо існує такий натуральний номер m, що $p_0 < p_1 < ... < p_m$; $p_m \geqslant p_{m+1} > p_{m-2} > ... > p_n$, тобто:
   - послідовність строго зростає на відрізку $[0, m]$, $m > 0$;
   - послідовність строго спадає на відрізку $[m+1, n]$, $m+1 < n$;
   - максимальне значення досягається не більше ніж у двох точках: m і, можливо, $m+1$.

Нагадаємо, що як $\lfloor x \rfloor$ позначають найбільше ціле число, яке менше чи дорівнює x (*цілу частину числа* x); наприклад, $\lfloor 3.14 \rfloor = 3$, $\lfloor -3.14 \rfloor = -4$.

ТЕОРЕМА 2.1. За фіксованого n послідовність біноміальних коефіцієнтів (C_n^k), $k = 0, 1, 2, ..., n$, унімодальна, $m = \left\lfloor \frac{n}{2} \right\rfloor$. У разі парного n максимум досягається в точці $m = \left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2}$, а в разі непарного — у двох точках: $m = \left\lfloor \frac{n}{2} \right\rfloor = \frac{n-1}{2}$ й $m+1 = \frac{n+1}{2}$.

Вказівка для доведення: оцінити відношення двох сусідніх членів послідовності $\frac{C_n^k}{C_n^{k-1}} = \frac{n-k+1}{k}$.

4. Рівність Вандермонда. Нехай m, n, r — невід'ємні цілі числа, причому $r \leqslant \min\{m, n\}$. Тоді $C_{n+m}^r = \sum_{k=0}^{r} C_m^{r-k} C_n^k$.

Вказівка для доведення: скористатися правилом добутку.

ТЕОРЕМА 2.2 (біноміальна). Нехай x та y — змінні, n — додатне ціле число. Тоді

$$(x+y)^n = \sum_{j=0}^{n} C_n^j x^j y^{n-j} = \sum_{j=0}^{n} C_n^j x^{n-j} y^j.$$

Доведення. Дамо комбінаторне доведення цієї теореми [1]. Оскільки $x^j y^{n-j}$ отримано внаслідок j-кратного вибору x і $(n-j)$-кратного вибору y з n співмножників у виразі $(x+y)^n$, то коефіцієнт при $x^j y^{n-j}$ дорівнює кількості способів j-кратного вибору x з n співмножників, тобто C_n^j. Друга рівність випливає з того, що $C_n^j = C_n^{n-j}$.

Легко переконатись, що $(x-y)^n = \sum_{j=0}^{n} (-1)^j C_n^j x^{n-j} y^j$.

Приклад 2.6. Знайдемо розклад виразу $(x+y)^4$. Скориставшись біноміальною теоремою, можемо записати:

$$(x+y)^4 = C_4^0 x^4 + C_4^1 x^3 y + C_4^2 x^2 y^2 + C_4^3 x y^3 + C_4^4 y^4 = \\ = x^4 + 4x^3 y + 6x^2 y^2 + 4xy^3 + y^4.$$

Біноміальні коефіцієнти можна брати з трикутника Паскаля чи обчислювати за формулою (2.3).

Приклад 2.7. Визначимо коефіцієнт при $x^{12} y^{13}$ в розкладі $(x+y)^{25}$.

Очевидно, що цей коефіцієнт дорівнює

$$C_{25}^{13} = \frac{25!}{13! \cdot 12!} = 5200300.$$

За допомогою біноміальної теореми можна довести ще дві властивості біноміальних коефіцієнтів.

5. $\sum_{k=0}^{n} C_n^k = 2^n$.

   Справді, $2^n = (1+1)^n = \sum_{k=0}^{n} C_n^k 1^{n-k} 1^k = \sum_{k=0}^{n} C_n^k$.

6. $\sum_{k=0}^{n} (-1)^k C_n^k = 0$.

   Аналогічно до попереднього, $0 = [1+(-1)]^n = \sum_{k=0}^{n} C_n^k 1^{n-k} (-1)^k = \sum_{k=0}^{n} (-1)^k C_n^k$.

2.5. Поліноміальна теорема

Як узагальнення бінома розглянемо вираз у вигляді $(x_1 + x_2 + ... + x_k)^n$. Основний результат сформульовано в наведеній нижче теоремі.

ТЕОРЕМА 2.3 (поліноміальна). Вираз $(x_1 + x_2 + ... + x_k)^n$ дорівнює сумі всіх можливих доданків $P_n(n_1, n_2, ..., n_k) x_1^{n_1} x_2^{n_2} ... x_k^{n_k}$, де $n_1 + n_2 + ... + n_k = n$, тобто

$$(x_1 + x_2 + ... + x_k)^n = \sum_{\substack{n_1 \geqslant 0, ..., n_k \geqslant 0 \\ n_1 + ... + n_k = n}} P_n(n_1, ..., n_k) x_1^{n_1} x_2^{n_2} ... x_k^{n_k}.$$

Доведення. Запишемо $(x_1 + x_2 + ... + x_k)^n$ у вигляді добутку n співмножників і розкриємо дужки. Коефіцієнт при $x_1^{n_1} x_2^{n_2} ... x_k^{n_k}$ дорівнює кількості перестановок із повтореннями таких, що елемент x_1 міститься в кожній з них n_1 разів, x_2 — n_2 разів, ..., x_k входить n_k разів, а всього елементів $n_1 + n_2 + ... + n_k = n$. Очевидно, що цей коефіцієнт дорівнює $P_n(n_1, ..., n_k)$.

Отриману формулу називають *поліноміальною*. Вона, зокрема, дає змогу доводити деякі властивості чисел $P_n(n_1, ..., n_k)$. Зазначимо дві з них.

1. Нехай $x_1 = x_2 = ... = x_k = 1$; тоді

$$\sum_{\substack{n_1 \geqslant 0, ..., n_k \geqslant 0 \\ n_1 + ... + n_k = n}} P_n(n_1, n_2, ..., n_k) = k^n.$$

2. Помножимо обидві частини поліноміальної формули для $n-1$ на $(x_1 + x_2 + ... + x_k)$ та порівняємо коефіцієнти при однакових доданках. Одержимо таке співвідношення:

$$P_n(n_1, n_2, ..., n_k) = P_{n-1}(n_1 - 1, n_2, ..., n_k) + P_{n-1}(n_1, n_2 - 1, ..., n_k) + ... + P_{n-1}(n_1, n_2, ..., n_k - 1).$$

2.6. Задача про цілочислові розв'язки

Цю задачу формулюють так: знайти кількість розв'язків рівняння $x_1 + x_2 + ... + x_r = n$ у цілих невід'ємних числах, де n — ціле невід'ємне число.

Узявши такі невід'ємні цілі числа $x_1, x_2, ..., x_r$, що $x_1 + x_2 + ... + x_r = n$, можна одержати сполучення з повтореннями з r елементів по n, а саме: елементів першого типу — x_1 одиниць, другого — x_2, ..., r-го — x_r. Навпаки, якщо є сполучення з повтореннями з r елементів по n, то кількості елементів кожного типу задовольняють рівнянню $x_1 + x_2 + ... + x_r = n$ у цілих невід'ємних числах. Отже, кількість цілих невід'ємних розв'язків цього рівняння дорівнює

$$H_r^n = C_{r+n-1}^n = \frac{(r+n-1)!}{n!\,(r-1)!}.$$

Приклад 2.8. Знайдемо кількість невід'ємних цілих розв'язків рівняння $x_1 + x_2 + x_3 = 11$. Безпосереднє використання попередньої формули дає

$$H_3^{11} = C_{3+11-1}^{11} = C_{13}^{11} = \frac{13!}{11!\cdot 2!} = \frac{12\cdot 13}{2} = 78.$$

Кількість розв'язків рівняння $x_1 + x_2 + ... + x_r = n$ у цілих невід'ємних числах можна визначити й тоді, коли на змінні накладено певні обмеження.

Приклад 2.9. Знайдемо кількість невід'ємних цілих розв'язків рівняння $x_1 + x_2 + x_3 = 11$ за умов $x_1 \geqslant 1$, $x_2 \geqslant 2$, $x_3 \geqslant 3$. Очевидно, що ця задача еквівалентна рівнянню $x_1 + x_2 + x_3 = 5$ без обмежень. Справді, потрібно взяти щонайменше один елемент першого типу, два елементи другого типу, три елементи третього — разом $1 + 2 + 3 = 6$ елементів; отже, $11 - 5 = 6$ елементів залишається для довільного вибору,

$$H_3^5 = C_{3+5-1}^5 = C_7^5 = \frac{7!}{5!\cdot 2!} = \frac{6\cdot 7}{2} = 21.$$

Приклад 2.10. Визначимо кількість розв'язків нерівності $x_1 + x_2 + x_3 \leqslant 11$ в невід'ємних цілих числах. Уведемо допоміжну змінну x_4, яка може набувати цілих невід'ємних значень, і перейдемо до еквівалентної задачі: визначити кількість розв'язків рівняння $x_1 + x_2 + x_3 + x_4 = 11$ в невід'ємних цілих числах. Отже,

$$H_4^{11} = C_{4+11-1}^{11} = C_{14}^{11} = \frac{14!}{11!\cdot 3!} = 364.$$

2.7. Числа Стірлінга другого роду та числа Белла

Розглянемо задачу визначення кількості розбиттів множини A на непорожні частини (розбиття означено в підрозділі 1.13).

Приклад 2.11. Якщо $A = \{a, b, c\}$, то є такі розбиття цієї множини на k непорожніх частин:

$k = 1$: $\{\{a, b, c\}\}$ (одне розбиття);

$k = 2$: $\{\{a, b\}, \{c\}\}$, $\{\{a, c\}, \{b\}\}$, $\{\{a\}, \{b, c\}\}$ (три розбиття);

$k = 3$: $\{\{a\}, \{b\}, \{c\}\}$ (одне розбиття).

Позначимо як $\Phi(n, k)$ кількість розбиттів n-елементної множини A на k непорожніх частин, $n \geqslant 1$, $1 \leqslant k \leqslant n$, як $\Phi(n)$ — кількість усіх розбиттів множини A на непорожні частини. Числа $\Phi(n, k)$ називають *числами Стірлінга другого роду*, а $\Phi(n)$ — *числами Белла*. Очевидно, що

$$\Phi(n) = \sum_{k=1}^{n} \Phi(n, k).$$

Для розглянутого прикладу $\Phi(3, 1) = 1$; $\Phi(3, 2) = 3$; $\Phi(3, 3) = 1$; $\Phi(3) = 1 + 3 + 1 = 5$.

Довільне розбиття множини A на k непорожніх частин можна одержати так:

- із розбиття множини $A\backslash\{a_n\}$ на $(k-1)$ непорожню частину додаванням підмножини $\{a_n\}$;
- із розбиття множини $A\backslash\{a_n\}$ на k непорожніх частин додаванням до однієї з цих частин елемента a_n (це можна зробити k способами).

Звідси випливає тотожність $\Phi(n, k) = \Phi(n-1, k-1) + k\Phi(n-1, k)$. За її допомогою можна побудувати таблицю для чисел $\Phi(n, k)$, а, отже, і $\Phi(n)$ (табл. 2.2).

Для чисел Белла існує проста рекурентна залежність $\Phi(n+1) = \sum_{i=0}^{n} C_n^i \Phi(i)$ (уважаємо, що $\Phi(0) = 1$).

Таблиця 2.2

n \ k	1	2	3	4	5	6	...	
1	1						...	1
2	1	1					...	2
3	1	3	1				...	5
4	1	7	6	1			...	15
5	1	15	25	10	1		...	52
6	1	31	90	65	15	1	...	203
...	...	...	...	...	...	...	...	...

ТЕОРЕМА 2.4. За фіксованого n послідовність $(\Phi(n, k))$, $k = 1, 2, ..., n$, унімодальна [49].

2.8. Генерування перестановок

Проблемі систематичної побудови всіх $n!$ перестановок n-елементної множини присвячено багато публікацій. Ця проблема має давню історію. Її появу можна віднести до початку XVII ст., коли в Англії виникло особливе мистецтво дзвонарства. Воно полягало у вибиванні на n різних дзвонах усіх $n!$ перестановок. Це слід було робити по пам'яті. Тому шанувальники цього мистецтва розробили перші прості методи систематичної побудови всіх перестановок без повторень.

Деякі з цих незаслужено забутих методів було знову відкрито в наш час у зв'язку з появою комп'ютерів. Зазначене мистецтво проіснувало довго. Знаменита „Книга рекордів Гіннеса" містить інформацію про вибивання всіх $8! = 40320$ перестановок на восьми дзвонах у 1963 р. Для цього було потрібно 17 год 58 хв 30 с. Звичайно, використання комп'ютерів дає змогу генерувати перестановки значно швидше.

Кожній n-елементній множині A можна поставити у взаємно-однозначну відповідність множину $A' = \{1, 2, ..., n\}$. Зручно спочатку генерувати перестановки n перших натуральних чисел, а потім замінити кожне число відповідним елементом множини A. Унаслідок цього отримаємо всі перестановки елементів даної множини A.

Існують різні алгоритми побудови всіх перестановок множини $A' = \{1, 2, ..., n\}$. Розглянемо один із них. Цей алгоритм ґрунтується на послідовній побудові перестановок множини A' у лексикографічному порядку [23, 52]. Далі перестановку $(a_1, a_2, ..., a_n)$ для спрощення записів позначатимемо як $a_1a_2 ... a_n$.

На множині всіх перестановок (загальніше — на множині всіх кортежів довжиною n з елементами з множини $A' = \{1, 2, ..., n\}$) означимо *лексикографічний порядок*: $a_1a_2 ... a_n < b_1b_2 ... b_n$, якщо для якогось k, $1 \leqslant k \leqslant n$, виконуються співвідношення $a_1 = b_1$, $a_2 = b_2$, ..., $a_{k-1} = b_{k-1}$, але $a_k < b_k$. У такому разі говорять, що перестановка $a_1a_2 ... a_n$ менша від перестановки $b_1b_2 ... b_n$, або перестановка $b_1b_2 ... b_n$ більша від перестановки $a_1a_2 ... a_n$. Якщо замість чисел 1, 2, ..., n узяти букви a, b, ..., z із природним порядком $a < b < ... < z$, то лексикографічний порядок — це стандартна послідовність, у якій слова довжиною n наведено в словнику.

Перестановку $b_1b_2 ... b_n$ називають *лексикографічно наступною* за $a_1a_2 ... a_n$, якщо не існує такої перестановки $c_1c_2 ... c_n$, що $a_1a_2 ... a_n < c_1c_2 ... c_n$ і $c_1c_2 ... c_n < b_1b_2 ... b_n$.

Приклад 2.12. Перестановка 23415 множини {1, 2, 3, 4, 5} менша від перестановки 23514.

Алгоритм генерування перестановок множини $A' = \{1, 2, ..., n\}$ ґрунтується на процедурі, що будує перестановку, лексикографічно наступну за даною перестановкою $a_1a_2 ... a_n$. Покажемо, як це можна зробити. Спочатку припустимо, що $a_{n-1} < a_n$. Поміняємо місцями a_{n-1} й a_n і одержимо більшу перестановку. Вона лексикографічно наступна, бо ніяка інша перестановка не більша за дану перестановку й не менша за отриману.

Приклад 2.13. Нехай 234156 — задана перестановка; тоді перестановка 234165 лексикографічно наступна.

Тепер розглянемо випадок $a_{n-1} > a_n$. Проглянемо останні три члени перестановки. Якщо $a_{n-2} < a_{n-1}$, то останні три члени можна переставити для отримання наступної перестановки. Поставимо менше з двох чисел a_{n-1} і a_n, яке, однак, більше, ніж a_{n-2}, на позицію $n-2$. Потім розмістимо число, яке залишилося, й a_{n-2} на останніх двох позиціях у висхідному порядку.

Приклад 2.14. Нехай 234165 — задана перестановка; тоді перестановка 234516 лексикографічно наступна.

Узагальнивши ці міркування, одержимо такий алгоритм.

Алгоритм побудови лексикографічно наступної перестановки за перестановкою $a_1a_2...a_n$

Наведемо кроки алгоритму.

Крок 1. Знайти такі числа a_j і a_{j+1}, що $(a_j < a_{j+1}) \wedge (a_{j+1} > a_{j+2} > ... > a_n)$. Для цього треба знайти в перестановці першу справа пару сусідніх чисел, у якій число, що ліворуч, менше від числа, що праворуч.

Крок 2. Записати в j-ту позицію таке найменше з чисел $a_{j+1}, a_{j+2}, ..., a_n$, яке водночас більше, ніж a_j.

Крок 3. Записати у висхідному порядку число a_j і решту чисел $a_{j+1}, a_{j+2}, ..., a_n$ у позиції $j+1, ..., n$.

Обґрунтування алгоритму

Доведемо, що не існує перестановки, яка водночас більша від $a_1a_2...a_n$, але менша від побудованої за цим алгоритмом. Це означає, що побудована перестановка дійсно лексикографічно наступна за даною перестановкою $a_1a_2...a_n$. Справді, за наведеним алгоритмом нова перестановка збігається зі старою в позиціях $1, ..., j-1$. У j-й позиції нова перестановка містить a_k, а стара — a_j, причому $a_k > a_j$. Отже, нова перестановка лексикографічно більша від старої. Окрім того, вона перша в лексикографічному порядку з $a_1, a_2, ..., a_{j-1}, a_k$ у позиціях з 1 до j. Стара перестановка остання з $a_1, a_2, ..., a_{j-1}, a_j$ у цих самих позиціях. Згідно з алгоритмом a_k вибирають найменшим з $a_{j+1}, a_{j+2}, ..., a_n$, але більшим, ніж a_j. Отже, не існує жодної перестановки між старою та новою.

Приклад 2.15. Побудуємо перестановку, наступну в лексикографічному порядку за 362541. Остання пара чисел, у якій перше число менше за друге, — 25. Отже, розглянемо послідовність чисел 541. Серед них найменше число, більше від 2, це — 4. Тепер 4 запишемо на місце 2, а решту чисел 251 розмістимо на останніх трьох позиціях у вихідному порядку: 364125.

Щоб побудувати всі $n!$ перестановок множини $A' = \{1, 2, ..., n\}$, починаємо з лексикографічно найменшої перестановки $123...n$ і послідовно $n!-1$ разів виконуємо алгоритм побудови лексикографічно наступної перестановки.

2.9. Генерування сполучень

Як і в підрозділі 2.8, розглянемо множину $A' = \{1, 2, ..., n\}$. Сполучення без повторень з n елементів по r — це r-елементна підмножина множини A'. Позаяк порядок запису елементів множини неістотний, то домовимося записувати елементи в кожному сполученні у висхідному порядку: наприклад, {3, 5, 1} будемо записуватимемо як {1, 3, 5}. Отже, сполучення $\{a_1, a_2, ..., a_r\}$ розглядатимемо як рядок чисел $a_1a_2...a_r$, причому $a_1 < a_2 < ... < a_r$.

Як і для перестановок, покажемо, як за даним сполученням знайти наступне відповідно до лексикографічного порядку [23, 52]. Припустимо, що $n = 5$ та $r = 3$. Якщо можна збільшити останню цифру, то так і будемо робити. Тому, маючи рядок 123, його можна замінити на 124. Якщо ж маємо 125, останнє число збільшити не можна. Тому переходимо до наступного (справа) числа й дивимось, чи можна його збільшити. У даному разі це можна зробити: потрібно замінити 2

на 3. Проте ми прагнемо побудувати найменший рядок із тих, котрі більші 125. Тому збільшуємо останнє число (тобто 3) на 1 і записуємо результат у наступну позицію. Отже, перші два числа — 1 і 3, тому наступний рядок — 134. Припустимо, що є рядок 145. Останнє й передостаннє числа збільшити не можна. Проте перше число можна збільшити, тому замість 1 пишемо 2. Щоб зробити рядок мінімальним, як останні числа візьмемо 3 та 4, унаслідок чого отримаємо рядок 234.

Узагальнимо ці міркування. Значення останнього числа в рядку — найбільше можливе, якщо воно дорівнює $n=n-r+r$. Якщо останнє число — найбільше можливе, то передостаннє — найбільше можливе, якщо воно дорівнює $n-r+(r-1)$ або $n-r+i$, де $i=r-1$ — позиція цього числа. Загалом, значення кожного i-го числа найбільше можливе, якщо числа праворуч від нього — найбільші можливі, і це значення дорівнює $n-r+i$. Отже, проглядаємо рядок справа наліво й визначаємо, чи дорівнює значення i-го елемента $n-r+i$ (це максимальне значення, яке може бути в i-й позиції). Перше значення, яке не задовольняє цю умову, можна збільшити. Нехай, наприклад, це значення дорівнює m і займає j-ту позицію. Збільшуємо m на 1, а значення кожного елемента, що стоїть після j-го, дорівнює значенню попереднього елемента плюс 1. Тепер можемо сформулювати потрібний алгоритм.

Алгоритм побудови лексикографічно наступного сполучення

Наведемо кроки алгоритму.

Крок 1. Знайти в рядку перший справа елемент a_i такий, що $a_i \neq n-r+i$.

Крок 2. Для знайденого елемента виконати присвоювання $a_i := a_i + 1$.

Крок 3. Для $j=i+1, i+2, ..., r$ виконати $a_j := a_i + j - i$ (або, що те саме, $a_j := a_{j-1} + 1$).

Приклад 2.16. Нехай $A'=\{1, 2, 3, 4, 5, 6\}$. Знайдемо сполучення, наступне за $\{1, 2, 5, 6\}$ у лексикографічному порядку.

Це сполучення подамо рядком 1256. Маємо $n = 6$, $r = 4$. Перший справа з таких елементів, що $a_i \neq 6-4+i$, — це $a_2=2$. Для обчислення наступного більшого сполучення збільшуємо a_2 на 1 й одержуємо $a_2=3$. Тепер нехай $a_3=3+1=4$ і $a_4=3+2=5$. Отже, наступне в лексикографічному порядку сполучення — те, що зображене рядком 1345, тобто $\{1, 3, 4, 5\}$.

Обґрунтування алгоритму

Доведемо, що наведений алгоритм дійсно будує наступне в лексикографічному порядку сполучення. Рядок чисел, яким подано лексикографічно наступне сполучення, відрізняється від рядка, що зображає дане сполучення, з позиції i, бо в даному сполученні в позиціях $i+1, i+2, ..., r$ є максимально можливі числа. Отже, a_i+1 — найменше можливе число, яке можна записати в позицію i, якщо хочемо отримати сполучення, більше від даного. Тоді $a_i+2, ..., a_i+r-i+1$ — найменші можливі числа, які можна записати в позиціях від $i+1$ до r.

Коротко зупинімося на питанні генерування всіх розміщень з n елементів по r. Знову розглядатимемо цю задачу лише для множини $A'=\{1, 2, ..., n\}$. Один із можливих способів її розв'язання такий. Використаємо алгоритм генерування

лексикографічно наступного сполучення для побудови r-елементних сполучень n-елементної множини A'. Після кожної стадії, коли побудовано чергове r-сполучення, застосуємо $r-1$ разів алгоритм побудови перестановки за умови $n=r$ для побудови всіх перестановок елементів цього сполучення як r-елементної множини.

2.10. Генерування розбиттів множини

Опишемо алгоритм генерування всіх розбиттів множини. Ідею цього алгоритму найпростіше пояснити, сформулювавши його в рекурентній формі. Зазначимо спочатку, що кожне розбиття π множини $\{1, 2, ..., n\}$ однозначно задає розбиття π_{n-1} множини $\{1, 2, ..., n-1\}$, одержане з π після вилучення елемента n із відповідного блока (і вилучення порожнього блока, якщо елемент n утворював одноелементний блок). Навпаки, якщо дано розбиття $\sigma = \{A_1, A_2, ..., A_k\}$ множини $\{1, 2, ..., n-1\}$, то легко знайти всі такі розбиття π_n множини $\{1, 2, ..., n-1, n\}$, що $\pi_{n-1} = \sigma$. Це такі розбиття:

$$\begin{array}{llll} \{A_1, & A_2, & ..., & A_k, \quad \{n\}\} \\ \{A_1 \cup \{n\}, & A_2, & ..., & A_k\} \\ \{A_1, & A_2 \cup \{n\}, & ..., & A_k\} \\ ... & ... & ... & ... \quad ... \\ \{A_1, & A_2, & ..., & A_k \cup \{n\}\}. \end{array} \qquad (2.5)$$

Наведені міркування підказують простий рекурентний спосіб генерування всіх розбиттів [23]. Якщо дано список L_{n-1} усіх розбиттів множини $\{1, 2, ..., n-1\}$, то список L_n усіх розбиттів множини $\{1, 2, ..., n-1, n\}$ утворюють заміною кожного розбиття σ в списку L_{n-1} на відповідну йому послідовність (2.5).

Приклад 2.17. На рис. 2.1 показано формування списку всіх розбиттів множини $\{1, 2, 3\}$. Усього розбиттів $\Phi(3) = 5$, де $\Phi(n)$ — число Белла.

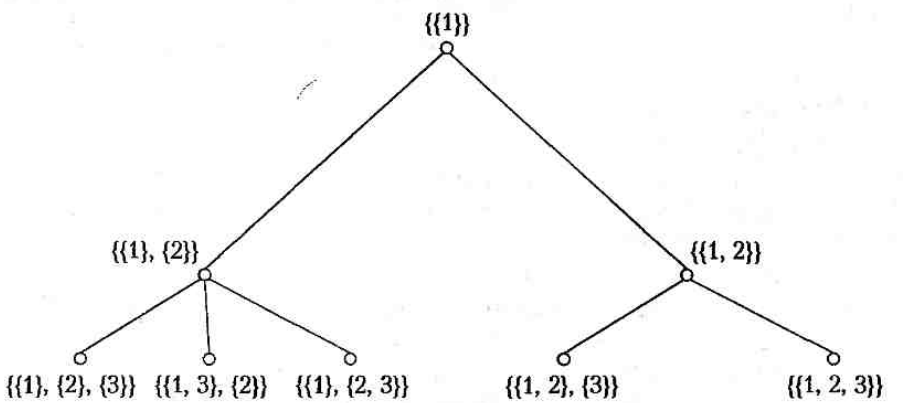


Рис. 2.1

2.11. Рекурентні рівняння

Числову послідовність (a_n) можна задати *рекурентним рівнянням* (використовують також термін *рекурентне співвідношення*). Таке рівняння описує правило для знаходження елементів послідовності через один або декілька попередніх, причому задано відповідну кількість початкових елементів.

Розв'язком рекурентного рівняння називають послідовність, яка задовольняє це рівняння. Інакше кажучи, послідовність задано рекурентною формулою, а потрібно знайти явний вираз для a_n через n.

Метод рекурентних рівнянь у комбінаториці полягає у зведенні комбінаторної задачі до аналогічної задачі для меншої кількості об'єктів.

Приклад 2.18. Розглянемо рекурентне рівняння

$$a_n = a_{n-1} + 2a_{n-2}, \quad n = 2, 3, \ldots, \quad a_0 = 2, a_1 = 7.$$

Його розв'язок — послідовність $a_n = 3 \cdot 2^n - (-1)^n$. Справді,

$$\begin{gathered} a_0 = 3 - 1 = 2, \\ a_1 = 6 + 1 = 7, \\ a_{n-1} + 2a_{n-2} = 3 \cdot 2^{n-1} - (-1)^{n-1} + 2 \cdot 3 \cdot 2^{n-2} - 2 \cdot (-1)^{n-2} = \\ = 3 \cdot 2^{n-1} - (-1)^{n-1} + 3 \cdot 2^{n-1} + 2 \cdot (-1)^{n-1} = \\ = 2 \cdot 3 \cdot 2^{n-1} + (-1)^{n-1} = \\ = 3 \cdot 2^n - (-1)^n = a_n. \end{gathered}$$

Розглянемо дві задачі, що приводять до рекурентних рівнянь.

Числа Фібоначчі. Цю задачу дослідив у XIII ст. Леонардо Пізанський, відомий як Фібоначчі. Молоду різностатеву пару кролів завезли на острів. Після досягнення двомісячного віку кожна пара щомісяця дає приплід — нову пару. Потрібно визначити кількість пар кролів на острові через n місяців.

У кінці першого місяця кількість пар кролів на острові $f_1 = 1$. Оскільки ця пара не дає приплоду впродовж двох місяців, то й $f_2 = 1$. Щоб визначити кількість пар після n місяців, додамо їх кількість у попередньому місяці f_{n-1} і кількість новонароджених пар f_{n-2}: кожна новонароджена пара походить від пари щонайменше двомісячного віку.

Отже, послідовність f_n задовольняє рівняння $f_n = f_{n-1} + f_{n-2}$ з початковими умовами $f_0 = 0, f_1 = 1$. Члени послідовності (f_n) називають *числами Фібоначчі*. Дослідимо зв'язок між числами Фібоначчі й такою комбінаторною задачею: знайти кількість рядків довжиною n з 0 і 1, у яких жодні дві одиниці не записано поспіль. Позначимо кількість таких рядків як g_n. Розглянемо будь-який рядок. Він може закінчуватись або на 0, або на 1. Якщо рядок закінчується на 1, то перед ним записано 0, тобто він закінчується на 01. Отже, кількість рядків довжиною n, що закінчуються на 0, дорівнює g_{n-1}, а таких, що закінчуються на 1 (тобто фактично на 01) — g_{n-2}. Тому $g_n = g_{n-1} + g_{n-2}$.

Легко перевірити, що $g_1 = 2$, $g_2 = 3$. Отже, $g_i = f_{i+1}$, тобто кількість рядків довжиною n з 0 і 1, у яких жодні дві одиниці не записано поруч, дорівнює $(n + 1)$-му числу Фібоначчі.

Задача про багатокутник. У коло вписано правильний $2n$-кутник. Скількома способами можна попарно з'єднати його вершини так, щоб отримані відрізки не перетинались?

Нехай t_n — кількість способів такого з'єднання. Позначимо точки в порядку, у якому їх розміщено на колі: A_1, A_2, ..., A_{2n}. Точку A_1 можна з'єднати лише з однією з точок A_2, A_4, ..., A_{2n}, а ні, то з кожного боку від хорди буде розміщено непарну кількість точок, і в разі попарного з'єднання принаймні одна хорда перетне ту, що виходить з A_1. Припустимо, що точку A_1 з'єднано з A_{2k}. По один бік від хорди A_1A_{2k} міститься $2k - 2$ точок; їх можна з'єднати попарно t_{k-1} способами. З іншого боку від A_1A_{2k} міститься $2(n - k)$ точок, їх можна з'єднати попарно t_{n-k} способами. За правилом добутку кількість таких способів попарного з'єднання, коли A_1 з'єднано з A_{2k}, дорівнює $t_{n-k}\,t_{k-1}$. Параметр k може набувати значень 1, 2, ..., n. Отже,

$$t_n = t_{n-1} + t_{n-2}t_1 + \ldots + t_{n-k}t_{k-1} + \ldots + t_1t_{n-2} + t_{n-1}.$$

2.12. Розв'язування рекурентних рівнянь

Загального методу розв'язування рекурентних рівнянь немає. Проте певний клас рівнянь можна розв'язувати однаковим методом [52].

Рекурентне рівняння називають *лінійним однорідним порядку k зі сталими коефіцієнтами*, якщо воно має вигляд

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}, \tag{2.6}$$

де c_1, c_2, ..., c_k — дійсні числа та $c_k \neq 0$.

Приклад 2.19. Розглянемо рекурентні рівняння:

- $a_n = 1{,}11a_{n-1}$ — лінійне однорідне першого порядку;
- $f_n = f_{n-1} + f_{n-2}$ — лінійне однорідне другого порядку;
- $a_n = a_{n-5}$ — лінійне однорідне п'ятого порядку;
- $a_n = a_{n-1} + a^2_{n-1}$ — нелінійне;
- $a_n = a_{n-1} + a_{n-2}a_{n-3}$ — нелінійне;
- $a_n = 2a_{n-1} + 1$ — лінійне неоднорідне;
- $a_n = 4a_{n-1} - 4a_{n-2} + n2^n$ — лінійне неоднорідне;
- $a_n = na_{n-1}$ — лінійне однорідне, але не зі сталими коефіцієнтами.

Розв'язок рекурентного рівняння k-го порядку називають *загальним*, якщо він залежить від k довільних сталих B_1, ..., B_k і будь-який його розв'язок можна одержати підбором цих сталих.

Щоб рекурентне рівняння визначало конкретну послідовність, достатньо задати k початкових умов: $a_0 = A_0$, $a_1 = A_1$, ..., $a_{k-1} = A_{k-1}$. Із цих умов і визначають сталі B_1, ..., B_k.

ТЕОРЕМА 2.5. Якщо послідовності $a_n^{(1)}, a_n^{(2)}, ..., a_n^{(p)}$ – це розв'язки рекурентного рівняння (2.6), то для довільних чисел $B_1, B_2, ..., B_p$ послідовність

$$a_n = B_1 a_n^{(1)} + B_2 a_n^{(2)} + ... + B_p a_n^{(p)}$$

також являє собою розв'язок цього рівняння.

Доведення. Кожну з тотожностей

$$a_n^{(i)} = c_1 a_{n-1}^{(i)} + c_2 a_{n-2}^{(i)} + ... + c_k a_{n-k}^{(i)}, \quad i = 1, 2, ..., p,$$

помножимо на B_i та додамо результати.

ТЕОРЕМА 2.6. Якщо число r_1 – корінь рівняння

$$r^k = c_1 r^{k-1} + c_2 r^{k-2} + ... + c_k, \tag{2.7}$$

то послідовність r_1^n $(n = 1, 2, ...)$ – розв'язок рекурентного рівняння (2.6).

Доведення. Нехай $a_n = r_1^n$. Підставимо a_n у рівняння (2.6) і одержимо рівність $r_1^n = c_1 r_1^{n-1} + c_2 r_1^{n-2} + ... + c_k r_1^{n-k}$. Вона правильна, оскільки за умовою теореми виконується рівність $r_1^k = c_1 r_1^{k-1} + c_2 r_1^{k-2} + ... + c_k$, і залишається помножити обидві її частини на r_1^{n-k}.

Рівняння (2.7) називають *характеристичним* для рекурентного рівняння (2.6). Це алгебраїчне рівняння степеня k. Його корені можуть бути як простими, так і кратними.

Нехай усі корені характеристичного рівняння прості. Тоді за теоремою 2.6 можна навести k різних розв'язків рекурентного рівняння (2.6): $r_1^n, r_2^n, ..., r_k^n$, де r_i $(i = 1, 2, ..., k)$ – корені характеристичного рівняння (2.7). Зазначимо, що всі r_i відмінні від нуля. Якщо б це було не так, то $c_k = 0$.

Доведемо, що коли всі корені характеристичного рівняння прості, то загальний розв'язок рекурентного рівняння має вигляд

$$a_n = B_1 r_1^n + B_2 r_2^n + ... + B_k r_k^n. \tag{2.8}$$

Безпосередньо з теорем 2.5 і 2.6 випливає, що послідовність (2.8) задовольняє рівняння (2.6). Отже, залишилося довести, що будь-який розв'язок рекурентного рівняння (2.6) можна подати у вигляді (2.8). Позаяк будь-який розв'язок повністю залежить від значень $a_0 = A_0$, $a_1 = A_1$, ..., $a_{k-1} = A_{k-1}$, то достатньо довести, що система лінійних алгебраїчних рівнянь

$$\begin{cases} B_1 + B_2 + ... + B_k = A_0, \\ B_1 r_1 + B_2 r_2 + ... + B_k r_k = A_1, \\ B_1 r_1^2 + B_2 r_2^2 + ... + B_k r_k^2 = A_2, \\ \quad ... \quad ... \quad ... \quad ... \quad ... \\ B_1 r_1^{k-1} + B_2 r_2^{k-1} + ... + B_k r_k^{k-1} = A_{k-1} \end{cases} \tag{2.9}$$

має розв'язок за будь-яких $A_0, A_1, ..., A_{k-1}$.

Визначник системи (2.9)

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ r_1 & r_2 & \dots & r_k \\ r_1^2 & r_2^2 & \dots & r_k^2 \\ \dots & \dots & \dots & \dots \\ r_1^{k-1} & r_2^{k-1} & \dots & r_k^{k-1} \end{vmatrix} -$$

це визначник Вандермонда; він дорівнює добутку $\prod_{i>j}(r_i - r_j)$. Оскільки всі r_i різні, то визначник відмінний від 0, і система (2.9) має єдиний розв'язок за будь-яких $A_0, A_1, \dots, A_{k-1}$.

Розглянемо тепер випадок кратних коренів. Вираз (2.8) у цьому разі — уже не загальний розв'язок. Справді, нехай, наприклад, $r_1 = r_2$. Тоді

$$a_n = (B_1 + B_2)r_1^n + B_3 r_3^n + \dots + B_k r_k^n = Br_1^n + B_3 r_3^n + \dots + B_k r_k^n.$$

Залишилося $(k-1)$ довільних сталих. Їх потрібно визначити так, щоб задовольнити k початкових умов $a_0 = A_0$, $a_1 = A_1$, ..., $a_{k-1} = A_{k-1}$. Узагалі кажучи, зробити це неможливо.

Нехай характеристичне рівняння (2.7) має s різних коренів $r_1, r_2, \dots, r_s$, кратність яких дорівнює відповідно $k_1, k_2, \dots, k_s$ $(k_1 + k_2 + \dots + k_s = k)$. Щоб побудувати загальний розв'язок рекурентного рівняння (2.6) у цьому разі, потрібно доповнити кількість розв'язків, яких не вистачає через кратність коренів $r_1, r_2, \dots, r_s$. Можна довести, що окрім r_j^n розв'язки рівняння (2.6) — це також nr_j^n, $n^2 r_j^n, \dots, n^{k_j-1} r_j^n$ $(j = 1, 2, \dots, s)$. Загальний розв'язок у разі кратних коренів має такий вигляд:

$$a_n = \sum_{j=1}^{s} (B_{j_1} + B_{j_2} n + \dots + B_{j_{k_j}} n^{k_j-1}) r_j^n.$$

Приклад 2.20. Послідовність чисел Фібоначчі задає рекурентне рівняння другого порядку $f_n = f_{n-1} + f_{n-2}$ з початковими умовами $f_0 = 0$, $f_1 = 1$. Характеристичне рівняння — $r^2 = r + 1$, тобто $r^2 - r - 1 = 0$, звідки випливає, що

$$r_1 = \frac{1+\sqrt{5}}{2}, \quad r_2 = \frac{1-\sqrt{5}}{2}.$$

Отже,

$$f_n = B_1\left(\frac{1+\sqrt{5}}{2}\right)^n + B_1\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Для визначення констант B_1 і B_2 скористаємося початковими умовами

$$\begin{cases} B_1 + B_2 = 0, \\ \left(\frac{1+\sqrt{5}}{2}\right)B_1 + \left(\frac{1-\sqrt{5}}{2}\right)B_2 = 1. \end{cases}$$

Отримаємо $B_1 = \frac{1}{\sqrt{5}}$; $B_2 = -\frac{1}{\sqrt{5}}$. Отже, $f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$.

Приклад 2.21. Розглянемо рекурентне рівняння четвертого порядку $a_n = 5a_{n-2} - 4a_{n-4}$ з початковими умовами $a_0 = 3$, $a_1 = 2$, $a_2 = 6$, $a_3 = 8$. Характеристичне рівняння $-r^4 - -5r^2 + 4 = 0$. Розклавши ліву частину на множники, послідовно одержимо

$$r^4 - 5r^2 + 4 = (r^2 - 1)(r^2 - 4) = (r-1)(r+1)(r-2)(r+2);$$
$$(r-1)(r+1)(r-2)(r+2) = 0;$$
$$r_1 = 1,\ r_2 = -1,\ r_3 = 2,\ r_4 = -2.$$

Отже, загальний розв'язок має вигляд $a_n = B_1 + B_2(-1)^n + B_3 2^n + B_4(-2)^n$.

Скориставшись початковими умовами, запишемо систему рівнянь для визначення констант:

$$\begin{cases} B_1 + B_2 + B_3 + B_4 = 3, \\ B_1 - B_2 + 2B_3 - 2B_4 = 2, \\ B_1 + B_2 + 4B_3 + 4B_4 = 6, \\ B_1 - B_2 + 8B_3 - 8B_4 = 8. \end{cases}$$

Розв'язавши її, одержимо $B_1 = B_2 = B_3 = 1$, $B_4 = 0$. Отже, $a_n = 1 + (-1)^n + 2^n$.

Приклад 2.22. Нехай задано рекурентне рівняння $a_n = 6a_{n-1} - 9a_{n-2}$, $a_0 = 3$, $a_1 = 3$. Його характеристичне рівняння $-r^2 + 6r + 9 = 0$; воно має розв'язки $r_1 = r_2 = -3$. Загальний розв'язок має вигляд

$$a_n = B_1(-3)^n + B_2 n(-3)^n.$$

Для визначення констант, виходячи з початкових умов, складемо систему лінійних рівнянь

$$\begin{cases} B_1 = 3, \\ -3B_1 - 3B_2 = -3, \end{cases}$$

з якої знаходимо $B_1 = 3$, $B_2 = -2$. Отже, $a_n = 3(-3)^n + 2n(-3)^n = (3 - 2n)(-3)^n$.

Коротко розглянемо лінійні неоднорідні рекурентні рівняння зі сталими коефіцієнтами

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} + q_n, \qquad (2.10)$$

де q_n — відома послідовність.

ТЕОРЕМА 2.7. Загальний розв'язок a_n лінійного неоднорідного рівняння (2.10) дорівнює сумі його часткового розв'язку $\tilde{a}_n$ і загального розв'язку α_n відповідного лінійного однорідного рівняння.

Доведення. Нехай $\tilde{a}_n$ — будь-який частковий розв'язок неоднорідного рівняння (2.10). Тоді замінимо a_n на $\tilde{a}_n + \alpha_n$ і отримаємо

$$\tilde{a}_n + \alpha_n = \sum_{i=1}^{k} c_i \left(\tilde{a}_{n-i} + \alpha_{n-i}\right) + q_n.$$

Оскільки $\tilde{a}_n$ — частковий розв'язок неоднорідного рівняння, то

$$\tilde{a}_n = \sum_{i=1}^{k} c_i \tilde{a}_{n-i} + q_n.$$

Отже, α_n задовольняє однорідному рекурентному рівнянню $\alpha_n = \sum_{i=1}^{k} c_i \alpha_{n-i}$.

Теорема 2.7 зводить задачу знаходження загального розв'язку неоднорідного рекурентного рівняння (2.10) до відшукання будь-якого його часткового розв'язку. У застосуваннях часто

$$q_n = (b_0 + b_1 n + b_2 n^2 + \ldots + b_p n^p) z^n, \qquad (2.11)$$

де $b_0, b_1, \ldots, b_p, z$ – задані дійсні числа.

Опишемо метод знаходження часткового розв'язку неоднорідного рекурентного рівняння (2.10) із q_n у вигляді (2.11). Припустимо, що відомі корені $r_1, r_2, \ldots, r_s$ характеристичного рівняння (2.7), кратності яких дорівнюють відповідно $k_1, k_2, \ldots, k_s$, причому $k_1 + k_2 + \ldots + k_s = k$. Можна довести, що тоді існує частковий розв'язок рівняння (2.10)

$$\tilde{a}_n = n^t (D_0 + D_1 n + D_2 n^2 + \ldots + D_p n^p) z^n,$$

де $t = 0$, якщо $z \neq r_i$ $(i = 1, \ldots, s)$, і $t = k_j$, якщо $z = r_j$ для якогось j (тобто t дорівнює кратності кореня r_j, якщо $z = r_j$).

Приклад 2.23. Розв'яжемо неоднорідне рекурентне рівняння $a_n = a_{n-2} + n + 1$ із початковими умовами $a_0 = 5$, $a_1 = -1$.

Відповідне однорідне рівняння – $a_n = a_{n-2}$. Характеристичне рівняння – $r^2 - 1 = 0$, його корені – $r_1 = -1$, $r_2 = 1$. Загальний розв'язок однорідного рівняння має вигляд

$$\alpha_n = B_1(-1)^n + B_2 \cdot 1^n = B_1(-1)^n + B_2.$$

Частковий розв'язок неоднорідного рівняння шукаємо у вигляді $\tilde{a}_n = n(D_0 + D_1 n)$. Підставивши його в неоднорідне рівняння, одержимо

$$D_0 n + D_1 n^2 = D_0(n-2) + D_1(n-2)^2 + n + 1,$$
$$(4D_1 - 2D_0 + 1) + (1 - 2D_1)n = 0,$$

звідки

$$\begin{cases} 1 - 2D_1 = 0, \\ 4D_1 - 2D_0 + 1 = 0. \end{cases}$$

З останньої системи отримаємо $D_1 = 1/2$, $D_0 = 3/2$. Отже, частковий розв'язок неоднорідного рівняння має вигляд $\tilde{a}_n = (3/2)n + (1/2)n^2$. На підставі теореми 2.7 запишемо загальний розв'язок неоднорідного рівняння:

$$a_n = \tilde{a}_n + \alpha_n = \frac{3}{2}n + \frac{1}{2}n^2 + B_1(-1)^n + B_2.$$

Підставивши початкові умови, одержимо систему лінійних алгебраїчних рівнянь для визначення констант B_1 і B_2:

$$\begin{cases} B_1 + B_2 = 5, \\ -B_1 + B_2 = -3, \end{cases}$$

її корені – $B_1 = 4$, $B_2 = 1$. Отже, $a_n = 4(-1)^n + 1 + (3/2)n + (1/2)n^2$.

Приклад 2.24. Знайдемо загальний розв'язок рекурентного рівняння

$$a_n = 4a_{n-1} - 4a_{n-2} + n \cdot 2^n.$$

Характеристичне рівняння має двократний корінь $r = 2$. Отже, частковий розв'язок неоднорідного рівняння має вигляд $\tilde{a}_n = n^2(D_0 + D_1 n)2^n$. Підставивши його у вихідне рівняння та зробивши скорочення на 2^n, одержимо

$$n^2(D_0 + D_1 n) = (6D_1 - 2D_0) + (1 - 6D_1)n + (D_0 + D_1 n)n^2,$$

звідки можна записати систему

$$\begin{cases} 6D_1 - 2D_0 = 0, \\ 1 - 6D_1 = 0. \end{cases}$$

Розв'язавши її, отримаємо $D_0 = 1/2$, $D_1 = 1/6$. Загальний розв'язок однорідного рівняння має вигляд $a_n = (B_0 + B_1 n)2^n$, а неоднорідного —

$$a_n = (B_0 + B_1 n)2^n + n^2\left(\frac{1}{2} + \frac{1}{6}n\right)2^n.$$

2.13. Принцип коробок Діріхле

Принцип коробок Діріхле широко застосовують у теорії скінченних автоматів, теорії чисел та інших розділах [6, 52].

ТЕОРЕМА 2.8 (принцип коробок Діріхле). Якщо $k + 1$ або більше предметів розкладено в k коробках, то існує принаймні одна коробка, яка містить два чи більше предметів.

Доведення. Припустимо, що жодна коробка не містить більше одного предмета. Тоді загальна кількість предметів становить щонайбільше k. Це суперечить тому, що є щонайменше $k + 1$ предмет.

Приклад 2.25. У будь-якій групі з 367 чоловік принаймні двоє народилися в один день (можливо, у різні роки).

Нагадаємо, що найменше ціле число, яке більше за x або дорівнює йому, позначають $\lceil x \rceil$. Наприклад, $\lceil 3{,}14 \rceil = 4$.

ТЕОРЕМА 2.9 (узагальнений *принцип коробок Діріхле*). *Якщо N предметів* розкладено в k коробках, то існує принаймні одна коробка, яка містить щонайменше $\lceil N/k \rceil$ предметів.

Доведення. Зазначимо, що справджується нерівність $\lceil N/k \rceil < (N/k) + 1$. Припустимо, що жодна коробка не містить більше ніж $\lceil N/k \rceil - 1$ предметів. Тоді загальна кількість предметів становить щонайбільше

$$k(\lceil N/k \rceil - 1) < k(((N/k) + 1) - 1) = N.$$

Це суперечить умові теореми, що загальна кількість предметів дорівнює N.

Приклад 2.26. Серед 100 чоловік принаймні $\lceil 100/12 \rceil = 9$ народилися в одному місяці.

2.14. Принцип включення-виключення

Цей принцип дає відповідь на запитання, як визначити кількість елементів у об'єднанні множин. Для двох множин правдива формула

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Приклад 2.27. Знайдемо кількість додатних цілих чисел, що не перевищують 1000 та діляться на 7 або на 11. Позначимо як A множину чисел, що діляться на 7, B — множину чисел, що діляться на 11. Тоді

$$|A \cup B| = |A| + |B| - |A \cap B| = \left\lfloor \frac{100}{7} \right\rfloor + \left\lfloor \frac{1000}{11} \right\rfloor - \left\lfloor \frac{1000}{7 \cdot 11} \right\rfloor = 142 + 90 - 12 = 220.$$

Для трьох множин формула для кількості елементів у їх об'єднанні ускладнюється:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|.$$

Приклад 2.28. Одну з мов (англійську, німецьку, іспанську) вивчає 231 студент, причому $|E| = 180, |D| = 110, |S| = 70, |E \cap D| = 82, |E \cap S| = 40, |D \cap S| = 15$, де як E, D, S позначено множини студентів, які відповідно вивчають англійську, німецьку й іспанську мови. Скільки студентів вивчають усі три мови? Маємо

$$231 = 180 + 110 + 70 - 82 - 40 - 15 + |E \cap D \cap S|,$$

звідки випливає, що $|E \cap D \cap S| = 8$ студентів.

ТЕОРЕМА 2.10 (принцип включення-виключення). Нехай $A_1, A_2, ..., A_n$ — скінченні множини. Тоді

$$|A_1 \cup A_2 \cup ... \cup A_n| = \sum_{1 \leqslant i \leqslant n} |A_i| - \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| +$$

$$+ \sum_{1 \leqslant i < j < k \leqslant n} |A_i \cap A_j \cap A_k| - ... + (-1)^{n+1} |A_1 \cap A_2 \cap ... \cap A_n|.$$

Доведення. Достатньо довести, що кожний елемент в об'єднанні множин враховано в правій частині рівності точно один раз. Припустимо, що елемент a належить рівно r множинам з $A_1, A_2, ..., A_n$, де $1 \leqslant r \leqslant n$. Тоді цей елемент враховано C_r^1 разів у $\sum_{1 \leqslant i \leqslant n} |A_i|$, C_r^2 разів у $\sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j|$; загалом його враховано C_r^m разів під час сумування членів, які містять перетин m множин A_i. Отже, елемент a враховано точно $C_r^1 - C_r^2 + C_r^3 - ... + (-1)^r C_r^r$ разів у виразі в правій частині рівності. За властивістю біноміальних коефіцієнтів $C_r^0 - C_r^1 + C_r^2 - ... + (-1)^r C_r^r = 0$. Отже, $C_r^0 = C_r^1 - C_r^2 + C_r^3 - ... + (-1)^{r+1} C_r^r$, але $C_r^0 = 1$, і тому $C_r^1 - C_r^2 + C_r^3 - ... + (-1)^{r+1} C_r^r = 1$. Це й означає, що кожний елемент об'єднання множин враховано в правій частині рівності точно один раз.

Зазначимо, що формула включення-виключення містить $2^n - 1$ доданків, по одному для кожної непорожньої підмножини з $\{A_1, A_2, ..., A_n\}$.

2.15. Принцип включення-виключення в альтернативній формі

Ця форма принципу включення-виключення може бути корисною для розв'язування задач, у яких потрібно знайти кількість елементів заданої множини A, які не мають жодної з n властивостей $\alpha_1, \alpha_2, ..., \alpha_n$.

Уведемо такі позначення:

- $A_i \subset A$ — підмножина елементів, що мають властивість α_i;
- $N(\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_k})$ — кількість елементів множини A, які водночас мають властивості $\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_k}$;
- $N(\bar{\alpha}_{i_1}, \bar{\alpha}_{i_2}, ..., \bar{\alpha}_{i_k})$ — кількість елементів множини A, які не мають жодної з властивостей $\alpha_1, \alpha_2, ..., \alpha_n$;
- N — кількість елементів у заданій множині A.

Тоді, очевидно,

$$N(\bar{\alpha}_1, \bar{\alpha}_2, ..., \bar{\alpha}_n) = N - |A_1 \cup A_2 \cup ... \cup A_n|.$$

За принципом включення-виключення можна записати

$$N(\bar{\alpha}_1, \bar{\alpha}_2, ..., \bar{\alpha}_n) = N - \sum_{1 \leqslant i \leqslant n} N(\alpha_i) + \sum_{1 \leqslant i < j \leqslant n} N(\alpha_i, \alpha_j) - \\ - \sum_{1 \leqslant i < j < k \leqslant n} N(\alpha_i, \alpha_j, \alpha_k) + ... + (-1)^n N(\alpha_1, \alpha_2, ..., \alpha_n). \tag{2.12}$$

Формула (2.12) подає принцип включення-виключення *в альтернативній формі.*

Приклад 2.29. Знайдемо кількість розв'язків рівняння $x_1 + x_2 + x_3 = 11$ у невід'ємних цілих числах у разі обмежень $x_1 \leqslant 3$, $x_2 \leqslant 4$, $x_3 \leqslant 6$.

Розглянемо альтернативні властивості: α_1: $x_1 \geqslant 4$; α_2: $x_2 \geqslant 5$; α_3: $x_3 \geqslant 7$. За формулою (2.12) кількість розв'язків, що водночас задовольняють нерівності $x_1 \leqslant 3$, $x_2 \leqslant 4$ та $x_3 \leqslant 6$, дорівнює

$$N(\bar{\alpha}_1, \bar{\alpha}_2, ..., \bar{\alpha}_n) = N - N(\alpha_1) + N(\alpha_2) + N(\alpha_3) + N(\alpha_1, \alpha_2) + \\ + N(\alpha_1, \alpha_3) + N(\alpha_2, \alpha_3) - N(\alpha_1, \alpha_2, \alpha_3).$$

Далі маємо:

$N = H_3^{11} = C_{13}^{11} = 78$ (загальна кількість розв'язків);

$N(\alpha_1) = H_3^7 = C_9^7 = 36$ (кількість розв'язків, що задовольняють умову $x_2 \geqslant 4$);

$N(\alpha_2) = H_3^6 = C_8^6 = 28$ $(x_2 \geqslant 5)$;

$N(\alpha_3) = H_3^4 = C_6^4 = 15$ $(x_3 \geqslant 7)$;

$N(\alpha_1, \alpha_2) = H_3^2 = C_4^2 = 6$ $(x_1 \geqslant 4) \wedge (x_2 \geqslant 5)$;

$N(\alpha_1, \alpha_3) = H_3^0 = 1$ $(x_1 \geqslant 4) \wedge (x_3 \geqslant 7)$;

$N(\alpha_2, \alpha_3) = 0$ $(x_2 \geqslant 5) \wedge (x_3 \geqslant 7)$;

$N(\alpha_1, \alpha_2, \alpha_3) = 0$ $(x_1 \geqslant 4) \wedge (x_2 \geqslant 5) \wedge (x_3 \geqslant 7)$.

Отже, кількість розв'язків із зазначеними обмеженнями дорівнює

$$N(\overline{\alpha}_1, \overline{\alpha}_2, ..., \overline{\alpha}_n) = 78 - 36 - 28 - 15 + 6 + 1 + 0 - 0 = 6.$$

Розглянемо частковий випадок формули (2.12). Припустимо, що величини $N(\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_k})$ не залежать від самих властивостей $\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_k}$, а залежать лише від їх кількості. Уважатимемо за означенням $N^{(k)} = N(\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_k})$ для будь-якого набору k властивостей. Тоді формула (2.12) набуває вигляду

$$N^{(0)} = N - C_n^1 N^{(1)} + C_n^2 N^{(2)} - ... + (-1)^n N^{(n)}. \qquad (2.13)$$

Приклад 2.30 (*задача про зміщення*). Знайдемо кількість перестановок з n елементів, у яких жодний елемент не залишається в початковому положенні.

Кількість таких перестановок позначають як D_n. Кількість розміщень, у яких не зміщений один елемент, дорівнює $n(n-1)! = C_n^1 P_{n-1}$. Аналогічно, кількість розміщень, у яких не зміщені k елементів, дорівнює $C_n^k P_{n-k}$. Отже, за формулою (2.13) можна записати:

$$D_n = P_n - C_n^1 P_{n-1} + C_n^2 P_{n-2} - ... + (-1)^n C_n^n = n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - ... + \frac{(-1)^n}{n!}\right].$$

2.16. Твірні функції

Метод твірних функцій – один із найуніверсальніших методів комбінаторики. Строге його обґрунтування спирається на поняття й результати математичного аналізу. Наведемо спочатку потрібні для нас відомості.

2.16.1. Степеневі ряди та їхні властивості

Степеневим називають ряд

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n + ...,$$

де $a_0, a_1, a_2, ..., a_n, ...$ – дійсні числа. Суму ряду позначатимемо як $f(x)$. Степеневі ряди мають такі важливі властивості.

1. *Область збіжності* ряду – множина $\{x \mid |x| < r\}$, до якої іноді можуть долучатися точки $x = r$ та $x = -r$ або одна з цих точок. Число r називають *радіусом збіжності* ряду.

2. Нехай $f(x) = \sum_{i=0}^{\infty} a_i x^i$ та $\varphi(x) = \sum_{i=0}^{\infty} b_i x^i$ – степеневі ряди; r_1, r_2 – радіуси їх збіжності, а $r_0 = \min\{r_1, r_2\}$. Тоді для $|x| < r_0$ ці ряди можна почленно додавати й множити:

$$\left.\begin{aligned} f(x) + \varphi(x) &= \sum_{i=0}^{\infty} (a_i + b_i) x^i; \\ f(x)\varphi(x) &= \sum_{i=0}^{\infty} (a_0 b_i + a_1 b_{i-1} + ... + a_i b_0) x^i. \end{aligned}\right\} \qquad (2.14)$$

3. Якщо два степеневі ряди збігаються в області $\{x \mid |x| < r\}$ і мають однакову суму для всіх x із цієї області, то коефіцієнти при відповідних степенях x цих рядів однакові.

4. Сума ряду $f(x) = \sum_{i=0}^{\infty} a_i x^i$ в області $\{x \mid |x| < r\}$, де r — радіус збіжності, має похідні всіх порядків, причому

$$f^{(k)}(x) = \sum_{n=k}^{\infty} n(n-1)\ldots(n-k+1)\,a_n x^{n-k};$$

зокрема,

$$a_k = \frac{f^{(k)}(0)}{k!}, \quad k = 0, 1, 2, \ldots.$$

Ця властивість дає змогу визначати окремі члени ряду, якщо є аналітичний вираз для його суми $f(x)$. Властивості 1–4 доводять у курсі математичного аналізу.

Ньютон узагальнив формулу $(1+x)^n$ для ненатуральних показників. Якщо $|x| < 1$, то для будь-якого дійсного значення v справджується рівність

$$(1+x)^v = 1 + vx + \frac{v(v-1)}{1\cdot 2}x^2 + \ldots + \frac{v(v-1)\ldots(v-r+1)}{1\cdot 2\cdot\ldots\cdot r}x^r + \ldots. \qquad (2.15)$$

Рівність (2.15) називають *біноміальним рядом Ньютона*. Нам потрібні два випадки формули (2.15) — для $v = -n$ (де n — натуральне) та $v = 1/2$:

$$\begin{aligned}(1+x)^{-n} &= 1 + (-n)x + \frac{(-n)(-n-1)}{1\cdot 2}x^2 + \ldots \\ &\ldots + \frac{(-n)(-n-1)\ldots(-n-r+1)}{1\cdot 2\cdot\ldots\cdot r}x^r + \ldots = \\ &= 1 - nx + \frac{n(n+1)}{1\cdot 2}x^2 + \ldots + (-1)^r\frac{n(n+1)\ldots(n+r-1)}{1\cdot 2\cdot\ldots\cdot r}x^r + \ldots = \\ &= 1 - \frac{n!}{1!(n-1)!}x + \frac{(n+1)!}{2!(n-1)!}x^2 + \ldots + (-1)^r\frac{(n+r-1)!}{r!(n-1)!}x^r + \ldots = \\ &= 1 - C_n^1 x + C_{n+1}^2 x^2 + \ldots + (-1)^r C_{n+r-1}^r x^r + \ldots;\end{aligned} \qquad (2.16)$$

$$\begin{aligned}\sqrt{1+x} &= 1 + \frac{1}{2}x + \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)}{1\cdot 2}x^2 + \ldots + \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\ldots\left(\frac{1}{2}-r+1\right)}{1\cdot 2\cdot\ldots\cdot r}x^r + \ldots = \\ &= 1 + \frac{1}{2}x - \frac{1}{2\cdot 2^3}C_2^1 x^2 + \ldots + \frac{(-1)^{r-1}}{r\cdot 2^{2r-1}}C_{2r-2}^{r-1}x^r + \ldots.\end{aligned} \qquad (2.17)$$

2.16.2. Поняття твірної функції

Нехай задано послідовність чисел $a_0, a_1, a_2, ..., a_n, ...$. Утворимо степеневий ряд $a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_nx^n + ...$. Якщо він збігається в якійсь області до функції $f(x)$, то її називають *твірною* для послідовності чисел $a_0, a_1, a_2, ..., a_n, ...$. Якщо послідовність $a_0, a_1, a_2, ..., a_n$ скінченна, то твірна функція для цієї послідовності — поліном $a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_nx^n$.

2.16.3. Твірні функції для сполучень

Розглянемо множину об'єктів $S = \{S_1, S_2, ..., S_n\}$. Поставимо їй у відповідність послідовність (s_i) $(i = 1, 2, ..., n)$ так, що елемент s_i цієї послідовності відповідає об'єкту S_i. Розглянемо добуток

$$(1 + s_1x)(1 + s_2x)...(1 + s_nx). \tag{2.18}$$

Формально перемножимо дужки, тобто дістанемо розклад

$$\begin{aligned} &1 + (s_1 + s_2 + ... + s_n)x + (s_1s_2 + s_1s_3 + ... + s_{n-1}s_n)x^2 + \\ &+(s_1s_2s_3 + s_1s_2s_4 + ... + s_{n-2}s_{n-1}s_n)x^3 + ... + s_1s_2...s_n\,x^n. \end{aligned} \tag{2.19}$$

Уведемо такі позначення:

$$\begin{aligned} E_0 &= 1, \\ E_1 &= s_1 + s_2 + ... + s_n, \\ E_2 &= s_1s_2 + s_1s_3 + ... + s_{n-1}s_n, \\ E_3 &= s_1s_2s_3 + s_1s_2s_4 + ... + s_{n-2}s_{n-1}s_n, \\ &\quad ... \quad ... \quad ... \quad ... \quad ... \\ E_n &= s_1s_2 \; ... \; s_n. \end{aligned}$$

Доданки в E_r відповідають усім сполученням без повторень з n об'єктів по r. Отже, кількість доданків у кожному коефіцієнті E_r дорівнює кількості сполучень без повторень з n елементів по r.

Узявши у виразі (2.18) $s_1 = s_2 = ... = s_n = 1$, одержимо $(1 + x)^n$, а коефіцієнти розкладу (2.19) дорівнюють кількості сполучень без повторень з n об'єктів по r:

$$(1 + x)^n = C_n^0 + C_n^1x + C_n^2x^2 + ... + C_n^rx^r + ... + C_n^nx^n.$$

Отже, $(1 + x)^n$ — твірна функція для послідовності $C_n^0, C_n^1, C_n^2, ..., C_n^n$ [19]. Використовуючи твірну функцію $(1 + x)^n$, можна довести різні властивості чисел C_n^r.

- *Рівність Паскаля*: $C_n^r = C_{n-1}^r + C_{n-1}^{r-1}$. Помноживши обидві частини рівності

$$(1 + x)^{n-1} = C_{n-1}^0 + C_{n-1}^1x + ... + C_{n-1}^{n-1}x^{n-1}$$

на $(1 + x)$, отримаємо

$$(1 + x)^n = (C_{n-1}^0 + C_{n-1}^1x + ... + C_{n-1}^{r-1}x^{n-1})(1 + x).$$

Порівняємо коефіцієнти при x^r у лівій і правій частинах і одержимо потрібну тотожність.

- *Рівність Вандермонда*: $C_{m+n}^r = \sum_{k=0}^{r} C_m^{r-k} C_n^k$, де $r \leqslant \min\{m, n\}$. Запишемо $(1+x)^{n+r}$ у вигляді

$$(1+x)^{n+m} = (C_n^0 + C_n^1 x + ... + C_n^n x^n)(C_m^0 + C_m^1 x + ... + C_m^m x^m).$$

Обчислимо коефіцієнти при x^r у правій частині за формулою (2.14) і порівняємо з коефіцієнтом при x^r у лівій частині. Одержимо рівність Вандермонда.

Розглянемо тепер, як можна побудувати твірну функцію для сполучень із повтореннями. У попередніх міркуваннях кожний об'єкт S_k ($k = 1, 2, ..., n$) міг з'явитись у будь-якому сполученні не більше одного разу, оскільки множники в добутку (2.18) складаються лише з двох доданків. Твірну функцію для сполучень із повтореннями, у яких об'єкти S_k можуть міститися 0, 1, 2, 3, ... разів, можна одержати аналогічно до попереднього. Потрібно лише замінити множники $(1 + s_k x)$ у виразі (2.18) множниками $(1 + s_k x + s_k^2 x^2 + s_k^3 x^3 + ...)$.

Звідси випливає, що твірна функція для сполучень із повтореннями —

$$(1 + x + x^2 + x^3 + ...)^n = \left(\frac{1}{1-x}\right)^n = (1-x)^{-n}.$$

Використовуючи розклад (2.16), отримаємо

$$(1-x)^{-n} = 1 + C_n^1 x + C_{n+1}^2 x^2 + ... + C_{n+r-1}^r x^r + ...$$

Отже, ми іншим способом одержали кількість $H_n^r = C_{n+r-1}^r$ сполучень із повтореннями з n елементів по r. Аналогічно можна вивчати сполучення, на які накладено інші, загальніші умови.

Визначимо кількість сполучень із повтореннями з n об'єктів по r, у яких кожний об'єкт зустрічається не менше одного разу. Аналогічно до попереднього дістанемо твірну функцію для таких сполучень:

$$(x + x^2 + x^3 + ...)^n = x^n (1-x)^{-n} = x^n \sum_{r=0}^{\infty} C_{n+r-1}^r x^r =$$

$$= \sum_{r=0}^{\infty} C_{n+r-1}^r x^{n+r} = \sum_{r=0}^{\infty} C_{n+r-1}^{n+r-n} x^{n+r} = \sum_{r=n}^{\infty} C_{r-1}^{r-n} x^r.$$

Тут використано формулу (2.16). Отже, кількість сполучень із повтореннями з n об'єктів по r, у яких кожний об'єкт зустрічається не менше одного разу, дорівнює C_{r-1}^{r-n}, де $r = n, n+1, ...$.

Приклад 2.31. Нехай є два об'єкти A та B. Кількість чотириелементних сполучень із цих об'єктів, у яких кожний із них зустрічається не менше одного разу, дорівнює $C_{r-1}^{r-n} = C_{4-1}^{4-2} = C_3^2 = \frac{3!}{2!\cdot 1!} = 3$. Випишемо ці сполучення:

$$[A, A, A, B], [A, A, B, B], [A, B, B, B].$$

Узагальнимо попередню задачу: обчислимо кількість сполучень з n об'єктів по r із повтореннями, у яких кожний об'єкт зустрічається не менше k разів. Із цією метою розглянемо таку твірну функцію:

$$(x^k + x^{k+1} + x^{k+2} + \ldots)^n = x^{kn}(1-x)^{-n} = x^{kn}\sum_{r=0}^{\infty} C_{n+r-1}^{r}\, x^r = \sum_{r=0}^{\infty} C_{n+r-1}^{r} x^{kn+r} =$$

$$= \sum_{r=0}^{\infty} C_{(k-1)n+n+r-1-(k-1)n}^{kn+r-kn}\, x^{kn+r} = \sum_{r=kn}^{\infty} C_{r-1-(k-1)n}^{r-kn}\, x^r.$$

Кількість сполучень із зазначеною властивістю дорівнює коефіцієнту при x^r, тобто

$$C_{r-1-(k-1)n}^{r-kn}, \quad 0 \leqslant k \leqslant r, \quad r = kn, kn+1, \ldots.$$

Приклад 2.32. Нехай є три об'єкти A, B, C. Кількість сполучень із повтореннями із цих об'єктів по 8, у яких кожний із них зустрічається не менше двох разів, дорівнює $C_{r-1-(k-1)n}^{r-kn} = C_{8-1-3}^{8-6} = C_4^2 = \dfrac{4!}{2!\cdot 2!} = 6$.

Випишемо ці сполучення:

$$[A, A, A, A, B, B, C, C], [A, A, B, B, B, B, C, C], [A, A, B, B, C, C, C, C],$$

$$[A, A, A, B, B, C, C, C], [A, A, B, B, B, C, C, C], [A, A, A, B, B, B, C, C].$$

Кількість сполучень із певними обмеженнями надалі позначатимемо як $\tilde{C}(n, r)$. Визначимо кількість сполучень із повтореннями з n об'єктів по r, у яких кожний об'єкт зустрічається парну кількість разів. Для цього розглянемо твірну функцію

$$(1 + x^2 + x^4 + x^6 + \ldots)^n = (1 - x^2)^{-n} = \sum_{r=0}^{\infty} C_{n+r-1}^{r} x^{2r}.$$

Отже, $\tilde{C}(n, 2r) = C_{n+r-1}^{r}$; $\tilde{C}(n, 2r+1) = 0$.

Якщо кожний об'єкт зустрічається в сполученні кількість разів, кратну k, потрібно використовувати таку твірну функцію:

$$(1 + x^k + x^{2k} + x^{3k} + \ldots)^n = (1 - x^k)^{-n} = \sum_{r=0}^{\infty} C_{n+r-1}^{r} x^{kr}.$$

Отже, $\tilde{C}(n, kr) = C_{n+r-1}^{r}$; $\tilde{C}(n, q) = 0$, якщо $q \neq kr$.

Розглянемо тепер умови загальнішого вигляду. Припустимо, що об'єкт S_1 зустрічається не менше k_1 разів, S_2 — не менше k_2 разів, ..., S_n — не менше k_n разів. Тоді, за аналогією із попереднім, з кожним об'єктом S_i пов'яжемо множник

$$(s_i^{k_i} x^{k_i} + s_i^{k_i+1} x^{k_i+1} + \ldots).$$

Тут коефіцієнт s_i відповідає об'єкту S_i. Для знаходження всіх сполучень із зазначеними обмеженнями розглянемо добуток

$$\prod_{i=1}^{n} (s_i^{k_i} x^{k_i} + s_i^{k_i+1} x^{k_i+1} + \ldots).$$

Щоб отримати твірну функцію для кількості сполучень із зазначеними обмеженнями, тепер достатньо в останньому виразі взяти $s_1 = s_2 = ... = s_n = 1$, що дасть

$$f(x) = \prod_{i=1}^{n} (x^{k_i} + x^{k_i+1} + ...).$$

Припустимо тепер, що об'єкт S_1 зустрічається не більше k_1 разів, S_2 — не більше k_2 разів, ..., S_n — не більше k_n разів. Тоді для знаходження всіх сполучень достатньо обчислити коефіцієнти розкладу

$$\prod_{i=1}^{n} (1 + s_i x + s_i^2 x^2 + ... + s_i^{k_i} x^{k_i}).$$

Щоб записати твірну функцію для сполучень із зазначеними обмеженнями, залишилось, як завжди, узяти $s_1 = s_2 = ... = s_n = 1$. Отже, твірна функція для сполучень із повтореннями, у яких об'єкт S_1 зустрічається не більше k_1 разів, S_2 — не більше k_2 разів, ..., S_n — не більше k_n разів, має вигляд

$$f(x) = \prod_{i=1}^{n} (1 + x + x^2 + ... + x^{k_i}).$$

Нехай тепер кількість поява об'єкта S_1 — одне з чисел $\alpha_1, \beta_1, ..., \lambda_1$, де $\alpha_1 < \beta_1 < ... < \lambda_1$; S_2 — $\alpha_2, \beta_2, ..., \lambda_2$, де $\alpha_2 < \beta_2 < ... < \lambda_2$; ...; S_n — одне з чисел $\alpha_n, \beta_n, ..., \lambda_n$, де $\alpha_n < \beta_n < ... < \lambda_n$. Тоді всі сполучення із зазначеними обмеженнями дають коефіцієнти розкладу

$$\prod_{i=1}^{n} (s_i^{\alpha_i} x^{\alpha_i} + s_i^{\beta_i} x^{\beta_i} + ... + s_i^{\lambda_i} x^{\lambda_i}),$$

а для запису твірної функції візьмемо $s_1 = s_2 = ... = s_n = 1$, що дасть

$$f(x) = \prod_{i=1}^{n} (x^{\alpha_i} + x^{\beta_i} + ... + x^{\lambda_i}).$$

Отже, для будь-якої „розумної" комбінації умов стосовно наявності об'єктів у сполученнях із повтореннями можна записати відповідний добуток і після цього, узявши $s_1 = s_2 = ... = s_n = 1$, одержати твірну функцію.

Приклад 2.33. Записати сполучення з повтореннями з трьох об'єктів S_1, S_2, S_3 по r за умови, що S_1 зустрічається не більше одного разу, S_2 — не більше двох, а S_3 — один або два рази. Записати твірну функцію для таких сполучень.

Запишемо добуток

$$(1 + s_1 x)(1 + s_2 x + s_2^2 x^2)(s_3 x + s_3^2 x^2) =$$

$$= s_3 x + (s_1 s_3 + s_2 s_3 + s_3 s_3) x^2 + (s_1 s_2 s_3 + s_2 s_2 s_3 + s_1 s_3 s_3 + s_2 s_3 s_3) x^3 +$$

$$+ (s_1 s_2 s_3 s_3 + s_2 s_2 s_3 s_3 + s_1 s_2 s_2 s_3) x^4 + (s_1 s_2 s_2 s_3 s_3) x^5.$$

Для можливих значень r маємо:

$r = 1$: $[S_3]$;

$r = 2$: $[S_1, S_3], [S_2, S_3], [S_3, S_3]$;

$r = 3$: $[S_1, S_2, S_3], [S_2, S_2, S_3], [S_1, S_3, S_3], [S_2, S_3, S_3]$;

$r = 4$: $[S_1, S_2, S_3, S_3], [S_2, S_2, S_3, S_3], [S_1, S_2, S_2, S_3]$;

$r = 5$: $[S_1, S_2, S_2, S_3, S_3]$.

Твірна функція —

$$f(x) = (1+x)(1+x+x^2)(x+x^2) = x + 3x^2 + 4x^3 + 3x^4 + x^5.$$

Отже, позначивши як $\tilde{C}(n, r)$ кількість сполучень із зазначеними обмеженнями, отримаємо $\tilde{C}(3, 1) = 1$, $\tilde{C}(3, 2) = 3$, $\tilde{C}(3, 3) = 4$, $\tilde{C}(3, 4) = 3$, $\tilde{C}(3, 5) = 1$.

Приклад 2.34. Розглянемо твірну функцію з прикладу 2.33:

$$f(x) = x + 3x^2 + 4x^3 + 3x^4 + x^5.$$

Коефіцієнт при x^r дорівнює кількості способів вибору r об'єктів із трьох, причому перший зустрічається не більше одного разу, другий — не більше двох, а третій — один або два рази, $1 \leqslant r \leqslant 5$. Цей коефіцієнт дорівнює кількості цілочислових розв'язків рівняння

$$e_1 + e_2 + e_3 = r$$

за умов $0 \leqslant e_1 \leqslant 1$, $0 \leqslant e_2 \leqslant 2$, $1 \leqslant e_3 \leqslant 2$.

Приклад 2.35. Розглянемо дві задачі. В урні чотири червоні, п'ять синіх і дві зелені кулі.

1. Скількома способами можна витягнути сім куль з урни?
2. Скількома способами можна витягнути з урни сім куль, якщо принаймні одна куля червона та принаймні дві сині?

Для задачі 1 твірна функція має вигляд

$$(1+x+x^2+x^3+x^4)(1+x+x^2+x^3+x^4+x^5)(1+x+x^2),$$

а кількість способів вибору з урни семи куль дорівнює коефіцієнту при x^7 у її розкладі. Для задачі 2 потрібно врахувати, що принаймні одна вийнята куля червона. Відповідний поліном має вигляд $(x + x^2 + x^3 + x^4)$, що відповідає витяганню однієї, двох, трьох або чотирьох червоних куль. Аналогічно, оскільки принаймні дві витягнуті кулі сині, то відповідний поліном має вигляд $(x^2 + x^3 + x^4 + x^5)$, що відповідає витяганню двох, трьох, чотирьох або п'яти синіх куль. Отже, твірний поліном має вигляд

$$(x+x^2+x^3+x^4)(x^2+x^3+x^4+x^5)(1+x+x^2),$$

а кількість способів витягти сім куль дорівнює коефіцієнту при x^7 у розкладі цієї твірної функції.

2.16.4. Твірні функції для розміщень

Розглянемо спочатку розміщення без повторень з n об'єктів по r. Маємо

$$\begin{aligned}(1+x)^n &= 1 + C_n^1 x + C_n^2 x^2 + C_n^3 x^3 + \ldots + C_n^n x^n = \\ &= 1 + A_n^1 \frac{x}{1!} + A_n^2 \frac{x^2}{2!} + A_n^3 \frac{x^3}{3!} + \ldots + A_n^n \frac{x^n}{n!}.\end{aligned} \tag{2.20}$$

Отже, у розкладі $(1+x)^n$ число A_n^r – коефіцієнт при $x^r/r!$. Це показує способи узагальнення формули (2.20). Якщо елемент S_i може з'явитись у розміщенні 0, 1, 2, ..., k_i разів, то множник $(1+x)$ у лівій частині (2.20) потрібно замінити множником [19]

$$1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^{k_i}}{k_i!}.$$

Приклад 2.36. Для кількості розміщень із повтореннями з n об'єктів по r одержимо

$$\left(1+x+\frac{x^2}{2!}+\ldots\right)^n=(e^x)^n=e^{nx}=1+nx+\frac{n^2x^2}{2!}+\ldots=\sum_{r=0}^{\infty}n^r\frac{x^r}{r!}.$$

Отже, кількість розміщень із повтореннями з n по r дорівнює n^r.

Уведені твірні функції називають експоненціальними: для послідовності чисел $a_0, a_1, \ldots, a_r, \ldots$ *експоненціальною твірною функцією* називають суму ряду

$$a_0+a_1x+a_2\frac{x^2}{2!}+\ldots+a_r\frac{x^r}{r!}+\ldots$$

за умови, що цей ряд збігається.

У прикладі 2.36 використано розклад

$$e^x=1+x+\frac{x^2}{2!}+\ldots+\frac{x^r}{r!}+\ldots,$$

який справджується для всіх x.

Щоб показати можливості експоненціальних твірних функцій, обчислимо кількість таких перестановок із повтореннями, що елемент S_1 зустрічається n_1 разів, елемент S_2 – n_2 разів, ..., елемент S_k – n_k разів, причому $n_1+n_2+\ldots+n_k=n$. На підставі наведених вище міркувань запишемо твірну функцію

$$\left(1+x+\frac{x^2}{2!}+\ldots+\frac{x^{n_1}}{n_1!}\right)\left(1+x+\frac{x^2}{2!}+\ldots+\frac{x^{n_2}}{n_2!}\right)\ldots\left(1+x+\frac{x^2}{2!}+\ldots+\frac{x^{n_k}}{n_k!}\right).$$

Коефіцієнт при $x^{n_1+n_2+\ldots+n_k}=x^n$ дорівнює $\dfrac{1}{n_1!n_2!\ldots n_k!}$, а коефіцієнт при $\dfrac{x^n}{n!}$ – $\dfrac{n!}{n_1!n_2!\ldots n_k!}$. Отже, ми іншим способом отримали відомий результат – формулу для перестановок із повтореннями.

Визначимо тепер кількість розміщень із повтореннями з n елементів по r таких, що кожний елемент зустрічається не менше одного разу. Твірна функція має вигляд

$$\left(x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots\right)^n=(e^x-1)^n=$$

$$=e^{nx}-ne^{(n-1)x}+\frac{n(n-1)}{2}e^{(n-2)x}+\ldots+(-1)^n. \qquad (2.21)$$

Приклад 2.37. Скористаємося формулою (2.21) і обчислимо кількість розміщень із повтореннями з трьох елементів по r, у яких кожний елемент зустрічається не менше одного разу:

$$(e^x-1)^3 = e^{3x}-3e^{2x}+3e^x-1 = \left(1+3x+\frac{(3x)^2}{2!}+\frac{(3x)^3}{3!}+\frac{(3x)^4}{4!}+\ldots\right)-$$

$$-3\left(1+2x+\frac{(2x)^2}{2!}+\frac{(2x)^3}{3!}+\frac{(2x)^4}{4!}+\ldots\right)+3\left(1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\ldots\right)-1=$$

$$=(3^3-3\cdot 2^3+3)\frac{x^3}{3!}+(3^4-3\cdot 2^4+3)\frac{x^4}{4!}+(3^5-3\cdot 2^5+3)\frac{x^5}{5!}+\ldots+(3^r-3\cdot 2^r+3)\frac{x^r}{r!}+\ldots.$$

Отже, кількість розміщень із повтореннями із трьох елементів по r, у яких кожний елемент зустрічається не менше одного разу, дорівнює $3^r-3\cdot 2^r+3$, де $r=3, 4, 5, 6, \ldots$

Приклад 2.38. Знайдемо кількість розміщень з n елементів по r, у яких кожний елемент зустрічається не більше двох разів. Очевидно, що твірна функція тут —

$$\left(1+x+\frac{x^2}{2!}\right)^n.$$

У разі $n=3$ маємо

$$\left(1+x+\frac{x^2}{2!}\right)^3 = 1+3x+\frac{9}{2}x^2+4x^3+\frac{9}{4}x^4+\frac{3}{4}x^5+\frac{1}{8}x^6 =$$

$$=1+3\frac{x}{1!}+\frac{9}{2}\cdot 2!\frac{x^2}{2!}+4\cdot 3!\frac{x^3}{3!}+\frac{9}{4}\cdot 4!\frac{x^4}{4!}+\frac{3}{4}\cdot 5!\frac{x^5}{5!}+\frac{1}{8}\cdot 6!\frac{x^6}{6!}.$$

У таблиці 2.3 для можливих значень r наведено кількості розміщень із повтореннями з трьох елементів по r, у яких кожний елемент зустрічається не більше двох разів.

Таблиця 2.3

Значення r	Кількість розміщень із зазначеними обмеженнями
1	3
2	$(9/2)\cdot 2! = 9$
3	$4\cdot 3! = 24$
4	$(9/4)\cdot 4! = 54$
5	$(3/4)\cdot 5! = 90$
6	$(1/8)\cdot 6! = 90$

Узагальнюючи розглянуті приклади, можна дійти такого висновку. Нехай розміщення з повтореннями з n об'єктів $S_1, S_2, \ldots, S_n$ по r такі, що об'єкт S_1 зустрічається $\alpha_1, \beta_1, \ldots, \lambda_1$ разів, де $\alpha_1<\beta_1<\ldots<\lambda_1$; об'єкт S_2 — $\alpha_2, \beta_2, \ldots, \lambda_2$ разів, де $\alpha_2<\beta_2<\ldots<\lambda_2$; …; S_n — $\alpha_n, \beta_n, \ldots, \lambda_n$ разів, де $\alpha_n<\beta_n<\ldots<\lambda_n$. Тоді твірна функція для таких розміщень має вигляд

$$\prod_{i=1}^{n}\left(\frac{x^{\alpha_i}}{\alpha_i!}+\frac{x^{\beta_i}}{\beta_i!}+\ldots+\frac{x^{\lambda_i}}{\lambda_i!}\right). \qquad (2.22)$$

Кількість розміщень, що задовольняють зазначеним обмеженням, дорівнює коефіцієнту при $x^r/r!$ у розкладі добутку (2.22).

2.16.5. Застосування твірних функцій до розв'язування рекурентних рівнянь

Нагадаємо, що

$$1 + ax + a^2x^2 + \ldots = \frac{1}{1-ax}, \tag{2.23}$$

якщо $|ax| < 1$.

На прикладах, поданих нижче, показано техніку застосування апарату твірних функцій до розв'язування рекурентних рівнянь [52].

Приклад 2.39. Розв'яжемо рекурентне рівняння $a_k = 3a_{k-1}$, $a_0 = 2$.

Нехай $G(x)$ — твірна функція для послідовності (a_k):

$$G(x) = \sum_{k=0}^{\infty} a_k x^k.$$

Помножимо обидві частини рівності на x:

$$xG(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k.$$

Використавши рекурентне рівняння, одержимо

$$G(x) - 3xG(x) = \sum_{k=0}^{\infty} a_k x^k - 3\sum_{k=1}^{\infty} a_{k-1} x^k =$$

$$= a_0 + \sum_{k=1}^{\infty} (a_k - 3a_{k-1}) x^k = 2,$$

бо $a_0 = 2$ та $a_k = 3a_{k-1}$.

Отже, $G(x) - 3xG(x) = 2$, звідки випливає, що $G(x) = 2/(1-3x)$. Використавши рівність (2.23), можемо записати

$$G(x) = \frac{2}{1-3x} = 2(1 + 3x + 3^2x^2 + \ldots + 3^k x^k + \ldots) =$$

$$= 2 + 2 \cdot 3x + 2 \cdot 3^2 x^2 + \ldots + 2 \cdot 3^k x^k + \ldots .$$

Тому $a_k = 2 \cdot 3^k$.

Приклад 2.40. Розв'яжемо неоднорідне рекурентне рівняння $a_n = 8a_{n-1} + 10^{n-1}$, $a_0 = 1$.

Запишемо твірну функцію для послідовності (a_n):

$$G(x) = \sum_{n=0}^{\infty} a_n x^n = 1 + \sum_{n=1}^{\infty} a_n x^n = 1 + \sum_{n=1}^{\infty} (8a_{n-1} x^n + 10^{n-1} x^n) =$$

$$= 1 + 8x\sum_{n=1}^{\infty} a_{n-1} x^{n-1} + x\sum_{n=1}^{\infty} 10^{n-1} x^{n-1} = 1 + 8x\sum_{n=0}^{\infty} a_n x^n + x\sum_{n=0}^{\infty} 10^n x^n =$$

$$= 1 + 8xG(x) + \frac{x}{1-10x}.$$

В останній рівності використано розклад (2.23). Отже,

$$G(x)=\frac{1-9x}{(1-8x)(1-10x)}. \tag{2.24}$$

Праву частину рівності (2.24) розкладемо на прості дроби:

$$G(x)=\frac{1}{2}\left(\frac{1}{1-8x}+\frac{1}{1-10x}\right).$$

Знову використаємо розклад (2.23):

$$G(x)=\frac{1}{2}\left(\sum_{n=0}^{\infty}8^n x^n+\sum_{n=0}^{\infty}10^n x^n\right)=\sum_{n=0}^{\infty}\frac{1}{2}\left(8^n+10^n\right)x^n.$$

Отже,

$$a_n=\frac{1}{2}(8^n+10^n).$$

Для розв'язання наступного прикладу потрібно розглянути композицію двох послідовностей (a_n) і (b_n). *Композицією* двох послідовностей (a_n) і (b_n) називають послідовність (c_n), загальний член якої має вигляд $c_n=a_0b_n+a_1b_{n-1}+\ldots+a_nb_0$.

ТЕОРЕМА 2.11. Твірна функція композиції двох послідовностей дорівнює добутку їх твірних функцій.

Доведення випливає з властивості 2 степеневих рядів (2.14).

Приклад 2.41. Розв'язати нелінійне рекурентне рівняння з п. 2.11 (задачу про багатокутник). Візьмемо $t_0=1$; тоді зазначене рекурентне рівняння набере вигляду

$$t_n=t_{n-1}t_0+t_{n-2}t_1+\ldots+t_0t_{n-1}, \tag{2.25}$$

де $n\geqslant 1$. Запишемо твірну функцію для послідовності (t_n), $n=0, 1, 2, \ldots$,

$$G(x)=\sum_{n=0}^{\infty}t_nx^n.$$

Помножимо обидві частини співвідношення (2.25) на x^n і просумуємо за n від 1 до ∞:

$$\sum_{n=1}^{\infty}t^nx^n=x\sum_{n=1}^{\infty}(t_{n-1}t_0+t_{n-2}t_1+\ldots+t_0t_{n-1})x^{n-1},$$

або

$$\sum_{n=0}^{\infty}t^nx^n-1=x\sum_{n=0}^{\infty}(t_0t_n+t_1t_{n-1}+\ldots+t_nt_0)x^n.$$

Згідно з теоремою 2.11 $G(x)-1=xG^2(x)$. Розв'яжемо квадратне рівняння щодо $G(x)$: $G(x)=(1\pm\sqrt{1-4x})/(2x)$. Оскільки $G(0)=1$, то потрібно взяти знак мінус (щоб знайти значення $G(x)$ у точці $x=0$, потрібно у виразі для $G(x)$ перейти до границі для $x\to 0$). Отже, твірна функція послідовності (t_n) це $G(x)=(1-\sqrt{1-4x})/(2x)$. За формулою (2.17) можемо записати:

$$\sqrt{1-4x}=1-2x-\frac{2}{2}C_2^1x^2-\frac{2}{3}C_4^2x^3-\ldots-\frac{2}{n+1}C_{2n}^nx^{n+1}-\ldots.$$

Отже,

$$G(x) = 1 + \frac{1}{2}C_2^1 x + \frac{1}{3}C_4^2 x^2 + \ldots + \frac{1}{n+1}C_{2n}^n x^n + \ldots .$$

Звідси випливає, що

$$t_n = \frac{1}{n+1}C_{2n}^n .$$

Контрольні запитання та завдання

1. Нехай $M = \{1, 2, 3, 4, 5\}$. Навести всі розміщення та сполучення без повторень з елементів множини M по 3 елементи.
2. Обчислити кількість перестановок множини $\{a, b, c, d, e, f, g\}$, які закінчуються буквою a.
3. Обчислити значення:
   а) A_5^3; б) A_6^5; в) A_8^1; г) A_8^5; д) A_8^8; є) A_{10}^9.
4. Обчислити значення:
   а) C_5^1; б) C_5^3; в) C_8^4; г) C_8^8; д) C_8^0; є) C_{12}^6.
5. Скількома способами можна визначити призові місця (перше, друге, третє) у забігу 12 коней?
6. У групі n чоловіків і n жінок. Скількома способами їх можна вишикувати в шеренгу так, щоб чергувалися чоловік і жінка?
7. Міста A та B з'єднано трьома різними дорогами. Скількома способами можна виконати коловий рейс від A до B та від B до A, якщо, їдучи від B до A, обов'язково треба вибирати нову дорогу?
8. Дано множину $M = \{1, 2, 3, \ldots, 99, 100\}$. Скільки існує розміщень без повторень з елементів множини M по чотири елементи, які містять:
   а) число 47;
   б) водночас числа 17 і 47;
   в) водночас числа 17, 47 і 73;
   г) водночас числа 17, 47, 73 і 97;
   д) три послідовні цілі числа у висхідному порядку?
9. Скількома способами можна розсадити шістьох осіб за круглим столом?
10. Скількома способами можна розсадити за круглим столом п'ятьох чоловіків і п'ятьох жінок, щоб двоє чоловіків не сиділи поруч?
11. Із цифр 1, 2, 3, 4, 5, не повторюючи їх, склали всі можливі п'ятицифрові числа. Скільки серед них таких чисел:
    а) які починаються цифрою 3;
    б) не починаються цифрою 5;
    в) починаються з 54?

12. Дано натуральні числа від 1 до 31. Скількома способами можна вибрати з них три числа так, щоб їх сума була парним числом?
13. Скількома способами можна поставити на полицю 10 книжок:
    а) якщо серед них є один тритомник, усі томи якого мають стояти поруч у довільному порядку;
    б) усі томи тритомника мають стояти поруч за зростанням номерів томів?
14. Скільки учасників у шаховому турнірі, якщо відомо, що кожний учасник зіграв із кожним із решти, а всього відбулося 210 партій?
15. Скількома способами з колоди 52 карт можна вийняти 10 карт, щоб серед них були такі:
    а) точно один туз;
    б) принаймні один туз;
    в) не менше двох тузів?
16. Скількома способами з 28 кісток доміно можна утворити пари кісток, які можна докласти одна до другої за правилами доміно?
17. Скількома способами можна вибрати пару однакових карт із колоди 36 карт?
18. Скількома способами можна вибрати пару з колоди 36 карт і одного джокера? (Джокер утворює пару з будь-якою картою.)
19. Скількома способами можна поселити дев'ять студентів у три кімнати гуртожитку, поселяючи їх по троє в кожній?
20. Скількома способами можна вибрати п'ять невпорядкованих елементів множини, що складається з трьох елементів, якщо повторення дозволені?
21. Скількома способами можна вибрати три невпорядкованих елементи множини, що складається з п'яти елементів, якщо повторення дозволені?
22. Скільки різних рядків із шести букв можна утворити з алфавіту, який має 26 букв, якщо повторення дозволені?
23. Знайти кількість розв'язків наведених нижче рівнянь у невід'ємних цілих числах.
    а) $x_1 + x_2 + x_3 = 15$;
    б) $x_1 + x_2 + x_3 + x_4 = 17$;
    в) $x_1 + x_2 + x_3 = 15$ за умов $x_1 \geqslant 2$, $x_2 \geqslant 3$, $x_3 \geqslant 5$.
24. Знайти кількість розв'язків рівняння $x_1 + x_2 + x_3 + x_4 + x_5 = 17$, де x_1, x_2, x_3, x_4, x_5 — такі невід'ємні цілі числа:
    а) $x_1 \geqslant 1$;
    б) $x_j \geqslant 2$ для $j = 1, 2, 3, 4, 5$;
    в) $0 \leqslant x_1 \leqslant 10$;
    г) $0 \leqslant x_1 \leqslant 3$, $1 \leqslant x_2 < 4$, $x_3 \geqslant 15$.
25. Знайти кількість розв'язків нерівності $x_1 + x_2 + x_3 \leqslant 15$ у невід'ємних цілих числах.

26. Знайти кількість таких додатних цілих чисел, менших за 1 000 000, що сума їх цифр дорівнює 19.
27. Знайти кількість додатних цілих чисел, менших за 1 000 000, що мають точно одну цифру 9, і сума всіх їх цифр дорівнює 13.
28. Скільки різних рядків можна утворити зі слова MISSISSIPPI, використовуючи всі букви? Скільки таких рядків починаються та закінчуються літерою S? У скількох таких рядках усі чотири букви S стоять поруч?
29. Обчислити кількість бітових рядків довжиною n. Користуючись цим результатом, довести, що кількість підмножин множини з n елементів дорівнює 2^n.
30. Множина містить 100 елементів. Знайти кількість підмножин цієї множини, що містять більше одного елемента.
31. Скільки бітових рядків можна утворити з шести одиниць і восьми нулів?
32. Скільки бітових рядків, які складаються з чотирьох одиниць і 12 нулів, можна утворити, якщо кожний рядок обов'язково має починатися з одиниці та після кожної одиниці має бути принаймні два нулі?
33. Побудувати розклад:
    а) $(x+y)^5$; б) $(x-y)^5$; в) $(x+y)^6$; г) $(x-y)^6$.
34. Визначити коефіцієнт:
    а) при x^5y^8 у розкладі $(x-y)^{13}$;
    б) при $x^{14}y^{11}$ у розкладі $(x-y)^{25}$.
35. Скільки членів у розкладі $(x+y)^{100}$?

У задачах 38–44 члени бінома пронумеровано від 1 до $n+1$:

$$(x \pm y)^n = \sum_{j=0}^{n} T_{j+1}, \quad \text{де} \quad T_{j+1} = (\pm 1)^j C_n^j x^{n-j} y^j.$$

36. Визначити п'ятий член розкладу бінома $\left(\frac{a}{\sqrt{x}} + \frac{\sqrt{x}}{a}\right)^n$, якщо відношення коефіцієнта третього члена до коефіцієнта другого члена дорівнює 11/2.
37. У розкладі бінома $(\sqrt{1+x} - \sqrt{1-x})^n$ коефіцієнт третього члена дорівнює 28. Визначити середній член розкладу.
38. Визначити найменше значення показника n у розкладі $(1+x)^n$, за якого відношення двох сусідніх коефіцієнтів дорівнює 7/15.
39. У розкладі бінома $\left(\sqrt[3]{a} + \sqrt{a^{-1}}\right)^{15}$ визначити член, який не залежить від a.
40. Скільки раціональних членів міститься в розкладі $\left(\sqrt{2} + \sqrt[4]{3}\right)^{100}$?
41. У розкладі бінома $\left(a\sqrt[5]{a/3} - b/\sqrt[7]{a^3}\right)^n$ визначити член, що містить a^3, якщо сума біноміальних коефіцієнтів на непарних місцях у розкладі дорівнює 2048.
42. За якого значення n коефіцієнти другого, третього та четвертого членів розкладу бінома $(x+y)^n$ утворюють арифметичну прогресію?
43. Довести тотожність Паскаля $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$ на основі алгебраїчних перетворень.

44. Нехай M — скінченна множина. Довести, що підмножин множини M із парною кількістю елементів стільки, скільки й підмножин із непарною кількістю елементів.
45. Довести, що $(C_n^0)^2 + (C_n^1)^2 + \ldots + (C_n^k)^k + \ldots + (C_n^n)^2 = C_{2n}^n$.
46. Довести біноміальну теорему алгебрично за допомогою математичної індукції.
47. Довести, що $C_n^r = P_n(r, n-r)$.
48. Записати розклад $(x+y+z)^4$.
49. Знайти коефіцієнт при $x^3y^2z^5$ у розкладі $(x+y+z)^{10}$.
50. Знайти кількість членів (доданків) у розкладі $(x_1 + x_2 + \ldots + x_k)^n$.
51. Знайти лексикографічно наступну перестановку для кожної з перестановок: 1432; 54123; 12453; 45231; 6714235; 31528764.
52. Розмістити наведені перестановки елементів множини {1, 2, 3, 4, 5, 6} у лексикографічному порядку: 234561, 231456, 165432, 156423, 543216, 541236, 231465, 314562, 432561, 654321, 654312, 435612.
53. За допомогою алгоритму побудови лексикографічно наступної перестановки записати перші 18 перестановок елементів множини {1, 2, 3, 4, 5, 6}.
54. Задати взаємнооднозначну відповідність між елементами множин $M = \{a, b, c, d, e\}$ й $X = \{1, 2, 3, 4, 5\}$. Побудувати перші 18 перестановок елементів множини M у лексикографічному порядку.
55. За допомогою алгоритму побудови лексикографічно наступного сполучення виписати всі сполучення по чотири елементи множини {1, 2, 3, 4, 5, 6}.
56. Задати взаємнооднозначну відповідність між елементами множин $M = \{x, y, z, t, u\}$ та $X = \{1, 2, 3, 4, 5\}$. За допомогою алгоритму виписати всі сполучення по три елементи множини M.
57. Описати алгоритм побудови розміщень по r елементів множини з n елементів. За його допомогою виписати всі розміщення по два елементи множини {1, 2, 3, 4, 5}.
58. Описати алгоритм побудови всіх розміщень по r елементів множини з n елементів, якщо повторення дозволені.
59. Описати алгоритм побудови всіх сполучень по r елементів множини з n елементів, якщо повторення дозволені.
60. Описати алгоритм побудови списку всіх розбиттів множини на непорожні частини. Виписати всі можливі розбиття множини $\{a, b, c, d\}$. Скільки їх?
61. Розв'язати рекурентні наведені нижче рівняння із заданими початковими умовами:

а) $a_n = a_{n-1} + 6a_{n-2}$, $n \geqslant 2$, $a_0 = 3$, $a_1 = 6$;
б) $a_n = 7a_{n-1} - 10a_{n-2}$, $n \geqslant 2$, $a_0 = 2$, $a_1 = 1$;
в) $a_n = 6a_{n-1} - 8a_{n-2}$, $n \geqslant 2$, $a_0 = 4$, $a_1 = 10$;
г) $a_n = 2a_{n-1} - a_{n-2}$, $n \geqslant 2$, $a_0 = 45$, $a_1 = 1$;
д) $a_n = a_{n-2}$, $n \geqslant 2$, $a_0 = 5$, $a_1 = -1$;

е) $a_n = -8a_{n-1} - 16a_{n-2}$, $n \geqslant 2$, $a_0 = 4$, $a_1 = -4$;

ж) $a_n = -7a_{n-1} - 16a_{n-2} - 12a_{n-3}$, $n \geqslant 3$, $a_0 = 2$, $a_1 = 9$, $a_2 = 29$.

62. Дано неоднорідне рекурентне рівняння $a_n = 3a_{n-1} + 2^n$. Показати, що $a_n = -2^{n+1}$ — його розв'язок. Знайти загальний розв'язок цього рекурентного рівняння. Знайти розв'язок за початкової умови $a_0 = 1$.

63. Дано неоднорідне рекурентне рівняння $a_n = 2a_{n-1} + n + 5$. Визначити такі константи s і t, що $a_n = sn + t$ — його розв'язок. Знайти загальний розв'язок цього рекурентного рівняння та розв'язок за початкової умови $a_0 = 4$.

64. Скільки потрібно запросити людей, аби щонайменше шість із них народилися під одним і тим самим знаком зодіаку?

65. Скільки має бути людей, щоб обов'язково принаймні двоє з них народилися в один і той самий день тижня та в один і той самий місяць (можливо, у різні роки)?

66. Позначимо як M множину з десяти натуральних чисел, які не перевищують 50. Довести, що є принаймні дві різні п'ятиелементні підмножини множини M такі, що суми їх елементів рівні.

67. Скільки елементів містить об'єднання п'яти множин, якщо кожна з них містить 10 000 елементів, кожна пара — 1000 спільних елементів, кожна трійка — 100, кожна четвірка — 10 спільних елементів і один елемент належить усім п'яти множинам?

68. За допомогою принципу включення-виключення в альтернативній формі визначити кількість простих чисел, що не перевищують 100.

70. Скільки розв'язків має рівняння $x_1 + x_2 + x_3 = 13$, якщо x_1, x_2, x_3 — невід'ємні цілі числа, менші, ніж 6?

71. Знайти кількість розв'язків рівняння $x_1 + x_2 + x_3 + x_4 = 17$, якщо x_1, x_2, x_3, x_4 — невід'ємні цілі числа такі, що $x_1 \leqslant 3$, $x_2 \leqslant 4$, $x_3 \leqslant 5$, $x_4 \leqslant 8$.

72. Знайти твірні функції для сполучень із повтореннями, у яких кожний об'єкт зустрічається:

а) не менше двох разів;

б) не більше чотирьох разів;

в) не менше одного й не більше п'яти разів;

г) кількість разів, кратну трьом.

73. Записати твірні функції для розміщень з n елементів по r із повтореннями, у яких кожний елемент зустрічається:

а) не менше двох разів;

б) точно два рази;

в) не більше двох разів;

г) парну кількість разів;

д) непарну кількість разів.

74. Методом твірних функцій розв'язати однорідні рекурентні рівняння:

а) $a_n = 7a_{n-1},\ a_0 = 5$;

б) $a_n = 5a_{n-1} - 6a_{n-2},\ a_0 = 6,\ a_1 = 30$.

75. Методом твірних функцій розв'язати неоднорідні рекурентні рівняння:

а) $a_n = 3a_{n-1} + 2,\ a_0 = 1$;

б) $a_n = 3a_{n-1} + 4^{n-1},\ a_0 = 1$.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано натуральне число n. Навести в лексикографічному порядку всі перестановки елементів множини $\{1, 2, ..., n\}$.
2. Задано натуральне число n і невід'ємне ціле число r ($r \leqslant n$). Навести в лексикографічному порядку всі r-сполучення без повторень з елементів множини $\{1, 2, ..., n\}$.
3. Задано натуральне число n і невід'ємне ціле число r ($r \leqslant n$). Навести в лексикографічному порядку всі r-розміщення без повторень з елементів множини $\{1, 2, ..., n\}$.
4. Задано натуральне число n. Навести в лексикографічному порядку всі сполучення без повторень з елементів множини $\{1, 2, ..., n\}$.
5. Задано натуральні числа n і r. Навести в лексикографічному порядку всі r-розміщення з повтореннями з елементів множини $\{1, 2, ..., n\}$.
6. Задано натуральні числа n і r. Навести в лексикографічному порядку всі r-сполучення з повтореннями з елементів множини $\{1, 2, ..., n\}$.
7. Задано натуральне число n. Навести всі перестановки елементів множини $\{1, 2, ..., n\}$, у яких жодний елемент не залишається на місці.
8. Задано рівняння $x_1 + x_2 + ... + x_n = C$, де C — ціла невід'ємна константа. Знайти всі розв'язки цього рівняння в невід'ємних цілих числах.

Розділ 3
Теорія графів

- Основні означення та властивості
- Деякі спеціальні класи простих графів
- Способи подання графів
- Шляхи та цикли. Зв'язність
- Ізоморфізм графів
- Ейлерів цикл у графі
- Гамільтонів цикл у графі
- Зважені графи й алгоритми пошуку найкоротших шляхів
- Обхід графів
- Планарні графи
- Розфарбовування графів
- Незалежні множини вершин. Кліки
- Паросполучення в графах. Теорема Холла
- Найбільше паросполучення у дводольних графах

Теорія графів — одна з істотних частин математичного апарату інформатики та кібернетики. У термінах теорії графів можна сформулювати багато задач, пов'язаних із дискретними об'єктами. Такі задачі виникають у проектуванні інтегральних схем і схем управління, у дослідженні автоматів, в економіці й статистиці, теорії розкладів і дискретній оптимізації.

3.1. Основні означення та властивості

Термін „граф" уперше з'явився в книзі видатного угорського математика Д. Кеніга 1936 р., хоча перші задачі теорії графів пов'язані ще з іменем Л. Ейлера (XVIII ст.).

Простим графом називають пару $G=(V, E)$, де V — непорожня скінченна множина елементів, називаних *вершинами*, E — множина невпорядкованих пар різних елементів із V. Елементи множини E (невпорядковані пари різних вершин) називають *ребрами*.

Приклад 3.1. На рис. 3.1 зображено простий граф G з множиною вершин $V=\{v_1, v_2, v_3, v_4\}$ і множиною ребер $E=\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}\}$.

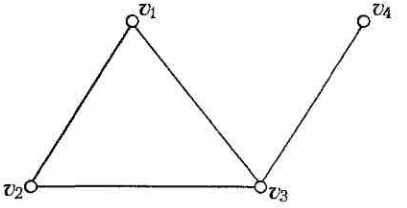


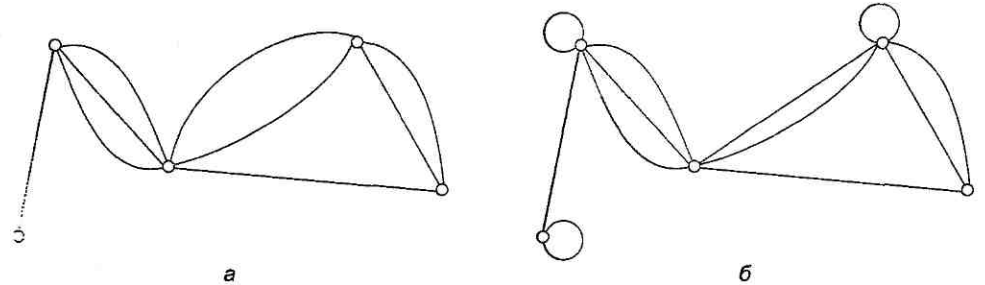
Рис. 3.1

Говорять, що ребро $\{u, v\}$ *з'єднує* вершини u та v. Оскільки E — множина, то в простому графі пару вершин може з'єднувати не більше ніж одне ребро.

Іноді розглядають графи, у яких дві вершини можуть бути з'єднані більше ніж одним ребром. Так виникає поняття мультиграфа. *Мультиграфом* називають пару (V, E), де V — скінченна непорожня множина вершин, а E — *сім'я* невпорядкованих пар різних елементів множини V. Тут застосовано термін „сім'я" замість поняття „множина", бо елементи в E (ребра) можуть повторюватись. Ребра, що з'єднують одну й ту саму пару вершин, називають *кратними* (або *паралельними*).

Окрім кратних ребер розглядають також *петлі*, тобто ребра, які з'єднують вершину саму із собою. *Псевдографом* називають пару (V, E), де V — скінченна непорожня множина вершин, а E — сім'я невпорядкованих пар не обов'язково різних вершин.

Приклад 3.2. На рис. 3.2 зображено мультиграф (рис. 3.2, *а*) і псевдограф (рис. 3.2, *б*).



а *б*

Рис. 3.2

Розглянуті три типи графів називають *неорієнтованими*. Псевдограф — це найзагальніший тип неорієнтованого графа, бо він може містити петлі й кратні ребра. Мультиграф — це неорієнтований граф, який може містити кратні ребра, але не може містити петель. Нарешті, простий граф — це неорієнтований граф без кратних ребер і без петель.

Розглядають також орієнтовані графи. *Орієнтованим графом* називають пару (V, E), де V — скінченна непорожня множина вершин, а E — множина впорядкованих пар елементів множини V. Елементи множини E в орієнтованому графі називають *дугами* (чи *орієнтованими ребрами*). Дугу (v, v) називають *петлею*.

Приклад 3.3. На рис. 3.3 зображено орієнтований граф із множиною вершин $V=\{v_1, v_2, v_3, v_4, v_5\}$ і множиною дуг $E=\{(v_2, v_1), (v_2, v_3), (v_3, v_2), (v_3, v_4), (v_4, v_3), (v_4, v_5), (v_5, v_5)\}$.

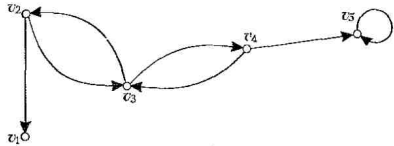


Рис. 3.3

Зазначимо, що дуга — це впорядкована пара вершин (записують у круглих дужках), тому в графі на рис. 3.3 дуги (v_2, v_3) та (v_3, v_2) різні. На рисунках дуги позначають стрілками.

Орієнтованим мультиграфом називають пару (V, E), де V — скінченна непорожня множина вершин, а E — сім'я впорядкованих пар елементів множини V.

Отже, елементи (дуги) в E в разі орієнтованого мультиграфа можуть повторюватись; такі дуги називають *кратними*. Зауважимо, що кратні дуги з'єднують одну пару вершин і однаково напрямлені.

Приклад 3.4. На рис. 3.4 наведено приклад орієнтованого мультиграфа. Дуги e_2 та e_3 — кратні, а дуги e_5, e_6 — ні.

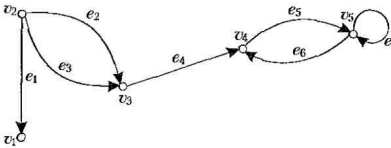


Рис. 3.4

Надалі ми будемо використовувати термін „граф" для опису довільних графів — орієнтованих і неорієнтованих, із петлями та кратними ребрами чи без них, а термін „неорієнтований граф" або „псевдограф" — для довільного неорієнтованого графа, який може мати кратні ребра й петлі [52]. Означення різних типів графів зведено в табл. 3.1.

Таблиця 3.1

Тип графа	Ребра	Чи дозволені *кратні ребра*?	Чи дозволені *петлі*?
Простий граф	Неорієнтовані	Ні	Ні
Мультиграф	Неорієнтовані	Так	Ні
Псевдограф	Неорієнтовані	Так	Так
Орієнтований граф	Орієнтовані (дуги)	Ні	Так
Орієнтований мультиграф	Орієнтовані (дуги)	Так	Так

Дві вершини u та v в неорієнтованому графі G називають *суміжними*, якщо $\{u, v\}$ — ребро: $\{u, v\} \in E$. Якщо $e = \{u, v\}$ — ребро, то вершини u та v називають його *кінцями*. Два ребра називають *суміжними*, якщо вони мають спільний кінець. Вершину v та ребро e називають *інцидентними*, якщо вершина v — кінець ребра e. Зазначимо, що суміжність — це зв'язок між однорідними елементами графа, а інцидентність — зв'язок між його різнорідними елементами.

Степінь вершини в неорієнтованому графі — це кількість інцидентних їй ребер, причому петлю враховують двічі. Степінь вершини v позначають $\deg(v)$. Якщо $\deg(v) = 0$, то вершину v називають *ізольованою*; якщо $\deg(v) = 1$ — *висячою*, або *кінцевою*.

Приклад 3.5. У неорієнтованому графі на рис. 3.5 степені вершин такі: $\deg(v_1) = 4$, $\deg(v_2) = 4$, $\deg(v_3) = 6$, $\deg(v_4) = 1$, $\deg(v_5) = 3$, $\deg(v_6) = 0$. Отже, вершина v_6 — ізольована, а v_4 — висяча.

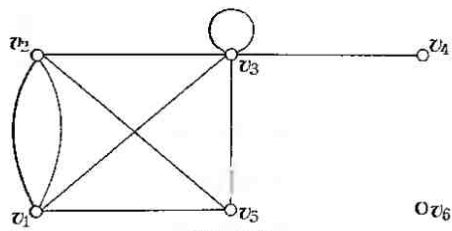


Рис. 3.5

Зв'язок між степенями вершин неорієнтованого графа та кількістю його ребер дає така теорема.

ТЕОРЕМА 3.1. Нехай $G = (V, E)$ — неорієнтований граф з m ребрами. Тоді

$$\sum_{v \in V} \deg(v) = 2m.$$

Це твердження стосується будь-якого неорієнтованого графа, зокрема з петлями та кратними ребрами.

Доведення. Додавання кожного нового ребра додає по одиниці до степенів двох вершин або двійку до степеня однієї вершини в разі петлі.

ТЕОРЕМА 3.2. Неорієнтований граф має парну кількість вершин непарного степеня.

Доведення. Позначимо як V_1 множину вершин парного степеня, як V_2 — непарного, а як m — кількість ребер графа. Тоді $2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v)$. Отже, $\sum_{v \in V_2} \deg(v)$ — парне число. Сума непарних чисел парна тоді й лише тоді, коли кількість доданків парна.

Якщо в орієнтованому мультиграфі $G = (V, E)$, $(u, v) \in E$, то вершину u називають *початковою* (*ініціальною*), а вершину v — *кінцевою* (*термінальною*) вершиною дуги $e = (u, v)$. Петля має початок і кінець в одній і тій самій вершині.

Для орієнтованого графа означення степеня вершини інше. В орієнтованому мультиграфі *напівстепенем входу* вершини v називають кількість дуг, для яких вершина v кінцева; позначають $\deg^-(v)$. *Напівстепенем виходу* вершини v називають кількість дуг, для яких вершина v початкова; позначають $\deg^+(v)$.

Приклад 3.6. Для графа, зображеного на рис. 3.6, напівстепені вершин такі:

$$\deg^-(v_1)=0,\ \deg^+(v_1)=1,\ \deg^-(v_2)=2,\ \deg^+(v_2)=0,\ \deg^-(v_3)=1,\ \deg^+(v_3)=2.$$

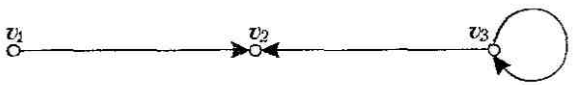


Рис. 3.6

ТЕОРЕМА 3.3. Нехай $G=(V, E)$ — орієнтований мультиграф, який має m дуг. Тоді

$$\sum_{v\in V}\deg^-(v)=\sum_{v\in V}\deg^+(v)=m.$$

Простий граф $H=(W, F)$ називають *підграфом* простого графа $G=(V, E)$, якщо $W\subset V$, $F\subset E$. Підграф H називають *каркасним* (або *фактором*), якщо $W=V$. Якщо $W\neq V$, а F — множина всіх ребер з E, які мають кінці в W, то підграф H називають *породженим* (або *індукованим*) *множиною* W і позначають як $G(W)$.

Приклад 3.7. На рис. 3.7 зображено граф G та три його підграфа H_1, H_2, H_3, серед яких H_2 породжений, а H_3 — каркасний.

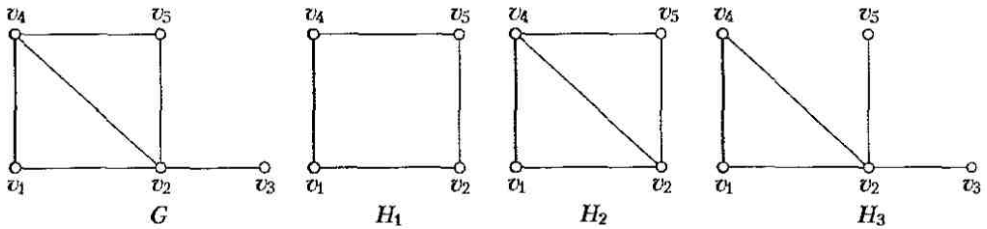


Рис. 3.7

Об'єднанням двох простих графів $G_1=(V_1, E_1)$ та $G_2=(V_2, E_2)$ називають такий простий граф $G=(V, E)$, що $V=V_1\cup V_2$, $E=E_1\cup E_2$.

Приклад 3.8. На рис. 3.8 зображено приклад об'єднання двох простих графів. Знаки „$\cup$" та „$=$" тут мають символічний зміст.

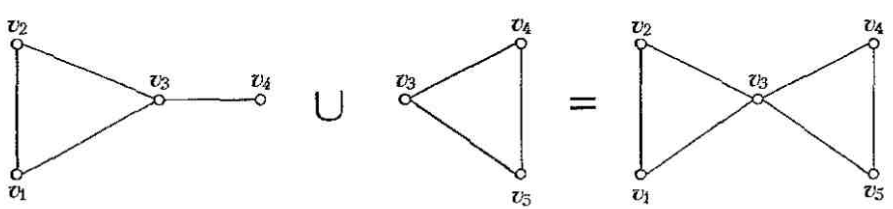


Рис. 3.8

Надалі значну увагу буде приділено алгоритмам на графах, тому дамо деякі означення, що стосуються алгоритмів. Під час аналізу алгоритму нас буде цікавити передусім його складність, під якою розуміємо час виконання відповідної програми на комп'ютері. Зрозуміло, що цей показник суттєво залежить від типу комп'ютера. Щоб висновки про складність були достатньо універсальними, будемо вважати, що всі обчислення виконуються на якомусь ідеалізованому комп'ютері. Цим питанням повністю присвячено розділ 10, а тут лише коротко окреслимо суть справи.

Означимо *складність* алгоритму розв'язування задачі як функцію f, яка кожному невід'ємному цілому числу n ставить у відповідність час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжиною n. Час роботи алгоритму вимірюють у кроках (операціях), виконуваних на ідеалізованому комп'ютері.

Наприклад, якщо алгоритм приймає як вхідні дані довільний граф $G=(V, E)$, то під довжиною входу можна розуміти $|V|$ чи $\max\{|V|, |E|\}$.

Аналіз ефективності алгоритмів полягає в з'ясуванні того, як швидко зростає функція $f(n)$ зі збільшенням n. Для порівняння швидкості зростання двох функцій $f(n)$ і $g(n)$ (з невід'ємними значеннями) використовують такі позначення:

- $f(n)=O(g(n))$ означає, що існують додатна стала c та натуральне число n_0 такі, що $f(n) \leqslant cg(n)$ для всіх $n \geqslant n_0$;
- $f(n)=\Omega(g(n))$ означає, що існують додатна стала c та натуральне число n_0 такі, що $f(n) \geqslant cg(n)$ для всіх $n \geqslant n_0$.

Для аналізу ефективності алгоритмів використовують O-символіку. Вираз „складність алгоритму дорівнює (становить) $O(g(n))$" має саме такий зміст, як у наведеному вище означенні. Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму не залежить від довжини входу.

Алгоритм зі складністю $O(n)$, де n — довжина входу, називають *лінійним*. Такий алгоритм для переважної більшості задач найліпший (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ — поліном, називають *поліноміальним*. Часто замість $O(p(n))$ пишуть $O(n^a)$, де a — константа. Особливу роль поліноміальних алгоритмів описано в розділі 10. Тут лише зазначимо, що всі задачі дискретної математики, які вважають важкими для алгоритмічного розв'язування, нині не мають поліноміальних алгоритмів. Крім того, поняття „поліноміальний алгоритм" — це найпоширеніша формалізація поняття „ефективний алгоритм".

Алгоритми, часова складність яких не піддається подібній оцінці, називають *експоненціальними*. Більшість експоненціальних алгоритмів — це просто варіанти повного перебору, а поліноміальні алгоритми здебільшого можна побудувати лише тоді, коли вдається заглибитись у суть задачі. Задачу називають *важкорозв'язною*, якщо для її розв'язання не існує поліноміального алгоритму.

3.2. Деякі спеціальні класи простих графів

Розглянемо деякі спеціальні класи простих графів, часто використовувані як приклади й широко застосовувані [15].

Повний граф з n вершинами (позначають як K_n) — це граф, у якому будь-яку пару вершин з'єднано точно одним ребром.

Кількість ребер у графі K_n дорівнює $C_n^2 = \frac{n(n-1)}{2}$.

Приклад 3.9. На рис. 3.9 зображено графи K_n для $n = 1, 2, 3, 4, 5$.

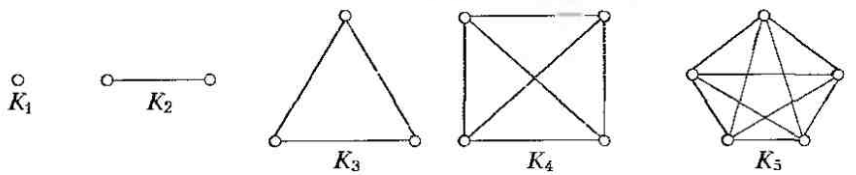


Рис. 3.9

Граф називають *порожнім*, якщо $E = \varnothing$, тобто такий граф не має ребер. Порожній граф з n вершинами позначають як O_n.

Граф $G = (V, E)$ називають *дводольним*, якщо V можна розбити на дві підмножини V_1 і V_2, що не перетинаються ($V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$), так, що кожне ребро з'єднує вершину з V_1 і вершину з V_2. Дводольний граф називають *повним*, якщо кожну вершину з V_1 з'єднано ребром із кожною вершиною з V_2. Повний дводольний граф позначають як $K_{m,n}$, де $m = |V_1|$, $n = |V_2|$. Граф $K_{1,n}$ називають *зіркою*.

Граф $K_{m,n}$ має $n + m$ вершин та mn ребер.

Приклад 3.10. На рис. 3.10 наведено повні дводольні графи $K_{1,5}$, $K_{3,2}$ та $K_{3,3}$.

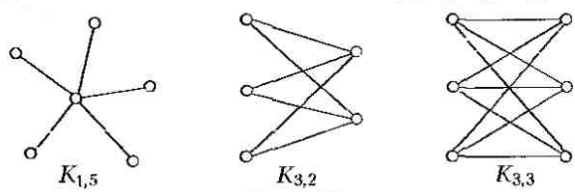


Рис. 3.10

Циклом C_n, $n \geqslant 3$, називають граф із множиною вершин $V = \{v_1, v_2, ..., v_n\}$ і множиною ребер $E = \{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$.

Приклад 3.11. На рис. 3.11 зображено цикли C_3, C_4, C_5 і C_6.

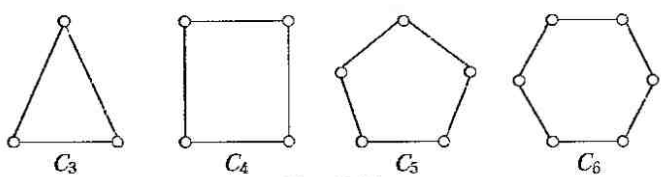


Рис. 3.11

Колесом W_n називається граф, який можна одержати з циклу C_n додаванням іще однієї вершини, з'єднаної з усіма n вершинами в C_n новими ребрами.

Приклад **3.12.** На рис. 3.12 зображено колеса W_3, W_4, W_5, W_6.

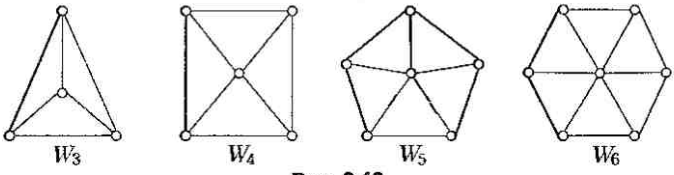


Рис. 3.12

Простий граф, вершини якого зображають усі 2^n бітові рядки довжиною n, називають *n-вимірним кубом* і позначають Q_n. Дві вершини в Q_n з'єднано ребром тоді і лише тоді, коли бітові рядки, які їх подають, відрізняються точно в одному біті.

Приклад **3.13.** На рис. 3.13 зображено графи Q_1, Q_2 та Q_3.

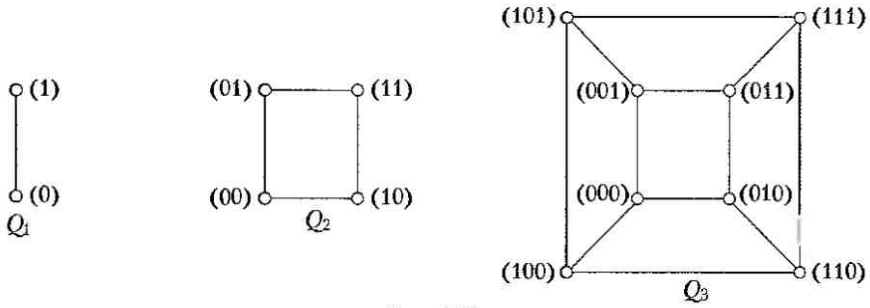


Рис. 3.13

Граф Q_{n+1} можна отримати з двох графів Q_n, з'єднавши ребрами їхні однаково позначені вершини. Після цього до бітових рядків у вершинах одного з графів зліва дописують 0, другого — 1.

3.3. Способи подання графів

Найзрозуміліший і корисний для людини спосіб подання (зображення) графів — це рисунок на площині у вигляді точок і ліній, які з'єднують ці точки. Проте цей спосіб подання абсолютно непридатний, якщо потрібно розв'язувати на комп'ютері задачі з графами.

Розглянемо декілька інших способів подання графів для двох найважливіших типів графів: простого (рис. 3.14) й орієнтованого (рис. 3.15).

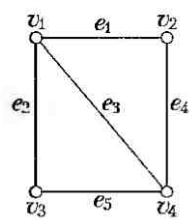


Рис. 3.14

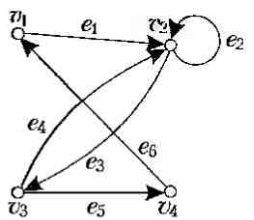


Рис. 3.15

Матрицю, кожний елемент якої дорівнює 0 чи 1, називають *булевою*.

3.3.1. Матриця інцидентності

Нехай $G=(V, E)$ — простий граф із множиною вершин $V=\{v_1, v_2, ..., v_n\}$ і множиною ребер $E=\{e_1, e_2, ..., e_m\}$.

Матрицею інцидентності графа G, яка відповідає заданій нумерації вершин і ребер, називають булеву $n\times m$ матрицю M з елементами m_{ij} ($i=1, ..., n; j=1, ..., m$), де

$$m_{ij}=\begin{cases}1, & \text{якщо вершина } v_i \text{ та ребро } e_j \text{ інцидентні,}\\ 0 & \text{у протилежному випадку.}\end{cases}$$

Приклад 3.14. Для графа, зображеного на рис. 3.14, матриця інцидентності має вигляд

$$\begin{array}{c} \\ v_1\\ v_2\\ v_3\\ v_4\end{array}\begin{array}{c}\begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5\end{array}\\ \begin{bmatrix}1&1&1&0&0\\1&0&0&1&0\\0&1&0&0&1\\0&0&1&1&1\end{bmatrix}\end{array}.$$

Отже, для простого графа в матриці інцидентності в кожному стовпці точно дві одиниці, і немає однакових стовпців. Матрицю інцидентності можна використовувати й для подання мультиграфа. Тоді з'являться однакові стовпці (вони відповідають кратним ребрам). Для подання псевдографа, якщо в ньому є петлі, у відповідних позиціях матриці ставимо 2 (у цьому разі матриця інцидентності не булева).

За допомогою матриці інцидентності можна подавати й орієнтовані графи. Для таких графів вона також не булева. Нехай $G=(V, E)$ — орієнтований граф із множиною вершин $V=\{v_1, v_2, ..., v_n\}$ і множиною дуг $E=\{e_1, e_2, ..., e_m\}$.

Матрицею інцидентності орієнтованого графа G, яка відповідає заданій нумерації вершин і дуг, називають $n\times m$ матрицю M з елементами m_{ij} ($i=1, ..., n; j=1, ..., m$). де

$$m_{ij}=\begin{cases}1, & \text{якщо дуга } e_j \text{ виходить з вершини } v_i,\\ -1, & \text{якщо дуга } e_j \text{ входить у вершину } v_i,\\ 2, & \text{якщо дуга } e_j \text{ — це петля у вершині } v_i,\\ 0 & \text{в інших випадках.}\end{cases}$$

Приклад 3.15. Для графа, зображеного на рис. 3.15, матриця інцидентності має вигляд

$$\begin{array}{c} \\ v_1\\ v_2\\ v_3\\ v_4\end{array}\begin{array}{c}\begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6\end{array}\\ \begin{bmatrix}1&0&0&0&0&-1\\-1&2&1&-1&0&0\\0&0&-1&1&1&0\\0&0&0&0&-1&1\end{bmatrix}\end{array}.$$

З алгоритмічного погляду матриця інцидентності — це, мабуть, найгірший спосіб подання графа, який тільки можна собі уявити [23]. По-перше, для цього

потрібно nm комірок пам'яті, більшість із яких зайняті нулями. По-друге, доступ до інформації незручний. Щоб отримати відповідь на елементарні питання (наприклад, чи існує дуга (v_i, v_j), до яких вершин ведуть дуги з v_i), у найгіршому випадку потрібно перебрати всі стовпці матриці, тобто виконати n кроків.

3.3.2. Матриця суміжності

Нехай $G = (V, E)$ — простий граф, $|V| = n$. Припустимо, що вершини графа G занумеровані: $v_1, v_2, \ldots, v_n$. *Матрицею суміжності графа G* (яка відповідає даній нумерації вершин) називають булеву $n \times n$ матрицю A з елементами a_{ij} $(i, j = 1, \ldots, n)$, де

$$a_{ij} = \begin{cases} 1, & \text{якщо } \{v_i, v_j\} \in E, \\ 0 & \text{в протилежному випадку.} \end{cases}$$

Приклад 3.16. Матриця суміжності для графа, зображеного на рис. 3.14, має вигляд

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}.$$

Цілком очевидно, що для неорієнтованого графа $a_{ij} = a_{ji}$, тобто матриця суміжності симетрична. Більше того, позаяк у простому графі немає петель, то для нього в матриці суміжності $a_{ii} = 0$ $(i = 1, \ldots, n)$.

Матрицю суміжності можна використовувати також для подання псевдографа. Тоді це не булева матриця: елемент a_{ij} дорівнює кількості ребер, що з'єднують v_i та v_j. Петлю у вершині v_i подають значенням діагонального елемента $a_{ii} = 1$.

Для подання орієнтованих графів також використовують матрицю суміжності. Це булева $n \times n$ матриця A з елементами a_{ij} $(i, j = 1, \ldots, n)$, де

$$a_{ij} = \begin{cases} 1, & \text{якщо } (v_i, v_j) \in E, \\ 0 & \text{в протилежному випадку.} \end{cases}$$

Приклад 3.17. Матриця суміжності для графа, зображеного на рис. 3.15, має вигляд

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}.$$

Зазначимо, що матриця суміжності орієнтованого графа, загалом кажучи, несиметрична.

Матрицю суміжності можна використовувати й для подання орієнтованого мультиграфа. У такому разі це не булева матриця: елемент a_{ij} дорівнює кількості дуг, які мають v_i початковою вершиною, а v_j — кінцевою.

Велика перевага матриці суміжності як способу подання графа — швидкий доступ до інформації: за один крок можна одержати відповідь на питання, чи існує ребро (дуга) з v_i у v_j. Вада полягає в тому, що незалежно від кількості ребер обсяг пам'яті становить n^2 комірок. Як іще один аргумент проти використання матриці суміжності можна навести теорему про кількість кроків алгоритму, який на основі матриці суміжності перевіряє якусь властивість простого графа (див. підрозділ 3.5, теорему 3.9).

Нарешті, розглянемо ще два способи подання графів у пам'яті комп'ютера. Будь-яку скінченну послідовність довільних елементів будемо називати *списком*, а кількість елементів списку — його *довжиною*.

3.3.3. Подання графа списком пар (списком ребер)

Метод зображення графа списком пар, які відповідають його ребрам (або дугам), значно економніший щодо пам'яті, особливо якщо m (кількість ребер) значно менша, ніж n^2 (n — кількість вершин). Пара $[u, v]$ відповідає ребру $\{u, v\}$ якщо граф неорієнтований, і дузі (u, v), якщо граф орієнтований.

Для графів, зображених на рис. 3.14 та 3.15, списки пар подані відповідно на рис. 3.16, *а* та 3.16, *б*.

v_1	v_2
v_1	v_3
v_1	v_4
v_2	v_4
v_3	v_4

а

v_1	v_2
v_2	v_2
v_2	v_3
v_3	v_2
v_3	v_4
v_4	v_1

б

Рис. 3.16

Очевидно, що обсяг пам'яті в цьому разі дорівнює $2m$ (m — кількість ребер або дуг). Це найекономніший щодо пам'яті спосіб. Його вада — велика (порядку m) кількість кроків для знаходження множини вершин, до яких ідуть ребра чи дуги із заданої вершини. Ситуацію можна значно поліпшити, упорядкувавши множину пар лексикографічно та застосувавши двійковий пошук.

3.3.4. Подання графа списками суміжності

Орієнтований граф G (без кратних дуг, але, можливо, з петлями) можна подати, зазначивши скінченну непорожню множину вершин V та відповідність Г, котра показує, як зв'язані між собою вершини. Відповідність Г — багатозначне відображення множини V у V. Граф у такому разі позначають парою $G = (V, \Gamma)$. У літературі часто означають (орієнтований) граф саме в таких поняттях [19, 20].

Приклад 3.18. Для орієнтованого графа, зображеного на рис. 3.15, відповідність Г задано табл. 3.2.

Таблиця 3.2

v	**Г(v) (термінальні вершини)**
v_1	v_2
v_2	v_2, v_3
v_3	v_2, v_4
v_4	v_1

Цей спосіб подання можна використовувати й для неорієнтованих простих графів, якщо кожне ребро умовно замінити двома протилежно спрямованими дугами.

Приклад 3.19. Для простого графа, зображеного на рис. 3.14 відповідність Г задано табл. 3.3.

Таблиця 3.3

v	**Г(v) (суміжні з v вершини)**
v_1	v_2, v_3, v_4
v_2	v_1, v_4
v_3	v_1, v_4
v_4	v_1, v_2, v_3

Якщо відображення Г діє не на одну вершину, а на множину вершин $A = \{v_{i_1}, v_{i_2}, ..., v_{i_p}\}$, то під $\Gamma(A)$ розуміють об'єднання множин

$$\Gamma(A) = \Gamma(v_{i_1}) \cup \Gamma(v_{i_2}) \cup ... \cup \Gamma(v_{i_p}).$$

Розглянемо спосіб комп'ютерного подання графа *списками суміжності*. Для цього використовують масив *Adj* із $|V| = n$ списків — по одному на кожну вершину. Для кожної вершини $u \in V$ список *Adj*[u] містить у довільному порядку покажчики на всі вершини множини $\Gamma(u)$.

Приклад 3.20. На рис. 3.17 зображено неорієнтований граф з рис. 3.14 за допомогою списків суміжності. Аналогічне зображення для орієнтованого графа з рис. 3.15 показано на рис. 3.18.

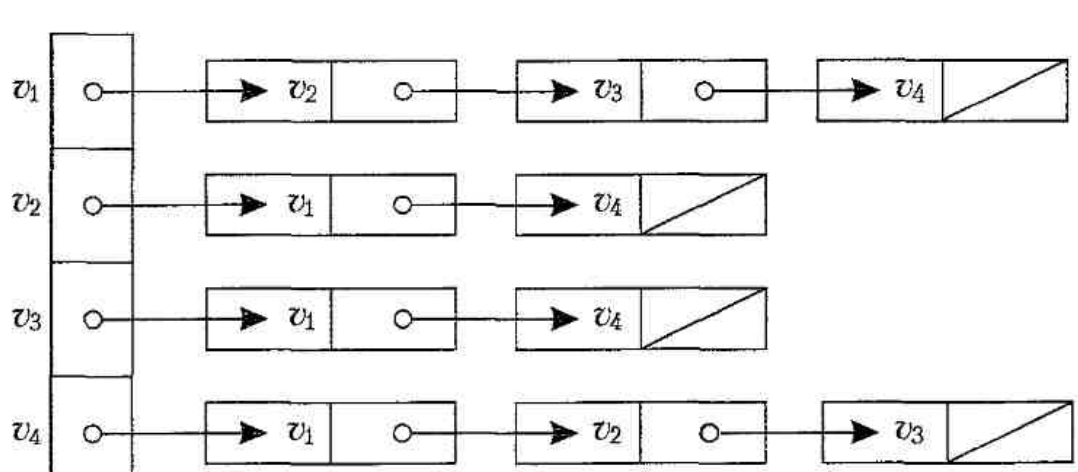


Рис. 3.17

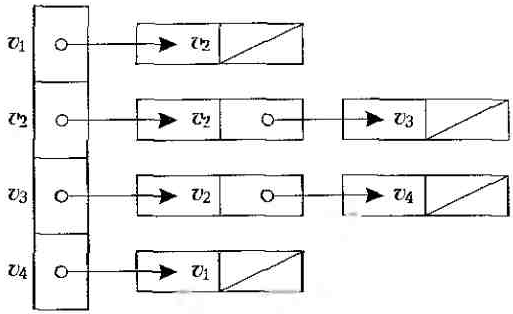


Рис. 3.18

Для орієнтованого графа сума довжин усіх списків суміжних вершин дорівнює загальній кількості дуг: дузі (u, v) відповідає елемент v зі списку $Adj[u]$. Для неорієнтованого графа ця сума дорівнює подвоєній кількості ребер, бо ребро $\{u, v\}$ породжує елемент у списку суміжних вершин як для вершини u, так і для вершини v

3.4. Шляхи та цикли. Зв'язність

Шляхом довжиною r [52] із вершини u в вершину v в неорієнтованому графі називають послідовність ребер $e_1 = \{x_0, x_1\}$, $e_2 = \{x_1, x_2\}$, ..., $e_r = \{x_{r-1}, x_r\}$, де $x_0 = u$, $x_r = v$, r – натуральне число. Отже, шлях довжиною r має r ребер, причому ребро враховують стільки разів, скільки воно міститься в шляху. Вершини u та v називають *крайніми*, а решту вершин шляху – *внутрішніми*.

Циклом у неорієнтованому графі називають шлях, який з'єднує вершину саму собою, тобто $u = v$.

У простому графі шлях можна задати послідовністю вершин, через які він проходить: $x_0, x_1, x_2, ..., x_{r-1}, x_r$.

Шлях або цикл називають *простим*, якщо він не містить повторюваних ребер.

Говорять, що шлях із крайніми вершинами u та v *з'єднує* ці вершини. Шлях, що з'єднує вершини u та v, позначають як $\langle u, v\rangle$ та називають $\langle u, v\rangle$-*шляхом*.

Приклад 3.21. На рис. 3.19 зображено простий граф. У ньому a, d, c, f, e – простий шлях довжиною 4, оскільки пари $\{a, d\}$, $\{d, c\}$, $\{c, f\}$ і $\{f, e\}$ – ребра. Однак d, e, c, b – не шлях, бо пара $\{e, c\}$ – не ребро. Зазначимо, що b, c, f, e, b – цикл довжиною 4, позаяк $\{b, c\}$, $\{c, f\}$, $\{f, e\}$ й $\{e, b\}$ – ребра та цей шлях починається й закінчується в одній і тій самій вершині b. Шлях a, b, e, d, a, b, довжина якого дорівнює 5, не простий, тому що він двічі проходить через ребро $\{a, b\}$.

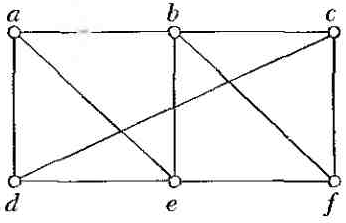


Рис. 3.19

Неорієнтований граф називають *зв'язним*, якщо будь-які дві його вершини з'єднані шляхом. Граф називають *незв'язним*, якщо він не є зв'язним. Незв'язний граф складається з двох або більше зв'язних підграфів, кожна пара з яких не має спільних вершин. Ці зв'язні підграфи називають *компонентами зв'язності* чи просто *компонентами* графа.

Приклад 3.22. Граф G на рис. 3.20 зв'язний; граф H — незв'язний, оскільки не існує шляху $\langle y, v\rangle$. Граф H має дві компоненти.

Рис. 3.20

Віддаллю $d(u, v)$ між вершинами u та v називають довжину найкоротшого $\langle u, v\rangle$-шляху, а сам цей шлях називають *геодезичним*. Легко переконатись, що найкоротший шлях не містить повторюваних вершин (і ребер). Звідси випливає така теорема.

ТЕОРЕМА 3.4. Між кожною парою різних вершин зв'язного неорієнтованого графа існує простий шлях.

Для орієнтованого графа вводять поняття *орієнтованого шляху* (або просто *шляху*) з вершини u у вершину v. Це скінченна послідовність дуг $e_1 = (x_0, x_1)$, $e_2 = (x_1, x_2)$, ..., $e_r = (x_{r-1}, x_r)$, де $x_0 = u$, $x_r = v$. Вершини u та v, як і в неорієнтованому графі, називають *кінцевими*, а решту вершин шляху — *внутрішніми*. *Довжиною* шляху називають кількість дуг, з яких він складається. *Орієнтованим циклом* називають орієнтований шлях, який з'єднує вершину саму із собою, тобто $u = v$. Орієнтований шлях або цикл називають *простим*, якщо жодна дуга не міститься в ньому більше одного разу.

Для орієнтованого графа поняття зв'язності вводять по-різному, залежно від того, чи враховано напрямок.

Орієнтований граф називають *сильно зв'язним*, якщо для будь-яких його різних вершин u та v існують орієнтовані шляхи від u до v та від v до u. Отже, для сильної зв'язності орієнтованого графа має існувати послідовність дуг з урахуванням орієнтації від будь-якої вершини графа до будь-якої іншої.

Орієнтований граф може не бути сильно зв'язним, але може бути, так би мовити, „в одному цілому". У зв'язку з цим дамо таке означення. Орієнтований граф називають *слабко зв'язним*, якщо існує шлях між будь-якими двома різними вершинами у відповідному йому неорієнтованому графі (тобто без урахування напрямку дуг).

Зрозуміло, що сильно зв'язний граф водночас і слабко зв'язний.

Приклад 3.23. На рис. 3.21 зображено графи G й H. Граф G сильно зв'язний. Граф H слабко зв'язний; він не сильно зв'язний, бо не існує орієнтованого шляху від w_1 до w_2.

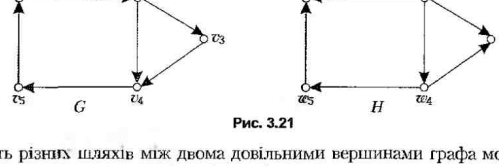


Рис. 3.21

Кількість різних шляхів між двома довільними вершинами графа можна підрахувати за допомогою матриці суміжності.

ТЕОРЕМА 3.5. Нехай G — граф (орієнтований або неорієнтований; можуть бути також кратні ребра й петлі), A — його матриця суміжності, яка відповідає заданій нумерації вершин $v_1, v_2, ..., v_n$. Тоді кількість різних шляхів довжиною r (r — натуральне) з вершини v_i у вершину v_j дорівнює (i, j)-му елементу матриці A^r.

Доведення (методом математичної індукції за r). Для $r = 1$ твердження теореми очевидне: кількість шляхів від v_i до v_j довжиною 1 дорівнює (i, j)-му елементу матриці A, бо цей елемент дорівнює кількості ребер (дуг), які з'єднують v_i та v_j.

Припустимо, що (i, j)-й елемент матриці A^r дорівнює кількості різних шляхів довжиною r від вершини v_i до вершини v_j. Це індуктивна гіпотеза. Оскільки $A^{r+1} = A^r A$, то (i, j)-й елемент матриці A^{r+1} дорівнює $\sum_{k=1}^{n} b_{ik} a_{kj}$, де b_{ik} — (i, k)-й елемент матриці A^r, a_{kj} — (k, j)-й елемент матриці суміжності A.

За індуктивною гіпотезою b_{ik} дорівнює кількості шляхів довжиною r із вершини v_i у вершину v_k. Шлях довжиною $r+1$ із v_i у v_j складається зі шляху довжиною r із v_i до якоїсь проміжної вершини v_k та ребра (дуги) з v_k до v_j. За правилом добутку з комбінаторики кількість таких шляхів дорівнює добутку кількості b_{ik} шляхів довжиною r із v_i до v_k та кількості a_{kj} ребер (дуг) із v_k до v_j, тобто $b_{ik}a_{kj}$. Якщо ці добутки підсумувати для всіх можливих проміжних вершин v_k, то потрібний результат випливає з комбінаторного правила суми.

Розглянемо неорієнтовані графи K_n і C_n (див. рис. 3.9). Обидва ці графи зв'язні, проте інтуїтивно зрозуміло, що для $n > 3$ граф K_n „сильніше зв'язаний", ніж граф C_n. Розглянемо два поняття, які характеризують міру зв'язності простого графа.

Числом вершинної зв'язності (або просто *числом зв'язності*) $\kappa(G)$ простого графа G називають найменшу кількість вершин, вилучення яких дає незв'язний або одновершинний граф. Зазначимо, що вершину вилучають разом із інцидентними їй ребрами. Наприклад, $\kappa(K_1) = 0$, $\kappa(K_n) = n - 1$, $\kappa(C_n) = 2$. Граф G, зображений на рис. 3.22, зв'язний, але його зв'язність можна порушити вилученням вершини u. Отже, $\kappa(G) = 1$. Якщо ж спробувати порушити зв'язність цього графа вилученням ребер (а не вершин), то потрібно вилучити не менше ніж три ребра.

Нехай G — простий граф з $n > 1$ вершинами. *Числом реберної зв'язності* $\lambda(G)$ графа G називають найменшу кількість ребер, вилучення яких дає незв'язний граф. Число реберної зв'язності одновершинного графа вважають таким, що дорівнює 0. Для графа G, зображеного на рис. 3.22, $\lambda(G) = 3$.

Рис. 3.22

Вершину u простого графа G називають *точкою з'єднання*, якщо граф G в разі її вилучення матиме більше компонент, ніж даний граф G. Зокрема, зв'язний граф G без вершини u стає незв'язним. Нагадаємо, що вершину u при цьому вилучають разом з інцидентними їй ребрами. Ребро графа G називають *мостом*, якщо його вилучення збільшує кількість компонент. Отже, точки з'єднання й мости — це своєрідні „вузькі місця" простого графа.

Приклад 3.24. Граф H, зображений на рис. 3.22, має три точки з'єднання t, s, p й один міст $\{t, s\}$.

Позначимо як $\delta(G)$ мінімальний степінь вершин графа G. Можна довести, що $\kappa(G) \leqslant \lambda(G) \leqslant \delta(G)$. Простий граф G називають *t-зв'язним*, якщо $\kappa(G) \geqslant t$, і *реберно t-зв'язним*, якщо $\lambda(G) \geqslant t$. Отже, відмінний від K_1 граф однозв'язний тоді й лише тоді, коли він зв'язний. Двозв'язні графи — це зв'язні графи без точок з'єднання, відмінні від графа K_1.

Приклад 3.25. Граф G, зображений на рис. 3.22, однозв'язний і реберно 3-зв'язний.

Очевидно, що кількість ребер у зв'язному простому графі з n вершинами не перевищує кількості ребер у графі K_n, тобто $n(n-1)/2$. Але скільки може бути ребер у простому графі з n вершинами й фіксованою кількістю компонент?

ТЕОРЕМА 3.6. Якщо простий граф G має n вершин і k компонент, то кількість m його ребер задовольняє нерівності

$$n-k \leqslant m \leqslant \frac{1}{2}(n-k)(n-k+1).$$

Доведення. Доведемо спочатку верхню оцінку. Нехай G — простий граф з n вершинами, k компонентами й максимальною для таких графів кількістю ребер m_{max}. Очевидно, що кожна компонента графа G — повний граф. Нехай K_p, K_q — дві компоненти, $p \geqslant q > 1$, v — вершина з другої компоненти. Вилучимо з графа всі ребра, інцидентні вершині v, і з'єднаємо цю вершину ребром із кожною вершиною з першої компоненти. Кількість вершин і компонент при цьому не зміниться, а кількість ребер зросте на $p-(q-1)=p-q+1>1$, що неможливо, бо граф G має максимально можливу кількість ребер. Отже, лише одна компонента графа G являє собою повний граф із більшою ніж 1 кількістю вершин $n-(k-1)=n-k+1$. Отже, $m_{max} = (1/2)\,(n-k)(n-k+1)$.

Доведемо нижню оцінку математичною індукцією за кількістю ребер m. Для $m=0$ твердження очевидне, оскільки тоді $k=n$ і, отже, $0 \leqslant 0$. Нехай тепер $m>0$, і нижня оцінка справджується для графів із меншою кількістю ребер, ніж m. Припустимо, що граф G має найменшу можливу кількість ребер m_{min} серед усіх

простих графів з n вершинами й k компонентами. Вилучивши довільне ребро, отримаємо граф з n вершинами, $k+1$ компонентою й $m_{\min}-1$ ребром. Для нього справджується припущення індукції: $n-(k-1)\leqslant m_{\min}-1$, звідки випливає нерівність $n-k\leqslant m_{\min}$.

Д. Кеніг сформулював простий критерій дводольності графа в термінах довжин простих циклів.

ТЕОРЕМА 3.7 (Кеніга, 1936 р.). Для того, щоб граф G був дводольним, необхідно й достатньо, щоб він не містив простих циклів із непарною довжиною.

Доведення. Необхідність. Нехай G — дводольний граф, C — один із його простих циклів довжиною k. Пройдемо всі ребра цього циклу, починаючи з вершини v. Зробивши k кроків, повернемось у вершину v. Оскільки кінці кожного ребра містяться в різних підмножинах вершин, то k — парне число.

Достатність. Нехай зв'язний граф $G=(V, E)$ з $n>1$ вершинами не має простих циклів із непарною довжиною та $v\in V$. Побудуємо розбиття $V=A\cup B$ ($A\cap B=\varnothing$) так. Довільну вершину $x\in V$ долучимо до множини A, якщо віддаль $d(x, v)$ парна, а ні, то до множини B. Залишилося довести, що породжені підграфи (див. підрозділ 3.1) $G(A)$ та $G(B)$ порожні. Припустимо, що це не так, тобто існують дві суміжні вершини u та w, які належать одній множині. Тоді жодна з цих вершин не збігається з v, бо $v\in A$, а всі вершини, суміжні з v, належать множині B. Нехай $U=\langle u, v\rangle$ та $W=\langle w, v\rangle$ — геодезичні шляхи, v_1 — остання (якщо починати від v) зі спільних вершин цих шляхів (рис. 3.23). Позначимо як X_U й Y_U відповідно частини шляху U від v до v_1 і від v_1 до u. Аналогічно, як X_W та Y_W позначимо відповідно частини шляху W від v до v_1 і від v_1 до w. Очевидно, що довжини шляхів X_U й X_W збігаються (тому ці шляхи геодезичні). Отже, довжини шляхів Y_U й Y_W мають один тип парності (позаяк вершини u та w належать одній множині, то їх віддалі від вершини v мають один тип парності). Але тоді об'єднання шляхів Y_U й Y_W та ребра $\{u, w\}$ являє собою простий цикл із непарною довжиною. Суперечність.

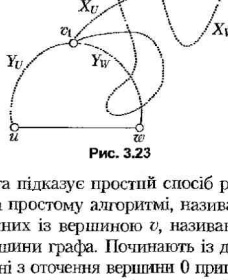


Рис. 3.23

Доведення теореми Кеніга підказує простий спосіб розпізнавання дводольності графа, що ґрунтується на простому алгоритмі, названому пошуком уширп [15]. Множину вершин, суміжних із вершиною v, називають *оточенням* вершини v. *Пошук уширп* нумерує вершини графа. Починають із довільної вершини, надають їй номер 0. Кожній вершині з оточення вершини 0 приписують номер 1. Тепер розглядають почергово оточення всіх вершин із номером 1, і всім вершинам, що нале-

жать цим оточенням і ще не мають номера, надають номер 2. Розглядають оточення всіх вершин із номером 2 та продовжують процес нумерації, доки це можливо. Якщо даний граф $G=(V, E)$ зв'язний, то пошук ушир занумерує всі його вершини.

Далі розіб'ємо множину вершин V на дві підмножини — A та B. До множини A долучимо всі вершини з парними номерами (та 0), а до множини B — з непарними. Розглянемо породжені підграфи $G(A)$ та $G(B)$. Якщо обидва вони порожні (достатньо перевірити, що всі пари вершин з однаковими номерами не суміжні), то G — дводольний граф, а ні, то — не дводольний.

Існує й інший, більш розповсюджений варіант пошуку ушир. Він відрізняється тим, що всі вершини отримують різні номери (див. підрозділ 3.9).

3.5. Ізоморфізм графів

У теорії графів і її застосуваннях істотно, що існують об'єкти (вершини графа) і зв'язки між ними (ребра). Тому доцільно не розрізняти графи, котрі можна одержати один з одного зміною позначень вершин. Сформулюємо ці міркування у вигляді такого означення.

Нехай $G_1=(V_1, E_1)$ і $G_2=(V_2, E_2)$ — прості графи, а φ: $V_1 \to V_2$ — бієкція. Якщо для будь-яких вершин u та v графа G_1 їх образи $\varphi(u)$ та $\varphi(v)$ суміжні в G_2 тоді й лише тоді, коли u та v суміжні в G_1, то цю бієкцію називають *ізоморфізмом* графа G_1 на граф G_2, а графи G_1 і G_2 — *ізоморфними* (записують $G_1 \cong G_2$).

Отже, прості графи G_1 і G_2 ізоморфні, якщо існує така бієкція $\varphi: V_1 \to V_2$, що

$$\forall\, u, v \in V_1\ (\{u, v\} \in E_1 \text{ тоді й лише тоді, коли } \{\varphi(u), \varphi(v)\} \in E_2).$$

Приклад 3.26. Графи на рис. 3.24 ізоморфні; бієкцію φ можна задати так: $\varphi(x_1)=y_1$; $\varphi(x_2)=y_4$; $\varphi(x_3)=y_3$; $\varphi(x_4)=y_2$.

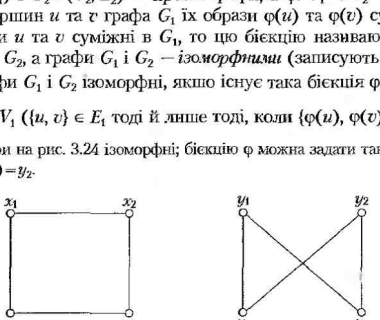


Рис. 3.24

Приклад 3.27. Усі три графи, зображені на рис. 3.25, ізоморфні. Довести це пропонуємо як вправу.

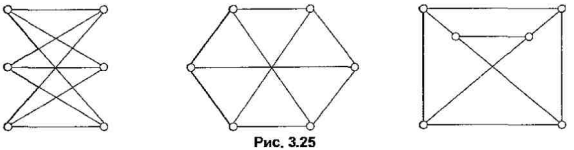


Рис. 3.25

Ізоморфні графи природно ототожнювати (їх можна зобразити одним рисунком). Вони могли б різнитися природою своїх елементів, але це не беруть до уваги, уводячи поняття „граф". Проте інколи все ж доводиться розрізняти ізоморфні графи, і тоді корисне поняття „позначеного графа". Граф з n вершинами називають *позначеним*, якщо його вершинам присвоєно якісь мітки, наприклад, числа 1, 2, ..., n. Ототожнимо кожну з вершин з її номером (і, отже, множину вершин — із множиною чисел $\{1, 2, ..., n\}$) й означимо рівність простих позначених графів G_1 і G_2 з однаковою кількістю вершин n так: $G_1 = G_2$ тоді й лише тоді, коли $E_1 = E_2$.

Приклад 3.28. На рис. 3.26 зображено три різні позначені графи.

Рис. 3.26

Щоб наголосити, що графи розрізняють лише з точністю до ізоморфізму, говорять про абстрактний граф. Останній має різні матриці суміжності залежно від нумерації вершин. З'ясуємо, як пов'язані між собою ці матриці. Нехай G і G_2 — прості позначені графи з n вершинами, і $G_1 \cong G_2$. Тоді ці графи різняться лише нумерацією вершин, тобто існує підстановка s на множині вершин, яка *зберігає суміжність*: вершини u та v тоді й лише тоді суміжні в графі G_1, коли їх образи $s(u)$ та $s(v)$ суміжні в G_2. Нехай $A = [a_{ij}]$ і $B = [b_{ij}]$ — матриці суміжності відповідно графів G_1 та G_2. Тоді $b_{s(i)s(j)} = a_{ij}$ $(i, j = 1, 2, ..., n)$. Тим самим доведено таку теорему.

ТЕОРЕМА 3.8. Прості графи ізоморфні тоді й лише тоді, коли їх матриці суміжності можна отримати одну з одної однаковими перестановками рядків і стовпців.

Виявити ізоморфізм дуже складно. Теоретично алгоритм перевірки пари простих графів на ізоморфізм існує — його сформульовано в попередній теоремі. Проте він незручний, бо для його виконання може бути потрібно до $n!$ перестановок і перевірок.

Часто неважко довести, що два прості графи не ізоморфні, якщо порушується властивість, інваріантна щодо ізоморфізму, наприклад:

- кількість вершин;
- кількість ребер;
- кількість вершин конкретного степеня (вершині $v \in V_1$, $\deg(v) = d$, має відповідати вершина $u = \varphi(v) \in V_2$, $\deg(u) = d$).

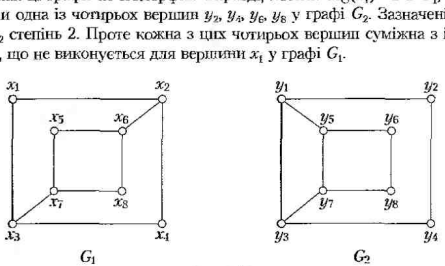
Є й інші інваріанти, але порушення інваріантності — це лише достатня умова неізоморфності графів. Не існує набору інваріантів для виявлення ізоморфізму.

Приклад **3.29.** Графи на рис. 3.27 неізоморфні. Вони мають по п'ять вершин і по шість ребер, проте граф G_2 має вершину степеня 1, якої не має граф G_1.

G_1 G_2

Рис. 3.27

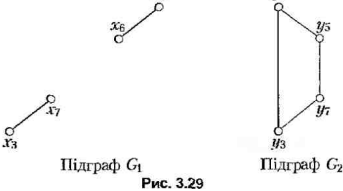
Приклад **3.30.** На рис. 3.28 зображено графи G_1 і G_2. Обидва вони мають по вісім вершин і по десять ребер. Вони також мають по чотири вершини степеня 2 і по чотири вершини степеня 3. *Однак ці графи не ізоморфні. Справді, позаяк* $\deg(x_1)=2$ *в* G_1, *то вершині* x_1 має відповідати одна із чотирьох вершин y_2, y_4, y_6, y_8 у графі G_2. Зазначені вершини мають у графі G_2 степінь 2. Проте кожна з цих чотирьох вершин суміжна з іншою вершиною степеня 2, що не виконується для вершини x_1 у графі G_1.



G_1 G_2

Рис. 3.28

Те, що графи, зображені на рис. 3.28, не ізоморфні, можна довести й інакше. На рис. 3.29 зображено підграфи графів G_1 і G_2, породжені вершинами степеня 3. Якщо графи G_1 і G_2 ізоморфні, то й зазначені підграфи мають бути ізоморфними. Проте підграфи з рис. 3.29 не ізоморфні.



Підграф G_1 Підграф G_2

Рис. 3.29

Ми розглянули поняття ізоморфізму для простого графа. Для неорієнтованих мультиграфів і псевдографів, а також орієнтованих графів природно вводять

поняття ізоморфізму як бієкції між множинами вершин, яка зберігає суміжність, кратності ребер, петлі та спрямування дуг. Теорема 3.8 залишається правильною для мультиграфів, псевдографів і орієнтованих графів.

Розглянемо тепер один результат, пов'язаний з оцінкою складності алгоритмів на графах, які задано матрицею суміжності. Нехай P — якась властивість простого графа: $P(G) = 1$ або $P(G) = 0$ залежно від того, чи має граф G цю властивість. Припустимо, що властивість P задовольняє такі умови.

1. $P(G) = P(G')$, якщо графи G та G' ізоморфні.
2. $P(G) = 0$ для довільного порожнього графа й $P(G) = 1$ для довільного повного графа з достатньо великою кількістю вершин.
3. Додавання ребра не порушує властивості P, тобто $P(G) \leqslant P(G')$ для довільних простих графів $G = (V, E)$ та $G' = (V, E')$ таких, що $E \subset E'$.

Приклад такої властивості P — існування простого циклу в графі, який має принаймні три вершини.

ТЕОРЕМА 3.9. Якщо P — властивість простого графа G, що задовольняє умови 1–3, то будь-який алгоритм, котрий перевіряє властивість P (тобто обчислює значення $P(G)$ для графа G), виходячи з матриці суміжності, виконує в найгіршому випадку $\Omega(n^2)$ кроків, де n — кількість вершин графа.

3.6. Ейлерів цикл у графі

Початок теорії графів як розділу математики пов'язують із задачею про кенігсберзькі мости. Сім мостів міста Кенігсберга (нині – Калінінград у Росії) було розміщено на річці Прегель так, як зображено на рис. 3.30. Чи можна, починаючи з якоїсь точки міста, пройти через усі мости точно по одному разу й повернутись у початкову точку? Швейцарський математик Л. Ейлер розв'язав цю задачу. Його розв'язання, опубліковане 1736 р., було першим явним застосуванням теорії графів. Ейлер поставив у відповідність плану міста мультиграф G, вершини якого відповідають чотирьом частинам A, B, C, D міста, а ребра — мостам. Цей мультиграф зображено на рис. 3.31.

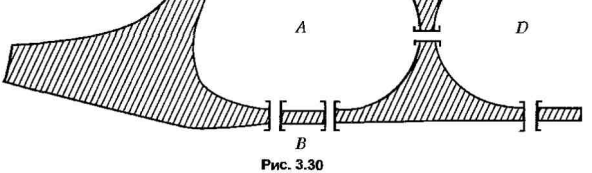


Рис. 3.30

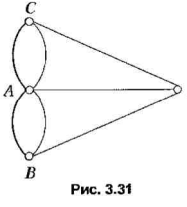


Рис. 3.31

Отже, задачу про кенігсберзькі мости мовою теорії графів можна сформулювати так: чи існує в мультиграфі G простий цикл, який містить усі ребра цього мультиграфа? Ейлер довів нерозв'язність задачі про кенігсберзькі мости. Нагадаємо, що в простому циклі ребра не повторюються, а вершини можуть повторюватись.

Ейлеровим циклом у зв'язному мультиграфі G називають простий цикл, який містить усі ребра графа, *ейлеровим шляхом* — простий шлях, що містить усі ребра графа.

Приклад 3.31. На рис. 3.32 проілюстровано концепцію ейлерових циклів і шляхів. Граф G_1 має ейлерів цикл, наприклад, a, e, c, d, e, b, a; граф G_3 не має ейлерового циклу, але має ейлерів шлях: a, c, d, e, b, d, a, b; граф G_2 не має ні ейлерового циклу, ні ейлерового шляху.

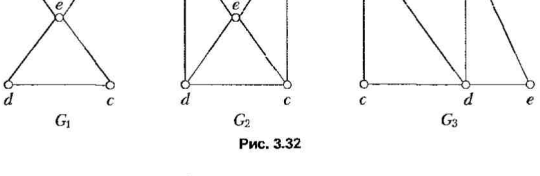


Рис. 3.32

Існує простий критерій (необхідна й достатня умова) для виявлення того, чи має граф ейлерів цикл.

ТЕОРЕМА 3.10. Зв'язний мультиграф G має ейлерів цикл тоді й лише тоді, коли степені всіх його вершин парні.

Доведення. Необхідність. Нехай у графі G існує ейлерів цикл. Тоді він проходить через кожну вершину графа та входить до неї по одному ребру, а виходить по іншому. Це означає, що кожна вершина інцидентна парній кількості ребер ейлерового циклу. Оскільки такий цикл містить усі ребра графа G, то звідси випливає парність степенів усіх його вершин.

Достатність. Припустимо тепер, що всі вершини графа G мають парний степінь. Почнемо шлях P_1 із довільної вершини v_1 і продовжимо його, наскільки це можливо, вибираючи щоразу нове ребро. Позаяк степені всіх вершин парні, то, увійшовши в будь-яку вершину, відмінну від v_1, ми завжди маємо можливість вийти з неї через іще не пройдене ребро. Тому шлях P_1 можна продовжити, додавши це ребро. Отже, побудова шляху P_1 завершиться у вершині v_1, тобто P_1 обов'язково виявиться циклом. Якщо з'ясується, що шлях P_1 містить усі ребра графа G, то це ейлерів цикл. У протилежному випадку вилучимо з G всі ребра циклу P_1 і всі вершини, інцидентні лише вилученим ребрам. Отримаємо якийсь зв'язний граф G_1. Оскільки P_1 та G мають вершини лише парних степенів, то, очевидно, і граф G_1 матиме цю властивість. Окрім того, позаяк граф G зв'язний, то графи P_1 і G_1 мають принаймні одну спільну вершину v_2. Тепер із вершини v_2 побудуємо цикл P_2 в графі G_1 аналогічно до того, як ми будували цикл P_1 у графі G. Цикл P_2 вставимо в цикл P_1 на місце вершини v_2. Одержимо цикл P_3. Описані побудови показано на рис. 3.33.

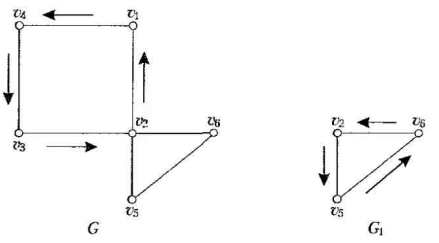


Рис. 3.33

Цикл $P_3 = v_1, v_4, v_3, P_2, v_1 = v_1, v_4, v_3, v_2, v_5, v_6, v_2, v_1$ ейлерів. Якби він виявився не ейлеровим, то потрібно продовжити аналогічні побудови й отримати ще більший цикл. Цей процес закінчиться побудовою ейлерового циклу. Зазначимо, що доведення достатності має конструктивний характер: подано алгоритм побудови ейлерового циклу.

Існує й інший алгоритм побудови ейлерового циклу, який дає змогу будувати цей цикл одразу. Це алгоритм Флері (Fleury).

Алгоритм Флері побудови ейлерового циклу

Робота алгоритму полягає в нумерації ребер у процесі побудови ейлерового циклу.

Крок 1. Початок. Починаємо з довільної вершини u та присвоюємо довільному ребру $\{u, v\}$ номер 1. Викреслюємо ребро $\{u, v\}$ й переходимо у вершину v.

Крок 2. Ітерація. Нехай v — вершина, у яку ми перейшли на попередньому кроці, k — останній присвоєний номер. Вибираємо довільне ребро, інцидентне вершині v, причому міст вибираємо лише тоді, коли немає інших можливостей. Присвоюємо вибраному ребру номер $(k+1)$ і викреслюємо його.

Крок 3. Закінчення. Цей процес закінчуємо, коли всі ребра графа викреслено та пронумеровано — ці номери задають послідовність ребер в ейлеровому циклі.

Повертаючись до задачі про кенігсберзькі мости, виявляємо, що мультиграф, зображений на рис. 3.31, має всі вершини непарного степеня. Отже, цей мультиграф не має ейлерового циклу, тому неможливо пройти кожний міст по одному разу й повернутись у початкову точку шляху.

ТЕОРЕМА 3.11. Зв'язний мультиграф має ейлерів шлях, але не має ейлерового циклу тоді й лише тоді, коли він має точно дві вершини непарного степеня.

Зазначимо, що будь-який ейлерів шлях починається в одній із цих двох вершин непарного степеня, а закінчується в іншій. Оскільки мультиграф для кенігсберзьких мостів має чотири вершини з непарними степенями, можна дійти висновку про неможливість пройти кожний міст по одному разу, навіть якщо не потрібно повертатись у початкову точку.

Граф, який має ейлерів цикл, часто називають *ейлеровим*.

Ейлеровим циклом у слабко зв'язному орієнтованому мультиграфі називають орієнтований простий цикл, який містить усі ребра графа.

ТЕОРЕМА 3.12. Орієнтований слабко зв'язний мультиграф має ейлерів цикл тоді й лише тоді, коли напівстепінь входу кожної вершини дорівнює її напівстепеню виходу.

3.7. Гамільтонів цикл у графі

Шлях $x_0, x_1, ..., x_{n-1}, x_n$ у зв'язному графі $G=(V, E)$ називають *гамільтоновим шляхом*, якщо $V=\{x_0, x_1, ..., x_{n-1}, x_n\}$ і $x_i \neq x_j$ для $0 \leqslant i < j \leqslant n$. Цикл $x_0, x_1, ..., x_{n-1}, x_n, x_0$ (тут $n>1$) у графі $G=(V, E)$ називають *гамільтоновим циклом*, якщо $x_0, x_1, ..., x_{n-1}, x_n$ — гамільтонів шлях. Інакше кажучи, гамільтонів цикл і гамільтонів шлях проходять через кожну вершину графа точно один раз (можливо, окрім першої й останньої вершин). Зазначимо, що гамільтонові цикл і шлях, узагалі кажучи, не містять усіх ребер графа.

Термін „гамільтонів" у цих означеннях походить від імені відомого ірландського математика У. Гамільтона (W. Hamilton), який 1857 р. запропонував гру „Навколосвітня подорож". Кожній із двадцяти вершин додекаедра (правильного дванадцятигранника, грані якого — п'ятикутники) приписано назву одного з великих міст світу. Потрібно, розпочавши з довільного міста, відвідати решту 19 міст точно один раз і повернутись у початкове місто. Перехід дозволено ребрами додекаедра [52].

Приклад 3.32. Ту саму задачу можна зобразити й на площині (рис. 3.34). Вона зводиться до відшукання в графі гамільтонового циклу. Один із можливих розв'язків показано потовщеними лініями.

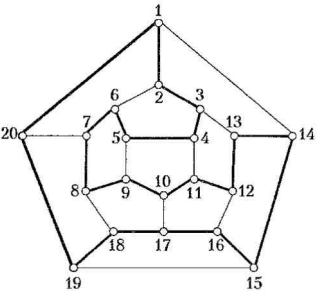


Рис. 3.34

Не всі зв'язні графи мають гамільтонів цикл хоча б тому, що такий граф має бути двозв'язним (тобто граф, який має точки з'єднання, не може мати гамільтонового циклу). Приклад графа, зображеного на рис. 3.35, свідчить, що двозв'язності недостатньо для наявності гамільтонового циклу. Незважаючи на зовнішню подібність формулювань задач про існування ейлерового й гамільтонового циклів, ці задачі принципово різні. Використовуючи результати попереднього підрозділа, легко виявити, чи має граф ейлерів цикл, і, якщо має, то побудувати його.

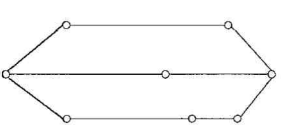


Рис. 3.35

Ситуація для гамільтонового циклу істотно інша. Відповісти на питання, чи має граф гамільтонів цикл, зазвичай, дуже важко. Вивчення достатніх умов наявності в графі гамільтонового циклу — один із важливих напрямів у теорії графів. Інтуїтивно зрозуміло, що граф із багатьма ребрами, достатньо рівномірно розподіленими, з великою ймовірністю має гамільтонів цикл. Доведемо одну з теорем такого типу.

ТЕОРЕМА 3.13 (Г. Дірака, 1952 р.). Якщо для кожної вершини v зв'язного простого графа з $n \geqslant 3$ вершинами виконується нерівність $\deg(v) \geqslant n/2$, то цей граф має гамільтонів цикл.

Доведення. Додамо до графа G k нових вершин і з'єднаємо ребром кожну з них із кожною вершиною з G. Отриманий граф з $n+k$ вершинами позначимо як G'.

Уважатимемо, що k — найменша кількість вершин, потрібних для того, щоб у графі G з'явився гамільтонів цикл. Доведемо, що припущення $k \geqslant 1$ призводить до суперечності.

Нехай $v, p, w, ..., v$ — гамільтонів цикл у графі G', де v та w — вершини з G, а p — одна з нових вершин. Тоді вершина w несуміжна з v, інакше ми могли б не використовувати вершину p, що суперечить мінімальності числа k. Більше того, вершина w', суміжна з вершиною w, не може в гамільтоновому циклі безпосередньо йти за вершиною v', суміжною з v. Справді, якщо є гамільтонів цикл $v, p, w, ..., v', w', ..., v$, то ми можемо замінити його на $v, v', ..., w, w', ..., v$, повернувши частину циклу між w та v'. Це знову суперечить мінімальності числа k (рис. 3.36).

Рис. 3.36

Отже, кількість вершин графа G', не суміжних з w, не менша від кількості вершин, суміжних з v (тобто дорівнює принаймні $n/2+k$). Натомість очевидно, що кількість вершин графа G', суміжних з w, також дорівнює принаймні $n/2+k$. Але жодна з вершин графа G' не може бути водночасно суміжною й несуміжною з вершиною w, тому загальна кількість вершин графа G' дорівнює щонайменше $n+2k$. Але це суперечить тому, що кількість вершин графа G' дорівнює $n+k$.

Як знайти гамільтонів цикл або переконатись, що його немає? Очевидний алгоритм, який можна застосувати, — це повний перебір усіх можливостей, тобто $n!$ перестановок усіх вершин графа й перевірок. Зрозуміло, що такий спосіб не практичний. Розгляд реального алгоритму бектрекінг відкладемо до наступного розділу (розділ 4.5, приклад 4.15).

Задача знаходження гамільтонового циклу *NP*-повна (див. розділ 10). Для таких задач невідомий ефективний (тобто з поліноміальною складністю) алгоритм їх розв'язання, і є вагомі підстави вважати, що його не існує.

Граф, який містить гамільтонів цикл, часто називають *гамільтоновим графом*.

3.8. Зважені графи й алгоритми пошуку найкоротших шляхів

У реальних задачах на графах часто потрібно брати до уваги додаткову інформацію — фактичну віддаль між окремими пунктами, вартість проїзду, час проїзду тощо. Для цього використовують поняття зваженого графа.

Зваженим називають простий граф, кожному ребру e якого приписано дійсне число $w(e)$. Це число називають *вагою ребра* e. Аналогічно означають *зважений*

орієнтований граф: це такий орієнтований граф, кожній дузі e якого приписано дійсне число $w(e)$, називане *вагою дуги*.

Розглянемо три способи зберігання зваженого графа $G = (V, E)$ в пам'яті комп'ютера. Нехай $|V| = n$, $|E| = m$.

Перший — подання графа матрицею ваг W, яка являє собою аналог матриці суміжності. Її елемент $w_{ij} = w(v_i, v_j)$, якщо ребро $\{v_i, v_j\} \in E$ (у разі орієнтованого графа — дуга $(v_i, v_j) \in E$). Якщо ж ребро $\{v_i, v_j\} \notin E$ (або дуга $(v_i, v_j) \notin E$), то $w_{ij} = 0$ чи $w_{ij} = \infty$ залежно від розв'язуваної задачі.

Другий спосіб — подання графа списком ребер. Для зваженого графа під кожний елемент списку E можна відвести три комірки — дві для ребра й одну для його ваги, тобто всього потрібно $3m$ комірок.

Третій спосіб — подання графа списками суміжності. Для зваженого графа кожний список $Adj[u]$ містить окрім покажчиків на всі вершини v множини $\Gamma(u)$ ще й числа $w(u, v)$.

Довжиною шляху в зваженому графі називають суму ваг ребер (дуг), які утворюють цей шлях. Якщо граф не зважений, то вагу кожного ребра (кожної дуги) уважають рівною 1 й отримують раніше введене поняття довжини шляху як кількості ребер (дуг) у ньому.

Задача про найкоротший шлях полягає в знаходженні найкоротшого шляху від заданої початкової вершини a до заданої вершини z. Наступні дві задачі — безпосередні узагальнення сформульованої задачі про найкоротший шлях.

1. Для заданої початкової вершини a знайти найкоротші шляхи від a до всіх інших вершин.
2. Знайти найкоротші шляхи між усіма парами вершин.

Виявляється, що майже всі методи розв'язання задачі про найкоротший шлях від заданої початкової вершини a до заданої вершини z також дають змогу знайти й найкоротші шляхи від вершини a до всіх інших вершин графа. Отже, за їх допомогою можна розв'язати задачу 1 із невеликими додатковими обчислювальними витратами. З іншого боку, задачу 2 можна розв'язати або n разів застосувавши алгоритм задачі 1 із різними початковими вершинами, або один раз застосувавши спеціальний алгоритм.

Розглянемо два алгоритми: перший алгоритм призначений для розв'язування задачі 1, другий — для задачі 2.

Найефективніший алгоритм визначення довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої запропонував 1959 р. датський математик Е. Дейкстра (E. Dijkstra). Цей алгоритм застосовний лише тоді, коли вага кожного ребра (дуги) додатна. Опишемо докладно цей алгоритм для орієнтованого графа.

Нехай $G = (V, E)$ — зважений орієнтований граф, $w(v_i, v_j)$ — вага дуги (v_i, v_j). Почавши з вершини a, знаходимо віддаль від a до кожної із суміжних із нею вершин. Вибираємо вершину, віддаль від якої до вершини a найменша; нехай це буде вершина v^*. Далі знаходимо віддалі від вершини a до кожної вершини, суміжної з v^* вздовж шляху, який проходить через вершину v^*. Якщо для якоїсь

із таких вершин ця віддаль менша від поточної, то замінюємо нею поточну віддаль. Знову вибираємо вершину, найближчу до a й не вибрану раніше; повторюємо процес.

Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Є мітки двох типів — тимчасові та постійні. Вершини з постійними мітками групують у множину M, яку називають *множиною позначених вершин*. Решта вершин має тимчасові мітки, і множину таких вершин позначимо як T, $T = V \setminus M$. Позначатимемо мітку (тимчасову чи постійну) вершини v як $l(v)$. Значення постійної мітки $l(v)$ дорівнює довжині найкоротшого шляху від вершини a до вершини v, тимчасової — довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.

Фіксованою початковою вершиною вважаємо вершину a; довжину найкоротшого шляху шукаємо до вершини z (або до всіх вершин графа).

Тепер формально опишемо алгоритм Дейкстри.

Алгоритм Дейкстри

Наведемо кроки алгоритму.

Крок 1. Присвоювання початкових значень. Виконати $l(a) := 0$ та вважати цю мітку постійною. Виконати $l(v) := \infty$ для всіх $v \neq a$ й уважати ці мітки тимчасовими. Виконати $x := a$, $M := \{a\}$.

Крок 2. Оновлення міток. Для кожної вершини $v \in \Gamma(x) \setminus M$ замінити мітки: $l(v) := \min\{l(v); l(x) + w(x, v)\}$, тобто оновлювати тимчасові мітки вершин, у які з вершини x іде дуга.

Крок 3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v^* з умови $l(v^*) = \min\{l(v)\}$, $v \in T$, де $T = V \setminus M$.

Крок 4. Уважати мітку вершини v^* постійною й виконати $M := M \cup \{v^*\}$; $x := v^*$ (вершину v^* включено в множину M).

Крок 5. а) Для пошуку шляху від a до z: якщо $x = z$, то $l(z)$ — довжина найкоротшого шляху від a до z, зупинитись; якщо $x \neq z$, то перейти до кроку 2.

б) Для пошуку шляхів від a до всіх інших вершин: якщо всі вершини отримали постійні мітки (включені в множину M), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

Обґрунтування алгоритму Дейкстри

Мітка вершини v ($l(v) \neq \infty$) дорівнює довжині $\langle a, v\rangle$-шляху, побудованого за алгоритмом до певного моменту. Тому достатньо довести, що постійна мітка $l(v)$ кожної вершини v дорівнює довжині найкоротшого $\langle a, v\rangle$-шляху. Отже, доведемо, що значення $l(v)$ — постійної мітки вершини v — дорівнює довжині найкоротшого шляху від початкової вершини a до вершини v. Для доведення цього застосуємо математичну індукцію.

Нехай вершина v одержала свою постійну мітку на k-й ітерації, тобто після k-го виконання кроку 4 алгоритму Дейкстри. У разі $k=1$ твердження очевидне. Припустимо, що воно справджується для вершин, які отримали свої постійні мітки на ітераціях із номерами 2, 3, ..., $k-1$. Позначимо як P^0 шлях від a до v, побудований за алгоритмом Дейкстри за k ітерацій, а як P^* — найкоротший шлях від a до v. Довжину шляху P позначатимемо як $w(P)$. За умовою $w(P^0)=l(v)$. Нехай M і $T=V\setminus M$ — множини вершин відповідно з постійними та тимчасовими мітками перед початком k-ї ітерації.

Розглянемо ситуацію перед початком k-ї ітерації. Можливі два випадки.

Випадок 1. Деякі вершини шляху P^* належать множині T. Нехай $\bar{v}$ — перша (починаючи від a) з тих вершин шляху P^*, які належать множині T, а вершина v' безпосередньо передує $\bar{v}$ на шляху P^*. За вибором $\bar{v}$ маємо, що $v'\in M$. Позначимо як P_1^* частину шляху P^* від a до $\bar{v}$. За припущенням індукції $l(v')$ — довжина найкоротшого шляху від a до v'. Тому $w(P_1^*)=l(v')+w(v',\bar{v})\geq l(\bar{v})$ (нерівність зумовлена кроком 2). Оскільки $l(\bar{v})$ — тимчасова мітка, а постійну мітку $l(v)$ вершини v вибрано на k-й ітерації як найменшу з тимчасових (крок 3), то $l(\bar{v})\geq l(v)$. Об'єднавши дві останні нерівності, одержимо $w(P^*)\geq w(P_1^*)\geq l(v)=w(P^0)$, тобто $w(P^*)\geq w(P^0)$. Але за означенням $w(P^*)\leq w(P^0)$; отже, $w(P^*)=w(P^0)$, тобто P^0 — найкоротший шлях від a до v.

Випадок 2. Усі вершини шляху P^* містяться в множині M. Нехай v' та v'' — вершини, які безпосередньо передують вершині v відповідно в шляхах P^* та P^0. Позначимо як P' частину шляху P^* від початкової вершини a до вершини v'. За припущеннями індукції $w(P')\geq l(v')$. Якщо $v'=v''$, то

$$w(P^*)=w(P')+w(v',v)\geq l(v')+w(v',v)=w(P^0).$$

Припустимо тепер, що $v'\neq v''$. Позаяк v одержує постійну мітку $l(v)$ з v'', а не з v', то

$$w(P^*)=l(v')+w(v',v)\geq l(v'')+w(v'',v)=w(P^0).$$

Отже, і у випадку 2 справджується нерівність $w(P^*)\geq w(P^0)$, тобто P^0 — найкоротший шлях від a до v.

Якщо граф подано матрицею суміжності, складність алгоритму Дейкстри становить $O(n^2)$. Коли кількість дуг m значно менша, ніж n^2, то найкраще подавати орієнтований граф списками суміжності. Тоді алгоритм можна реалізувати зі складністю $O(m\log n)$, що в цьому разі істотно менше, ніж $O(n^2)$ [2, 23].

Алгоритм Дейкстри дає змогу обчислити довжину найкоротшого шляху від початкової вершини a до заданої вершини z. Для знаходження самого шляху потрібно лише нагромаджувати вектор вершин, з яких найкоротший шлях безпосередньо потрапляє в дану вершину. Для цього з кожною вершиною v графа G, окрім вершини a, зв'язують іще одну мітку — $\theta(v)$. Крок 2 модифікують так. Для кожної вершини $v\in\Gamma(x)\setminus M$ якщо $l(v)>l(x)+w(x,v)$, то $l(v):=l(x)+w(x,v)$ та $\theta(v):=x$, а ні, то не змінювати $l(v)$ та $\theta(v)$. Коли мітка $l(v)$ стане постійною, найкоротший $\langle a,v\rangle$-шлях буде потрапляти у вершину v безпосередньо з вершини x. Із постійних міток $l(v)$ та $\theta(v)$ утворюємо вектори l і θ.

Приклад 3.33. Знайдемо довжину найкоротшого шляху від початкової вершини a до вершини z у графі на рис. 3.37, *а*. Послідовність дій зображено на рис. 3.37, *б–е*, мітки записано в дужках біля вершин. Вершини, які включено в множину M, обведено кружечками, мітки таких вершин оголошують постійними. У процесі роботи алгоритму будують два вектори: вектор l постійних міток (довжини найкоротших шляхів від вершини a до даної вершини) і вектор θ вершин, з яких у дану вершину безпосередньо потрапляє найкоротший шлях. У табл. 3.4 в першому рядку містяться довільно впорядковані вершини графа, у другому — відповідні постійні мітки (компоненти вектора l), а в третьому — компоненти вектора θ. Постійна мітка вершини z дорівнює 13. Отже, довжина найкоротшого шляху від a до z дорівнює 13. Сам шлях знаходять за допомогою першого й третього рядків таблиці та будують у зворотному порядку. Кінцева вершина — z; у неї потрапляємо з вершини e (див. вектор θ). У вершину e потрапляємо з вершини d, у d — з b та продовжуємо цей процес до вершини a: $z \leftarrow e \leftarrow d \leftarrow b \leftarrow c \leftarrow a$. Отже, найкоротший шлях такий: *a. c. b. d. e. z.*

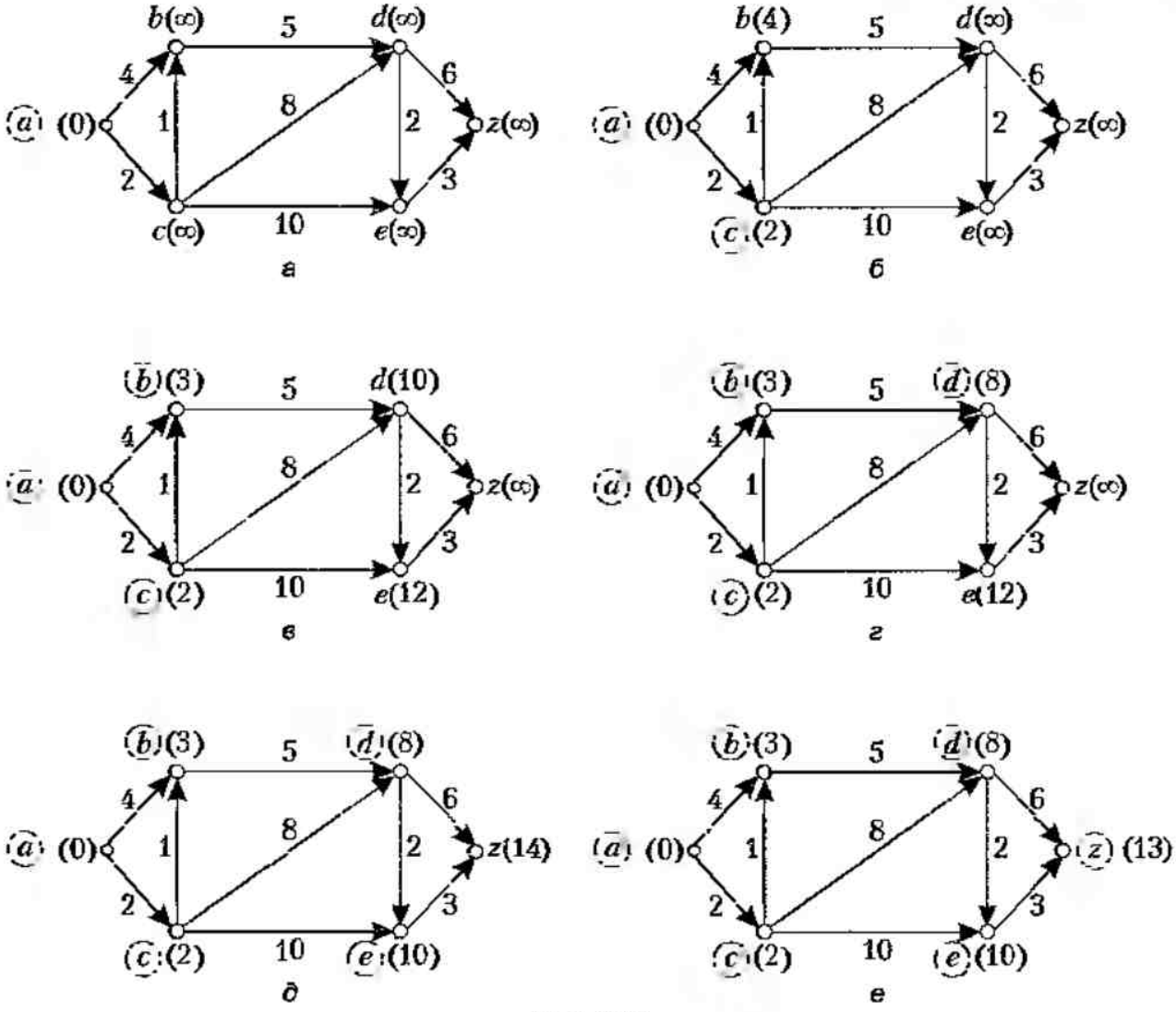


Рис. 3.37

Таблиця 3.4

Вершини графа (елементи множини V)	a	b	c	d	e	z
Вектор l (постійні мітки вершин)	0	3	2	8	10	13
Вектор θ (вершини, з яких у дану вершину заходить найкоротший шлях)	–	c	a	b	d	e

Ми розглянули задачу відшукання в графі найкоротшого шляху від якоїсь виділеної (початкової) вершини до будь-якої іншої. Розглянемо задачу пошуку в графі найкоротшого шляху між кожною парою вершин. Звичайно, цю загальнішу задачу можна розв'язати багатократним застосуванням алгоритму Дейкстри з послідовним вибором кожної вершини графа як початкової. Проте є й прямий спосіб розв'язання цієї задачі за допомогою алгоритму Флойда. У ньому довжини дуг можуть бути від'ємними, проте не може бути циклів із від'ємною довжиною.

Нехай $G = (V, \Gamma)$ — орієнтовний граф. Нагадаємо, що внутрішні вершини шляху $a, x_1, x_2, ..., x_{n-1}, b$ в графі G — $x_1, x_2, ..., x_{n-1}$. Наприклад, внутрішні вершини шляху a, c, d, a, f, b — c, d, a, f. Пронумеруємо вершини графа цілими числами від 1 до n. Позначимо як $w_{ij}^{(k)}$ довжину найкоротшого шляху з вершини i у вершину j, у якому як внутрішні можуть бути лише перші k вершин графа G. Якщо між вершинами i та j не існує жодного такого шляху, то умовно вважатимемо, що $w_{ij}^{(k)} = \infty$. Зі сказаного випливає, що $w_{ij}^{(0)}$ — це вага дуги (i, j), а якщо такої дуги немає, то $w_{ij}^{(0)} = \infty$. Для довільної вершини i вважатимемо $w_{ii}^{(0)} = 0$. Отже $w_{ij}^{(n)}$ дорівнює довжині найкоротшого шляху з вершини i у вершину j. Позначимо як $W^{(k)}$ $n \times n$ матрицю (i, j)-й елемент якої дорівнює $w_{ij}^{(k)}$. Якщо в заданому орієнтованому графі G відома вага кожної дуги, то, виходячи з попередніх міркувань, можна сформувати матрицю $W^{(0)}$. Тепер потрібно визначити матрицю $W^{(n)}$, елементи якої дорівнюють довжинам найкоротших шляхів між усіма парами вершин графа G.

В алгоритмі Флойда як початкову беруть матрицю $W^{(0)}$. Спочатку за нею обчислюють матрицю $W^{(1)}$, потім — $W^{(2)}$, і процес повторюють доти, доки за матрицею $W^{(n-1)}$ не буде обчислено матрицю $W^{(n)}$. Розглянемо ідею, на якій ґрунтується алгоритм Флойда. Припустимо, що відомі:

1. Найкоротший шлях із вершини i у вершину k, у якому як внутрішні використано лише перші $(k-1)$ вершин.
2. Найкоротший шлях із вершини k у вершину j, у якому як внутрішні використано лише перші $(k-1)$ вершин.
3. Найкоротший шлях із вершини i у вершину j, у якому як внутрішні використано лише перші $(k-1)$ вершин.

Оскільки за припущенням граф G не містить циклів із від'ємною довжиною, то один із двох шляхів — шлях із п. 3 чи об'єднання шляхів із пп. 1 і 2 — найкоротший шлях із вершини i у вершину j, у якому як внутрішні використано лише перші $(k-1)$ вершин. Отже,

$$w_{ij}^{(k)} = \min\{w_{ik}^{(k-1)} + w_{kj}^{(k-1)}, w_{ij}^{(k-1)}\}. \qquad (3.1)$$

Зі співвідношення (3.1) видно, що для обчислення елементів матриці $W^{(k)}$ потрібні тільки елементи матриці $W^{(k-1)}$. Тепер ми можемо формально описати алгоритм Флойда для знаходження в графі найкоротших шляхів між усіма парами вершин.

Алгоритм Флойда

Наведемо кроки алгоритму.

Крок 1. Присвоювання початкових значень. Пронумерувати вершини графа G цілими числами від 1 до n. Побудувати матрицю $W^{(0)} = W$, задавши кожний її (i, j)-елемент такий, що дорівнює вазі дуги, котра з'єднує вершину i з вершиною j. Якщо в графі G ці вершини не з'єднано дугою, то виконати $w_{ij}^{(0)} := \infty$. Крім того, для всіх i виконати $w_{ii}^{(0)} := 0$.

Крок 2. Цикл. Для k, що послідовно набуває значення 1, 2, ..., n, визначити за елементами матриці $W^{(k-1)}$ елементи матриці $W^{(k)}$, використовуючи рекурентне співвідношення (3.1).

Після закінчення цієї процедури (i, j)-елемент матриці $W^{(n)}$ дорівнює довжині найкоротшого шляху з вершини i у вершину j.

Якщо під час роботи алгоритму для якихось k й i виявиться, що $w_{ii}^{(k)} < 0$, то в графі G існує цикл із від'ємною довжиною, який містить вершину i. Тоді роботу алгоритму потрібно припинити.

Якщо заздалегідь відомо, що в графі G немає циклів із від'ємною довжиною, то обсяг обчислень можна дещо зменшити. У цьому разі для всіх i та всіх k має бути $w_{ii}^{(k)} = 0$. Тому не потрібно обчислювати діагональні елементи матриць $W^{(1)}$, $W^{(2)}$, ..., $W^{(n)}$. Окрім того, для всіх $i = 1, 2, ..., n$ справджуються співвідношення $w_{ik}^{(k-1)} = w_{ik}^{(k)}$, $w_{ki}^{(k-1)} = w_{ki}^{(k)}$, які випливають із того, що коли немає циклів із від'ємною довжиною, вершина k не може бути внутрішньою в будь-яких найкоротших шляхах, котрі починаються чи закінчуються в самій вершині k. Отже, обчислюючи матрицю $W^{(k)}$, немає потреби переобчислювати елементи k-го рядка й k-го стовпця матриці $W^{(k-1)}$. Отже, у матриці $W^{(k)}$ за формулою (3.1) потрібно обчислювати лише $n^2 - 3n + 2$ елементи. Очевидно, що складність алгоритму Флойда становить $O(n^3)$.

Щоб після закінчення його роботи можна було швидко знайти найкоротший шлях між будь-якою парою вершин, на k-й ітерації разом із матрицею $W^{(k)}$ побудуємо матрицю $\Theta^{(k)} = [\theta_{ij}^{(k)}]$. Спочатку беремо $\theta_{ij}^{(0)} := i$ для всіх $i, j = 1, ..., n$, $i \neq j$; $\theta_{ii}^{(0)} := 0$. Далі, на k-й ітерації візьмемо $\theta_{ij}^{(k)} = \theta_{ij}^{(k-1)}$, якщо $w_{ij}^{(k)} = w_{ij}^{(k-1)}$, і $\theta_{ij}^{(k)} = \theta_{kj}^{(k-1)}$, якщо $w_{ij}^{(k)} = w_{ik}^{(k-1)} + w_{kj}^{(k-1)}$. Отже, $\theta_{ij}^{(k)}$ — номер вершини, яка є перед вершиною j у поточному $\langle i, j\rangle$-шляху, тобто найкоротшому $\langle i, j\rangle$-шляху, усі внутрішні вершини якого містяться в множині $\{1, 2, ..., k\}$. Матриця $\Theta^{(n)} = [\theta_{ij}^{(n)}]$ відіграє ту саму роль, що й вектор θ в алгоритмі Дейкстри. За допомогою матриці $\Theta^{(n)}$ вершини, через які проходить найкоротший $\langle i, j\rangle$-шлях, визначають так: $i, ..., j_3, j_2, j_1, j$, де $j_1 = \theta_{ij}^{(n)}$, $j_2 = \theta_{ij_1}^{(n)}$, $j_3 = \theta_{ij_2}^{(n)}$, ...

Приклад 3.34. Знайти найкоротші шляхи між усіма парами вершин орієнтованого графа на рис. 3.38.

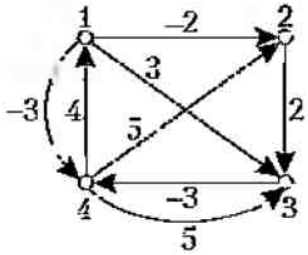


Рис. 3.38

Нижче наведено результати виконання кожної з чотирьох ітерацій алгоритму Флойда:

$$W^{(0)} = W = \begin{pmatrix} 0 & -2 & 3 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 5 & 5 & 0 \end{pmatrix}; \quad \Theta^{(0)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 4 & 4 & 0 \end{pmatrix};$$

$$W^{(1)} = \begin{pmatrix} 0 & -2 & 3 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 5 & 0 \end{pmatrix}; \quad \Theta^{(1)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 4 & 0 \end{pmatrix};$$

$$W^{(2)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}, \quad \Theta^{(2)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix},$$

$$W^{(3)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & -1 \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; \quad \Theta^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 3 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix};$$

$$W^{(4)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ 3 & 0 & 2 & -1 \\ 1 & -1 & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; \quad \Theta^{(4)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 4 & 0 & 2 & 3 \\ 4 & 1 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix}.$$

Матриця $W^{(4)}$ дає довжини найкоротших шляхів між усіма парами вершин графа на рис. 3.38. Зокрема, $a_{21}^{(4)} = 3$, тобто довжина найкоротшого шляху від вершини 2 до вершини 1 дорівнює 3. Для знаходження самого шляху скористаємося матрицею $\Theta^{(4)}$ та запишемо у зворотному порядку вершини, через які він проходить: $\theta_{21}^{(4)} = 4$, $\theta_{24}^{(4)} = 3$, $\theta_{23}^{(4)} = 2$. Отже, найкоротший шлях із вершини 2 у вершину 1 проходить через вершини 2, 3, 4, 1.

Очевидно, що як алгоритм Дейкстри, так і алгоритм Флойда можна застосовувати без жодних змін і до неорієнтованих графів. Для цього достатньо кожне ребро $\{u, v\}$, що має вагу $w(u, v)$, розглядати як пару дуг (u, v) та (v, u) з тією самою вагою. Слід ураховувати, що неорієнтоване ребро із від'ємною вагою одразу приводить до циклу із від'ємною довжиною, що робить алгоритм Флойда незастосовним.

3.9. Обхід графів

Існує багато алгоритмів на графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина одержує унікальний порядковий номер. Алгоритми обходу вершин графа називають *методами пошуку* [23].

3.9.1. Пошук углиб у простому зв'язному графі

Опишемо метод пошуку в простому зв'язному графі, який являє собою одну з основних методик проектування алгоритмів, пов'язаних із графами. Цей метод називають *пошуком углиб*, або DFS-методом (від англ. depth first search).

Нехай $G = (V, E)$ — простий зв'язний граф, усі вершини якого позначено попарно різними символами. У процесі пошуку вглиб вершинам графа G надають номери (DFS-номери) та певним способом позначають ребра. У ході роботи алгоритму використовують структуру даних для збереження множин, яку називають *стеком* (англ. stack — стіг). Зі стеку можна вилучити тільки той елемент, котрий було додано до нього останнім: стек працює за принципом „останнім прийшов — першим вийшов" (last in, first out — скорочено LIFO). Інакше кажучи, додавання й вилучення елементів у стеку відбувається з одного кінця, який називають *верхівкою стеку*. DFS-номер вершини x позначають DFS(x).

Алгоритм пошуку вглиб у простому зв'язному графі

Наведемо кроки алгоритму.

Крок 1. Почати з довільної вершини v_s. Виконати DFS(v_s) := 1. Включити цю вершину в стек.

Крок 2. Розглянути вершину у верхівці стеку: нехай це вершина x. Якщо всі ребра, інцидентні вершині x, позначено, то перейти до кроку 4, інакше — до кроку 3.

Крок 3. Нехай $\{x, y\}$ — непозначене ребро. Якщо DFS(y) уже визначено, то позначити ребро $\{x, y\}$ штриховою лінією та перейти до кроку 2. Якщо DFS(y) не визначено, то позначити ребро $\{x, y\}$ потовщеною суцільною лінією, визначити DFS(y) як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.

Крок 4. Виключити вершину x зі стеку. Якщо стек порожній, то зупинитись, інакше — перейти до кроку 2.

Щоб вибір номерів був однозначним, домовились, що вершини, суміжні з тією, яка вже отримала DFS-номер, аналізують за зростанням їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму зручно відображати за допомогою таблиці з трьома стовпцями: вершина, DFS-номер, уміст стеку. Її називають *протоколом обходу* графа пошуком вглиб.

Приклад 3.35. Виконаємо обхід графа на рис. 3.39 пошуком вглиб, починаючи з вершини b. Розв'язок подано на рис. 3.40; протокол такого пошуку подано в табл. 3.5. У цій таблиці в третьому стовпці вважаємо, що верхівка стеку праворуч.

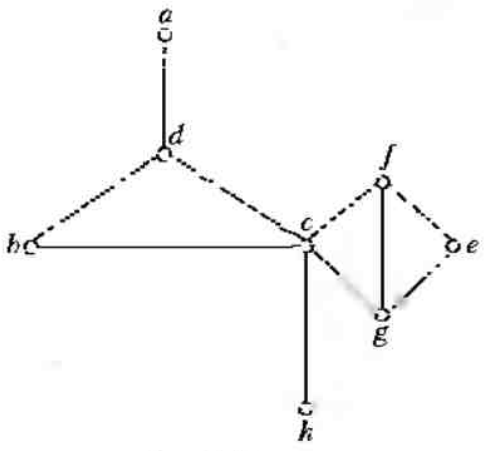


Рис. 3.39

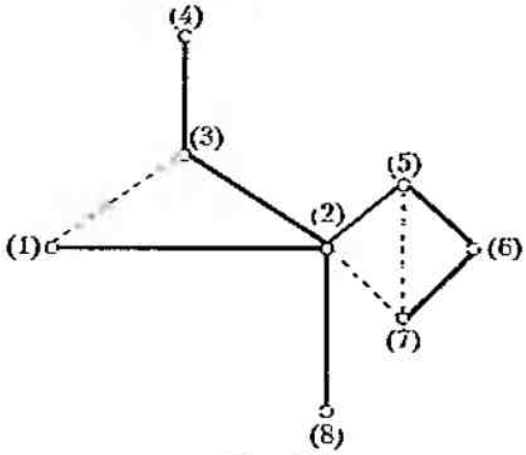


Рис. 3.40

Таблиця 3.5

Вершина	DFS-номер	Вміст стеку
b	1	*b*
c	2	*bc*
d	3	*bcd*
a	4	*bcda*
–	–	*bcd*
–	–	*bc*
f	5	*bcf*
e	6	*bcfe*
g	7	*bcfeg*
–	–	*bcfe*
–	–	*bcf*
–	–	*bc*
h	8	*bch*
–	–	*bc*
–	–	*b*
–	–	∅

3.9.2. Пошук ушир у простому зв'язному графі

У процесі *пошуку вшир* вершини графа проглядають в іншій послідовності, ніж у методі пошуку вглиб, і їм надають BFS-номери (від англ. breadth first search). BFS-номер вершини x позначають BFS(x). Під час пошуку рухаються вшир, а не вглиб: спочатку проглядають усі сусіди вершини, після цього — сусіди сусідів і так далі.

У ході реалізації алгоритму використовують структуру даних для збереження множин, яку називають *чергою* (англ. queue). Із черги можна вилучити тільки той елемент, який перебував у ній найдовше: працює принцип „першим прийшов — першим вийшов" (first in, first out — скорочено FIFO). Елемент включається у *хвіст* черги, а виключається з її *голови*. Пошук ушир, узагалі кажучи, відрізняється від пошуку вглиб заміною стеку на чергу. Після такої модифікації що раніше відвідується вершина (включається в чергу), то раніше вона використовується (і виключається з черги). Використання вершини полягає в перегляді одразу всіх іще не відвіданих її сусідів. Усю процедуру подано нижче.

Алгоритм пошуку вшир у простому зв'язному графі

Наведемо кроки алгоритму.

Крок 1. Почати з довільної вершини v_s. Виконати BFS(v_s):= 1. Включити вершину v_s у чергу.

Крок 2. Розглянути вершину, яка перебуває на початку черги; нехай це буде вершина x. Якщо для всіх вершин, суміжних із вершиною x, уже визначено BFS-номери, то перейти до кроку 4, інакше — до кроку 3.

Крок 3. Нехай $\{x, y\}$ — ребро, у якому номер BFS(y) не визначено. Позначити це ребро потовщеною суцільною лінією, визначити BFS(y) як черговий BFS-номер, включити вершину y у чергу й перейти до кроку 2.

Крок 4. Виключити вершину x із черги. Якщо черга порожня, то зупинитись, інакше — перейти до кроку 2.

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною x, аналізують за зростанням їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму пошуку вшир також зручно відображати за допомогою протоколу обходу. Він аналогічний попередньому й відрізняється лише третім стовпцем: тепер це — уміст черги (уважаємо, що голова черги ліворуч, а хвіст — праворуч).

Приклад 3.36. Виконаємо обхід графа на рис. 3.39 пошуком вшир, починаючи з вершини b. Розв'язок зображено на рис. 3.41, протокол пошуку вшир подано в табл. 3.6.

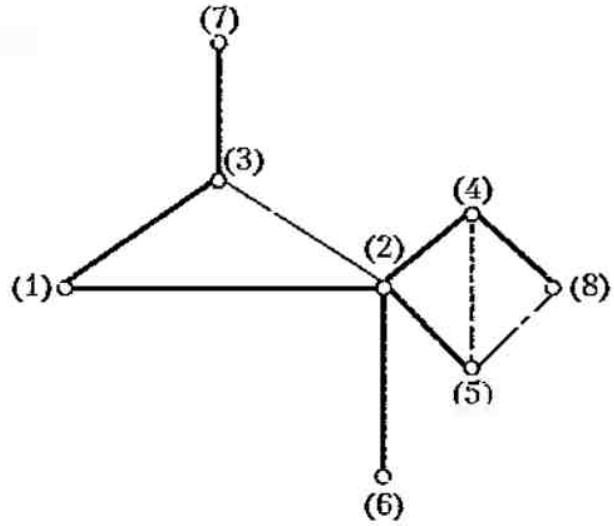


Рис. 3.41

Таблиця 3.6

Вершина	BFS-номер	Вміст черги
b	1	b
c	2	bc
d	3	bcd
-	–	cd
f	4	cdf
g	5	$cdfg$
h	6	$cdfgh$
–	–	$dfgh$
a	7	$dfgha$
–	–	$fgha$
e	8	$fghae$
–	–	$ghae$
–	–	hae
–	–	ae
–	–	e
–	–	∅

У процесі роботи наведених алгоритмів будується *дерево T пошуку* відповідно *вглиб* і *вшир* — на рис. 3.40 і 3.41 його виділено потовщеними лініями (див. розділ 4).

Обчислювальна складність обох алгоритмів обходу однакова й у разі подання графа списками суміжності становить $O(m+n)$, де m — кількість ребер, а n — кількість вершин графа.

3.10. Планарні графи

Розглянемо неорієнтовані графи. Часто не має значення, як зобразити граф на рисунку, бо ізоморфні графи несуть одну й ту саму інформацію. Проте інколи важливо, чи можна подати граф на площині так, щоб його зображення задовольняло певним вимогам. Наприклад, у радіоелектроніці в процесі виготовлення мікросхем друкованим способом електричні ланцюги наносять на плоску поверхню ізоляційного матеріалу. Оскільки провідники не ізольовані, то вони не мають перетинатись. Аналогічна задача виникає під час проектування залізничних та інших шляхів, де переїзди небажані. Так виникає поняття плоского графа.

Плоским називають граф, зображений на площині так, що ніякі два його ребра геометрично не перетинаються ніде, окрім інцидентних їм вершин. Граф, ізоморфний до плоского графа, називають *планарним*.

Приклад 3.37. Усі три графи на рис. 3.42 планарні, але лише другий і третій із них плоскі.

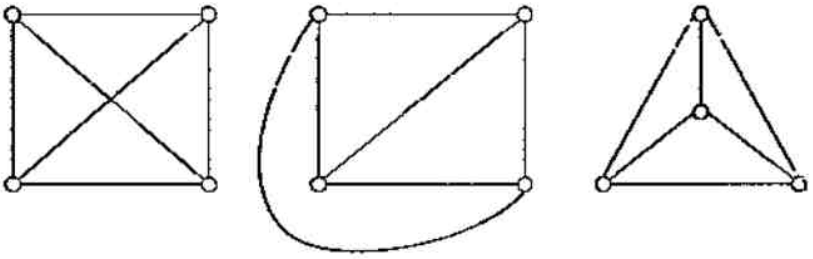


Рис. 3.42

Плоский граф розбиває площину на області, одна з яких необмежена. Їх називають *гранями* плоского графа; необмежену область називають *зовнішньою гранню*.

Приклад 3.38. На рис. 3.43 зображено плоский граф. Він має чотири грані: r_1, r_2, r_3, r_4; грань r_4 зовнішня.

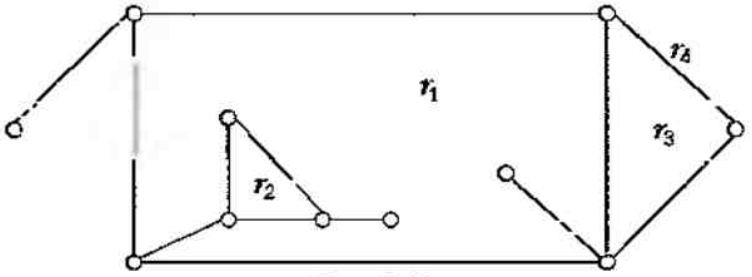


Рис. 3.43

ТЕОРЕМА 3.14 (Ейлера про плоскі графи). Нехай зв'язний плоский граф G має n вершин, m ребер і r граней. Тоді $n + r = m + 2$.

Доведення проводимо індукцією за кількістю ребер у графі G. Якщо $m = 0$, то $n = 1$, $r = 1$, і теорема справджується. Допустимо, що теорема справджується для довільного плоского графа G, який має $m - 1$ ребро, і додамо до G нове ребро e. Можливі три випадки.

1. Ребро e — петля; тоді виникне нова грань, а кількість вершин залишиться незмінною.
2. Ребро e з'єднує дві різні вершини графа G; у такому разі одна з граней розпадеться на дві, тому кількість граней збільшиться на одну, а кількість вершин не зміниться.
3. Ребро e інцидентне лише одній вершині в G; тоді потрібно додати ще одну вершину; отже, кількість вершин збільшиться на одну, а кількість граней не зміниться.

Твердження теореми залишається правильним у кожному з цих випадків. Оскільки інші випадки неможливі, то індукцію завершено й теорему доведено.

ТЕОРЕМА 3.15. Графи K_5 і $K_{3,3}$ непланарні.

Доведення. Граф K_5 має п'ять вершин і десять ребер (див. рис. 3.9). Припустимо, що він планарний; тоді існує ізоморфний до нього плоский граф. За теоремою Ейлера $r = m + 2 - n = 10 + 2 - 5 = 7$. Зазначимо, що будь-яке ребро плоского графа або розділяє дві різні грані, або являє собою міст. Позаяк граф K_5 не має петель і кратних ребер, то кожна грань обмежена принаймні трьома ребрами. Тому число $3r$ — оцінка знизу подвоєної кількості ребер графа, тобто $3r \leqslant 2m$, звідки випливає, що $21 \leqslant 20$. Суперечність.

У графі $K_{3,3}$ кількість вершин $n = 6$, а кількість ребер $m = 9$ (див. рис. 3.10). Припустимо, що він планарний. Тоді в ізоморфному до нього плоскому графі за теоремою Ейлера кількість граней $r = m + 2 - n = 9 + 2 - 6 = 5$. Будь-яка грань дводольного графа має бути обмежена принаймні чотирма ребрами. Отже, $4r \leqslant 2m$, звідки випливає, що $10 \leqslant 9$. Суперечність.

Два графи називаються *гомеоморфними*, якщо їх можна отримати з одного графа додаванням до його ребер нових вершин степеня 2 (рис. 3.44).

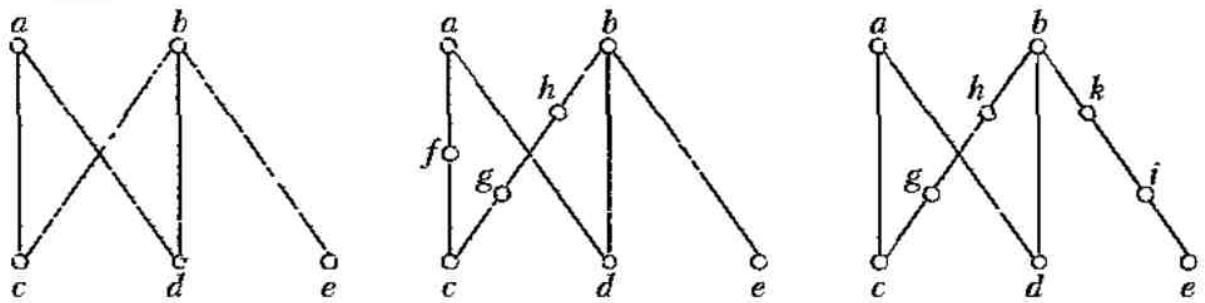


Рис. 3.44

Наступна теорема дає критерій (необхідну й достатню умову) планарності графа.

ТЕОРЕМА 3.16 (Куратовського). Граф планарний тоді й лише тоді, коли він не містить підграфів, гомеоморфних графам K_5 або $K_{3,3}$.

Необхідність умов теореми вже доведено, бо доведено непланарність графів K_5 і $K_{3,3}$, а доведення достатності складне, і ми його не наводимо.

Приклад 3.39. На рисунку 3.45 зображено *граф Петерсена*, а на рис 3.46 — його підграф, гомеоморфний $K_{3,3}$. Отже за теоремою Куратовського, граф Петерсена непланарний.

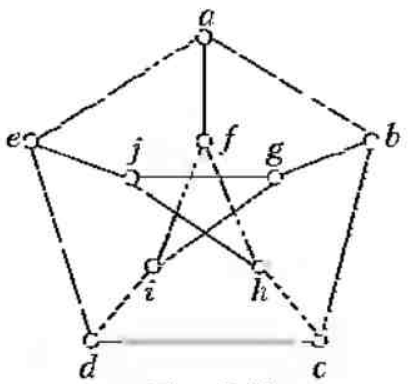


Рис. 3.45

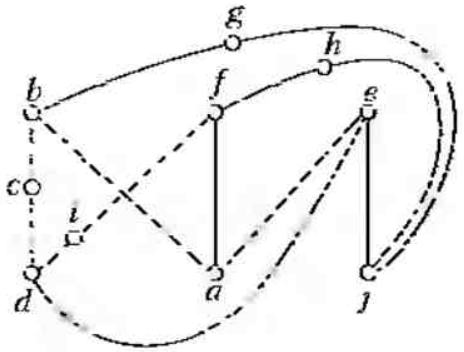


Рис. 3.46

Окрім теореми Куратовського є й інші критерії планарності графів. Практично перевірити умови, якими характеризуються планарні графи, не завжди просто. Проте розроблено ефективні алгоритми, які дають змогу для будь-якого заданого графа знайти його зображення на площині без перетину ребер або переконатись, що це неможливо (якщо граф непланарний) [15].

3.11. Розфарбовування графів

У цьому підрозділі розглянуто лише прості графи. *Розфарбовуванням простого графа* G називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають *хроматичним числом* і позначають $\chi(G)$. Очевидно, що існує розфарбування графа G в k кольорів (*k-розфарбування*) для будь-якого k в діапазоні $\chi(G) \leq k \leq n$, де n — кількість вершин графа. Множину вершин, розфарбованих в один колір, називають *одноколірним класом*. Такі класи утворюють незалежні множини вершин, тобто ніякі дві вершини в одноколірному класі не суміжні (про незалежні множини вершин див. підрозділ 3.12).

Очевидно, що $\chi(K_n) = n$, і, отже, легко побудувати графи з як завгодно великим хроматичним числом. З іншого боку, $\chi(G) = 1$ тоді й лише тоді, коли G - вироджений граф; $\chi(G) = 2$ тоді й лише тоді, коли G — дводольний граф (тому дводольний граф називають *бінарним*).

ТЕОРЕМА 3.17. Якщо найбільший зі степенів вершин графа дорівнює ρ, то цей граф можна розфарбувати в $\rho + 1$ колір.

Доведення. Застосуємо індукцію за кількістю вершин графа. Нехай граф G має n вершин; вилучимо з нього довільну вершину v разом з усіма інцидентними їй ребрами. Отримаємо граф з $n-1$ вершиною; степінь кожної вершини не більший ніж ρ. За припущенням індукції цей граф можна розфарбувати в $\rho+1$ колір. Отже, у $\rho+1$ колір можна розфарбувати й граф G, якщо розфарбувати вершину v кольором, що відрізняється від тих, якими розфарбовано суміжні з нею вершини (а їх разом не більше ніж ρ).

Із середини XIX ст. відкритою залишалася проблема, відома під назвою гіпотези чотирьох фарб. Її формулюють так: будь-який планарний граф можна розфарбувати в чотири кольори, тобто $\chi(G) \leqslant 4$ для будь-якого планарного графа G.

Перше з помилкових „доведень" належить А. Кемпе (1879 р.), але помилку було виявлено не відразу. Її знайшов Р. Хейвуд 1890 р. й тоді ж довів, що будь-який планарний граф можна розфарбувати в п'ять кольорів.

ТЕОРЕМА 3.18 (Р. Хейвуда). Будь-який планарний граф можна розфарбувати в п'ять кольорів, тобто $\chi(G) \leqslant 5$ для будь-якого планарного графа G.

Доведення цієї теореми ґрунтується на тому, що в будь-якому простому планарному графі є вершина, степінь якої не більший ніж п'ять. Цей результат легко одержати як наслідок із теореми Ейлера про плоскі графи (див. задачу 65).

Застосуємо математичну індукцію за кількістю вершин графа. Теорема справджується для графів із не більше ніж п'ятьма вершинами. Припустимо, що вона справджується для графів із не більше ніж n вершинами, де $n \geqslant 5$. Розглянемо довільний плоский граф G з $(n+1)$ вершиною. Він містить вершину v_0, степінь якої не більший ніж п'ять. Нехай $W = \Gamma(v_0)$ — множина вершин, суміжних із вершиною v_0 в графі G. Окремо розглянемо два випадки.

Випадок 1. $|W| \leqslant 4$. Позначимо як $G - v_0$ граф, отриманий із графа G вилученням вершини v_0 та всіх інцидентних їй ребер. За індуктивним припущенням граф $G - v_0$ можна розфарбувати в п'ять кольорів. Зафіксуємо одне з таких розфарбувань і зафарбуємо вершину v_0 в той із п'яти кольорів, який не використано для фарбування вершин із множини W.

Випадок 2. $|W| = 5$. У множині W є дві несуміжні вершини v_1 і v_2, а ні, то граф $G(W)$, породжений множиною вершин W, — це K_5, і тоді граф G непланарний. Граф G', одержаний із графа $G - v_0$ злиттям вершин v_1 і v_2 в одну вершину v (рис. 3.47), плоский, і за індуктивним припущенням його можна розфарбувати в п'ять кольорів. Зафіксуємо одне з таких розфарбувань графа G'. Тепер у графі G розфарбуємо вершини v_1 і v_2 в колір вершини v, а решту вершин, відмінних від v_0, — у ті самі кольори, що й відповідні вершини графа G'. Нарешті, пофарбуємо вершину v_0 колір, не використаний для розфарбовування вершин із W.

Американські математики К. Аппель (K. Appel) і У. Гайкен (W. Haken) 1976 р. довели гіпотезу чотирьох фарб, суттєво використавши комп'ютерні обчислення. Це перший випадок, коли настільки відому математичну проблему було розв'язано за допомогою комп'ютера [21]. Спочатку проблему було зведено до розгляду скінченної (хоча й великої) кількості випадків. Надалі список „підозрілих" графів було зменшено й за допомогою комп'ютера розфарбовано в чотири кольори

планарні графи з цього списку. За різними джерелами [15, 21], 1976 р. це можна було зробити за 1500-2000 годин роботи потужного комп'ютера. Отже, проблему чотирьох фарб було розв'язано. Хоча сам по собі метод доведення являє собою видатне досягнення й цілком достатній, щоб припинити пошуки контрприкладу, було б добре, щоб хтось знайшов елегантніше доведення гіпотези. Найцікавіше в цьому доведенні те, що воно розширило наше уявлення про математичне доведення. Проте не всі поділяють цю думку. Опоненти вважають, що важко погодитись із таким доведенням, бо й зведення загального випадку до скінченної множини графів, і розфарбовування останніх дуже важко повторити [15].

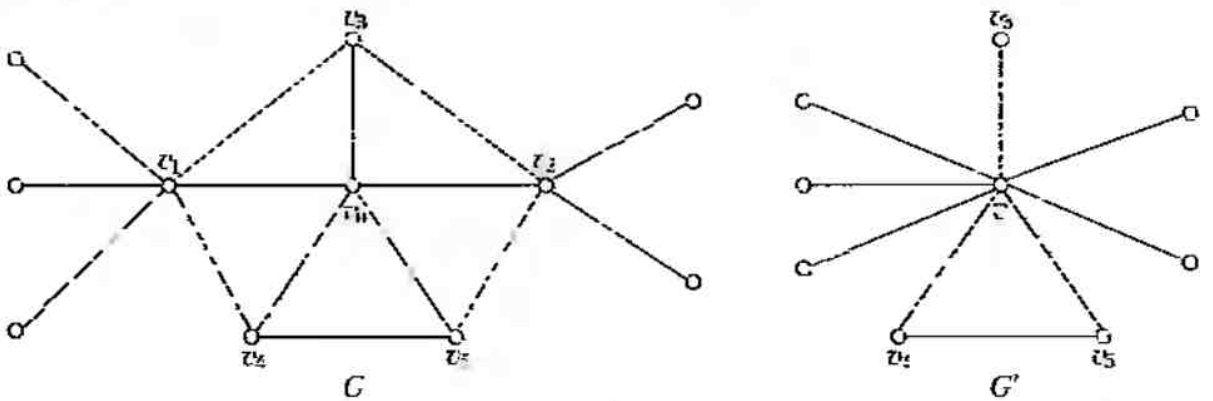


Рис. 3.47

Нехай G — простий граф, а $P_G(k)$ дорівнює кількості способів, якими можна розфарбувати вершини графа G в k кольорів. $P_G(k)$ називають *хроматичною функцією* графа G. Для повного графа K_n виконується рівність $P_G(k) = k(k-1)(k-2)\ldots$ $\ldots(k-n+1)$. Справді, для першої вершини є k варіантів кольору. Оскільки друга вершина суміжна з першою, то її можна розфарбувати в один з $(k-1)$ кольорів. Третя вершина суміжна з першою та другою, тому для неї є $(k-2)$ варіантів вибору. Продовжуючи ці міркування, доходимо висновку, що для останньої вершини залишиться $k-(n-1)$ варіантів вибору кольору.

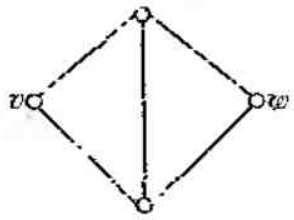
Якщо $k < \chi(G)$, то $P_G(k) = 0$, а для $k \geq \chi(G)$ виконується нерівність $P_G(k) > 0$. Гіпотеза чотирьох фарб еквівалентна такому твердженню: якщо G — простий планарний граф, то $P_G(4) > 0$. Якщо задано довільний простий граф, то в загальному випадку важко отримати його хроматичну функцію за допомогою простих міркувань. Наступна теорема й наслідок із неї дають систематичний метод одержання хроматичної функції простого графа у вигляді суми хроматичних функцій повних графів.

ТЕОРЕМА 3.19. *Нехай G — простий граф, а v та w — його несуміжні вершини.* Якщо граф G_1 отримано з G за допомогою з'єднання вершин v та w ребром, а граф G_2 — ототожненням вершин v та w (і кратних ребер, якщо їх буде одержано), то $P_G(k) = P_{G_1}(k) + P_{G_2}(k)$.

Доведення. За будь-якого допустимого розфарбовування вершин графа G існує альтернатива: v та w мають або різні кольори, або той самий. Кількість розфарбовувань, за яких v та w мають різні кольори, не зміниться, якщо долучити ребро $\{v, w\}$. Отже, ця кількість дорівнює $P_{G_1}(k)$. Аналогічно, кількість розфарбовувань, за яких v та w мають один колір, не зміниться, якщо ототожнити v та w.

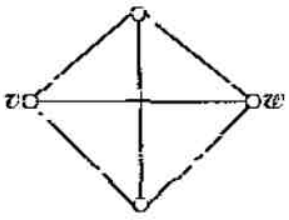
Отже, ця кількість дорівнює $P_{G_2}(k)$. Залишилося застосувати комбінаторне правило суми. Теорему доведено.

Приклад 3.40. На рис. 3.48 зображено граф G, а на рис 3.49 — графи G_1 і G_2.

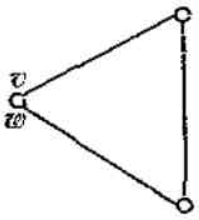


G

Рис. 3.48



G_1



G_2

Рис. 3.49

Наслідок. Хроматична функція простого графа — поліном.

Тепер ми будемо називати $P_G(k)$ *хроматичним поліномом*. Зазначимо деякі властивості хроматичного полінома. Якщо граф G має n вершин, то степінь полінома $P_G(k)$ дорівнює n. Коефіцієнт при k^n дорівнює 1, а при k^{n-1} дорівнює $-m$, де m — кількість ребер графа G; знаки коефіцієнтів чергуються; вільний член хроматичного полінома дорівнює 0.

Хроматичний поліном будують на основі теореми 3.19 у вигляді суми хроматичних поліномів повних графів.

Приклад 3.41. На рис 3.50 зображено процес побудови хроматичного полінома, де знаки „ = " та „ + " мають умовний зміст. Отже,

$$P_G(k) = k(k-1)(k-2)(k-3) + 2k(k-1)(k-2) + k(k-1) = k^4 - 4k^3 + 6k^2 - 3k.$$

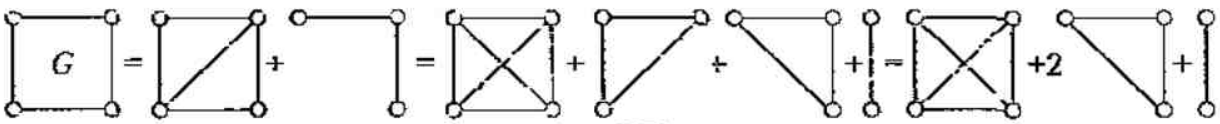


Рис. 3.50

Нарешті, розглянемо деякі практичні задачі, які зводяться до розфарбовування графів.

Задача складання розкладу

Припустимо, що потрібно прочитати декілька лекцій у найкоротший термін. Кожна лекція окремо займає одну годину, але деякі лекції не можна читати одночасно (наприклад, якщо це робить один і той самий лектор). Побудуємо граф G, вершини якого взаємно однозначно відповідають лекціям, і дві вершини суміжні тоді й лише тоді, коли відповідні лекції не можна читати одночасно. Очевидно, що будь-яке розфарбовування цього графа задає можливий розклад: лекції, які відповідають вершинам одноколірного класу, читають одночасно. Навпаки, будь-який можливий розклад задає розфарбовування графа G. Оптимальні розклади відповідають розфарбовуванням мінімальною кількістю кольорів, а час, потрібний для читання всіх лекцій, дорівнює $\chi(G)$.

Задача розподілу обладнання

Задано множини $V = \{v_1, ..., v_n\}$ і $S = \{s_1, ..., s_m\}$ відповідно робіт і механізмів. Для виконання кожної роботи потрібен певний час, однаковий для всіх робіт, і якісь

механізми. Жоден із механізмів не може бути зайнятий на декількох роботах одночасно. Потрібно розподілити механізми так, щоб загальний час виконання всіх робіт був мінімальним. Побудуємо граф G з множиною вершин V, вершини v_i та v_j $(i \neq j)$ якого суміжні тоді й лише тоді, коли для виконання робіт v_i та v_j потрібний хоча б один спільний механізм. Розфарбуємо граф G. Роботи, що відповідають вершинам одного кольору, можна виконувати одночасно, а мінімальний час виконання всіх робіт відповідає розфарбуванню мінімальною кількістю кольорів.

Задача призначення телевізійних каналів

Передавальні станції зображають вершинами графа. Якщо віддаль між будь-якими двома станціями менша за l, то відповідні вершини графа з'єднують ребром. Граф розфарбовують зіставляючи різним кольорам вершин різні канали. Мінімальна кількість каналів відповідає розфарбуванню графа мінімальною кількістю кольорів.

3.12. Незалежні множини вершин. Кліки

Нехай G — простий граф. Множину його вершин називають *незалежною* (або *внутрішньо стійкою*), якщо ніякі вершини цієї множини не суміжні. Незалежну множину називають *максимальною*, якщо вона не є підмножиною жодної іншої незалежної множини з більшою кількістю вершин.

Найпотужнішу максимальну незалежну множину називають *найбільшою*. Кількість вершин у найбільшій незалежній множині графа G називають *числом незалежності* (*числом внутрішньої стійкості*, *нещільністю*) і позначають $\alpha(G)$.

Приклад 3.42. У графі, зображеному на рис. 3.51, розглянемо множини вершин $Y_1 = \{v_7, v_8, v_2, v_5\}$, $Y_2 = \{v_7, v_8, v_2\}$, $Y_3 = \{v_1, v_3, v_7\}$, $Y_4 = \{v_4, v_6\}$. Максимальні незалежні множини — Y_1, Y_3 та Y_4. Множина вершин Y_2 незалежна, але не максимальна. Найбільша незалежна множина — $Y_1 = \{v_7, v_8, v_2, v_5\}$. Отже, $\alpha(G) = 4$.

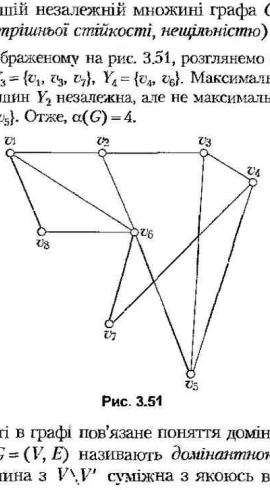


Рис. 3.51

Із поняттям незалежності в графі пов'язане поняття домінування [15]. Підмножину V' вершин графа $G = (V, E)$ називають *домінантною* (або *зовнішньо стійкою*), якщо кожна вершина з $V \setminus V'$ суміжна з якоюсь вершиною з V'. Інакше

кажучи, кожна вершина графа віддалена від домінантної множини не більше, ніж на одиницю. Домінантна множина має назву *мінімальної*, якщо жодна з її власних підмножин не домінантна. Домінантну множину з найменшою потужністю називають *найменшою*.

У відшуканні в графі найменшої домінантної множини полягає зміст багатьох практичних задач. Типова задача така. Є множина населених пунктів, зв'язаних дорогами. У деяких із них потрібно розмістити підприємства обслуговування так, щоб віддаль від кожного з населених пунктів до будь-якого підприємства не перевищувала заданої величини. Розміщення слід виконати так, щоб обійтися мінімальною кількістю підприємств. Населеним пунктам поставимо у відповідність вершини графа, у якому дві вершини з'єднано ребром тоді й лише тоді, коли віддаль між відповідними пунктами не перевищує заданої величини. Тоді задача зводиться до побудови в цьому графі найменшої домінантної множини.

Уведемо ще одне поняття, пов'язане з поняттям незалежності. Говорять, що вершина й ребро графа *покривають одне одного*, якщо вони інцидентні. Отже, ребро $e = \{u, v\}$ покриває вершини u та v, а кожна з цих вершин покриває ребро e.

Підмножину вершин $V' \subset V$ називають *покриттям* (*вершинним покриттям, опорою*) графа $G = (V, E)$, якщо кожне ребро з E інцидентне принаймні одній вершині з V'. Покриття графа G називають *мінімальним*, якщо воно не містить покриття з меншою кількістю вершин, і найменшим, якщо кількість вершин у ньому найменша серед усіх покриттів графа G. Кількість вершин у найменшому покритті графа G називають *числом покриття* (або *числом вершинного покриття*) графа G та позначають як $\beta(G)$.

Приклад 3.43. У графі, зображеному на рис. 3.51, кожна з множин $X_1 = \{v_1, v_3, v_4, v_6\}$, $X_2 = \{v_1, v_3, v_4, v_5, v_6\}$, $X_3 = \{v_2, v_4, v_5, v_6, v_7\}$, $X_4 = \{v_1, v_2, v_3, v_5, v_7, v_8\}$ являє собою покриття, причому X_1 — найменше покриття, а X_3, X_4 — мінімальні.

Наступна теорема свідчить про тісний зв'язок між покриттями та незалежними множинами графа.

ТЕОРЕМА 3.20. Множина X вершин графа $G = (V, E)$ являє собою (найменше, мінімальне) покриття тоді й лише тоді, коли $Y = V \setminus X$ — (найбільша, максимальна) незалежна множина. Отже, $\alpha(G) + \beta(G) = |V|$.

Доведення. За означенням множина Y незалежна тоді й лише тоді, коли в графі немає ребра, обидва кінці якого містяться в Y, тобто хоча б один із кінців кожного ребра належить X. Останнє означає, що X — вершинне покриття. Оскільки $|Y| + |X| = |V|$, то, очевидно, найбільшим Y відповідають найменші X і навпаки.

Протилежне до поняття незалежної множини поняття кліки. Підмножину $V' \subset V$ вершин графа G називають *клікою*, якщо будь-які дві вершини з V' суміжні, тобто породжений підграф $G(V')$ повний.

Кліку називають *максимальною*, якщо вона не є підмножиною жодної іншої кліки з більшою кількістю вершин. Найпотужнішу максимальну кліку називають *найбільшою*. Кількість вершин у найбільшій кліці графа називають *щільністю*, або *кліковим числом*, і позначають $\varphi(G)$. *Доповнювальним* до графа G називають

граф $\overline{G}$ з тією самою множиною вершин, будь-які дві вершини якого з'єднано ребром тоді й лише тоді, коли їх не з'єднано ребром у графі G.

ТЕОРЕМА 3.21. Підмножина вершин графа G являє собою кліку тоді й лише тоді, коли вона незалежна в доповнювальному графі $\overline{G}$. Отже, $\varphi(G) = \alpha(\overline{G})$.

Доведення випливає з означень.

Із теореми 3.21 випливає, що побудову максимальної кліки можна звести до побудови максимальної незалежної множини вершин. Метод побудови максимальних незалежних множин розглянуто в підрозділі 4.5 (приклад 4.19).

3.13. Паросполучення в графах. Теорема Холла

Теорема Холла має багато різних застосувань, три з яких ми розглянемо перед тим, як сформулювати цю теорему.

1. **Задача про весілля.** Розглянемо множину юнаків, кожний з яких знайомий із кількома дівчатами (табл. 3.7). Потрібно визначити умови, за яких кожен з юнаків міг би одружитися зі знайомою йому дівчиною.

Таблиця 3.7

Юнак	Дівчата, з якими знайомий юнак
b_1	g_1, g_4, g_5
b_2	g_1
b_3	g_2, g_3, g_4
b_4	g_3, g_4

2. **Трансверсаль.** Нехай $S = \{S_1, ..., S_m\}$ – сім'я підмножин скінченної множини M; S_i $(i = 1, ..., m)$ необов'язково різні та можуть перетинатись. *Системою різних представників* S (або *трансверсаллю* S) називають підмножину $C = \{c_1, ..., c_m\}$ з m таких елементів множини M, що $c_i \in S_i$. Потрібно визначити умови, за яких існує трансверсаль.

3. **Досконале паросполучення.** *Паросполученням* (або *незалежною множиною ребер*) у простому графі $G = (V, E)$ називають множину ребер, у якій ніякі два ребра не суміжні. Паросполучення графа G називають *максимальним*, якщо воно не міститься в жодному паросполученні з більшою кількістю ребер, і *найбільшим*, якщо кількість ребер у ньому найбільша серед усіх паросполучень графа G.

   Нехай $G = (V, E)$ – дводольний граф ($V = V_1 \cup V_2$, $V_1 \cap V_2 = \varnothing$, кінці ребер належать різним множинам V_1, V_2). *Досконалим паросполученням* із V_1 у V_2 називають паросполучення, яке *покриває* вершини V_1 (тобто всі вершини з V_1 інцидентні ребрам, що утворюють паросполучення). За яких умов існує досконале паросполучення з V_1 у V_2?

Можна довести, що задачі 1–3 – це по суті, одна й та сама задача. Справді, задача 1 зводиться до задачі 3 так. Нехай V_1 – множина юнаків, V_2 – множина

дівчат, ребра — знайомства юнаків із дівчатами (рис. 3.52). У такому разі досконале паросполучення у дводольному графі — це шукана множина весіль (один із можливих розв'язків на рисунку зображено потовщеними лініями).

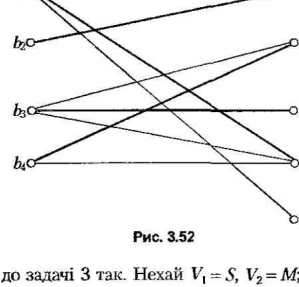


Рис. 3.52

Задача 2 зводиться до задачі 3 так. Нехай $V_1 = S$, $V_2 = M$; ребро $\{S_k, c_i\}$ існує, якщо $c_i \in S_k$. Тоді досконале паросполучення — це шукана трансверсаль.

Отже, задачі 1–3 мають спільну відповідь [35]:

$$\text{тоді й лише тоді, коли будь-які } k \begin{Bmatrix} \text{юнаків} \\ \text{підмножин} \\ \text{вершин із } V_1 \end{Bmatrix}$$

$$\text{у сукупності } \begin{Bmatrix} \text{знайомі з} \\ \text{містять} \\ \text{суміжні з} \end{Bmatrix} \text{ не менше ніж } k \begin{Bmatrix} \text{дівчатами} \\ \text{елементів} \\ \text{вершинами з } V_2 \end{Bmatrix}$$

$$\text{для всіх } k = 1, 2, \ldots, m.$$

Цю загальну схему обґрунтовує така теорема.

ТЕОРЕМА 3.22 (Ф. Холла). Нехай $G = (V, E)$ — дводольний граф, $V = V_1 \cup V_2$. Досконале паросполучення з V_1 у V_2 існує тоді й лише тоді, коли для кожної множини $A \subset V_1$ виконується умова $|A| \leqslant |\Gamma(A)|$ (означення множини $\Gamma(A)$ наведено в підрозділі 3.3).

Доведення. Необхідність. Нехай існує досконале паросполучення з V_1 у V_2. Тоді множина $\Gamma(A)$ містить $|A|$ вершин із V_2, які відповідають вершинам з A у цьому паросполученні, і, можливо, ще якісь вершини з V_2. Отже, $|A| \leqslant |\Gamma(A)|$.

Достатність. Застосуємо індукцію за кількістю вершин у множині V_1. Нехай $|V_1| = m > 0$. У разі $m = 1$ єдина вершина з V_1 інцидентна принаймні одному ребру. Це ребро і являє собою потрібне паросполучення. Нехай $m > 1$ й теорема справджується для графів, у яких $|V_1| < m$. Окремо розглянемо два можливі випадки.

Випадок 1. Для кожної підмножини $A \subset V_1$, $A \neq V_1$ виконується строга нерівність $|A| < |\Gamma_G(A)|$. Тут і далі в доведенні індекс біля Г показує, якого графа стосується позначення. Виберемо в графі G довільне ребро $\{x, y\}$. Розглянемо

граф $\tilde{G}$, одержаний із графа G вилученням вершин x та y і ребер, які інцидентні цим вершинам:

$$\tilde{V}_1 = V_1 \setminus \{x\}, \quad \tilde{V}_2 = V_2 \setminus \{y\},$$
$$\tilde{E} = E \setminus \{\{x, u\}, \{v, y\} \mid u \in V_2, v \in V_1\}.$$

Отриманий граф $\tilde{G} = (\tilde{V}_1 \cup \tilde{V}_2, \tilde{E})$ дводольний, причому $|\tilde{V}_1| = m - 1$. Нехай $\tilde{A}$ – довільна підмножина множини $\tilde{V}_1$. Оскільки $|\tilde{A}| < |\Gamma_{\tilde{G}}(\tilde{A})|$, а з множини V_2 вилучено лише одну вершину, то $|\tilde{A}| \leqslant |\Gamma_{\tilde{G}}(\tilde{A})|$. За індуктивним припущенням у графі $\tilde{G}$ існує паросполучення M, яке покриває $\tilde{V}_1$. Додамо до паросполучення M ребро $\{x, y\}$ і одержимо потрібне паросполучення в графі G.

Випадок 2. У множині V_1 існує така підмножина $A_0 \subset V_1$, $A_0 \neq V_1$, що

$$|A_0| = |\Gamma_G(A_0)|. \tag{3.2}$$

Позначимо $B = A_0 \cup \Gamma_G(A_0)$, і нехай H_1 та H_2 – підграфи графа G, породжені відповідно множинами вершин B та $V \setminus B$.

Розглянемо спочатку підграф H_1. Для будь-якої множини $A \subset A_0$ маємо $\Gamma_G(A) = \Gamma_{H_1}(A)$; отже, $|A| \leqslant |\Gamma_{H_1}(A)|$. Тому за індуктивним припущенням в H_1 існує паросполучення, яке покриває множину A_0.

Тепер розглянемо підграф H_2. Для будь-якої підмножини $A \subset V_1 \setminus A_0$

$$|A_0| + |A| = |A_0 \cup A| \leqslant |\Gamma_G(A_0 \cup A)| = |\Gamma_G(A_0)| + |\Gamma_{H_2}(A)|.$$

З урахуванням рівності (3.2) одержимо, що $|A| \leqslant |\Gamma_{H_2}(A)|$, і за припущенням індукції в графі H_2 існує паросполучення, яке покриває множину $V_1 \setminus A_0$. Об'єднаємо його з паросполученням, яке покриває множину A_0, й отримаємо паросполучення, що покриває всю множину V_1.

3.14. Найбільше паросполучення у дводольних графах

Задача побудови найбільшого паросполучення в графі (див. підрозділ 3.13) широко розповсюджена, і є ефективні алгоритми її розв'язування. Ці алгоритми ґрунтуються на методі почергових шляхів, ідея якого належить Дж. Петерсену [15].

Нехай M – паросполучення в графі G. Простий шлях, ребра якого почергово входять і не входять в M, називають *почерговим шляхом* відносно паросполучення M. Шлях довжиною 1 за означенням також почерговий. Ребра шляху називають *темними* (*світлими*), якщо вони належать (не належать) паросполученню M. Вершини графа G, інцидентні ребрам із паросполучення M, називають *насиченими*, усі інші вершини – *ненасиченими*.

Приклад 3.44. Розглянемо, наприклад, граф, зображений на рис. 3.53. Множина ребер $M = \{e_1, e_7, e_{10}\}$ являє собою паросполучення; $L = 7, 8, 4, 1, 2, 5$ — почерговий щодо M шлях: $e_1 = \{1, 2\}$, $e_{10} = \{4, 8\}$ — темні ребра шляху L; $e_3 = \{1, 4\}$, $e_5 = \{2, 5\}$, $e_{13} = \{7, 8\}$ — світлі; {1, 2, 3, 4, 6, 8} і {5, 7, 9, 10} — множини відповідно насичених і ненасичених вершин.

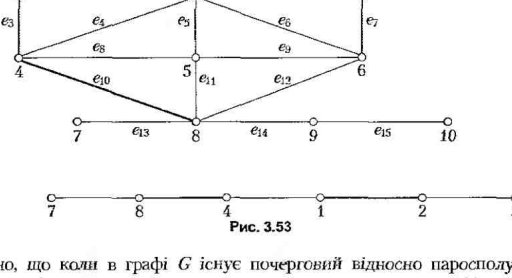


Рис. 3.53

Очевидно, що коли в графі G існує почерговий відносно паросполучення M шлях, який з'єднує дві різні ненасичені вершини, то в цьому графі можна побудувати паросполучення з більшою кількістю ребер, ніж у паросполученні M. Справді, у такому разі кількість темних ребер на одиницю менша, ніж кількість світлих. Вилучивши з M усі темні ребра та приєднавши всі світлі, отримаємо нове паросполучення, у якому на одне ребро більше. Тому почерговий щодо паросполучення M шлях, який з'єднує дві різні ненасичені вершини, називають *збільшувальним* стосовно M шляхом у графі G. Отже, відсутність збільшувальних відносно M шляхів — необхідна умова того, що паросполучення найбільше. Цяж умова виявляється й достатньою.

ТЕОРЕМА 3.23. Паросполучення M у графі найбільше тоді й лише тоді, коли в цьому графі немає збільшувальних щодо M шляхів.

Доведення. Необхідність. як уже було зазначено. очевидна. Достатність доведемо від протилежного. Нехай M_1 — також паросполучення в графі G, $|M_1| > |M|$. Розглянемо граф H, утворений ребрами $(M \cup M_1) \setminus (M \cap M_1)$. Оскільки довільна його вершина v інцидентна не більше ніж одному ребру кожного паросполучення M і M_1, то її степінь не більший ніж 2. Якщо $\deg(v) = 2$, то одне з інцидентних вершині v ребер належить паросполученню M, друге — M_1. Тому будь-яка зв'язна компонента графа H являє собою або цикл, що містить однакову кількість ребер із паросполучень M і M_1, або почерговий відносно M шлях. Але $|M_1| > |M|$, тому серед цих компонент обов'язково є почерговий щодо паросполучення M шлях, крайні ребра якого (перше й останнє) належать M_1. Тоді крайні вершини цього шляху не насичені паросполученням M, що суперечить умові теореми.

Приклад 3.45. Знову розглянемо граф, зображений на рис. 3.53. Почерговий шлях з'єднує ненасичені вершини 5 і 7. Отже, можна побудувати паросполучення M' із більшою кількістю ребер: $M' = (M \setminus \{e_1, e_{10}\}) \cup \{e_3, e_5, e_{13}\} = \{e_3, e_5, e_7, e_{13}\}$. Паросполучення M' також не

найбільше: 9, 10 – збільшувальний щодо M' шлях. Паросполучення $M'' = M' \cup \{e_{15}\} = \{e_3, e_5, e_7, e_{13}, e_{15}\}$ найбільше в графі.

Отже, теорема 3.23 підказує таку стратегію пошуку найбільшого паросполучення: почати з довільного паросполучення M_1 і будувати послідовність $M_1, M_2, M_3, \ldots$, у якій паросполучення M_{k+1} отримано з M_k за допомогою щойно розглянутої зміни вздовж якогось збільшувального шляху. Оскільки $|M_{k+1}| = |M_k| + 1$, то для одержання найбільшого паросполучення потрібно не більше ніж $|V|/2$ ітерацій (тобто переходів від M_k до M_{k+1}). Початкове паросполучення M_1 завжди є в нашому розпорядженні: можна взяти довільне ребро графа чи, краще, якесь максимальне паросполучення. Тому розробка ефективного алгоритму, що ґрунтується на зазначеній стратегії, зводиться до побудови процедури, яка швидко знаходить збільшувальний шлях у графі чи виявляє, що його немає. Обмежимося дводольними графами [2], хоча така процедура відома й для довільних графів [25].

Отже, нехай $G = (V, E)$ – дводольний граф і множину його вершин V розбито на дві підмножини V_1 і V_2, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$ (рис. 3.54).

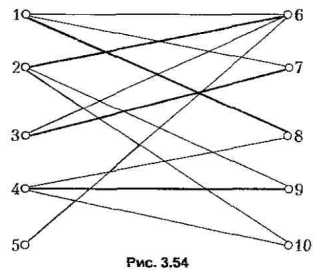


Рис. 3.54

Побудуємо граф збільшувального шляху за рівнями $i = 0, 1, 2, \ldots$, використовуючи процес, подібний до пошуку вшир. Граф збільшувального шляху рівня 0 містить усі ненасичені вершини з множини V_1. На рівні з непарним номером i додають нові вершини, суміжні з вершинами рівня $i-1$ і з'єднані світлим ребром (яке не належить паросполученню). Це ребро також додають до будованого графа. На рівні з парним номером i також додають нові вершини, суміжні з вершинами рівня $i-1$, але з'єднані темним ребром (яке належить паросполученню). Це ребро також додають до графа збільшувального шляху.

Процес побудови продовжують доти, доки до графа почергового шляху можна приєднувати нові вершини. Зазначимо, що ненасичену вершину можна приєднати до цього графа тільки на непарному рівні. У побудованому графі шлях від будь-

якої ненасиченої вершини (яка може бути тільки на непарному рівні) до будь-якої вершини рівня 0 — збільшувальний шлях відносно паросполучення M.

На рис. 3.55 зображено граф збільшувального шляху для дводольного графа, який зображено на рис. 3.54, щодо паросполучення, показаного потовщеними лініями (темні ребра).

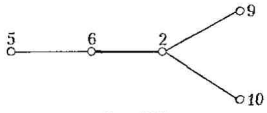


Рис. 3.55

На рівні 0 маємо одну ненасичену вершину 5. На рівні 1 додано світле ребро {5, 6} (яке не належить паросполученню з рис. 3.54). На рівні 2 додано темне ребро {6, 2} (яке належить паросполученню). На рівні 3 додано світлі ребра {2, 9} і {2, 10}, які не належать паросполученню. Оскільки вершина 10 у графі, зображеному на рис. 3.54, ненасичена, то можна зупинити процес побудови графа збільшувального шляху. Шлях 10, 2, 6, 5 збільшувальний відносно паросполучення, показаного на рис. 3.54. Із паросполучення рис. 3.54 вилучимо всі темні ребра, що належать цьому шляху, і додамо всі світлі. Отримаємо нове паросполучення, яке містить на одне ребро більше (рис. 3.56).

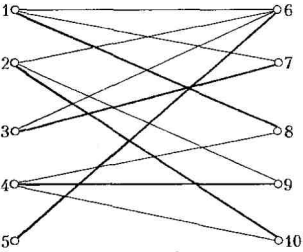


Рис. 3.56

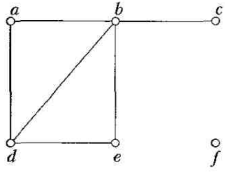
Для паросполучення, зображеного на рис. 3.56, очевидно, не існує збільшувальних шляхів. Отже, це паросполучення найбільше.

Нехай граф $G = (V, E)$ задано списками суміжності. Тоді на побудову графа збільшувального шляху потрібно часу порядку $O(|E|)$. Для знаходження найбільшого паросполучення, як уже було зазначено, потрібно побудувати не більше ніж $|V|/2$ збільшувальних шляхів. Тому найбільше паросполучення дводольного графа можна знайти за час порядку $O(|V| \cdot |E|)$.

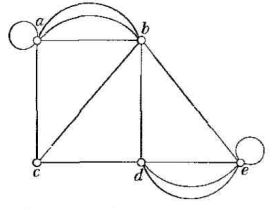
Контрольні запитання та завдання

1. Знайти кількість вершин, ребер і степені кожної вершини несорієнтованих графів:

а)

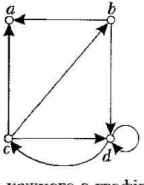


б)

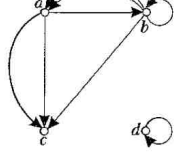


2. Знайти суму степенів вершин кожного з графів задачі 1 та переконатись, що вона вдвічі більша за кількість ребер графа.
3. Визначити кількість вершин та дуг і знайти напівстепені входу й виходу для кожної вершини орієнтованих мультиграфів:

а)



б)

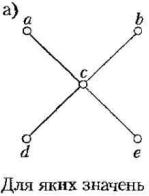


4. Для кожного з графів задачі 3 знайти суму напівстепенів входу та суму напівстепенів виходу вершин. Переконатись, що кожна з них дорівнює кількості дуг графа.
5. Побудувати графи:

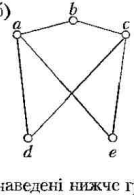
а) K_7; б) $K_{1,8}$; в) $K_{1,4}$; г) C_7; д) W_7; е) Q_4.

6. Які з наведених нижче графів дводольні?

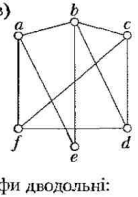
а)



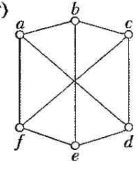
б)



в)



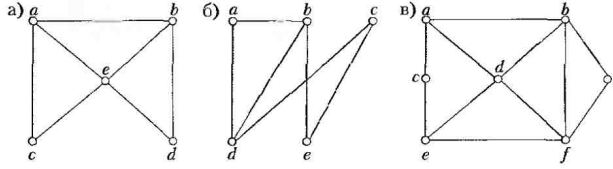
г)



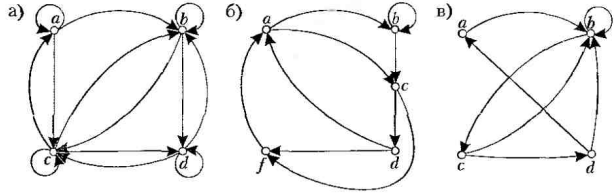
7. Для яких значень n наведені нижче графи дводольні:

а) K_n; б) C_n; в) W_n; г) Q_n?

8. Скільки вершин і ребер мають наведені нижче графи:
   а) K_n; б) C_n; в) W_n; г) Q_n; д) $K_{n,n}$?
9. Скільки ребер має граф, у якого вершини мають такі степені: 4, 3, 3, 2, 2? Зобразити його.
10. Чи існує простий граф із вершинами таких степенів? Якщо так, то зобразити його:
    а) 3, 3, 3, 3, 2; б) 3, 4, 3, 4, 3; в) 1, 2, 3, 4, 5;
    г) 1, 2, 3, 4, 4; д) 0, 1, 2, 2, 3; е) 1, 1, 1, 1, 1.
11. Нехай G — граф з n вершинами та m ребрами. Нехай d_{max} — максимальний степінь вершини цього графа, а d_{min} — мінімальний. Довести, що $d_{min} \leqslant 2m/n \leqslant d_{max}$.
12. Простий граф називають *регулярним*, якщо всі його вершини мають однаковий степінь. Граф називають ρ-регулярним, якщо кожна його вершина має степінь ρ. Для якого n наведені нижче графи регулярні:
    а) K_n; б) C_n; в) W_n; г) Q_r
13. Для яких m і n граф $K_{m,n}$ регулярний?
14. Скільки вершин має регулярний граф степеня 4 з 10 ребрами?
15. Задати прості графи за допомогою матриць інцидентності.



16. Задати орієнтовані графи за допомогою матриць інцидентності.



17. Задати графи із задачі 15 за допомогою матриць суміжності.
18. Задати графи із задачі 16 за допомогою матриць суміжності.
19. Задати графи задачі 15 списками пар (списками ребер), записаних у лексикографічному порядку.

20. Задати графи задачі 16 списками пар (списками дуг), записаних у лексикографічному порядку.

21. Задати графи задачі 15 списками суміжності.

22. Задати графи задачі 16 списками суміжності.

23. Зобразити неорієнтовані графи за матрицями суміжності:

а) $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$; б) $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$; в) $\begin{pmatrix} 1 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$.

24. Чи є будь-яка квадратна симетрична (0,1)-матриця, що містить 0 на головній діагоналі, матрицею суміжності якогось простого графа?

25. Зобразити орієнтовані графи за матрицями суміжності:

а) $\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$; б) $\begin{pmatrix} 0 & 2 & 3 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix}$.

26. Зобразити неорієнтований граф, заданий множиною вершин і багатозначним відображенням Γ.

v	v_1	v_2	v_3	v_4	v_5
$\Gamma(v)$	v_2, v_3, v_5	v_1	v_1, v_4, v_5	v_3, v_5	v_1, v_3, v_4

27. Зобразити орієнтований граф, заданий множиною вершин і багатозначним відображенням Γ.

v	v_1	v_2	v_3	v_4	v_5
$\Gamma(v)$	v_2, v_3, v_4, v_5	v_2, v_4	v_1, v_3, v_5	–	v_2, v_3, v_4

28. Знайти матриці суміжності для графів:
а) K_n; б) C_n; в) W_n; г) $K_{m,n}$; д) Q_n.

29. Нехай G – простий граф. Нагадаємо, що *доповнювальним* називають такий граф $\overline{G}$, у якого та сама множина вершин і дві вершини в $\overline{G}$ з'єднано ребром тоді й лише тоді, коли їх не з'єднано ребром у графі G. Знайти:
а) $\overline{K}_n$; б) $\overline{K}_{m,n}$; в) $\overline{C}_n$.

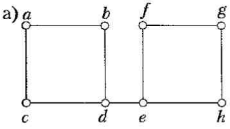
30. Нехай G – простий граф з n вершинами та m ребрами. Довести, що об'єднання G та $\overline{G}$ утворює граф K_n.

31. Простий граф G має 15 ребер, а граф $\overline{G}$ – 13. Знайти кількість вершин графа G.

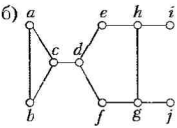
32. Простий граф G має n вершин і m ребер. Знайти кількість ребер графа $\overline{G}$.

33. Обійти наведені нижче графи пошуком углиб. Уважати, що вершини впорядковано за алфавітом, а початкова — вершина a.

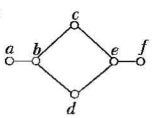
а)



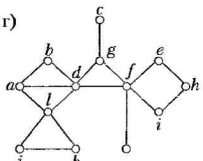
б)



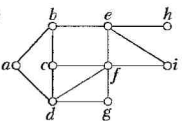
в)



г)



д)

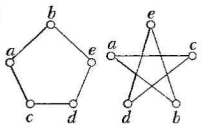


34. Обійти графи із задачі 33 пошуком ушир. Уважати, що вершини впорядковано за алфавітом, а початкова — вершина a.

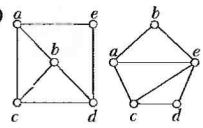
35. Розв'язати задачі 33 і 34 за умови, що початкова вершина обходу графа — d.

36. Визначити, які пари графів, наведених нижче, ізоморфні.

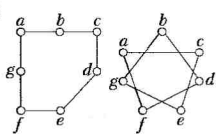
а)



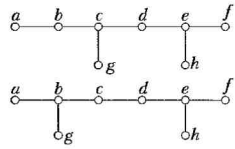
б)



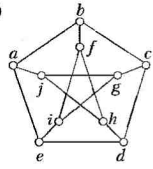
в)

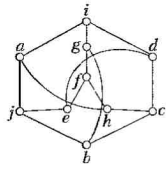


г)



д)

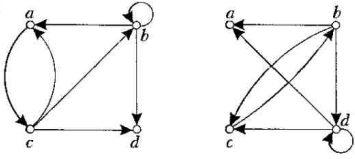


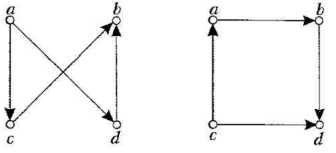


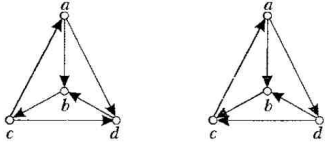
37. Знайти кількість неізоморфних простих графів з n вершинами, якщо n дорівнює:
а) 2; б) 3; в) 4.

38. Знайти кількість неізоморфних простих графів з п'ятьма вершинами та трьома ребрами.

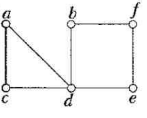
39. Визначити, які пари орієнтованих графів ізоморфні.

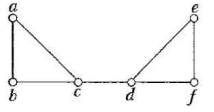
а)


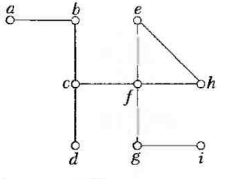
б)


в)


40. Знайти точки з'єднання графів.

а)


б)


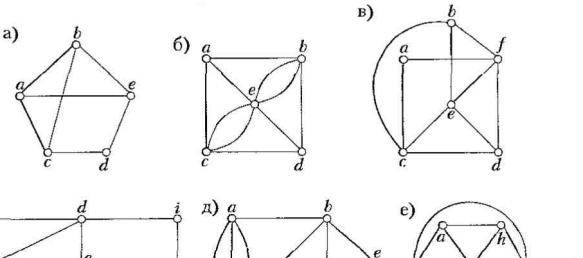
в)


41. Знайти всі мости в графах задачі 40.

42. Знайти кількість шляхів довжиною n між двома різними вершинами графа K_6, якщо n дорівнює 2; 3; 4; 5.

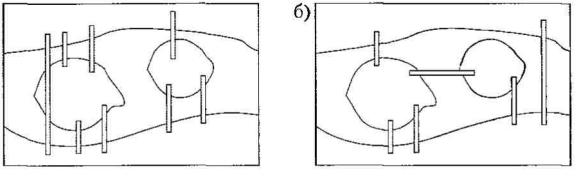
43. Знайти кількість шляхів довжиною n між двома несуміжними вершинами графа $K_{3,3}$, якщо n дорівнює 2, 3, 4, 5. Визначити те саме для суміжних вершин.

44. Визначити, які з наведених нижче графів мають ейлерів цикл. Зобразити його.



45. Який із графів задачі 44 має ейлерів шлях, але не має ейлерового циклу?

46. Чи можна утворити ейлерів цикл у разі розміщення мостів?

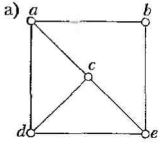


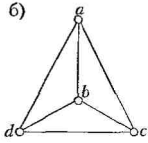
47. Для яких значень n наведені нижче графи мають ейлерів цикл:
а) C_n; б) K_n; в) W_n; г) Q_n?

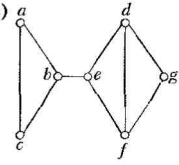
48. Для яких значень n наведені нижче графи мають ейлерів шлях, але не мають ейлерового циклу:
а) C_n; б) K_n; в) W_n; г) Q_n?

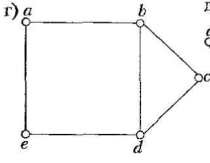
49. Визначити довжину найкоротшого циклу в наведених нижче графах, який проходять через кожне ребро принаймні один раз:
а) K_6; б) K_n; в) $K_{2,5}$; г) $K_{m,n}$.

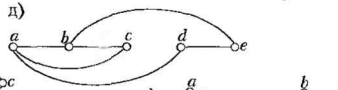
50. Які з наведених нижче графів мають гамільтонів цикл?

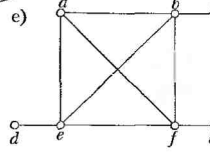
а) 

б) 

в) 

г) 

д) 

е) 

51. Які з графів задачі 50, що не мають гамільтонового циклу, мають гамільтонів шлях?

52. Для яких значень n наведені нижче графи мають гамільтонів цикл:

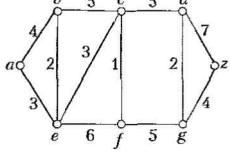
а) C_n; б) K_n; в) W_n; г) Q_n?

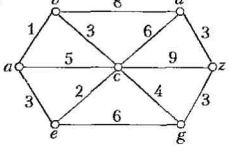
53. Для яких значень m і n граф $K_{m,n}$ має гамільтонів цикл?

54. Навести приклад графа, який має ейлерів цикл, але не має гамільтонового циклу, а також графа, який має гамільтонів цикл, але не має ейлерового циклу. Як можна охарактеризувати графи, які мають водночас і ейлерів, і гамільтонів цикли?

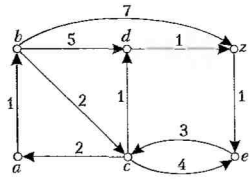
55. Навести контрприклад, який показує, що у формулюванні теореми Дірака умову $\deg(v) \geqslant \frac{n}{2}$ не можна замінити слабшою умовою $\deg(v) \geqslant \frac{n}{2} - 1$.

56. За допомогою алгоритму Дейкстри знайти найкоротший шлях від a до z у неорієнтованих графах, зображених на рисунках.

а) 

б) 

57. За допомогою алгоритму Дейкстри знайти найкоротший шлях від a до z в орієнтованому графі, зображеному на рисунку.



58. Довести (навести приклад), що алгоритм Дейкстри може й не визначити найкоротший шлях, якщо в графі є дуги з від'ємними довжинами.

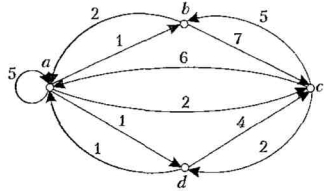
59. Нехай алгоритм Дейкстри застосовано до графа з вершинами $v_1, v_2, ..., v_7$, унаслідок чого одержано вектор довжин (постійних міток) $l = (0, 6, 5, 8, 12, 13, 14)$ і вектор вершин $\theta = (-, 1, 3, 4, 4, 6)$, у якому вершину v_i позначено як i. Виконати такі завдання:

а) визначити найкоротший шлях від v_1 до v_7;

б) визначити найкоротший шлях від v_1 до v_5;

в) зобразити цей граф, якщо це можливо.

60. За допомогою алгоритму Флойда визначити довжини найкоротших шляхів між усіма парами вершин орієнтованого графа. У процесі розв'язування будувати матриці W та Θ. За матрицею Θ визначити найкоротші шляхи від вершини a до вершини d та від вершини b до вершини d.



61. Зв'язний планарний граф має шість вершин степеня 4. Скільки граней має цей граф?

62. Зв'язний планарний граф регулярний і містить 30 ребер. Кількість його граней дорівнює 20. Скільки вершин має цей граф?

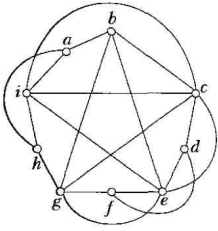
63. Нехай зв'язний простий планарний граф містить $n \geqslant 3$ вершин і m ребер. Довести, що $m \leqslant 3n - 6$.

64. Нехай зв'язний дводольний планарний граф містить $n \geqslant 3$ вершин і m ребер. Довести, що $m \leqslant 2n - 4$.

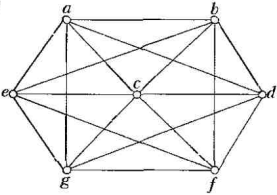
65. Довести, що в будь-якому простому планарному графі є вершина, степінь якої не більший ніж 5.

66. За допомогою теореми Куратовського довести, що графи, наведені нижче, непланарні.

а)

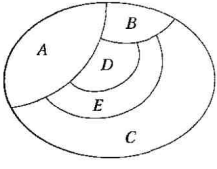


б)

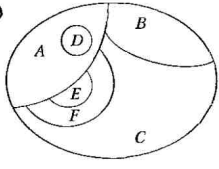


67. Побудувати графи, які відповідають наведеним нижче картам, і визначити найменшу кількість кольорів для їх розфарбовування.

а)



б)



68. Знайти хроматичні числа графів:

а) C_n; б) W_n; в) Q_n.

69. Довести, що простий граф, який містить цикл із непарною кількістю вершин, не можна розфарбувати у два кольори.

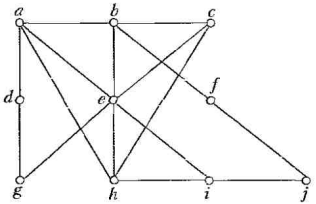
70. Для розфарбовування простого графа G можна застосувати такий алгоритм. Перенумерувати вершини $v_1, v_2, ..., v_n$ за спаданням степенів:

$$\deg(v_1) \geqslant \deg(v_2) \geqslant ... \geqslant \deg(v_n).$$

Зафарбувати кольором 1 вершину v_1 і послідовно кожну вершину зі списку, не суміжну з вершиною кольору 1. Зафарбувати кольором 2 першу серед не зафарбованих іще вершин зі списку та послідовно зафарбувати кольором 2 вершини зі списку, які ще не зафарбовані й не суміжні з вершинами кольору 2. Далі вибрати колір 3 та продовжити процес.

За допомогою цього алгоритму розфарбувати граф, зображений на рисунку. Довести (навести приклад), що цей алгоритм може й не дати оптимального

розфарбування графа G, тобто розфарбування в найменшу можливу кількість кольорів $\chi(G)$.



71. Скільки каналів щонайменше потрібно для шести телевізійних станцій, відстані між якими задано таблицею, якщо дві станції не можуть працювати на одному каналі, коли відстані між ними менша ніж 250 км?

Телевізійна станція	1	2	3	4	5	6
1	–	145	280	300	90	160
2	145	–	200	280	150	270
3	280	200	–	220	350	400
4	300	280	220	–	310	360
5	90	150	350	310	–	120
6	160	270	400	360	120	–

72. На кафедрі працюють шість студентських наукових семінарів один раз на місяць в один і той самий час. Скільки різних днів щонайменше потрібно для проведення цих семінарів, якщо склад їх керівників такий:

1) $C1$ = {Пасічник, Щербина, Нікольський};

2) $C2$ = {Щербина, Годич, Катренко};

3) $C3$ = {Пасічник, Катренко, Нікольський};

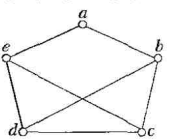
4) $C4$ = {Годич, Нікольський, Катренко};

5) $C5$ = {Пасічник. Щербина};

6) $C6$ = {Щербина, Катренко, Нікольський}.

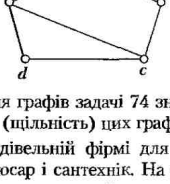
Умову задачі зобразити у вигляді графа.

73. Для графа, зображеного на рисунку, побудувати хроматичний поліном.



74. Для наведених нижче графів знайти всі максимальні незалежні множини вершин. Зазначити найбільшу незалежну множину та число незалежності кожного з цих графів.

а)



б)

75. Для графів задачі 74 знайти всі максимальні кліки. Визначити клікове число (щільність) цих графів.

76. Будівельній фірмі для виконання певної роботи потрібні муляр, столяр, слюсар і сантехнік. На ці місця є п'ять претендентів: один може працювати муляром, другий — столяром, третій — муляром і сантехніком, ще двоє мають по дві спеціальності — сантехніка та слюсаря. Чи можуть виконати всі роботи четверо з цих робітників? Для обґрунтування відповіді докладно перевірити виконання умов теореми Холла.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано множину пар вершин, що відповідають ребрам неорієнтованого графа. Знайти степінь кожної вершини.
2. Задано множину впорядкованих пар вершин, що відповідають дугам орієнтованого графа. Знайти напівстепені входу та виходу кожної вершини.
3. Задано множину пар вершин, що відповідають ребрам простого графа. Визначити, чи дводольний цей граф.
4. Задано множину пар вершин, що відповідають ребрам графа. Побудувати матрицю суміжності графа з урахуванням того, що граф може мати петлі та кратні ребра, а також бути орієнтованим.
5. Задано матрицю суміжності неорієнтованого графа. Побудувати множину пар вершин, що відповідають їй, і знайти кратність кожного ребра.
6. Задано множину пар вершин, що відповідають ребрам неорієнтованого графа, а також кратність кожного ребра. Побудувати матрицю інцидентності цього графа.
7. Задано матрицю інцидентності неорієнтованого графа. Побудувати множину пар вершин, що відповідають його ребрам, і визначити кратність кожного ребра.
8. Задано множини пар вершин, що відповідають ребрам двох простих графів не більше ніж із шістьма вершинами. Визначити, чи ізоморфні ці графи.
9. Задано матрицю суміжності графа та натуральне число r. Знайти кількість шляхів довжиною r між двома вершинами (граф може бути як орієнтованим, так і неорієнтованим).

10. Задано множину пар вершин, що відповідають ребрам простого графа. Визначити, чи зв'язний граф. Якщо виявиться, що граф незв'язний, знайти кількість його компонент.
11. Задано матрицю суміжності мультиграфа. Зазначити, чи має він ейлерів цикл, а якщо ні, то ейлерів шлях. Побудувати ейлерів цикл або шлях, якщо вони існують.
12. Задано множину пар вершин, що відповідають ребрам мультиграфа, і кратність кожного ребра. Визначити, чи має цей мультиграф ейлерів цикл, а якщо ні, то ейлерів шлях. Побудувати ейлерів цикл або шлях, якщо вони існують.
13. Задано матрицю суміжності орієнтованого мультиграфа. Визначити, чи має він ейлерів цикл, а якщо ні, то ейлерів шлях. Побудувати ейлерів цикл або шлях, якщо вони існують.
14. Задано множину впорядкованих пар вершин, що відповідають дугам орієнтованого мультиграфа, і кратність кожної дуги. Визначити, чи має він ейлерів цикл, а якщо ні, то ейлерів шлях. Побудувати ейлерів цикл або шлях, якщо вони існують.
15. Задано матрицю суміжності простого графа. Побудувати цикл із найменшою довжиною, який проходить через кожне ребро графа принаймні один раз.
16. Задано зважений неорієнтований граф із двома вершинами непарного степеня. Знайти цикл із найменшою довжиною, який проходить через кожне ребро графа принаймні один раз.
17. Задано множину пар вершин, що відповідають ребрам зваженого зв'язного простого графа, і дві його вершини. За допомогою алгоритму Дейкстри знайти довжину найкоротшого шляху між зазначеними вершинами та побудувати цей шлях.
18. Задано віддалі між парами телевізійних станцій. Призначити розподіл каналів для них, тобто побудувати граф, у якому станції позначені вершинами. Дві вершини з'єднано ребром, якщо віддаль між ними не більша ніж 250 км. Призначенню каналів відповідає таке розфарбування вершин графа, у якому різні кольори відповідають різним каналам. Використати найменшу можливу кількість каналів.
19. Знайти всі точки з'єднання заданого простого графа.
20. Знайти всі мости заданого простого графа за допомогою алгоритму пошуку вглиб.
21. Задано дводольний граф. Знайти найбільше паросполучення.

Розділ 4

Дерева та їх застосування

- Основні означення та властивості
- Рекурсія. Обхід дерев.
  Префіксна та постфіксна форми запису виразів
- Бінарне дерево пошуку
- Дерево прийняття рішень
- Бектрекінг (пошук із поверненнями)
- Каркаси (з'єднувальні дерева)

Поняття дерева широко застосовують у багатьох розділах математики й інформатики. Наприклад, дерева використовують як інструмент обчислень, зручний спосіб збереження даних, їх сортування чи пошуку.

4.1. Основні означення та властивості

Деревом називають зв'язний граф без простих циклів. Граф, який не містить простих циклів і складається з k компонент, називають *лісом* із k дерев.

Приклад 4.1. На рис. 4.1 зображено приклади дерев. Граф, зображений на рис. 4.2 — не дерево, бо він незв'язний.

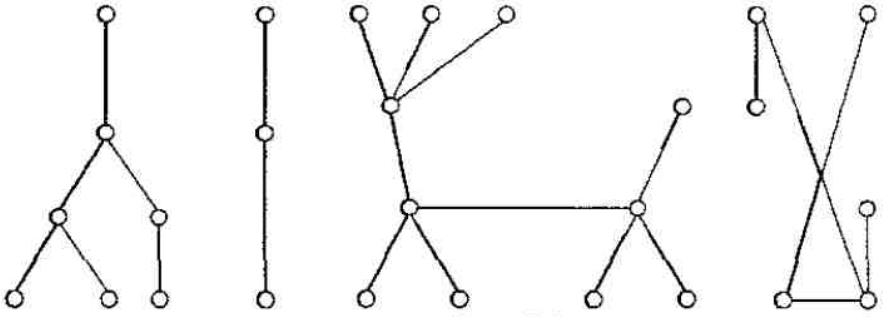


Рис. 4.1

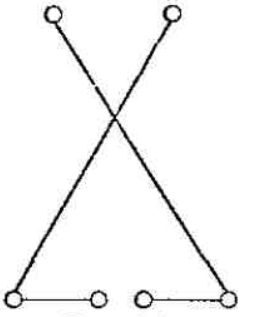


Рис. 4.2

Зауваження. З означення випливає, що дерева й ліси являють собою прості графи.

ТЕОРЕМА 4.1. Нехай граф T має n вершин. Тоді такі твердження еквівалентні:

(I) граф T – дерево;

(II) граф T не містить простих циклів і має $(n-1)$ ребро;

(III) граф T зв'язний і має $(n-1)$ ребро;

(IV) граф T зв'язний, але вилучення довільного ребра робить його незв'язним;

(V) довільні дві вершини графа T з'єднані точно одним простим шляхом;

(VI) граф T не містить простих циклів, але, додавши до нього довільне нове ребро, ми отримаємо точно один простий цикл.

Доведення (математичною індукцією). У разі $n=1$ твердження тривіальні; припустимо, що $n \geqslant 2$.

(I) → (II). За означенням T не містить простих циклів. Отже, вилучивши довільне ребро, ми одержимо два графи, кожний з яких являє собою дерево з меншою, ніж у T, кількістю вершин. За припущенням індукції кількість ребер у кожному з отриманих дерев на 1 менша за кількість вершин. Звідси випливає, що граф T має $(n-1)$ ребро.

(II) → (III). Припустимо, що граф T незв'язний. Тоді кожна його компонента являє собою зв'язний граф без простих циклів, тобто дерево. Звідси випливає, що кількість вершин у кожній компоненті на одиницю більша від кількості ребер. Отже, загальна кількість вершин графа T більша за кількість ребер принаймні на 2. Але це суперечить тому, що граф T має $(n-1)$ ребро.

(III) → (IV). Вилучимо довільне ребро, отримаємо граф з n вершинами та $(n-2)$ ребрами. Припущення про зв'язність такого графа суперечить теоремі про оцінку (знизу) кількості ребер звичайного графа (теорема 3.6).

(IV) → (V). Оскільки граф T зв'язний, то кожну пару його вершин з'єднано принаймні одним простим шляхом (теорема 3.4). Якщо якусь пару вершин з'єднано двома простими шляхами, вони замикаються в простий цикл. Але це суперечить тому, що вилучення довільного ребра робить граф T незв'язним.

(V) → (VI). Припустимо, що граф T містить простий цикл. Тоді довільні дві вершини цього циклу з'єднано принаймні двома простими шляхами, що суперечить твердженню (V). Додавши тепер до графа T ребро e, одержимо єдиний простий цикл, бо інцидентні ребру e вершини вже з'єднано в графі T точно одним простим шляхом.

(VI) → (I). Припустимо, що граф T незв'язний. Тоді додавання будь-якого ребра, що з'єднує вершину однієї компоненти з вершиною іншої, не зумовлює утворення простого циклу, що суперечить твердженню (VI).

Наслідок із твердження (II). Ліс із k дерев, який містить n вершин, має $(n-k)$ ребер.

У багатьох застосуваннях певну вершину дерева означають як *коріньо*. Тоді можна природно приписати напрямок кожному ребру. Оскільки існує єдиний простий шлях від кореня до кожної вершини графа, то можна орієнтувати кожне ребро в напрямку від кореня. Отже, дерево разом із виділеним коренем утворює орієнтований граф, які називають *кореневим деревом*.

Різні способи вибору кореня дають змогу утворити різні кореневі дерева.

Приклад 4.2. На рис. 4.3, *а* зображено дерево, а на рис. 4.3, *б*, *в* — кореневі дерева з коренями відповідно у вершинах *a* та *c*.

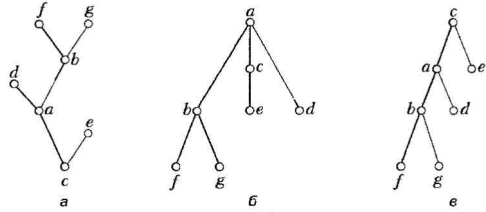


Рис. 4.3

Нехай T — кореневе дерево. Якщо v – його вершина, відмінна від кореня, то її *батьком* називають єдину вершину u таку, що є орієнтоване ребро (u, v). Якщо u — батько, то v — *син*. Аналогічно за генеалогічною термінологією можна означити інших предків і нащадків вершини v. Вершини дерева, які не мають синів, називають *листками*. Вершини, які мають синів, називають *внутрішніми*. Нехай a — вершина дерева. Тоді *піддеревом* із коренем a називають підграф, що містить a та всі вершини — нащадки вершини a, а також інцидентні їм ребра.

Кореневе дерево називають *m-арним*, якщо кожна його внутрішня вершина має не більше ніж m синів. Дерево називають *повним m-арним*, якщо кожна його внутрішня вершина має точно m синів. У разі $m=2$ дерево називають *бінарним*.

Кореневе дерево, у якому сини кожної внутрішньої вершини впорядковано, називають *упорядкованим*. Таке дерево зображають так, щоб сини кожної вершини були розміщені зліва направо.

Якщо внутрішня вершина впорядкованого бінарного дерева має двох синів, то першого називають *лівим*, а другого — *правим*. Піддерево з коренем у вершині, яка являє собою лівого сина вершини v, називають *лівим піддеревом у цій вершині*. Якщо корінь піддерева — правий син вершини v, то таке піддерево називають *правим піддеревом у цій вершині*.

Приклад 4.3. У дереві, зображеному на рис. 4.4, Л і П — відповідно ліве та праве піддерева у вершині *c*.

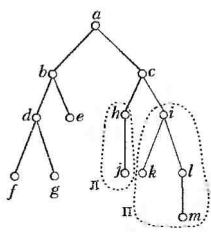


Рис. 4.4

ТЕОРЕМА 4.2. Повне m-арне дерево з i внутрішніми вершинами містить $n=mi+1$ вершин.

Доведення. Кожна вершина, окрім кореня, — син внутрішньої вершини. Оскільки кожна з i внутрішніх вершин має m синів, то всього є, якщо не враховувати корінь, mi вершин, а з урахуванням кореня їх $mi+1$.

Рівнем вершини v в кореневому дереві називають довжину простого шляху від кореня до цієї вершини (цей шлях, очевидно, єдиний). Рівень кореня вважають нульовим. *Висотою* кореневого дерева називають максимальний із рівнів його вершин. Інакше кажучи, висота кореневого дерева — це довжина найдовшого простого шляху від кореня до будь-якої вершини. Повне m-арне дерево, у якого всі листки на одному рівні, називають *завершеним*.

Кореневе m-арне дерево з висотою h називають *збалансованим* [52], якщо всі його листки на рівнях h або $h-1$.

Приклад 4.4. На рис. 4.5 зображено збалансоване бінарне дерево, яке має висоту 4: усі його листки на рівнях 3 та 4.

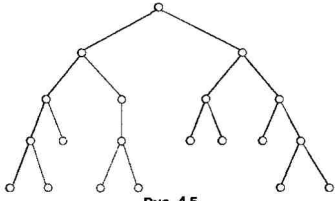


Рис. 4.5

Збалансованість можна означити й інакше [7, 35].

ТЕОРЕМА 4.3. Нехай m-арне дерево має висоту h. Тоді в ньому не більше ніж m^h листків.

Доведення. Застосуємо математичну індукцію за h. У разі $h=1$ твердження очевидне. Припустимо, що воно справджується для всіх m-арних дерев із меншою висотою, ніж h. Нехай T — m-арне дерево з висотою h. Тоді його листки — це листки піддерев, отриманих із T вилученням ребер, що з'єднують корінь дерева T з кожною вершиною рівня 1 (рис. 4.6).

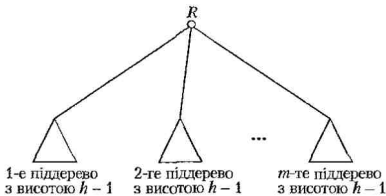


Рис. 4.5

Кожне з цих піддерев має не більшу висоту, ніж $h-1$. За індуктивною гіпотезою всі вони мають не більше ніж m^{h-1} листків. Позаяк таких піддерев не більше ніж m, то загальна кількість листків у дереві T не перевищує $mm^{h-1}=m^h$.

Наслідок. Якщо m-арне дерево з висотою h має l листків, то $h \geqslant \lceil \log_m l \rceil$. Якщо m-арне дерево повне та збалансоване, то $h=\lceil \log_m l \rceil$. Нагадаємо, що $\lceil x \rceil$ – це найменше ціле число, яке більше чи дорівнює x.

Доведення. За теоремою 4.3 $l \leqslant m^h$. Прологарифмуємо цю нерівність за основою m: $\log_m l \leqslant h$. Оскільки h – ціле, то $h \geqslant \lceil \log_m l \rceil$. Тепер припустимо, що дерево повне та збалансоване. Вилучимо всі листки на рівні h (разом з інцидентними їм ребрами). Одержимо завершене m-арне дерево висотою $h-1$. Воно має m^{h-1} листків (див. задачу 12). Отже, $m^{h-1} < l \leqslant m^h$. Звідси випливає, що $h-1 < \log_m l \leqslant h$, тобто $h=\lceil \log_m l \rceil$.

4.2. Рекурсія. Обхід дерев. Префіксна та постфіксна форми запису виразів

Об'єкт називають *рекурсивним*, якщо він містить сам себе чи його означено за допомогою самого себе. Рекурсія – потужний засіб у математичних означеннях.

Приклад 4.5. У підрозділі 4.1 сформульовано означення повного бінарного дерева. Тепер означимо його рекурсивно:

а) o (ізольована вершина) – *повне бінарне дерево*;

б) якщо A та B – повні бінарні дерева, то конструкція, зображена на рис. 4.7, – *повне бінарне дерево*.

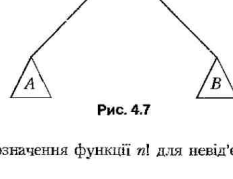


Рис. 4.7

Приклад 4.6. Рекурсивне означення функції $n!$ для невід'ємних цілих чисел має такий вигляд:

а) $0!=1$;

б) якщо $n>0$, то $n!=n(n-1)!$.

Очевидно, що важливість рекурсії пов'язана з тим, що вона дає змогу означити нескінченну множину об'єктів за допомогою скінченного висловлювання. Так само нескінченні обчислення можна описати за допомогою скінченної рекурсивної програми, навіть якщо вона не містить явних циклів. Проте найдоцільніше ви-

користовувати рекурсивні алгоритми тоді, коли розв'язувану задачу, обчислювану функцію чи оброблювані дані задано за допомогою рекурсії.

Чимало задач можна моделювати з використанням кореневих дерев. Поширене таке загальне формулювання задачі: виконати задану операцію Φ з кожною вершиною дерева. Тут Φ – параметр загальнішої задачі відвідування всіх вершин, або так званого *обходу дерева*. Розглядаючи розв'язування цієї задачі як єдиний послідовний процес відвідування вершин дерева в певному порядку, можна вважати їх розміщеними одна за одною. Опис багатьох алгоритмів істотно спрощується, якщо можна говорити про наступну вершину дерева, маючи на увазі якесь упорядкування. Є три принципи впорядкування вершин, які природно випливають зі структури дерева. Як і саму деревоподібну структуру, їх зручно формулювати за допомогою рекурсії.

Звертаючись до бінарного дерева, де R – корінь, A та B – ліве та праве піддерева (рис. 4.7), можна означити такі впорядкування.

1. Обхід *у прямому порядку* (*preorder*), або *зверху вниз*: R, A, B (корінь відвідують до обходу піддерев).
2. Обхід *у внутрішньому порядку* (*inorder*), або *зліва направо*: A, R, B.
3. Обхід *у зворотному порядку* (*postorder*), або *знизу вверх*: A, B, R (корінь відвідують після обходу піддерев).

Приклад 4.7. На рис. 4.8 зображено бінарне дерево. Різні обходи дадуть такі послідовності вершин:

- обхід у прямому порядку: *a b d e h o c f m p q*;
- обхід у внутрішньому порядку: *d b h e o a f c p m q*;
- обхід у зворотному порядку: *d h o e b f p q m c a*.

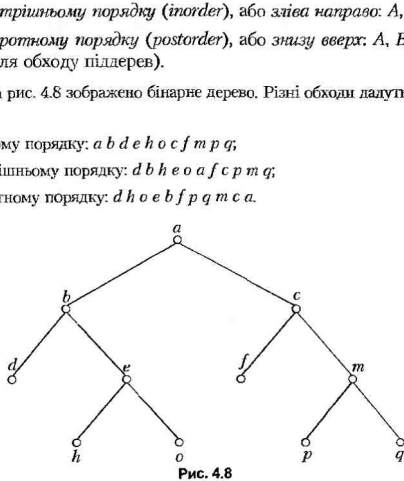


Рис. 4.8

Зазначені способи обходу бінарних дерев можна узагальнити й на довільні m-арні дерева. Обхід таких дерев у прямому порядку (зверху вниз) схематично зображено на рис. 4.9, у внутрішньому порядку (зліва направо) – на рис. 4.10, у зворотному (знизу вверх) – на рис. 4.11.

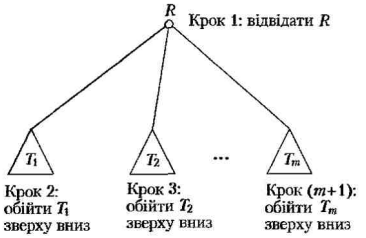


Рис. 4.9

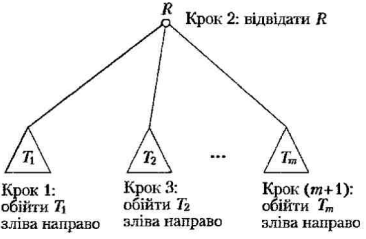


Рис. 4.10

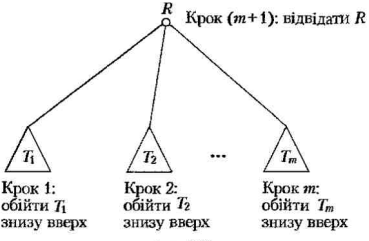


Рис. 4.11

Надзвичайно поширене в інформатиці застосування обходу дерев — зіставлення виразам (арифметичним, логічним тощо) дерев і побудова на цій основі різних форм запису виразів. Суть справи зручно пояснити на прикладі. Розглянемо арифметичний вираз

$$\left(a+\frac{b}{c}\right)*(d-e*f).$$

Подамо його у вигляді дерева. Послідовність дій відтворено на рис. 4.12. Рамкою на ньому обведено дерево, яке відповідає заданому арифметичному виразу. Внутрішнім вершинам цього дерева відповідають символи операцій, а листкам — операнди.

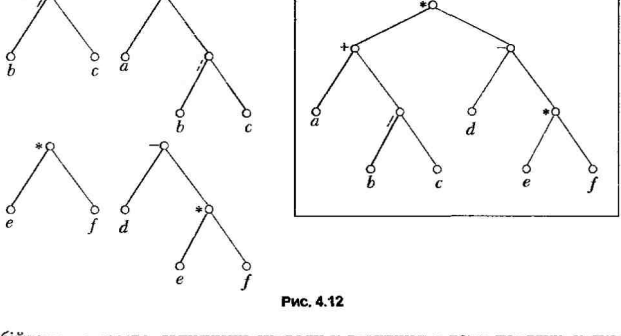


Рис. 4.12

Обійдемо це дерево, записуючи символи у вершинах у тому порядку, у якому вони зустрічаються в разі заданого способу обходу. Отримаємо такі три послідовності:

- у разі обходу в прямому порядку — *префіксний (польський) запис*

$$*+a\,/\,b\,c-d*e\,f;$$

- у разі обходу у внутрішньому порядку — *інфіксний запис* (поки що без дужок, потрібних для визначення порядку операцій)

$$a+b\,/\,c*d-e*f;$$

- у разі обходу в зворотному порядку — *постфіксний (зворотний польський) запис*

$$a\,b\,c\,/+d\,e\,f*-*.$$

Звернімося спочатку до інфіксної форми запису виразу. Без дужок вона неоднозначна: один запис може відповідати різним деревам. Наприклад, дереву, зображеному на рис. 4.13, у разі обходу зліва направо відповідає той самий вираз $a+b\,/\,c*d-e*f$, що й дереву на рис. 4.12 (у рамці), хоча на цих рисунках зображено різні дерева. Щоб уникнути неоднозначності інфіксної форми, використовують круглі дужки щоразу, коли зустрічають операцію. Повністю „одужкований" вираз, одержаний під час обходу дерева у внутрішньому порядку, називають *інфіксною формою* запису. Отже, для дерева з рис. 4.12 інфіксна форма така: $((a+(b\,/\,c))*(d-(e*f)))$; для дерева, зображеного на рис. 4.13, інфіксна форма має такий вигляд: $(a+(((b\,/\,(c*d))-e)*f))$.

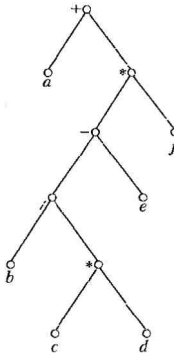


Рис. 4.13

Наведені міркування свідчать, що інфіксна форма запису виразів незручна. На практиці використовують префіксну та постфіксну форми, бо вони однозначно відповідають виразу й не потребують дужок. Ці форми запису називають ***польським записом*** (на честь польського математика й логіка Яна Лукасевича, українця за походженням).

Приклад 4.8. Розглянемо логічний вираз $(\neg(p \wedge q)) \sim (\neg p \vee \neg q)$. Послідовні етапи побудови відповідного бінарного дерева зображено на рис. 4.14.

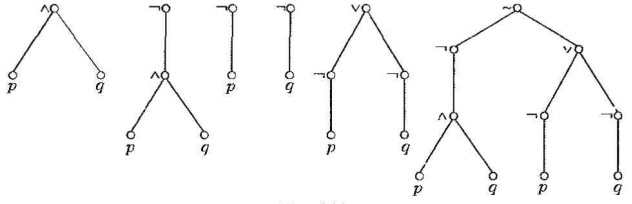


Рис. 4.14

Отримаємо такі форми запису виразу:

- інфіксна форма запису (відповідає обходу дерева виразу у внутрішньому порядку) —

$$(((p \wedge q)\neg) \sim ((p\neg) \vee (q\neg)));$$

- польський запис (відповідає обходу дерева виразу в прямому порядку) —

$$\sim \neg \wedge pq \vee \neg p \neg q;$$

- зворотний польський запис (відповідає обходу дерева виразу в зворотному порядку) —

$$pq \wedge \neg p \neg q \neg \vee \sim.$$

Для обчислення значення виразу в польському записі його проглядають справа наліво та знаходять два операнди разом зі знаком операції перед ними. Ці операнди та знак операції вилучають із запису, виконують операцію, а її результат записують на місце вилучених символів.

Приклад 4.9. Обчислимо значення виразу в польському записі (стрілка означає піднесення до степеня)

$$+-*2\,3\,5\,/\uparrow 2\,3\,4.$$

За сформульованим правилом виділимо $\uparrow$ 2 3, ці символи вилучимо й обчислимо $2\uparrow 3=8$; результат запишемо на місце вилучених символів:

$$+-*2\,3\,5\,/\,8\,4.$$

Продовжимо обчислення. Динаміку процесу відображено в табл. 4.1.

Таблиця 4.1

Крок	Вираз	Виділені символи	Виконання операції
1	+−*235/↑234	↑23	2↑3=8
2	+−*235/84	/84	8/4=2
3	+−*2352	*23	2*3=6
4	+−652	−65	6−5=1
5	+12	+12	1+2=3
6	3		

Для обчислення значення виразу в зворотному польському записі його проглядають зліва направо та виділяють два операнди разом зі знаком операції після них. Ці операнди та знак операції вилучають із запису, виконують операцію, а її результат записують на місце вилучених символів.

Приклад 4.10. Обчислимо значення виразу в зворотному польському записі

$$7\,2\,3*-4\uparrow 9\,3\,/+.$$

Динаміку обчислень відображено в табл. 4.2.

Таблиця 4.2

Крок	Вираз	Виділені символи	Виконання операції
1	723*−4↑93/+	23*	2*3=6
2	76−4↑93/+	76−	7−6=1
3	14↑93/+	14↑	1↑4=1
4	193/+	93/	9/3=3
5	13+	13+	1+3=4
6	4		

Оскільки польські записи однозначні та їх значення можна легко обчислити без сканування назад і вперед, їх широко використовують у комп'ютерних науках, особливо для конструювання компіляторів.

4.3. Бінарне дерево пошуку

Бінарне дерево забезпечує дуже зручний метод організації даних, у разі використання якого можна легко знайти будь-які конкретні дані чи виявити, що їх немає. Очевидно, що найнеефективніший спосіб пошуку – послідовний перегляд усіх даних. Справді, якщо потрібних даних немає, то для виявлення цього потрібно переглянути весь список. Бінарне дерево пошуку дає змогу уникнути цього. Єдина вимога – уведення для даних якогось лінійного порядку. Ним може бути, наприклад, алфавітний або числовий порядок. Лінійно впорядкувати можна теги, покажчики, файли чи інші ключі, які визначають дані. Але нас цікавитиме лише наявність якогось лінійного порядку.

У *бінарному дереві пошуку* кожній вершині присвоєно значення, яке називають *ключем*. Ключ – це елемент якоїсь лінійно впорядкованої множини: будь-які два її елементи можна порівняти (див. підрозділ 5.3).

Під час побудови бінарного дерева пошуку використовують його рекурсивну властивість, яку можна описати так. Кожна вершина розбиває дерево на два піддерева. Ліве піддерево містить лише ключі, менші від ключа цієї вершини, а праве – ключі, більші від ключа вершини. Ця властивість повторюється для кожної вершини (рис. 4.15, 4.16).

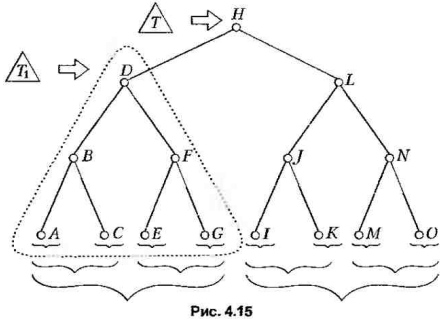


Рис. 4.15

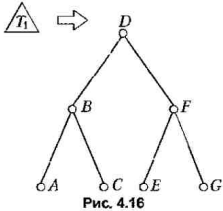


Рис. 4.16

Розглянемо алгоритм додавання об'єкта до дерева пошуку, який будує бінарне дерево пошуку. Почнемо з дерева, що містить лише одну вершину. Означимо її як корінь. Перший об'єкт списку присвоюємо кореню; це ключ кореня. Щоб додати новий об'єкт, виконуємо таку процедуру.

Алгоритм додавання об'єкта до дерева

Наведемо кроки алгоритму.

Крок 1. Почати з кореня.

Крок 2. Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.

Крок 3. Якщо об'єкт більший, ніж ключ у вершині, то перейти до правого сина.

Крок 4. Повторювати кроки 2 та 3, доки не досягнемо вершини, яку не визначено (тобто її немає).

Крок 5. Якщо досягнуто невизначену вершину, то визначити (тобто додати) вершину з новим об'єктом як ключем.

Приклад 4.11. Побудуємо бінарне дерево пошуку для такого списку слів в українському алфавіті: математика, фізика, географія, зоологія, метеорологія, біологія, психологія, хімія. Процес побудови зображено на рис. 4.17.

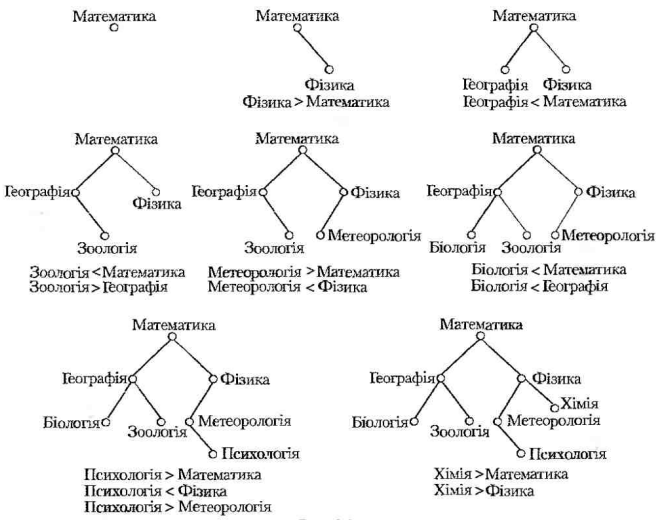


Рис. 4.17

Оскільки метод побудови дерева пошуку описано, легко зрозуміти, як відбувається пошук елемента в дереві. Застосовують переважно той самий підхід. Окрім перевірки того, чи даний об'єкт більший або менший, ніж ключ у вершині, перевіряють також, чи збігається даний об'єкт із ключем у вершині. Якщо так, то процес пошуку завершено, якщо ні — описані дії повторюють. Якщо ж досягнуто вершину, яку не визначено, то це означає, що даний об'єкт не зберігається в дереві. Нижче наведено алгоритм пошуку об'єкта в бінарному дереві.

Алгоритм пошуку об'єкта в дереві

Наведемо кроки алгоритму.

Крок 1. Почати з кореня.

Крок 2. Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.

Крок 3. Якщо об'єкт більший, ніж ключ у вершині, то перейти до правого сина.

Крок 4. Якщо об'єкт дорівнює ключу у вершині, то об'єкт знайдено; виконати потрібні дії й вийти.

Крок 5. Повторювати кроки 2. 3 та 4, доки не досягнемо вершини, яку не визначено.

Крок 6. Якщо досягнуто невизначену вершину, то даний об'єкт не зберігається в дереві; виконати потрібні дії й вийти.

Оцінимо обчислювальну складність алгоритмів включення та пошуку (локалізації) об'єкта в бінарному дереві пошуку. Припустимо, що є бінарне дерево пошуку T для списку з n об'єктів. Із дерева T утворимо повне бінарне дерево U, для чого додамо нові вершини (без ключів) так, щоб кожна вершина з ключем мала двох синів. Це показано на рис. 4.18, де вершини без ключів позначено подвійними кружечками.

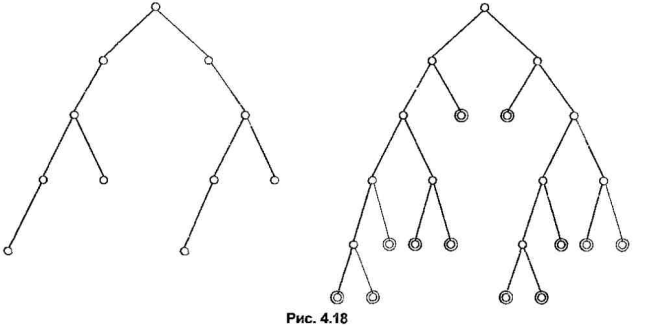


Рис. 4.18

Коли це зроблено, можна легко локалізувати об'єкт або додати новий об'єкт як ключ без додавання вершини. Найбільша кількість порівнянь, потрібних для

додавання нового об'єкта, дорівнює довжині найдовшого шляху в U від кореня до листка, тобто висоті дерева. Внутрішні вершини дерева U – це (всі) вершини дерева T. Отже, дерево U має n внутрішніх вершин. За теоремою 4.2 кількість листків у ньому дорівнює $2n+1-n=n+1$. Згідно з наслідком із теореми 4.3 висота дерева U більша чи дорівнює $\lceil \log(n+1) \rceil$. Отже, потрібно щонайменше $\lceil \log(n+1) \rceil$ порівнянь, щоб додати чи локалізувати довільний об'єкт. Якщо дерево збалансоване, то його висота дорівнює $\lceil \log(n+1) \rceil$. Отже, у цьому разі для додавання чи локалізації об'єкта потрібно не більше ніж $\lceil \log(n+1) \rceil$ порівнянь.

Бінарне дерево пошуку може розбалансуватись унаслідок додавання нових об'єктів, тому потрібен алгоритм *ребалансування* (тобто відновлення збалансованості). Проте процедура додавання об'єкта, яка відновлює збалансоване дерево, навряд чи завжди доцільна, бо відновлення збалансованості дерева після випадкового додавання – досить складна операція.

Тому розглядають також збалансованість із дещо послабленими вимогами; зокрема, означають *АВЛ-дерево* – бінарне дерево, у якому висоти двох піддерев кожної з його вершин відрізняються не більше ніж на одиницю. Такс означення збалансованості широко використовують на практиці; його ввели 1962 р. Г. М. Адельсон-Вельський і Є. М. Ландіс. Приклад АВЛ-дерева наведено на рис. 4.19.

Рис. 4.19

Для АВЛ-дерев є дуже ефективна процедура ребалансування [7]. Водночас висота АВЛ-дерева незалежно від кількості вершин ніколи не перевищує висоту збалансованого дерева більше, ніж на 45 %. Якщо позначити висоту АВЛ-дерева з n вершинами як $h(n)$, то $\log(n+1) \leqslant h(n) < 1.44 \log(n+2) - 0.328$.

4.4. Дерево прийняття рішень

Кореневі дерева можна застосовувати для розв'язування задач прийняття рішень. Для цього використовують *дерева прийняття рішень*, або *дерева рішень* [24, 29]. Кожному листку такого дерева поставлено у відповідність рішення зі скінченної множини відомих рішень, а кожній внутрішній вершині v – перевірку умови $P(v)$. Прийняття рішень за допомогою такого дерева полягає в побудові простого шляху від кореневої вершини до листка. Під час побудови шляху

виконують перевірку умов у внутрішніх вершинах дерева. Значення умови $P(v)$ у вершині v задає ребро, яке буде додано в шлях після вершини v. Кінцева вершина побудованого шляху відповідає прийнятому рішенню.

Приклад 4.12. Серед восьми монет одна фальшива, вона має меншу вагу. Знайдемо цю монету зважуваннями на балансових терезах за допомогою якнайменшої кількості зважувань. Розв'язок подано на рис. 4.20. Занумеруємо монети числами 1, 2, ..., 8. Порівнюємо вагу груп монет 1, 2, 3 та 4, 5, 6. Якщо одна із цих груп виявиться легшою, то це означає, що фальшива монета є в цій групі, і її можна виявити другим зважуванням. У разі балансу терез фальшива одна з монет 7, або 8, що також можна виявити другим зважуванням. Очевидно, що кількість зважувань дорівнює висоті дерева, тобто двом (рис. 4.21).

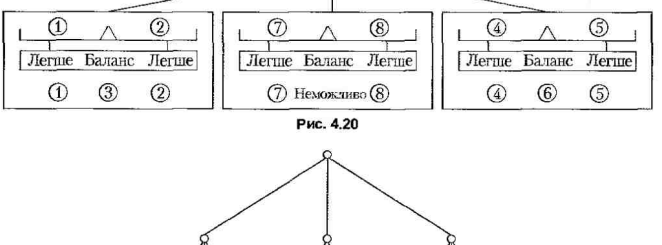


Рис. 4.20

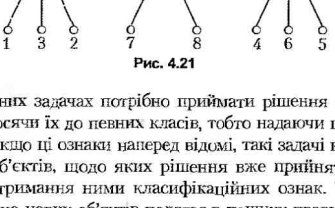


Рис. 4.21

У багатьох практичних задачах потрібно приймати рішення стосовно досліджуваних об'єктів, відносячи їх до певних класів, тобто надаючи цим об'єктам класифікаційних ознак. Якщо ці ознаки наперед відомі, такі задачі називають *задачами класифікації*. Для об'єктів, щодо яких рішення вже прийнято, узагалі кажучи, невідомі правила отримання ними класифікаційних ознак. Тому задача прийняття рішень відносно нових об'єктів полягає в пошуку правил, за якими надано такі ознаки.

Розглянемо таблицю даних, рядки якої – характеристики певного об'єкта чи явища. Нехай a – ім'я стовпця таблиці, яке називають *умовним атрибутом*; множину всіх умовних атрибутів цієї таблиці позначають як A. У стовпці містяться значення відповідного умовного атрибута; множину цих значень позначають як E_a. Таку таблицю називають *інформаційною системою*. Формально це пара $D=(W, A)$, де W – непорожня скінченна множина об'єктів або явищ, A – множина *умовних*

атрибутів. Кортеж значень умовних атрибутів кожного об'єкта з множини W називають *прикладом*.

Система прийняття рішень — це інформаційна система $D=(W, A\cup\{d\})$, де $d\notin A$ — *атрибут прийняття рішень*. Він може набувати декількох значень, однак найчастіше використовують двійкові значення {0, 1} або {Yes, No}. Систему прийняття рішень можна подати у вигляді таблиці, у якій останній стовпець позначено атрибутом прийняття рішень. Її називають *таблицею прийняття рішень*.

Розглянемо задачу класифікації як задачу прийняття рішень. У такій постановці задача побудови дерева рішень має такий вигляд: значення атрибута прийняття рішень d задає певний клас, до якого належить об'єкт із множини W, а множина значень атрибута d розбиває множину W на класи, кожний з яких задає прийняте рішення. Таблиці прийняття рішень поставимо у відповідність дерево прийняття рішень, внутрішнім вершинам якого відповідають перевірки значень атрибутів із множини умовних атрибутів A. Ребрам приписано значення атрибутів, що містяться в цих вершинах, а листки такого дерева відповідають класам.

Дерево рішень ставить у відповідність прикладу номер класу. Для цього використовують множину перевірок значень атрибутів даного прикладу у вершинах дерева на шляху від кореня до листка. Якщо атрибут прийняття рішень має декілька значень, то таке дерево може набути вигляду, наведеного на рис. 4.22. Тут 1, 2 та 3 — класифікаційні ознаки, або значення атрибута прийняття рішень. Наприклад, перевірці атрибута a_2 відповідають три його значення: ліве значення атрибута a_2 надає прикладу ознаку 3, середнє потребує для прийняття рішення перевірки атрибута a_4, а праве надає прикладу ознаку 1.

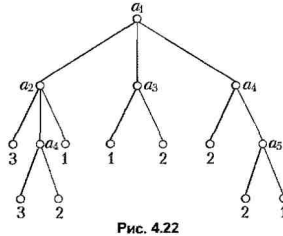


Рис. 4.22

Розглянемо приклад класифікації, виконаної за допомогою дерева прийняття рішень. На рис. 4.23 наведено класифікацію за атрибутами *температура* (зі значеннями *низька*, *висока*, *середня*) та *тиск* (зі значеннями *спадає*, *не змінюється* та *зростає*). Процедура прийняття рішення складається з побудови шляху від кореня до листків і перевірок, виконуваних у кожній внутрішній вершині. Ребра, які починаються в них вершинах, позначено можливими відповідями на питання перевірок. Результатами класифікації позначено листки дерева.

Рис. 4.23

Класифікатор записують у вигляді правил *if K then d=b*, де *K* — кон'юнкція підстановок вигляду *a=e* (під *підстановкою* тут розуміють надання атрибуту *a* значення *e*), *d=b* — результат класифікації (*d* — ім'я атрибута прийняття рішень, а *b* — його значення). Правила будують на шляху від кореня дерева до відповідного листка. Для дерева на рис. 4.23 множина правил така:

```
if (тиск=спадає) ∧ (температура=низька) then (погода=можливий сніг);
if (тиск=спадає) ∧ (температура=висока) then (погода=гроза);
if (тиск=спадає) ∧ (температура=середня) then (погода=можливий дощ);
if (тиск=не змінюється) then (погода=без змін);
if (тиск=зростає) then (погода=сонячно)
```

Нині для побудови дерев рішень на основі таблиць прийняття рішень активно застосовують алгоритм ID3, запропонований 1983 р. Куінланом (J. Quinlan). Цей алгоритм будує дерево рішень і на основі прикладів генерує правила.

Алгоритм ID3 будує дерево рішень від кореня. Кожній внутрішній вершині (включно з коренем) ставлять у відповідність певну перевірку з множини перевірок. Для побудови перевірок використовують той факт, що значення кожного умовного атрибута дає змогу розбити множину прикладів на підмножини, у яких усі приклади мають однакове значення цього атрибута. Якщо рекурсивно застосовувати це під час побудови дерева, ставлячи у відповідність кожній внутрішній вершині дерева підмножину, отриману внаслідок розбиття, то для кожної з одержаних підмножин буде побудовано нове піддерево. Процес побудови піддерев зупиняють, коли кожна підмножина складається лише з елементів з однаковими значеннями атрибута прийняття рішень. Листок дерева рішень задає клас, у який потрапляє об'єкт із множини *W*.

Наведемо кроки застосування алгоритму ID3 під час побудови дерева рішень зверху вниз від кореня до листків. Ці кроки застосовують рекурсивно в кореневій і в кожній із внутрішніх вершин дерева прийняття рішень. Процес продовжують доти, доки не буде реалізовано крок 1 для всіх вершин дерева. Множину всіх умовних атрибутів позначено *A*.

Алгоритм ID3

Наведемо кроки алгоритму.

Крок 1. Якщо вершині дерева прийняття рішень поставлено у відповідність множину прикладів R і всі ці приклади мають однакове значення атрибута прийняття рішення d, то цю вершину визначають як листок дерева та позначають цим значенням d.

Крок 2. Якщо для певної вершини умови кроку 1 не виконано, то розглядають множину умовних атрибутів A. Якщо $A \neq \varnothing$, то вибирають довільний атрибут $a \in A$; нехай множина його значень $E_a = \{e_1, e_2, \ldots, e_k\}$. Цю вершину позначають атрибутом a, сам атрибут a вилучають із множини A й виконують такі дії:

- у вершині a утворюють ребра $e_1, e_2, \ldots, e_k$ (їх кількість становить $|E_a|$, і кожне ребро позначають елементом множини E_a);
- множину прикладів R вершини a розбивають на підмножини з однаковим значенням атрибута a (усього таких підмножин k, значення атрибута a в першій підмножині дорівнює e_1, у другій — e_2, ..., у k-й — e_k);
- кінцю ребра e_j ставлять у відповідність підмножину прикладів, у яких значення атрибута a дорівнює e_j.

Крок 3. Якщо на кроці 2 виявиться, що $A = \varnothing$, то щодо вершини, яка має стати листком, приймають спеціальне рішення залежно від специфіки задачі.

Отже, кожна внутрішня вершина являє собою корінь піддерева, якому відповідають усі приклади з однаковим значенням одного з атрибутів і різними значеннями атрибута прийняття рішень. Кожному листку дерева відповідають приклади, що мають однакові значення одного з атрибутів і однакові значення атрибута прийняття рішень.

Приклад 4.13. Побудуємо дерево рішень для наведеної в табл. 4.3 інформації щодо проведення змагань із тенісу залежно від погоди. *Гра* — це атрибут прийняття рішень. Множина всіх умовних атрибутів $A = \{$*погода, температура, вологість, вітер*$\}$ відповідає кореневій вершині. Виберемо атрибут *погода* й позначимо ним кореневу вершину. Множина значень цього атрибута складається з трьох елементів: *сонце, хмари, дощ*. Кореневій вершині поставимо у відповідність три ребра, кожному з яких припишемо значення атрибута *погода*. Множину прикладів розіб'ємо на три підмножини, які відповідають значенням атрибута *погода*. Атрибут *погода* вилучимо з множини A й отримаємо множину $A = \{$*температура, вологість, вітер*$\}$, яка відповідає кожній із вершин 2, 3, 4 дерева, зображеного на рис. 4.24.

Рис. 4.24

Таблиця 4.3

ДЕНЬ	ПОГОДА	ТЕМПЕРАТУРА	ВОЛОГІСТЬ	ВІТЕР	ГРА
D1	Сонце	Спека	Висока	Слабкий	Ні
D2	Сонце	Спека	Висока	Сильний	Ні
D3	Хмари	Спека	Висока	Слабкий	Так
D4	Дощ	Помірно	Висока	Слабкий	Так
D5	Дощ	Холод	Норма	Слабкий	Так
D6	Дощ	Холод	Норма	Сильний	Ні
D7	Хмари	Холод	Норма	Сильний	Так
D8	Сонце	Помірно	Висока	Слабкий	Ні
D9	Сонце	Холод	Норма	Слабкий	Так
D10	Дощ	Помірно	Норма	Слабкий	Так
D11	Сонце	Помірно	Норма	Сильний	Так
D12	Хмари	Помірно	Висока	Сильний	Так
D13	Хмари	Спека	Норма	Слабкий	Так
D14	Дощ	Помірно	Висока	Сильний	Ні

Розглянемо вершину з номером 2. Їй відповідає підмножина прикладів {*D9*, *D11*}, які мають значення атрибута прийняття рішення *так*, і підмножина прикладів {*D1*, *D2*, *D8*}, які мають значення атрибута прийняття рішення *ні*. Виберемо наступний атрибут із множини *A*; нехай це *температура*. Позначимо ним вершину 2, побудуємо три ребра зі значеннями цього атрибута й розіб'ємо множину прикладів у вершині 2 на три підмножини, у кожній з яких значення температури однакові.

На рис. 4.25 у вершині 5 приклади *D1* і *D2* мають однакові значення (*ні*) атрибута *гра*. Тому цю вершину позначимо *ні*, і вона стане листком. Аналогічно, вершину 7 позначимо *так*, і вона також стане листком. Видучимо атрибут *температура* з множини *A*. Одержимо множину *A*={*вологість*, *вітер*}. Вершину 6 позначимо атрибутом *вологість*.

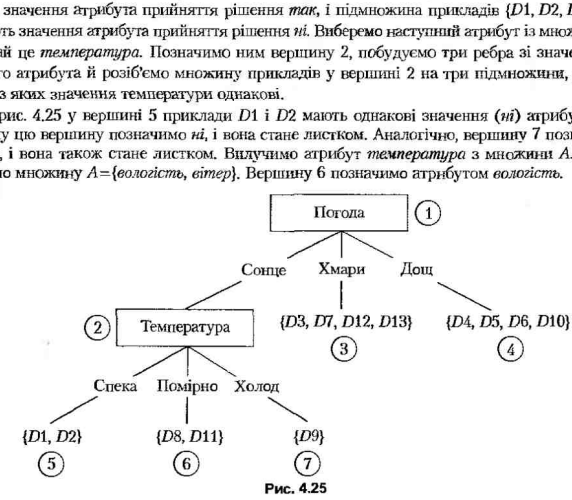


Рис. 4.25

На рис. 4.26 показано два листки, які відповідають значенням *так* і *ні* відповідно для прикладів *D11* і *D8*. Видучимо атрибут *вологість* із множини *A* й отримаємо *A*={*вітер*}. Тепер розглянемо вершину 3 на рис. 4.24. Усі приклади, приписані цій вершині, мають

значення *так* атрибута *гра*, тому ця вершина являє собою листок; позначимо її *так*. У вершині 4 приклади мають різні значення атрибута *гра*, тому припишемо їй атрибут *вітер*, який має два значення: *слабкий* і *сильний*. Ці значення припишемо ребрам, інцидентним вершині 4. Множину прикладів розіб'ємо на дві підмножини: {*D*4, *D*5, *D*10} зі значенням *так* і {*D*6} зі значенням *ні*. Остаточно дерево рішень набуде вигляду, зображеного на рис. 4.27.

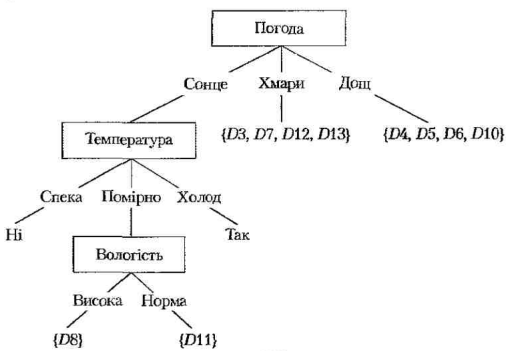


Рис. 4.26

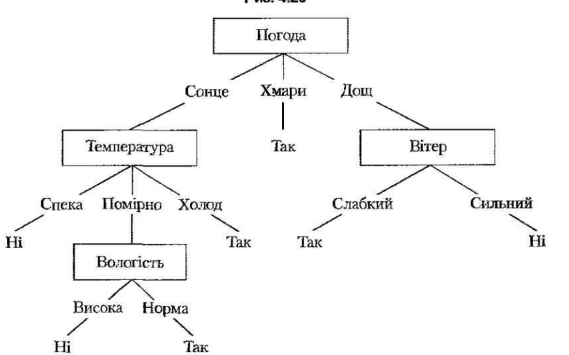


Рис. 4.27

Приклад 4.14. Вибираючи іншу послідовність атрибутів і обходу вершин, можна одержати інше дерево рішень. Дерево рішень, наведене на рис. 4.28, має менше вершин (немає атрибута *температура*). Для нього множина правил має такий вигляд:

```
If (погода=сонце) ∧ (вологість=висока) then (гра=ні);
If (погода=сонце) ∧ (вологість=норма) then (гра=так).
```

```
If (погода=хмари) then (гра=так);
If (погода=дощ) ∧ (вітер=слабкий) then (гра=так);
If (погода=дощ) ∧ (вітер=сильний) then (гра=ні).
```

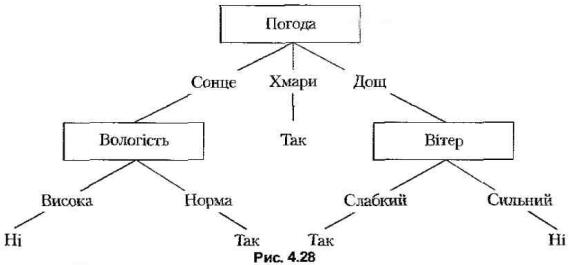


Рис. 4.28

Отже, за наявними прикладами будують дерево рішень. На його основі виписують множину правил, використовуваних для прийняття рішень у нових прикладах.

Порівнявши приклади 4.13 і 4.14, зазначимо, що на практиці реалізують версію алгоритму ID3, у якій аналізують важливість певних атрибутів для прийняття рішення. У такому разі під час вибору кореня поточного піддерева оцінюють частку інформації, додаваної кожним атрибутом. Потім вибирають той атрибут, який надає найбільше нової інформації.

Алгоритм ID3 — один із найважливіших методів індуктивного відновлення правил за прикладами, який забезпечує автоматичну побудову баз знань діагностичних експертних систем.

4.5. Бектрекінг (пошук із поверненнями)

Опишемо загальний метод, який дає змогу значно зменшити обсяг обчислень в алгоритмах типу повного перебору всіх можливостей. Щоб застосувати цей метод, розв'язок задачі повинен мати вигляд скінченної послідовності $(x_1, ..., x_n)$. Головна ідея методу полягає в тому, що розв'язок будують поступово, починаючи з порожньої послідовності λ (довжиною 0). Загалом, якщо є частковий (неповний) розв'язок $(x_1, ..., x_i)$, де $i < n$, то намагаємося знайти таке допустиме значення x_{i+1}, що можна продовжувати $(x_1, ..., x_i, x_{i+1})$ до одержання повного розв'язку. Якщо таке допустиме, але ще не використане значення x_{i+1} існує, то долучаємо цю нову компоненту до часткового розв'язку та продовжуємо процес для послідовності $(x_1, ..., x_i, x_{i+1})$. Якщо такого значення x_{i+1} немає, то повертаємося до попередньої послідовності $(x_1, ..., x_{i-1})$ і продовжуємо процес, шукаючи нове, іще не використане значення x'_i. Тому процес називають *бектрекінгом* (англ. backtracking — пошук із поверненнями).

Роботу цього алгоритму можна інтерпретувати як процес обходу якогось дерева. Кожна його вершина відповідає якійсь послідовності $(x_1, ..., x_i)$, причому верши-

ни, які відповідають послідовностям вигляду $(x_1, ..., x_i, y)$, – сини цієї вершини. Корінь дерева відповідає порожній послідовності.

Виконується обхід цього дерева пошуком углиб. Орім того, задають предикат P, означений на всіх вершинах дерева. Якщо $P(v) = F$, то вершини піддерева з коренем у вершині v не розглядають, і обсяг перебору зменшується. Предикат $P(v)$ набуває значення F тоді, коли стає зрозумілим, що послідовність $(x_1, ..., x_i)$, яка відповідає вершині v, ніяким способом не можна добудувати до повного розв'язку.

Проілюструємо застосування алгоритму бектрекінг на конкретних прикладах.

Приклад 4.15. Побудова гамільтонових циклів у графі [23]. Починаємо з довільної вершини. Будуємо шлях без повторення вершин, доки це можливо. Якщо вдалося пройти всі вершини, то перевіряємо, чи існує ребро, що з'єднує останню й початкову вершини цього шляху. Якщо описаний процес у певний момент неможливо продовжити, то повертаємося на одну вершину назад і намагаємося продовжити побудову шляху (без повторення вершин) іншим способом.

Пошук усіх гамільтонових циклів у графі з п'ятьма вершинами (рис. 4.29) можна проілюструвати за допомогою дерева, зображеного на рис. 4.30. Роботу алгоритму почато з одноелементної послідовності, бо в циклі вибір першої вершини неістотний. Розв'язки задачі обведено, і до них у прямокутних рамках приписано відповідні гамільтонові цикли.

Отже, замість побудови й аналізу $5! = 120$ послідовностей довжиною 5 вершин графа, зображеного на рис. 4.28, ми розглянули лише 23 послідовності довжиною від 1 до 5.

Рис. 4.29

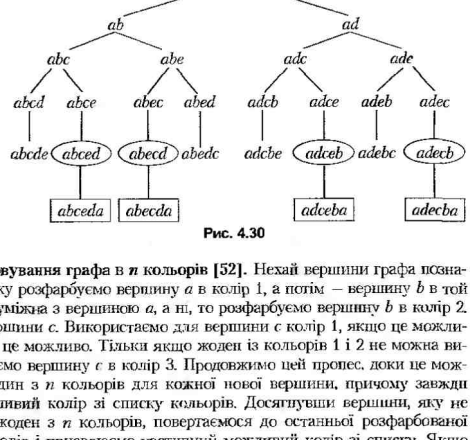


Рис. 4.30

Приклад 4.16. Розфарбовування графа в n кольорів [52]. Нехай вершини графа позначено як $a, b, c, ...$. Спочатку розфарбуємо вершину a в колір 1, а потім – вершину b в той самий колір, якщо b не суміжна з вершиною a, а ні, то розфарбуємо вершину b в колір 2. Перейдемо до третьої вершини c. Використаємо для вершини c колір 1, якщо це можливо, а ні, то колір 2, якщо це можливо. Тільки якщо жоден із кольорів 1 і 2 не можна використовувати, розфарбуємо вершину c в колір 3. Продовжимо цей процес, доки це можливо, використовуючи один з n кольорів для кожної нової вершини, причому завжди братимемо перший можливий колір зі списку кольорів. Досягнувши вершини, яку не можна розфарбувати в жоден з n кольорів, повертаємося до останньої розфарбованої вершини, відміняємо її колір і присвоюємо наступний можливий колір зі списку. Якщо

й це неможливо, то ще раз повертаємося до попередньої вершини, відміняємо її колір і намагаємося присвоїти новий колір, наступний можливий зі списку. Цей процес продовжуємо аналогічно. Якщо розфарбування в n кольорів існує, то така процедура дає змогу знайти його. На рис. 4.31 зображено граф і процес присвоювання трьох кольорів вершинам із використанням алгоритму бектрекінг.

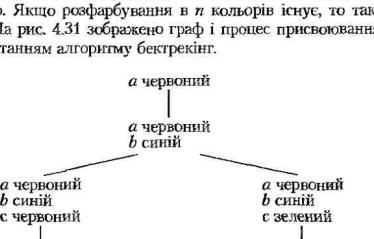


Рис. 4.31

Приклад 4.17. Задача про n ферзів [7, 52]. Як n ферзів можна розмістити на шахівниці $n \times n$ так, щоб жодні два не били один одного? Для розв'язання цієї задачі потрібно визначити n позицій на шахівниці так, щоб жодні дві позиції не були в одному рядку, в одному стовпці та на одній діагоналі. Діагональ містить усі позиції з координатами (i, j) такі, що $i+j=m$ для якогось m, або $i-j=m$ (тут i — номер рядка, j — номер стовпця). Починаємо з порожньої шахівниці. На $(k+1)$-му кроці намагаємося розмістити нового ферзя в $(k+1)$-му стовпці, причому в перших k стовпцях уже є ферзі. Перевіряємо клітинки в $(k+1)$-му стовпці, починаючи з верхньої. Шукаємо таку позицію для ферзя, щоб він не був у рядку та на діагоналі з тими ферзями, які вже є на шахівниці. Якщо це неможливо, то повертаємося до місця ферзя на попередньому k-му кроці та розміщаємо цього ферзя на наступному можливому рядку в цьому k-му стовпці, якщо такий рядок є, а ні, то повертаємося до ферзя в $(k-1)$-му стовпці. Алгоритм бектрекінг для $n=4$ проілюстровано на рис. 4.32.

Кожній вершині дерева на рис. 4.32 відповідає послідовність довжиною від 0 до 4. Її k-й член дорівнює номеру клітинки з ферзем у k-му стовпці. Наприклад, вершинам шляху, який веде до розв'язку, відповідають такі послідовності: λ, (2), (2, 4). (2. 4, 1), (2. 4. 1, 3).

Приклад 4.18. Суми елементів підмножин [52]. Задано множину натуральних чисел $\{x_1, x_2, ..., x_n\}$. Потрібно знайти її підмножину, сума елементів якої дорівнює заданому числу M.

Починаємо з порожньої множини. Нагромаджуємо суму, послідовно добираючи доданки. Число з послідовності $x_1, x_2, ..., x_n$ долучають до суми, якщо сума після додавання цього числа не перевищує M. Якщо сума настільки велика, що додавання будь-якого нового

числа перевищує M, то повертаємось і змінюємо останній доданок у сумі. На рис. 4.33 проілюстровано алгоритм бектрекінг для задачі відшукання підмножини множини {31, 27, 15, 11, 7, 5} із сумою 39.

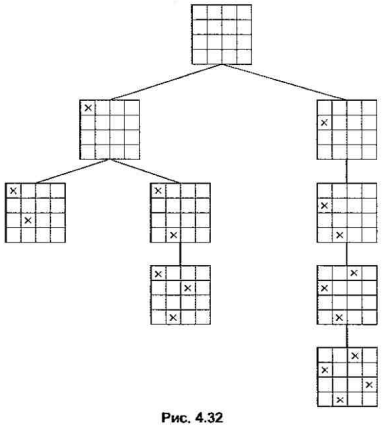


Рис. 4.32

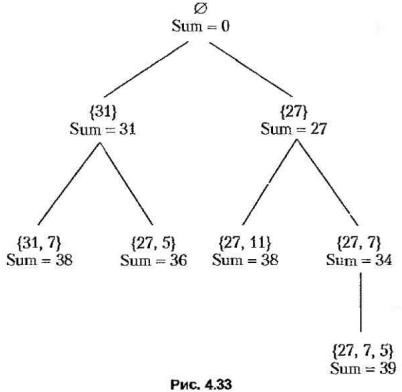


Рис. 4.33

Приклад 4.19. Побудова всіх максимальних незалежних множин вершин у простому графі $G=(V, \Gamma)$ [20, 35]. Тут граф подано як пару, утворену множиною вершин V та

відповідністю Г, котра показує, як зв'язані між собою вершини (див. підрозділ 3.3). Почнемо з порожньої множини й будемо додавати до неї вершини зі збереженням незалежності. Нехай S_k — уже отримана множина з k вершин, Q_k — множина вершин, котрі можна додати до S_k, тобто $S_k \cap \Gamma(Q_k) = \varnothing$. Серед вершин Q_k розрізняють ті, котрі вже було використано для розширення S_k (їх позначають Q_k^-), і ті, котрі ще не використано (їх позначають Q_k^+). Загальна схема алгоритму бектрекінг для задачі побудови максимальних незалежних множин вершин у простому графі має такий вигляд.

Прямий крок від k до $k+1$ полягає у виборі вершини $x \in Q_k^+$:

$$S_{k+1} := S_k \cup \{x\}; \qquad Q_{k+1}^- := Q_k^- \setminus \Gamma(x); \qquad Q_{k+1}^+ := Q_k^+ \setminus (\Gamma(x) \cup \{x\}).$$

Крок повернення від $k+1$ до k:

$$S_k := S_{k+1} \setminus \{x\}; \qquad Q_k^+ := Q_k^+ \setminus \{x\}; \qquad Q_k^- := Q_k^- \cup \{x\}.$$

Якщо множина вершин S_k максимальна, то $Q_k^+ = \varnothing$. Якщо $Q_k^- \neq \varnothing$, то множину S_k було розширено раніше, і вона не максимальна. Отже, перевірку максимальності задають такою умовою: $Q_k^+ = Q_k^- = \varnothing$.

Доцільно намагатися почати кроки повернення якомога раніше, бо це обмежить розміри „непотрібної" частини дерева пошуку. У зв'язку з цим зауважимо таке. Нехай $v \in Q_k^-$ і $\Gamma(v) \cap Q_k^+ = \varnothing$. Цю вершину неможливо вилучить з Q_k^-, оскільки можна вилучати лише вершини, суміжні з вершинами множини Q_k^+. Отже, існування такої вершини v, що $v \in Q_k^-$ і $\Gamma(v) \cap Q_k^+ = \varnothing$ — достатня умова для повернення. Окрім того, $k \leqslant n-1$.

Алгоритм побудови всіх максимальних незалежних множин вершин у простому графі $G = (V, \Gamma)$

Наведемо кроки алгоритму.

Крок 1. Ініціалізація. Виконати $S_0 := \varnothing$, $Q_0^- := \varnothing$, $Q_0^+ := V$, $k = 0$.

Крок 2. Прямий крок. Вибрати вершину $x \in Q_k^+$ і побудувати, як описано вище, множини S_{k+1}, Q_{k+1}^-, Q_{k+1}^+; при цьому залишити Q_k^- та Q_k^+ без змін. Виконати $k := k+1$.

Крок 3. Перевірка. Якщо виконано умову $\exists v \in Q_k^- : \Gamma(v) \cap Q_k^+ = \varnothing$, то перейти до кроку 5, інакше до кроку 4.

Крок 4. Якщо $Q_k^+ = Q_k^- = \varnothing$, то надрукувати максимальну незалежну множину S_k та перейти до кроку 5. Якщо $Q_k^+ = \varnothing$, а $Q_k^- \neq \varnothing$, то перейти до кроку 5. Інакше перейти до кроку 2.

Крок 5. Крок повернення. Виконати $k := k-1$. Вилучити вершину x із S_{k+1}, щоб одержати S_k. Виправити Q_k^- та Q_k^+, для чого вилучити вершину x із множини Q_k^+ і долучити її до Q_k^-. Якщо $k=0$ та $Q_0^+ = \varnothing$, то зупинитися (на цей момент уже буде надруковано всі максимальні незалежні множини). Інакше перейти до кроку 3.

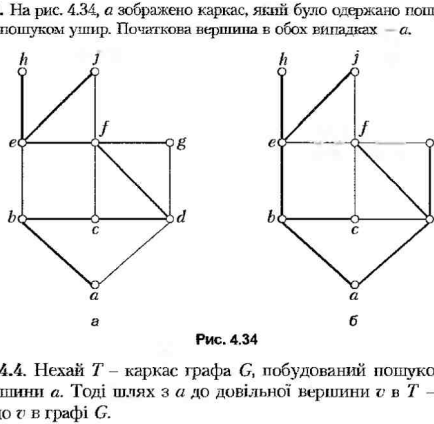
Унаслідок зв'язку між кліками й незалежними множинами (див. теорему 3.21) цей алгоритм можна використати також і для побудови максимальних клік.

4.6. Каркаси (з'єднувальні дерева)

Нехай G – простий зв'язний граф. *Каркасом*, або *з'єднувальним деревом*, графа G називають його підграф, який являє собою дерево та містить усі вершини графа G. Нехай граф G має n вершин і m ребер. Щоб отримати каркас, можна використати процедуру вилучення ребер, які належать простим циклам. Очевидно, що потрібно вилучити $\gamma(G)=m-(n-1)=m-n+1$ ребер. Число $\gamma(G)$ називають *цикломатичним числом* графа G. Із теореми 3.6 випливає, що $\gamma(G)\geqslant 0$. Цикломатичне число – це певною мірою числова характеристика зв'язності графа; цикломатичне число дерева дорівнює 0.

Побудова каркаса – поширена задача. Алгоритм, який полягає у вилученні ребер із простих циклів, неефективний для комп'ютерної реалізації. Для його виконання потрібно ідентифікувати прості цикли, а це складна задача. Ефективний із погляду комп'ютерної реалізації алгоритм побудови каркаса – це послідовний добір ребер у каркас. Це можна зробити за допомогою обходу графа G як пошуком углиб, так і пошуком ушир. Під час виконання цих алгоритмів природним способом будують каркас (див. підрозділ 3.9, ребра каркаса позначено потовщеними лініями). Якщо до протоколу обходу графа додати четвертий стовпчик, то туди можна записувати ребра, які під час роботи алгоритму позначають потовщеною лінією, і, отже, включають у каркас.

Приклад 4.20. На рис. 4.34, *а* зображено каркас, який було одержано пошуком углиб, а на рис. 4.34, *б* – пошуком ушир. Початкова вершина в обох випадках – a.



а *б*

Рис. 4.34

ТЕОРЕМА 4.4. Нехай T – каркас графа G, побудований пошуком ушир, починаючи з вершини a. Тоді шлях з a до довільної вершини v в T – найкоротший шлях від a до v в графі G.

Доведення ґрунтується на аналізі алгоритму пошуку вшир у графі G. Пропонуємо виконати його як вправу.

Зауваження. Використавши для побудови дерева найкоротших шляхів від вершини a алгоритм Дейкстри (уважаємо, що довжина кожного ребра дорівнює 1),

отримаємо те саме дерево, що й за алгоритмом пошуку вшир. Проте складність цього алгоритму становить $O(n^2)$ чи $O(m \log n)$ залежно від способу подання графа, а складність алгоритму пошуку вшир — $O(m+n)$, де n — кількість вершин, m — кількість ребер у графі G. Проте алгоритм Дейкстри більш універсальний, він придатний для будь-яких додатних довжин ребер (а не тільки 1).

Тепер розглянемо одну важливу задачу, пов'язану з ідеєю оптимізації. Нехай G — зв'язний зважений граф (див. підрозділ 3.8). Потрібно описати алгоритм побудови каркаса T, у якому сума ваг ребер $M(T)=\Sigma w(e)$ якнайменша (суму Σ беруть за всіма ребрами дерева T). Каркас T називають *мінімальним*. Розв'язати цю задачу дає змогу *алгоритм Краскала* (J. Kruskal).

Алгоритм Краскала

Нехай G — зв'язний зважений граф з n вершинами та m ребрами. Виконати такі дії.

1. Вибрати ребро e_1, яке має в графі G найменшу вагу.
2. Визначити послідовність ребер $e_2, e_3, ..., e_{n-1}$; на кожному кроці вибирати відмінне від попередніх ребро з найменшою вагою й таке, що не утворює простих циклів з уже вибраними ребрами. Одержане дерево T з множиною ребер $ET=\{e_1, e_2, e_3, ..., e_{n-1}\}$ — мінімальний каркас графа G.

Розглянемо одну з можливих реалізацій алгоритму Краскала.

Крок 1. Упорядкувати множину ребер за зростанням ваг: $e_1, e_2, e_3, ..., e_m$.

Крок 2. Утворити розбиття множини вершин на одноелементні підмножини: $\{\{v_1\}, \{v_2\}, ..., \{v_n\}\}$.

Крок 3. Вибирати таке чергове ребро з упорядкованої послідовності ребер, що його кінці містяться в різних множинах розбиття (це забезпечить відсутність простих циклів). Якщо вибрано ребро $e_i=\{v, w\}$, то множини розбиття, які містять вершини v та w, об'єднують в одну множину.

Крок 4. Якщо вже вибрано $(n-1)$ ребро (у такому разі всі підмножини розбиття об'єднаються в одну), то зупинитись, бо вибрані ребра утворюють мінімальний каркас. Інакше перейти до кроку 3.

Приклад 4.21. На рис. 4.35 зображено граф, ребра якого пронумеровано за зростанням ваг. Мінімальний каркас виділено потовщеними лініями. Процес відбору ребер наведено в табл. 4.4. Мінімальний каркас утворюють ребра $e_1, e_2, e_3, e_5, e_8, e_{11}$.

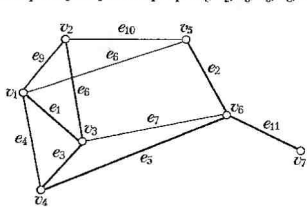


Рис. 4.35

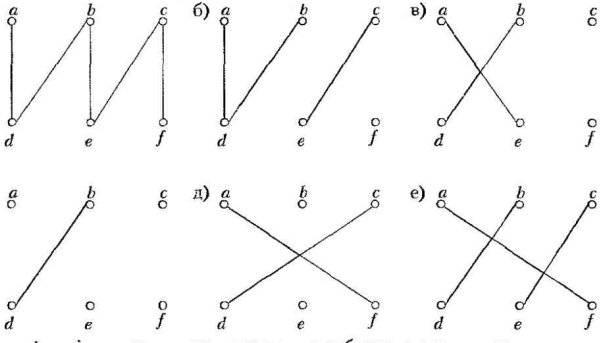
Таблиця 4.4

Ребро	Розбиття множини вершин	Відбір ребра у мінімальний каркас
$e_1=\{v_1,v_3\}$	$\{\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}\}$	e_1
$e_2=\{v_5,v_6\}$	$\{\{v_1,v_3\}, \{v_2\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}\}$	e_2
$e_3=\{v_3,v_4\}$	$\{\{v_1,v_3\}, \{v_2\}, \{v_4\}, \{v_5,v_6\}, \{v_7\}\}$	e_3
$e_4=\{v_1,v_4\}$	$\{\{v_1,v_3,v_4\}, \{v_2\}, \{v_5,v_6\}, \{v_7\}\}$	–
$e_5=\{v_4,v_6\}$	$\{\{v_1,v_3,v_4\}, \{v_2\}, \{v_5,v_6\}, \{v_7\}\}$	e_5
$e_6=\{v_1,v_5\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	–
$e_7=\{v_5,v_6\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	–
$e_8=\{v_2,v_3\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	e_8
$e_9=\{v_1,v_2\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	–
$e_{10}=\{v_2,v_5\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	–
$e_{11}=\{v_6,v_7\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	e_{11}
	$\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$	

Алгоритм Краскала належить до *жадібних* [23]. Так називають алгоритми оптимізації, які на кожному кроці вибирають найкращий із можливих варіантів.

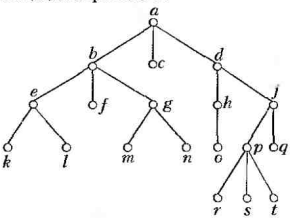
Контрольні запитання та завдання

1. Які з наведених нижче графів являють собою дерева?



2. Дати відповідь на запитання щодо дерева, зображеного на рисунку.
   а) Яка вершина являє собою корінь?
   б) Які вершини внутрішні?
   в) Які вершини являють собою листки?
   г) Які вершини – сини вершини *j*?

д) Яка вершина — батько вершини n?
е) Які вершини — предки вершини n?
ж) Які вершини — нащадки вершини b?

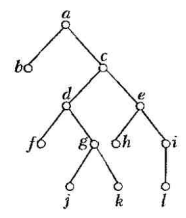


3. Визначити рівень кожної вершини та висоту дерева, зображеного на рисунку.
4. Для яких значень $m > 0$ й $n > 0$ повний дводольний граф $K_{m,n}$ являє собою дерево?
5. Скільки ребер має дерево з 1000 вершинами?

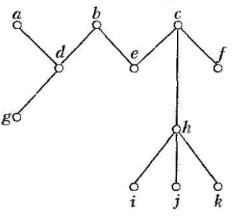
У задачах 6–10 використати теорему 4.2.

6. Скільки вершин має повне 5-арне дерево зі 100 внутрішніми вершинами?
7. Скільки ребер має повне бінарне дерево з 1000 внутрішніх вершин?
8. Скільки листків має повне 3-арне дерево зі 100 вершинами?
9. У шаховому турнірі беруть участь 1000 гравців. Скільки ігор потрібно зіграти для визначення переможця, якщо турнір проводять за олімпійською системою (той, хто програв, вибуває)?
10. Чи існує повне m-арне дерево T, яке має 84 листки та висоту 3?
11. Побудувати завершене бінарне дерево висотою 4 та завершене 3-арне дерево висотою 3.
12. Довести, що завершене m-арне дерево висотою h має $(m^{h+1} - 1)/(m - 1)$ вершин і m^h листків.
13. *Ексцентриситет* вершини в некореневому дереві — це довжина найдовшого простого шляху, який починається в цій вершині. Вершину називають *центром*, якщо вона має найменший ексцентриситет. Знайти центри дерев:

а)

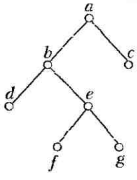


б)

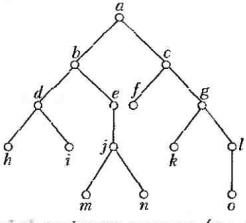


14. Довести, що центр у некореневому дереві можна знайти як корінь кореневого дерева з мінімальною висотою.
15. Довести, що дерево має один або два центри, суміжні між собою. Навести приклад дерева, яке має два центри.
16. Записати послідовності вершин упорядкованих кореневих дерев, наведених нижче, в разі їх обходу в прямому, зворотному та внутрішньому порядку.

а)

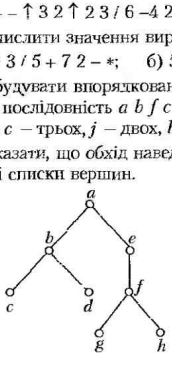


б)

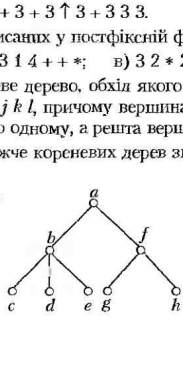


17. Побудувати кореневі дерева, які відповідають виразам $(x + x * y) + (x/y)$ та $x + ((x * y + x)/y)$. Записати ці вирази в інфіксній, префіксній та постфіксній формах.
18. Побудувати кореневі дерева, які відповідають виразам $(\neg(p \wedge q)) \sim ((\neg p) \vee (\neg q))$ та $((\neg p) \wedge (q \sim (\neg p))) \vee (\neg q)$. Записати ці вирази в інфіксній, префіксній та постфіксній формах.
19. Зобразити впорядковане кореневе дерево, яке відповідає кожному з виразів, записаних у префіксній формі. Записати їх у постфіксній формі:
а) ↑ + 2 3 – 5 1; б) + * + – 5 3 2 1 4; в) * / 9 3 + * 2 4 – 7 6.
20. Обчислити значення виразів, записаних у префіксній формі:
а) – * 2 / 8 4 3; б) ↑ – * 3 3 * 4 2 5;
в) + – ↑ 3 2 ↑ 2 3 / 6 –4 2; г) * + 3 + 3 ↑ 3 + 3 3 3.
21. Обчислити значення виразів, записаних у постфіксній формі:
а) 9 3 / 5 + 7 2 – *; б) 5 2 1 – – 3 1 4 + + *; в) 3 2 * 2 ↑ 5 3 – 8 4 / * –.
22. Побудувати впорядковане кореневе дерево, обхід якого в прямому порядку дає послідовність $a\ b\ f\ c\ g\ h\ i\ d\ e\ j\ k\ l$, причому вершина a має чотирьох синів, c – трьох, j – двох, b й e – по одному, а решта вершин – листки.
23. Показати, що обхід наведених нижче кореневих дерев зверху вниз дає однакові списки вершин.

а)

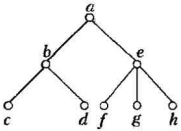


б)

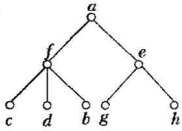


24. Показати, що обхід наведених нижче кореневих дерев знизу вверх дає однакові списки вершин.

а)



б)



У задачах 25, 26 доведення виконати методом математичної індукції.

25. Довести, що впорядковане кореневе дерево можна однозначно задати списком вершин, одержаним обходом у прямому порядку, якщо для кожної вершини зазначити кількість її синів.

26. Довести, що впорядковане кореневе дерево можна однозначно задати списком вершин, отриманим обходом у зворотному порядку, якщо для кожної вершини зазначити кількість її синів.

27. Побудувати бінарне дерево пошуку для слів *banana, peach, pear, apple, coconut, mango, papaya.*

28. Скільки потрібно порівнянь, щоб знайти чи додати кожне зі слів до дерева пошуку із задачі 27:

а) *pear*; б) *banana*;

в) *plum*; г) *orange*.

29. Використати бектрекінг для відшукання гамільтонових циклів у графах із задачі 50 розділу 3.

30. Використовуючи бектрекінг, розфарбувати в три кольори графи із задач 73 і 70 розділу 3.

31. Використовуючи бектрекінг, розв'язати задачу про n ферзів для значень n, що дорівнюють 3, 5 і 6.

32. Використати бектрекінг для відшукання всіх максимальних незалежних множин вершин графів із задачі 74 розділу 3.

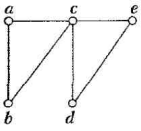
33. Звести задачу про n ферзів до задачі про найбільшу незалежну множину вершин у графі. Проілюструвати для значень n, що дорівнюють 3 та 4.

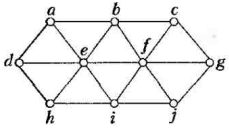
34. Використовуючи бектрекінг, знайти підмножину (якщо вона існує) множини {27, 24, 19, 14, 11, 8} із зазначеною сумою:

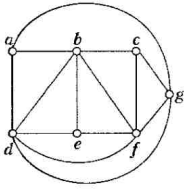
а) 20; б) 41; в) 60.

35. Зв'язний граф G має n вершин і m ребер. Скільки ребер потрібно вилучити з графа G, щоб одержати каркас?

36. Знайти каркас для кожного з наведених нижче графів способом вилучення ребер із простих циклів.

а) 

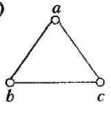
б) 

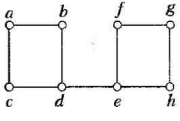
в) 

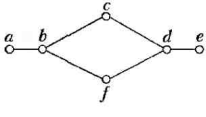
37. Побудувати каркаси для кожного з наведених нижче графів. Знайти цикломатичне число кожного графа:

а) K_5; б) $K_{1,6}$; в) C_5; г) $K_{4,4}$; д) Q_3; е) W_5.

38. Для простих графів побудувати всі можливі каркаси.

а) 

б) 

в) 

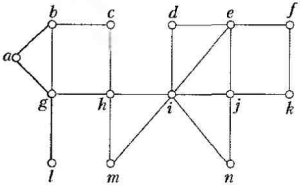
39. Довести *теорему Келі*: кількість позначених дерев з n вершинами дорівнює n^{n-2}. Зазначимо, що всі позначені дерева з n вершинами — це всі каркаси графа K_n.

40. Зобразити всі позначені дерева з n вершинами для значень n, що дорівнюють 2, 3 та 4.

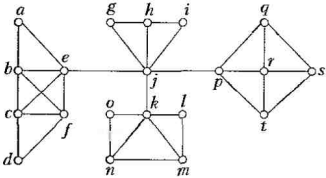
41. Скільки існує неізоморфних дерев з n вершинами для значень n, що дорівнюють 3, 4 та 5? Зобразити всі ці дерева.

42. Використовуючи обхід графа пошуком углиб, побудувати каркаси для графів, наведених нижче. Як початкову вибрати вершину a.

а)



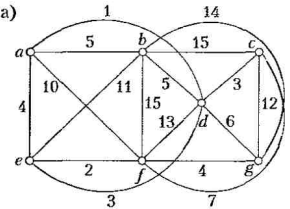
б)



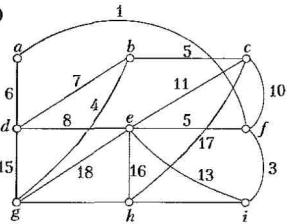
43. Розв'язати задачу 42, використовуючи пошук ушир.

44. За алгоритмом Краскала для наведених нижче графів побудувати мінімальний каркас.

а)



б)



Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано матрицю суміжності простого графа. Визначити, чи є цей граф деревом.
2. Простий граф задано списками суміжності. Визначити, чи є цей граф деревом.
3. Задано матрицю суміжності кореневого дерева та якусь його вершину. Знайти батька, синів, предків, нащадків і рівень цієї вершини.

4. Задано список ребер кореневого дерева та якусь його вершину. Знайти батька, синів, предків, нащадків і рівень цієї вершини.
5. Задано список об'єктів. Побудувати бінарне дерево пошуку, що містить ці об'єкти.
6. Задано бінарне дерево пошуку й об'єкт. Локалізувати цей об'єкт у дереві пошуку чи додати його до дерева.
7. Задано список ребер некореневого дерева. Знайти центри цього дерева (див. задачу 13).
8. Задано впорядкований список ребер упорядкованого кореневого дерева. Записати його вершини в разі обходу в прямому, зворотному та внутрішньому порядку.
9. Задано арифметичний вираз у польському записі. Обчислити його значення.
10. Задано арифметичний вираз у зворотному польському записі. Обчислити його значення.
11. Задано матрицю суміжності простого зв'язного графа. Побудувати каркас цього графа з використанням пошуку вглиб.
12. Простий зв'язний граф задано списками суміжності. Побудувати каркас цього графа з використанням пошуку вглиб.
13. Задано матрицю суміжності простого зв'язного графа. Побудувати каркас цього графа з використанням пошуку вшир.
14. Простий зв'язний граф задано списками суміжності. Побудувати каркас цього графа з використанням пошуку вшир.
15. Задано скінченну множину натуральних чисел і натуральне число n. Використовуючи бектрекінг, знайти таку підмножину цих чисел, щоб їх сума дорівнювала n.
16. Задано матрицю суміжності простого зв'язного графа. Побудувати гамільтонів цикл або виявити, що даний граф не має такого циклу. Використати бектрекінг.
17. Простий зв'язний граф задано списками суміжності. Побудувати гамільтонів цикл або виявити, що даний граф не має такого циклу. Використати бектрекінг.
18. Задано матрицю суміжності простого зв'язного графа. Побудувати гамільтонів шлях або виявити, що даний граф не має його. Використати бектрекінг.
19. Простий зв'язний граф задано списками суміжності. Побудувати гамільтонів шлях або виявити, що даний граф не має його. Використати бектрекінг.
20. Задано матрицю суміжності простого зв'язного графа та натуральне число n. Використовуючи бектрекінг, розфарбувати цей граф у n кольорів або переконатись, що це неможливо.
21. Задано списки суміжності простого зв'язного графа та натуральне число n. Використовуючи бектрекінг, розфарбувати цей граф у n кольорів або переконатись, що це неможливо.

22. Задано натуральне число n. Використовуючи бектрекінг, розв'язати задачу про n ферзів.
23. Задано шахівницю $n \times n$. Кінь, який ходить за шаховими правилами, перебуває в клітинці з початковими координатами (i_0, j_0), де i_0 – номер рядка, j_0 – номер стовпця. Визначити, чи можна покрити всю шахівницю за $n^2 - 1$ ходів коня так, щоб кожну клітинку було відвідано точно один раз. Використати бектрекінг.
24. Задано матрицю суміжності простого зв'язного графа. Знайти всі максимальні незалежні множини вершин цього графа. Використати бектрекінг.
25. Простий зв'язний граф задано списками суміжності. Знайти всі максимальні незалежні множини вершин цього графа. Використати бектрекінг.
26. Задано матрицю суміжності простого зв'язного графа. Знайти всі максимальні кліки. Використати бектрекінг.
27. Простий зв'язний граф задано списками суміжності. Знайти всі максимальні кліки. Використати бектрекінг.
28. Зважений простий зв'язний граф задано матрицею ваг. Використовуючи алгоритм Краскала, побудувати мінімальний каркас цього графа.
29. Задано список ребер та їх ваг для зваженого простого зв'язного графа. За допомогою алгоритму Краскала побудувати мінімальний каркас цього графа.

Розділ 5
Відношення

- Відношення та їх властивості
- Відношення еквівалентності
- Відношення часткового порядку
- Топологічне сортування
- Операції над відношеннями
- Замикання відношень
- Бази даних і відношення

Відношення – одне з основних понять сучасної математики. Мову відношень використовують для опису зв'язків між об'єктами та поняттями. Зокрема, поняття бінарного відношення дає змогу формалізувати операції попарного порівняння, і тому його широко використовують у теорії вибору, а реляційні бази даних ґрунтуються на концепції n-арних відношень.

5.1. Відношення та їх властивості

Найпростіший спосіб задати зв'язок між елементами двох множин – записати впорядковані пари елементів, що перебувають у цьому зв'язку. Нехай A та B – множини. *Бінарне відношення з A в B* – це якась підмножина R декартового добутку $A \times B$ цих множин: $R \subset A \times B$. Інакше кажучи, бінарне відношення з A в B – це множина впорядкованих пар, у якій перший елемент пари належить множині A, а другий – множині B. Використовують запис aRb, якщо $(a, b) \in R$, і запис $a\overline{R}b$, якщо $(a, b) \notin R$.

Бінарні відношення описують зв'язки між елементами двох множин. Зв'язки між елементами більше ніж двох множин задають n-арними відношеннями (див. підрозділ 5.7). Розглядаючи в певному контексті лише бінарні відношення, уживають термін *„відношення"* замість „бінарне відношення".

Приклад 5.1. Нехай $A = \{0, 1, 2\}$, $B = \{a, b\}$ та задано відношення $R = \{(0, a), (0, b), (1, a), (2, b)\}$. Отже, $0Ra$, бо $(0, a) \in R$, але $1\overline{R}b$, оскільки $(1, b) \notin R$.

Здебільшого розглядають бінарні відношення за умови $A = B$. *Відношенням на множині* A називають бінарне відношення з A в A. Інакше кажучи, відношення R на множині A — це підмножина декартового квадрату множини A, тобто $R \subset A^2$.

Приклад 5.2. Нехай $A = \{1, 2, 3, 4\}$. Які впорядковані пари утворюють відношення $R = \{(a, b) \mid a$ ділить $b\}$? Очевидно, що $R = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 3), (4, 4)\}$.

Бінарне відношення на множині A можна задати за допомогою булевої матриці чи орієнтованого графа. На n-елементній множині A відношення R задає $n \times n$ матриця $M_R = [m_{ij}]$, $i, j = 1, \ldots, n$, де

$$m_{ij} = \begin{cases} 1, \text{ якщо } (a_i, a_j) \in R, \\ 0, \text{ якщо } (a_i, a_j) \notin R. \end{cases}$$

Граф G_R, який задає відношення R на множині A, будують так. Вершини графа позначають елементами цієї множини, а дуга (a_i, a_j) існує тоді й лише тоді, коли $(a_i, a_j) \in R$. Такий граф G_R називають *графом, асоційованим із відношенням* R, або просто *графом відношення* R.

Приклад 5.3. На рис. 5.1 зображено матрицю та граф, які задають відношення з прикладу 5.2.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

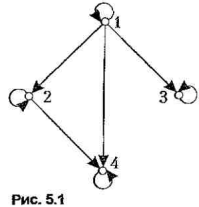


Рис. 5.1

Розглянемо властивості відношень на множині A. Відношення R на множині A називають *рефлексивним*, якщо для будь-якого $a \in A$ виконується $(a, a) \in R$.

Приклад 5.4. Розглянемо шість відношень на множині $A = \{1, 2, 3, 4\}$:

$$R_1 = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 4), (4, 1), (4, 4)\};$$
$$R_2 = \{(1, 1), (1, 2), (2, 1)\};$$
$$R_3 = \{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (3, 3), (4, 1), (4, 4)\};$$
$$R_4 = \{(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\};$$
$$R_5 = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\};$$
$$R_6 = \{(3, 4)\}.$$

Відношення R_3 та R_5 рефлексивні, бо вони містять усі пари вигляду (a, a), тобто $(1, 1)$, $(2, 2)$, $(3, 3)$, $(4, 4)$. Решта відношень не рефлексивні. Зокрема, відношення R_1, R_2, R_4, R_6 не містять пари $(3, 3)$.

Відношення R на множині A називають *іррефлексивним*, якщо для будь-якого $a \in A$ виконується $(a, a) \notin R$. Наприклад, відношення R_4, R_6 із прикладу 5.4 іррефлексивні, а R_1, R_2 — не рефлексивні й не іррефлексивні.

Відношення R на множині A називають *симетричним*, якщо для будь-яких $a, b \in A$ з того, що $(a, b) \in R$, випливає, що $(b, a) \in R$. У прикладі 5.4 лише відношення R_2 та R_3 симетричні.

Відношення R на множині A називають *антисиметричним*, якщо для всіх $a, b \in A$ з того, що $(a, b) \in R$ і $(b, a) \in R$, випливає, що $a = b$. Інакше кажучи, відношення антисиметричне, якщо в разі $a \neq b$ воно водночас не містить пар (a, b) та (b, a). У прикладі 5.4 антисиметричні лише відношення $R_4 - R_6$. У кожному з них немає таких пар елементів a та b $(a \neq b)$, що водночас $(a, b) \in R$ і $(b, a) \in R$. Властивості симетричності й антисиметричності не антагоністичні: існують відношення, які мають обидві ці властивості. Наприклад, відношення $R = \varnothing$ на множині $A = \{a\}$ водночас і симетричне, й антисиметричне. Є також відношення, які не мають жодної з цих двох властивостей, наприклад R_1 із прикладу 5.4.

Відношення R на множині A називають *асиметричним*, якщо для всіх $a, b \in A$ з того, що $(a, b) \in R$, випливає, що $(b, a) \notin R$. Зрозуміло, що будь-яке асиметричне відношення має бути й антисиметричним. Обернене твердження неправильне. Відношення R_5 із прикладу 5.4 антисиметричне, проте не асиметричне, бо містить пари (1, 1), (2, 2), (3, 3), (4, 4).

Відношення R на множині A називають *транзитивним*, якщо для будь-яких $a, b, c \in A$ з того, що $(a, b) \in R$ і $(b, c) \in R$, випливає $(a, c) \in R$. Відношення $R_4 - R_6$ із прикладу 5.4 транзитивні, відношення $R_1 - R_3$ не транзитивні: $(3, 4) \in R_1$, $(4, 1) \in R_1$, але $(3, 1) \notin R_1$; $(2, 1) \in R_2$, $(1, 2) \in R_2$, але $(2, 2) \notin R_2$; $(2, 1) \in R_3$, $(1, 4) \in R_3$, але $(2, 4) \notin R_3$.

Розглянемо, як деякі властивості відношень відображаються в їх матрицях і графах. Якщо відношення R рефлексивне, то на головній діагоналі матриці M_R лише одиниці, якщо іррефлексивне — то нулі. Матриця симетричного відношення симетрична, а матриця M_R антисиметричного відношення R має таку властивість: якщо $i \neq j$, то з $m_{ij} = 1$ випливає $m_{ji} = 0$ (але може бути $m_{ij} = m_{ji} = 0$) (рис. 5.2).

Граф G_R рефлексивного відношення R має петлю в кожній вершині. У графі транзитивного відношення в разі наявності пари дуг (a, b) та (b, c) обов'язково є дуга (a, c) (рис. 5.3).

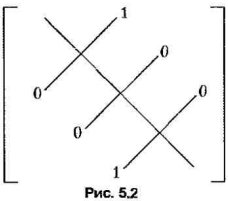


Рис. 5.2

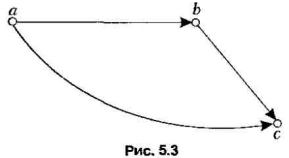


Рис. 5.3

5.2. Відношення еквівалентності

Розглянемо відношення, які мають водночас декілька зазначених вище властивостей у певній комбінації. Відношення на множині A називають *відношенням еквівалентності*, якщо воно рефлексивне, симетричне й транзитивне. Два елементи множини A, пов'язані відношенням еквівалентності, називають *еквівалентними*. Оскільки відношення еквівалентності за означенням рефлексивне, то кожний його елемент еквівалентний до самого себе. Більше того, позаяк відношення еквівалентності за означенням транзитивне, то з того, що елементи a та b еквівалентні й b та c еквівалентні, випливає, що a та c також еквівалентні.

Приклад 5.5. Нехай R — таке відношення на множині цілих чисел: aRb тоді й тільки тоді, коли $(a=b) \vee (a=-b)$. Воно рефлексивне, симетричне й транзитивне, тому являє собою відношення еквівалентності.

Приклад 5.6. Нехай R — таке відношення на множині дійсних чисел: aRb тоді й лише тоді, коли $(a-b)$ — ціле число. Оскільки число $a-a=0$ ціле для всіх дійсних чисел a, то aRa для всіх дійсних чисел a. Отже, відношення R рефлексивне. Нехай тепер aRb. Звідси випливає, що $a-b$ — ціле число. Тому й число $b-a$ також ціле, тобто bRa. Ми довели, що відношення R симетричне. Якщо aRb та bRc, то числа $a-b$ та $b-c$ цілі. Але тоді число $a-c=(a-b)+(b-c)$ також ціле, тобто aRc. Отже, відношення R транзитивне. Тому це відношення еквівалентності.

Приклад 5.7. Конгруентність за модулем m. Нехай $m>1$ — ціле число. Доведемо, що $R=\{(a,b) \mid a \equiv b (\mathrm{mod}\ m)\}$ — відношення еквівалентності на множині Z цілих чисел. За означенням, $a \equiv b (\mathrm{mod}\ m)$ еквівалентно тому, що m ділить $(a-b)$. Зазначимо, що $a-a=0$ ділиться на m, бо $0=0 \cdot m$. Отже, $a \equiv a (\mathrm{mod}\ m)$, тобто відношення рефлексивне. Окрім того, $a \equiv b (\mathrm{mod}\ m)$, якщо $a-b=km$, де k — ціле число. Тоді $b-a=(-k)m$, тобто $b \equiv a (\mathrm{mod}\ m)$, і відношення симетричне. Нарешті, нехай $a \equiv b (\mathrm{mod}\ m)$ і $b \equiv c (\mathrm{mod}\ m)$. Це означає, що $a-b=km$, $b-c=lm$, де k, l — цілі числа. Додамо останні дві рівності: $a-b+b-c=(k+l)m$, тобто $a-c=(k+l)m$. Отже, $a \equiv c (\mathrm{mod}\ m)$, тобто відношення транзитивне. Отже, конгруентність за модулем m — відношення еквівалентності на множині цілих чисел.

Нехай R — відношення еквівалентності на множині A. Множину всіх елементів, які еквівалентні до елемента $a \in A$, називають *класом еквівалентності* (елемента a). Клас еквівалентності, породжений елементом a за відношенням R, позначають $[a]_R$. Маючи на увазі якесь певне відношення еквівалентності, використовують позначення $[a]$. Отже, $[a]_R=\{x \in A \mid aRx\}$. Елемент $b \in [a]_R$ називають *представником* цього класу еквівалентності.

Приклад 5.8. Знайдемо класи еквівалентності відношення з прикладу 5.5. Оскільки ціле число еквівалентне до самого до себе та до протилежного числа, то класи еквівалентності за цим відношенням такі: $[a]=\{-a, a\}$, $a \neq 0$, та $[0]=\{0\}$.

Приклад 5.9. Знайдемо класи еквівалентності елементів 0 і 1 для відношення конгруентності за mod 4 (див. приклад 5.7). Клас еквівалентності елемента 0 містить усі цілі числа b такі, що $0 \equiv b (\mathrm{mod}\ 4)$, тобто ті, що діляться на 4. Отже, $[0]=\{..., -8, -4, 0, 4, 8, ...\}$. Клас еквівалентності елемента 1 містить усі цілі числа b такі, що $1 \equiv b (\mathrm{mod}\ 4)$. Звідси випливає, що $[1]=\{..., -7, -3, 1, 5, 9, ...\}$. Класи еквівалентності, подібні до розглянутих у цьому прикладі, називають *класами конгруентності за модулем m* і позначають $[a]_m$. Отже, $[0]_4=\{..., -8, -4, 0, 4, 8, ...\}$, $[1]_4=\{..., -7, -3, 1, 5, 9, ...\}$.

Нехай R — відношення еквівалентності на множині A. Зазначимо, що класи еквівалентності, породжені двома елементами множини A, або збігаються, або не перетинаються. Про це твердить наступна лема.

ЛЕМА 5.1. Нехай R — відношення еквівалентності на множині A. Тоді такі твердження еквівалентні:

(I) aRb,

(II) $[a] = [b]$,

(III) $[a] \cap [b] \neq \varnothing$.

Доведення. Спочатку доведемо, що з (I) випливає (II). Припустимо, що aRb. Щоб довести рівність $[a] = [b]$, покажемо, що $[a] \subset [b]$ та $[b] \subset [a]$. Нехай $c \in [a]$; тоді aRc. Оскільки aRb, а R — симетричне відношення, то bRa. Позаяк відношення R транзитивне, то з bRa й aRc випливає bRc, тому $c \in [b]$. Отже, $[a] \subset [b]$. Аналогічно можна довести, що $[b] \subset [a]$.

Доведемо тепер, що з (II) випливає (III). Справді, $[a] \neq \varnothing$, бо $a \in [a]$ внаслідок рефлексивності. Отже, з $[a] = [b]$ випливає $[a] \cap [b] \neq \varnothing$.

Нарешті, доведемо, що з (III) випливає (I). Припустимо, що $[a] \cap [b] \neq \varnothing$. Тоді існує такий елемент c, що $c \in [a]$ та $c \in [b]$, тобто aRc та bRc. Із симетричності відношення R випливає cRb. Оскільки відношення R транзитивне, то з aRc та cRb випливає aRb.

Позаяк з (I) випливає (II), з (II) випливає (III) та з (III) випливає (I), то твердження (I), (II), (III) еквівалентні.

Відношення еквівалентності R, задане на множині A, тісно пов'язане з розбиттям множини. Цей зв'язок виражено у двох наступних теоремах. Нагадаємо, що систему S підмножин множини A називають її розбиттям, якщо всі множини системи S непорожні, попарно не перетинаються й об'єднання їх усіх дорівнює множині A (див. підрозділ 1.13).

ТЕОРЕМА 5.1. Кожне відношення еквівалентності R на множині A породжує її розбиття на класи еквівалентності.

Доведення. Об'єднання класів еквівалентності за відношенням R покриває множину A, бо кожен елемент a з множини A належить своєму власному класу еквівалентності $[a]_R$. Інакше кажучи,

$$\bigcup_{a \in A} [a]_R = A.$$

Із леми 5.1 випливає, що коли $[a]_R \neq [b]_R$, то $[a]_R \cap [b]_R = \varnothing$.

Приклад 5.10. Відношення конгруентності за mod 4 (див. приклад 5.9) породжує розбиття множини Z цілих чисел на чотири класи еквівалентності: $[0]_4$, $[1]_4$, $[2]_4$ та $[3]_4$. Вони попарно не перетинаються, а їх об'єднання дорівнює множині Z.

ТЕОРЕМА 5.2. Будь-яке розбиття множини A задає відношення еквівалентності на множині A.

Доведення. Нехай $a, b \in A$; будемо вважати, що aRb тоді й лише тоді, коли a та b належать одній і тій самій множині розбиття. Залишилося довести, що одержане відношення на множині A являє собою відношення еквівалентності. Для цього потрібно переконатись, що воно рефлексивне, симетричне й транзитивне. Справді, оскільки елемент a належить якійсь множині розбиття, то aRa, тобто відношення рефлексивне. Нехай A_i — якась множина розбиття й $a, b \in A_i$. Тоді й $b, a \in A_i$, тобто з aRb випливає bRa. Симетричність доведено. Нарешті, з aRb та bRc випливає $a, b, c \in A_i$. Тому aRc, тобто відношення R транзитивне.

5.3. Відношення часткового порядку

Відношення R на множині A називають *відношенням часткового порядку* (або *частковим порядком*), якщо воно рефлексивне, антисиметричне й транзитивне. Множину A з частковим порядком R називають *частково впорядкованою* й позначають (A, R).

Приклад 5.11. Нехай $A = \{1, 2, 3, 4, 6, 8, 12\}$. Відношення R задамо як звичайне порівняння чисел: $(a, b) \in R$ тоді й лише тоді, коли $a \leqslant b$ $(a, b \in A)$. Неважко безпосередньо переконатись, що це частковий порядок на множині A.

Приклад 5.12. Нехай A — множина з прикладу 5.11. Відношення R_1 задамо так: $(a, b) \in R_1$ тоді й лише тоді, коли a ділить b. Отже, $R_1 = \{(1, 1), (2, 2), (3, 3), (4, 4), (6, 6), (8, 8), (12, 12), (1, 2), (1, 3), (1, 4), (1, 6), (1, 8), (1, 12), (2, 4), (2, 6), (2, 8), (2, 12), (3, 6), (3, 12), (4, 8), (4, 12), (6, 12)\}$. Легко переконатись, що це відношення рефлексивне, антисиметричне й транзитивне, тому являє собою відношення часткового порядку на множині A.

Два елементи a та b частково впорядкованої множини (A, R) називають *порівнянними*, якщо aRb чи bRa. Якщо a та b — такі елементи, що ні aRb, ні bRa, то їх називають *непорівнянними*.

Приклад 5.13. Елементи 3 та 4 множини (A, R_1) із прикладу 5.12 непорівнянні.

Якщо (A, R) — частково впорядкована множина, у якій будь-які два елементи порівнянні, то її називають *лінійно*, або *тотально впорядкованою*, а частковий порядок R — *лінійним* (*тотальним*) порядком. Отже, множина (A, R) із прикладу 5.11 лінійно впорядкована, множина (A, R_1) із прикладу 5.12 частково впорядкована, але не лінійно впорядкована. Лінійно впорядковану множину називають також *ланцюгом*.

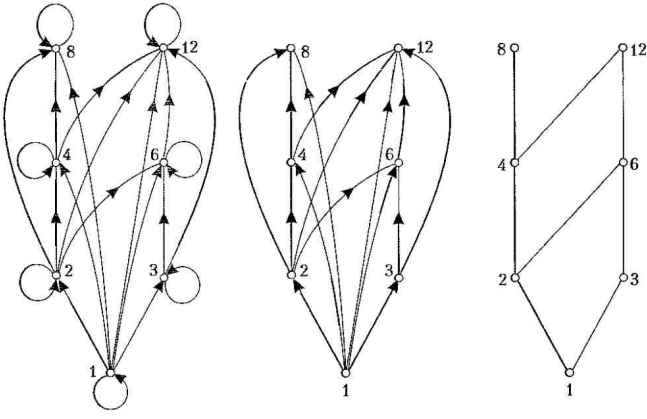
Приклад 5.14. Нехай $A = E_2^n$ — множина всіх векторів довжиною n з булевими компонентами 0, 1. Задамо частковий порядок на цій множині так: $(a_1, a_2, ..., a_n) \leqslant (b_1, b_2, ..., b_n)$ тоді й лише тоді, коли $a_i \leqslant b_i$ $(i = 1, ..., n)$. Цей частковий порядок не лінійний. Наприклад, не можна порівняти вектори (010000) і (101000).

Нехай на скінченній множині A задано якесь відношення часткового порядку R, а G_R — граф, асоційований із цим відношенням. Відношення R можна задати

за допомогою такої процедури. Починають із графа G_R. Оскільки відношення часткового порядку рефлексивне, то в кожній вершині графа G_R є петля. Потрібно вилучити всі ці петлі. Потім вилучають усі дуги графа G_R, які є в ньому внаслідок транзитивності. Наприклад, якщо пари (a, b) та (b, c) належать відношенню, то в графі вилучають дугу (a, c). Більше того, якщо пари (c, d) також належить відношенню R, то в графі GR вилучають дугу (a, d). Після цього всі вершини графа розміщують на площині так, щоб початкова вершина кожної дуги була нижче, ніж кінцева вершина. Тепер усувають усі стрілки, бо дуги спрямовано вгору, до кінцевих вершин.

Усі ці кроки задано коректно, і в разі скінченної множини A кількість таких кроків скінченна. Отримаємо *діаграму Гассе* (H. Hasse), яка містить усю інформацію, потрібну для визначення відношення часткового порядку.

Приклад 5.15. Зобразимо діаграму Гассе для відношення часткового порядку з прикладу 5.12. Почнемо з орієнтованого графа, асоційованого з відношенням R_1 (рис. 5.4, *а*). Вилучимо всі петлі (рис. 5.4, *б*), а потім — усі дуги, зумовлені властивістю транзитивності; це дуги (1, 4), (1, 6), (1, 8), (1, 12), (2, 8), (2, 12), (3, 12). Орієнтуємо всі дуги в напрямку знизу вверх і усуємо стрілки. Отримаємо діаграму Гассе (рис. 5.4, *в*).



а *б* *в*

Рис. 5.4

Символом $\leqslant$ часто позначають довільне відношення часткового порядку. У частково впорядкованій множині (A, R) запис $a \leqslant b$ означає, що $(a, b) \in R$, а запис $a < b$ — що $a \leqslant b$, але $a \neq b$. Якщо $a < b$, то говорять, що елемент *a передує елементу b* (a менше, ніж b), або елемент b *виходить* з елемента a (b більше, ніж a).

Елемент $b \in A$ *безпосередньо виходить* з $a \in A$ тоді й лише тоді, коли $a < b$ та не існує такого елемента $u \in A$, що $a < u < b$. У такому разі елемент *a безпосередньо передує* елементу *b*.

Є алгоритм, який дає змогу побудувати діаграму Гассе для частково впорядкованої множини без використання графа відношення. Цей алгоритм ґрунтується на такій властивості. Нехай (A, R) — скінченна частково впорядкована множина; тоді $a_1 < a_n$ у тому й лише тому разі, якщо існує послідовність $a_1 < a_2 < ... < a_n$, у якій a_{i+1} безпосередньо виходить з a_i для $i = 1, 2, ..., n - 1$. Зобразимо кожний елемент $a_i \in A$ точкою a_i на площині й розглянемо всі впорядковані пари (a_i, a_j). Точку a_j розмістимо вище точки a_i тоді й лише тоді, коли $a_i < a_j$, і з'єднаємо точки a_i й a_j лінією, якщо a_j безпосередньо виходить з a_i. Отримаємо діаграму Гассе; у ній існує шлях, який веде від точки a_n до точки a_m, якщо $a_n < a_m$.

Елементи частково впорядкованих множин, які мають певні екстремальні властивості, дуже важливі в багатьох застосуваннях. Елемент частково впорядкованої множини називають *максимальним*, якщо він не менший за будь-який елемент цієї множини. Отже, a — максимальний елемент частково впорядкованої множини (A, R), якщо не існує такого елемента $b \in A$, що $a < b$. Аналогічно, елемент називають *мінімальним*, якщо він не більший за будь-який елемент частково впорядкованої множини. Отже, елемент a мінімальний, якщо не існує такого елемента $b \in A$, що $b < a$. Максимальні та мінімальні елементи легко визначити на діаграмі Гассе: це відповідно „верхні" й „нижні" її елементи (для „верхніх" елементів немає висхідних ребер, а для „нижніх" — низхідних).

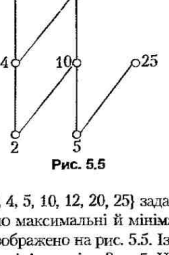


Рис. 5.5

Приклад 5. 16. На множині $A = \{2, 4, 5, 10, 12, 20, 25\}$ задано відношення часткового порядку $R = \{(a, b) \mid a \text{ ділить } b\}$. Знайдемо максимальні й мінімальні елементи множини (A, R). Діаграму Гассе для цієї множини зображено на рис. 5.5. Із неї доходимо висновку, що максимальні елементи — 12, 20 і 25, а мінімальні — 2 та 5. Цей приклад свідчить, що частково впорядкована множина може мати більше одного максимального чи мінімального елемента.

5.4. Топологічне сортування

Хороший приклад використання частково впорядкованих множин — процес топологічного сортування. Це сортування елементів, для яких означено відношення часткового порядку, тобто порядок задано не для всіх, а лише для деяких пар.

Із цілком зрозумілих міркувань будемо вважати, що частково впорядкована множина, яка підлягає топологічному сортуванню, скінченна. Частковий порядок можна подати у вигляді діаграми Гассе, як це показано в підрозділі 5.3. Мета топологічного сортування — перетворити частковий порядок на лінійний.

Почнемо з означення. Лінійний порядок $\leqslant$ називають *сумісним* із частковим порядком R, якщо з aRb випливає $a \leqslant b$. Побудову лінійного порядку, сумісного із заданим частковим порядком, називають *топологічним сортуванням*.

Наведемо два приклади застосування топологічного сортування.

1. Певна задача (наприклад, технічний проект) розпадається на низку підзадач. Виконання деяких підзадач можливе лише після завершення інших. Якщо підзадачу v потрібно виконати до підзадачі w, то будемо писати $v < w$. Топологічне сортування — це такий розподіл робіт, за якого кожна з підзадач не розпочнеться до завершення всіх підзадач, які потрібно виконати до неї.

2. В університетських програмах деякі курси потрібно читати раніше за інші, бо останні ґрунтуються на попередньо викладеному матеріалі. Якщо для курсу w потрібно спочатку ознайомитись із курсом v, то пишемо $v < w$. Тут топологічне сортування означає, що жоден курс не можна читати раніше, ніж ті, що його підтримують.

ТЕОРЕМА 5.3. Кожна скінченна непорожня частково впорядкована множина (A, R) має принаймні один мінімальний елемент.

Доведення. Виберемо з множини A елемент a_0. Якщо він не мінімальний, то існує такий елемент a_1, що $a_1 < a_0$. Якщо елемент a_1 не мінімальний, то існує такий елемент a_2, що $a_2 < a_1$. Продовжимо цей процес: якщо елемент a_k не мінімальний, то існує такий елемент a_{k+1}, що $a_{k+1} < a_k$. Оскільки множина скінченна, то описаний процес закінчиться на мінімальному елементі a_m.

Опишемо роботу алгоритму топологічного сортування для довільної непорожньої частково впорядкованої множини (A, R). Спочатку виберемо мінімальний елемент a_1 (він існує за теоремою 5.3). Множина $(A\backslash\{a_1\}, R)$ також частково впорядкована (обґрунтування цього пропонуємо як вправу). Якщо вона непорожня, то виберемо з неї мінімальний елемент a_2. Вилучимо a_2; якщо ще залишились елементи, то виберемо мінімальний елемент a_3 з $A\backslash\{a_1, a_2\}$. Продовжимо процес вибору мінімального елемента a_{k+1} з $A\backslash\{a_1, a_2, ..., a_k\}$, доки залишаються елементи.

Позаяк A — скінченна множина, то цей процес завершиться. Отримаємо послідовність елементів $a_1, a_2, ..., a_n$. Шуканий лінійний порядок означають так: $a_1 \leqslant a_2 \leqslant ... \leqslant a_n$. Він сумісний із заданим частковим порядком. Щоб переконатись у цьому, зазначимо, що коли $b < c$ в заданому частковому впорядкуванні, то c вибирають як мінімальний елемент, коли елемент b вже вилучено, бо інакше c не був би мінімальним елементом. Якщо є декілька мінімальних елементів, вибираємо будь-який. Опишемо цей алгоритм.

Алгоритм топологічного сортування

Наведемо кроки алгоритму.

Крок 1. Виконати $k := 1$.

Крок 2. Виконати $a_k :=$ мінімальний елемент множини A.

Крок 3. Виконати $A := A \setminus \{a_k\}$.

Крок 4. Виконати $k := k + 1$.

Крок 5. Якщо $A = \varnothing$, то зупинитись ($a_1, a_2, ..., a_n$ – результат топологічного сортування множини A), інакше перейти до кроку 2.

Приклад 5.17. Згідно з проектом у комп'ютерній компанії потрібно виконати сім завдань. Деякі з них можна розпочати лише після завершення певних інших завдань. Частковий порядок на множині завдань задамо так: $X < Y$, якщо завдання Y не можна розпочати до завершення завдання X. Діаграму Гассе для множини цих завдань подано на рис. 5.6. Потрібно знайти порядок, у якому ці завдання можна виконати для завершення всього проекту.

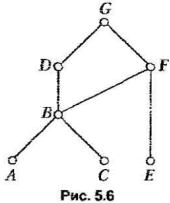


Рис. 5.6

Очевидно, що порядок виконання цих завдань можна дістати, виконавши топологічне сортування множини всіх завдань. Послідовність кроків такого сортування наведено на рис. 5.7. Результат цього сортування $A < C < B < E < F < D < G$ задає можливий порядок виконання завдань.

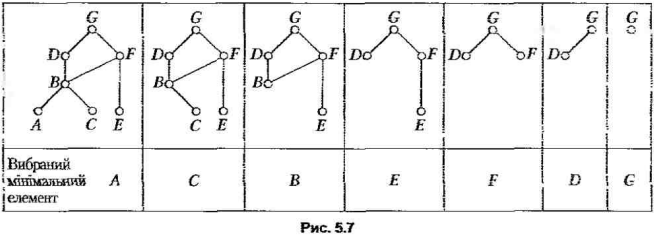


Рис. 5.7

5.5. Операції над відношеннями

Оскільки відношення з множини A в множину B — підмножина декартового добутку множин $A \times B$, то над будь-якими двома відношеннями з A в B можна виконувати звичайні теоретико-множинні операції.

Приклад 5.18. Нехай $A = \{1, 2, 3\}$ та $B = \{1, 2, 3, 4\}$. Означимо відношення R_1 і R_2 з A в B:

$$R_1 = \{(1, 1), (2, 2), (3, 3)\}, R_2 = \{(1, 1), (1, 2), (1, 3), (1, 4)\}.$$

Тоді

$$R_1 \cup R_2 = \{(1. 1). (1, 2), (1, 3), (1, 4), (2. 2), (3, 3)\}.$$
$$R_1 \cap R_2 = \{(1. 1)\},$$
$$R_1 \backslash R_2 = \{(2, 2), (3, 3)\},$$
$$R_2 \backslash R_1 = \{(1, 2), (1, 3), (1, 4)\}.$$

Розглянемо тепер іншу операцію — композицію відношень. Нехай R — відношення з множини A в множину B, а S — відношення з множини B в множину C. *Композицією* відношень R і S називають відношення, яке складається з усіх можливих упорядкованих пар (a, c), де $a \in A$, $c \in C$, для яких існує такий елемент $b \in B$, що $(a, b) \in R$ і $(b, c) \in S$. Композицію відношень R і S позначають як $S \circ R$.

Приклад 5.19. Знайдемо композицію відношень R і S, де R — відношення з множини $A = \{1, 2, 3\}$ в множину $B = \{1, 2, 3, 4\}$:

$$R = \{(1, 1), (1, 4), (2, 3), (3, 1), (3, 4)\};$$

S — відношення з множини B в множину $C = \{0, 1, 2\}$:

$$S = \{(1, 0), (2, 0), (3, 1), (3, 2), (4, 1)\}.$$

Композицію $S \circ R$ будують, використовуючи всі впорядковані пари з відношень R і S такі, що другий елемент пари з R збігається з першим елементом пари з S. Наприклад, пари $(2, 3) \in R$ та $(3, 1) \in S$ породжують пару $(2, 1) \in S \circ R$. Виконавши описані дії, отримаємо

$$S \circ R = \{(1, 0), (1, 1), (2, 1), (2, 2), (3, 0), (3, 1)\}.$$

Нехай R — відношення на множині A. Степінь R^n, $n = 1, 2, 3, \ldots$, означають за допомогою рекурсії:

$$R^1 = R,$$
$$R^{n+1} = R^n \circ R$$

Отже, зокрема,

$$R^2 = R \circ R,$$
$$R^3 = R^2 \circ R = (R \circ R) \circ R.$$

Приклад 5.20. Нехай на множині $A = \{1, 2, 3, 4\}$ задано відношення $R = \{(1, 1), (2, 1), (3, 2), (4, 3)\}$. Знайдемо R^n, $n = 2, 3, 4, 5$. За означенням послідовно одержимо

$$R^2 = R \circ R = \{(1, 1), (2, 1), (3, 1), (4, 2)\},$$
$$R^3 = R^2 \circ R = \{(1, 1), (2, 1), (3, 1), (4, 1)\},$$
$$R^4 = R^3 \circ R = \{(1, 1), (2, 1), (3, 1), (4, 1)\},$$

тобто $R^4 = R^3$. Можна переконатись, що $R^5 = R^4$.

ТЕОРЕМА 5.4. Нехай R — транзитивне відношення на множині A. Тоді $R^n \subset R$, $n = 1, 2, 3, \ldots$.

Доведення. Застосуємо математичну індукцію. У разі $n = 1$ твердження теореми тривіальне. Гіпотеза індукції: $R^n \subset R$. Для завершення доведення потрібно переконатися, що з цієї гіпотези випливає включення $R^{n+1} \subset R$. Припустимо, що $(a, b) \in R^{n+1}$. Оскільки $R^{n+1} = R^n \circ R$, то існує такий елемент $x \in A$, що $(a, x) \in R$ та $(x, b) \in R^n$. За індуктивною гіпотезою $R^n \subset R$, звідки випливає, що $(x, b) \in R$. Позаяк відношення R транзитивне, то з $(a, x) \in R$ та $(x, b) \in R$ випливає, що $(a, b) \in R$. Отже, $R^{n+1} \subset R$.

Уведемо операції над булевими матрицями (тобто матрицями з елементами 0 і 1).

Диз'юнкція булевих $m \times n$ матриць P та Q — це $m \times n$ матриця $Z = P \vee Q$ з елементами $z_{ij} = p_{ij} \vee q_{ij}$, де $i = 1, \ldots, m$, $j = 1, \ldots, n$.

Кон'юнкція булевих $m \times n$ матриць P та Q — це $m \times n$ матриця $Z = P \wedge Q$ з елементами $z_{ij} = p_{ij} \wedge q_{ij}$, де $i = 1, \ldots, m$, $j = 1, \ldots, n$.

Нехай P — $m \times k$ матриця, Q — $k \times n$ матриця. Тоді *булевий добуток* матриць P та Q — це $m \times n$ матриця $Z = P \odot Q$ з елементами

$$z_{ij} = (p_{i1} \wedge q_{1j}) \vee (p_{i2} \wedge q_{2j}) \vee \ldots \vee (p_{ik} \wedge q_{kj}),$$

або, коротше,

$$z_{ij} = \bigvee_{r=1}^{k} (p_{ir} \wedge q_{rj}),$$

де

$$i = 1, \ldots, m,$$
$$j = 1, \ldots, n.$$

Булевий степінь булевих $n \times n$ матриць $A^{[r]}$, $r \in N$, означають так:

$$A^{[r]} = \underbrace{A \odot A \odot \ldots \odot A}_{r \text{ разів}}.$$

Це означення коректне оскільки булевий добуток матриць асоціативний. За означенням $A^{[0]} = I_n$, де I_n — одинична $n \times n$ матриця.

Операції над відношеннями легко виразити через матриці, які задають ці відношення:

$$M_{R_1 \cup R_2} = M_{R_1} \vee M_{R_2},$$
$$M_{R_1 \setminus R_2} = M_{R_1} \wedge M_{R_2},$$
$$M_{S \circ R} = M_R \odot M_S,$$
$$M_{R^k} = M_R^{[k]}.$$

5.6. Замикання відношень

Нехай R — відношення на множині A. Воно може не мати деяких властивостей. Наприклад, це відношення може не бути рефлексивним, симетричним або транзитивним.

Замиканням відношення R за властивістю P називають найменше відношення $C \supset R$, яке має властивість P. Термін *„найменше відношення"* означає, що C — підмножина будь-якого відношення S, для якого виконуються такі умови.

1. S має властивість P.
2. $R \subset S$.

Можна довести, що замикання відношення R за властивістю P, якщо воно існує, являє собою перетин усіх відношень із властивістю P, які містять R.

Як властивість P можна брати, зокрема, рефлексивність, симетричність і транзитивність. Отже, розглянемо відповідні замикання відношення R.

Приклад 5.21. Відношення $R=\{(1, 1), (1, 2), (2, 1), (3, 2)\}$ на множині $A=\{1, 2, 3\}$ не рефлексивне. Щоб отримати його рефлексивне замикання, додамо пари (2, 2) та (3, 3), бо лише цих пар вигляду (a, a) не має в R. Очевидно, що це нове відношення рефлексивне, і R — його підмножина. Більше того, воно являє собою підмножину будь-якого рефлексивного відношення S, для якого $R \subset S$. Отже, ми справді одержали рефлексивне замикання відношення R.

З останнього прикладу зрозумілий спосіб побудови рефлексивного замикання: достатньо додати до відношення R усі ті пари (a, a), де $a \in A$, яких немає в R. Отже, рефлексивне замикання R дорівнює $R \cup \Delta$, де $\Delta=\{(a, a) \mid a \in A\}$. Відношення Δ на множині A називають *діагональним*. Що стосується орієнтованого графа, асоційованого з відношенням R, то рефлексивному замиканню відповідає граф, одержаний із нього додаванням петель у тих вершинах, де їх немає.

Приклад 5.22. Відношення $R=\{(1, 1), (1, 2), (2, 2), (2, 3), (3, 1), (3, 2)\}$ на множині $A=\{1, 2, 3\}$ не симетричне. Для отримання його симетричного замикання додамо пари (2, 1) і (1, 3), бо лише цих пар вигляду (b, a), для яких $(a, b) \in R$, немає в R. Очевидно, це нове відношення симетричне, й R — його підмножина. Окрім того, воно являє собою підмножину будь-якого симетричного відношення S, для якого $R \subset S$. Отже, ми справді одержали симетричне замикання відношення R.

Наведені міркування мають загальний характер: для отримання симетричного замикання потрібно додати всі такі пари (b, a), що $(b, a) \notin R$, але $(a, b) \in R$. Отже, симетричне замикання відношення R дорівнює $R \cup R^{-1}$, де $R^{-1}=\{(b, a) \mid (a, b) \in R\}$. Це можна сформулювати й у термінах графа відношення R: якщо в ньому є дуга (a, b), але немає дуги (b, a), то потрібно додати її.

Як знайти транзитивне замикання відношення R? Спочатку на прикладі покажемо, що попередня методика не приведе до успіху.

Приклад 5.23. Нехай на множині $\{1, 2, 3, 4\}$ задано відношення $R=\{(1, 3), (1, 4), (2, 1), (3, 2)\}$. Очевидно, що воно не транзитивне: не вистачає пар (1, 2), (2, 3), (2, 4), (3, 1).

Додавши ці пари, отримаємо відношення $R_1 = \{(1, 3), (1, 4), (2, 1), (3, 2), (1, 2), (2, 3), (2, 4), (3, 1)\}$. Воно також не транзитивне, бо містить пари (3, 1) і (1, 4), але не містить пари (3, 4).

Отже, побудувати транзитивне замикання відношення складніше, ніж рефлексивне чи симетричне.

Шляхом у відношенні R від елемента a до елемента b називають послідовність елементів $a, x_1, x_2, ..., x_{n-1}, b$ множини A таких, що $(a, x_1) \in R$, $(x_1, x_2) \in R$, ..., $(x_{n-1}, b) \in R$. Доведемо, що процедура відшукання транзитивного замикання відношення R еквівалентна процедурі визначення того, які пари вершин з'єднано шляхом. Зазначимо, що шлях у відношенні R від a до b відповідає шляху з вершини a у вершину b в графі G_R цього відношення.

ТЕОРЕМА 5.5. Нехай R — відношення на множині A. Шлях довжиною n від елемента a до елемента b у відношенні R існує тоді й лише тоді, коли $(a, b) \in R^n$.

Доведення. Застосуємо математичну індукцію. За означенням, шлях від елемента a до елемента b довжиною 1 існує тоді й лише тоді, коли $(a, b) \in R$. Отже, теорема справджується для $n = 1$.

Гіпотеза індукції: нехай теорема справджується для цілого невід'ємного $n > 1$. Шлях довжиною $n+1$ існує тоді й лише тоді, коли є такий елемент $x \in A$, що існує шлях довжиною 1 від елемента a до елемента x (тобто, $(a, x) \in R$) й існує шлях довжиною n від елемента x до елемента b (тобто, $(x, b) \in R^n$). Отже, з урахуванням індуктивної гіпотези, шлях довжиною $n+1$ з a в b існує тоді й лише тоді, коли є такий елемент $x \in A$, що $(a, x) \in R$ та $(x, b) \in R^n$. Але такий елемент x існує тоді й лише тоді, коли $(a, b) \in R^{n+1}$. Отже, шлях довжиною $n+1$ існує тоді й лише тоді, коли $(a, b) \in R^{n+1}$.

Нехай R — відношення на множині A. *З'єднувальним* називають відношення R^*, яке складається з таких пар (a, b), що існує шлях від елемента a до елемента b у відношенні R. Отже, з урахуванням теореми 5.5 $R^* = \bigcup_{n=1}^{\infty} R^n$.

ТЕОРЕМА 5.6. Транзитивне замикання відношення R дорівнює з'єднувальному відношенню R^*.

Доведення. Очевидно, що $R \subset R^*$. Потрібно довести такі твердження: відношення R^* транзитивне; $R^* \subset S$, де S — будь-яке транзитивне відношення таке, що $R \subset S$.

1. Нехай $(a, b) \in R^*$, $(b, c) \in R^*$. Звідси випливає, що існує шлях від елемента a до елемента b та шлях від елемента b до елемента c у відношенні R. Отже, існує шлях від елемента a до елемента c у відношенні R (він проходить через вершину b). Звідси випливає, що $(a, c) \in R^*$, тобто відношення R^* транзитивне.
2. Нехай відношення S транзитивне та $R \subset S$. Оскільки $S^* = \bigcup_{n=1}^{\infty} S^n$ та $S^n \subset S$ (за теоремою 5.4), то $S^* \subset S$. З умови $R \subset S$ випливає включення $R^* \subset S^*$, бо кожний шлях в R — це також шлях в S. Отже, $R^* \subset S^* \subset S$. Звідси випливає, що для будь-якого транзитивного відношення S такого, що $R \subset S$, виконується включення $R^* \subset S$. Це означає, що R^* — транзитивне замикання відношення R.

ТЕОРЕМА 5.7. Нехай множина A має n елементів і R — відношення на множині A. Тоді

$$R^* = \bigcup_{k=1}^{n} R^k = R \cup R^2 \cup R^3 \cup ... \cup R^n .$$

Доведення. Достатньо довести, що коли в графі відношення R існує шлях із вершини a у вершину b, то існує й шлях довжиною $m \leqslant n$ з a в b. Нехай $P = x_0, x_1, x_2, ..., x_{m-1}, x_m$ — найкоротший шлях з $a = x_0$ в $b = x_m$; його довжина дорівнює m. Припустимо, що $m > n$. Тоді за принципом коробок Дірихле серед вершин шляху P обов'язково є щонайменше дві однакові: нехай $x_i = x_j$, де $0 \leqslant i < j \leqslant m$. Тоді існує цикл, що проходить через вершину x_i та міститься в P; його можна вилучити. Отримаємо шлях із вершини a у вершину b, який проходить через вершини $x_0, x_1, ..., x_i, x_{j+1}, ..., x_{m-1}, x_m$ і має меншу, ніж m, довжину. Але це суперечить тому, що шлях P найкоротший і m — його довжина. Отже, $m \leqslant n$.

З теореми 5.7 випливає таке матричне подання відношення R^*:

$$M_{R^*} = M_R \vee M_R^{[2]} \vee M_R^{[3]} \vee ... \vee M_R^{[n]} .$$

Згідно з алгоритмом, що ґрунтується на цій формулі, для обчислення матриці M_{R^*} потрібно $O(n^4)$ операцій. Алгоритм Уоршалла (S. Warshall) ефективніший для побудови матриці M_{R^*}. Для його виконання потрібно $O(n^3)$ операцій. Розглянемо цей алгоритм.

Припустимо, що R — відношення на n-елементній множині A. Нехай $a_1, a_2, ..., a_n$ — довільна нумерація елементів цієї множини. В алгоритмі Уоршалла використано концепцію внутрішніх вершин шляху. Нагадаємо, що внутрішніми вершинами шляху $a, x_1, x_2, ..., x_{r-1}, b$ називають $x_1, x_2, ..., x_{r-1}$.

Алгоритм Уоршалла будує послідовність булевих матриць $W^{(0)}, W^{(1)}, ..., W^{(n)}$, де $W^{(0)} = M_R$ — матриця відношення R. Елементи матриці $W^{(k)}$ позначимо як $w_{ij}^{(k)}$. Якщо існує шлях із вершини a_i у вершину a_j, такий, що всі його внутрішні вершини містяться в множині $\{a_1, a_2, ..., a_k\}$, утвореній першими k вершинами, то $w_{ij}^{(k)} = 1$, а ні, то $w_{ij}^{(k)} = 0$. Перша й остання вершини такого шляху можуть і не належити множині вершин $\{a_1, a_2, ..., a_k\}$. Зазначимо, що $W^{(n)} = M_{R^*}$. Справді, (i, j)-й елемент цієї матриці дорівнює 1 тоді й лише тоді, коли існує шлях із вершини a_i до вершини a_j, внутрішні вершини якого належать множині $\{a_1, a_2, ..., a_n\}$.

Алгоритм Уоршалла ефективно обчислює матрицю $W^{(k)}$ за матицею $W^{(k-1)}$. Шлях із вершини a_i у вершину a_j із внутрішніми вершинами в множині $\{a_1, a_2, ..., a_k\}$ існує лише у двох випадках.

1. Якщо існує шлях з a_i у a_j із внутрішніми вершинами лише в множині $\{a_1, a_2, ..., a_{k-1}\}$ (рис. 5.8).
2. Якщо існує шлях з a_i в a_k та з a_k в a_j і кожний із цих шляхів має внутрішні вершини лише в множині $\{a_1, a_2, ..., a_{k-1}\}$ (рис. 5.9).

Усі внутрішні вершини
в множині
$\{a_1, a_2, ..., a_{k-1}\}$

Рис. 5.8

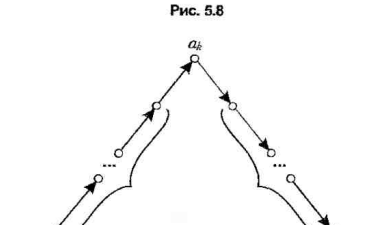


Рис. 5.9

У першому випадку шлях існує тоді й лише тоді, коли $w_{ij}^{(k-1)} = 1$; у другому — коли обидва елементи $w_{ik}^{(k-1)}$ та $w_{kj}^{(k-1)}$ дорівнюють 1. Отже,

$$w_{ij}^{(k)} = w_{ij}^{(k-1)} \vee \left(w_{ik}^{(k-1)} \wedge w_{kj}^{(k-1)}\right), \quad k = 1, 2, ..., n.$$

Для практичної реалізації останню формулу зручно перетворити так:

$$w_{ij}^{(k)} = w_{ij}^{(k-1)} \vee \left(w_{ik}^{(k-1)} \wedge w_{kj}^{(k-1)}\right) = \left(w_{ij}^{(k-1)} \vee w_{ik}^{(k-1)}\right) \wedge \left(w_{ij}^{(k-1)} \vee w_{kj}^{(k-1)}\right) =$$

$$= \begin{cases} w_{ij}^{(k-1)} \vee w_{kj}^{(k-1)}, \text{ якщо } w_{ik}^{(k-1)} = 1, \\ w_{ij}^{(k-1)}, \text{ якщо } w_{ik}^{(k-1)} = 0. \end{cases}$$

Крім того, очевидно, що в разі $i = k$ дії в першому рядку формули можна не виконувати. Отже,

$$w_{ij}^{(k)} = \begin{cases} w_{ij}^{(k-1)} \vee w_{kj}^{(k-1)}, \text{ якщо } i \neq k \text{ та } w_{ik}^{(k-1)} = 1, \\ w_{ij}^{(k-1)}, \text{ якщо } i = k \text{ чи } w_{ik}^{(k-1)} = 0. \end{cases}$$

Остання формула дає таке правило переходу від матриці $W^{(k-1)}$ до матриці $W^{(k)}$: для значень $i \neq k$ в разі $w_{ik}^{(k-1)} = 1$ замінити i-й рядок матриці $W^{(k-1)}$ на диз'юнкцію i-го й k-го рядків цієї матриці. Нижче подано алгоритм. Зазначимо, що i та j змінюються від 1 до n.

Алгоритм Уоршалла

Наведемо кроки алгоритму.

Крок 1. Присвоювання початкових значень. Виконати $W := M_R$, $k := 0$.

Крок 2. Виконати $k := k + 1$.

Крок 3. Для всіх $i \neq k$ таких, що $w_{ik} = 1$, і для всіх j виконати операцію $w_{ij} := w_{ij} \vee w_{kj}$.

Крок 4. Перевірка на закінчення. Якщо $k = n$, то зупинитись. Отримано розв'язок $W = M_{R^*}$, інакше перейти до кроку 2.

Приклад 5.24. Відношення задано матрицею

$$W^{(0)} = M_R = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

За алгоритмом Уоршалла побудуємо транзитивне замикання цього відношення.
Для $k = 1$ перший рядок залишаємо без змін ($i = k$), другий і третій рядки заміняємо на відповідну диз'юнкцію з першим:

$$W^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Для $k = 2$ $W^{(2)} = W^{(1)}$, бо всі елементи другого стовпця матриці $W^{(1)}$ нульові.
Далі, для $k = 3$ одержимо

$$W^{(3)} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix},$$

і, нарешті, коли $k = 4$, матимемо остаточний результат — матрицю транзитивного замикання

$$M_{R^*} = W^{(4)} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}.$$

5.7. Бази даних і відношення

Нехай $A_1, A_2, ..., A_n$ — множини. Підмножину декартового добутку $A_1 \times A_2 \times ... \times A_n$ називають *n-арним відношенням* на цих множинах. Самі множини $A_1, A_2, ..., A_n$ називають *доменами* відношення, а n — його *степенем*.

Приклад 5.25. Нехай R — відношення, що складається з трійок (a, b, c), де a. b та c — цілі числа й $a < b < c$. Тоді $(2, 7, 9) \in R$, але $(2, 9, 5) \notin R$. Степінь цього відношення дорівнює трьом. Усі три домени — множини цілих чисел.

Приклад 5.26. Нехай відношенням R складається з п'ятимісних кортежів (N. S, D, $T_в$, $T_п$), які подають розклад авіаліній. Тут N — номер рейсу, S — пункт відправлення, D — пункт призначення, $T_в$ — час вильоту, $T_п$ — час прибуття. Зокрема, якщо рейс № 17 вилітає зі Львова о 9:30 і прибуває до Києва об 11:10, то кортеж (17, Львів. Київ, 9:30. 11:10) належить відношенню R. Степінь цього відношення дорівнює п'яти, а його домени — це, відповідно, множина всіх номерів рейсів. дві множини міст. дві множини значень часу.

Час, потрібний для обробки інформації в базах даних, залежить від того, як цю інформацію нагромаджено. Операції додавання та вилучення записів, їх пошуку й комбінування виконуються у великих базах даних мільйони разів. У зв'язку з важливістю цих операцій розроблено різні методи для подання баз даних. Розглянемо один із них. який називають реляційною моделлю даних. Цей метод грунтується на концепції n-арних відношень.

База даних складається із *записів*. які являють собою n-місні кортежі. Елементи кортежів утворюють *поля*. Наприклад, база даних, яка містить певну інформацію про студентів, може бути утворена полями, що містять. відповідно. прізвище, студентський номер, назву основної спеціальності та середній бал навчання. Цю реляційну базу даних подають у вигляді записів кортежів 4-арного відношення. Отже, студентові відповідає чотиримісний кортеж вигляду (ПРІЗВИЩЕ, СТУД_НОМЕР, СПЕЦ, БАЛ).

Приклад 5.27. У цьому прикладі наведено базу даних із шести записів:
(Бабенко П. С., 270436, Інформатика, 3.95),
(Біленко Б. М., 184931, Фізика, 4.09).
(Грінченко І. І., 270297, Інформатика. 4.15).
(Дацків І. А., 488934, Математика, 3.88),
(Яворський П. М., 488721. Математика, 4.15).
(Яковенко С. Ю., 295681. Психологія, 4.27).

Відношення, використовувані для подання баз даних, називають також *таблицями*, бо їх часто зображають у вигляді таблиць. Зокрема, базу даних студентів із прикладу 5.27 можна подати у вигляді табл. 5.1.

Таблиця 5.1

Прізвище	Студ. номер	Спеціальність	Бал
Бабенко П. С.	270436	Інформатика	3.95
Біленко Б. М.	184931	Фізика	4.09
Грінченко І. І.	270297	Інформатика	4.15
Дацків І. А.	488934	Математика	3.88
Яворський П. М.	488721	Математика	4.15
Яковенко С. Ю.	295681	Психологія	4.27

Домен n-арного відношення називають ***первинним ключем***, якщо значення з цього домену в кортежі однозначно задає весь кортеж. Інакше кажучи, домен являє собою первинний ключ, якщо у відношенні немає двох кортежів з одним і тим самим значенням із цього домену.

Приклад 5.28. У відношенні з табл. 5.1 первинними ключами можуть бути домени „прізвище" та „студ. номер", а домени „спеціальність" і „бал" не можуть бути первинними ключами, тому що більше ніж один кортеж має однакові значення з цих доменів.

Комбінації доменів також можуть однозначно визначати кортежі в n-арних відношеннях. Якщо значення якоїсь сукупності доменів однозначно задають кортеж у відношенні, то декартів добуток цих доменів називають *комбінованим ключем*.

Записи можна вилучати з бази та додавати до неї. Тому потрібно бути впевненим, що кожний новий запис відрізняється в полі ключа (або в полях комбінованого ключа) від усіх інших записів у цій таблиці. Зокрема, як первинний ключ у прикладах 5.27, 5.28 доцільно вибрати „студ. номер", бо прізвище з ініціалами може повторюватись, а номер унікальний.

Операції над n-арними відношеннями використовують для утворення нових n-арних відношень. Розглянемо дві операції — проекцію та з'єднання.

Проекція $P_{i_1,i_2,\ldots,i_m}$ відображає кортеж $(a_1, a_2, \ldots, a_n)$ довжиною n у кортеж $(a_{i_1}, a_{i_2}, \ldots, a_{i_m})$ довжиною m, де $m \leqslant n$. Інакше кажучи, проекція $P_{i_1,i_2,\ldots,i_m}$ виключає $n-m$ компонент кортежу довжиною n, залишаючи тільки компоненти з номерами $i_1, i_2, \ldots, i_m$.

Якщо після виконання проекції з'являться однакові кортежі, то їх записують лише один раз.

Приклад 5.29. У табл. 5.2 задано відношення, а в табл. 5.3 — його проекцію $P_{1,2}$.

Таблиця 5.2

Викладач	Факультет	Назва курсу
Зінченко С. В.	Математичний	Алгебра
Зінченко С. В.	Математичний	Аналітична геометрія
Коваль Ю. М.	Інформатики	Дискретна математика
Коваль Ю. М.	Інформатики	Системний аналіз
Яремчук А. І.	Фізичний	Теоретична фізика
Яремчук А. І.	Фізичний	Загальна фізика

Таблиця 5.3

Викладач	Факультет
Зінченко С. В.	Математичний
Коваль Ю. М.	Інформатики
Яремчук А. І.	Фізичний

Операцію з'єднання використовують для об'єднання двох таблиць в одну за умови, що вони мають деякі однакові поля. Нехай R — відношення степеня m, а S —

відношення степеня n. *З'єднання* $J_p(R, S)$, де $p \leqslant \min\{m, n\}$ — це відношення степеня $m+n-p$, яке містить усі кортежі $(a_1, a_2, ..., a_{m-p}, c_1, c_2, ..., c_p, b_1, b_2, ..., b_{n-p})$ довжиною $m+n-p$, де $(a_1, a_2, ..., a_{m-p}, c_1, c_2, ..., c_p) \in R$, $(c_1, c_2, ..., c_p, b_1, b_2, ..., b_{n-p}) \in S$. Інакше кажучи, операція J_p утворює нове відношення з двох наявних відношень R степеня m та S степеня n, з'єднуючи всі кортежі відношення R з усіма кортежами відношення S, де останні p компонент кортежів з R збігаються з першими p компонентами кортежів з S.

Приклад 5.30. Знайти відношення, отримане після застосування операції J_2 до відношень, заданих табл. 5.2, 5.4.

Таблиця 5.4

Факультет	Назва курсу	Аудиторія	Час
Інформатики	Дискретна математика	А111	9:30
Інформатики	Системний аналіз	А270	13:25
Математичний	Алгебра	А439	11:35
Математичний	Аналітична геометрія	А377	15:20
Фізичний	Теоретична фізика	Г216	9:30
Фізичний	Загальна фізика	Г105	13:25

Результат виконання операції з'єднання J_2 наведено в табл. 5.5.

Таблиця 5.5

Викладач	Факультет	Назва курсу	Аудиторія	Час
Зінченко С.В.	Математичний	Алгебра	А439	11:35
Зінченко С.В.	Математичний	Аналітична геометрія	А377	15:20
Коваль Ю.М.	Інформатики	Дискретна математика	А111	9:30
Коваль Ю.М.	Інформатики	Системний аналіз	А270	13:25
Яремчук А.І.	Фізичний	Теоретична фізика	Г216	9:30
Яремчук А.І.	Фізичний	Загальна фізика	Г105	13:25

Окрім проекції та з'єднання можна виконувати й інші операції над n-арними відношеннями. Їх докладно описано в літературі з теорії баз даних.

Контрольні запитання та завдання

1. Записати всі впорядковані пари, які утворюють відношення R із множини $A = \{0, 1, 2, 3, 4\}$ в множину $B = \{0, 1, 2, 3\}$, де $(a, b) \in R$, якщо й лише якщо:

   а) $a = b$; б) $a + b = 4$; в) $a > b$;

   г) a ділить b; д) НСД$(a, b) = 1$; е) НСК$(a, b) = 2$.

   Тут НСД — найбільший спільний дільник, НСК — найменше спільне кратне.

2. Для кожного з відношень, наведених нижче, на множині {1, 2, 3, 4} визначити, чи воно рефлексивне, іррефлексивне, симетричне, антисиметричне, асиметричне, транзитивне:

   а) {(2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4)};

   б) {(1, 1), (1, 2). (2, 1), (2, 2), (3, 3), (4, 4)};

   в) {(2, 4), (4, 2)};

   г) {(1, 2), (2, 3), (3, 4)};

   д) {(1, 1), (2, 2), (3, 3), (4, 4)};

   е) {(1, 3), (1, 4), (2, 3), (2, 4), (3, 1), (3, 4)}.

3. Визначити, чи відношення R на множині всіх людей рефлексивне, іррефлексивне, симетричне, антисиметричне, асиметричне, транзитивне, де $(a, b) \in R$, якщо й лише якщо:

   а) a вищий, ніж b;

   б) a та b народилися в один і той самий день;

   в) a має те саме прізвище, що й b;

   г) a та b мають спільних дідуся й бабусю.

4. Визначити, чи відношення R на множині цілих чисел рефлексивне, симетричне, антисиметричне, транзитивне, де $(x, y) \in R$, якщо й лише якщо:

   а) $x \neq y$;

   б) $xy \leqslant 1$;

   в) $x = y + 1$ або $x = y - 1$;

   г) x та y обидва від'ємні чи невід'ємні;

   д) $x = y^2$;

   е) $x \geqslant y^2$.

Нехай R — відношення з множини A в множину B. Відношення $\overline{R} = \{(a,b) \mid (a,b) \notin R\}$ називають *доповнювальним*. Відношення $R^{-1} = \{(b,a) \mid (a,b) \in R\}$ називають *обертним*.

5. Нехай R — відношення на множині цілих чисел, $R = \{(a,b) \mid a < b\}$. Знайти:

   а) $\overline{R}$; б) R^{-1}.

6. Нехай R — відношення на множині натуральних чисел, $R = \{(a,b) \mid a$ ділить $b\}$. Знайти:

   а) $\overline{R}$; б) R^{-1}.

7. Записати всі 16 різних відношень на множині {0, 1}. Скільки з них містять пару (0, 1)?

8. Скільки з 16 відношень на множині {0, 1}, записаних у розв'язанні задачі 7:

   а) рефлексивні;

   б) іррефлексивні;

   в) симетричні;

г) антисиметричні;
д) асиметричні;
е) транзитивні?

9. Скільки різних відношень на множині з n елементів:
а) симетричні;
б) антисиметричні;
в) асиметричні;
г) іррефлексивні;
д) рефлексивні й симетричні;
е) ні рефлексивні, ні іррефлексивні?

10. Скільки є транзитивних відношень на множині з n елементів, якщо:
а) $n=1$; б) $n=2$; в) $n=3$?

11. Знайти помилку в „доведенні" такої „теореми".

ТЕОРЕМА. Нехай R – симетричне й транзитивне відношення на множині A. Тоді R є рефлексивне.

Доведення. Нехай $a \in A$. Виберемо такий елемент $b \in A$, що $(a, b) \in R$. Оскільки відношення R симетричне, то й $(b, a) \in R$. Позаяк відношення R транзитивне, то з $(a, b) \in R$ і $(b, a) \in R$ випливає $(a, a) \in R$. Отже, відношення R рефлексивне.

12. Довести, що відношення R на множині A симетричне тоді й лише тоді, коли $R = R^{-1}$, де R^{-1} – обернене відношення (див. інформацію перед задачею 5).

13. Довести, що відношення R на множині A антисиметричне тоді й лише тоді, коли $R \cap R^{-1}$ – підмножина діагонального відношення $\Delta = \{(a, a) \mid a \in A\}$.

14. Довести, що відношення R на множині A рефлексивне тоді й лише тоді, коли обернене відношення R^{-1} рефлексивне.

15. Довести, що відношення R на множині A рефлексивне тоді й лише тоді, коли доповнювальне відношення $\overline{R}$ іррефлексивне.

16. Задати кожне з відношень, наведених нижче, на множині {1, 2, 3} за допомогою матриці:
а) {(1, 1), (1, 2), (1, 3)};
б) {(1, 2), (2, 1), (2, 2), (3, 3)};
в) {(1, 1), (1, 2), (1, 3), (2, 2), (2. 3), (3, 3)};
г) {(1, 3), (3, 1)}.

17. Виписати впорядковані пари елементів відношення на множині {1, 2, 3}, які відповідають наведеним нижче матрицям (рядки та стовпці відповідають числам, розміщеним у порядку зростання):

а) $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$; б) $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$; в) $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

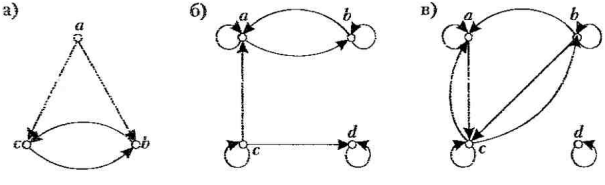
Визначити, які з цих відношень рефлексивні, іррефлексивні, симетричні, антисиметричні, асиметричні, транзитивні.

18. Для кожного з відношень задачі 16 побудувати граф.

19. Для кожного з відношень задачі 17 побудувати граф.

20. Зобразити граф відношення $R=\{(a, a), (a, b), (b, c), (c, b), (c, d), (d, a), (d, b)\}$ на множині $A=\{a, b, c, d\}$.

21. Записати впорядковані пари елементів, які подають кожне відношення, задане графами, і визначити властивості цих відношень.



22. Які з наведених нижче відношень на множині {0, 1, 2, 3} являють собою відношення еквівалентності? Зазначити, чому інші відношення не є відношеннями еквівалентності:

а) {(0, 0), (1, 1), (2, 2), (3, 3)};

б) {(0, 0), (0, 2), (2, 0), (2, 2), (2, 3), (3, 2), (3, 3)};

в) {(0, 0), (1, 1), (1, 2), (2, 1), (2, 2), (3, 3)};

г) {(0, 0), (1, 1), (1, 3), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)};

д) {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 2), (3, 3)}.

23. Які з наступних відношень на множині всіх людей являють собою відношення еквівалентності? Зазначити, чому інші відношення не є відношеннями еквівалентності:

а) $\{(a, b) \mid a$ та b одного віку$\}$;

б) $\{(a, b) \mid a$ та b мають одних і тих самих батьків$\}$;

в) $\{(a, b) \mid a$ та b мають спільного батька чи матір$\}$;

г) $\{(a, b) \mid a$ та b зустрілись$\}$;

д) $\{(a, b) \mid a$ та b розмовляють однією мовою$\}$.

24. Які з наступних відношень на множині всіх функцій із Z у Z являють собою відношення еквівалентності? Зазначити, чому інші відношення не є відношеннями еквівалентності (Z — множина цілих чисел):

а) $\{(f, g) \mid f(1)=g(1)\}$;

б) $\{(f, g) \mid f(0)=g(0)$ чи $f(1)=g(1)\}$;

в) $\{(f, g) \mid f(x)-g(x)=1$ для всіх $x \in Z\}$;

г) $\{(f, g) \mid f(x)-g(x)=C$ для якогось $C \in Z$ і для всіх $x \in Z\}$;

д) $\{(f, g) \mid f(0)=g(1)$ і $f(1)=g(0)\}$.

25. Задайте три відношення еквівалентності на множині студентів вашої академічної групи. Визначте класи еквівалентності для цих відношень еквівалентності.

26. Нехай A — непорожня множина, f — функція, означена на множині A. Відношення R складається з усіх упорядкованих пар (x, y) таких, що $f(x)=f(y)$:

а) довести, що R — відношення еквівалентності на A;

б) які класи еквівалентності породжує відношення R?

27. Нехай A — непорожня множина, R — відношення еквівалентності на A. Довести, що існує така функція, означена на множині A, що $(x, y) \in R$ тоді й лише тоді, коли $f(x)=f(y)$.

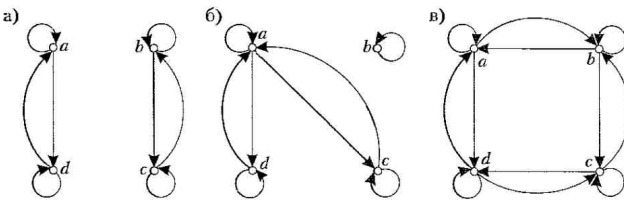
28. Відношення R, яке складається з усіх пар (α, β), де α та β — бітові рядки довжиною не менше ніж три, що збігаються в перших трьох бітах, являє собою відношення еквівалентності на множині всіх бітових рядків. Довести.

29. Довести, що тотожність формул логіки висловлювань — відношення еквівалентності на множині всіх формул.

30. Які з наведених нижче матриць подають відношення еквівалентності:

а) $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$; б) $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$; в) $\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$?

31. Визначити, які з графів являють собою графи відношень еквівалентності.



32. Довести, що відношення R на множині всіх бітових рядків таке, що $\alpha R \beta$ тоді й лише тоді, коли α та β мають однакову кількість одиниць, являє собою відношення еквівалентності.

33. Для відношень еквівалентності із задач 22–24 наведіть класи еквівалентності.

34. Знайти клас еквівалентності бітового рядка 011 для відношення еквівалентності із задачі 32.

35. Для бітових рядків, наведених нижче, знайти класи еквівалентності відношення еквівалентності із задачі 28:

а) 010; б) 1011; в) 11111; г) 01010101.

36. Знайти класи конгруентності $[4]_m$ для таких значень m:

а) 2; б) 3;

в) 6; г) 8.

37. Опишіть кожний із класів конгруентності за mod 6.

38. Які з наступних систем підмножин — розбиття множини $A=\{1, 2, 3, 4, 5, 6\}$? Для кожної системи підмножин, що являє собою розбиття множини A, побудувати відповідне відношення еквівалентності на множині A:

а) $\{\{1, 2\}, \{2, 3, 4\}, \{4, 5, 6\}\}$; б) $\{\{1\}, \{2, 3, 6\}, \{4\}, \{5\}\}$;

в) $\{\{2, 4, 6\}, \{1, 3, 5\}\}$; г) $\{\{1, 4, 5\}, \{2, 6\}\}$.

39. Скільки різних відношень еквівалентності можна задати на множині з чотирьох елементів?

40. На множині цілих чисел Z задано відношення R. У яких випадках множина (Z, R) частково впорядкована:

а) aRb тоді й лише тоді, коли $a=b$;

б) aRb тоді й лише тоді, коли $a \neq b$;

в) aRb тоді й лише тоді, коли $a \geqslant b$;

г) aRb тоді й лише тоді, коли a не ділить b?

41. Визначити, які з матриць подають відношення часткового порядку:

а) $\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, б) $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$, в) $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$.

42. Які з графів зображають відношення часткового порядку?

43. Нехай (A, R) — частково впорядкована множина. Довести, що множина (A, R^{-1}) також частково впорядкована. Тут R^{-1} — обернене відношення (див. інформацію перед задачею 5).

44. Побудувати діаграму Гассе для відношення „більше чи дорівнює" на множині $\{0, 1, 2, 3, 4, 5\}$.

45. Побудувати діаграму Гассе для відношення $R=\{(a, b) \mid a$ ділить $b\}$ на множині A:

а) $A=\{1, 2, 3, 4, 5, 6, 7, 8\}$; б) $A=\{1, 2, 3, 5, 7, 11, 13\}$;

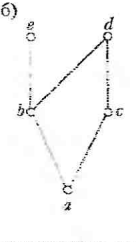
в) $A=\{1, 2, 3, 6, 12, 24, 36, 48\}$; г) $A=\{1, 2, 4, 8, 16, 32, 64\}$.

46. Побудувати діаграму Гассе для відношення $R=\{(A, B) \mid A \subset B\}$ на булеані $P(A)$, де $A=\{a, b, c\}$ (див. підрозділ 1.12).

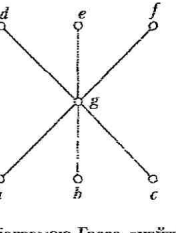
47. Записати всі впорядковані пари відношення часткового порядку з такою діаграмою Гассе:

а)

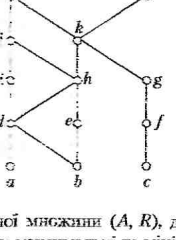
б)



в)



48. Для відношення часткового порядку, поданого діаграмою Гассе, знайти максимальні та мінімальні елементи.



49. Для частково впорядкованої множини (A, R), де $A=\{3, 5, 9, 15, 24, 45\}$, $R=\{(a, b) \mid a$ ділить $b\}$, знайти максимальні та мінімальні елементи.

50. Виконати топологічне сортування для частково впорядкованої множини, заданої діаграмою Гассе із задачі 48.

51. Виконати топологічне сортування для частково впорядкованої множини (A, R), де $A=\{1, 2, 3, 6, 8, 12, 24, 36\}$, $R=\{(a, b) \mid a$ ділить $b\}$.

52. Знайти відмінну від наведеної в прикладі 5.17 послідовність робіт для виконання завдань, з яких складається проект комп'ютерної компанії.

53. Нехай R і S — відношення на множині $A=\{1, 2, 3\}$, задані матрицями

$$M_R=\begin{pmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 1 & 0 & 0\end{pmatrix}, \qquad M_S=\begin{pmatrix}0 & 1 & 0\\ 0 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}.$$

Знайти матриці відношень:

а) $R\cup S$; б) $R\cap S$; в) $R\oplus S$; г) $S\circ R$; д) $R\circ R$.

54. Нехай відношення R задано матрицею

$$M_R=\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\end{pmatrix}.$$

Знайти матриці відношень:

а) R^2; б) R^3; в) R^4.

55. Нехай R — відношення на множині $A=\{0, 1, 2, 3\}$, яке складається з упорядкованих пар (0, 1), (1, 1), (1, 2), (2, 0), (2, 2) та (3, 0). Знайти:

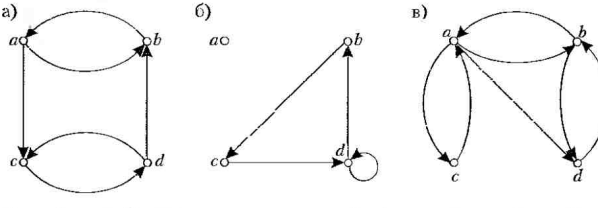
а) рефлексивне замикання відношення R;

б) симетричне замикання відношення R.

56. Нехай на множині Z цілих чисел задано відношення $R=\{(a, b)\mid a\neq b\}$. Знайти його рефлексивне замикання.

57. Як граф, що зображає рефлексивне замикання відношення на скінченній множині, можна побудувати з графа цього відношення?

58. Зобразити граф рефлексивного замикання для кожного з відношень, заданих графами:



59. Як граф, що зображає симетричне замикання відношення на скінченній множині, можна побудувати з графа цього відношення?

60. Знайти графи симетричного замикання відношень, заданих графами задачі 58.

61. Знайти найменше відношення, яке містить відношення $R=\{(a, b)\mid a>b\}$ на множині цілих чисел і водночас рефлексивне та симетричне.

62. Знайти граф найменшого відношення, яке водночас рефлексивне та симетричне для кожного з відношень, заданих графами задачі 58.

63. Відношення R на скінченній n-елементній множині A задано матрицею M_R. Довести, що матриця, яка задає рефлексивне замикання R, має вигляд $M_R \vee I_n$, де I_n — одинична $n \times n$ матриця.

64. Відношення R на скінченній множині A задано матрицею M_R. Довести, що матриця, яка задає симетричне замикання відношення R, має вигляд $M_R \vee M_R^T$.

65. Довести, що замикання відношення R за властивістю P, якщо воно існує, являє собою перетин усіх відношень, що містять R і мають властивість P.

66. Нехай R — відношення на множині {1, 2, 3, 4, 5}, яке складається з упорядкованих пар (1, 3), (2, 4), (3, 1), (3, 5), (4, 3), (5, 1), (5, 2), (5, 4). Знайти:

а) R^2; б) R^3; в) R^4;

г) R^5; д) R^6; е) R^*.

67. Нехай відношення R утворено парами (a, b), де a та b — міста, між якими є пряма авіалінія. Коли пара (a, b) міститься в таких множинах:

а) R^2; б) R^3; в) R^*?

68. Нехай R — відношення на множині всіх студентів, яке складається з усіх пар (a, b), де студенти a та b слухають принаймні один спільний курс і $a \neq b$. Коли пара (a, b) міститься в таких множинах:

а) R^2; б) R^3; в) R^*?

69. Нехай відношення R рефлексивне. Довести, що відношення R^* також рефлексивне.

70. Нехай відношення R симетричне. Довести, що відношення R^* також симетричне.

71. Нехай відношення R іррефлексивне. Чи обов'язково буде іррефлексивним відношення R^2?

72. За алгоритмом Уоршалла побудувати транзитивні замикання відношень на множині {1, 2, 3, 4}:

а) {(1, 2), (2, 1), (2, 3), (3, 4), (4, 1)};

б) {(2, 1), (2, 3), (3, 1), (3, 4), (4, 1), (4, 3)};

в) {(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)};

г) {(1. 1), (1. 4), (2, 1), (2, 3). (3, 1), (3, 2), (3, 4), (4, 2)}.

73. За алгоритмом Уоршалла побудувати транзитивні замикання наведених нижче відношень на множині $\{a, b, c, d, e\}$:

а) {(a, c), (b, d), (c, a), (d, b), (e, d)};

б) {(b, c), (b, e), (c, e), (d, a), (e, b), (e, c)};

в) {(a, b), (a, c), (a, e), (b, a), (b, c), (c, a), (c, b), (d, a), (e, d)};

г) {(a, e), (b, a), (b, d), (c, d), (d, a), (d, c), (e, a), (e, b), (e, c), (e, e)}.

74. Знайти найменше відношення на множині $A = \{1, 2, 3, 4\}$, яке містить відношення $R = \{(1, 2), (1, 4), (3, 3), (4, 1)\}$ і має такі властивості:

а) воно рефлексивне та транзитивне;

б) симетричне та транзитивне;

в) рефлексивне, симетричне та транзитивне.

75. Довести, що замикання за властивістю P відношення $R = \{(0, 0), (0, 1), (1, 1), (2, 2)\}$ на множині $\{0, 1, 2\}$ не існує, якщо властивість P така:

а) відношення не рефлексивне;

б) кількість елементів відношення непарна.

76. Чи ми обов'язково одержимо відношення еквівалентності, побудувавши транзитивне замикання симетричного замикання рефлексивного замикання довільного відношення?

77. Чи ми обов'язково одержимо відношення еквівалентності, побудувавши симетричне замикання рефлексивного замикання транзитивного замикання довільного відношення?

78. Модифікувати алгоритм Уоршалла для відшукання рефлексивного замикання транзитивного замикання відношення на n-елементній множині.

79. Запропонувати алгоритм для побудови найменшого відношення еквівалентності, яке містить дане відношення.

80. Записати всі трійки, які утворюють відношення $\{(a, b, c) \mid a, b, c$ цілі та $0 < a < b < c < 5\}$.

81. Записати всі четвірки, які утворюють відношення $\{(a, b, c, d) \mid a, b, c, d$ цілі додатні й $abcd = 6\}$.

82. Знайти результат застосування проекції $P_{1,2,4}$ до відношення із задачі 81.

83. Скільки компонент є в кортежі з таблиці, одержаній застосуванням оператора f_3 до двох таблиць, що складаються відповідно з п'ятикомпонентних і восьмикомпонентних кортежів?

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Відношення на скінченній множині задано матрицею. Визначити, чи воно рефлексивне, іррефлексивне.
2. Відношення на скінченній множині задано матрицею. Визначити, чи воно симетричне, антисиметричне, асиметричне.
3. Відношення на скінченній множині задано матрицею. Визначити, чи воно транзитивне.
4. На скінченній множині задано частковий порядок. Побудувати діаграму Гассе та знайти всі максимальні й мінімальні елементи.

5. На скінченній множині задано частковий порядок. За допомогою топологічного сортування знайти сумісний із ним лінійний порядок.
6. Відношення на скінченній множині задано матрицею. Знайти матрицю його рефлексивного замикання.
7. Відношення на скінченній множині задано матрицею. Знайти матрицю його симетричного замикання.
8. Відношення на скінченній множині задано матрицею. Знайти матрицю його транзитивного замикання. Застосувати алгоритм Уоршалла.
9. Задано матрицю відношення R на скінченній множині. Знайти матрицю найменшого відношення еквівалентності, яке містить відношення R.
10. Задано n-арне відношення. Знайти його проекцію; поля проекції зазначено.
11. Задано m-арне й n-арне відношення, а також сукупність їх спільних полів. Знайти з'єднання заданих відношень за цими полями.

Розділ 6
Основи теорії кодування

- Алфавітне й рівномірне кодування
- Достатні умови однозначності декодування. Властивості роздільних кодів
- Оптимальне кодування
- Коди, стійкі до перешкод. Коди Хеммінга

Кодування (у широкому розумінні) – це перехід від одного способу подання інформації до іншого, який дає змогу відновлювати початкову інформацію. Теорія кодування виникла в 40-х роках XX століття після появи робіт К. Шеннона (C. Shannon). У цій теорії досліджуються методи кодування, пов'язані з основними математичними моделями, які відображають істотні риси реальних інформаційних систем.

6.1. Алфавітне й рівномірне кодування

Без утрати загальності можна сформулювати задачу кодування так. Нехай задано алфавіт $A=\{a_1, ..., a_r\}$ зі скінченної кількості символів, які називають *буквами*. Скінченну послідовність букв $a_{i_1}a_{i_2}...a_{i_l}$ алфавіту A називають *словом* у цьому алфавіті. Для позначення слів будемо використовувати малі грецькі літери: $\alpha = a_{i_1}a_{i_2}...a_{i_l}$. Число l – кількість букв у слові α – називають *довжиною* слова α та позначають $l(\alpha)$. Множину всіх слів у алфавіті A позначають як A^*. Порожнє слово позначають λ; зазначимо, що $\lambda \notin A$, $\lambda \in A^*$, $l(\lambda)=0$.

Якщо слово α має вигляд $\alpha=\alpha_1\alpha_2$, то α_1 називають *початком* (*префіксом*) слова α, а α_2 – його *закінченням* (*постфіксом*). Якщо при цьому $\alpha_1 \neq \lambda$ (відповідно $\alpha_2 \neq \lambda$), то α_1 (відповідно α_2) називають *власним початком* (відповідно *власним закінченням*) слова α.

Нехай S – підмножина множини A^*: $S \subset A^*$; елементи множини S називають *повідомленнями*. Нехай також задано скінченний алфавіт $B=\{b_1, ..., b_q\}$. Як β позначимо слово в алфавіті B, як B^* – множину всіх слів у алфавіті B. Задамо відображення F, яке кожному слову $\alpha \in S$ ставить у відповідність слово $\beta=F(\alpha)$, $\beta \in B^*$. Слово β називають *кодом повідомлення* α. Перехід від слова α до його коду називають *кодуванням*.

Якщо $|B| = q$, то кодування називають *q-ковим*. Найчастіше використовують алфавіт $B = \{0, 1\}$, тобто *двійкове кодування*. Саме його ми й розглядатимемо в наступних підрозділах, випускаючи слово „двійкове".

Відображення F задають якимось алгоритмом. Є два способи задати відображення F.

1. *Алфавітне кодування* задають схемою (або таблицею кодів) σ:

$$a_1 \to \beta_1,$$
$$a_2 \to \beta_2,$$
$$\ldots\ \ldots\ \ldots$$
$$a_r \to \beta_r,$$

де $a_i \in A$, $\beta_i \in B^*$, $i = 1, \ldots, r$. Схема σ задає відповідність між буквами алфавіту A та деякими словами в алфавіті B. Вона визначає алфавітне кодування так: кожному слову $\alpha = a_{i_1} a_{i_2} \ldots a_{i_l}$ із повідомленням S поставлено у відповідність слово $\beta = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_l}$ — його *код*. Слова $\beta_1, \ldots, \beta_r$ називають *елементарними кодами*.

2. *Рівномірне кодування* з *параметрами* k й n задають так. Повідомлення α розбивають на блоки довжиною k:

$$\alpha = (x_1 \ldots x_k)\ (x_{k+1} \ldots x_{2k}) \ldots (x_{mk+1} \ldots x_{mk+s}),$$

де $s \leqslant k$ (останній блок може бути коротшим, у такому разі спосіб його кодування спеціально обумовлюють), $x_i \in A$, $i = 1, \ldots, mk+s$. Блоки довжиною k розглядають як „букви" якогось алфавіту (таких блоків, очевидно, r^k, бо алфавіт A складається з r букв) і кодують словами в алфавіті B довжиною n за *схемою рівномірного кодування* $\sigma_{k,n}$:

$$\alpha_1 \to \beta_1,$$
$$\alpha_2 \to \beta_2,$$
$$\alpha_3 \to \beta_3,$$
$$\ldots\ \ldots\ \ldots$$
$$\alpha_{r^k} \to \beta_{r^k}.$$

Надлишковістю схеми $\sigma_{k,n}$ *на символ повідомлення* називають величину $R = (n-k)/k = (n/k) - 1$.

Отже, ми описали дві можливості задати відображення F: $S \to B^*$. Однозначне декодування можливе, якщо існує обернене відображення F^{-1}.

6.2. Достатні умови однозначності декодування. Властивості роздільних кодів

Розглянемо схему алфавітного кодування σ та різні слова, складені з елементарних кодів. Схему σ називають *роздільною*, якщо з рівності $\beta_{i_1} \ldots \beta_{i_k} = \beta_{j_1} \ldots \beta_{j_l}$ випливає, що $k = l$ та $i_t = j_t$ для кожного $t = 1, \ldots, k$, тобто будь-яке слово, складене

з елементарних кодів, можна єдиним способом розкласти на елементарні коди. Очевидно, що алфавітне кодування з роздільною схемою вможливлює однозначне декодування [35, 49].

Схему σ називають *префіксною*, якщо для будь-яких i та j $(i, j = 1, ..., r;\ i \neq j)$ елементарний код β_i не є префіксом елементарного коду β_j.

ТЕОРЕМА 6.1. Префіксна схема є роздільною.

Доведення. Доведемо цю теорему від протилежного. Оскільки схема σ префіксна, то всі елементарні коди в ній попарно різні, тобто $\beta_i \neq \beta_j$, якщо $i \neq j$. Нехай кодування зі схемою σ не роздільне. Тоді існує таке слово $\beta \in B^*$, що можуть бути два розклади на елементарні коди:

$$\beta = \beta_{i_1} ... \beta_{i_n} = \beta_{j_1} ... \beta_{j_l}.$$

Нехай $\beta_{i_1} = \beta_{j_1}$, ..., $\beta_{i_{p-1}} = \beta_{j_{p-1}}$ але $\beta_{i_p} \neq \beta_{j_p}$. У такому разі одне зі слів β_{i_p}, β_{j_p} являє собою префікс іншого, а це суперечить тому, що схема σ префіксна.

Властивість префіксності достатня, але не необхідна умова роздільності схеми.

Приклад 6.1. Нехай $A = \{a, b\}$, $B = \{0, 1\}$, σ: $a \to 0$, $b \to 01$. Схема σ не префіксна, але роздільна. Справді, перед кожним входженням 1 в слові β_2 безпосередньо є 0. Це дає змогу видлити всі пари (01). Частина слова, що залишилась, складається із символів 0.

Нехай $\beta = b_{i_1} b_{i_2} ... b_{i_n}$ — слово з B^*. Позначимо як $\tilde{\beta}$ слово, одержане оберненням слова β, тобто $\tilde{\beta} = b_{i_n} b_{i_{n-1}} ... b_{i_1}$. Позначимо як $\tilde{\sigma}$ схему

$$a_1 \to \tilde{\beta}_1,$$
$$a_2 \to \tilde{\beta}_2,$$
$$\ldots\ \ldots\ \ldots$$
$$a_r \to \tilde{\beta}_r.$$

Приклад 6.2. Візьмемо схему σ з прикладу 6.1. Тоді $\tilde{\sigma}$ має такий вигляд: $a \to 0$, $b \to 10$. Схема $\tilde{\sigma}$ префіксна, тому за теоремою 6.1 вона роздільна.

Зауваження. Схеми σ та $\tilde{\sigma}$ водночас або роздільні, або ні.

Це зауваження дає змогу посилити теорему 6.1.

ТЕОРЕМА 6.2. Якщо або схема σ, або схема $\tilde{\sigma}$ префіксна, то обидві схеми σ та $\tilde{\sigma}$ роздільні.

Можна навести приклад такої роздільної схеми σ, що ні σ, ні $\tilde{\sigma}$ не префіксні. Отже, теорема 6.2 також дає достатню, але не необхідну умову роздільності схеми.

Нехай задано схему алфавітного кодування σ: $a_i \to \beta_i$, $a_i \in A$, $\beta_i \in B^*$, $i = 1, ..., r$. Для того, щоб схема алфавітного кодування була роздільною, необхідно, щоб довжини елементарних кодів задовольняли *нерівність Мак-Міллана* (B. McMillan).

ТЕОРЕМА 6.3 (нерівність Мак-Міллана). Якщо схема алфавітного кодування σ роздільна, то

$$\sum_{i=1}^{r} \frac{1}{2^{l_i}} \leqslant 1, \text{ де } l_i = l(\beta_i).$$

Доведення. Позначимо $L = \max(l_1, l_2, ..., l_r)$. Запишемо n-й степінь лівої частини нерівності:

$$\left(\sum_{i=1}^{r} 2^{-l_i} \right)^n.$$

Розкривши дужки, отримаємо суму

$$\sum_{(i_1,\ldots,i_n)} \left(2^{l_{i_1}+\ldots+l_{i_n}}\right)^{-1},$$

де $(i_1, ..., i_n)$ — різні набори номерів елементарних кодів. Позначимо як $\nu(n, m)$ кількість доданків вигляду $1/2^m$, які входять у цю суму; тут $m = l_{i_1} + ... + l_{i_n}$. Для деяких m може бути $\nu(n, m) = 0$. Зведемо подібні члени й одержимо суму

$$\sum_{m=1}^{nL} \frac{\nu(n,m)}{2^m}.$$

Кожному доданку вигляду $(2^{l_{i_1}+\ldots+l_{i_n}})^{-1}$ можна однозначно зіставити код (слово в алфавіті B) вигляду $\beta_{i_1}...\beta_{i_n}$. Воно складається з n елементарних кодів і має довжину m. Отже, $\nu(n, m)$ — це кількість деяких слів вигляду $\beta_{i_1}...\beta_{i_n}$ таких, що $l(\beta_{i_1}...\beta_{i_n}) = m$. Позаяк схема σ роздільна, то $\nu(n, m) \leqslant 2^m$. Справді, коли припустити, що $\nu(n, m) > 2^m$, то за принципом коробок Діріхле існує два однакових слова $\beta_{i_1}...\beta_{i_n} = \beta_{j_1}...\beta_{j_n}$, які можна розкласти по-різному. Можна записати

$$\sum_{m=1}^{nL} \frac{\nu(n,m)}{2^m} \leqslant \sum_{m=1}^{nL} \frac{2^m}{2^m} = nL.$$

Отже, для кожного n виконується нерівність $\left(\sum_{i=1}^{r} 2^{-l_i} \right)^n \leqslant nL$, тобто $\sum_{i=1}^{r} 2^{-l_i} \leqslant \sqrt[n]{nL}$, звідки випливає

$$\sum_{i=1}^{r} 2^{-l_i} \leqslant \lim_{n\to\infty} \sqrt[n]{nL} = 1.$$

Зауваження. У теоремі 6.3 розглянуто нерівність Мак-Міллана для двійкового кодування. Для загального випадку q-кового кодування вона має вигляд

$$\sum_{i=1}^{r} \frac{1}{q^{l_i}} \leqslant 1, \text{ де } l_i = l(\beta_i).$$

Розглянемо ще один важливий факт.

ТЕОРЕМА 6.4. Якщо числа $l_1, ..., l_n$ задовольняють нерівність

$$\sum_{i=1}^{r} \frac{1}{2^{l_i}} \leqslant 1 \text{ (нерівність Мак-Міллана),}$$

то існує префіксна схема алфавітного кодування σ:

$$a_1 \to \beta_1,$$
$$\dots\ \dots\ \dots$$
$$a_r \to \beta_r,$$

де $l_1 = l(\beta_1), ..., l_r = l(\beta_r)$.

Доведення. Без утрати загальності можна вважати, що $l_1 \leqslant l_2 \leqslant ... \leqslant l_r$. Розіб'ємо множину $\{l_1, ..., l_r\}$ на класи еквівалентності так: l_i та l_j належать одному класу тоді й лише тоді, коли $l_i = l_j$. Нехай μ ($1 \leqslant \mu \leqslant r$) — кількість класів, а $\lambda_1, ..., \lambda_\mu$ та $\nu_1, ..., \nu_\mu$ позначають відповідно представників та потужності класів. Можна також уважати, що $\lambda_1 < \lambda_2 < ... < \lambda_\mu$.

Нерівність Мак-Міллана можна переписати у вигляді

$$\sum_{i=1}^{\mu} \frac{\nu_i}{2^{\lambda_i}} \leqslant 1.$$

Вона породжує серію допоміжних нерівностей:

$$\frac{\nu_1}{2^{\lambda_1}} \leqslant 1, \text{ звідки випливає } \nu_1 \leqslant 2^{\lambda_1},$$

$$\frac{\nu_1}{2^{\lambda_1}} + \frac{\nu_2}{2^{\lambda_2}} \leqslant 1, \text{ звідки випливає } \nu_2 \leqslant 2^{\lambda_2} - \nu_1 2^{\lambda_2 - \lambda_1},$$

$$\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots$$

$$\frac{\nu_1}{2^{\lambda_1}} + \frac{\nu_2}{2^{\lambda_2}} + \dots + \frac{\nu_\mu}{2^{\lambda_\mu}} \leqslant 1, \text{ звідки випливає}$$
$$\nu_\mu \leqslant 2^{\lambda_\mu} - \nu_1 2^{\lambda_\mu - \lambda_1} - \nu_2 2^{\lambda_\mu - \lambda_2} - \dots - \nu_{\mu-1} 2^{\lambda_\mu - \lambda_{\mu-1}}.$$

Розглянемо слова довжиною λ_1 у двійковому алфавіті B. Оскільки $\nu_1 \leqslant 2^{\lambda_1}$, то можна вибрати з них ν_1 різних слів $\beta_1, ..., \beta_{\nu_1}$ довжиною λ_1. Вилучимо з подальшого розгляду всі слова з множини B^*, які починаються зі слів $\beta_1, ..., \beta_{\nu_1}$. Потім розглянемо миожину слів у алфавіті B, що мають довжину λ_2 та не починаються зі слів $\beta_1, ..., \beta_{\nu_1}$. Таких слів, очевидно, $2^{\lambda_2} - \nu_1 2^{\lambda_2 - \lambda_1}$. Позаяк $\nu_2 \leqslant 2^{\lambda_2} - \nu_1 2^{\lambda_2 - \lambda_1}$, то з цієї множини можна вибрати ν_2 різних слів; позначимо їх як $\beta_{\nu_1+1}, ..., \beta_{\nu_1+\nu_2}$. Вилучимо з множини B^* слова, які починаються з $\beta_{\nu_1+1}, ..., \beta_{\nu_1+\nu_2}$, з подальшого розгляду. Продовжимо цей процес, поступово використовуючи допоміжні нерівності. Одержимо слова $\beta_1, ..., \beta_r$, де $r = \sum_{i=1}^{\mu} \nu_i$. Узявши їх як елементарні коди, отримаємо якусь схему алфавітного кодування σ. Вона, очевидно, префіксна. Крім того, $l(\beta_1) = l_1, ..., l(\beta_r) = l_r$.

Наслідок 1. Нерівність Мак-Міллана – необхідна й достатня умова існування алфавітного кодування з префіксною схемою та довжинами елементарних кодів, що дорівнюють $l_1, ..., l_r$.

Наслідок 2. Якщо існує роздільна схема алфавітного кодування із заданими довжинами елементарних кодів, то існує й префіксна схема з тими самими довжинами елементарних кодів.

Для доведення потрібно спочатку застосувати теорему 6.3, а потім – теорему 6.4.

6.3. Оптимальне кодування

Досі термін „код" було використано в загальноприйнятому розумінні (див. підрозділ 6.1). Однак у теорії кодування, а також у техніці слово *„код"* трактують також і як *множину елементарних кодів* [49]. Починаючи з цього місця, будемо використовувати слово „код" у двох значеннях. З контексту буде зрозуміло, яке саме з них ми маємо на увазі.

Нехай задано алфавіт $A = \{a_1, ..., a_r\}$ і ймовірності появи букв у повідомленні $P = (p_1, ..., p_r)$; тут p_i – імовірність появи букви a_i. Не втрачаючи загальності, можна вважати, що

$$p_1 \geqslant p_2 \geqslant ... \geqslant p_r > 0,$$

тобто можна одразу вилучити букви, які не можуть з'явитись у повідомленні, і впорядкувати букви за спаданням імовірностей їх появи. Крім того, $p_1 + p_2 + ... + p_r = 1$.

Для кожної роздільної схеми алфавітного кодування σ математичне сподівання коефіцієнта збільшення довжини повідомлення в разі кодування (позначають $l^{\sigma}_{\text{сер}}$) визначають так:

$$l^{\sigma}_{\text{сер}}(P) = \sum_{i=1}^{r} p_i l_i,$$

де $l_i = l(\beta_i)$, $i = 1, ..., r$. i називають *середньою довжиною* кодування за схемою σ для розподілу ймовірностей P.

Уведемо величину $l_* = \inf\limits_{\sigma} l^{\sigma}_{\text{сер}}$, де нижню грань беруть за всіма роздільними схемами σ. Легко довести, що

$$1 \leqslant l_* \leqslant \lceil \log_2 r \rceil,$$

де верхню оцінку дає схема з елементарними кодами з однаковою довжиною $\lceil \log_2 r \rceil$. Звідси випливає, що для побудови кодів, у яких величина $l_{\text{сер}}$ близька до l_*, можна не враховувати коди з більшим $l_{\text{сер}}$, ніж $\lceil \log_2 r \rceil$. Отже, будемо розглядати лише схеми з $p_i l_i \leqslant \lceil \log_2 r \rceil$. Позначимо $p_* = \min(p_1, ..., p_r)$, тоді

$$l_i \leqslant \lceil \log_2 r \rceil / p_*$$

для всіх $i = 1, 2, ..., r$.

Звідси випливає, що є лише скінченна кількість варіантів значень $l_{\text{сер}}$, для яких $l_* \leqslant l_{\text{сер}} \leqslant \lceil \log_2 r \rceil$. Отже, значення l_* досягається на якійсь схемі σ; його можна визначити як

$$l_* = \min_{\sigma} l_{\text{сер}}^{\sigma}.$$

Коди, визначені схемою σ з $l_{\text{сер}} = l_*$, називають *кодами з мінімальною надлишковістю* (або *оптимальними кодами*) для розподілу ймовірностей P. За наслідком 2 з теорем 6.3 та 6.4 існує алфавітне кодування з префіксною схемою, яке дає оптимальні коди. У зв'язку з цим, будуючи оптимальні коди, можна обмежитися префіксними схемами.

Проста ідея побудови коду, близького до оптимального, належить американському математикові Р. Фано (R. Fano). Сформулюємо алгоритм кодування за методом Фано.

1. Упорядковуємо букви алфавіту A за спаданням імовірностей їх появи в повідомленні.
2. Розбиваємо множину букв, записаних у зазначеному порядку, на дві послідовні (без перестановок букв) частини так, щоб сумарні ймовірності кожної з них були якомога близькими одна до одної. Кожній букві з першої частини приписуємо символ 0, другої — символ 1. Далі те саме робимо з кожною частиною, якщо вона містить принаймні дві букви. Процедуру продовжуємо доти, доки всю множину не буде розбито на окремі букви.

Приклад 6.3. Нехай задано розподіл імовірностей $P = (0.4, 0.15, 0.15, 0.15, 0.15)$. Побудуємо код за методом Фано. Розв'язок подано в табл. 6.1.

Таблиця 6.1

Буква алфавіту A	Імовірність появи букви	Розбиття множини букв			Елементарний код
a_1	0.4	0	0		00
a_2	0.15		1		01
a_3	0.15	1	0		10
a_4	0.15		1	0	110
a_5	0.15			1	111

Середня довжина коду дорівнює

$$l_{\text{сер}}^{F} = 0.4 \cdot 2 + 0.15 \cdot 2 + 0.15 \cdot 2 + 0.15 \cdot 3 + 0.15 \cdot 3 = 2.3;$$

тут верхній індекс F означає, що код отримано методом Фано.

Алгоритм Фано має просту інтерпретацію за допомогою бінарного дерева. Від кореня відходять два ребра, одне з яких позначено символом 0, друге — символом 1. Ці два ребра відповідають розбиттю множини всіх букв на дві майже рівноймовірні частини, одній з яких поставлено у відповідність символ 0, а другій — символ 1. Ребра, що виходять із вершин наступного рівня, відповідають розбиттю одержаних частин знову на дві майже рівноймовірні послідовні частини.

Цей процес продовжують доти, доки множину букв не буде розбито на окремі букви. Кожний листок дерева відповідає якомусь елементарному коду. Щоб виписати цей код, потрібно пройти шлях від кореня до відповідного листка.

Зазначимо, що незалежно від способу кодування кожному бінарному дереву відповідає набір двійкових елементарних кодів. У такому разі дерево називають *кодовим*. Якщо елементарні коди відповідають листкам кодового дерева, то відповідна схема алфавітного кодування є префіксною (отже, забезпечується однозначність декодування).

Приклад 6.4. На рис. 6.1. зображено кодове дерево, отримане методом Фано для розподілу ймовірностей із прикладу 6.3.

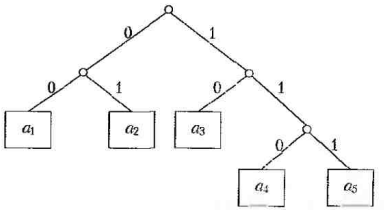


Рис. 6.1

Що ймовірніша поява букви, то швидше вона утворить „самостійну" підмножину, й, отже, її буде закодовано коротшим елементарним кодом. Тому метод Фано доволі ефективний. Але чи завжди він дає змогу отримати оптимальний код? Виявляється, що ні. Поданий тут спосіб побудови оптимального коду потребує тонших міркувань. Передусім сформулюємо й доведемо деякі допоміжні твердження.

ЛЕМА 6.1. В оптимальному коді букву з меншою ймовірністю її появи не може бути закодовано коротшим словом. Інакше кажучи, для оптимального коду з того, що $p_i < p_j$, випливає, що $l_i \geqslant l_j$.

Доведення. Припустимо протилежне: нехай є дві букви a_i й a_j такі, що $p_i < p_j$ і $l_i < l_j$. Поміняємо місцями β_i та β_j у схемі кодування. Тоді середня довжина елементарних кодів зміниться на

$$p_i l_j + p_j l_i - p_i l_i - p_j l_j = (p_i - p_j) l_j - (p_i - p_j) l_i = (p_i - p_j)(l_j - l_i) < 0,$$

тобто зменшиться, що суперечить оптимальності коду.

ЛЕМА 6.2. Якщо код оптимальний, то можна так перенумерувати букви алфавіту $A = \{a_1, ..., a_r\}$ і відповідні елементарні коди $\beta_1, \beta_2, ..., \beta_r$, що $p_1 \geqslant p_2 \geqslant ... \geqslant p_r$, і при цьому $l_1 \leqslant l_2 \leqslant ... \leqslant l_r$.

Доведення. Якщо $p_i > p_{i+1}$, то з леми 6.1 випливає, що $l_i \leqslant l_{i+1}$. Якщо ж $p_i = p_{i+1}$ але $l_i > l_{i+1}$, то поміняємо місцями нумерацію букв a_i, a_{i+1} і відповідних елементарних кодів. Повторюючи цю процедуру потрібну кількість разів, одержимо бажану нумерацію.

З нерівності $l_1 \leqslant l_2 \leqslant \ldots \leqslant l_r$ випливає, що букву a_r (найменш імовірну) закодовано словом β_r із найбільшою довжиною.

ЛЕМА 6.3. В оптимальному коді є два елементарні коди з найбільшою довжиною l_r, які відрізняються лише останніми символами.

Доведення. Припустимо, що це не так. Тоді можна відкинути останній символ елементарного коду β_r, не порушуючи властивості префіксності. При цьому, очевидно, зменшиться середня довжина елементарного коду. Це суперечить твердженню, що код оптимальний.

ТЕОРЕМА 6.5. Існує такий оптимальний код, у якому елементарні коди двох найменш імовірних букв a_{r-1} і a_r відрізняються лише останніми символами.

Доведення. За лемою 6.3 знайдеться елементарний код β_r, який має ту саму довжину, що й β_r, і відрізняється від нього лише останнім символом. Із лем 6.1 і 6.2 випливає, що $l_r = l_{r-1} = \ldots = l_r$. Якщо $i \neq r-1$, то можна поміняти місцями β_i та β_{r-1}, не порушуючи нерівності $l_1 \leqslant l_2 \leqslant \ldots \leqslant l_r$.

Теорема 6.5 дає змогу розглядати лише такі схеми алфавітного кодування, у яких елементарні коди β_{r-1} і β_r (для двох найменш імовірних букв a_{r-1} і a_r) мають найбільшу довжину й відрізняються тільки останніми символами. Це означає, що листки β_{r-1} і β_r кодового дерева оптимального коду мають бути з'єднані в одній вершині попереднього рівня (рис. 6.2).

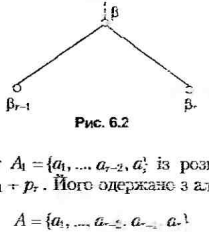


Рис. 6.2

Розглянемо новий алфавіт $A_1 = \{a_1, \ldots, a_{r-2}, a_r'\}$ із розподілом імовірностей $P_1 = (p_1, \ldots, p_{r-2}, p)$, де $p = p_{r-1} + p_r$. Його одержано з алфавіту

$$A = \{a_1, \ldots, a_{r-2}, a_{r-1}, a_r\}$$

об'єднанням двох найменш імовірних букв a_{r-1} та a_r в одну букву a з імовірністю $p = p_{r-1} + p_r$. Говорять, що A_1 отримано *стисненням* алфавіту A.

Нехай для алфавіту A_1 побудовано схему σ_1 з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta$. Інакше кажучи, є якесь кодове дерево з листками $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta$. Схемі σ_1 можна поставити у відповідність схему σ з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{r-2}, \beta_{r-1}, \beta_r$ для початкового алфавіту A, узявши $\beta_{r-1} = \beta 0$, $\beta_r = \beta 1$ (тобто елементарні коди β_{r-1} та β_r одержують з елементарного коду β приписуванням справа відповідно 0 та 1).

Процедуру переходу від σ_1 до σ називають *розщепленням*.

ТЕОРЕМА 6.6. Якщо схема σ_1 оптимальна для алфавіту A_1, то схема σ оптимальна для алфавіту A.

Доведення. Легко переконатись, що середні довжини $l^{\sigma}_{сер}$ та $l^{\sigma_1}_{сер}$ елементарних кодів у схемах σ та σ_1 пов'язані співвідношенням $l^{\sigma}_{сер} = l^{\sigma_1}_{сер} + p$.

Припустимо, що схема σ не оптимальна для алфавіту A, тобто існує схема Σ, для якої $l^{\Sigma}_{сер} < l^{\sigma}_{сер}$. Нехай її елементарні коди — β'_1, β'_2, ..., β'_{r-2}, β'_{r-1}, β'_r. Можна вважати, що листки β'_{r-1} і β'_r кодового дерева схеми Σ відповідають двом найменш імовірним буквам алфавіту A й відрізняються лише останніми символами.

Розглянемо схему Σ_1 для алфавіту A_1. Елементарні коди схеми Σ_1 — β'_1, β'_2, ..., β'_{r-2}, β'. Елементарний код β' одержують відкиданням останнього символу від β'_r (або від β'_{r-1}, результат буде тим самим). Середні довжини кодів $l^{\Sigma}_{сер}$ й $l^{\Sigma_1}_{сер}$ пов'язані співвідношенням $l^{\Sigma}_{сер} = l^{\Sigma_1}_{сер} + p$.

Зі співвідношень $l^{\sigma}_{сер} = l^{\sigma_1}_{сер} + p$, $l^{\Sigma}_{сер} = l^{\Sigma_1}_{сер} + p$ та $l^{\Sigma}_{сер} < l^{\sigma}_{сер}$ випливає, що $l^{\Sigma_1}_{сер} < l^{\sigma_1}_{сер}$. Це суперечить оптимальності схеми σ_1 для алфавіту A_1.

Із теореми 6.6 випливає такий метод побудови оптимальної схеми алфавітного кодування. Спочатку послідовно стискають алфавіт A до отримання алфавіту з двох букв, оптимальна схема кодування для якого очевидна: першу букву кодують символом 0, другу — символом 1. Потім послідовно розщеплюють одержану схему. Очевидно, що отримана в результаті схема префіксна.

Описаний метод кодування запропоновано 1952 р. американським математиком Д. Хаффманом (D. Huffman).

Приклад 6.5. Нехай задано розподіл імовірностей P=(0.4, 0.15, 0.15, 0.15, 0.15), як і в прикладі 6.3. Побудуємо код методом Хаффмана. Розв'язок наведено в табл. 6.2.

Таблиця 6.2

Букви алфавіту A	Імовірність та позначення елементарних кодів							
	Початковий алфавіт A		Стиснуті алфавіти					
			A_1		A_2		A_3	
a_1	0.4	1	0.4	1	0.4	1	0.6	0
a_2	0.15	010	0.3	00	0.3	00	0.4	1
a_3	0.15	011	0.15	010	0.3	01		
a_4	0.15	000	0.15	011				
a_5	0.15	001						

Кожний з алфавітів A_1 — A_3 одержують стисненням попереднього алфавіту. Алфавіт A_3 складається з двох букв, тому оптимальна схема містить лише два елементарні коди — символи 0 і 1. Послідовне розщеплення дає оптимальну схему для початкового алфавіту A (у процесі розщеплення потрібно рухатися проти стрілок).

Середня довжина побудованого коду, яка дорівнює $l^{H}_{сер} = 0.4 \cdot 1 + 4 \cdot 0.15 \cdot 3 = 2.2$, як це випливає з попереднього, мінімально можлива для даного розподілу імовірностей P. Тут індекс H означає, що код отримано методом Хаффмана.

У прикладі 6.3 одержано код методом Фано із середньою довжиною $l^{F}_{сер} = 2.3$. Отже, метод кодування Фано не завжди дає оптимальний код.

За допомогою алгоритму Хаффмана можна безпосередньо побудувати кодове дерево оптимального коду, яке називають *деревом Хаффмана* [50]. Бінарне дерево, що відповідає оптимальному коду, будують знизу вверх, починаючи з $|A| = r$ листків, і виконують $r-1$ злиття. Злиття полягає в побудові нового дерева з двох наявних дерев (або листків) із найменшими ймовірностями. При цьому листок або дерево з більшою ймовірністю утворює ліве піддерево; сума ймовірностей лівого та правого піддерев дорівнює ймовірності отриманого дерева (її записують у корінь). На ребро до лівого піддерева помищають 0, до правого — 1. Розподіл імовірностей перед кожним злиттям упорядковують за спаданням.

Алгоритм Хаффмана належить до жадібних алгоритмів [23].

Приклад 6.6. Побудуємо дерево Хаффмана для розподілу ймовірностей $P = (0.34, 0.18, 0.17, 0.16, 0.15)$. Оскільки є всього п'ять букв, то для побудови дерева потрібно зробити чотири злиття. На кожному кроці зливають два піддерева з найменшими ймовірностями. Одержують нове дерево із сумарною ймовірністю лівого та правого піддерев і впорядковують розподіл імовірностей за спаданням. Листки зображають прямокутниками, у яких букви та їх імовірності відокремлюють двокрапкою (рис. 6.3). Внутрішні вершини зображають кружечками, у яких зазначають суми ймовірностей їхніх синів. Дивиміку роботи алгоритму відображено на рис. 6.4.

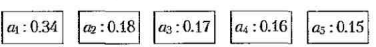


Рис. 6.3

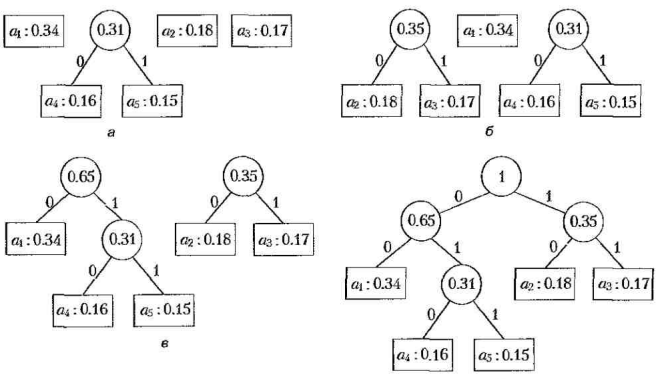


Рис. 6.4

Щоб отримати елементарний код для певної букви, потрібно пройти єдиний шлях від кореня до відповідного листка, записуючи послідовність бітів. Одержимо таку оптимальну схему алфавітного кодування за заданого розподілу ймовірностей: $a_1 \to 00$, $a_2 \to 10$, $a_3 \to 11$, $a_4 \to 010$, $a_5 \to 011$.

Узагалі кажучи, оптимальна схема не єдина. Так, у табл. 6.3 для розподілу ймовірностей $P = (0.4, 0.2, 0.2, 0.1, 0.1)$ наведено три різні оптимальні схеми ($l^1_{сер} = l^2_{сер} = l^3_{сер} = l^* = 2.2$).

Таблиця 6.3

Буква алфавіту	Імовірність	Схема 1	Схема 2	Схема 3
a_1	0.4	11	1	1
a_2	0.2	10	011	01
a_3	0.2	01	010	001
a_4	0.1	001	001	0001
a_5	0.1	000	000	0000

6.4. Коди, стійкі до перешкод. Коди Хеммінга

Розглянемо один частковий випадок рівномірного двійкового кодування, коли $A = B = \{0, 1\}$ [27]. Розглянемо схему рівномірного кодування $\sigma_{k,n}$ із параметрами k, n:

$$\alpha_1 \to \beta_1,$$
$$\alpha_2 \to \beta_2,$$
$$\alpha_3 \to \beta_3,$$
$$\ldots \ldots \ldots$$
$$\alpha_{2^k} \to \beta_{2^k},$$

де α_i, β_i — відповідно слова довжиною k та n, $n > k$. Говорять, що *схему* $\sigma_{k,n}$ *задає код* $V = \{\beta_1, \beta_2, \beta_3, \ldots, \beta_{2^k}\} \subset E_2^n$ (див. підрозділ 6.3). Слова в алфавіті $\{0, 1\}$ – це впорядковані набори з нулів і одиниць (їх називають також двійковими векторами). Зазначимо, що кількість усіх слів α_i довжиною k в алфавіті $\{0, 1\}$ дорівнює 2^k.

Із методичних міркувань у цьому підрозділі нам зручно використовувати такі позначення. Елсменти множини E_2^n (двійкові вектори довжиною n) позначатимемо великими латинськими буквами X, Y, Z, ..., а їх компоненти — відповідними малими буквами з індексами. Зокрема, елементарні коди позначатимсмо як традиційно (β_1, β_2, β_3, ...), так і X, Y, Z, ..., залежно від контексту.

Нормою $\|X\|$ двійкового вектора $X = x_1 x_2 \ldots x_n$ називають число, яке дорівнює кількості його одиничних компонент. Отже,

$$\|X\| = \sum_{i=1}^{n} x_i.$$

Нагадаємо, що операцію $\oplus$ — „альтернативне або" над двійковими розрядами (бітами) — означують так: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$ (див. підрозділ 1.1). Цю операцію називають також *додаванням за модулем 2* (mod 2). Якщо X та Y – двійкові вектори, то $X \oplus Y$ – *порозрядне* додавання за модулем 2.

Припустимо, що в каналі зв'язку діє *джерело адитивних перешкод*, яке описують множиною $P(n, t)$. Її елементи — двійкові вектори-помилки $x_1 x_2 \ldots x_n$, у яких

норма будь-якого фрагмента $x_i x_{i+1} \dots x_{i+l-1}$ не більша ніж t, якщо довжина фрагмента $l \leqslant n$, (тобто на n переданих поспіль двійкових символів припадає не більше ніж t помилок). Це означає, що коли на вході каналу зв'язку передано повідомлення α, то на виході можна отримати будь-яке слово з множини $\{\alpha \oplus \gamma | \gamma \in P(n, t), l(\alpha) = l(\gamma)\}$, де $\alpha \oplus \gamma$ – порозрядне додавання за mod 2.

Позаяк проблема локалізації інформації (тобто розділення закодованого повідомлення на елементарні коди) у моделі рівномірного кодування тривіальна, то виявлення помилок полягає у відшуканні незбігу локалізованої групи n символів ні з яким елементарним кодом. Якщо через помилку елементарний код перейде в інший елементарний код, то помилку не буде виявлено. Іноді можна виправити помилку. Якщо групу бітів локалізовано правильно, то для цього необхідно й достатньо, щоб помилкова група була „синонімом" єдиного елементарного коду.

Канал зв'язку називають *надійним*, якщо будь-які помилки можна виявити чи виправити відповідно до заданої мети декодування. Далі наведено головні положення побудови кодів, які забезпечують надійність найпростіших каналів зв'язку.

Віддалю Хеммінга називають функцію $\rho(X, Y)$ двох змінних, означену на множині E_2^n:

$$\rho(X, Y) = \sum_{i=1}^{n} (x_i \oplus y_i)$$

(вона дорівнює кількості розрядів, у яких вектори X та Y не збігаються).

Скалярний добуток векторів $X, Y \in E_2^n$ означають так:

$$\langle X, Y \rangle = \sum_{i=1}^{n} x_i y_i;$$

він дорівнює кількості розрядів, у яких X та Y збігаються й дорівнюють 1.

Легко перевірити такі співвідношення:

$$\rho(X, 0) = \|X\| = \sum_{i=1}^{n} x_i, \tag{6.1}$$

де 0 – n-вимірний вектор із нульовими компонентами;

$$\rho(X, Y) = \|X \oplus Y\|, \tag{6.2}$$

де $X \oplus Y$ – порозрядне додавання за mod 2;

$$\rho(X \oplus Z, Y \oplus Z) = \rho(X, Y), \tag{6.3}$$

$$\rho(X, Y) = \|X\| + \|Y\| - 2\langle X, Y \rangle. \tag{6.4}$$

Для віддалі Хеммінга виконуються *аксіоми метрики*:

- $\rho(X, Y) \geqslant 0$, причому $\rho(X, Y) = 0$ в тому й лише в тому разі, якщо $X = Y$;
- $\rho(X, Y) = \rho(Y, X)$;
- $\rho(X, Y) + \rho(Y, Z) \geqslant \rho(X, Z)$ (нерівність трикутника).

Метрика Хеммінга — зручне математичне поняття для формулювання умов надійності кодування в разі адитивних помилок. Нехай схему $\sigma_{k,n}$ задано кодом $V = \{\beta_1, \beta_2, \beta_3, ..., \beta_{2^k}\}$. *Кодовою віддаллю* для коду V називають величину

$$\rho(V) = \min\{\rho(X, Y) \mid X, Y \in V, X \neq Y\}.$$

ТЕОРЕМА 6.7. Якщо в каналі зв'язку діє джерело адитивних перешкод $P(n, t)$, то правдиві такі твердження.

1. Для виявлення будь-яких помилок необхідно й достатньо, щоб $\rho(V) > t$.
2. Для виправлення будь-яких помилок необхідно й достатньо, щоб $\rho(V) > 2t$.

Зауваження. Інакше кажучи, код може виявляти будь-які комбінації з t й меншої кількості помилок тоді й лише тоді, коли його кодова віддаль більша ніж t; код може виправляти будь-які комбінації з t й меншої кількості помилок тоді й лише тоді, коли його кодова віддаль більша ніж $2t$.

Доведення. 1. Нехай $\rho(V) > t$. Якщо $X \in V$, $Y \in P(n, t)$, $Y \neq 0$, то, використовуючи спочатку рівність (6.3), а потім — (6.1), можемо записати $\rho(X, X \oplus Y) = \rho(0, Y) = \|Y\| \leqslant t$. Отже, $X \oplus Y \notin V$, і помилку виявлено.

Навпаки, нехай $\rho(X, Y) \leqslant t$ й $X, Y \in V$, $X \neq Y$. Тоді, застосувавши рівність (6.2), маємо $\|X \oplus Y\| = \rho(X, Y) \leqslant t$; отже, $Z = X \oplus Y \in P(n, t)$. Звідси випливає, що $X \oplus Z = Y$, тобто помилку в елементарному коді Y виявити не можна.

2. Нехай $\rho(V) > 2t$. Якщо $X \in V$, $Z \in P(n, t)$, то X — єдиний елементарний код із V, який міг перейти внаслідок помилки в $X \oplus Z$. Справді, припустимо, що існує такий двійковий вектор $Y \neq X$, що $Y \in V$ та $Y \oplus Z_1 = X \oplus Z$ для якогось $Z_1 \in P(n, t)$. Додавши до обох частин останньої рівності $X \oplus Z_1$, отримаємо $X \oplus Y = Z_1 \oplus Z$. Але, згідно з рівністю (6.2), можемо записати $\|X \oplus Y\| = \rho(X, Y) > 2t$, а $\|Z_1 \oplus Z\| \leqslant \|Z_1\| + \|Z\| \leqslant 2t$. Одержали суперечність.

Навпаки, нехай $\rho(X, Y) \leqslant 2t$ для якихось різних $X, Y \in V$. Тоді $\|X \oplus Y\| \leqslant 2t$, й існують такі двійкові вектори Z_1, Z_2, що $\|Z_1\| \leqslant t$, $\|Z_2\| \leqslant t$ (тобто вони належать $P(n, t)$) й $X \oplus Y = Z_1 \oplus Z_2$. Додавши до обох частин останньої рівності $Y \oplus Z_1$, одержимо $X \oplus Z_1 = Y \oplus Z_2 = W$. Отже, у разі отримання спотвореного елементарного коду W неможливо виявити, що було передано насправді: X чи Y.

Доведене твердження має геометричну інтерпретацію.

Множину $S_t(X) = \{Z \mid \rho(X, Z) \leqslant t\}$ називають *кулею радіусом t з центром у точці X*.

ТЕОРЕМА 6.8. Якщо в каналі зв'язку діє джерело адитивних перешкод $P(n, t)$, то правдиві такі твердження.

1. Для виявлення будь-яких помилок необхідно й достатньо, щоб для будь-якого $X \in V$ куля $S_t(X)$ не містила інших елементарних кодів, окрім X.
2. Для виправлення будь-яких помилок необхідно й достатньо, щоб для будь-яких X, $Y \in V$ було виконано умову $S_t(X) \cap S_t(Y) = \varnothing$.

Рівномірне кодування $\sigma_{k,n}$: $\alpha_i \to \beta_i$ ($i = 1, 2, 3, ..., 2^k$) називають *систематичним*, якщо можна виділити множину k розрядів $I = \{i_1, ..., i_k\} \subset \{1, 2, ..., n\}$, які називають *інформаційними*, так, що коли $\beta_i = x_1 ... x_n$ ($i = 1, 2, 3, ..., 2^k$), то $\alpha_i = x_{i_1} ... x_{i_k}$. Решту розрядів у такому разі називають *контрольними*.

Рівномірне кодування $\sigma_{k,n}$: $\alpha_i \to \beta_i$ ($i = 1, 2, 3, ..., 2^k$) називають *лінійним*, або *груповим*, якщо код $V = \{\beta_1, \beta_2, \beta_3, ..., \beta_{2^k}\}$ утворює підгрупу E_2^n щодо операції $\oplus$ порозрядного додавання векторів за модулем 2. У такому разі код V водночас являє собою лінійний підпростір простору E_2^n над полем E_2. Усі лінійні коди систематичні.

Лінійні коди можна задавати простіше, ніж коди загального типу. Достатньо зазначити твірні групи V. Їх можна подати матрицею з k рядками та n стовпцями $G(V)$ — базисом векторного простору V; $G(V)$ називають *породжувальною матрицею коду* V. Базис можна вибрати не єдиним способом, тому матрицю $G(V)$ означено неоднозначно. За допомогою породжувальної матриці $G = G(V)$ можна кодувати повідомлення. Якщо $\alpha_i = x_{i_1} ... x_{i_k}$ — повідомлення, то його код $\beta_i = \alpha_i G$.

Двоїстість, яка пов'язує ортогональні підпростори, дає ще один спосіб подання лінійних кодів. Вектори $X, Y \in E_2^n$ називають *ортогональними*, якщо $(X, Y) = 0 (\mathrm{mod}\ 2)$. Ортогональний підпростір $V^\perp = \{X \mid (X, V) = 0 (\mathrm{mod}\ 2)\}$ для лінійного коду V називають *двоїстим кодом* для V. Його розмірність дорівнює $n - k$, і код $V^\perp$ задає *двоїсту схему* алфавітного кодування $\sigma_{n-k,n}$. Матрицю $H(V) = G(V^\perp)$ — породжувальну матрицю коду $V^\perp$ — називають *перевірною матрицею* коду V. Отже, маємо $V = \{X \mid (X, H) = 0 (\mathrm{mod}\ 2)\}$ — *двоїстий спосіб подання лінійного коду V перевірною матрицею* $H = H(V)$. Легко перевірити, що коли $G(V) = [J_k A]$, де J_k — одинична $k \times k$ матриця, то $H(V) = [A^T J_{n-k}]$, де A^T — транспонована матриця. Звідси випливає, що коли I — множина інформаційних розрядів для коду V, то як множину інформаційних розрядів для двоїстого коду $V^\perp$ можна взяти $\bar{I} = \{1, 2, ..., n\} \setminus I$.

Приклад 6.7. Для лінійного коду $V = \{000, 011, 101, 110\}$, який задає схему лінійного рівномірного кодування $\sigma_{2,3}$

$$00 \to 000,\ 01 \to 011,\ 10 \to 101,\ 11 \to 110$$

з інформаційними розрядами $I = \{1, 2\}$, маємо

$$G(V) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},\ H(V) = [1\ \ 1\ \ 1],\ V^\perp = \{000, 111\}$$

і двоїсту схему $\sigma_{1,3}$:

$$0 \to 000,\ 1 \to 111.$$

ТЕОРЕМА 6.9. Для кодової віддалі лінійного коду виконується рівність

$$\rho(V) = \min\{\|X\| \mid X \in V, X \neq 0\}. \qquad (6.5)$$

Доведення. Зазначимо, що нульовий вектор 0 міститься в будь-якому лінійному коді. Рівність (6.5) випливає з того, що

$$\min\{\rho(X, Y) \mid X, Y \in V, X \neq Y\} = \min\{\rho(0, X \oplus Y) \mid X \oplus Y \in V, X \oplus Y \neq 0\}.$$

Р. Хеммінг (R. Hamming) 1950 р. запропонував коди для виявлення та виправлення помилок у разі $t = 1$. Коди Хеммінга лінійні; вони мають найменшу надлишковість, можливу для даного k.

Коди Хеммінга $H_{\text{вияв}}(n)$ для виявлення помилок у каналі зв'язку із джерелом перешкод $P(n,1)$ означено для будь-якого n:

$$H_{\text{вияв}}(n) = \{X \mid X \in E_2^n, \|X\| = 0(\mathrm{mod}\,2)\}.$$

Код $H_{\text{вияв}}(n)$ лінійний. Справді, якщо X, $Y \in H_{\text{вияв}}(n)$, то з урахуванням рівностей (6.2) та (6.4) можемо записати

$$\|X \oplus Y\| = \|X\| + \|Y\| - 2\langle X, Y\rangle = -2\langle X, Y\rangle = 0(\mathrm{mod}\,2),$$

тобто $X \oplus Y \in H_{\text{вияв}}(n)$. Згідно зі співвідношенням (6.5), $\rho(H_{\text{вияв}}(n)) = 2$; отже, можна виявити будь-яку помилку в каналі з джерелом перешкод $P(n,1)$. Для коду $H_{\text{вияв}}(n)$ як інформаційні можна взяти будь-які $n-1$ розрядів, бо значення довільного розряду слова $x_1x_2 \ldots x_n \in H_{\text{вияв}}(n)$ можна однозначно знайти за значеннями інших розрядів із рівняння $x_1 \oplus x_2 \oplus \ldots \oplus x_n = 0$. Маємо

$$H(H_{\text{вияв}}(n)) = [1 \quad 1 \quad \ldots \quad 1], \quad H^{\perp}_{\text{вияв}}(n) = \{00\ldots0,\ 11\ldots1\}.$$

Як одну з породжувальних матриць можна взяти

$$G(H_{\text{вияв}}(n)) = \begin{bmatrix} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & 1 \end{bmatrix}.$$

Вибравши $I = \{1, 2, \ldots, n-1\}$, отримаємо відповідну схему рівномірного кодування $\sigma_{n-1,n}$:

$$x_1x_2 \ldots x_{n-1} \to x_1x_2 \ldots x_{n-1}x_n,$$

де $x_n = x_1 \oplus x_2 \oplus \ldots \oplus x_{n-1}$. Надлишковість у разі використання коду $H_{\text{вияв}}(n)$ становить $R = (n-1)^{-1}$.

У цьому коді Хеммінга втілено ідею перевірки на парність.

Коди Хеммінга для виправлення помилок у каналі зв'язку із джерелом перешкод $P(n,1)$ будують для значень $n = 2^s - 1$ ($s = 2, 3, \ldots$). Код $H_{\text{вищ}}(n)$ зручно задавати перевірною матрицею, яка має s рядків і $2^s - 1$ стовпців. Тут стовпці — усі можливі ненульові двійкові набори довжиною s. Їх зручно розміщувати так, щоб i-й зліва стовпець h_i був двійковим розкладом числа i (старші розряди — зверху):

$$H(H_{\text{вищ}}(n)) = \begin{bmatrix} 0 & 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & 0 & \ldots & 1 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 1 & 1 & \ldots & 1 & 1 \\ 1 & 0 & 1 & \ldots & 0 & 1 \end{bmatrix}.$$

Таке розміщення стовпців перевірної матриці зумовлене вибором як контрольних розрядів тих, у яких номери являють собою степені двійки: $T=\{1, 2, 4, 8, 16, ..., 2^{s-1}\}$.

Приклад 6.8. Для значень $n=3$, $n=7$ маємо такі перевірні матриці коду $H_{\text{вип}}(n)$:

$$H(H_{\text{вип}}(3))=\begin{bmatrix}0 & 1 & 1\\ 1 & 0 & 1\end{bmatrix};$$

$$H(H_{\text{вип}}(7))=\begin{bmatrix}0 & 0 & 0 & 1 & 1 & 1 & 1\\ 0 & 1 & 1 & 0 & 0 & 1 & 1\\ 1 & 0 & 1 & 0 & 1 & 0 & 1\end{bmatrix}.$$

За допомогою перевірної матриці $H=H(H_{\text{вип}}(n))$ код Хеммінга $H_{\text{вип}}(n)$ можна задати так:

$$H_{\text{вип}}(n)=\{X \mid \langle X,H\rangle = 0 (\text{mod}\, 2),\ X\in E_2^n\}=$$

$$=\left\{X \;\middle|\; x_1\begin{pmatrix}h_{11}\\ h_{21}\\ \dots\\ h_{s1}\end{pmatrix}\oplus x_2\begin{pmatrix}h_{12}\\ h_{22}\\ \dots\\ h_{s2}\end{pmatrix}\oplus\dots\oplus x_n\begin{pmatrix}h_{1n}\\ h_{2n}\\ \dots\\ h_{sn}\end{pmatrix}=\begin{pmatrix}0\\ 0\\ \dots\\ 0\end{pmatrix}\right\}. \qquad (6.6)$$

Код Хеммінга $H_{\text{вип}}(n)$ у разі $n=7$ називають (7, 4)-кодом Хеммінга, а в разі $n=15$ – (15, 11)-кодом Хеммінга. Тут перше число в круглих дужках дорівнює довжині елементарного коду, а друге – кількості інформаційних розрядів.

Приклад 6.9. Закодуємо повідомлення *1001* за допомогою (7, 4)-коду Хеммінга. Запишемо макет коду, беручи до уваги розміщення контрольних розрядів: $x_1x_2\text{1}x_4\text{001}$. Для відшукання значень контрольних розрядів використаємо умову (6.6) (доданки з нульовими коефіцієнтами випущено):

$$x_1\begin{pmatrix}0\\ 0\\ 1\end{pmatrix}\oplus x_2\begin{pmatrix}0\\ 1\\ 0\end{pmatrix}\oplus\begin{pmatrix}0\\ 1\\ 1\end{pmatrix}\oplus x_4\begin{pmatrix}1\\ 0\\ 0\end{pmatrix}\oplus\begin{pmatrix}1\\ 1\\ 1\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix},$$

звідки $x_4\oplus 1=0$, $x_2\oplus 1\oplus 1=0$, $x_1\oplus 1\oplus 1=0$. Отже, $x_1=0$, $x_2=0$, $x_4=1$, й отримаємо такий код заданого повідомлення: 0011001.

Доведемо, що $\rho(H_{\text{вип}}(n))\geqslant 3$, тобто код $H_{\text{вип}}(n)$ забезпечує корекцію будь-яких помилок у каналі зв'язку із джерелом перешкод $P(n,1)$. Справді, якщо $X\neq 0$ та $X\in H_{\text{вип}}(n)$, то $\|X\|\neq 1$, оскільки всі стовпці перевірної матриці ненульові. Далі, $=\|X\|\neq 2$, бо якщо X має *одиниці тільки у двох розрядах, скажімо* $x_i=x_j=1$, $i\neq j$, то $h_i\oplus h_j=0$ (тут h_i й h_j – стовпці перевірної матриці H), що рівносильно рівності $h_i=h_j$, а всі стовпці перевірної матриці різні. Отже, найменша вага ненульового вектора з $H_{\text{вип}}(n)$ не менша 3 і, відповідно до співвідношення (6.5), $\rho(H_{\text{вип}}(n))\geqslant 3$.

Нехай в елементарному коді $X \in H_{вип}(n)$ виникла помилка в i-му розряді: код $X = x_1x_2 \ldots x_i \ldots x_n$ перетворився на $X' = X \oplus e_i = x_1x_2 \ldots \bar{x}_i \oplus 1 \ldots x_n$ (тут e_i — вектор, i-та компонента якого дорівнює 1, а решта компонент — 0). Тоді

$$\langle X', H\rangle = \langle X \oplus e_i, H\rangle = \langle X, H\rangle \oplus h_i = h_i \pmod 2,$$

бо $\langle X, H\rangle = 0 \pmod 2$. Звідси випливає, що коли локалізовано послідовність символів $X \in E_2^n$, то достатньо визначити вектор $\langle X, H\rangle \pmod 2$. Якщо він нульовий, то помилки немає, а ні, то цей вектор являє собою двійковий розклад номера розряду, у якому виникла помилка.

Приклад 6.10. Нехай під час декодування повідомлення в (7, 4)-коді Хеммінга локалізовано послідовність $X = 0110010$. Знаходимо

$$\langle X, H\rangle = \begin{pmatrix}0\\1\\0\end{pmatrix} \oplus \begin{pmatrix}0\\1\\1\end{pmatrix} \oplus \begin{pmatrix}1\\1\\0\end{pmatrix} = \begin{pmatrix}1\\1\\1\end{pmatrix}.$$

Доходимо висновку, що виникла помилка в сьомому розряді елементарного коду; виправляємо її (інвертуємо помилковий розряд): 0110011. Із відкоректованого коду виділяємо інформаційну групу розрядів (третій, п'ятий, шостий і сьомий). У результаті отримуємо 1011.

Надлишковість у разі використання коду $H_{вип}(n)$ зменшується зі зростанням n:

$$R = \frac{n}{k} - 1 = \frac{2^s - 1}{2^s - s - 1} - 1 = \frac{s}{2^s - s - 1}.$$

Якщо $n \neq 2^s - 1$, то будують *укорочений код Хеммінга*. Його задають перевірною матрицею, утвореною першими n стовпцями перевірної матриці $H_{вип}(n_1)$, де n_1 — найменше ціле із чисел, які більші ніж n і мають вигляд $2^s - 1$. Очевидно, що всі попередні міркування правильні й для вкороченого коду Хеммінга.

Контрольні запитання та завдання

1. Нехай числа 1, 2, 4, 17, 98 закодовано їх двійковими розкладами з мінімально можливою довжиною. Наприклад, код одиниці — 1, двійки — 10, четвірки — 100. Чи це кодування роздільне?
2. Задано коди. Для кожного роздільного коду V побудувати префіксний код із тим самим набором довжин елементарних кодів:
   а) $V = \{01, 10, 100, 111, 011\}$;
   б) $V = \{1, 10, 00, 0100\}$;
   в) $V = \{10, 101, 111, 1011\}$.
3. Для заданих розподілів імовірностей появи букв побудувати коди методом Фано:
   а) $P = \{0.6, 0.1, 0.09, 0.08, 0.07, 0.06\}$;
   б) $P = \{0.4, 0.4, 0.1, 0.03, 0.03, 0.02, 0.02\}$;

в) $P=\{0.3, 0.2, 0.2, 0.1, 0.1, 0.05, 0.05\}$;

г) $P=\{0.25, 0.2, 0.15, 0.15, 0.15, 0.1\}$;

д) $P=\{0.4, 0.18. 0.1. 0.1. 0.07. 0.06. 0.05. 0.04\}$;

е) $P=\{0.2, 0.2, 0.19, 0.12, 0.11, 0.09, 0.09\}$.

Для кожного розподілу визначити $l^F_{\text{сер}}$.

4. Для розподілів імовірностей задачі 3 побудувати оптимальні коди методом Хаффмана. Для кожного розподілу визначити $l^H_{\text{сер}}$ і порівняти з відповідним значенням $l^F_{\text{сер}}$. Указати розподіли, для яких метод Фано не дає оптимального коду.

5. Довести, що якщо ймовірності появи букв являють собою степені двійки $p_i = 2^{-n_i}$, то довжини відповідних елементарних кодів, одержаних за методами Фано та Хаффмана, збігаються й дорівнюють n_i.

6. За допомогою алгоритму Хаффмана стиснути наведений нижче текст. Визначити коефіцієнт стиснення (уважати, що в нестисненому тексті кожний символ закодовано одним байтом):

а) THIS IS A SIMPLE EXAMPLE OF HUFFMAN ENCODING;

б) МІНІМІЗАЦІЯ ЧАСУ ВИКОНАННЯ ПРОГРАМИ.

7. Для коду V з'ясувати, чи він лінійний, і знайти кодову віддаль $\rho(V)$:

а) $V=\{000000, 011100, 100111, 111011\}$;

б) $V=\{00001, 01110, 10010, 11101\}$;

в) $V=\{00001, 01111, 10010, 11100\}$;

г) $V=\{00011, 00001, 10110, 11110\}$.

8. Нехай G – породжувальна матриця лінійного коду. Знайти перевірну матрицю та двоїстий код:

а) $G=\begin{bmatrix}1&0&1&0&1\\0&1&0&0&1\\1&0&0&1&0\end{bmatrix}$;

б) $G=\begin{bmatrix}0&0&1&0&1&1\\1&0&0&0&1&0\\0&1&0&1&0&1\end{bmatrix}$;

в) $G=\begin{bmatrix}0&1&1&0&1&1&0&1\\1&0&0&1&0&0&1&0\end{bmatrix}$.

9. За допомогою коду Хеммінга $H_{\text{вияв}}(n)$ для значення параметра $n=8$ закодувати повідомлення 01101111001101011101.

10. Повідомлення закодовано за допомогою коду Хеммінга $H_{\text{вияв}}(8)$. На виході каналу зв'язку із джерелом адитивних перешкод $P(8, 1)$ отримано код 01101001100101001110110. Чи можна твердити, що під час його передавання виникла помилка? Якщо так, то в якій групі цифр?

11. Для повідомлень 1101 і 1011 побудувати (7, 4)-код Хеммінга, використовуючи перевірну матрицю H.

12. За допомогою (7, 4)-коду Хеммінга закодувати такі повідомлення:

    а) 1100101111011; б) 100111010111.

    Для кодування використати перевірну матрицю H.

13. На виході каналу зв'язку із джерелом адитивних перешкод $P(7, 1)$ отримано такі комбінації в (7, 4)-коді Хеммінга:

    а) 1001001; б) 0110001; в) 0011111; г) 0110100.

    Застосувати корекцію кодів і декодувати ці повідомлення.

14. У каналі зв'язку із джерелом перешкод $P(7, 1)$ використано (7, 4)-код Хеммінга. Застосувати корекцію й декодувати повідомлення 11100101100111100011110010001.

15. Знайти породжувальну матрицю G для (7, 4)-коду Хеммінга.

16. Для повідомлень 1101 і 1011 побудувати (7, 4)-код Хеммінга, використовуючи породжувальну матрицю G (див. задачу 15).

17. За допомогою (7, 4)-коду Хеммінга закодувати такі повідомлення:

    а) 1100101111011; б) 100111010111.

    Для кодування використати породжувальну матрицю G (див. задачу 15).

18. Побудувати код Хеммінга для виправлення помилки в одному розряді та виявлення помилки у двох розрядах у разі передавання 4-розрядної двійкової комбінації. Навести перевірну матрицю цього коду.

    Вказівка. Використати (7, 4)-код Хеммінга, додавши перевірку на парність (розширений (8, 4)-код Хеммінга).

19. Під час передавання за розширеним (8, 4)-кодом Хеммінга (див. задачу 18) отримано такі повідомлення:

    а) 00100001; б) 00110001.

    Що можна твердити у наведених випадках?

20. Записати перевірну матрицю (15, 11)-коду Хеммінга. За її допомогою закодувати повідомлення 10110110110. Показати процес виявлення помилки в п'ятому розряді коду, отриманого на виході каналу зв'язку.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано повідомлення. Закодувати його методом Фано.
2. Задано повідомлення. Закодувати його методом Хаффмана.
3. Задано схему алфавітного кодування й код повідомлення. Декодувати це повідомлення.
4. Задано повідомлення. Закодувати його за допомогою (7, 4)-коду Хеммінга.
5. У каналі зв'язку із джерелом перешкод $P(7, 1)$ використано (7, 4)-код Хеммінга. Задано код повідомлення (можливо, з помилками, зумовленими властивостями каналу зв'язку). Декодувати це повідомлення.

Розділ 7

Булеві функції

- Означення булевої функції. Реалізація функцій формулами
- Алгебри булевих функцій
- Спеціальні форми подання булевих функцій
- Повнота й замкненість
- Мінімізація булевих функцій
- Реалізація булевих функцій схемами з функціональних елементів

У цьому розділі розглянуто теорію булевих функцій. Апарат булевих функцій широко застосовують у математичній і технічній кібернетиці, зокрема для проектування мікропроцесорів, побудови логічних форм функцій вибору в теорії вибору та прийняття рішень тощо.

7.1. Означення булевої функції. Реалізація функцій формулами

Булевою називають функцію $f(x_1, ... x_n)$ з областю значень {0, 1}, змінні $x_1, ... x_n$ якої також набувають лише цих двох значень. Множину всіх булевих функцій позначають P_2, множину всіх булевих функцій від n змінних — $P_2(n)$.

Булеву функцію від n змінних називають *n-місною*. Область її визначення — множина E_2^n усіх можливих двійкових наборів довжиною n. Отже, область визначення n-місної булевої функції скінченна й складається з 2^n наборів. Для набору $(a_1, ..., a_n)$ у цьому розділі будемо використовувати позначення $\tilde{a}^n$ або $\tilde{a}$ (якщо довжина набору зрозуміла з контексту).

Щоб зробити викладення цього розділу незалежним, сформулюємо ще раз означення норми двійкового набору та віддалі Хеммінга між наборами. *Нормою* набору $\tilde{a}^n$ називають число $\|\tilde{a}^n\|$, яке дорівнює кількості його одиничних компонент. *Віддаллю Хеммінга* між наборами $\tilde{a}^n$ і $\tilde{b}^n$ називають число $\rho(\tilde{a}^n, \tilde{b}^n)$, що дорівнює кількості компонент, у яких набори $\tilde{a}^n$ і $\tilde{b}^n$ різняться.

Набори $\tilde{a}^n$ і $\tilde{b}^n$ називають *сусідніми*, якщо $\rho(\tilde{a}^n, \tilde{b}^n) = 1$, і *протилежними*, якщо $\rho(\tilde{a}^n, \tilde{b}^n) = n$. Отже, сусідні набори різняться точно однією компонентою, а протилежні — усіма n компонентами. Наприклад, набори (0100) і (1100) сусідні, а (0100) і (1011) — протилежні.

Скінченність області визначення булевих функцій дає змогу задавати такі функції за допомогою таблиць. Розглянемо двійкові набори значень змінних як записи цілих чисел у двійковій системі числення. Це означає, що набір $\tilde{a}^n = (a_1, ..., a_n)$ ототожнюють із записом числа $a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + ... + a_{n-1} \cdot 2 + a_n$. Назвемо це число номером набору $\tilde{a}^n$. Наприклад, для тримісної булевої функції номером набору (110) — число $1 \cdot 2^2 + 1 \cdot 2 + 0 = 6$.

Номери наборів значень змінних n-місної булевої функції змінюються від 0 до $2^n - 1$. Розмістимо набори в стовпчик за зростанням їх номерів і покажемо значення функції на кожному наборі. Одержимо *таблицю булевої функції* (табл. 7.1).

Таблиця 7.1

x_1	x_2	...	x_{n-1}	x_n	$f(x_1, x_2, ..., x_{n-1}, x_n)$
0	0	...	0	0	$f(0, 0, ..., 0, 0)$
0	0	...	0	1	$f(0, 0, ..., 0, 1)$
0	0	...	1	0	$f(0, 0, ..., 1, 0)$
...	...	...	...	...	... ...
1	1	...	1	1	$f(1, 1, ..., 1, 1)$

Правий стовпець (стовпець значень функції) складається з 2^n нулів і одиниць. Отже, n-місних булевих функцій стільки, скільки наборів довжиною 2^n з 0 і 1. Тому справджується таке твердження.

ТЕОРЕМА 7.1. Кількість різних булевих функцій від n змінних дорівнює 2^{2^n}.

Далі ми завжди будемо вважати, що набори значень булевих функцій розміщено за зростанням їх номерів (від 0 до $2^n - 1$). Тому функцію $f(\tilde{x}^n) = f(x_1, ..., x_n)$ можна задати вектором $\tilde{y}_f = (y_0, y_1, ..., y_{2^n-1})$. у якому компонента y_i — це значення функції $f(\tilde{x}^n)$ на i-му наборі значень змінних, де $i = 0, 1, ..., 2^n - 1$.

Множину наборів, на яких булева функція $f(\tilde{x}^n)$ набуває значення 1, позначають N_f:

$$N_f = \{\tilde{a}^n \mid \tilde{a}^n \in E_2^n, f(\tilde{a}^n) = 1\}.$$

Очевидно, що множина N_f повністю визначає функцію f.

Змінну x_i функції $f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)$ називають *істотною*, якщо існує такий набір $(a_1, ..., a_{i-1}, a_{i+1}, ..., a_n)$ значень решти змінних, що

$$f(a_1, ..., a_{i-1}, 0, a_{i+1}, ..., a_n) \neq f(a_1, ..., a_{i-1}, 1, a_{i+1}, ..., a_n).$$

Змінну, яка не є істотною, називають *неістотною* або *фіктивною*. Отже, змінна x_i функції $f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)$ неістотна (фіктивна), якщо

$$f(x_1, ..., x_{i-1}, 0, x_{i-1}, ..., x_n) = f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$$

для будь-яких значень решти змінних. Це означає, що зміна значення x_i в довільному наборі значень $x_1, ..., x_n$ не змінює значення функції. Тоді функція $f(x_1, ..., x_n)$ насправді залежить від $(n-1)$ змінної, тобто являє собою функцію $g(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$. У такому разі можна говорити, що функцію g отримано

з функції f вилученням фіктивної змінної, а функцію f отримано з g *введенням* фіктивної змінної. Функції f і g називають *рівними*, якщо функцію g можна одержати з f уведенням або вилученням фіктивних змінних.

Фіктивні змінні вилучають тому, що вони не впливають на значення функції, тому зайві. Проте іноді корисно вводити фіктивні змінні. Завдяки цьому будь-яку функцію n змінних можна зробити функцією довільної більшої кількості змінних. Тому можна вважати, що всі булеві функції зі скінченної множини $\{f_1, ..., f_s\}$ залежать від одних і тих самих змінних $x_1, ..., x_n$. Зокрема, твердження $|P_2(n)| = 2^{2^n}$ теореми 7.1 передбачає, що ми враховуємо всі булеві функції від n змінних, включаючи функції з фіктивними змінними.

Зі зростанням кількості змінних швидко збільшується кількість залежних від них булевих функцій. Наприклад, різних булевих функцій чотирьох змінних є $2^{2^4} \approx 65{,}5$ тис., а п'яти змінних — $2^{2^5} \approx 4$ млрд. Зі збільшенням кількості змінних таблиці для булевих функцій стають громіздкими, і ними незручно користуватись.

Розглянемо аналітичний метод подання булевих функцій, тобто зображення функцій формулами. Спочатку за допомогою табл. 7.2, 7.3 означають функції, які називають *елементарними*.

Таблиця 7.2

x	0	1	x	$\bar{x}$
0	0	1	0	1
1	0	1	1	0

Таблиця 7.3

x_1	x_2	x_1x_2	$x_1 \vee x_2$	$x_1 \to x_2$	$x_1 \sim x_2$	$x_1 \oplus x_2$	$x_1 \mid x_2$	$x_1 \downarrow x_2$
0	0	0	0	1	1	0	1	1
0	1	0	1	1	0	1	1	0
1	0	0	1	0	0	1	1	0
1	1	1	1	1	1	0	0	0

Уведені елементарні функції мають такі назви:

- $f_1(x) = 0$ — *константа* 0;
- $f_2(x) = 1$ — *константа* 1;
- $f_3(x) = x$ — *тотожна функція*;
- $f_4(x) = \bar{x}$ — *заперечення* x, читають „не x";
- $f_5(x_1, x_2) = x_1x_2$ — *кон'юнкція*, читають „x_1 і x_2" (іноді використовують символи $\wedge$ та &);
- $f_6(x_1, x_2) = x_1 \vee x_2$ — *диз'юнкція*, читають „x_1 або x_2";
- $f_7(x_1, x_2) = x_1 \to x_2$ — *імплікація*, читають „із x_1 випливає x_2" (іноді використовують символ $\supset$);

- $f_8(x_1, x_2) = x_1 \sim x_2$ — *еквівалентність* (використовують також символ ≡);
- $f_9(x_1, x_2) = x_1 \oplus x_2$ — *додавання за* mod2, читають також як альтернативне „або" („або, або");
- $f_{10}(x_1, x_2) = x_1 | x_2$ — *штрих Шеффера*;
- $f_{11}(x_1, x_2) = x_1 \downarrow x_2$ — *стрілка Пірса*.

За допомогою елементарних функцій можна подати будь-яку булеву функцію аналітично, тобто у вигляді формули. Нехай задано булеві функції $f(x)$ і $g(x)$. Говорять, що функцію $h(x) = g(f(x))$ отримано *підстановкою* f у g. По-перше, можливі будь-які підстановки булевих функцій замість змінних у багатомісні булеві функції $f_1, ..., f_m$. По-друге, можливі будь-які *перейменування* змінних; наприклад, перейменування x_3 в x_2 перетворює функцію $f(x_1, x_2, x_3, x_4)$ на функцію трьох змінних $f(x_1, x_2, x_2, x_4)$ (у такому разі говорять, що змінні x_2 та x_3 *ототожнено*). Функцію, одержану з функцій $f_1, ..., f_m$ якоюсь підстановкою їх одна в одну або перейменуванням змінних, називають *суперпозицією функцій* $f_1, ..., f_m$. Вираз, який описує цю суперпозицію та містить функціональні знаки, круглі дужки й символи змінних, називають *формулою*. Поняття суперпозиції дуже важливе, тому розглянемо його докладніше.

Нехай задано множину елементарних функцій $Q = \{f_1, ..., f_m\}$. Символи змінних $x_1, ..., x_n$ уважають *формулами глибиною* 0. Формула F має *глибину* $k+1$, якщо вона має вигляд $f_i(F_1, ..., F_{n_i})$, де $f_i \in Q$, n_i – кількість аргументів функції f_i, а $F_1, ..., F_{n_i}$ — формули, максимальна з глибин яких дорівнює k. $F_1, ..., F_{n_i}$ називають *підформулами* формули F, а функцію f_i – зовнішньою (головною) операцією формули F. Усі підформули формул $F_1, ..., F_{n_i}$ також називають підформулами формули F. Наприклад, $f_5(x_1, x_2)$ — це формула глибиною 1, а $f_9(f_6(x_3, x_1), f_5(x_1, f_9(x_1, x_2)))$ – формула глибиною 3, яка містить одну підформулу глибиною 2 та дві підформули глибиною 1.

Надалі будемо розглядати конкретні формули в більш звичному вигляді, коли знаки функцій стоять між аргументами (такий запис називають *інфіксним*).

Приклад 7.1. Якщо f_3 – кон'юнкція, f_6 – диз'юнкція, f_9 – додавання за mod2, то наведена вище формула має вигляд

$$(x_3 \vee x_1) \oplus (x_1 \wedge (x_1 \oplus x_2)).$$

Цю тримісну функцію задано не таблицею, а формулою. Вона являє собою суперпозицію функцій f_3, f_6 і f_9.

Формулу, побудовану описаним вище способом, тобто таку, що містить лише символи змінних, дужки та знаки функцій із множини Q, називають *формулою над* Q.

Приклад 7.2. Нехай $Q = \{\bar{x}, xy, x \vee y\}$; тоді вираз $\overline{(x \vee y)} z \vee t$ – формула над Q.

Щоб зменшити кількість дужок у формулах, уводять пріоритет операцій:

- заперечення;
- кон'юнкція;
- усі інші операції.

Крім того, домовляються, що символ заперечення відіграє роль дужок, якщо він є над частиною формули.

Можна й інакше означити поняття глибини. Наприклад, часто вважають, що розставлення заперечень над змінними не збільшує глибини: коли формула Q містить асоціативну операцію f, можна означити глибину так, що застосування f до формул з тою самою зовнішньою операцією f не збільшує глибини формули. Наприклад, формули $x_1(x_2 \vee x_3x_4)$ та $x_2x_1(x_2 \vee x_3x_4)$ мають одну й ту саму глибину 3; диз'юнктивні та кон'юнктивні нормальні форми (див. підрозділ 7.3) завжди мають глибину 2.

Із вигляду формули, у якій функцію f виражено як суперпозицію інших функцій, випливає очевидний спосіб її обчислення за таким правилом: формулу можна обчислити лише тоді, коли вже обчислено значення всіх її підформул. Отже, формула кожному набору значень аргументів ставить у відповідність значення функції, тому її можна вважати способом подання й обчислення функції. Зокрема, за формулою обчисленням її значень на всіх 2^n наборах можна відновити таблицю функції. Про формулу, що подає функцію, говорять також, що вона *реалізує* (*задає*) цю функцію.

Приклад 7.3. Функцію задано формулою $\overline{x} \to (\overline{z} \sim (y \oplus xz))$. Потрібно задати її таблицею. Процес розв'язування цієї задачі проілюстровано в табл. 7.4.

Таблиця 7.4

x	y	z	xz	$y \oplus xz$	$\overline{z}$	$\overline{z} \sim (y \oplus xz)$	$\overline{x}$	$\overline{x} \to (\overline{z} \sim (y \oplus xz))$
0	0	0	0	0	1	0	1	0
0	0	1	0	0	0	1	1	1
0	1	0	0	1	1	1	1	1
0	1	1	0	1	0	0	1	0
1	0	0	0	0	1	0	0	1
1	0	1	1	1	0	0	0	1
1	1	0	0	1	1	1	0	1
1	1	1	1	0	0	1	0	1

Подання функції формулою не єдине. Формули називають *еквівалентними* (*рівносильними*), якщо вони реалізують рівні булеві функції. Еквівалентність формул позначають символом $=$.

Приклад 7.4. Формули $x \to y$ і $(\overline{x} \vee y) \vee z\overline{z}$ еквівалентні: $x \to y = (\overline{x} \vee y) \vee z\overline{z}$. У цьому можна переконатись, побудувавши таблиці відповідних булевих функцій. Очевидно, що змінна z у другій формулі фіктивна.

Крім побудови таблиць є й інші методи доведення еквівалентності формул і побудови нових формул, рівносильних даним. Ці методи називають *рівносильними (еквівалентними) перетвореннями формул*.

7.2. Алгебри булевих функцій

Нехай функцію f_1 задано формулою F_1, а функцію f_2 – формулою F_2. Підстановка F_1 і F_2, наприклад, у диз'юнкцію $x_1 \vee x_2$ дає формулу $F_1 \vee F_2$. Уявши формулу Φ_1, рівносильну F_1 (тобто формула Φ_1 також реалізує функцію f_1) та формулу Φ_2, рівносильну F_2, отримаємо формулу $\Phi_1 \vee \Phi_2$, рівносильну формулі $F_1 \vee F_2$.

Отже, диз'юнкцію можна розглядати як двомісну операцію на множині всіх булевих функцій. Ця операція кожній парі функцій f_1 і f_2, незалежно від вигляду формул, якими їх подано, однозначно ставить у відповідність функцію $f_1 \vee f_2$. Аналогічно, й інші булеві функції можна розглядати як операції на множині P_2 всіх булевих функцій. Наприклад, заперечення – одномісна операція, кон'юнкція та диз'юнкція – двомісні.

Множину P_2 всіх булевих функцій разом з уведеною на ній системою операцій називають *алгеброю булевих функцій* [11, 47].

Розглянемо дві алгебри. Алгебру $(P_2; \neg, \wedge, \vee)$ з операціями заперечення, кон'юнкції та диз'юнкції називають *алгеброю Буля*, а алгебру $(P_2; \wedge, \oplus)$ з операціями кон'юнкції та додавання за mod2 – *алгеброю Жегалкіна*. Формули цих алгебр будують зі знаків операцій, круглих дужок, символів змінних і констант 0 і 1. Знак кон'юнкції $\wedge$ у формулах обох алгебр зазвичай не пишуть.

Якщо немає дужок, пріоритет операцій у булевій алгебрі такий: заперечення, кон'юнкція, диз'юнкція. В алгебрі Жегалкіна спочатку виконується кон'юнкція, а потім – додавання за mod2. За наявності дужок спочатку виконуються операції всередині їх. У булевій алгебрі як дужки в разі заперечення виразів використовують сам символ заперечення.

Приклад 7.5. Замість $\overline{(x \vee y)} \vee (xy)$ можна написати $\overline{x \vee y} \vee xy$, а замість $(xy) \oplus (yz) \oplus xy \oplus yz$.

Одна з найважливіших задач – виявлення основних еквівалентностей. Ці еквівалентності називають *законами* відповідної алгебри.

Закони алгебри Буля:

- закони асоціативності $(xy)z = x(yz) = xyz$, $(x \vee y) \vee z = x \vee (y \vee z) = x \vee y \vee z$;
- закони комутативності $xy = yx$, $x \vee y = y \vee x$;
- дистрибутивний закон для кон'юнкції щодо диз'юнкції $(x \vee y)\, z = xz \vee yz$;
- дистрибутивний закон для диз'юнкції щодо кон'юнкції $xy \vee z = (x \vee z)(y \vee z)$;
- закон подвійного заперечення $\overline{\overline{x}} = x$;
- закони де Моргана $\overline{xy} = \overline{x} \vee \overline{y}$, $\overline{x \vee y} = \overline{x}\,\overline{y}$;
- закони ідемпотентності $xx = x$, $x \vee x = x$;
- закони поглинання $x \vee xy = x$, $x(x \vee y) = x$;
- співвідношення для констант $\overline{1} = 0$, $\overline{0} = 1$, $1x = x$, $0x = 0$, $1 \vee x = 1$, $0 \vee x = x$;
- закон виключеного третього $x \vee \overline{x} = 1$;
- закон суперечності $x\overline{x} = 0$.

Закони алгебри Жегалкіна:

- закони асоціативності $(xy)z = x(yz) = xyz$, $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$;
- закони комутативності $xy = yx$, $x \oplus y = y \oplus x$;
- дистрибутивний закон для кон'юнкції щодо додавання за $\mathrm{mod}2$ $x(y \oplus z) = xy \oplus xz$;
- співвідношення для констант $1x = x$, $0x = 0$, $x \oplus 0 = x$;
- закон ідемпотентності для кон'юнкції $xx = x$;
- закон зведення подібних членів у разі додавання за $\mathrm{mod}2$ $x \oplus x = 0$.

Правильність цих еквівалентностей можна довести за допомогою таблиць.

Наведені еквівалентності справджуються й у разі підстановки замість змінних довільних булевих функцій (тобто формул, які подають ці функції). Важливо лише дотримуватися такого правила підставлення формули замість змінної: підставляючи формулу F замість змінної x, усі входження змінної x у цю еквівалентність потрібно водночас замінити формулою F. Наприклад, підставивши замість змінної x формулу xy, а замість $y - z \vee u$, з $\overline{xy} = \overline{x} \vee \overline{y}$ отримаємо $\overline{xy(z \vee u)} = \overline{xy} \vee \overline{z \vee u}$.

Закони алгебр Буля та Жегалкіна дають змогу доводити нові еквівалентності вже без таблиць, на основі тотожних перетворень.

Приклад 7.6. Доведемо рівності

а) $(x \vee y)(z \vee u) = xz \vee yz \vee xu \vee yu$;
б) $xy \vee zu = (x \vee z)(y \vee z)(x \vee u)(y \vee u)$.

Двічі застосувавши закони дистрибутивності, одержимо:

а) $(x \vee y)(z \vee u) = (x \vee y)z \vee (x \vee y)u = xz \vee yz \vee xu \vee yu$.
б) $xy \vee zu = (x \vee zu)(y \vee zu) = (x \vee z)(y \vee z)(x \vee u)(y \vee u)$.

Приклад 7.7. Доведемо, що $\overline{xzy} = x \vee \overline{z} \vee \overline{y}$. Послідовно застосувавши закони де Моргана, подвійного заперечення й асоціативності, запишемо

$$\overline{xzy} = \overline{x\overline{z}} \vee \overline{y} = \overline{\overline{x}} \vee \overline{\overline{z}} \vee \overline{y} = x \vee \overline{z} \vee \overline{y}.$$

Наведемо еквівалентності, які дають змогу перетворити будь-яку формулу булевої алгебри в рівносильну до неї формулу алгебри Жегалкіна й навпаки:

- $\overline{x} = 1 \oplus x$;
- $x \vee y = x \oplus y \oplus xy$;
- $x \oplus y = \overline{x}\,y \vee x\,\overline{y}$.

За допомогою законів алгебр Буля та Жегалкіна можна спрощувати різні формули в цих алгебрах.

Приклад 7.8. Перетворимо $\overline{xy \vee \overline{z}}$ на рівносильну формулу алгебри Жегалкіна. Одержану формулу спростимо.

$$\overline{xy \vee \overline{z}} = (xy \vee (z \oplus 1)) \oplus 1 = (xy \oplus (z \oplus 1) \oplus xy(z \oplus 1)) \oplus 1 =$$
$$= xy \oplus z \oplus 1 \oplus xyz \oplus xy \oplus 1 = xyz \oplus z.$$

Запишемо формули $x \to y$ та $x \sim y$ за допомогою операцій алгебри Буля. Оскільки $x \sim y = (x \to y)(y \to x)$, то достатньо виразити формулу $x \to y$. Порівнявши таблицю для $x \to y$ із таблицею для $\bar{x} \vee y$, отримаємо еквівалентність $x \to y = \bar{x} \vee y$.

Розглянемо тепер іще одне важливе поняття. Функцію $f^*(x_1, ..., x_n)$ називають *двоїстою* до функції $f(x_1, ..., x_n)$, якщо $f^*(x_1, ..., x_n) = \bar{f}(\bar{x}_1, ..., \bar{x}_n)$. Візьмемо заперечення обох частин останньої рівності й підставимо $\bar{x}_1, ..., \bar{x}_n$ замість $x_1, ..., x_n$; одержимо $f^*(\bar{x}_1, ..., \bar{x}_n) = \bar{f}(\bar{\bar{x}}_1, ..., \bar{\bar{x}}_n) = f(x_1, ..., x_n)$. Звідси випливає, що функція f двоїста до f^*, тобто $(f^*)^* = f$. Отже, відношення двоїстості між функціями симетричне. Пари двоїстих функцій утворюють, наприклад, кон'юнкція та диз'юнкція, константи 0 і 1, додавання за mod2 й еквівалентність. Таблицю для двоїстої функції для вибраного нами порядку наборів отримують із таблиці для функції $f(x_1, ..., x_n)$ інвертуванням (тобто заміною 0 на 1 й 1 на 0) стовпця функції та його перевертанням.

З означення двоїстості зрозуміло, що для довільної функції двоїста функція визначається однозначно. Функція може бути двоїстою до самої себе. Тоді її називають *самодвоїстою*. Наприклад, заперечення $\bar{x}$ та $x \oplus y \oplus z$ – самодвоїсті функції.

Нехай функцію задано формулою F. Який вигляд має формула F^*, що задає двоїсту функцію? Відповідь на це питання дає така теорема.

ТЕОРЕМА 7.2. Якщо

$$F(x_1, ..., x_n) = f(f_1(x_1, ..., x_n), ..., f_s(x_1, ..., x_n)),$$

то

$$F^*(x_1, ..., x_n) = f^*(f_1^*(x_1, ..., x_n), ..., f_s^*(x_1, ..., x_n)).$$

Доведення.

$$F^*(x_1, ..., x_n) = \bar{F}(\bar{x}_1, ..., \bar{x}_n) = \bar{f}(f_1(\bar{x}_1, ..., \bar{x}_n), ..., f_s(\bar{x}_1, ..., \bar{x}_n)) =$$
$$= \bar{f}(\bar{\bar{f}}_1(\bar{x}_1, ..., \bar{x}_n), ..., \bar{\bar{f}}_s(\bar{x}_1, ..., \bar{x}_n)) = \bar{f}(\bar{f}_1^*(x_1, ..., x_n), ..., \bar{f}_s^*(x_1, ..., x_n)) =$$
$$= f^*(f_1^*(x_1, ... x_n), ..., f_s^*(x_1, ... x_n)).$$

З теореми 7.2 випливає таке твердження.

Принцип двоїстості. Якщо у формулі F, що реалізує функцію f, усі символи функцій замінити символами відповідних двоїстих функцій, то одержана формула F^* реалізує функцію f^*, двоїсту до f.

Інакше це твердження можна сформулювати так: функція, двоїста до суперпозиції функцій, дорівнює відповідній суперпозиції двоїстих функцій.

В алгебрі Буля принцип двоїстості має простіший вигляд.

Принцип двоїстості в алгебрі Буля. Якщо у формулі F, що реалізує функцію f, усі кон'юнкції замінити на диз'юнкції, диз'юнкції – на кон'юнкції, 1 – на 0, 0 – на 1, то отримаємо формулу F^*, яка реалізує функцію f^*, двоїсту до f.

Приклад 7.9. Знайдемо функцію, двоїсту до $f(x,y,z) = xy \vee \bar{z}(x \vee \bar{u}y)$.
За принципом двоїстості маємо $f^*(x,y,z) = (x \vee y)(\bar{z} \vee x(\bar{u} \vee y))$. Застосовуючи принцип двоїстості, слід ураховувати пріоритет операцій, тобто в разі потреби розставляти дужки. Так, у формулі $xy \vee \bar{z}(x \vee \bar{u}y)$ спочатку виконуються кон'юнкції xy та $\bar{u}y$ (дужки випущено за домовленістю про пріоритет операцій); отже, у формулі, що реалізує двоїсту функцію, спочатку мають виконуватися диз'юнкції $x \vee y$ та $\bar{u} \vee y$, тому потрібні дужки.

Якщо функції рівні, то й двоїсті до них функції також рівні. Це дає змогу отримувати нові еквівалентності за допомогою принципу двоїстості. Для цього потрібно від еквівалентності $F_1 = F_2$ перейти до $F_1^* = F_2^*$.

7.3. Спеціальні форми подання булевих функцій

Спеціальними формами подання булевих функцій є диз'юнктивні й кон'юнктивні нормальні форми та поліном Жегалкіна.

7.3.1. Диз'юнктивні нормальні форми

Уведемо позначення $x^\sigma = x\sigma \vee \bar{x}\bar{\sigma}$, де σ — параметр, який дорівнює 0 чи 1. Очевидно, що

$$x^\sigma = \begin{cases} \bar{x}, & \text{якщо } \sigma = 0, \\ x, & \text{якщо } \sigma = 1. \end{cases}$$

Зауважимо, що $\sigma^\sigma = 1$.

Зафіксуємо множину змінних $X = \{x_1, x_2, ..., x_n\}$.

Елементарною кон'юнкцією називають вираз $k = x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_r}^{\sigma_r}$, де x_{i_j} — змінні з множини X, причому всі x_{i_j} різні. Число r називають *рангом* кон'юнкції. У разі $r = 0$ кон'юнкцію називають *порожньою* та вважають такою, що дорівнює 1.

Приклад 7.10. Елементарними кон'юнкціями є, наприклад 1, $\bar{x}_1$, x_1x_2, x_2, $\bar{x}_1\bar{x}_2x_3$, а вирази 0, $x_1x_2x_1$, $\bar{x}_1x_1$, $\bar{x}_1\bar{x}_2$ — не елементарні кон'юнкції.

Елементарну кон'юнкцію, яка містить усі змінні з множини X, називають *конституентою одиниці*. Інакше кажучи, конституента одиниці — це елементарна кон'юнкція з рангом n. Очевидно, що всіх різних конституент одиниці для фіксованої множини n змінних x_1, x_2, ..., x_n стільки, скільки двійкових наборів з n компонентами, тобто 2^n.

Диз'юнктивною нормальною формою (*ДНФ*) називають диз'юнкцію $D = k_1 \vee k_2 \vee ... \vee k_s$ елементарних кон'юнкцій k_j, у якій k_j попарно різні.

Є алгоритм, який дає змогу для будь-якої формули булевої алгебри на основі тотожних перетворень знайти рівносильну до неї ДНФ. На першому його етапі формулу перетворюють на рівносильну, побудовану зі змінних і їх заперечень

за допомогою самих лише кон'юнкцій і диз'юнкцій (тобто заперечення можуть стояти лише над змінними). Для цього використовують закони де Моргана та закон подвійного заперечення.

На другому етапі домагаються, щоб усі кон'юнкції виконувалися раніше, ніж диз'юнкції, для чого розкривають дужки на підставі дистрибутивного закону для кон'юнкції $(x \vee y)z = xz \vee yz$ або рівносильності $(x \vee y)(z \vee u) = xz \vee yz \vee xu \vee yu$ (див приклад 7.6). Далі з використанням співвідношень для констант і закону суперечності вилучають нулі та, виходячи із законів ідемпотентності, об'єднують рівні члени. На цьому процес отримання ДНФ закінчують.

Приклад 7.11. Зведемо формулу $\overline{x \vee z(x \to y)}$ до ДНФ. Застосовуючи сформульований алгоритм, можемо записати

$$\overline{x \vee z(x \to y)} = \overline{x} \vee \overline{z(\overline{x} \vee y)} = \overline{x}\overline{z} \vee \overline{x}\overline{z}y = \overline{x}\overline{z} \vee \overline{x}yz.$$

Зазначимо, що ДНФ булевої функції не єдина; наприклад,

$$xz \vee y\overline{z} \vee xy = xz \vee y\overline{z}.$$

Досконалою диз'юнктивною нормальною формою (*ДДНФ*) називають ДНФ, у якій кожна елементарна кон'юнкція k_j $(j = 1, ..., s)$ — конституента одиниці.

ТЕОРЕМА 7.3. Будь-яку булеву функцію $f(x_1, ..., x_n) \neq 0$ можна єдиним способом подати в ДДНФ.

Доведення. Нехай задано функцію $f(x_1, ..., x_n) \neq 0$. Кожному двійковому набору $\tilde{a}_n = (a_1, a_2, .., a_n)$ значень змінних відповідає єдина конституента одиниці $k = x_1^{a_1} x_2^{a_2} ... x_n^{a_n}$, яка перетворюється на цьому наборі в 1. Усі інші конституенти одиниці на цьому наборі перетворюються в 0. Наприклад, набору (0101) відповідає конституента $\overline{x_1}x_2\overline{x_3}x_4$.

Нехай f_i — значення функції f, якого вона набуває на i-му двійковому наборі $(i = 0, 1, ..., 2^n - 1)$, а k_i — конституента одиниці, що відповідає i-му набору. Доведемо рівність

$$f(x_1, ..., x_n) = f_0 k_0 \vee f_1 k_1 \vee ... \vee f_{2^n-1} k_{2^n-1}.$$

Для l-го набору $f_l = 0 \vee 0 \vee ... \vee 0 \vee f_l 1 \vee 0 \vee ... \vee 0 = f_l$. Нульові члени в диз'юнкції можна випустити. Отже, диз'юнкція конституент одиниці, що відповідають усім двійковим наборам, на яких булева функція набуває значення 1, являє собою ДДНФ функції $f(x_1, ..., x_n)$:

$$f(x_1, x_2, ..., x_n) = \bigvee_{\substack{(\sigma_1, \sigma_2, .., \sigma_n) \\ f(\sigma_1, \sigma_2, ..., \sigma_n)=1}} x_1^{\sigma_1} x_1^{\sigma_2} ... x_n^{\sigma_n}.$$

Для доведення єдиності ДДНФ скористаємося таким комбінаторним міркуванням. Знайдемо кількість ДДНФ від n змінних $x_1, ..., x_n$. Для цього будь-яким способом занумеруємо конституенти одиниці, їх буде 2^n. Кожній ДДНФ від змінних $x_1, ..., x_n$ можна таким взаємно однозначним способом поставити у відповідність набір із 2^n нулів і одиниць. На позиції з номерами тих конституент одиниці, які входять у ДДНФ, поставимо одиниці, а на решту позицій — нулі.

Нульовий набір при цьому не отримаємо, бо він відповідав би порожній ДДНФ. Отже, різних ДДНФ стільки, скільки існує наборів довжиною 2^n, відмінних від набору із самих лише нулів, тобто $2^{2^n} - 1$. Функцій (окрім тотожного нуля) від змінних $x_1, ..., x_n$ також $2^{2^n} - 1$. Кожну з них можна подати ДДНФ, до того ж єдиним способом.

Із доведення теореми 7.3 випливає, що для функції, заданої таблицею, ДДНФ будують так: для кожного набору, на якому функція набуває значення 1, будують відповідну йому конституенту одиниці; диз'юнкція всіх цих конституент — це ДДНФ даної функції.

Приклад 7.12. Побудуємо ДДНФ для функції, заданої табл. 7.5.

Таблиця 7.5

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

Функція набуває значення 1 на наборах (00) і (11), отже $f(x_1, x_2) = \overline{x_1}\,\overline{x_2} \vee x_1 x_2$.

Будь-яку ДНФ можна звести до ДДНФ *розщепленням* кон'юнкцій, які містять не всі змінні: якщо кон'юнкція k не містить змінної x, то

$$k = k(x \vee \overline{x}) = kx \vee k\overline{x}.$$

Приклад 7.13. Перетворимо ДНФ $\overline{x}\,\overline{z} \vee \overline{x}yz$ на досконалу. Застосувавши розщеплення для кон'юнкції $\overline{x}\,\overline{z}$, одержимо

$$\overline{x}\,\overline{z} \vee \overline{x}yz = \overline{x}(y \vee \overline{y})\overline{z} \vee \overline{x}yz = \overline{x}y\overline{z} \vee \overline{x}\,\overline{y}\,\overline{z} \vee \overline{x}yz.$$

Якщо з формули F_1 за допомогою тотожних перетворень можна отримати формулу F_2, то з F_2 можна одержати F_1, обернувши ці перетворення.

ТЕОРЕМА 7.4. Для довільних двох рівносильних формул алгебри Буля F_1 і F_2 існує еквівалентне перетворення F_1 на F_2 за допомогою законів цієї алгебри.

Доведення. Перетворимо F_1 і F_2 на ДДНФ. Оскільки формули F_1 і F_2 рівносильні, то їх ДДНФ збігаються. Обернувши друге перетворення, матимемо такий ланцюжок перетворень: з F_1 одержимо ДДНФ, з ДДНФ — F_2.

Важливість цієї теореми полягає в тому, що набір законів алгебри Буля виявляється достатнім для довільних еквівалентних перетворень у ній.

7.3.2. Кон'юнктивні нормальні форми

Двоїстим способом, замінюючи в означеннях нулі одиницями й навпаки, диз'юнкції кон'юнкціями й навпаки, означають поняття елементарної диз'юнкції, конституенти нуля, кон'юнктивної нормальної форми, досконалої кон'юнктивної нормальної форми.

Нехай, як і раніше, зафіксовано множину $X = \{x_1, x_2, ..., x_n\}$. *Елементарною диз'юнкцією* називають вираз $d = x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee ... \vee x_{i_r}^{\sigma_r}$, у якому всі x_{i_j} різні, $x_{i_j} \in X$. Число r називають *рангом* диз'юнкції. Якщо $r = 0$, диз'юнкцію називають *порожньою* та вважають такою, що дорівнює 0. Наприклад, елементарні диз'юнкції – це $x_1 \vee x_3 \vee \overline{x_7}$, 0, $\overline{x_1} \vee x_2$.

Кон'юнктивною нормальною формою (*КНФ*) називають кон'юнкцію $d_1 \wedge d_2 \wedge ... \wedge d_s$ елементарних диз'юнкцій d_j, у якій усі d_j різні.

Є алгоритм, який дає змогу для будь-якої формули булевої алгебри знайти рівносильну до неї КНФ. Перший етап цього алгоритму такий самий, як і для побудови ДНФ. На другому етапі домагаються, щоб усі диз'юнкції виконувалися раніше кон'юнкцій. Для цього потрібно скористатися дистрибутивним законом $x \vee yz = (x \vee y)(x \vee z)$ або наслідком із нього $xy \vee zu = (x \vee y)(x \vee u)(y \vee z)(y \vee u)$ (див. приклад 7.6). Потім на підставі співвідношень для констант і закону виключеного третього вилучають одиниці та на підставі законів ідемпотентності об'єднують рівні члени.

Приклад 7.14. Знайдемо КНФ для формули $\overline{x \vee z}\ (x \to y)$. Застосовуючи сформульований алгоритм, одержимо

$$\overline{x \vee z}\ (x \to y) = \overline{x \vee z}\ (\overline{x} \vee y) = \overline{x}\,\overline{z}(\overline{x} \vee y).$$

Елементарну диз'юнкцію, яка містить усі змінні з множини X, називають *конституентою нуля*. Інакше кажучи, конституента нуля – це елементарна диз'юнкція з рангом n. Кожному двійковому набору $\tilde{b}^n = (b_1, b_2, ..., b_n)$ взаємно однозначно відповідає конституента нуля $x_1^{\bar{b}_1} \vee x_2^{\bar{b}_2} \vee ... x_n^{\bar{b}_n}$, яка перетворюється на ньому в 0. Усі інші конституенти нуля на цьому наборі перетворюються в 1. Наприклад, набору 0111 відповідає конституента нуля $x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}$.

Досконалою кон'юнктивною нормальною формою (ДКНФ) називають КНФ, у якій кожна елементарна диз'юнкція d_j ($j = 1, ..., s$) – конституента нуля.

Доведемо, що кожну булеву функцію $f(x_1, ..., x_n) \neq 1$ можна подати досконалою КНФ. Запишемо ДДНФ для f^* (зазначимо, що $f^* \neq 0$).

$$f^*(x_1, x_2, ..., x_n) = \bigvee_{\substack{(\tau_1, ..., \tau_n) \\ f^*(\tau_1, ..., \tau_n)=1}} x_1^{\tau_1} \wedge ... \wedge x_n^{\tau_n}.$$

Запишемо тотожність для двоїстих формул

$$(f^*)^*(x_1, x_2, ..., x_n) = \bigwedge_{\substack{(\tau_1, ..., \tau_n) \\ f^*(\tau_1, ..., \tau_n)=1}} (x_1^{\tau_1} \vee ... \vee x_n^{\tau_n}).$$

Ліва частина дорівнює $f(x_1, ..., x_n)$, а праву перетворюємо далі:

$$\bigwedge_{\substack{(\tau_1, ..., \tau_n) \\ f^*(\tau_1, ..., \tau_n)=1}} (x_1^{\tau_1} \vee ... \vee x_n^{\tau_n}) = \bigwedge_{\substack{(\tau_1, ..., \tau_n) \\ f(\bar{\tau}_1, ..., \bar{\tau}_n)=0}} (x_1^{\tau_1} \vee ... \vee x_n^{\tau_n}) =$$

$$= \bigwedge_{\substack{(\sigma_1, ..., \sigma_n) \\ f(\sigma_1, ..., \sigma_n)=0}} (x_1^{\bar{\sigma}_1} \vee ... \vee x_n^{\bar{\sigma}_n}).$$

Отже,

$$f(x_1, x_2, ..., x_n) = \bigwedge_{\substack{(\sigma_1, ..., \sigma_n) \\ f(\sigma_1, ..., \sigma_n)=0}} (x_1^{\bar{\sigma}_1} \vee x_2^{\bar{\sigma}_2} \vee ... \vee x_n^{\bar{\sigma}_n}).$$

Звідси випливає, що ДКНФ за таблицею булевої функції f будують так. Виділяють набори, на яких функція має значення 0, і для кожного з них записують відповідну конституенту нуля. Кон'юнкція цих конституент нуля являє собою ДКНФ функції f.

Приклад 7.15. Побудуємо ДКНФ для функції, заданої табл. 7.5. Функція набуває значення 0 на наборах (01) і (10), отже $f(x_1, x_2) = (x_1 \vee \bar{x}_2)(\bar{x}_1 \vee x_2)$.

Зазначимо, що за допомогою тотожних перетворень будь-яку КНФ можна перетворити на ДКНФ. Якщо в якусь елементарну диз'юнкцію d не входить змінна x, то потрібно записати рівносильний вираз $d \vee x\bar{x}$ та застосувати дистрибутивний закон: $d \vee x\bar{x} = (d \vee x)(d \vee \bar{x})$. Після тривіальних перетворень отримаємо ДКНФ.

Приклад 7.16. Перетворимо КНФ $\bar{x}z(\bar{x} \vee y)$ на ДКНФ. Розщепивши диз'юнкції, можемо записати

$$\bar{x}\,\bar{z}(\bar{x} \vee y) = (\bar{x} \vee y\bar{y} \vee z\bar{z})(x\bar{x} \vee y\bar{y} \vee \bar{z})(\bar{x} \vee y \vee z\bar{z}) =$$

$$= (\bar{x} \vee (y \vee z)(y \vee \bar{z})(\bar{y} \vee z)(\bar{y} \vee \bar{z})) \wedge$$

$$\wedge((x \vee y)(x \vee \bar{y})(\bar{x} \vee y)(\bar{x} \vee \bar{y}) \vee \bar{z})(\bar{x} \vee (y \vee z)(y \vee \bar{z})) =$$

$$= (\bar{x} \vee y \vee z)(\bar{x} \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee z)(\bar{x} \vee \bar{y} \vee \bar{z}) \wedge$$

$$\wedge(x \vee y \vee \bar{z})(x \vee \bar{y} \vee \bar{z})(\bar{x} \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee \bar{z}) \wedge$$

$$\wedge(\bar{x} \vee y \vee z)(\bar{x} \vee y \vee \bar{z}) =$$

$$= (\bar{x} \vee y \vee z)(\bar{x} \vee y \vee \bar{z})(x \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee z)(\bar{x} \vee \bar{y} \vee \bar{z})(x \vee \bar{y} \vee \bar{z}).$$

Зазначимо, що ДКНФ єдина.

7.3.3. Поліном Жегалкіна

Елементарну кон'юнкцію називають *монотонною*, якщо вона не містить заперечень змінних. Наприклад, $x_1x_2x_3$, x_1, 1 — монотонні кон'юнкції.

Формулу

$$P(\bar{x}^n) = k_1 \oplus k_2 \oplus ... \oplus k_s,$$

де k_1, k_2, ..., k_s — попарно різні монотонні кон'юнкції змінних із множини $X = \{x_1, x_2, ..., x_n\}$, називають *поліномом Жегалкіна*. Найбільший із рангів елементарних кон'юнкцій, що входять у поліном, називають *степенем* полінома. За окремим означенням 0 також уважатимемо поліномом Жегалкіна.

Приклад 7.17. Формули x, 1, $xyz \oplus xy \oplus z \oplus 1$ — поліноми Жегалкіна, а $x\bar{x}$, $xy \oplus yx \oplus x \oplus 1$ — ні.

Щоб із будь-якої формули алгебри Жегалкіна одержати поліном Жегалкіна, достатньо розкрити дужки (за дистрибутивним законом), застосувати, якщо можливо, закон ідемпотентності для кон'юнкції та звести подібні члени.

ТЕОРЕМА 7.5. Будь-яку булеву функцію можна єдиним способом подати у вигляді полінома Жегалкіна.

Доведення. Нехай $f(x_1, ..., x_n)$ – довільна булева функція. Достатньо провести доведення для функцій, відмінних від констант, тому що 1 та 0 – поліномами Жегалкіна. Задамо функцію f досконалою ДНФ, тобто диз'юнкцією конституент одиниці $f = k_1 \vee k_2 \vee ... \vee k_s$. Замінивши знаки $\vee$ на $\oplus$, одержимо формулу $g = k_1 \oplus k_2 \oplus ... \oplus k_s$. Покажемо, що функції f і g рівні. На жодному наборі $(a_1, ..., a_n)$ значень змінних дві різні конституенти одиниці не можуть водночас перетворитися на 1, бо i-та конституента одиниці набуває значення 1 лише на i-му наборі. Отже, обчислюючи значення функцій f і g, потрібно обчислити значення багатомісної диз'юнкції чи, відповідно, суми за mod2 членів, із яких тільки один може дорівнювати 1, а решта – обов'язково 0. Але тоді диз'юнкція та додавання за mod2 дають один і той самий результат (0, якщо всі члени дорівнюють 0, й 1, якщо один і тільки один член дорівнює 1).

У формулі $k_1 \oplus k_2 \oplus ... \oplus k_s$ за тотожністю $\bar{x} = 1 \oplus x$ замінимо всі заперечення змінних. Одержимо формулу алгебри Жегалкіна. Залишається лише розкрити дужки, застосувати закон ідемпотентності для кон'юнкції та звести подібні члени. Як результат отримаємо подання булевої функції f у вигляді полінома Жегалкіна.

Залишилося довести, що таке подання єдине. Для цього підрахуємо кількість поліномів Жегалкіна від n змінних $x_1, x_2, ..., x_n$. Кількість кон'юнкцій вигляду $x_{i_1} x_{i_2} ... x_{i_s}$ дорівнює кількості підмножин $\{i_1, i_2, ..., i_s\}$ множини $\{1, 2, ..., n\}$, тобто 2^n. Кожна з цих кон'юнкцій може входити в поліном із коефіцієнтом 0 або 1. Отже, кількість поліномів Жегалкіна від n змінних дорівнює кількості кортежів довжиною 2^n із компонентами 0 або 1, тобто 2^{2^n}, що дорівнює кількості всіх булевих функцій від тих самих змінних $x_1, x_2, ..., x_n$. Позаяк доведено, що будь-яку булеву функцію можна подати поліномом Жегалкіна, то звідси випливає єдиність цього подання.

Розглянемо методи побудови полінома Жегалкіна.

Метод невизначених коефіцієнтів. Для функції $f(x_1, ..., x_n)$ записують найзагальніший вигляд полінома Жегалкіна $P(x_1, ..., x_n)$ із невизначеними коефіцієнтами (їх 2^n). Зокрема, поліном від двох змінних має загальний вигляд

$$P(x, y) = c_0 \oplus c_1 x \oplus c_2 y \oplus c_3 xy,$$

а від трьох змінних –

$$P(x, y, z) = c_0 \oplus c_1 x \oplus c_2 y \oplus c_3 z \oplus c_4 xy \oplus c_5 xz \oplus c_6 yz \oplus c_7 xyz.$$

Для кожного двійкового набору $(a_1, ..., a_n)$ значень змінних записують 2^n рівнянь $f(a_1, ..., a_n) = P(a_1, ..., a_n)$. Розв'язавши їх, отримують коефіцієнти полінома $P(x_1, ..., x_n)$.

Приклад 7.18. Побудуємо поліном Жегалкіна для функції $f(x, y) = x \sim y$. Прирівняємо значення функції та полінома на всіх чотирьох наборах значень змінних і одержимо систему рівнянь відносно невизначених коефіцієнтів:

$$f(0, 0) = 1 = c_0,$$
$$f(0, 1) = 0 = c_0 \oplus c_2,$$
$$f(1, 0) = 0 = c_0 \oplus c_1,$$
$$f(1, 1) = 1 = c_0 \oplus c_1 \oplus c_2 \oplus c_3.$$

Розв'язавши її, визначаємо, що $c_0 = 1$, $c_1 = 1$, $c_2 = 1$, $c_3 = 0$ й, отже, $x \sim y = 1 \oplus x \oplus y$.

Побудова полінома Жегалкіна на основі рівносильних перетворень. Спочатку будують рівносильну формулу, у якій є лише операції кон'юнкції та заперечення, а потім всюди замінюють $\bar{x}$ на $1 \oplus x$. Після цього тривіальними перетвореннями отримують поліном Жегалкіна.

Приклад 7.19. Побудуємо поліном Жегалкіна для функції $f(x, y) = x \to y$. Використовуючи введені раніше еквівалентності, одержимо

$$x \to y = \bar{x} \vee y = \overline{x\bar{y}} = 1 \oplus x(1 \oplus y) = 1 \oplus x \oplus xy.$$

Побудова полінома Жегалкіна за ДДНФ булевої функції. Цей спосіб ґрунтується на доведенні теореми 7.5. Його доцільно застосовувати тоді, коли функцію задано ДДНФ або цю форму легко знайти.

Приклад 7.20. Побудуємо таким методом поліном Жегалкіна для функції $f(x, y, z) = xy\bar{z} \vee \bar{x}yz \vee xz$. Спочатку перетворимо цю ДНФ на досконалу:

$$xy\bar{z} \vee \bar{x}yz \vee xz = xy\bar{z} \vee \bar{x}yz \vee x(y \vee \bar{y})z = xy\bar{z} \vee \bar{x}yz \vee xyz \vee x\bar{y}z.$$

Повторивши доведення теореми 7.5, маємо

$$xy\bar{z} \vee \bar{x}yz \vee xyz \vee x\bar{y}z = xy\bar{z} \oplus \bar{x}yz \oplus xyz \oplus x\bar{y}z =$$
$$= xy(1 \oplus z) \oplus (1 \oplus x)yz \oplus xyz \oplus x(1 \oplus y)z =$$
$$= xy \oplus xyz \oplus yz \oplus xyz \oplus xyz \oplus xz \oplus xyz = xy \oplus yz \oplus xz.$$

Поліном Жегалкіна має цікаву властивість, зручну для знаходження істотних змінних функції.

ТЕОРЕМА 7.6. Усі змінні булевої функції, які входять у її поліном Жегалкіна, істотні.

Доведення. Нехай змінна x_1 уходить у поліном Жегалкіна функції $f(x_1, x_2, ..., x_n)$. Згрупуємо члени, які містять x_1, і винесемо x_1 за дужки:

$$f(x_1, x_2, ..., x_n) = x_1 f_1(x_2, ..., x_n) \oplus f_2(x_2, ..., x_n).$$

Функція $f_1(x_2, ..., x_n) \neq 0$, оскільки в протилежному випадку змінна x_1 не входила б у поліном для функції f (унаслідок єдиності полінома Жегалкіна). Нехай на наборі $(a_2, ..., a_n)$ значення функції f_1 дорівнює 1. Тоді $f(x_1, a_2, ..., a_n) = x_1 \oplus \sigma$, де $\sigma = f_2(a_2, ..., a_n)$. Отже, зміна значення x_1 тоді, коли значення решти змінних задано набором $(a_2, ..., a_n)$, змінює значення функції $f(x_1, x_2, ..., x_n)$. Звідси випливає, що змінна x_1 істотна.

7.4. Повнота й замкненість

Ми розглянули два способи подання булевих функцій — табличний і формульний. Таблиця задає функцію безпосередньо як відповідність між двійковими наборами та значеннями функції на них. Вада табличного способу подання булевих функцій — його громіздкість.

Формула — значно компактніший спосіб подання функції, проте вона задає одну функцію через інші. Тому для довільної системи функцій Q виникає запитання: чи кожну булеву функцію можна подати формулою над Q? У попередньому підрозділі було отримано позитивну відповідь на це запитання для системи $Q_0 = \{\overline{x}, xy, x \vee y\}$. Справді, будь-яку булеву функцію $f \neq 0$ можна подати досконалою ДНФ. Якщо функція f тотожно дорівнює 0, маємо таке її подання через функції системи Q_0: $0 = x\overline{x}$. У цьому підрозділі показано, як розв'язати сформульовану проблему для довільної системи булевих функцій Q.

7.4.1. Функціонально повні системи

Систему булевих функцій Q називають *функціонально повною*, якщо довільну булеву функцію можна подати формулою над Q, тобто вона являє собою суперпозицію функцій із Q.

Приклад 7.21. Наведемо приклади повних систем.

1. Як уже було доведено, система $\{\overline{x}, xy, x \vee y\}$ повна.
2. Система $\{xy, x \oplus y, 1\}$ повна. Справді, якщо функція $f \neq 0$, то подамо її поліномом Жегалкіна; якщо $f = 0$, то $0 = x \oplus x$.

Очевидно, що не кожна система булевих функцій повна. Наприклад, система $\{\overline{x}, 0, 1\}$ неповна: за її допомогою не можна подати функцію двох змінних.

Дослідження повноти одних систем можна звести до дослідження повноти інших.

ТЕОРЕМА 7.7. Нехай задано дві системи булевих функцій Q_1 і Q_2, причому система Q_1 повна, і кожну її функцію можна виразити формулою через функції системи Q_2. Тоді система Q_2 також функціонально повна.

Доведення. Нехай f — довільна булева функція. Оскільки система Q_1 повна, то функцію f можна виразити формулою F_1, яка містить скінченну кількість функцій системи Q_1. Нехай це функції $g_1, g_2, \ldots, g_k$. За умовою теореми кожну з цих функцій можна виразити формулою через скінченну кількість функцій системи Q_2; нехай це функції $h_1, h_2, \ldots, h_m$. Тому у формулі F_1 ми можемо виключити входження функцій $g_1, g_2, \ldots, g_k$, замінивши їх формулами, які містять функції $h_1, h_2, \ldots, h_m$. Одержимо формулу F_2, що містить лише функції $h_1, h_2, \ldots, h_m$. Отже, ми виразили функцію f як формулу F_2 через функції $h_1, h_2, \ldots, h_m$ системи Q_2. Позаяк функція f довільна, то система Q_2 функціонально повна.

Приклад 7.22. Розглянемо інші приклади функціонально повних систем.

3. Система $\{\overline{x}, xy\}$ повна, бо система 1 повна та $x \vee y = \overline{\overline{x}\,\overline{y}}$.
4. Система $\{\overline{x}, x \vee y\}$ повна, бо система 1 повна та $xy = \overline{\overline{x} \vee \overline{y}}$.
5. Система $\{x \mid y\}$ повна, бо система 3 повна та $\overline{x} = x \mid x$, $xy = \overline{x \mid y} = (x \mid y) \mid (x \mid y)$.

7.4.2. Замкнені класи

Множину K булевих функцій називають *замкненим класом*, якщо довільна суперпозиція функцій із K також належить K. Будь-яка система Q булевих функцій породжує замкнений клас, що складається з усіх функцій, які можна одержати суперпозиціями функцій із Q; його називають *замиканням системи функцій* Q. Замикання системи Q позначають $[Q]$. Очевидно, що коли K — замкнений клас, то $[K] = K$, а якщо Q — функціонально повна система, то $[Q] = P_2$.

Існує п'ять найважливіших замкнених класів:

- T_0 — функції, що зберігають 0;
- T_1 — функції, що зберігають 1;
- S — самодвоїсті функції;
- M — монотонні функції;
- L — лінійні функції.

Розглянемо ці класи.

Клас T_0. Булеву функцію $f(x_1, ..., x_n)$ називають *функцією, яка зберігає* 0, якщо $f(0, ..., 0) = 0$.

Наприклад, функції 0, x, xy, $x \vee y$, $x \oplus y$ зберігають 0, тобто належать класу T_0, а функції 1, $\overline{x}$, $x \to y$ — не зберігають, тобто не належать класу T_0. Таблиця значень функції з класу T_0 містить 0 у першому рядку. Отже, у класі T_0 є $2^{2^n-1} = 1/2 \cdot 2^{2^n}$ булевих функцій, що залежать від n змінних $x_1, ..., x_n$.

ТЕОРЕМА 7.8. T_0 — замкнений клас. Інакше кажучи, із функцій, що зберігають 0, суперпозицією можна одержати лише функції, які зберігають 0.

Доведення. Клас T_0 містить тотожну функцію. Отже, досить довести, що функція $F(x_1, ..., x_n) = f(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$ зберігає 0, якщо функції f, f_1, f_2, ..., f_m зберігають 0. Справді, $f(f_1(0, ..., 0), f_2(0, ..., 0), ..., f_m(0, ..., 0)) = f(0, ..., 0) = 0$.

Наслідок. Повна система функцій має містити хоча б одну функцію, яка не зберігає 0. Інакше кажучи, повна система функцій не може цілком міститись у замкненому класі T_0.

Клас T_1. Булеву функцію $f(x_1, ..., x_n)$ називають *функцією, яка зберігає* 1, якщо $f(1, ..., 1) = 1$.

Наприклад, функції 1, x, xy, $x \vee y$, $x \to y$ належать класу T_1, а функції 0, $\overline{x}$, $x \oplus y$ — ні.

ТЕОРЕМА 7.9. T_1 — замкнений клас.

Доведення. Клас T_1 містить тотожну функцію. Отже, досить довести, що функція $F(x_1, ..., x_n) = f(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$ зберігає 1, якщо функції f, f_1, f_2, ..., f_m зберігають 1. Справді, $f(f_1(1, ..., 1), f_2(1, ..., 1), ..., f_m(1, ..., 1)) = f(1, ..., 1) = 1$.

Наслідок. Повна система функцій має містити хоча б одну функцію, яка не зберігає 1. Інакше кажучи, повна система функцій не може цілком міститись у замкненому класі T_1.

Легко довести, що клас T_1 має $2^{2^n-1} = 1/2 \cdot 2^{2^n}$ функцій, що залежать від n змінних $x_1, ..., x_n$. Зазначимо, що існують булеві функції, які зберігають і 0, і 1; водночас є функції, які не зберігають ні 0, ні 1. Наприклад, xy, $x \vee y$, $x \oplus y \oplus z \in T_0 \cap T_1$, а $x \mid y$, $x \downarrow y \notin T_0 \cup T_1$.

Клас *S*. Нагадаємо означення самодвоїстої функції: функцію називають *самодвоїстою*, якщо вона двоїста до самої себе: $f^* = f$. Наприклад, функції x, $\bar{x}$, $x \oplus y \oplus z$ самодвоїсті, а функції xy, $x \vee y$, $x \to y$ — ні.

Пригадавши означення двоїстої функції, можна дати таке означення самодвоїстої функції, еквівалентне до попереднього. Булеву функцію $f(x_1, ..., x_n)$ називають *самодвоїстою*, якщо вона набуває протилежних значень на протилежних наборах значень змінних: $f(x_1, x_2, ..., x_n) = \overline{f(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)}$. Звідси, зокрема, випливає, що самодвоїсту функцію можна повністю задати її значеннями на першій половині наборів. Отже, кількість самодвоїстих функцій від n змінних $x_1, ..., x_n$ дорівнює кількості двійкових наборів довжиною 2^{n-1}, тобто $2^{2^{n-1}} = \sqrt{2^{2^n}}$.

ТЕОРЕМА 7.10. Клас S самодвоїстих функцій замкнений.

Доведення. Оскільки тотожна функція належить класу S, достатньо довести, що функція $F(x_1, ..., x_n) = f(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$ самодвоїста, якщо функції $f, f_1, f_2, ..., f_m$ самодвоїсті. Застосувавши принцип двоїстості, отримаємо:

$$F^*(x_1, ..., x_n) = f^*(f_1^*(x_1, ..., x_n), ..., f_m^*(x_1, ..., x_n)) =$$
$$= f(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)) = F(x_1, ..., x_n).$$

Наслідок. Повна система булевих функцій має містити хоча б одну несамодвоїсту функцію.

ЛЕМА 7.1 (про несамодвоїсту функцію). Із несамодвоїстої функції $f(x_1, ..., x_n)$ підстановкою функцій x та $\bar{x}$ можна отримати несамодвоїсту функцію однієї змінної, тобто константу.

Доведення. Нехай $f(x_1, ..., x_n) \notin S$. Отже, існує хоча б один набір $\tilde{a}^n = (a_1, ..., a_n)$ значень змінних такий, що $f(a_1, ..., a_n) = f(\bar{a}_1, ..., \bar{a}_n)$. За цим набором означимо допоміжні функції $\varphi_i(x)$, $i = 1, 2, ..., n$:

$$\varphi_i(x) = \begin{cases} x, & \text{якщо } a_i = 0, \\ \bar{x}, & \text{якщо } a_i = 1. \end{cases}$$

Легко переконатись, що ці функції мають властивість $\varphi_i(0) = a_i$, $\varphi_i(1) = \bar{a}_i$ ($i = 1, 2, ..., n$). Розглянемо функцію $\Phi(x) = f(\varphi_1(x), ..., \varphi_n(x))$. Її отримано з функції $f(x_1, ..., x_n)$ підстановкою x та $\bar{x}$. Функція $\Phi(x)$ — константа, бо

$$\Phi(0) = f(\varphi_1(0), ..., \varphi_n(0)) = f(a_1, ..., a_n) =$$
$$= f(\bar{a}_1, ..., \bar{a}_n) = f(\varphi_1(1), ..., \varphi_n(1)) = \Phi(1).$$

Клас *M*. Уведемо на множині E_2^n усіх n-місних двійкових наборів відношення часткового порядку. Нехай $\tilde{a}^n = (a_1, ..., a_n)$, $\tilde{b}^n = (b_1, ..., b_n)$ — двійкові набори; $\tilde{a}^n \leqslant \tilde{b}^n$, якщо $a_i \leqslant b_i$ для всіх $i = 1, 2, ..., n$. Наприклад, (010) $\leqslant$ (110), а набори (010) і (100) порівняти не можна.

Функцію $f(\tilde{x}^n) = f(x_1, ..., x_n)$ називають *монотонною*, якщо для будь-яких двійкових наборів $\tilde{a}^n$ і $\tilde{b}^n$ із того, що $\tilde{a}^n \leqslant \tilde{b}^n$, випливає, що $f(\tilde{a}^n) \leqslant f(\tilde{b}^n)$. Наприклад, 0, 1, x, xy, $x \vee y$ — монотонні функції, а $\bar{x}$, $x \oplus y$, $x \to y$ — немонотонні.

Щоб перевірити функцію на монотонність безпосередньо за означенням, потрібно проаналізувати таблицю функції, що може виявитися досить громіздкою справою. Проте часто досить легко виявити, що функція немонотонна.

ТЕОРЕМА 7.11. Нехай $f(\tilde{x}^n)$ — монотонна функція. Якщо ця функція не зберігає 0, то вона тотожно дорівнює 1. Якщо функція $f(\tilde{x}^n)$ не зберігає 1, то вона тотожно дорівнює 0.

Доведення. Для будь-якого набору $\tilde{a}^n = (a_1, ..., a_n)$ виконується умова $(0, ..., 0) \leqslant (a_1, ..., a_n)$. Далі, $1 = f(0, ..., 0) \leqslant f(a_1, ..., a_n)$. Отже, $f(\tilde{a}^n) = 1$, тобто $f(\tilde{x}^n)$ — константа 1. Друге твердження теореми можна довести аналогічно.

Наслідок. Якщо функція $f \notin T_0 \cap T_1$ — не константа, то вона немонотонна.

Отже, всі монотонні функції, окрім констант 0 і 1, належать $T_0 \cap T_1$. Зауважимо, що існують немонотонні функції, які належать $T_0 \cap T_1$, наприклад $x \oplus y \oplus z$.

ТЕОРЕМА 7.12. Клас M монотонних функцій замкнений.

Доведення. Оскільки тотожна функція належить класу M, то для доведення замкненості M досить довести, функція $F(x_1, ..., x_n) = f(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$ монотонна в разі монотонності функцій $f, f_1, ..., f_m$. Нехай $\tilde{a}^n \leqslant \tilde{b}^n$; тоді $f_i(\tilde{a}^n) \leqslant f_i(\tilde{b}^n)$ для всіх $i = 1, 2, ..., n$. Звідси випливає, що

$$F(\tilde{a}^n) = f(f_1(\tilde{a}^n), ..., f_m(\tilde{a}^n)) \leqslant f(f_1(\tilde{b}^n), ..., f_m(\tilde{b}^n)) = F(\tilde{b}^n).$$

Отже, $F(\tilde{a}^n) \leqslant F(\tilde{b}^n)$.

Наслідок. Повна система булевих функцій має містити хоча б одну немонотонну функцію.

ЛЕМА 7.2 (про немонотонну функцію). Із немонотонної функції підстановкою констант 0, 1 і функції x можна отримати функцію $\bar{x}$.

Доведення. Нехай $f(\tilde{x}^n) \notin M$. Отже, існують такі два набори $\tilde{a}_n = (a_1, ..., a_n)$, $\tilde{b}_n = (b_1, ..., b_n)$ значень змінних $x_1, ..., x_n$, що $\tilde{a}^n \leqslant \tilde{b}^n$, але $f(\tilde{a}_n) > f(\tilde{b}_n)$, тобто $f(\tilde{a}_n) = 1$, $f(\tilde{b}_n) = 0$.

Якщо $\tilde{a}_n$ та $\tilde{b}_n$ відрізняються k компонентами, то в цих компонентах у наборі $\tilde{a}_n$ є нулі, а в наборі $\tilde{b}_n$ — одиниці (бо $\tilde{a}^n \leqslant \tilde{b}^n$). Підставимо у функцію $f(\tilde{x}^n)$ замість змінних, яким у наборах $\tilde{a}_n$ та $\tilde{b}_n$ відповідають однакові значення, просто ці значення, а на місце решти k змінних — функцію x. Тоді отримаємо функцію однієї змінної $g(x)$. З урахуванням того, що $f(\tilde{a}_n) = 1$, $f(\tilde{b}_n) = 0$, одержимо $g(0) = f(\tilde{a}_n) = 1$, $g(1) = f(\tilde{b}_n) = 0$. Отже, $g(x) = \bar{x}$.

Проілюструємо процес отримання функцій $\bar{x}$ на конкретному прикладі. Нехай $f = x \to y$. Очевидно, що $x \to y \notin M$. Далі, $(0, 0) \leqslant (1, 0)$, $0 \to 0 = 1$, $1 \to 0 = 0$. Отже, $g(x) = x \to 0 = \bar{x}$.

Клас *L*. Булеву функцію $f(\tilde{x}^n) = f(x_1, x_2, ..., x_n)$ називають *лінійною*, якщо її поліном Жегалкіна має вигляд $f(\tilde{x}^n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus ... \oplus c_nx_n$. Це поліном першого степеня. або *лінійний* поліном. Він не має багатомісних кон'юнкцій x_ix_j, $x_ix_jx_k$ тощо. Коефіцієнти c_0, c_1, ..., c_n лінійного полінома можуть утворювати довільний набір значень з $n + 1$ нулів і одиниць. Унаслідок єдиності полінома Жегалкіна різним наборам коефіцієнтів відповідають різні булеві функції. Отже, є 2^{n+1} лінійних булевих функцій від n змінних. Приклади лінійних булевих функцій — 0, 1, x, $\bar{x}$, $x \sim y$, а функції $x \to y$, $x \vee y$ нелінійні.

Неважко переконатись, що серед лінійних функцій самодвоїсті ті, поліном Жегалкіна яких містить непарну кількість змінних, а несамодвоїсті ті, поліном Жегалкіна яких містить парну кількість змінних. Наприклад, функції x, $x \oplus 1$, $x \oplus y \oplus z$, $x \oplus y \oplus z \oplus 1$ самодвоїсті, а функції $x \oplus y$, $x \oplus y \oplus 1$ несамодвоїсті.

Серед лінійних функцій монотонні лише три функції: 0. 1, x.

ТЕОРЕМА 7.13. Клас *L* лінійних функцій замкнений.

Доведення. Множина *L* усіх лінійних функцій — замкнений клас, бо підстановка формул вигляду $c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus ... \oplus c_nx_n$ у формулу такого самого вигляду знову дає формулу того самого вигляду.

Наслідок. Повна система булевих функцій має містити хоча б одну нелінійну функцію.

ЛЕМА 7.3 (про нелінійну функцію). Якщо функція $f(\tilde{x}^n)$ нелінійна, то кон'юнкцію двох змінних можна подати як суперпозицію констант 0, 1, заперечення $\bar{x}$ і функції $f(\tilde{x}^n)$.

Доведення. Нехай $f \notin L$. Тоді поліном Жегалкіна функції f містить кон'юнкції змінних. Виберемо серед них кон'юнкцію з найменшим рангом $r \geqslant 2$; нехай це

$$k = x_{i_1}x_{i_2}x_{i_3}...x_{i_r}, \quad r \geqslant 2.$$

Надамо значення $x_{i_3} = ... = x_{i_r} = 1$, а всім змінним x_j, які не входять у кон'юнкцію k, надамо значення $x_j = 0$. Підстановка цих констант у поліном перетворить кон'юнкцію k на $x_{i_1}x_{i_2}$, а решту кон'юнкцій — на 0. При цьому функція f набере вигляду

$$\varphi(x_{i_1}, x_{i_2}) = x_{i_1}x_{i_2} \oplus \alpha x_{i_1} \oplus \beta x_{i_2} \oplus \gamma,$$

де α, β, γ — коефіцієнти, що дорівнюють 0 чи 1 залежно від конкретної функції $f(\tilde{x}^n)$. Розглянемо функцію

$$\begin{aligned} F(x, y) &= \varphi(x \oplus \beta, y \oplus \alpha) = \\ &= (x \oplus \beta)(y \oplus \alpha) \oplus \alpha(x \oplus \beta) \oplus \beta(y \oplus \alpha) \oplus \gamma = \\ &= xy \oplus \alpha x \oplus \beta y \oplus \alpha\beta \oplus \alpha x \oplus \alpha\beta \oplus \beta y \oplus \alpha\beta \oplus \gamma = \\ &= xy \oplus \alpha\beta \oplus \gamma. \end{aligned}$$

Отже, $F(x, y) = xy \oplus \alpha\beta \oplus \gamma$. Цю функцію отримано суперпозицією констант 0 і 1, заперечення $\bar{x} = x \oplus 1$ та функції $f(\bar{x}^n) \notin L$. Якщо $\alpha\beta \oplus \gamma = 0$, то $F(x, y) = xy$; якщо $\alpha\beta \oplus \gamma = 1$, то $\bar{F}(x, y) = xy$.

7.4.3. Критерій функціональної повноти системи булевих функцій

Розглянемо критерій функціональної повноти — теорему про функціональну повноту, доведену Е. Постом (E. Post) 1921 р.

ТЕОРЕМА 7.14. Для того щоб система булевих функцій Q була функціонально повною, необхідно й достатньо, щоб вона містила:

1) функцію, яка не зберігає 0;
2) функцію, яка не зберігає 1;
3) несамодвоїсту функцію;
4) немонотонну функцію;
5) нелінійну функцію.

Інакше кажучи, для повноти системи Q необхідно й достатньо, щоб для кожного з п'яти замкнених класів T_0, T_1, S, M, L вона містила функцію, яка цьому класу не належить.

Доведення. Необхідність випливає з того, що класи T_0, T_1, S, M, L замкнені (наслідки з теорем 7.8–7.10, 7.12, 7.13).

Достатність. Уведемо для функцій із системи Q такі позначення: f_t — функція, що не зберігає 0; f_j — функція, що не зберігає 1; f_k — несамодвоїста функція; f_m — немонотонна функція; f_l — нелінійна функція.

Доведення достатності проведемо у два етапи.

Перший етап. Одержимо константи 0 і 1. Розглянемо функцію f_t: $f_t(0, ..., 0) = 1$. Можливі два випадки.

1. Якщо $f_t(1, ..., 1) = 1$, то функція $\varphi(x) = f_t(x, ..., x)$ тотожно дорівнює 1, бо $\varphi(0) = f_t(0, ..., 0) = 1$, $\varphi(1) = f_t(1, ..., 1) = 1$. Із функції f_j у цьому разі можна отримати константу 0, оскільки $f_j(\varphi(x), ..., \varphi(x)) = f_j(1, ..., 1) = 0$.
2. Якщо $f_t(1, ..., 1) = 0$. Тоді функція $\varphi(x) = f_t(x, ..., x)$ — заперечення: $\varphi(x) = \bar{x}$, бо $\varphi(0) = f_t(0, ..., 0) = 1$, $\varphi(1) = f_t(1, ..., 1) = 0$. Із несамодвоїстої функції f_k та побудованої функції $\bar{x}$ за лемою 7.1 про несамодвоїсту функцію одержимо константу. Другу константу отримаємо, використавши $\bar{x}$, тому що $\bar{0} = 1$, $\bar{1} = 0$.

Отже, в обох випадках ми одержали константи 0 і 1.

Другий етап. Використовуючи константи 0, 1 та немонотонну функцію f_m, за лемою 7.2 про немонотонну функцію одержимо $\bar{x}$. Потім за допомогою констант 0, 1, функцій $\bar{x}$ і $f_l \notin L$ за лемою 7.3 про нелінійну функцію отримаємо кон'юнкцію двох змінних xy. Отже, через функції системи Q ми виразили $\bar{x}$ та xy. Позаяк система $\{\bar{x}, xy\}$ повна, то за теоремою 7.7 система Q також повна.

Щоб перевірити, чи виконуються для скінченної системи функцій $\{f_1, ..., f_q\}$ умови теореми Поста, складають *таблицю Поста*. Її рядки позначають функціями системи, а стовпці — назвами п'яти основних замкнених класів. У клітках таблиці Поста ставлять знак „+" або „–" залежно від того, чи належить функція відповідному замкненому класу. Для повноти системи функцій необхідно й достатньо, щоб у кожному стовпці таблиці Поста стояв хоча б один знак „–".

Приклад 7.23. Дослідимо, чи функціонально повна система $\{x \to y, 0\}$. Побудувавши таблицю Поста (табл. 7.6), переконуємось, що система функціонально повна.

Таблиця 7.6

	T_0	T_1	S	M	L
$x \to y$	–	+	–	–	–
0	+	–	–	+	+

Мінімальну повну систему функцій (тобто таку, вилучення з якої довільної функції робить систему неповною) називають *базисом*. Із теореми Поста випливає, що базис не може містити більше п'яти функцій. Насправді можна довести точніше твердження.

ТЕОРЕМА 7.15. Максимальна кількість функцій базису дорівнює 4.

Доведення. Функція $f_i \notin T_0$ або несамодвоїста, тобто $f_i(0, ..., 0) = f_i(1, ..., 1) = 1$ (випадок 1 доведення теореми 7.14), або не зберігає 1 та немонотонна (випадок 2 доведення теореми 7.14). Отже, повна або система $\{f_n, f_j, f_m, f_l\}$, або $\{f_n, f_j, f_l\}$. Константу 4 у цій теоремі зменшити неможливо. Система з чотирьох функцій $\{f_1, f_2, f_3, f_4\}$, де $f_1 = 0$, $f_2 = 1$, $f_3 = xy$, $f_4 = x \oplus y \oplus z$ повна. Із неї не можна вилучити жодної функції без порушення повноти. Справді, f_1 — єдина функція цієї системи, що не зберігає 1, f_2 — єдина функція, що не зберігає 0, f_3 — єдина нелінійна функція, f_4 — єдина немонотонна. Несамодвоїсті тут три функції: f_1, f_2, f_3.

7.4.4. Послаблена функціональна повнота

Систему булевих функцій Q називають *послаблено функціонально повною*, якщо будь-яку булеву функцію можна подати формулою над системою $Q \cup \{0, 1\}$, тобто суперпозицією констант 0, 1 і функцій із системи Q.

ТЕОРЕМА 7.16. Для того щоб система булевих функцій була послаблено функціонально повною, необхідно й достатньо, щоб вона містила хоча б одну нелінійну та хоча б одну немонотонну функцію.

Доведення. Якщо є константи 0 і 1, ми маємо функцію, що не зберігає 0 (функцію 1), функцію, що не зберігає 1 (функцію 0) і несамодвоїсту функцію (обидві функції 0 і 1). Водночас константи 0 і 1, очевидно, лінійні та монотонні. Застосування теореми Поста завершує доведення.

7.5. Мінімізація булевих функцій

Мінімізацією булевої функції називають відшукання найпростішого її подання у вигляді суперпозиції функцій якоїсь функціонально повної системи.

7.5.1. Головні результати

Розглянемо лише спрощення ДНФ, тому що за принципом двоїстості з методів спрощення ДНФ можна отримати методи спрощення КНФ.

Мінімальною ДНФ булевої функції називають її ДНФ, що складається з найменшої можливої кількості букв. При цьому кожну букву враховують стільки разів, скільки вона зустрічається в ДНФ. Наприклад, ДНФ $\bar{x}y \vee x\bar{y} \vee xz$ складається з шести букв.

Елементарну кон'юнкцію $k = x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \dots x_{i_r}^{\sigma_r}$ називають *імплікантою* булевої функції $f(x_1, \dots, x_n)$, якщо на довільному наборі значень змінних, на якому $k = 1$, значення функції f також дорівнює 1. Інакше кажучи, k — імпліканта функції f, якщо функція $k \to f$ тотожно дорівнює 1 (тобто виключено можливість $k = 1$, $f = 0$). Тут $x_{i_1}, x_{i_2}, \dots, x_{i_r}$ — якісь зі змінних $x_1, \dots, x_n$.

Приклад 7.24. Елементарна кон'юнкція $k = x\bar{y}\bar{z}$ — імпліканта функції

$$f = x\bar{y}\bar{z} \vee \bar{x}\bar{y}z,$$

бо в разі $k = 1$ значення функції f дорівнює 1.

Елементарну кон'юнкцію k називають *простою імплікантою* булевої функції f, якщо k — імпліканта функції f, а елементарна кон'юнкція, одержана з k вилученням довільної букви, — не імпліканта. Диз'юнктивну нормальну форму, що складається з усіх простих імплікант булевої функції, називають її *скороченою диз'юнктивною нормальною формою* (*СДНФ*).

ТЕОРЕМА 7.17. СДНФ $D_{\text{скор}}$ булевої функції f задає цю функцію, тобто $f = D_{\text{скор}}$.

Доведення. Якщо $f(\tilde{x}^n) = 0$, то очевидно, що функція f не має жодної простої імпліканти. Нехай тепер $f(\tilde{x}^n) \neq 0$. Розглянемо такий довільний набір $\tilde{a}^n = (a_1, \dots, a_n)$, що $f(\tilde{a}^n) = 1$. Елементарна кон'юнкція $k = x_1^{\sigma_1} x_2^{\sigma_2} \dots x_n^{\sigma_n}$, яка входить у ДДНФ функції f, — імпліканта цієї функції. Якщо імпліканта k не проста, то можна вилучити з неї хоча б одну букву так, щоб одержана елементарна кон'юнкція k_1 була імплікантою. Якщо імпліканта k_1 знову не проста, то вилучимо ще одну букву так, що отримана елементарна кон'юнкція k_2 буде імплікантою функції f. Продовжуючи цей процес, за скінченну кількість кроків ми одержимо просту імпліканту k'. За побудовою імпліканта k' набуває на наборі $\tilde{a}^n$ значення 1. Оскільки формула $D_{\text{скор}}$ складається з усіх простих імплікант функції f, то вона має диз'юнктивним членом імпліканту k', і формула $D_{\text{скор}}$ набуває значення 1 на наборі $\tilde{a}^n$.

З іншого боку, нехай формула $D_{\text{скор}}$ на якомусь наборі $\tilde{b}^n$ набуває значення 1. Тоді якась *проста імпліканта* k_0 з $D_{\text{скор}}$ на цьому наборі дорівнює 1. Оскільки k_0 — імпліканта функції f, то $f(\tilde{b}^n) = 1$.

Отже, значення функції f і формули $D_{скор}$ на будь-якому наборі значень змінних збігаються.

Зазначимо, що СДНФ булевої функції f єдина, бо множина всіх простих імплікант булевої функції визначається однозначно (адже СДНФ являє собою диз'юнкцію їх усіх).

Зв'язок між мінімальною та скороченою ДНФ виражає така теорема.

ТЕОРЕМА 7.18. Мінімальну ДНФ булевої функції f можна одержати з її СДНФ вилученням деяких елементарних кон'юнкцій.

Доведення. Потрібно довести, що мінімальна ДНФ $D_{мін}$ довільної булевої функції f являє собою диз'юнкцію її простих імплікант (можливо, не всіх). Припустимо, що імпліканта k_1 із $D_{мін}$ не проста. Тоді з неї можна вилучити хоча б одну букву так, щоб отримана елементарна кон'юнкція k_2 також була імплікантою функції f. Окрім того, імпліканта k_2 набуває значення 1 на всіх тих наборах, на яких набуває значення 1 імпліканта k_1. Отже, у ДНФ $D_{мін}$ імпліканту k_1 можна замінити на k_2 й отримати ДНФ D_*, яка також задає функцію f. За побудовою формула D_* має менше букв, ніж $D_{мін}$. Одержали суперечність.

Диз'юнктивну нормальну форму $D_{туп}$, яка задає функцію f (тобто $f = D_{туп}$), називають *тупиковою ДНФ* цієї функції, якщо:

а) кожна елементарна кон'юнкція з $D_{туп}$ – проста імпліканта функції f;

б) вилучення з формули $D_{туп}$ довільного диз'юнктивного члена призводить до ДНФ D_1, яка не задає функцію f, тобто $f \neq D_1$.

ТЕОРЕМА 7.19. Мінімальна ДНФ булевої функції являє собою її тупикову ДНФ.

Доведення цієї теореми випливає безпосередньо з означень мінімальної та тупикової ДНФ.

Існують тупикові, але не мінімальні ДНФ; одна й та сама булева функція f може мати декілька різних мінімальних ДНФ.

Із теорем 7.18 і 7.19 випливає, що відшукання мінімальних ДНФ можна поділити на два етапи:

1. Побудова скороченої ДНФ.
2. Побудова всіх тупикових ДНФ і вибір із них мінімальних.

7.5.2. Методи побудови скороченої ДНФ

Один із методів знаходження СДНФ булевої функції запропонував 1952 р. Куайн (W. Quine). Згідно з цим методом до ДДНФ булевої функції послідовно застосовують такі рівносильності:

$$ku \vee k\bar{u} = k \vee ku \vee k\bar{u} \quad \text{(неповне склеювання)},$$
$$ku \vee k = k \quad \text{(поглинання члена } ku),$$

де k – елементарна кон'юнкція, u – змінна. Говорять, що члени ku та $k\bar{u}$ *склеюються* по змінній u та в результаті дають k. Склеювання називають *неповним*, оскільки члени ku та $k\bar{u}$ залишаються в правій частині.

Алгоритм Куайна

Наведемо кроки алгоритму.

Крок 1. Булеву функцію $f(x_1, ..., x_n)$, яку потрібно мінімізувати, записати в ДДНФ; позначити її f_0. Виконати $i := 0$.

Крок 2. Якщо до ДНФ f_i не можна застосувати жодного неповного склеювання, то зупинитись: f_i — СДНФ. Інакше на основі ДНФ f_i побудувати ДНФ f_{i+1} за таким правилом: у формі f_i виконати всі неповні склеювання, які можна застосувати до елементарних кон'юнкцій із рангом $n-i$, а потім вилучити всі елементарні кон'юнкції з рангом $n-i$, до яких можна застосувати поглинання.

Крок 3. Виконати $i := i + 1$ і перейти до кроку 2.

Отже, в алгоритмі Куайна, починаючи з ДДНФ f_0 будують послідовність ДНФ $f_0, f_1, f_2, ...$ дотя, доки не отримають СДНФ.

ТЕОРЕМА 7.20. Для будь-якої булевої функції f результат застосування алгоритму Куайна до ДДНФ цієї функції — її СДНФ.

Приклад 7.25. Побудуємо СДНФ булевої функції, яку задано досконалою ДНФ f_0:

$$f_0 = \bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee xyz.$$

Виконаємо $i := 0$. Застосувавши неповне склеювання до членів 1 і 2, 2 та 5, 3 та 4, 4 та 5, одержимо

$$f_0' = \bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee xyz \vee \bar{x}\bar{y} \vee \bar{y}z \vee x\bar{y} \vee xz.$$

Після п'ятикратного застосування поглинання отримаємо $f_1 = \bar{x}\bar{y} \vee yz \vee x\bar{y} \vee xz$. Виконаємо $i := 1$. Оскільки жодне неповне склеювання не застосовне до функції f_1, то ми маємо кінцевий результат, й f_1 — СДНФ.

Мак-Класкі (E. McCluskey) удосконалив алгоритм Куайна [5].

Алгоритм Мак-Класкі

Наведемо кроки алгоритму.

Крок 1. Записати булеву функцію, яку потрібно мінімізувати, у ДДНФ.

Крок 2. Упорядкувати змінні й записати їх у кожній елементарній кон'юнкції у вибраному порядку. Після цього подати кожну елементарну кон'юнкцію послідовністю з 1, 0 і — (рисок): на i-й позиції записати 1, якщо i-та змінна входить до елементарної кон'юнкції без заперечення, 0 — якщо вона входить із запереченням, і рисочку, якщо зовсім не входить. Наприклад, елементарні кон'юнкції xyz, $x\bar{z}$, $x\bar{y}$ записують відповідно у вигляді 111–, 1–0–, 1—0.

Крок 3. Розбити двійкові вирази, які відповідають елементарним кон'юнкціям, на класи за кількістю одиниць і розмістити списки цих класів за зростанням

кількості одиниць. Наприклад, для ДДНФ $f_0 = \bar{x}\bar{y}\bar{z} \vee \bar{x}yz \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee xyz$ із прикладу 7.25 отримаємо список із п'яти елементів, розбитих на три класи:

$$\begin{array}{c} 010 \\ \hline 100 \\ \hline 011 \\ \hline 101 \\ \hline 111 \end{array}$$

Крок 4. Виконати всі можливі склеювання $k u \vee k\bar{u} = k$. Їх можна застосовувати лише до тих елементів списку, які містяться в сусідніх класах. Склеювані елементи знаходять у сусідніх класах простим порівнянням: ці елементи мають різнитися точно однією позицією, і в цій позиції має бути не риска (–).

Крок 4 повторюють доти, доки можна застосувати склеювання. Якщо помістити до одного класу всі імпліканти, отримані з двох сусідніх класів, то на черговому повторенні кроку 4 нам знову доведеться порівнювати лише елементи із сусідніх класів. Склеювані елементи позначають *; надалі вони не увійдуть у список простих імплікант. Попередній список після опрацювання має такий вигляд:

$$\begin{array}{c} {}^{*}010 \\ \hline {}^{*}100 \\ \hline {}^{*}011 \\ \hline {}^{*}101 \\ \hline {}^{*}111 \end{array} \qquad \begin{array}{c} 01- \\ \hline 10- \\ \hline -11 \\ \hline 1-1 \end{array}$$

Далі неможливо застосувати склеювання. Непозначеними залишилися чотири елементи: 01–; 10–; –11; 1–1. Отже, множина всіх простих імплікант функції f_0 така: $\{\bar{x}y, x\bar{y}, yz, xz\}$, а її СДНФ — $\bar{x}y \vee x\bar{y} \vee yz \vee xz$.

Метод Блейка. У розглянутих алгоритмах Куайна та Мак-Класкі відшукання СДНФ починається з ДДНФ. Проте часто доводиться будувати СДНФ функції, заданої довільною ДНФ (а не ДДНФ). Тоді доцільно застосувати метод Блейка (A. Blake).

Цей метод ґрунтується на використанні рівносильності *узагальненого склеювання*:

$$AC \vee B\bar{C} = AC \vee B\bar{C} \vee AB,$$

де A, B, C — довільні формули. Якщо в цій рівносильності $AB \neq 0$, то говорять, що до членів AC та $B\bar{C}$ можна застосувати *нетривіальне* узагальнене склеювання.

Доведемо рівносильність узагальненого склеювання, використовуючи закони алгебри Буля:

$$AC \vee B\bar{C} = AC \vee ABC \vee B\bar{C} \vee AB\bar{C} =$$
$$= AC \vee B\bar{C} \vee AB(C \vee \bar{C}) = AC \vee B\bar{C} \vee AB.$$

ТЕОРЕМА 7.21. Виконавши в будь-якій ДНФ булевої функції f усі можливі узагальнені склеювання, а потім — усі поглинання, одержимо СДНФ цієї функції.

Доведення ґрунтується на тому, що після багатократного застосування рівносильності узагальненого склеювання до довільної ДНФ булевої функції можна отримати будь-яку її просту імпліканту.

Метод Блейка полягає в тому, що в довільній ДНФ заданої булевої функції спочатку виконують усі допустимі узагальнені склеювання (причому одержані внаслідок узагальнених склеювань члени беруть участь у нових узагальнених склеюваннях). Після цього виконують поглинання, тобто вилучають диз'юнктивні члени у вигляді AB, якщо є диз'юнктивні члени A чи B. У результаті отримаємо СДНФ.

Приклад 7.26. Знайдемо методом Блейка СДНФ булевої функції $f = x\bar{y} \vee \bar{x}yz \vee yz$. Очевидно, що перший і другий члени формули f можна піддати узагальненим склеюванням як по x, так і по y. Але члени, які виникають унаслідок цих склеювань, дорівнюють нулю. Нетривіальне узагальнене склеювання тут можливе лише для першого та третього членів формули f. Застосувавши його, одержимо ДНФ $f_1 = x\bar{y} \vee \bar{x}yz \vee yz \vee xz$.

У цій формі нетривіальне узагальнене склеювання можна застосувати до першого та третього, а також до другого й четвертого членів, проте обидва ці склеювання дають члени, які вже є у формі f_1. Отже, можна вважати, що у формі f_1 виконано всі можливі в ній узагальнені склеювання. Виконавши елементарне поглинання (член $\bar{x}yz$ поглинається членом yz), отримаємо СДНФ $f_2 = x\bar{y} \vee yz \vee xz$.

Інше один метод побудови СДНФ розробив Нельсон (R. Nelson). Ним зручно користуватись, коли функцію задано довільною КНФ.

Метод Нельсона. Цей метод обґрунтовує така теорема.

ТЕОРЕМА 7.22. Розкривши в будь-якій КНФ булевої функції всі дужки відповідно до дистрибутивного закону й виконавши всі поглинання, одержимо СДНФ цієї функції.

Для доведення достатньо переконатись, що, розкривши дужки в довільній КНФ булевої функції f, можна отримати будь-яку наперед задану просту імпліканту k цієї функції.

Приклад 7.27. Розглянемо булеву функцію f, задану КНФ $f = (x \vee \bar{y})(\bar{x} \vee z)(x \vee y \vee \bar{z})$. Розкривши дужки, отримаємо $f = xz \vee xzy \vee \bar{x}\bar{y}\bar{z} \vee \bar{x}yz$. Виконавши поглинання, одержимо СДНФ булевої функції f:

$$f = xz \vee \bar{x}\bar{y}\bar{z}.$$

Отже, ми описали методи Куайна, Мак-Класкі, Блейка та Нельсона. Їх застосовують для отримання СДНФ булевої функції — першого етапу мінімізації.

7.5.3. Побудова тупикових ДНФ

На другому етапі мінімізації знаходять усі тупикові ДНФ, із яких вибирають мінімальні. Основний апарат для виконання другого етапу — *імплікантна таблиця*

булевої функції. Це прямокутна таблиця, рядки якої позначено різними простими імплікантами функції f, а стовпці — наборами значень змінних (або відповідними їм конституентами одиниці), на яких функція набуває значення 1. Якщо якась проста імпліканта k_p перетворюється в 1 на наборі $\tilde{a}^n$, котрий позначає якийсь стовпець імплікантної таблиці, то на перетині рядка, позначеного k_p, і стовпця, позначеного набором $\tilde{a}^n$ (або відповідною йому конституентою одиниці k), ставлять зірочку. Тоді говорять, що проста імпліканта *накриває* одиницю булевої функції. Якщо стовпці імплікантної таблиці позначено конституентами одиниці, то очевидно, що таблицю слід заповнювати за таким правилом: на перетині рядка k_p та стовпця k імплікантної таблиці тоді й лише тоді ставлять зірочку, коли імпліканта k_p становить частину конституенти k (можливо, збігається з нею).

Вище методом Куайна для функції $f = \bar{x}\bar{y}\bar{z} \vee \bar{x}yz \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee xyz$ ми знайшли СДНФ $f = \bar{x}y \vee x\bar{y} \vee xz \vee yz$. Ця функція має чотири прості імпліканти $\bar{x}y$, $x\bar{y}$, xz, yz і набуває значення 1 на п'яти наборах (010), (011), (100), (101), (111), котрим відповідають конституенти одиниці $\bar{x}y\bar{z}$, $\bar{x}yz$, $x\bar{y}\bar{z}$, $x\bar{y}z$, xyz. За сформульованим правилом будуємо імплікантну таблицю для цієї функції (табл. 7.7).

Таблиця 7.7

k_p \ k	$\bar{x}y\bar{z}$	$\bar{x}yz$	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz
$\bar{x}y$	*	*			
$x\bar{y}$			*	*	
xz				*	*
yz		*			*

Тупикові ДНФ можна будувати безпосередньо за імплікантою таблицею. Якщо в стовпці лише одна зірочка, то просту імпліканту, яка позначає рядок із цією зірочкою, потрібно вибрати обов'язково. Множину таких простих імплікант називають *ядром* булевої функції. У розглянутому прикладі ядро утворюють прості імпліканти $\bar{x}y$ та $x\bar{y}$. Імпліканти ядра входять у будь-яку тупикову ДНФ, але вони можуть накривати лише частину одиниць булевої функції. Стосовно імплікантної таблиці зручно говорити, що прості імпліканти *накривають конституенти*, які відповідають цим одиницям. Вилучимо з імплікантної таблиці стовпці, що мають зірочки на перетині з рядками, позначеними імплікантами ядра. Після цього методом перебору можна знайти мінімальні системи простих імплікант, що накривають решту конституент одиниці. Таким способом ми знайдемо всі тупикові ДНФ, із яких потрібно вибирати мінімальні. У нашому прикладі єдина конституента, що лишається не накритою імплікантами ядра, — це xyz. Її може накрити як імпліканта xz, так і імпліканта yz. Отримаємо дві тупикові ДНФ: $f_1 = \bar{x}y \vee x\bar{y} \vee xz$ і $f_2 = \bar{x}y \vee x\bar{y} \vee yz$. Обидві вони мінімальні (мають по шість букв).

Зазначимо, що метод перебору практично можна застосовувати лише для відносно простих імплікантних таблиць, а також тоді, коли потрібно знайти не всі, а тільки одну тупикову ДНФ. Для відшукання всіх тупикових ДНФ у разі складних таблиць можна застосувати метод, запропонований 1956 р. Петріком (S. Petrick).

Опишемо коротко цей метод [11].

Крок 1. Прості імпліканти позначають великими латинськими буквами. Для табл. 7.7 це має такий вигляд:

$$A - \bar{x}y; \quad B - x\bar{y}; \quad C - xz; \quad D - yz.$$

Крок 2. Для кожного стовпця імплікантної таблиці будують диз'юнкцію букв, які відповідають рядкам із зірочками в цьому стовпці.

Крок 3. Записують кон'юнкцію отриманих диз'юнкцій. Для табл. 7.7 одержимо вираз $A(A \vee D)\, B(B \vee C)(C \vee D)$. Його називають *кон'юнктивним поданням* імплікантної таблиці.

Крок 4. В отриманому на кроці 3 кон'юнктивному поданні імплікантної таблиці розкривають усі дужки за дистрибутивним законом. Одержаний вираз називають *диз'юнктивним поданням* імплікантної таблиці.

Крок 5. До отриманого на кроці 4 диз'юнктивного подання імплікантної таблиці застосовують усі можливі поглинання $A \vee AB = A$ та усувають усі повторення $AA = A$, $A \vee A = A$. Отриманий вираз називають *зведеним диз'юнктивним поданням* імплікантної таблиці. У ході його одержання з кон'юнктивного подання можна застосовувати різні перетворення згідно з тотожностями булевої алгебри, які не містять заперечень, іще до отримання звичайного (не зведеного) диз'юнктивного подання, тобто кроки 4 та 5 можна об'єднати й одразу шукати зведене диз'юнктивне подання імплікантної таблиці.

Крок 6. Прості імпліканти, позначення яких уходять у будь-який фіксований диз'юнктивний член зведеного диз'юнктивного подання імплікантної таблиці, утворюють тупикову ДНФ. Щоб отримати всі тупикові ДНФ, потрібно розглянути всі диз'юнктивні члени цього подання. Продовжуючи розгляд табл. 7.7, можемо записати:

$$A(A \vee D)\, B(B \vee C)(C \vee D) = AB(C \vee D) = ABC \vee ABD.$$

Очевидно, що кон'юнкції ABC відповідає тупикова ДНФ $\bar{x}y \vee x\bar{y} \vee xz$, а кон'юнкції ABD – $\bar{x}y \vee x\bar{y} \vee yz$.

7.5.4. Властивості скороченої ДНФ

Розглянемо важливі властивості СДНФ.

ТЕОРЕМА 7.23. Якщо СДНФ булевої функції не містить жодної букви, яка входить до неї водночас із запереченням і без заперечення, то вона являє собою мінімальну ДНФ.

Доведення. До ДНФ f, яка не має жодної букви водночас із запереченням і без заперечення, не можна застосувати рівносильність узагальненого склеювання. Це саме стосується диз'юнкції довільної кількості членів цієї форми. Якщо f – СДНФ, то її можна відновити за допомогою узагальненого склеювання з мінімальної ДНФ, що являє собою якусь частину f. Отже, у цьому разі мінімальна ДНФ збігається з СДНФ.

ТЕОРЕМА 7.24. Скорочена ДНФ монотонної функції f не містить заперечень змінних.

Доведення. Нехай k_p – проста імпліканта функції f, що містить заперечення змінних. Позначимо як k кон'юнкцію всіх змінних, що входять в імпліканту k_p без заперечень. Побудуємо набір $\tilde{a}^n = (a_1, \ldots, a_n)$ значень змінних так: усім змінним, які входять у кон'юнкцію k, надамо значення 1, а решті змінних – значення 0. Цілком очевидно, що імпліканта k_p, а, отже, і функція f має на цьому наборі значення 1. З іншого боку, кон'юнкція k перетворюється в 1 лише на наборах $\tilde{b}^n \geqslant \tilde{a}^n$ (зокрема, і на наборі $\tilde{a}^n$). Але на всіх таких наборах $\tilde{b}^n$ функція f, за означенням монотонності, також дорівнює 1. Отже, k – імпліканта функції f, а імпліканта k_p не проста. Отримали суперечність.

Наслідок. Скорочена ДНФ монотонної функції являє собою водночас і її мінімальну ДНФ.

7.5.5. Метод карт Карно побудови мінімальних ДНФ

Для відшукання мінімальних ДНФ функцій невеликої кількості змінних (не більше шести) можна застосувати метод карт Карно [52]. Цей візуальний метод запропонував 1953 р. Карно (M. Karnaugh), виходячи з дещо раніше опублікованої праці Вейча (E. Veitch). Розглянемо метод карт Карно для функцій трьох і чотирьох змінних. Карта Карно для функції трьох змінних складається з $2^3 = 8$ комірок. Кожному з наборів значень аргументів відповідає одна комірка. Кожна комірка зображає відповідну конституенту одиниці (рис. 7.1).

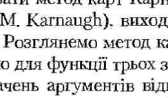


Рис. 7.1

Якщо на певному наборі значень аргументів функція дорівнює 1, то у відповідній комірці записують 1.

Приклад 7.28. Побудуємо карту Карно для функції

$$f(x, y, z) = xy\bar{z} \vee x\bar{y}\,\bar{z} \vee \bar{x}yz \vee \bar{x}\,\bar{y}\,\bar{z}.$$

На рис. 7.2 зображено комірки, які відповідають конституентам одиниці функції f, а на рис. 7.3 — заповнену карту.

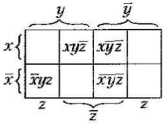


Рис. 7.2

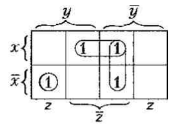


Рис. 7.3

Рівносильні склеювання $ku \vee k\bar{u} = k$ можна застосувати тільки до конституент (у загалі кажучи, до елементарних кон'юнкцій) зі змінними, у яких усі степені, окрім однієї, збігаються. Наприклад, $xyz \vee x\bar{y}z$ можна склеїти, оскільки степені x і z збігаються (x^1 та z^0), а степені змінної y різні. Такі конституенти називаються *сусідніми*.

Уважають, що в карті Карно для трьох змінних ліва та права межі тотожні (карту згорнуто в циліндр). Тоді всі сусідні конституенти мають на карті сусідні комірки. Блоку з двох сусідніх комірок відповідає елементарна кон'юнкція, що являє собою спільну частину двох конституент і містить на одну букву менше. Прямокутному блоку з чотирьох сусідніх комірок відповідає елементарна кон'юнкція, що являє собою спільну частину відповідних чотирьох конституент і містить на дві букви менше. Тому *об'єднувати можна прямокутні блоки на карті Карно, які містять сусідні дві, чотири, або — у загальному випадку — 2^i одиниць*.

Відшукання мінімальної ДНФ функції за допомогою карти Карно здійснюють так. Знаходять найбільші блоки на карті, і накривають усі одиниці найменшою кількістю блоків, першими використовуючи найбільші блоки. Можливо, це можна зробити не одним способом.

Приклад 7.28 (*закінчення*). На рис. 7.3 зображено покриття одиниць карти Карно блоками. Отримаємо мінімальну ДНФ $f_{\min} = x\bar{z} \vee \bar{y}\,\bar{z} \vee \bar{x}yz$.

Приклад 7.29. Розглянемо функцію

$$f(x,y,z) = x\bar{y}z \vee x\bar{y}\,\bar{z} \vee \bar{x}yz \vee \bar{x}\,\bar{y}z \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Карту Карно для неї зображено на рис. 7.4. Блоку з чотирьох одиниць відповідає елементарна кон'юнкція $\bar{y}$ із рангом 1, а блоку з двох одиниць — елементарна кон'юнкція $\bar{x}z$ із рангом 2. Отже, $f_{\min} = \bar{y} \vee \bar{x}z$.

Рис. 7.4

Приклад 7.30. Карту Карно для функції

$$f(x,y,z) = xyz \vee xy\bar{z} \vee x\bar{y}z \vee x\bar{y}\bar{z} \vee \bar{x}yz \vee \bar{x}\bar{y}z \vee \bar{x}\bar{y}\bar{z}$$

зображено на рис. 7.5. Мінімальна ДНФ цієї функції $f_{\min} = x \vee z \vee \bar{y}$.

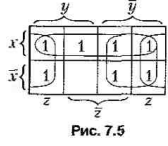


Рис. 7.5

У карті Карно для чотирьох змінних ототожнюють ліву та праву, а також верхню та нижню сторони (рис. 7.6).

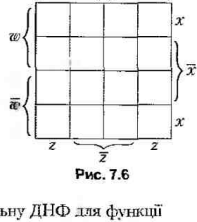


Рис. 7.6

Приклад 7.31. Знайти мінімальну ДНФ для функції

$$f(w,x,y,z) = wxy\bar{z} \vee w\bar{x}\bar{y}\bar{z} \vee \bar{w}\bar{x}\bar{y}z \vee \bar{w}\bar{x}yz \vee w\bar{x}\bar{y}\bar{z} \vee \bar{w}\bar{x}\bar{y}\bar{z} \vee \bar{w}xyz \vee \bar{w}\bar{x}\bar{y}\bar{z}.$$

Карту Карно для неї зображено на рис. 7.7. Отже,

$$f_{\min} = \bar{z} \vee \bar{w}x \vee w\bar{x}y.$$

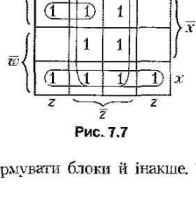


Рис. 7.7

У цьому прикладі можна сформувати блоки й інакше. Тоді одержимо іншу мінімальну ДНФ.

Інколи доводиться працювати з булевими функціями, заданими не для всіх наборів значень аргументів. Тоді в карті Карно використовують позначку d для тих комірок, які відповідають наборам із невідомим значенням булевої функції (на цих наборах функцію можна довизначити довільно). Тепер під час формування найбільших блоків можна вважати, що в деяких (або всіх) комірках із літерою d містяться одиниці.

Приклад 7.32. Задати булеву функцію чотирьох змінних, визначену на наборах, які відповідають двійковому коду десяткових цифр 0, 1, 2, ..., 9, диз'юнктивною нормальною формою, що містить найменшу кількість букв. Значення функції дорівнює 1, якщо набір відповідає цифрам, більшим або рівним 5, і дорівнює 0, якщо набір відповідає цифрам, меншим ніж 5. Шукану функцію $F(w, x, y, z)$ задано табл. 7.8. Відповідну карту Карно зображено на рис. 7.8. Отже, $F(w, x, y, z) = w \vee xy \vee xz$.

Таблиця 7.8

Цифра	w	x	y	z	$F(w,x,y,z)$
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1

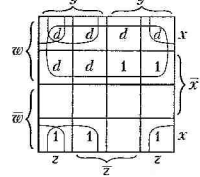


Рис. 7.8

7.6. Реалізація булевих функцій схемами з функціональних елементів

Під *функціональним елементом* розуміють пристрій із такими властивостями: він має $n \geqslant 1$ впорядкованих відростків зверху — *входів* — і один відросток знизу — *виходу*; на входи цього пристрою можуть подаватися сигнали, які мають значення 0 і 1; на кожному наборі сигналів на входах у той самий момент, коли вони надійшли, пристрій видає на виході один із сигналів (0 або 1); набір

сигналів на входах однозначно задає сигнал на виході, тобто якщо в різні моменти на входи надійшли однакові набори сигналів, то в ці моменти на виході буде один і той самий сигнал. Кожному функціональному елементу з n входами співвідносять булеву функцію від n змінних $f(\tilde{x}^n)$ таким способом. Входу з номером i $(1 \leqslant i \leqslant n)$ ставлять у відповідність змінну x_i та з кожним набором $\tilde{a}^n = (a_1, ..., a_n)$ значень змінних співвідносять число $f(\tilde{a}^n)$, яке дорівнює 0 чи 1 залежно від сигналу на виході в разі подання цього набору сигналів на входи функціонального елемента. Щодо функції $f(\tilde{x}^n)$ говорять, що даний функціональний елемент *реалізує* її.

Далі ми розглядатимемо лише функціональні елементи, зображені на рис. 7.9, які реалізують відповідно булеві функції заперечення, кон'юнкцію та диз'юнкцію.

Рис. 7.9

Тепер означимо поняття схеми з функціональних елементів і її входи й вихід.

1. Кожний функціональний елемент являє собою схему з функціональних елементів із тими самими входами та виходом, що й у цього елемента.
2. Якщо S_1 — схема з функціональних елементів і два її входи з'єднано (рис. 7.10), то одержана конструкція S являє собою схему з функціональних елементів. Входи схеми S — усі не з'єднані входи S_1 і ще один вхід, який відповідає двом з'єднаним входам схеми S_1.

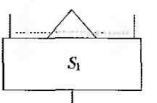


Рис. 7.10

3. Якщо S_1 і S_2 — дві схеми з функціональних елементів, то конструкція S, яку одержано з'єднанням якогось входу схеми S_2 з виходом схеми S_1, також являє собою схему з функціональних елементів (рис. 7.11). Входи схеми S — усі входи схеми S_1 і всі входи схеми S_2, окрім того, який з'єднано з виходом схеми S_1. Вихід схеми S — вихід схеми S_2.

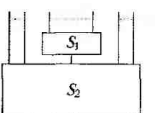


Рис. 7.11

4. Якщо в схемі з функціональних елементів S_1 її вхід з'єднати з виходом якогось функціонального елемента схеми S_1 без утворення циклу для жодного функціонального елемента (тобто вихід жодного функціонального елемента не має бути з'єднано з його ж входом — можливо, через інші елементи схеми S_1), то отримана конструкція являє собою схему з функціональних елементів (приклад такої конструкції наведено на рис 7.12).

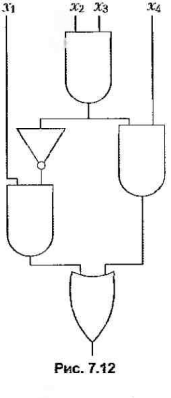


Рис. 7.12

Означення схеми з функціональних елементів завершено. Згідно з ним схема з функціональних елементів — це математичний об'єкт. Вона має різні технічні застосування.

Очевидно, що схема з функціональних елементів реалізує певну булеву функцію. Оскільки в ній допустимі з'єднання елементів, які відповідають суперпозиціям функцій системи $\{\overline{x}, x \wedge y, x \vee y\}$, а ця система повна, то будь-яку булеву функцію можна реалізувати схемою з функціональних елементів, зображених на рис. 7.9.

Під час побудови схем використовують викладені в підрозділі 7.5 методи мінімізації булевих функцій.

Приклад 7.33. Побудувати схему з функціональних елементів, яка реалізує булеву функцію

$$f = x\overline{y}z \vee x\overline{y}\,\overline{z} \vee \overline{x}yz \vee \overline{x}\,\overline{y}z \vee \overline{x}\,\overline{y}\,\overline{z}.$$

За методом карт Карно знаходимо мінімальну ДНФ (див. приклад 7.29) $f_{\min} = \overline{y} \vee \overline{x}z$. Відповідну схему наведено на рис. 7.13.

Зауважимо, що мінімізація булевих функцій не вичерпує всіх можливостей мінімізації схем. Наприклад, схема, зображена на рис. 7.12, реалізує булеву функцію $f(\tilde{x}^4) = x_1\overline{x_2x_3} \vee \vee x_2x_3x_4$. Вона має п'ять елементів. Це менше, ніж символів операцій у будь-якій формулі булевої алгебри, яка реалізує цю функцію.

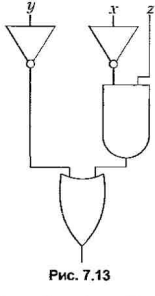


Рис. 7.13

Нехай S — схема з елементів кон'юнкції, диз'юнкції та заперечення, яка реалізує булеву функцію f, а $L_f(S)$ — кількість її елементів. Нехай також $L(f) = \min L_f(S)$, де мінімум узято за всіма схемами S, які реалізують функцію f. Нарешті, уведемо функцію $L(n) = \max L(f)$, де максимум узято за всіма булевими функціями від n змінних.

ТЕОРЕМА 7.25 (Шеннона–Лупанова). Для функції $L(n)$ виконується співвідношення

$$L(n) \sim \frac{2^n}{n},$$

причому для довільного $\varepsilon > 0$ кількість булевих функцій f, для яких $L(f) \leqslant (1-\varepsilon)\cdot 2^n/n$, прямує до 0 зі зростанням n.

Зауваження. 1. Функцію $L(n)$ називають *функцією Шеннона* (на честь американського математика К. Шеннона). 2. Символ $\sim$ означає асимптотичну рівність: $\lim\limits_{n\to\infty}(nL(n)/2^n) = 1$. 3. Зміст другого твердження теореми полягає в тому, що зі зростанням n майже всі булеві функції реалізуються зі складністю, близькою до верхньої межі, тобто $L(n)$.

Теорема 7.25 свідчить, що більшість булевих функцій (у разі $n \to \infty$) мають складні мінімальні схеми. З огляду на побудову схем практичну цінність має лише достатньо вузький клас булевих функцій. Тому разом з універсальними методами побудови схем [49] потрібно мати й такі, що призначені для певних класів булевих функцій, тому що в них краще враховано їх властивості. Розглянемо один підхід до реалізації достатньо вузького класу функцій. Покажемо, як схеми з функціональних елементів можна використовувати для додавання двох цілих додатних чисел, заданих у двійковій системі числення. Спочатку побудуємо схему, яка обчислює $x + y$, де x та y — біти. Входи схеми — x та y; на кожний із них може бути поданий сигнал 0 або 1. Наша конструкція буде об'єднанням двох схем (матиме два виходи). Вихід s дає суму бітів у даному розряді, а вихід c використано для біта перенесення в наступний розряд. У табл. 7.9 наведено значення на входах і виходах схеми, яку називають *півсуматором*. Саму схему наведено на рис. 7.14. Зазначимо, що з табл. 7.9 маємо: $c = xy$; $s = x\bar{y} \vee \bar{x}y = (x \vee y)\overline{xy}$.

Таблиця 7.9

Вхід		Вихід	
x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

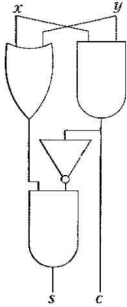


Рис. 7.14

Щоб урахувати біт перенесення c, використовують схему, яку називають *повним суматором*. Її входи — x, y і біт перенесення з попереднього розряду c_i. Виходів два: сума s у даному розряді та нове перенесення c_{i+1} (у наступний розряд). Відповідні булеві функції задано в табл. 7.10.

Таблиця 7.10

Вхід			Вихід	
x	y	c_i	s	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Подамо функції s і c_{i+1} у ДДНФ: $s = \bar{x}\bar{y}c_i \vee \bar{x}y\bar{c}_i \vee x\bar{y}\bar{c}_i \vee xyc_i$; $c_{i+1} = \bar{x}yc_i \vee x\bar{y}c_i \vee xy\bar{c}_i \vee xyc_i$. Замість того, щоб будувати повні суматори безпосередньо за зазначеними формулами, використаємо півсуматори (рис. 7.15).

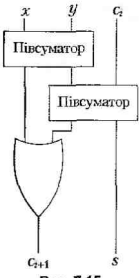


Рис. 7.15

Нарешті, на рис. 7.16 показано, як повні суматори та півсуматор можна використати для додавання трирозрядних цілих чисел $x_3x_2x_1$ та $y_3y_2y_1$ у двійковій системі числення. Результат – двійкове число $s_4s_3s_2s_1$. Зазначимо, що s_4 (старший розряд суми) збігається з c_3.

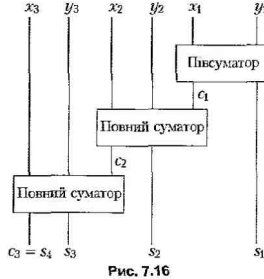


Рис. 7.16

Аналогічно будують схему для додавання n-розрядних двійкових чисел $x_nx_{n-1}...x_1$ та $y_ny_{n-1}...y_1$.

Контрольні запитання та завдання

1. Довести рівносильність наведених нижче формул F та G. Використати таблиці та тотожні перетворення.
   а) $F = \overline{x}\,\overline{z} \vee xy \vee x\overline{z}$, $G = xyz \vee \overline{z}$;
   б) $F = (x \to y) \to (x\overline{y} \oplus (x \sim \overline{y}))$, $G = (x \vee y)(\overline{x} \vee \overline{y})$;
   в) $F = xy \oplus xz \oplus yz$, $G = xy\overline{z} \vee xz \vee yz$.

2. За принципом двоїстості побудувати формулу, яка реалізує функцію, двоїсту до f:
   а) $f = xy \vee yz \vee xt \vee zt$; б) $f = x \vee y(zt \vee 0) \vee \bar{x}\bar{y}t$;
   в) $f = (\bar{x} \vee y) \oplus ((x \downarrow y) \mid (\bar{x} \sim yz))$.
3. Перейти від заданої ДНФ до ДДНФ:
   а) $f(\tilde{x}^3) = x_1 \vee \overline{x_2}x_3$; б) $f(\tilde{x}^3) = x_1\overline{x_2} \vee \overline{x_1}x_3$;
   в) $f(\tilde{x}^3) = x_1 \vee \overline{x_1}x_2 \vee \overline{x_2}x_3$; г) $f(\tilde{x}^3) = \overline{x_1}x_2 \vee x_2\overline{x_3} \vee \overline{x_1}\,\overline{x_3}$.
4. За допомогою тотожних перетворень побудувати ДДНФ:
   а) $f(\tilde{x}^3) = (x_1 \vee x_2\overline{x_3})(x_1 \vee x_3)$: б) $f(\tilde{x}^3) = (x_1 \vee \overline{x_2x_3})(x_1x_2 \vee \overline{x_1x_3})$:
   в) $f(\tilde{x}^4) = (x_1 \vee \overline{x_2})x_3x_4 \vee \overline{x_1x_4}$.
5. Перетворити ДНФ задачі 3 на КНФ.
6. Побудувати ДКНФ для функцій задачі 5.
7. Методом невизначених коефіцієнтів знайти поліном Жегалкіна для функцій:
   а) $f(\tilde{x}^3) = (11111000)$; б) $f(\tilde{x}^3) = (00110100)$; в) $f(\tilde{x}^3) = (00111001)$.
8. За допомогою тотожних перетворень побудувати поліном Жегалкіна для функцій:
   а) $f(\tilde{x}^3) = (x_1 \mid x_2) \downarrow x_3$; б) $f(\tilde{x}^3) = (x_1 \to x_2)(x_2 \downarrow x_3)$;
   в) $f(\tilde{x}^3) = ((x_1 \to x_2) \vee \overline{x_3}) \mid x_1$.
9. Скориставшись властивістю полінома Жегалкіна, знайти істотні змінні функцій:
   а) $f(\tilde{x}^2) = (x_1 \oplus x_2)(x_1 \downarrow x_2)$;
   б) $f(\tilde{x}^3) = (((x_3 \to x_2) \vee x_1)(x_2 \to x_1)x_3\overline{x_1}) \oplus x_3$;
   в) $f(\tilde{x}^3) = ((x_1 \vee x_2)(x_1 \vee \overline{x_3}) \to (\overline{x_1} \to x_2\overline{x_3}))x_2$.
10. Довести, що якщо $f(\tilde{x}^n)$ — лінійна функція, відмінна від константи, то $|N_f| = 2^{n-1}$. Чи правильне обернене твердження?
11. Чи можна з функції $f = xy \vee xz \vee yz$ за допомогою суперпозиції одержати функцію xy?
12. Довести, що серед лінійних функцій самодвоїсті ті, поліном Жегалкіна яких має непарну кількість змінних, а несамодвоїсті — парну.
13. Довести, що серед лінійних функцій монотонні лише константи 0, 1 і тотожна функція x.
14. Використовуючи критерій повноти, з'ясувати, чи функціонально повна система Q:
   а) $Q = \{x \vee y, x \oplus y, 1\}$;
   б) $Q = \{x \oplus y \oplus z, x \vee y, 0, 1\}$;
   в) $Q = \{x \vee y, \bar{x}\}$;
   г) $Q = \{x \to y, x \oplus y, 1\}$;
   д) $Q = \{xy \vee xz \vee yz, \bar{x}\}$;

е) $Q = \{x \to y, 0\}$;

ж) $Q = \{xy \vee xz \vee yz, 0, 1\}$;

и) $Q = \{x \to y, x \to \overline{yz}\}$;

к) $Q = \{x\overline{y}, \overline{x} \sim yz\}$;

л) $Q = \{0, 1, x(y \sim z) \vee \overline{x}(y \oplus z)\}$;

м) $Q = \{(01101001), (10001101), (00011100)\}$;

н) $Q = (S \setminus M) \cup (L \setminus (T_0 \cup T_1))$;

п) $Q = (S \cap M) \cup (L \setminus M) \cup (T_0 \setminus S)$;

р) $Q = (M \setminus (T_0 \cap T_1)) \cup (L \setminus S)$.

15. Які неповні системи задачі 14 слабко функціонально повні?

16. Для кожної з наведених нижче функцій методом Куайна побудувати СДНФ і за імплікантною таблицею знайти всі тупикові та мінімальні ДНФ:

а) $\overline{x}yz \vee \overline{x}\overline{y}z$;

б) $xyz \vee xy\overline{z} \vee \overline{x}yz \vee \overline{x}y\overline{z}$;

в) $xy\overline{z} \vee x\overline{y}z \vee x\overline{y}\overline{z} \vee \overline{x}yz \vee \overline{x}\overline{y}z$;

г) $xyz \vee x\overline{y}z \vee xy\overline{z} \vee \overline{x}yz \vee \overline{x}y\overline{z} \vee \overline{x}\overline{y}\overline{z}$;

д) $N_f = \{(000), (001), (011), (100), (111)\}$;

е) $N_f = \{(001), (011), (101), (110)\}$;

ж) $N_f = \{(001), (011), (100), (110)\}$;

и) $N_f = \{(000), (010), (011), (100)\}$.

17. Для кожної з функцій задачі 16 побудувати СДНФ методом Мак-Класкі.

18. Знайти мінімальні ДНФ для функцій задачі 16 методом карт Карно.

19. Для кожної з функцій, наведених нижче, побудувати СДНФ методом Мак-Класкі та знайти всі мінімальні ДНФ:

а) $wxyz \vee w\overline{x}yz \vee wx\overline{y}z \vee wxy\overline{z} \vee w\overline{x}\overline{y}z \vee w\overline{x}y\overline{z}$;

б) $wxy\overline{z} \vee wx\overline{y}z \vee w\overline{x}\overline{y}z \vee w\overline{x}y\overline{z} \vee \overline{w}xyz \vee \overline{w}x\overline{y}z \vee \overline{w}\overline{x}yz$;

в) $wxy\overline{z} \vee w\overline{x}yz \vee wx\overline{y}z \vee w\overline{x}\overline{y}z \vee \overline{w}xyz \vee \overline{w}x\overline{y}z \vee \overline{w}\overline{x}yz \vee \overline{w}\overline{x}\overline{y}z$;

г) $wxyz \vee wxy\overline{z} \vee wx\overline{y}z \vee w\overline{x}yz \vee w\overline{x}y\overline{z} \vee \overline{w}xyz \vee \overline{w}x\overline{y}z \vee \overline{w}\overline{x}yz \vee \overline{w}\overline{x}\overline{y}z$.

20. Знайти мінімальні ДНФ для функцій задачі 19 методом карт Карно.

21. Для кожної з функцій, наведених нижче, побудувати СДНФ методом Блейка:

а) $\overline{w}\overline{x} \vee w\overline{y}z \vee x\overline{y}z$;

б) $\overline{w}\,\overline{x}\,y \vee w\overline{x}z \vee yz$;

в) $wx \vee \overline{w}y \vee w\overline{x}yz \vee \overline{w}\overline{x}yz$.

22. Для кожної з функцій, наведених нижче, побудувати СДНФ методом Нельсона:

а) $(x \vee y \vee \overline{z})(\overline{x} \vee y \vee z)(\overline{y} \vee \overline{z})$;

б) $(w \vee z)(x \vee \overline{y} \vee \overline{z})(\overline{w} \vee \overline{x} \vee \overline{y})$;

в) $(w \vee \overline{x} \vee \overline{y})(\overline{w} \vee z)(w \vee y \vee \overline{z})$.

23. Побудувати схеми з функціональних елементів кон'юнкції, диз'юнкції та заперечення, які реалізують функції:

а) $xy \vee xz \vee yz$; б) $\overline{x \vee \bar{y}z}$; в) $x\bar{y}z \vee xy\bar{z} \vee \bar{x}yz$;

г) $x\bar{y}\,\bar{z} \vee \bar{x}y\bar{z} \vee \bar{x}\,\bar{y}z \vee xyz$; д) $(x \vee y)(\bar{z} \vee \bar{y})$;

е) $x \to y\bar{z}$; ж) $x \sim \bar{y}z$; и) $\bar{x}\,\bar{y} \to \bar{y} \vee \bar{z}$.

24. Побудувати схему з функціональних елементів кон'юнкції, диз'юнкції та заперечення, яка має 4 входи. На входи подаються комбінації двійкового коду десяткових цифр 0, 1, 2, ..., 9. На виході має бути 1 в тому й лише в тому разі, коли комбінації на входах відповідають одноцифровим числам:

а) непарним;

б) таким, що не діляться на 3;

в) таким, що не дорівнюють 4, 5, 6.

Скористатись методом карт Карно.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Булеву функцію n змінних задано таблицею. Побудувати її ДДНФ.
2. Булеву функцію n змінних задано таблицею. Побудувати її ДКНФ.
3. Булеву функцію n змінних задано таблицею. Побудувати формулу, яка подає її лише за допомогою операцій $\neg$ та $\wedge$.
4. Булеву функцію n змінних задано таблицею. Побудувати формулу, яка подає її лише за допомогою операцій $\neg$ та $\vee$.
5. Булеву функцію n змінних задано таблицею. Побудувати формулу, яка подає її лише за допомогою операції $|$.
6. Булеву функцію n змінних задано таблицею. Побудувати формулу, яка подає її функцію лише за допомогою операції $\downarrow$.
7. Булеву функцію трьох змінних задано таблицею. Побудувати її карту Карно.
8. Булеву функцію чотирьох змінних задано таблицею. Побудувати її карту Карно.
9. Булеву функцію n змінних задано таблицею. Побудувати її СДНФ методом Мак-Класкі.

Розділ 8

Мови, граматики й автомати

- Мови
- Формальні породжувальні граматики
- Типи граматик (ієрархія Хомскі)
- Дерева виведення
- Форми Бекуса–Наура
- Скінченні автомати з виходом
- Скінченні автомати без виходу
- Подання мов

У цьому розділі розглянуто головні поняття теорії формальних мов і теорії формальних граматик, показано зв'язок між граматиками й автоматами.

Формальні мови та граматики мають велике значення в побудові й реалізації мов програмування. Скінченні автомати й такі тісно пов'язані з ними конструкції, як, наприклад, регулярні граматики та регулярні вирази, належать до найважливіших понять інформатики. Різні варіанти скінченних автоматів використовують для опису й аналізу технічних пристроїв, різних систем і процесів, програм і алгоритмів. На базі теорії скінченних автоматів сформовано багато складних концепцій теоретичної інформатики. Ця теорія має чимало застосувань у технічній інформатиці та становить важливу частину теоретичної інформатики.

Ми розглядатимемо скінченні автомати як абстрактні моделі найпростіших пристроїв опрацювання даних. Спосіб викладення орієнтовано передусім на теорію формальних мов.

8.1. Мови

Буква (або *символ*) — це простий неподільний знак; множина букв утворює *алфавіт V*. Позаяк алфавіт — це множина, до нього можна застосовувати теоретико-множинні позначення.

Ланцюжок над алфавітом V — це впорядкована сукупність букв алфавіту. Це елемент множини $V^n = V \times V \times ... \times V$, проте в цьому розділі буде природніше записувати ланцюжки у вигляді $a_1a_2...a_n$, а не $(a_1, a_2, ..., a_n)$. Буква — це ланцюжок дов-

жиною 1. Ми будемо розглядати також *порожній ланцюжок*, який не містить букв, і позначатимемо його як λ. Зазначимо, що λ не символ, тобто $\lambda \notin V$ для будь-якого алфавіту V.

За аналогією з лінгвістикою ланцюжки іноді називають *словами*. Множину всіх ланцюжків над алфавітом V називають *замиканням* цього алфавіту й позначають V^*:

$$V^* = V^0 \cup V^1 \cup V^2 \cup \ldots = \bigcup_{n=0}^{\infty} V^n,$$

де $V^0 = \{\lambda\}$.

Головна операція над ланцюжками — конкатенація. Нехай σ й τ — ланцюжки над алфавітом V. *Конкатенацією* σ й τ називають ланцюжок ρ = στ (тобто до ланцюжка σ дописано ланцюжок τ). Якщо ланцюжок складається з повторюваних символів, то використовують скорочені позначення: для $a \in V$ та цілого невід'ємного числа n записують

$$a^0 = \lambda; \quad aa = a^2; \quad aa^{n-1} = a^n.$$

Зазначимо, що є альтернативний набір термінів для букви, алфавіту й слова — відповідно *слово*, *словник* і *речення*. У деяких контекстах ці терміни природніші, тоді особливу увагу потрібно звертати на використання терміна „слово", бо він двозначний.

Множину ланцюжків називають мовою. Формально *мова* L над алфавітом V — це множина ланцюжків із V^*, тому $L \subset V^*$. Правила, що задають множину речень, утворюють *синтаксис* мови, а опис множини змістів і відповідності між реченнями та змістами — її *семантику*. Семантика мови залежить від характеру описуваних нею об'єктів, засоби її вивчення для різних типів мов різні. Семантика мови математики — формальні теорії. Дослідження семантики мов програмування стало окремою частиною теоретичного програмування. Спроби точно описувати семантику природних мов стосуються передусім машинного перекладу. Що ж до синтаксису, то його особливості значно менше залежать від призначення мови. Можна сформулювати поняття й методи дослідження синтаксису, які не залежать від змісту та призначення мов. Тому найбільших успіхів математична лінгвістика досягла у вивченні синтаксису, де із середини XX ст. розвинувся спеціальний математичний апарат — теорія формальних породжувальних граматик. Вона дуже важлива як така й ефективна в застосуваннях (мовах програмування, штучному інтелекті, машинному перекладі).

Зосередимо увагу на тому, як можна складати слова в речення, і зазначимо, що множина всіх речень, які мають зміст, утворює мову. Нас будуть цікавити здебільшого формальні мови, такі як мови програмування чи мови, що описують правильні математичні вирази. Спочатку наведемо приклад із природної мови [10].

Приклад 8.1. Розглянемо речення „маленька дівчинка нагодувала котика". Проаналізуємо його синтаксис. Розглянемо діаграму на рис. 8.1. Вона означає, що (речення) можна побудувати за допомогою злиття (групи підмета) й (групи присудка), хоча це потребує

формального означення. Група підмета складається з ⟨означення⟩ та ⟨підмета⟩, а група присудка — із ⟨присудка⟩ та ⟨додатка⟩. Остаточно отримаємо ⟨означення⟩ „маленька". ⟨підмет⟩ „дівчинка", ..., що дає нам речення „маленька дівчина нагодувала котика".

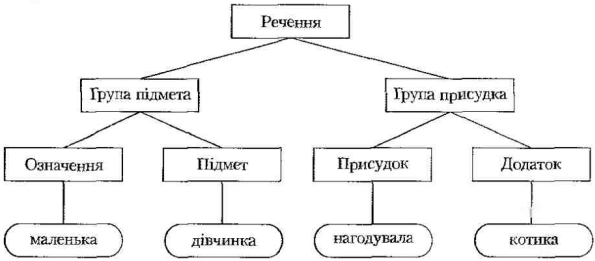


Рис. 8.1

Перш ніж увести термінологію та позначення, потрібні для уточнення загальних понять у конкретній ситуації, яка зображена на рис. 8.1, зазначимо основні задачі теорії мов.

Нагадаємо, що для заданого алфавіту V мова L — довільна підмножина множини V^*, проте довільні підмножини становлять незначний інтерес. Ми хочемо зосередити увагу на спеціальних мовах, що містять ланцюжки, які завдяки зовнішній інформації про їх семантику можна вважати осмисленими чи добре сконструйованими.

Найцікавіші мови нескінченні й, отже, їх неможливо виписати явно. Потрібно придумати способи породження таких мов; як породжувальну систему можна розглядати граматику G. Сформулюємо дві основні задачі формальної теорії мов.

1. Як за заданою граматикою G (і пов'язаною з нею мовою L) породжувати речення $\alpha \in L$?
2. Як за заданими мовою $L \subset V^*$ та реченням $\alpha \in V^*$ з'ясувати, чи $\alpha \in L$?

Щоб перевірити, чи належить якийсь ланцюжок (рченння) мові L, потрібно знати, як граматика G породжує L. Далі опишемо загальні принципи породжувальних граматик.

8.2. Формальні породжувальні граматики

У лінгвістиці природних мов терміни „речення" та „слово" мають різний зміст; тому в математичній лінгвістиці послідовність символів зазвичай називають нейтральним терміном „ланцюжок", а мову, яку розуміють як множину формальних ланцюжків, — формальною.

Формальна породжувальна граматика G (далі — *граматика* G) — це формальна система, задана четвіркою об'єктів $G = (V, T, S, P)$, де V — скінченна непорожня множина, яку називають *алфавітом* (або *словником*); T — її підмножина, елементи якої називають *термінальними* (*основними*) символами; S — *початковий символ* ($S \in V$), P — скінченна множина *продукцій* (або *правил перетворення*) вигляду $\xi \to \eta$, де ξ та η — ланцюжки над алфавітом V.

Множину $V \setminus T$ позначають N, її елементи називають *нетермінальними* (*допоміжними*) символами.

Формальні породжувальні граматики часто називають *граматиками з фразовою структурою*, *граматиками безпосередніх складових* [52, 9]. Термінальні символи часто називають *терміналами*, а нетермінальні — *нетерміналами*.

У теорії формальних граматик усталилися традиції позначень, яких ми будемо дотримуватись. Символи термінального алфавіту позначають малими латинськими буквами чи цифрами, символи нетермінального алфавіту — великими латинськими буквами, ланцюжки над алфавітом V — грецькими буквами. Довжину ланцюжка α позначають $l(\alpha)$ або $|\alpha|$. Нагадаємо, що множину всіх ланцюжків у алфавіті V позначають V^*.

Нас цікавитимуть ланцюжки, які можуть бути породжені продукціями граматики.

Нехай $G = (V, T, S, P)$ — граматика, і нехай $\alpha_0 = \sigma\xi\tau$, $\xi \neq \lambda$ (тобто α_0 — конкатенація ланцюжків σ, ξ й τ), $\alpha_1 = \sigma\eta\tau$ — ланцюжки над V. Якщо $\xi \to \eta$ — продукція граматики G, то говорять, *що* α_1 *безпосередньо виводиться з* α_0, і записують $\alpha_0 \Rightarrow \alpha_1$.

Якщо $\alpha_0, \alpha_1, \ldots, \alpha_n$ — ланцюжки над алфавітом V такі, що $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$, то говорять, *що* α_n *виводиться з* α_0, і використовують запис $\alpha_0 \overset{*}{\Rightarrow} \alpha_n$.

Послідовність кроків для отримання α_n з α_0 називають *виведенням*.

Приклад 8.2. Речення українською мовою, наведене в прикладі 8.1, можна вивести в граматиці G, де

$$V = \{\langle\text{речення}\rangle, \langle\text{група підмета}\rangle, \langle\text{група присудка}\rangle, \langle\text{означення}\rangle, \langle\text{підмет}\rangle, \langle\text{присудок}\rangle, \langle\text{додаток}\rangle, \text{маленька, дівчинка, нагодувала, котика}\},$$
$$T = \{\text{маленька, дівчинка, нагодувала, котика}\},$$
$$\text{початковий символ } \langle\text{речення}\rangle,$$

а множина P містить такі продукції:

$$\langle\text{речення}\rangle \to \langle\text{група підмета}\rangle\ \langle\text{група присудка}\rangle,$$
$$\langle\text{група підмета}\rangle \to \langle\text{означення}\rangle\ \langle\text{підмет}\rangle,$$
$$\langle\text{група присудка}\rangle \to \langle\text{присудок}\rangle\ \langle\text{додаток}\rangle,$$
$$\langle\text{означення}\rangle \to \text{маленька},$$
$$\langle\text{підмет}\rangle \to \text{дівчинка},$$
$$\langle\text{присудок}\rangle \to \text{нагодувала},$$
$$\langle\text{додаток}\rangle \to \text{котика}.$$

Ця система породжує тільки одне речення, тому її можна замінити на

$$L = \{\text{маленька дівчинка нагодувала котика}\}.$$

Якщо ми захочемо розширити мову, щоб увести до неї речення з означеннями „сусідська", „весела", присудком „попестила", і додатками „цуцика", „кролика" (тоді мова L матиме вісімнадцять речень), то це можна зробити відповідним розширенням алфавіту та додаванням лише п'яти продукцій до множини P.

Приклад 8.3. Розглянемо граматику $G = (V, T, S, P)$. Нехай $V = \{a, b, A, B, S\}$, $T = \{a, b\}$, S — початковий символ, $P = \{S \to ABa,\ A \to BB,\ B \to ab,\ AB \to b\}$. Ланцюжок $Aaba$ безпосередньо виводиться з ABa в цій граматиці, оскільки $B \to ab$ — продукція граматики. Ланцюжок $abababa$ виводиться з ланцюжка ABa, бо

$$ABa \Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$$

за допомогою послідовного використання продукцій $B \to ab$, $A \to BB$, $B \to ab$ та $B \to ab$.

Мовою, породжуваною граматикою $G = (V, T, S, P)$, називають множину всіх ланцюжків терміналів, які виводяться з початкового символу S. Її позначають $L(G)$. Отже,

$$L(G) = \{\omega \in T^* \mid S \overset{*}{\Rightarrow} \omega\},$$

де T^* — множина всіх ланцюжків терміналів, включаючи порожній ланцюжок.

Приклад 8.4. Нехай G — граматика з алфавітом $V = \{S, A, a, b\}$, множиною терміналів $T = \{a, b\}$, початковим символом S і множиною продукцій $P = \{S \to aA,\ S \to b,\ A \to aa\}$. Знайдемо мову $L(G)$, породжувану цією граматикою.

Із початкового символу S можна вивести ланцюжок aA за допомогою продукції $S \to aA$ чи застосувати продукцію $S \to b$, щоб вивести b. З aA, скориставшись продукцією $A \to aa$, можна вивести ланцюжок aaa. Ніяких інших ланцюжків вивести неможливо. Отже, $L(G) = \{b, aaa\}$.

Приклад 8.5. Нехай G — граматика з алфавітом $V = \{S, 0, 1\}$, множиною терміналів $T = \{0, 1\}$, початковим символом S і множиною продукцій $P = \{S \to 11S, S \to 0\}$. Знайдемо $L(G)$.

Із початкового символу S одержимо 0 (за допомогою другої продукції) чи $11S$ (за допомогою першої). З $11S$ можна отримати 110 або $1111S$, з $1111S$ виводиться 11110 або $111111S$. Отже, після кожного виведення ми додаємо дві одиниці в кінець ланцюжка чи закінчуємо ланцюжок нулем, тобто $L(G) = \{0, 110, 11110, 1111110, \ldots\}$ — це множина всіх ланцюжків із парною кількістю тільки 1, після яких (у кінці) обов'язково є один 0.

Зауваження. Ми отримали нескінченну мову ($L(G)$ складається з нескінченної кількості ланцюжків). Щоб граматика G породжувала нескінченну мову, у множині продукцій має бути принаймні одне рекурсивне правило (у прикладі 8.5 $S \to 11S$).

Дуже важлива проблема побудови граматики для заданої мови.

Приклад 8.6. Знайдемо граматику, яка породжує мову $\{0^m 1^m \mid m = 0, 1, 2, \ldots\}$. Потрібні дві продукції, щоб побудувати ланцюжок із якоїсь кількості 0, після яких є така сама кількість 1. Перша продукція збільшує ланцюжок ізсередини, додаючи зліва 0, а справа -1. Друга продукція замінює початковий символ S на порожній ланцюжок λ. Одержимо граматику $G = (V, T, S, P)$, де $V = \{0, 1, S\}$, $T = \{0, 1\}$, S — початковий символ, $P = \{S \to 0S1,\ S \to \lambda\}$.

Приклад 8.7. Знайти граматику, яка породжує мову $\{0^m1^n | m, n = 0, 1, 2, ...\}$.
Наведемо дві такі граматики:

G_1: $V = \{S, 0, 1\}$, $T = \{0, 1\}$, $P = \{S \to 0S, S \to S1, S \to \lambda\}$;
G_2: $V = \{S, A, 0, 1\}$, $T = \{0, 1\}$, $P = \{S \to 0S, S \to 1A, S \to 1, A \to 1A, A \to 1, S \to \lambda\}$.

Пропонуємо обґрунтувати, що обидві граматики справді породжують мову $\{0^m1^n | m, n = 0, 1, 2, ...\}$ (див. задачі 3, 4). Цей приклад свідчить, що дві різні граматики можуть породжувати одну мову.

Граматики G_1 і G_2 називають *еквівалентними*, якщо $L(G_1) = L(G_2)$.

Граматики G_1 і G_2 з прикладу 8.7 сквівалентні.

Іноді мову, яку легко описати, доводиться задавати досить складною граматикою.

Приклад 8.8. Побудуємо граматику, яка породжує мову $\{0^m1^m2^m | m = 0, 1, 2, ...\}$.
Розв'язок цієї задачі — така граматика: $G = (V, T, S, P)$, де $V = \{0, 1, 2, S, A, B\}$, $T = \{0, 1, 2\}$, початковий символ — S, множина продукцій

$$P = \{S \to 0SAB,\ S \to \lambda,\ BA \to AB,\ 0A \to 01,\ 1A \to 11,\ 1B \to 12,\ 2B \to 22\}$$

8.3. Типи граматик (ієрархія Хомскі)

Продукція граматики дає змогу заміняти одну послідовність символів іншою. Граматики класифікують за типами продукцій. Розглянемо класифікацію, яку запропонував американський математик і лінгвіст Н. Хомскі (N. Chomsky) (табл. 8.1) [52].

Таблиця 8.1

Тип граматики	Обмеження на продукції $\xi \to \eta$
0	Немає обмежень
1	$\lvert\xi\rvert \le \lvert\eta\rvert$ чи $\eta = \lambda$
2	$\xi = A$, де A — нетермінальний символ
3	$\xi = A$, причому $\eta = aB$ чи $\eta = a$, де A, B — нетермінальні символи, a — термінальний символ, або $S \to \lambda$

Принцип цієї класифікації полягає в тому, що на продукції накладено певні обмеження.

У граматиці *типу 0* немає жодних обмежень на продукції. Граматика *типу 1* може мати продукції лише у вигляді $\xi \to \eta$, де довжина ланцюжка η не менша, ніж довжина ланцюжка ξ, або у вигляді $\xi \to \lambda$. У граматиці *типу 2* можуть бути лише продукції $A \to \eta$, де A — нетермінальний символ. Граматика *типу 3* може мати такі продукції: $A \to aB$, $A \to a$, $S \to \lambda$, де A, B — нетермінали, a — термінал.

Із цих означень випливає, що кожна граматика типу n, де $n = 1, 2, 3$, являє собою граматику типу $n - 1$. Граматики типу 2 називають *контекстно вільними*, бо нетермінал A в лівій частині продукції $A \to \eta$ можна замінити ланцюжком η в довільному оточенні щоразу, коли він зустрічається, тобто незалежно від контексту. Мову, яку породжує граматика типу 2, називають *контекстно вільною*.

Якщо в множині продукцій P є продукція вигляду $\gamma\mu\delta \to \gamma\nu\delta$ (але не $\mu \to \nu$), $|\mu| \leqslant |\nu|$, то граматику називають граматикою типу 1, або контекстно залежною, оскільки μ можна замінити на ν лише в оточенні ланцюжків $\gamma...\delta$, тобто у відповідному контексті. Мову, яку породжує граматика типу 1, називають *контекстно залежною*. Граматику типу 3 називають *регулярною*. У ній можуть бути лише продукції $A \to aB$, $A \to a$, $S \to \lambda$, де A, B — нетермінали, a — термінал. Мову, яку породжує граматика типу 3, називають *регулярною*. У підрозділі 8.8 розглянуто зв'язок між регулярними мовами та скінченними автоматами. На рис 8.2 наведено діаграму, яка показує співвідношення між граматиками різних типів.

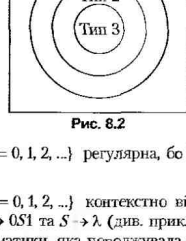


Рис. 8.2

Приклад 8.9. Мова $\{0^m1^n | m, n = 0, 1, 2, ...\}$ регулярна, бо її породжує регулярна граматика G_2 з прикладу 8.7.

Приклад 8.10. Мова $\{0^m1^m | m = 0, 1, 2, ...\}$ контекстно вільна, оскільки вона породжена граматикою з продукціями $S \to 0S1$ та $S \to \lambda$ (див. приклад 8.6). Проте ця мова не регулярна: не існує регулярної граматики, яка породжувала б цю мову. Це потрібно довести окремо (див. приклад 8.37).

Приклад 8.11. Мова $\{0^m1^m2^m | m = 0, 1, 2, ...\}$ контекстно залежна, тому що вона може бути породжена граматикою типу 1 (див. приклад 8.8). Проте вона не може бути породжена жодною граматикою типу 2; це треба довести окремо (див. приклад 8.38).

Приклад 8.12. Приклад контекстно вільної мови — мова булевих формул зі змінними a, b, c, яку породжує контекстно вільна граматика $G = (V, T, S, P)$, де $V = \{S, a, b, c, \vee, \wedge, \neg, (,)\}$, $T = \{a, b, c, \vee, \wedge, \neg, (,)\}$, а множина продукцій

$$P = \{S \to (S \vee S),\ S \to (S \wedge S),\ S \to \neg S,\ S \to a,\ S \to b,\ S \to c\}.$$

Приклад 8.13. Замінивши в граматиці з прикладу 8.12 перші три правила, які вводять операції $\vee$, $\wedge$ та $\neg$, чотирма правилами, що вводять операції $+$, $-$, $*$ та $/$, отримаємо контекстно вільну граматику, яка породжує мову арифметичних виразів. Ця мова відрізняється від звичайної мови арифметичних виразів тим, що не враховує асоціативності додавання та множення, а також пріоритетів операцій, тому її вирази містять забагато дужок. Аналогічне зауваження стосується й мови з прикладу 8.12. Ближчу до звичайної мову арифметичних виразів зі змінними x, y, z можна задати складнішою контекстно вільною граматикою. Її алфавіти $V = \{S, R, K, M, a, b, c, +, -, *, /, (,)\}$, $T = \{a, b, c, +, -, *, /, (,)\}$, а множина продукцій P містить такі правила: $S \to R$, $S \to S + R$, $S \to S - R$, $R \to M$, $R \to R * M$, $R \to R/M$, $M \to (S)$, $M \to K$, $K \to a$, $K \to b$, $K \to c$. Щоб ця мова повністю збігалася зі звичайною мовою арифметичних виразів, до цієї граматики потрібно додати правила, які породжують числові константи й довільні ідентифікатори змінних. Залишаємо це як вправу читачеві.

8.4. Дерева виведення

У мовах, породжених контекстно вільними граматиками, виведення можна зображати графічно за допомогою орієнтованих кореневих дерев. Їх називають *деревами виведення*, або *деревами синтаксичного розбору* [21, 52]. Кореню цього дерева відповідає початковий символ, внутрішнім вершинам – нетермінальні символи, що зустрічаються у виведенні, листкам – термінальні символи. Нехай γ – ланцюжок і $A \to \gamma$ – продукція, використана у виведенні. Тоді вершина, що відповідає нетерміналному символу A, має синами вершини, які відповідають кожному символу ланцюжка γ в порядку зліва направо.

Приклад 8.14. Визначимо, чи ланцюжок *cbab* належить мові, породженій граматикою $G = (V, T, S, P)$, де $V = \{a, b, c, A, B, C, S\}$, $T = \{a, b, c\}$, S – початковий символ, а множина продукцій

$$P = \{S \to AB,\ A \to Ca,\ B \to Ba,\ B \to Cb,\ B \to b,\ C \to cb,\ C \to b\}.$$

Розв'язати цю задачу можна двома способами.

1. *Розбір зверху вниз.* Оскільки є лише одна продукція з початковим символом S у лівій частині, то почнемо виведення з $S \Rightarrow AB$. Далі використаємо продукцію $A \to Ca$. Отже, $S \Rightarrow AB \Rightarrow CaB$. Позаяк ланцюжок *cbab* починається із символів *cb*, то, використавши продукцію $C \to cb$, одержимо $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB$. Завершуємо виведення застосуванням продукції $B \to b$:

$$S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab.$$

   Отже, ланцюжок *cbab* належить мові $L(G)$.

2. *Розбір знизу вверх.* Почнемо з ланцюжка *cbab*, який потрібно вивести. Можна використати продукцію $C \to cb$; отже, $Cab \Rightarrow cbab$. Застосувавши продукцію $A \to Ca$, отримаємо $Ab \Rightarrow Cab \Rightarrow cbab$. Тепер використаємо продукцію $B \to b$; отже, $AB \Rightarrow Ab \Rightarrow$ $\Rightarrow Cab \Rightarrow cbab$. Нарешті, застосуємо продукцію $S \to AB$:

$$S \Rightarrow AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab.$$

Дерево виведення для рядка *cbab* в граматиці G зображено на рис. 8.3.

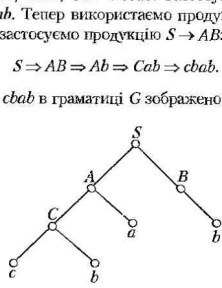


Рис. 8.3

Виведення називають *еквівалентними*, якщо їм відповідають однакові дерева. Отже, у прикладі 8.14 ми одержали еквівалентні виведення. Перевага дерева виведення порівняно з виведенням – його компактність. Дерево на рис. 8.3 має вісім вершин, а відповідні виведення – 18 або 16 символів.

Взаємно однозначної відповідності між ланцюжками мови L і деревами виведення в граматиці G, яка породжує L, може й не бути. Контекстно вільну граматику G називають *неоднозначною*, якщо існує хоча б один ланцюжок у мові $L(G)$, який має в G більше одного дерева виведення.

Приклад 8.15. Розглянемо граматику, отриману з граматики, яку задано в прикладі 8.13, ототожненням усіх нетермінальних символів і вилученням усіх тривіальних правил $S \to S$, що виникнуть у такому разі. Одержимо граматику з такою множиною правил:

$$S \to S+S,\ S \to S-S,\ S \to S*S,\ S \to S/S,\ S \to (S),\ S \to a,\ S \to b,\ S \to c.$$

Ця граматика еквівалентна до вихідної граматики; окрім того, виведення в ній суттєво коротші, а дерева мають значно меншу висоту. Проте ця граматика неоднозначна. Вираз $a * b + c$ має в ній два дерева виведення (рис. 8.4).

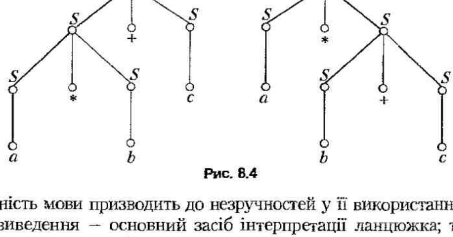


Рис. 8.4

Неоднозначність мови призводить до незручностей у її використанні. Річ у тому, що дерево виведення – основний засіб інтерпретації ланцюжка; тому синтаксична неоднозначність (тобто наявність декількох дерев виведення) ланцюжка спричинює її семантичну неоднозначність — наявність різних інтерпретацій. Наприклад, для ланцюжка $a * b + c$ з прикладу 8.15 різні дерева виведення інтерпретовано як різні способи розставлення дужок: $(a * b) + c$ в першому випадку й $a * (b + c)$ в другому. Це зумовлює різну послідовність операцій і, відповідно, різні результати обчислень. Щоб забезпечити однозначність, достатньо явно ввести дужки після кожної неодномісної операції в мовах формул (логічних, арифметичних, алгебричних тощо). Граматика з дужками (див. приклад 8.12) хоча й призводить до надлишкових дужок у формулах, однак дає змогу зменшити кількість нетермінальних символів і тим самим спростити синтаксичну структуру формули, задану її деревом.

8.5. Форми Бекуса–Наура

Для граматик типу 2 (контекстно вільних), окрім звичайного, є й інший спосіб подання — *форми Бекуса–Наура*. У лівій частині продукцій граматик типу 2 один символ (нетермінальний). Замість того щоб виписувати окремо всі продукції, можна об'єднати в один вираз продукції з однаковими символами в лівій частині. Тоді замість символу $\to$ в продукціях використовують символ ::=.

Усі нетермінали в цьому разі беруть у трикутні дужки ⟨ ⟩. Праві частини продукцій в одному виразі відокремлюють одну від одної символом |.

Приклад 8.16. Три продукції $A \to Aa$, $A \to a$ та $A \to AB$ можна подати одним таким виразом у формі Бекуса–Наура: $\langle A \rangle ::= \langle A \rangle a \mid a \mid \langle A \rangle \langle B \rangle$.

Приклад 8.17. Знайдемо продукції граматики, якщо у формі Бекуса–Наура їх записано так:

$$\langle expression \rangle ::= (\langle expression \rangle) \mid \langle expression \rangle + \langle expression \rangle \mid \langle expression \rangle * \langle expression \rangle \mid \langle variable \rangle;$$
$$\langle variable \rangle ::= x \mid y.$$

Зобразимо дерево виведення в цій граматиці для ланцюжка $(x * y) + x$.

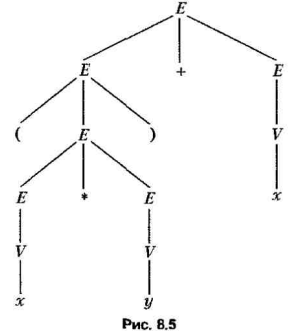


Рис. 8.5

Для зручності використаємо позначення E для ⟨*expression*⟩ (це початковий символ) і V для ⟨*variable*⟩. Тоді правила перетворення (продукції граматики) $-E \to (E)$, $E \to E + E$, $E \to E * E$ та $E \to V$ з першого виразу, а також $V \to x$ і $V \to y$ із другого. Дерево виведення для ланцюжка $(x * y) + x$ зображено на рис. 8.5.

8.6. Скінченні автомати з виходом

Скінченним автоматом називають систему $M = (S, I, O, f, g, s_0)$, у якій S, I, O — скінченні множини, а f: $S \times I \to S$ і g: $S \times I \to O$ — функції, означені на декартовому добутку $S \times I$. Множину S називають *множиною станів*, I — *вхідним алфавітом*, O — *вихідним алфавітом*, f — *функцією переходів*, g — *функцією виходів*, виділений елемент $s_0 \in S$ — *початковим станом*.

Елементи вхідного алфавіту називають *вхідними символами*, або *входами*, а вихідного — *вихідними символами*, або *виходами*. Рівність $f(s_i, x) = s_j$ означає, що в разі входу x автомат, який перебуває в стані s_i, переходить у стан s_j, а рівність $g(s_i, x) = y$ — що в цьому разі на виході з'являється y; тут $s_i, s_j \in S$, $x \in I$, $y \in O$.

Оскільки функції f і g означено на скінченних множинах, то їх можна задавати таблицями. Зазвичай дві таблиці зводять у *таблицю станів*, або *автоматну таблицю*. Вона містить значення функції переходів f і функції виходів g для всіх пар (s, x), де $s \in S$, $x \in I$.

Приклад 8.18. У табл. 8.2 задано функції переходів і виходів для автомата з множиною станів $S = \{s_0, s_1, s_2, s_3\}$ та вхідним і вихідним алфавітами $I = \{0, 1\}$, $O = \{0, 1\}$.

Таблиця 8.2

Стан	f		g	
	Вхід		Вхід	
	0	**1**	**0**	**1**
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

Ще один поширений і наочний спосіб подання автомата — за допомогою орієнтованого мультиграфа, який називають *діаграмою станів*. Вершини графа відповідають станам; якщо $f(s_i, x_j) = s_k$ та $g(s_i, x_j) = y_r$, то з вершини s_i у вершину s_k веде дуга з позначкою x_j, y_r. Тут $x_j \in I$, $y_r \in O$.

Приклад 8.19. Діаграму станів для автомата, заданого табл. 8.2, показано на рис. 8.6.

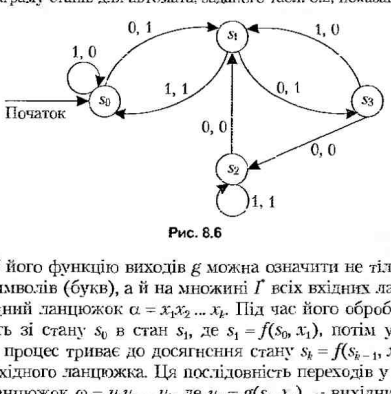


Рис. 8.6

Для автомата M його функцію виходів g можна означити не тільки на множині I всіх вхідних символів (букв), а й на множині I^* всіх вхідних ланцюжків (слів). Розглянемо вхідний ланцюжок $\alpha = x_1x_2 \ldots x_k$. Під час його обробки автомат спочатку переходить зі стану s_0 в стан s_1, де $s_1 = f(s_0, x_1)$, потім у стан s_2, де $s_2 = f(s_1, x_2)$, і цей процес триває до досягнення стану $s_k = f(s_{k-1}, x_k)$. Тут x_k — останній символ вхідного ланцюжка. Ця послідовність переходів у нові стани формує вихідний ланцюжок $\omega = y_1y_2 \ldots y_k$, де $y_1 = g(s_0, x_1)$ — вихідний символ, який відповідає переходу з s_0 в s_1, $y_2 = g(s_1, x_2)$ — вихідний символ, що відповідає переходу з s_1 в s_2, і так до отримання вихідного символу $y_k = g(s_{k-1}, x_k)$. Загалом $y_j = g(s_{j-1}, x_j)$ для $j = 1, 2, \ldots, k$. Отже, ми можемо розширити означення функції виходів на вхідні ланцюжки так, що $g(\alpha) = \omega$, де вихідний ланцюжок ω відповідає вхідному ланцюжку α.

Приклад 8.20. Знайдемо вихідний ланцюжок, який видає скінченний автомат, зображений на рис. 8.6, якщо вхідний ланцюжок — 101001.

Автомат видає на виході ланцюжок 011110. Послідовність його станів і вихідних символів наведено в табл. 8.3.

Таблиця 8.3

Вхід	1	0	1	0	0	1
Стан	s_0	s_0	s_1	s_0	s_1	s_2
Вихід	0	1	1	1	1	0
Новий стан	s_0	s_1	s_0	s_1	s_2	s_1

Розглянемо скінченний автомат M. Кожному вхідному ланцюжку α описаним вище способом поставимо у відповідність вихідний ланцюжок ω. Цю відповідність, що відображає вхідні ланцюжки у вихідні, називають *автоматним відображенням*, а також *автоматним* (або *обмежено детермінованим*) *оператором* [49], реалізованим автоматом M. Іноді коротко його називають *оператором* M; якщо результат застосування цього оператора до ланцюжка α — вихідний ланцюжок ω, то це позначають $M(\alpha) = \omega$. Кількість символів у ланцюжку α, як завжди, називають довжиною ланцюжка α та позначають $|\alpha|$ чи $l(\alpha)$.

Автоматне відображення має дві властивості.

1. Ланцюжки α та $\omega = M(\alpha)$ мають однакову довжину: $|\alpha|=|\omega|$ (збереження довжини).
2. Якщо $\alpha = \alpha_1\alpha_2$ й $M(\alpha_1\alpha_2) = \omega_1\omega_2$, де $|\alpha_1|=|\omega_1|$, то $M(\alpha_1) = \omega_1$, тобто образ відрізка довжиною l дорівнює відрізку образу з такою самою довжиною.

Властивість 2 означає, що автоматні оператори — це оператори *без випередження*, тобто такі, котрі, обробляючи ланцюжок зліва направо, „не підглядають уперед": i-та буква вихідного ланцюжка залежить тільки від перших i букв вхідного ланцюжка. Приклад оператора з випередженням — той, який ланцюжку $\alpha = x_1x_2 \dots x_k$ ставить у відповідність ланцюжок $x_k \dots x_2x_1$; перша буква вихідного ланцюжка тут дорівнює останній букві вхідного ланцюжка. Зазначимо, що ці дві властивості — це не достатні умови автоматності відображення: існують відображення, які задовольняють умови 1 і 2, але не реалізовані в скінченному автоматі.

Розглянемо деякі корисні приклади скінченних автоматів, які свідчать, що стани скінченного автомата дають змогу застосовувати їх як скінченну пам'ять. Стани можна використовувати для запам'ятовування ситуацій або символів, які читає автомат. Проте через скінченну множину станів скінченні автомати не можна використовувати в деяких важливих застосуваннях.

Приклад 8.21. Важливий елемент багатьох пристроїв — автомат одиничної затримки. Він видає на виході вхідний ланцюжок, затриманий на одиницю часу. Отже, якщо на вхід подано двійковий ланцюжок $x_1x_2 \dots x_k$, то на виході буде ланцюжок $0x_1x_2 \dots x_{k-1}$.

У такого автомата має бути два вхідні символи (наприклад, 0 і 1), початковий стан s_0; він має „пам'ятати", який із двох символів 0 або 1 був на вході в попередній момент. Отже, потрібно ще два стани: нехай автомат перебуває в стані s_1, якщо попереднім вхідним символом була 1, і в стані s_2 — якщо 0. На виході в початковому стані завжди 0 незалежно від входу. Кожний перехід зі стану s_1

дає на виході 1, а зі стану s_2 — 0. Діаграму станів цього автомата зображено на рис. 8.7.

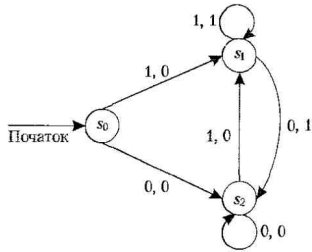


Рис. 8.7

Приклад 8.22. Побудуємо скінченний автомат для додавання двох цілих додатних чисел у двійковій системі.

Візьмемо числа $(x_n \ldots x_1x_0)_2$ та $(y_n \ldots y_1y_0)_2$. Додавши розряди x_0 та y_0, отримаємо розряд суми z_0 та біт перенесення c_0, який дорівнює 0 чи 1. Потім додамо розряди x_1 та y_1 і біт перенесення c_0. Одержимо розряд суми z_1 і біт перенесення c_1. Процедуру продовжуємо до стадії підсумовування розрядів x_n, y_n і попереднього біта перенесення c_{n-1}; отримаємо розряд суми z_n і біт перенесення c_n, який дорівнює розряду суми z_{n+1}.

Таблиця 8.4

Стан	f				g			
	Вхід				Вхід			
	00	**01**	**10**	**11**	**00**	**01**	**10**	**11**
s_0	s_0	s_0	s_0	s_1	0	1	1	0
s_1	s_0	s_1	s_1	s_1	1	0	0	1

Вхідний алфавіт автомата складається з чотирьох символів: $I = \{00, 01, 10, 11\}$. Вони потрібні для подання можливих значень x_i й y_i i-го розряду обох доданків. Вихідний алфавіт $O = \{0, 1\}$, множина станів $S = \{s_0, s_1\}$. Стан s_0 відповідає ситуації, коли немає одиниці перенесення з попереднього розряду; цей стан початковий. Стан s_1 відповідає наявності одиниці перенесення з попереднього розряду. Розв'язок наведено у вигляді таблиці станів (табл. 8.4) та діаграми станів (рис. 8.8).

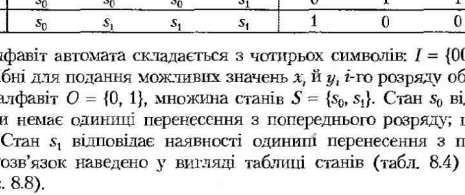


Рис. 8.8

Приклад 8.23. Побудуємо скінченний автомат, який видає на виході 1 тоді й лише тоді, коли останні три символи на вході — 1.

У цього автомата має бути три стани: початковий стан s_0, його використовують для запам'ятовування ситуації, коли попередній вхідний символ — 0; стан s_1 відповідає ситуації, коли попередній символ на вході — 1, а символ перед ним — 0; стан s_2 запам'ятовує ситуацію двох поспіль вхідних 1. Отже, якщо автомат перейшов зі стану s_1 у стан s_2, то на вхід подано дві 1 поспіль. Вхід 1 у стані s_2 означає, що це була третя поспіль 1, і на виході з'являється 1. У всіх інших ситуаціях на виході з'являється 0. Діаграму станів цього автомата зображено на рис. 8.9.

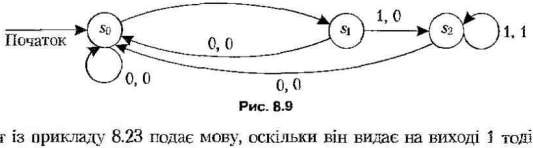


Рис. 8.9

Автомат із прикладу 8.23 подає мову, оскільки він видає на виході 1 тоді й тільки тоді, коли вхідний ланцюжок (слово) має спеціальні властивості (у цьому прикладі мова складається з ланцюжків нулів і одиниць, які закінчуються трьома 1 поспіль). Розпізнавання мов — одне із найважливіших застосувань скінченних автоматів.

Розглянуті автомати називають автоматами Мілі (G. Mealy). Уперше їх уведено 1955 р. Є також інший тип автоматів із виходом — так звані автомати Мура (E. Moore), запроваджені 1956 р. У цих автоматах вихід залежить лише від стану, а не від вхідного сигналу.

У прикладі 8.23 описано, як автомат Мілі можна використати для розпізнавання мови. Проте для цього зазвичай застосовують інший тип автоматів — скінченні автомати без виходу. Вони мають множину заключних (або приймаючих) станів і допускають ланцюжок тоді й лише тоді, коли він переводить автомат без виходу з початкового стану в заключний.

8.7. Скінченні автомати без виходу

Одне з найважливіших застосувань скінченних автоматів — розпізнавання (подання) мов, яке має фундаментальне значення в дослідженні й побудові компіляторів для мов програмування. У прикладі 8.23 описано, як скінченний автомат із виходом може розпізнати мову: він видає на виході 1, якщо вхідний ланцюжок належить мові, і 0 — у протилежному випадку. Проте є інший тип скінченних автоматів, спеціально призначений для розпізнавання мов. Замість виходу ці автомати мають множину заключних станів. Автомат допускає ланцюжок, якщо той переводить автомат із початкового стану в один із заключних.

Скінченним автоматом без виходу називають систему $M = (S, I, f, s_0, F)$, у якій S — скінченна *множина станів*, I — скінченний *вхідний алфавіт*, f: $S \times I \to S$ — *функція переходів*, означена на декартовому добутку $S \times I$, $s_0 \in S$ — *початковий стан*, $F \subset S$ — множина *заключних* (або *приймаючих*) *станів*.

Елементи вхідного алфавіту, як і раніше, називають *вхідними символами* чи *входами*.

Скінченні автомати без виходу можна задавати таблицями станів або діаграмами станів. Заключні стани на діаграмі зображають подвійнними кружечками. Позаяк у автоматах без виходу є тільки входи (символи вхідного алфавіту I), то на дугах діаграми записують тільки їх.

Далі ми будемо розглядати лише скінченні автомати без виходу, тому називатимемо їх *скінченними автоматами* чи просто *автоматами*.

Приклад 8.24. Побудуємо діаграму станів для скінченного автомата $M = (S, I, f, s_0, F)$, де $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$. Функцію переходів цього автомата задано табл. 8.5, діаграму станів зображено на рис. 8.10. Оскільки обидва входи 0 і 1 переводять автомат зі стану s_2 в стан s_0, то замість двох дуг від s_2 до s_0 використано лише одну дугу, на якій написано два входи: 0 і 1.

Таблиця 8.5.

Стан	f	
	Вхід	
	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1

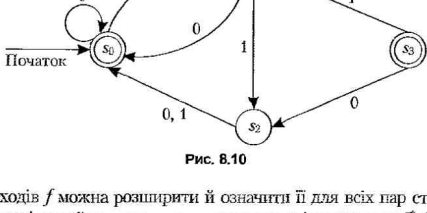


Рис. 8.10

Функцію переходів f можна розширити й означити її для всіх пар станів і ланцюжків. У такому разі нехай $\alpha = x_1x_2 \ldots x_k$ — ланцюжок із множини I^*. Тоді $f(s_1, \alpha)$ — стан, обчислений із використанням послідовних символів ланцюжка α зліва направо як вхідних символів, починаючи зі стану s_1. Процес відбувається так: $s_2 = f(s_1, x_1)$; $s_3 = f(s_2, x_2)$, ... Уважаємо, що $f(s_1, \alpha) = f(s_k, x_k)$.

Говорять, що скінченний автомат $M = (S, I, O, f, s_0, F)$ *допускає* (*приймає*) ланцюжок α, якщо він переводить початковий стан s_0 в заключний стан; це означає, що стан $f(s_0, \alpha)$ — елемент множини F. Мова $L(M)$, яку *розпізнає* автомат M, — це множина всіх ланцюжків, які допускає автомат M. Два автомати називають *еквівалентними*, якщо вони розпізнають одну й ту саму мову.

Приклад 8.25. Знайдемо мову, яку розпізнає скінченний автомат M_1 із діаграмою станів, зображеною на рис. 8.11.

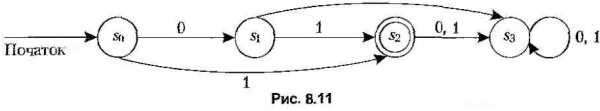


Рис. 8.11

Автомат M_1 має тільки один заключний стан s_2. Тільки два ланцюжки переводять s_0 в s_2: 1 і 01. Отже, $L(M_1) = \{1, 01\}$.

Приклад 8.26. Знайдемо мову, яку розпізнає скінченний автомат M_2 з діаграмою станів, зображеною на рис. 8.12.

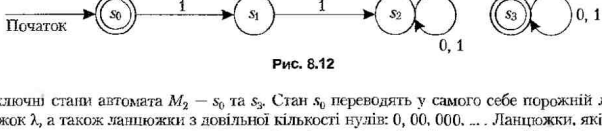


Рис. 8.12

Заключні стани автомата M_2 — s_0 та s_2. Стан s_0 переводять у самого себе порожній ланцюжок λ, а також ланцюжки з довільної кількості нулів: 0, 00, 000, ... Ланцюжки, які переводять стан s_0 в s_2, складаються з якоїсь кількості нулів, після яких є 10 і довільний ланцюжок β з нулів і одиниць. Отже,

$$L(M_2) = \{0^n, 0^n10\beta \mid n = 0, 1, 2, ...; \beta - \text{довільний ланцюжок}\}.$$

Розглянуті скінченні автомати без виходу називають *детермінованими*, бо для кожної пари „стан — вхід" існує єдиний наступний стан, заданий функцією переходів. Є й інший тип автоматів без виходу — недетерміновані. У них може бути декілька можливих наступних станів для кожної пари „стан — вхідний символ".

Недетермінованим скінченним автоматом без виходу називають систему $M = (S, I, f, s_0, F)$, у якій S — скінченна *множина станів*, I — скінченний *вхідний алфавіт*, f — *функція переходів*, яка кожній парі „стан — вхід" ставить у відповідність множину станів, $s_0 \in S$ — *початковий стан*, $F \subset S$ — множина *заключних* (або *приймаючих*) *станів*.

Єдина відмінність між недетермінованим і детермінованим автоматами — тип значень функції переходів f. Для недетермінованого автомата це множина станів (вона може бути й порожньою), а для детермінованого — один стан. Недетермінований скінченний автомат задають таблицею чи діаграмою станів. У таблиці для кожної пари „стан — вхід" записують множину всіх можливих наступних станів (якщо вона порожня, то ставлять прочерк). У діаграмі переходів проводять дуги з кожного стану до всіх можливих наступних станів, на цих дугах записують входи, які спричинюють переходи одного стану в інший.

Приклад 8.27. На рис. 8.13 і в табл. 8.6 подано відповідно діаграму та таблицю станів якогось недетермінованого автомата.

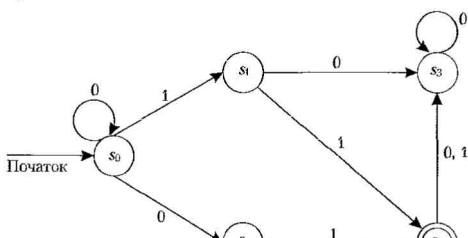


Рис. 8.13

Таблиця 8.6

Стан	f	
	Вхід	
	0	1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2	–	s_4
s_3	s_3	–
s_4	s_3	s_3

Тепер визначимо, як недетермінований скінченний автомат допускає (приймає) ланцюжок $\alpha = x_1x_2 \ldots x_k$. Перший вхідний символ x_1 переводить стан s_0 в множину S_1, яка може містити більше одного стану. Наступний вхідний символ x_2 переводить *кожний* зі станів множини S_1 у якусь множину станів, і нехай S_2 — об'єднання цих множин. Цей процес продовжують, вибираючи на кожній стадії всі стани, отримані з використанням поточного вхідного символу й усіх станів, одержаних на попередній стадії. Недетермінований скінченний автомат *допускає* (*приймає*) ланцюжок α, якщо в множині станів, отриманій з початкового стану s_0 під дією ланцюжка α, є заключний. Мова, яку *розпізнає* недетермінований скінчений автомат — це множина всіх ланцюжків, які він допускає.

Приклад 8.28. Знайдемо мову, яку розпізнає недетермінований скінченний автомат, поданий на рис. 8.13.

Оскільки s_0 — заключний стан і вхід 0 переводить його в себе, то автомат допускає ланцюжки λ, 0, 00, 000, 0000, ... Стан s_4 також заключний, і нехай s_4 є в множині станів, що досягаються зі стану s_0 з ланцюжком α на вході. Тоді ланцюжок α допускається. Такими ланцюжками є 0^n01 і 0^n11, де $n = 0, 1, 2, \ldots$. Інших заключних станів немає, тому цей недетермінований автомат розпізнає таку мову: $\{0^n, 0^n01, 0^n11 | n = 0, 1, 2, \ldots\}$.

ТЕОРЕМА 8.1. Якщо мову L розпізнає недетермінований скінченний автомат M_0, то її розпізнає також детермінований скінченний автомат M_1.

Доведення. Опишемо, як можна побудувати автомат M_1 за автоматом M_0. Кожний стан автомата M_1 – підмножина станів автомата M_0. Символ початкового стану автомата M_1 – $\{s_0\}$, тобто це множина, єдиний елемент якої – початковий стан s_0 автомата M_0. Вхідний алфавіт автомата M_1 той самий, що й у автоматі M_0. Нехай *задано* стан $\{s_{i_1}, s_{i_2}, ..., s_{i_k}\}$ автомата M_1. Вхідний символ x переводить його в об'єднання множин наступних станів для елементів цієї множини, тобто $f(s_{i_1}, x) \cup \cup f(s_{i_2}, x) \cup ... \cup f(s_{i_k}, x)$. Стани автомата M_1 – це всі такі підмножини множини S (множини станів автомата M_0), одержані зазначеним способом, починаючи від $\{s_0\}$. (Отже, у детермінованому автоматі може бути до 2^n станів, де n – кількість станів у недетермінованому автоматі, проте зазвичай їх буває значно менше.) Заключні стани автомата M_1 – підмножини, які містять заключні стани автомата M_0.

Нехай автомат M_0 допускає вхідний ланцюжок α. Тоді один зі станів, досяжний зі стану s_0 на вході α, – заключний. Це означає, що в автоматі M_1 ланцюжок α веде зі стану $\{s_0\}$ до такої підмножини станів автомата M_0, яка містить заключний стан. За побудовою ця підмножина – заключний стан автомата M_1. Отже, автомат M_1 також допускає ланцюжок α. Якщо ж автомат M_0 не допускає вхідний ланцюжок β, то цей ланцюжок не веде до жодного із заключних станів автомата M_0. Звідси випливає, що ланцюжок β не веде зі стану $\{s_0\}$ до жодного заключного стану автомата M_1.

Із теореми 8.1 випливає, що недетерміновані скінченні автомати розпізнають ті самі мови (множини ланцюжків), що й детерміновані. Проте є причини розглядати недетерміновані автомати, бо вони часто компактніші та їх легше побудувати, ніж детерміновані. Крім того, хоча недетермінований автомат завжди можна перетворити на детермінований, останній може мати експоненціально більше станів. На щастя, таке трапляється дуже рідко.

Приклад 8.29. Побудуємо детермінований скінченний автомат, який розпізнає ту саму мову, що й недетермінований автомат, зображений на рис. 8.13.

Такий детермінований скінченний автомат зображено на рис. 8.14. Його стани – підмножини множини станів недетермінованого автомата, зображеного на рис. 8.13.

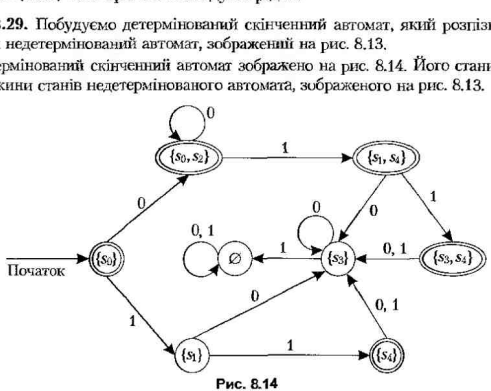


Рис. 8.14

Їх одержано за схемою доведення теореми 8.1. Зокрема, вхід 0 переводить стан $\{s_0\}$ в стан $\{s_0, s_2\}$ в детермінованому автоматі, бо стан s_0 на вході 0 переходить у самого себе й у стан s_2 в недетермінованому автоматі. Вхід 1 переводить множину станів $\{s_0, s_2\}$ в множину $\{s_1, s_4\}$, оскільки стан s_0 на вході 1 у недетермінованому автоматі переходить в s_1, а стан s_2 — в s_4. Нарешті, вхід 0 переводить множину станів $\{s_1, s_4\}$ в множину $\{s_3\}$, тому що в недетермінованому автоматі вхід 0 переводить обидва стани s_1 і s_4 в стан s_3. Усі одержані таким способом підмножини — стани детермінованого скінченного автомата, зображеного на рис. 8.14. Один зі станів цього автомата — порожня множина: вона містить усі стани, у які вхід 1 переводить $\{s_3\}$. Початковий стан — $\{s_0\}$, а заключні — усі стани, які містять стани s_0 чи s_4.

8.8. Подання мов

Спочатку розглянемо деякі фундаментальні питання, що стосуються множин ланцюжків. Нехай A та B — підмножини множини V^*, де V — алфавіт. *Конкатенацією множин* A та B (позначають AB) називають множину всіх ланцюжків $\alpha\beta$, де α — ланцюжок із множини A, а β — ланцюжком із множини B.

Приклад 8.30. Нехай $A = \{0, 11\}$ і $B = \{1, 10, 110\}$. Знайдемо множини AB та BA. Множина AB містить усі конкатенації ланцюжка з A та ланцюжка з B, тому

$$AB = \{01, 010, 0110, 111, 1110, 11110\}.$$

Множина BA містить усі конкатенації ланцюжка з B та ланцюжка з A:

$$BA = \{10, 111, 100, 1011, 1100, 11011\}.$$

Загалом $AB \neq BA$.

Використавши означення конкатенації двох множин ланцюжків, можна означити степінь множини ланцюжків A^n для $n = 0, 1, 2, \ldots$. Це роблять за допомогою рекурсії:

$$A^0 = \{\lambda\};$$
$$A^{n+1} = A^n A \text{ для } n = 0, 1, 2, \ldots.$$

Тут, як і раніше, λ — порожній ланцюжок.

Приклад 8.31. Нехай $A = \{1, 00\}$. Знайдемо A^n для $n = 1, 2, 3$.

$A^0 = \{\lambda\}$, $A^1 = A^0A = \{\lambda\}A = \{1, 00\}$. Щоб знайти A^2, візьмемо конкатенацію всіх пар елементів з A: $A^2 = \{11, 100, 001, 0000\}$. Для відшукання A^3 потрібно взяти конкатенації елементів з A^2 й A:

$$A^3 = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}.$$

Нехай A — підмножина множини V^*. *Замиканням Кліні множини* A (позначають A^*) називають множину всіх ланцюжків, які можна утворити конкатенацією довільної кількості ланцюжків з A. Отже, $A^* = \bigcup_{k=0}^{\infty} A^k$.

Приклад 8.32. Знайдемо замикання Кліні для множин $A = \{0\}$, $B = \{0, 1\}$ та $C = \{1, 1\}$. Замикання Кліні множини A — конкатенація ланцюжка 0 із собою довільну скінченну кількість разів: $A^* = \{0^n \mid n = 0, 1, 2, \ldots\}$. Замикання Кліні множини B — конкатенація довільної кількості 0 і 1. Це множина всіх ланцюжків над алфавітом $V = \{0, 1\}$, тобто

$B^* = V^*$. Нарешті, замикання Кліні множини C – конкатенація ланцюжка 11 із собою довільну скінченну кількість разів. Отже, C^* — множина ланцюжків із парною кількістю одиниць: $C^* = \{1^{2n} \mid n = 0, 1, 2, ...\}$.

Скінченні автомати можна використовувати для подання (розпізнавання) мов (множин ланцюжків). Які саме множини розпізнаються ними? Цю проблему вперше розв'язав американський математик Кліні (S. Kleene) 1956 р. Він довів, що існує скінченний автомат, який розпізнає множину ланцюжків тоді й лише тоді, коли її можна побувати з порожньої множини (що не містить жодного ланцюжка), множини, що містить тільки порожній ланцюжок, і множин, що містять один односимвольний ланцюжок, за допомогою операцій конкатенації, об'єднання та замикання Кліні в довільному порядку. Множини ланцюжків, які можна побудувати таким способом, називають регулярними.

Регулярний вираз над множиною I рекурсивно означають так:

- символ $\varnothing$ – регулярний вираз;
- символ λ – регулярний вираз;
- символ x – регулярний вираз, якщо $x \in I$;
- вирази **(AB)**, **(A ∪ B)** й **A*** регулярні, якщо вирази **A** та **B** регулярні.

Кожний регулярний вираз задає множину ланцюжків. визначену за такими правилами:

- $\varnothing$ – порожню множину, тобто таку, що не містить жодного ланцюжка;
- λ – множину $\{\lambda\}$, що містить тільки порожній ланцюжок;
- x – множину $\{x\}$, яка має один ланцюжок, що складається з одного символу x;
- **(AB)** – конкатенацію множин, поданих виразами **A** та **B**;
- **(A ∪ B)** – об'єднання множин, поданих виразами **A** та **B**;
- **A*** – замикання Кліні множини, поданої виразом **A**.

Множину, задану регулярним виразом, називають *регулярною*. Відтепер регулярні вирази будемо використовувати для опису регулярних множин. Це означає таке: посилаючись на регулярну множину **A**, ми будемо мати на увазі регулярну множину, задану регулярним виразом **A**.

Нижче наведено приклад того, як регулярні вирази використовують для того, щоб задавати регулярні множини.

Приклад 8.33. Покажемо, з яких ланцюжків утворено регулярні множини, задані регулярними виразами **10***, **(10)***, **0 ∪ 01**, **0(0 ∪ 1)*** та **(0*1)*** (табл. 8.7).

Таблиця 8.7

Вираз	Ланцюжки, з яких складається відповідна регулярна множини
10*	1, а потім довільна кількість 0 (або без нулів)
(10)*	Довільна кількість повторень 10 (включно з порожнім ланцюжком)
0 ∪ 01	Ланцюжок 0 і ланцюжок 01
0(0 ∪ 1)*	Довільний ланцюжок, який починається з 0
(0*1)*	Довільний ланцюжок, який закінчується 1

ТЕОРЕМА 8.2 (Кліні). Для того щоб множина була регулярною, необхідно й достатньо, щоб її розпізнав скінченний автомат [52].

Доведення. Доведемо лише необхідність. Доведення достатності складне й виходить за межі цієї книги. Позаяк регулярну множину означено в термінах регулярних виразів, заданих рекурсивно, то доведемо, що скінченний автомат розпізнає будь-яку регулярну множину, якщо зробимо такі обґрунтування.

1. Доведемо, що скінченний автомат розпізнає $\varnothing$.
2. Скінченний автомат розпізнає $\{\lambda\}$.
3. Скінченний автомат розпізнає $\{a\}$, якщо a — символ з I.
4. Скінченний автомат розпізнає AB, якщо він розпізнає A та B.
5. Скінченний автомат розпізнає $A \cup B$, якщо він розпізнає A та B.
6. Скінченний автомат розпізнає A^*, якщо він розпізнає A.

Тепер розглянемо кожну з цих задач.

1. Множину $\varnothing$ розпізнає недетермінований автомат без заключних станів, зображений на рис. 8.15.

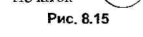


Рис. 8.15

2. Недетермінований автомат, який розпізнає $\{\lambda\}$, має початковим і заключним станами s_0 (рис. 8.16).

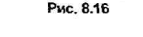


Рис. 8.16

3. Доведемо, що недетермінований скінченний автомат розпізнає множину $\{a\}$. Розглянемо автомат із початковим станом s_0 та заключним станом s_1. Потрібно, щоб був лише один перехід з s_0 в s_1, коли на вхід подано символ a; ніяких інших переходів не має бути. Діаграму такого автомата зображено на рис. 8.17. Єдиний ланцюжок, який допускає цей автомат, — односимвольний ланцюжок a.

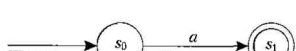


Рис. 8.17

4. Нехай скінченний автомат розпізнає множини ланцюжків A та B. Нехай автомат $M_A = (S_A, I, f_A, s_A, F_A)$ розпізнає множину ланцюжків A, а автомат $M_B = (S_B, I, f_B, s_B, F_B)$ — множину ланцюжків B. Побудуємо скінченний автомат $M_{AB} = (S_{AB}, I, f_{AB}, s_{AB}, F_{AB})$, який розпізнає множину ланцюжків AB, тобто конкатенацію множин A та B. Автомат M_{AB} являє собою послідовне з'єднання автоматів M_A та M_B. Отже, ланцюжок із множини A використовує стани

автомата M_{AB} від s_A (початкового стану автомата M_A) до s_B (початкового стану автомата M_B). Ланцюжок із множини B використовує стани автомата M_{AB} від s_B до заключного стану автомата M_{AB}. Побудуємо таку конструкцію. Нехай $S_{AB} = S_A \cup S_B$. Початковий стан s_{AB} збігається зі станом s_A. Множина заключних станів F_{AB} збігається з множиною заключних станів автомата M_B, і додатково до F_{AB} додамо ще й стан s_{AB}, якщо й лише якщо $\lambda \in A \cap B$.

Переходи в автоматі M_{AB} містять усі переходи автоматів M_A й M_B, а також деякі нові переходи, які сформуємо так. Для кожного переходу в автоматі M_A, який веде зі стану s_k до заключного стану в автоматі M_A, формуємо перехід в автоматі M_{AB} зі стану s_k до стану s_B з тим самим вхідним символом. Окрім того, у разі $\lambda \in A$ для кожного переходу зі стану s_B до стану s_m в автоматі M_B формуємо перехід в автоматі M_{AB} зі стану $s_{AB} = s_A$ до стану s_m (із тим самим вхідним символом). Усі ці конструкції проілюстровано на рис. 8.18.

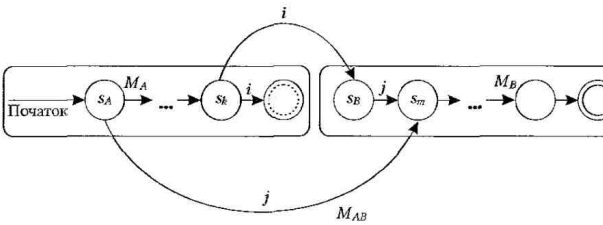


Рис. 8.18

5. Побудуємо скінченний автомат $M_{A\cup B} = (S_{A\cup B}, I, f_{A\cup B}, s_{A\cup B}, F_{A\cup B})$, який розпізнає множину ланцюжків $A \cup B$. Він являє собою паралельне з'єднання автоматів M_A та M_B. Уводимо новий початковий стан $s_{A\cup B}$. Нехай в автоматі M_A є перехід із початкового стану s_A до стану s_k, а в автоматі M_B — перехід із початкового стану s_B до стану s_m. Тоді в автоматі $M_{A\cup B}$ є перехід від стану $s_{A\cup B}$ до стану s_k та до стану s_m.

   Нехай $S_{A\cup B} = S_A \cup S_B \cup \{s_{A\cup B}\}$, де $s_{A\cup B}$ — новий стан, який являє собою початковий стан в автоматі $M_{A\cup B}$.

   Множина заключних станів $F_{A\cup B}$ дорівнює $F_A \cup F_B \cup \{s_{A\cup B}\}$, якщо $\lambda \in A \cup B$, а ні — то $F_A \cup F_B$. З описаної конструкції зрозуміло, що ланцюжок із множини A переводить автомат $M_{A\cup B}$ з початкового стану $s_{A\cup B}$ в заключний стан і ланцюжок із множини B переводить автомат $M_{A\cup B}$ з початкового стану $s_{A\cup B}$ в заключний стан. На рис. 8.19 показано конструкцію автомата $M_{A\cup B}$.

6. Нарешті, побудуємо автомат $M_{A^*} = \{S_{A^*}, I, f_{A^*}, s_{A^*}, F_{A^*}\}$, який розпізнає A^* — замикання Кліні множини ланцюжків A. Нехай множина S_{A^*} містить усі стани з множини S_A та додатковий стан s_{A^*}, який являє собою початковий стан нового автомата. Множина заключних станів F_{A^*} містить усі стани з F_A та початковий стан s_{A^*}, бо автомат M_{A^*} має допускати порожній ланцюжок λ.

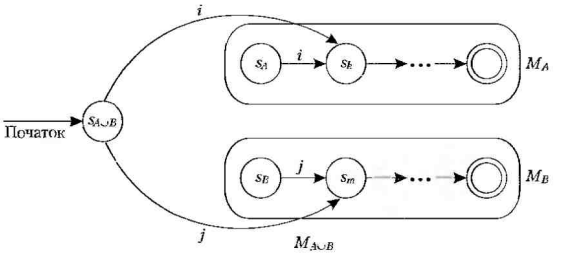


Рис. 8.19

Щоб автомат M_{A^*} допускав конкатенацію довільної кількості ланцюжків із множини A, уведемо в його структуру всі переходи автомата M_A й деякі нові переходи. Якщо є перехід зі стану s_A – початкового стану автомата M_A, то введемо в автоматі M_{A^*} аналогічний перехід зі стану s_{A^*} з тим самим вхідним символом. Окрім того, уведемо переходи з кожного заключного стану автомата M_A в стани, у які є переходи зі стану s_A (знову з тим самим вхідним символом).

Тепер будь-який ланцюжок, який являє собою конкатенацію ланцюжків із множини A, переведе стан s_{A^*} в один із заключних станів, коли буде прочитано перший ланцюжок, а потім переведе до одного із заключних станів, коли буде прочитано другий ланцюжок, і цей процес продовжиться. Використану тут конструкцію показано на рис. 8.20.

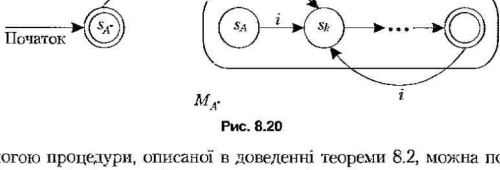


Рис. 8.20

За допомогою процедури, описаної в доведенні теореми 8.2, можна побудувати недетермінований скінченний автомат для довільної регулярної множини. Проілюструємо, як це можна зробити, на такому прикладі.

Приклад 8.34. Побудуємо недетермінований скінченний автомат, який розпізнає регулярну множину $\mathbf{1}^* \cup \mathbf{01}$. Послідовність потрібних дій показано на рис. 8.21. Почнемо з побудови автомата, який розпізнає $\mathbf{1}^*$. Для цього використаємо автомат, який розпізнає **1**, а потім – конструкцію для автомата M_{A^*}, описану в доведенні теореми. Далі побудуємо автомат, який розпізнає множину **01**, використавши автомати, які розпізнають множини **0** і **1**, а також конструкцію для автомата M_{AB} з доведення. Нарешті, за допомогою конструкції $M_{A\cup B}$ побудуємо автомат для множини $\mathbf{1}^* \cup \mathbf{01}$. У послідовно одержуваних автоматах позначатимемо стани щоразу новими символами.

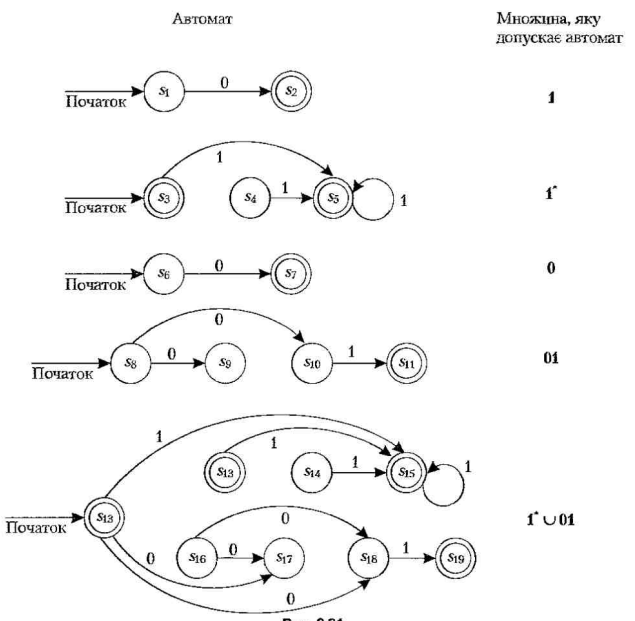


Рис. 8.21

Побудову виконано точно відповідно до схеми доведення теореми. Значно простіший автомат, який розпізнає шю саму мову, зображено на рис. 8.22.

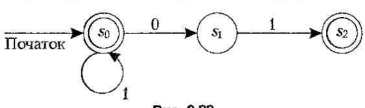


Рис. 8.22

Є тісний зв'язок між регулярними множинами та регулярними граматиками. Його розкрито в теоремі 8.3.

ТЕОРЕМА 8.3. Регулярна граматика породжує регулярну множину й лише її.

Доведення. Спочатку доведемо, що множина, породжена регулярною граматикою, регулярна. Нехай $G = (V, T, S, P)$ — регулярна граматика, яка породжує множину $L(G)$. Доведемо, що $L(G)$ — регулярна множина. Для цього побудуємо

недетермінований скінченний автомат $M = (S, I, f, s_0, F)$, який розпізнає множину $L(G)$. Множина станів S автомата M містить стан s_A для кожного нетермінального символу A граматики G та заключний стан s_F. Початковий стан s_0 відповідає початковому символу S граматики G.

Переходи в автоматі M утворимо за допомогою продукцій граматики G таким способом. Якщо в граматиці є продукція $A \to a$, то в автоматі M має бути перехід зі стану s_A до заключного стану s_F для вхідного символу a. За наявності продукції $A \to aB$ утворюють перехід зі стану s_A до стану s_B для вхідного символу a. Множина F заключних станів автомата M містить стан s_F і, якщо $S \to \lambda$ — продукція граматики G, то ще й стан s_0. Тепер неважко переконатись, що мова $L(M)$, яку розпізнає побудований автомат M, збігається з мовою $L(G)$, породженою граматикою G. Отже, $L(M) = L(G)$. Це можна зробити, перевіривши ланцюжки, які переводять автомат у заключний стан. Отже, ми побудували недетермінований скінченний автомат, який розпізнає множину, породжену регулярною граматикою G. Тепер регулярність цієї множини випливає з теореми 8.2.

Перш ніж завершити доведення, наведемо приклад описаної вище методики побудови скінченного автомата, який розпізнає ту саму множину, що й породжує регулярна граматика.

Приклад 8.35. Побудуємо недетермінований скінченний автомат для розпізнавання мови, породженої регулярною граматикою $G = (V, T, S, P)$, де $V = \{0, 1, A, S\}$, $T = \{0, 1\}$, а множина P складається з таких продукцій: $S \to 1A$; $S \to 0$; $S \to \lambda$; $A \to 0A$; $A \to 1A$ та $A \to 1$.

Діаграму переходів для автомата, який розпізнає мову $L(G)$, зображено на рис. 8.23. Тут s_0 — стан, який відповідає початковому символу S граматики G, s_1 — стан, який відповідає нетерміналу A, s_2 — заключний стан.

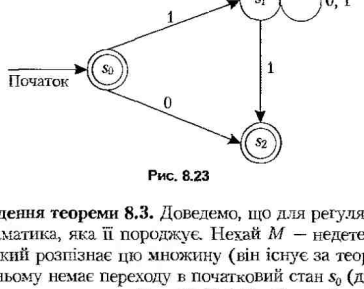


Рис. 8.23

Завершення доведення теореми 8.3. Доведемо, що для регулярної множини існує регулярна граматика, яка її породжує. Нехай M — недетермінований скінченний автомат, який розпізнає цю множину (він існує за теоремою 8.2). Можна вважати, що в ньому немає переходу в початковий стан s_0 (див. приклад 8.35). Побудуємо регулярну граматику $G = (V, T, S, P)$. Кожний символ алфавіту V в граматиці G вводимо відповідно до символу стану чи вхідного символу автомата M. Множина T термінальних символів граматики G якраз і відповідає символам вхідного алфавіту I автомата M, а початковий символ S граматики G — символу початкового стану s_0 автомата M.

Множину P продукцій граматики G сформуємо на основі переходів у автоматі M. Якщо стан s у разі вхідного символу a переходить у кінцевий стан, то до множини P додають продукцію $A_s \to a$, де A_s – нетермінальний символ граматики G, що відповідає стану s. Якщо стан s переходить у стан t в разі вхідного символу a, то до множини P додають продукцію $A_s \to aA_t$. Продукцію $S \to \lambda$ додають до P, якщо й тільки якщо $\lambda \in L(M)$. Отже, продукції в граматиці G відповідають переходам в автоматі M. Тепер легко переконатись, що $L(G) = L(M)$.

Приклад 8.36. На рис. 8.24 зображено діаграму переходів скінченного автомата. Знайдемо регулярну граматику, яка породжує розпізнавану ним регулярну множину.

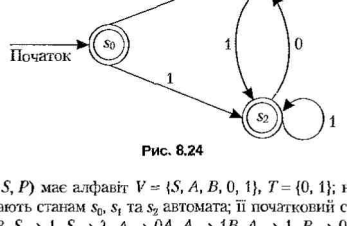


Рис. 8.24

Граматика $G = (V, T, S, P)$ має алфавіт $V = \{S, A, B, 0, 1\}$, $T = \{0, 1\}$; нетермінальні символи S, A та B відповідають станам s_0, s_1 та s_2 автомата; її початковий символ – S, а продукції – $S \to 0A$, $S \to 1B$, $S \to 1$, $S \to \lambda$, $A \to 0A$. $A \to 1B$, $A \to 1$, $B \to 0A$, $B \to 1B$ та $B \to 1$. Усі ці продукції отримано так, як це описано в другій частині доведення теореми 8.3.

Із теорем 8.2 та 8.3 безпосередньо випливає таке твердження.

ТЕОРЕМА 8.4. Для того щоб мова була регулярною, необхідно й достатньо, щоб її розпізнав скінченний автомат.

Типова мова нескінченна, і тому немає сенсу наводити її ланцюжки та перевіряти нескінченну множину ланцюжків. Набагато розумніше використовувати один зі способів скінченного подання мови, а саме – детерміновані та недетерміновані скінченні автомати, регулярні вирази. Очевидно, що подані одним із цих способів мови регулярні.

Скінченний автомат для мов типу 3 (регулярних) – адекватна модель, а для складніших мов адекватні інші автоматні моделі, які відрізняються від скінченних автоматів нескінченністю пам'яті. Проте на цю нескінченність накладають різні обмеження залежно від типу моделі та пов'язаної з нею мови. Пам'ять може бути *магазинною*, тобто доступною лише з одного кінця (стек); такий автомат – адекватне подання мов типу 2. Вона може бути також *лінійно обмеженою*, тобто такою, що лінійно залежить від довжини розпізнаваного слова. Зокрема, такі автомати розпізнають мови типу 1. Усі названі обмеження на нескінченність пам'яті обмежують можливості цих моделей порівняно з *машинами Тьюрінга* (див. розділ 9). Машини Тьюрінга можна вважати автоматною моделлю мов типу 0, однак їх використовують переважно для уточнення поняття алгоритму.

Розглянемо інструмент для доведення нерегулярності деяких мов — *лему про накачування* (*розростання*).

ЛЕМА 8.1 (про накачування для регулярних мов). Нехай L — регулярна мова. Існує залежна від L константа n така, що кожний ланцюжок $\alpha \in L$, який задовольняє нерівність $|\alpha| \geqslant n$, можна розбити на три ланцюжки $\alpha = \beta\gamma\omega$ так, що виконуються такі умови.

1. $\gamma \neq \lambda$.
2. $|\beta\gamma| \leqslant n$.
3. Для довільного $k \geqslant 0$ ланцюжок $\beta\gamma^k\omega \in L$.

Це означає, що завжди можна знайти такий ланцюжок γ недалеко від початку ланцюжка α, котрий можна „накачати". Отже, якщо ланцюжок γ повторити довільну кількість разів або вилучити ($k = 0$), то одержаний ланцюжок належить мові L.

Доведення. Нехай L — регулярна мова. Тоді існує детермінований скінченний автомат M, для якого $L = L(M)$, тобто такий, що розпізнає цю мову. Нехай кількість станів автомата M дорівнює n (константа, яка фігурує у формулюванні теореми). Припустимо, що послідовні стани, у які автомат M переходить під дією вхідного ланцюжка α, — s_0, s_{i_1}, s_{i_2}, ..., s_{i_m}, де $m = |\alpha|$ — довжина ланцюжка α. За умовою теореми $m \geqslant n$, тому в списку s_0, s_{i_1}, s_{i_2}, ..., s_{i_m}, який складається принаймні з $n + 1$ стану, обов'язково є повторення (це випливає з принципу коробок Дірихле). Нехай s_r — перший повторюваний стан. Позначимо як γ ту частину ланцюжка α, що переводить автомат зі стану s_r, коли він зустрічається вперше, до стану s_r, коли він зустрічається вдруге. Позначимо як β частину ланцюжка α перед γ, а як ω — його частину після γ. Очевидно, що $|\gamma| \geqslant 1$ (отже, $\gamma \neq \lambda$) та $|\beta| \leqslant n$ (оскільки всі стани до другої появи s_r різні). Більше того, ланцюжок $\beta\gamma^k\omega$ для будь-якого цілого невід'ємного числа k має переводити автомат у той самий заключний стан, що й ланцюжок $\beta\gamma\omega$. Справді, частина γ^k ланцюжка $\beta\gamma^k\omega$ просто переводить стани автомата вздовж петлі, яка починається та закінчується в стані s_r. Ця петля проходиться k разів (рис. 8.25). Звідси випливає, що автомат допускає ланцюжки $\beta\gamma^k\omega$ тому, що він допускає ланцюжок $\beta\gamma\omega$; отже, усі вони належать мові $L(M)$.

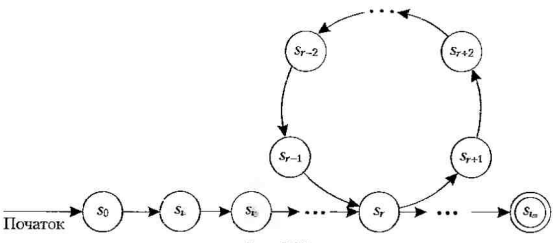


Рис. 8.25

Приклад 8.37. Доведемо, що мова $L = \{0^m1^m \mid m = 0, 1, 2, ...\}$ нерегулярна.

Припустимо, що мова L регулярна. Нехай ланцюжок $\alpha = 0^n1^m$ і $|\alpha| = 2m \geqslant n$. Тоді за лемою про накачування ланцюжки $\alpha = 0^n1^m = \beta\gamma\omega$ та $\beta\gamma^k\omega$ належать мові L, причому $\gamma \neq \lambda$. Ланцюжок γ не може містити водночас нулі й одиниці, бо тоді ланцюжок γ^2 містив би 10, і ланцюжок $\beta\gamma^2\omega$ не належав би мові L. Отже, ланцюжок γ складається чи тільки з нулів, чи тільки з одиниць. Але тоді ланцюжок $\beta\gamma^2\omega$ містить або забагато нулів, або забагато одиниць. Звідси випливає, що ланцюжок $\beta\gamma^2\omega$ не належить мові L. Ця суперечність доводить, що мова $L = \{0^m1^m \mid m = 0, 1, 2, ...\}$ нерегулярна.

Лема про накачування для контекстно вільних мов подібна до леми про накачування для регулярних мов, але кожний ланцюжок α контекстно вільної мови L розбивають на п'ять частин і сумісно „накачують" другу й четверту з них.

ЛЕМА 8.2 (про накачування для контекстно вільних мов). Нехай L – контекстно вільна мова. Тоді існує така залежна від L константа n, що довільний ланцюжок $\alpha \in L$, довжина якого $|\alpha| \geqslant n$, можна так розбити на п'ять ланцюжків $\alpha = \beta\gamma\omega\delta\eta$, що виконуються такі умови.

1. $|\gamma\omega\delta| \leqslant n$ (отже, середня частина не дуже довга).
2. $\gamma\delta \neq \lambda$ (позаяк γ й δ – ланцюжки, які потрібно „накачувати", ця умова означає, що хоча б один із них непорожній).
3. $\beta\gamma^k\omega\delta^k\eta \in L$ для всіх $k \geqslant 0$ (два ланцюжки γ й δ можна „накачати" довільну кількість разів або вилучити, якщо $k = 0$, і отриманий ланцюжок $\beta\gamma^k\omega\delta^k\eta$ також належить мові L).

Приклад 8.38. Доведемо, що мова $L = \{0^m1^m2^m \mid m = 0, 1, 2, ...\}$ не контекстно вільна.

Припустимо, що мова L контекстно вільна. Тоді можна знайти такі ланцюжки β, γ, ω, δ й η, що хоча б один із ланцюжків γ чи δ непорожній, і $\beta\gamma^k\omega\delta^k\eta$ для всіх $k \geqslant 0$ можна подати у вигляді $0^n1^n2^n$. Кожний із ланцюжків γ й δ може бути утворений тільки символом одного типу. Справді, якщо хоча б один із цих двох ланцюжків утворено символами двох або трьох типів, то в ланцюжку $\beta\gamma^2\omega\delta^2\eta$ порушено потрібний порядок символів, тобто він не має вигляду $0^n1^n2^n$. Отже, у ланцюжку $\gamma\delta$ немає принаймні одного символу (наприклад, 0). З іншого боку, оскільки $\gamma\delta \neq \lambda$, то принаймні один символ (наприклад, 1) належить ланцюжку $\gamma\delta$. Тоді для великих k ланцюжок $\beta\gamma^k\omega\delta^k\eta$ містить більше одиниць, ніж нулів, і, отже, не належить мові L. Це суперечить лемі про накачування для контекстно вільних мов. Звідси випливає, що мова $L = \{0^m1^n2^m \mid m = 0, 1, 2, ...\}$ не контекстно вільна.

Контрольні запитання та завдання

1. Нехай $V = \{S, A, B, a, b\}$, $T = \{a, b\}$. Знайти мову, породжену граматикою $G = \{V, T, S, P\}$ із зазначеною множиною P продукцій:

   а) $P = \{S \to AB,\ A \to ab,\ B \to bb\}$;

   б) $P = \{S \to AB,\ S \to aA,\ A \to a,\ B \to ba\}$;

   в) $P = \{S \to AB,\ S \to AA,\ A \to aB,\ A \to ab,\ B \to b\}$;

   г) $P = \{S \to AA,\ S \to B,\ A \to aaA,\ A \to aa,\ B \to bB,\ B \to b\}$;

   д) $P = \{S \to AB,\ A \to aAb,\ B \to bBa,\ A \to \lambda,\ B \to \lambda\}$.

2. Задано граматику $G = \{V, T, S, P\}$, де $V = \{0, 1, S\}$, $T = \{0, 1\}$, $P = \{S \to 0S1, S \to \lambda\}$. Потрібно:

   а) побудувати виведення для ланцюжка $0^3 1^3$;

   б) довести, що ця граматика породжує мову $\{0^m 1^m \mid m = 0, 1, 2, ...\}$.

3. Задано граматику $G_1 = (V. T. S, P)$, де $V = \{S, 0, 1\}$, $T = \{0, 1\}$, $P = \{S \to 0S, S \to S1. S \to \lambda\}$. Потрібно:

   а) побудувати виведення для ланцюжка $0^2 1^4$;

   б) довести, що ця граматика породжує мову $\{0^m 1^n \mid m, n = 0, 1, 2, ...\}$.

4. Задано граматику $G_2 = (V, T, S, P)$, де $V = \{S, A. 0. 1\}$, $T = \{0, 1\}$, $P = \{S \to 0S, S \to 1A, S \to 1. A \to 1A, A \to 1, S \to \lambda\}$. Потрібно:

   а) побудувати виведення для ланцюжка $0^2 1^4$;

   б) довести, що ця граматика породжує ту саму мову, що й граматика G_1 із задачі 3.

5. Побудувати граматики, які породжують такі мови:

   а) $\{01^{2n} \mid n = 0, 1, 2, ...\}$;

   б) $\{0^n 1^{2n} \mid n = 0, 1, 2, ...\}$;

   в) $\{0^n 1^m 0^n \mid m, n = 0, 1, 2, ...\}$.

6. Нехай $V = \{S, A, B, a, b\}$, $T = \{a, b\}$. Множини продукцій P задано нижче. Визначити, чи має граматика $G = (V, T, S, P)$ тип 0, а не 1; 1, а не 2; 2, а не 3, тип 3:

   а) $P = \{S \to aAB. A \to Bb. B \to \lambda\}$;

   б) $P = \{S \to aA, A \to a, A \to b\}$;

   в) $P = \{S \to ABa, AB \to a\}$;

   г) $P = \{S \to ABA, A \to aB, B \to ab\}$;

   д) $P = \{S \to bA, A \to B, B \to a\}$;

   е) $P = \{S \to aA, aA \to B, B \to aA, A \to b\}$;

   ж) $P = \{S \to bA, A \to b, S \to \lambda\}$;

   и) $P = \{S \to aB, B \to aAb, aAb \to b\}$;

   к) $P = \{S \to aA, A \to bB, B \to b, B \to \lambda\}$;

   л) $P = \{S \to A, A \to B, B \to \lambda\}$.

7. *Паліндромом* називають ланцюжок, який однаково читається в прямому та зворотному напрямках. Знайти контекстно вільну граматику, що породжує всі паліндроми над алфавітом $\{0, 1\}$.

8. Нехай G – граматика з $V = \{S, a, b, c\}$, $T = \{a, b, c\}$. S – початковий символ і $P = \{S \to abS, S \to bcS, S \to bbS, S \to a, S \to cb\}$. Побудувати дерево виведення для кожного з ланцюжків:

   а) *bcbba*;

   б) *bbbcbba*;

   в) *bcabbbbbcb*.

9. Нехай G – граматика з $V = \{S, A, B, C, a, b, c\}$, $T = \{a, b, c\}$, S – початковий символ і $P = \{S \to AB, A \to Ca, B \to Ba, B \to Cb, B \to b, C \to cb, C \to b\}$. Застосувати граматичний розбір зверху вниз для визначення того, чи належить кожний із наведених нижче ланцюжків мові, породженій цією граматикою:
   а) *baba*;
   б) *abab*;
   в) *cbaba*;
   г) *bbbcba*.
10. Розв'язати задачу 9 із застосуванням граматичного розбору знизу вверх.
11. Побудувати діаграму станів для скінченного автомата з виходом, заданого таблицею станів.

Стан	f		g	
	Вхід		Вхід	
	0	**1**	**0**	**1**
s_0	s_1	s_0	0	1
s_1	s_0	s_2	0	1
s_2	s_1	s_1	0	0

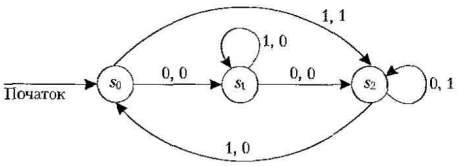
12. Побудувати діаграму станів для скінченного автомата з виходом, заданого таблицею станів.

Стан	f		g	
	Вхід		Вхід	
	0	**1**	**0**	**1**
s_0	s_1	s_0	0	0
s_1	s_2	s_0	1	1
s_2	s_0	s_3	0	1
s_3	s_0	s_2	1	0

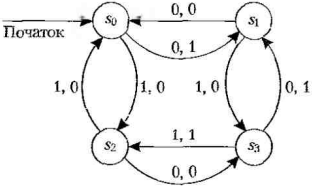
13. Побудувати діаграму станів для скінченного автомата з виходом, заданого таблицею станів.

Стан	f		g	
	Вхід		Вхід	
	0	**1**	**0**	**1**
s_0	s_0	s_4	1	1
s_0	s_0	s_3	0	1
s_2	s_0	s_2	0	0
s_3	s_1	s_0	1	1
s_4	s_1	s_0	1	0

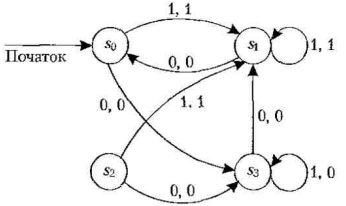
14. За діаграмою станів скінченного автомата з виходом побудувати таблицю станів.



15. За діаграмою станів скінченного автомата з виходом побудувати таблицю станів.



16. За діаграмою станів скінченного автомата з виходом побудувати таблицю станів.



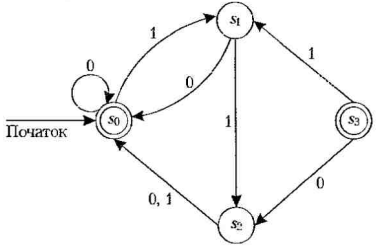
17. Побудувати скінченний автомат із виходом, який затримує вхідний ланцюжок із 0 і 1 на два розряди та видає перші 00 у вихідному ланцюжку.

18. Побудувати скінченний автомат із виходом, у якого вхідний алфавіт {0, 1, *q*}, а вихідний – {0, 1, even, odd}. Вихідний ланцюжок із 0 і 1 збігається з вхідним, а на кожний символ запиту *q* на виході виводиться even, якщо кількість 1 від початку роботи парна, й odd – якщо непарна.

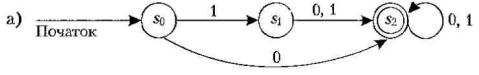
19. Побудувати скінченний автомат для реалізації процедури входу в систему. Користувач уводить ідентифікаційний номер, а потім – на відповідний запит – пароль. Якщо ідентифікаційний номер або пароль некоректні, то виводиться запит ідентифікаційного номера користувача. Ідентифікаційний номер і пароль для спрощення вважати символами (а не ланцюжками).

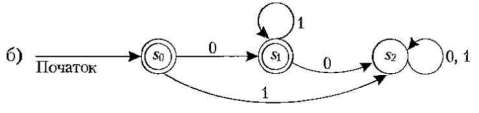
20. Побудувати скінченний автомат із вхідним і вихідним алфавітами {0, 1}. Автомат видає на виході 1, якщо три останні символи на вході були 101, і 0 у всіх інших випадках.
21. Визначити, які з наведених нижче ланцюжків допускає скінченний автомат без виходу:

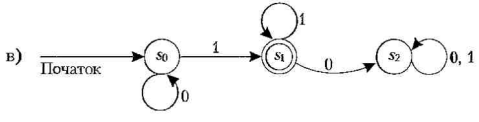
а) 010; б) 1101; в) 1111110; г) 010101010.

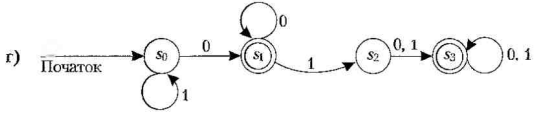


22. Знайти мову, яку розпізнає скінченний автомат без виходу, заданий діаграмою станів.

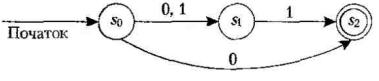
а)


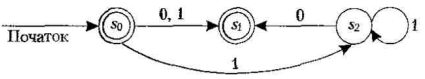
б)


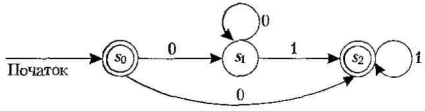
в)


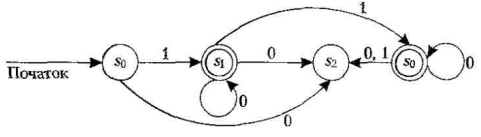
г)


23. Знайти мову, яку розпізнає недетермінований скінченний автомат, заданий діаграмою станів.

а)


б)


в)


г)


24. Для кожного з недетермінованих скінченних автоматів задачі 23 побудувати детермінований скінченний автомат, який розпізнає ту саму мову.

Вказівка. Цю задачу розв'язувати не за схемою доведення теореми 8.1, а з використанням задачі 23, тобто будувати детерміновані автомати за мовами – розв'язками цієї задачі.

25. Побудувати детерміновані скінченні автомати, які розпізнають такі мови:

а) {0}; б) {1, 00}; в) $\{1^n \mid n = 2, 3, 4, \ldots\}$.

26. Побудувати недетерміновані скінченні автомати, які розпізнають мови задачі 25 і, якщо можливо, мають менше станів, ніж детерміновані автомати, побудовані в цій задачі.

27. Описати, з яких ланцюжків складаються регулярні множини:

а) $\mathbf{1^*0}$; б) $\mathbf{1^*00^*}$; в) $\mathbf{111 \cup 001}$; г) $\mathbf{(1 \cup 00)^*}$; д) $\mathbf{(00^*1)^*}$.

28. Виявити належність ланцюжка 1011 до регулярної множини:

а) $\mathbf{10^*1^*}$; б) $\mathbf{0^*(10 \cup 11)^*}$;

в) $\mathbf{1(01)^*1^*}$; г) $\mathbf{1^*01(0 \cup 1)}$;

д) $\mathbf{1(00)^*(11)^*}$; е) $\mathbf{(1 \cup 00)(01 \cup 0)1^*}$.

29. Побудувати недетерміновані скінченні автомати, які розпізнають мови:

а) {λ, 0}; б) {0, 11}; в) {0, 11, 000}.

30. З використанням конструкцій, описаних у доведенні теореми Кліні, побудувати недетерміновані скінченні автомати, які розпізнають такі регулярні множини:

а) $\mathbf{0^*1^*}$; б) $\mathbf{(0 \cup 11)^*}$; в) $\mathbf{01^* \cup 00^*1}$.

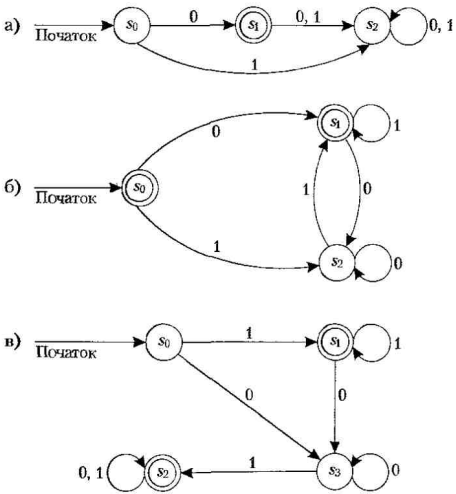
31. Побудувати недетермінований скінченний автомат, який розпізнає мову, породжену регулярною граматикою $G = \{V, T, S, P\}$, де $V = \{0, 1, S, A, B\}$, $T = \{0, 1\}$, S — початковий символ, а множину продукцій задано нижче:

а) $P = \{S \to 0A,\ S \to 1B.\ A \to 0,\ B \to 0\}$;

б) $P = \{S \to 1A,\ S \to 0,\ S \to \lambda,\ A \to 0B,\ B \to 1B,\ B \to 1\}$;

в) $P = \{S \to 1B,\ S \to 0,\ A \to 1A,\ A \to 0B,\ A \to 1,\ A \to 0,\ B \to 1\}$.

32. Побудувати граматику $G = \{V, T, S, P\}$, яка породжує мову, що її розпізнає скінченний автомат.



33. Довести, що скінченний автомат, побудований на основі регулярної граматики в доведенні теореми 8.3, розпізнає множину, породжену цією граматикою.

34. Довести, що регулярна граматика, побудована на основі скінченного автомата в доведенні теореми 8.3, породжує множину, яку розпізнає цей автомат.

35. Довести, що для кожного недетермінованого скінченного автомата існує еквівалентний недетермінований скінченний автомат, у якому немає переходу до початкового стану s_0.
36. Нехай $M = (S, I, f, s_0, F)$ — недетермінований скінченний автомат. Довести, що мова $L(M)$, яку він розпізнає, нескінченна тоді й лише тоді, коли існує ланцюжок x, який допускає автомат, й $l(x) \geqslant |S|$.
37. За допомогою леми про накачування для регулярних мов довести нерегулярність таких мов:

    а) $\{0^{2m}1^m \mid m = 0, 1, 2, ...\}$;

    б) множина ланцюжків, утворених „збалансованими" дужками „(" та „)", які зустрічаються в правильно побудованому арифметичному виразі;

    в) $\{0^m1^m \mid m = 0, 1, 2, ...\}$;

    г) $\{0^m1^n2^m \mid m, n = 0, 1, 2, ...\}$;

    д) $\{0^n1^m \mid n \leqslant m\}$;

    е) $\{1^n \mid n$ — повний квадрат$\}$;

    ж) $\{1^n \mid n$ — повний куб$\}$;

    и) $\{1^n \mid n$ — степінь числа два$\}$;

    к) множина паліндромів над алфавітом {0, 1};

    л) $\{1^p \mid p$ — просте число$\}$.
38. За допомогою леми про накачування для контекстно вільних мов довести, що кожна з наведених нижче мов не є контекстно вільною:

    а) $\{a^mb^nc^k \mid m < n < k\}$;

    б) $\{a^nb^nc^m \mid m \leqslant n\}$;

    в) $\{1^p \mid p$ — просте число$\}$.
39. Побудувати недетермінований скінченний автомат, який розпізнає ключові слова web і ebay.
40. Перетворити недетермінований скінченний автомат із задачі 39 на детермінований.

Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано множину продукцій породжувальної граматики. Визначити тип цієї граматики за класифікацією Хомскі.
2. Задано множину продукцій контекстно вільної граматики й ланцюжок, який належить мові, породженій цією граматикою. Побудувати дерево виведення для заданого ланцюжка.
3. Автомат Мура задано таблицею станів. Побудувати вихідний ланцюжок за заданим вхідним ланцюжком.

4. Автомат Мілі задано таблицею станів. Побудувати вихідний ланцюжок за заданим вхідним ланцюжком.
5. Скінченний детермінований автомат без виходу задано таблицею станів. Визначити, чи допускає цей автомат заданий ланцюжок.
6. Скінченний недетермінований автомат без виходу задано таблицею станів. Визначити, чи допускає цей автомат заданий ланцюжок.
7. Скінченний недетермінований автомат без виходу задано таблицею станів. Побудувати таблицю станів скінченного детермінованого автомата, який розпізнає ту саму мову, що й заданий недетермінований автомат.
8. Задано регулярний вираз. Побудувати недетермінований скінченний автомат, який розпізнає відповідну регулярну множину.
9. Задано регулярну граматику. Побудувати скінченний автомат без виходу, який розпізнає мову, породжену цією граматикою.
10. Задано скінченний автомат без виходу. Побудувати регулярну граматику, що породжує мову, яку розпізнає заданий автомат.

Розділ 9

Основи теорії алгоритмів

- Основні вимоги до алгоритмів
- Машини Тьюрінга
- Обчислення числових функцій на машинах Тьюрінга
- Теза Тьюрінга. Приклади алгоритмічно нерозв'язних проблем
- Рекурсивні функції
- Теза Чорча. Зв'язок рекурсивних функцій із машинами Тьюрінга

Поняття алгоритму — одне з найголовніших у математиці. Глибоке розуміння його потрібне, по-перше, для розробки конкретних алгоритмів, особливо коли потім планують програмувати їх. По-друге, шоб орієнтуватись у великій кількості алгоритмів, треба вміти порівнювати різні алгоритми розв'язування одних і тих самих задач, причому не тільки за якістю розв'язку, але й за характеристиками самих алгоритмів (кількістю операцій, потрібним обсягом пам'яті тощо). Таке порівняння неможливе без уведення точної мови для пояснення цих понять. По-третє, може виникнути потреба довести, що для розв'язання певного класу задач алгоритму взагалі не існує. Отже, слід розглядати алгоритми як об'єкти точного дослідження.

9.1. Основні вимоги до алгоритмів

Строге дослідження основних понять теорії алгоритмів розпочнемо в наступному підрозділі, а тепер неформально розглянемо деякі головні принципи, на яких будують алгоритми, і виявимо, що саме слід уточнити в понятті алгоритму [21].

1. Будь-який алгоритм застосовують до початкових даних, і він видає результати. У звичних технічних термінах це означає, що алгоритм має входи й виходи. Отже, алгоритм застосовують для розв'язування цілого класу задач із різними початковими даними (цю властивість називають *масовістю*). Крім того, під час роботи алгоритму з'являються проміжні результати, використовувані в подальшому. Звідси випливає, що, кожний алгоритм обробляє *дані* — вхідні, проміжні та вихідні. Оскільки ми збираємось уточнити поняття алгоритму, потрібно уточнити й поняття даних, тобто зазначити, яким вимогам мають задовольняти об'єкти, щоб алгоритми могли з ними працювати.
2. Для розміщення даних потрібна пам'ять. Її зазвичай уважають однорідною й дискретною, тобто такою, що складається з однакових комірок, причому кож-

на комірка може містити один символ алфавіту даних. Отже, одиниці виміру обсягу даних і пам'яті узгоджені, при цьому пам'ять може бути нескінченною. Питання про те, чи потрібна одна пам'ять, чи декілька й, зокрема, чи потрібна окрема пам'ять для кожного з трьох типів даних, вирішують по-різному.

3. Алгоритм складається з окремих *елементарних кроків* (або *дій*), причому множина різних кроків, з яких складено алгоритм, скінченна. Типовий приклад множини елементарних дій — система команд процесора. Зазвичай на елементарному кроці обробляється фіксована кількість символів, проте є команди, які працюють із полями пам'яті зі змінною довжиною.
4. Послідовність кроків алгоритму *детермінована*, тобто після кожного кроку зазначено, який крок робити далі, або виконується команда зупинки, після чого робота алгоритму вважається закінченою.
5. Алгоритм має бути *результативним*, тобто зупинятися після скінченної кількості кроків (залежно від початкових даних) із зазначенням того, що вважати результатом. Зокрема, алгоритм для обчислення функції $f(x)$ має зупинятися після скінченної кількості кроків для будь-якого x із області визначення функції f. У такому разі говорять, що алгоритм *збігається*. Проте перевірити результативність (збіжність) значно важче, ніж вимоги, викладені в пп. 1–4. На відміну від них збіжність переважно неможливо виявити простим переглядом опису алгоритму. Загального ж методу перевірки збіжності, придатного для довільного алгоритму A й довільних початкових даних α, узагалі не існує (див. підрозділ 9.4).

Потрібно розрізняти:

- опис алгоритму (інструкцію чи програму);
- механізм реалізації алгоритму (наприклад, персональний комп'ютер), який містить засоби запуску, зупинки, реалізації елементарних кроків, видачі результатів і забезпечення детермінованості, тобто керування перебігом обчислення;
- процес реалізації алгоритму, тобто послідовність кроків, яка буде породжена в разі застосування алгоритму до конкретних даних.

Уважатимемо, що опис алгоритму й механізм його реалізації скінченні (пам'ять, як ми вже зазначали, може бути нескінченною, але вона не входить у механізм). Вимога скінченності процесу реалізації збігається з вимогою результативності (див. п. 5).

Ми сформулювали головні вимоги до алгоритмів. Прості поняття, використані в них, інтуїтивні. Тому в теорії алгоритмів застосовують інший підхід: вибирають скінченний набір основних об'єктів, які оголошують елементарними, і скінченний набір способів побудови з них нових об'єктів. Уточненням поняття „дані" вважатимемо множини слів (ланцюжків) у скінченних алфавітах. Для уточнення детермінізму використовуватимемо опис механізму реалізації алгоритму. Крім того, потрібно зафіксувати набір елементарних кроків і домовитися про організацію пам'яті. Коли це буде зроблено, отримаємо конкретну алгоритмічну модель.

Алгоритмічні моделі, які ми розглянемо в цьому розділі, можна вважати формалізацією поняття „алгоритм". Це означає, що вони мають бути універсальними, тобто за їх допомогою можна описати будь-які алгоритми. Тому може виникнути природне заперечення проти запропонованого підходу: чи не призведе

вибір конкретних засобів до втрати загальності формалізації? Маючи на увазі головні цілі, поставлені під час розробки теорії алгоритмів, — універсальність і пов'язану з нею можливість говорити в рамках якоїсь моделі про властивості алгоритмів узагалі, — це запереження можна зняти так. По-перше, доводять звідність одних моделей до інших, тобто те, що будь-який алгоритм, описаний засобами однієї моделі, можна описати й засобами іншої. По-друге, завдяки взаємній звідності моделей у теорії алгоритмів удалося виробити інваріантну щодо моделей систему понять, яка дає змогу говорити про властивості алгоритмів незалежно від того, яку формалізацію алгоритму вибрано. Ця система ґрунтується на понятті *обчислюваної функції*, тобто такої, для обчислення якої існує алгоритм.

Однак, хоча загальність формалізації в конкретній моделі не втрачається, унаслідок різного вибору початкових засобів виникають різні моделі. Можна виділити три основні типи універсальних алгоритмічних моделей, які різняться початковими евристичними міркуваннями відносно того, що таке алгоритм.

У першому типі поняття алгоритму пов'язане з найтрадиційнішими поняттями математики — обчисленнями та числовими функціями (числовою називають функцію, значення якої та значення її аргументів — невід'ємні цілі числа). Найпопулярніша модель цього типу — *рекурсивні функції* — історично перша формалізація поняття алгоритму.

Другий тип ґрунтується на уявленні про алгоритм як детермінований пристрій, здатний виконувати в кожний окремий момент лише дуже примітивні операції. У такому разі не залишається сумнівів у однозначності алгоритму й елементарності його кроків. Окрім того, евристика цих моделей близька до комп'ютерів. Основна теоретична модель цього типу (створена в 30-х роках XX століття — раніше комп'ютерів) — *машина Тьюрінга*.

Нарешті, третій тип алгоритмічних моделей — це перетворення слів у довільних алфавітах; у цих моделях елементарні операції — це підстановки, тобто заміна частини слова (підслова) іншим словом. Приклади моделей цього типу — *канонічні системи Поста* й *нормальні алгорифми Маркова*.

Розглянемо алгоритмічні моделі двох перших типів; почнемо з машин Тьюрінга.

9.2. Машини Тьюрінга

Машина Тьюрінга — це математична модель пристрою, який породжує обчислювальні процеси. Її використовують для теоретичного уточнення поняття алгоритму та його дослідження. Названо цю модель ім'ям англійського математика А. Тьюрінга (A. Turing), який запровадив її 1936 р. У кожній машині Тьюрінга є три частини (рис. 9.1).

1. Стрічка, поділена на комірки.
2. Пристрій керування (ПК).
3. Головка читання–запису (Г).

Із кожною машиною Тьюрінга пов'язано два скінченні алфавіти: *алфавіт зовнішніх символів A* й *алфавіт внутрішніх станів* $Q = \{q_0, q_1, ..., q_k\}$. Із різними машинами Тьюрінга можна пов'язувати різні алфавіти.

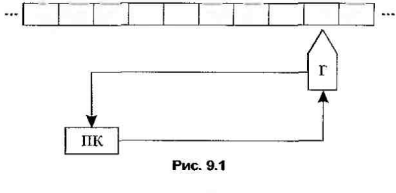


Рис. 9.1

Алфавіт зовнішніх символів A часто називають *зовнішнім*, а його елементи — *буквами*. Один символ з алфавіту A називають *порожнім*, зазвичай його позначають Λ. Усі інші букви з алфавіту A, крім Λ, називають *непорожніми*. Комірку стрічки, у якій записано букву Λ, називають *порожньою*.

Машина Тьюрінга працює в часі, який уважають дискретним; його моменти занумеровано: 1, 2, 3, ... У кожний момент стрічка містить скінченну кількість комірок. Головка пересувається вздовж стрічки; у кожний момент вона перебуває над певною коміркою стрічки. У такому разі говорять, що головка *зчитує* букву, записану в цій комірці. У наступний момент головка залишається над цією самою коміркою (що позначають Н), або пересувається на одну комірку вправо (що позначають П), або пересувається на одну комірку вліво (що позначають Л). Якщо в даний момент t головка перебуває над крайньою коміркою та зсувається на комірку, якої немає, то автоматично прибудовується нова порожня (тобто з порожньою буквою Λ) комірка, над якою головка перебуватиме в момент $t + 1$. Отже, стрічка *потенційно нескінченна в обидва боки*, тобто до неї завжди можна додати нові комірки як зліва, так і справа.

Алфавіт внутрішніх станів $Q = \{q_0, q_1, ..., q_k\}$ — це *внутрішня пам'ять*. На відміну від нескінченної *зовнішньої пам'яті* на стрічці, вона скінченна. Елемент q_0 називають *заключним внутрішнім станом*, а елемент q_1 — *початковим внутрішнім станом*. Пересування головки вздовж стрічки залежить від зчитаної букви та внутрішнього стану машини.

У кожний момент t залежно від букви, зчитуваної зі стрічки, і внутрішнього стану машини пристрій керування виконує такі дії.

1. Змінює букву a_i, зчитувану в момент t зі стрічки, на нову букву a_j (зокрема, може бути $a_j = a_i$).
2. Пересуває головку в одному з напрямків Н, П, Л.
3. Змінює внутрішній стан машини q_i в момент t на новий стан q_p, у якому машина перебуватиме в момент $t + 1$ (зокрема, може бути $q_p = q_i$).

Таку дію пристрою керування називають *командою* й записують

$$q_i a_i \rightarrow a_j D q_p,$$

де q_i — внутрішній стан машини в даний момент; a_i — буква на стрічці, зчитувана в цей момент; a_j — буква, на яку змінюється буква a_i (може бути $a_j = a_i$); D — Н, П чи Л — напрямок пересування головки; q_j — внутрішній стан машини в наступний момент часу (може бути $q_j = q_i$).

Вирази $q_i a_i$ й $a_j D q_j$ називають відповідно лівою та правою частинами цієї команди. У лівій частині немає жодної команди q_0. Усіх команд, у яких ліві частини попарно різні, скінченна кількість (бо множини $Q \setminus \{q_0\}$ й A скінченні); їх сукупність називають *програмою машини*. Жодні дві команди не можуть мати однакові ліві частини.

Виконання однієї команди називають *кроком*. Обчислення (або робота) машини Тьюрінга — це послідовність кроків одного за другим без пропусків, починаючи з першого. Роботу машини Тьюрінга повністю визначено, якщо в перший (тобто початковий) момент задано.

1. Слово на стрічці, тобто послідовність букв, записаних у її комірках (слово одержують читанням цих букв у комірках стрічки зліва направо).
2. Положення головки Г.
3. Внутрішній стан машини.

Сукупність цих трьох умов (у даний момент t) називають *конфігурацією* (у даний момент t). Зазвичай у початковий момент внутрішній стан машини — q_1, а головка перебуває над першою зліва коміркою стрічки.

Отже, у початковий момент конфігурація така: на стрічці, що складається з якоїсь кількості комірок (не менше однієї), у кожній комірці записано одну з букв зовнішнього алфавіту A, головка над першою зліва коміркою стрічки, внутрішній стан машини — q_1. Наприклад, якщо в початковий момент на стрічці записано слово $a_1 \Lambda a_2 a_1 a_1$ то початкова конфігурація така (рис. 9.2):

a_1	Λ	a_2	a_1	a_1
q_1				

Рис. 9.2

(під коміркою, над якою перебуває головка, зазначено внутрішній стан машини). Отже, програма й початкова конфігурація повністю задають роботу машини над словом, яке є на стрічці в початковій конфігурації. Тому вважатимемо машину Тьюрінга заданою, якщо відома її програма.

Якщо в роботі машини Тьюрінга в момент t виконається команда, права частина котрої містить символ q_0, то в такий момент роботу машини вважають завершеною; тоді говорять, що машина *застосовна* до слова на стрічці в початковій конфігурації. Справді, символ q_0 не зустрічається в лівій частині жодної команди, тому його називають заключним станом. Результатом роботи машини в такому разі вважають слово, яке буде записано на стрічці в *заключній конфігурації*, тобто в конфігурації з внутрішнім станом q_0.

Якщо ж у роботі машини в жодний момент не зустрінеться заключний стан, то процес обчислень буде нескінченним. У такому разі говорять, що машина *незастосовна* до слова на стрічці в початковій конфігурації.

Зазначимо, що $q_0 \in Q$ для кожної машини Тьюрінга.

Приклад 9.1. Побудуємо машину Тьюрінга T_1, котра застосовна до всіх слів у алфавіті $\{a, b\}$ та перетворює довільне слово $x_1 x_2 \ldots x_n$, де $x_i \in \{a, b\}$, $i = 1, \ldots, n$ на слово $x_2 \ldots x_n x_1$.

Як зовнішній алфавіт машини візьмемо множину $A = \{\Lambda, a, b\}$; команди разом із коментарями запишемо у два стовпчики. Для запам'ятовування букв алфавіту стрічки потрібно перейти в новий стан.

Команда	Коментар
$q_1a \to \Lambda \Pi q_2$	Перша буква a запам'ятовується переходом у стан q_2 та стирається
$q_1b \to \Lambda \Pi q_3$	Перша буква b запам'ятовується переходом у стан q_3 та стирається
$q_2a \to a \Pi q_2$ $q_2b \to b \Pi q_2$ $q_2\Lambda \to a \mathrm{H} q_0$	Проходження головки через слово $x_2 \ldots x_n$ і запис у його кінці букви $x_1 = a$, зупинка
$q_3a \to a \Pi q_3$ $q_3b \to b \Pi q_3$ $q_3\Lambda \to b \mathrm{H} q_0$	Проходження головки через слово $x_2 \ldots x_n$ і запис у його кінці букви $x_1 = b$, зупинка

Ці команди можна записати у вигляді таблиці, яку називають *функціональною схемою Тьюрінга* (табл. 9.1).

Таблиця 9.1

Стан	Λ	a	b
q_1	–	$\Lambda \Pi q_2$	$\Lambda \Pi q_3$
q_2	$a \mathrm{H} q_0$	$a \Pi q_2$	$b \Pi q_2$
q_3	$b \mathrm{H} q_0$	$a \Pi q_3$	$b \Pi q_3$

У таблиці прочерком позначено неістотну команду, тобто таку, яка не зустрінеться під час роботи цієї машини.

Розглянемо роботу машини T_1 над словом *abb*. Конфігурації, які при цьому виникають, зображено на рис. 9.3.

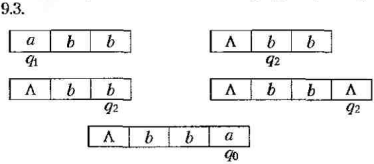


Рис. 9.3

Приклад 9.2. Побудуємо машину Тьюрінга T_2, яка застосовна до всіх слів у алфавіті $\{a, b\}$ та знімає копію слова на стрічці, тобто перетворює слово $x_1 \ldots x_n$ у початковій конфігурації на слово $x_1 \ldots x_n \Lambda x_1 \ldots x_n$ у заключній конфігурації.

Цю машину можна побудувати, наприклад, так. Нехай задано якесь слово в алфавіті $\{a, b\}$, що складається з n букв ($n \geqslant 1$). Робота машини містить n циклів. Для $i \leqslant n$ на початок i-го циклу конфігурацію зображено на рис. 9.4. Отже, скопійовано $i-1$ букв слова, і головка зчитує букву x_i. Черговий цикл виконуватиметься в три етапи.

1. Запам'ятовування букви x_i (при цьому вона позначається, тобто фактично замінюється іншою буквою) та пересування головки вправо.
2. Пошук правої крайньої порожньої комірки, у яку записується x_i.
3. Повернення головки вліво до позначеної букви, стирання мітки (тобто відновлення букви x_i) та пересування головки на одну клітку вправо.

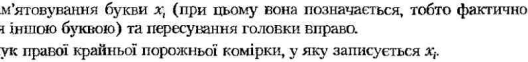


Рис. 9.4

Перший етап реалізують такі команди.

Команда	Коментар
$q_1a \to a'\Pi q_2$	a' — позначена буква a, q_2 — стан запам'ятовування букви a
$q_1b \to b'\Pi q_3$	b' — позначена буква b, q_3 — стан запам'ятовування букви b

Другий етап.

Команда	Коментар
$q_2a \to a\Pi q_2$ $q_2b \to b\Pi q_2$	Проходження залишку слова в стані q_2
$q_3a \to a\Pi q_3$ $q_3b \to b\Pi q_3$	Проходження залишку слова в стані q_3
$q_2\Lambda \to \Lambda\Pi q_4$ $q_3\Lambda \to \Lambda\Pi q_5$	Проходження головки через порожню букву Λ між словами; q_4 – копіюватиметься буква a, q_5 — копіюватиметься буква b
$q_4a \to a\Pi q_4$ $q_4b \to b\Pi q_4$ $q_5a \to a\Pi q_5$ $q_5b \to b\Pi q_5$	Проходження початкової частини копії слова
$q_4\Lambda \to a\Lambda q_6$ $q_5\Lambda \to b\Lambda q_6$	Запис копійованої букви

Третій етап.

Команда	Коментар
$q_6a \to a\Lambda q_6$ $q_6b \to b\Lambda q_6$ $q_6\Lambda \to \Lambda\Lambda q_6$ $q_6a' \to a\ \Pi q_1$ $q_6b' \to b\Pi q_1$	Рух головки вліво до позначеної букви, стирання мітки, пересування на одну клітку вправо з переходом у початковий стан q_1. Машина готова до виконання чергового циклу копіювання букви
$q_1\Lambda \to \Lambda\Pi q_0$	Команда зупинки, копію слова побудовано

Отже, зовнішній алфавіт машини T_2 — множина $A = \{\Lambda, a, b, a', b'\}$, множина внутрішніх станів $Q = \{q_0, q_1, ..., q_6\}$. Нарешті, запишемо команди у вигляді функціональної схеми Тьюрінга (табл. 9.2).

Таблиця 9.2

Стан	Λ	a	b	a'	b'
q_1	$\Lambda\Pi q_0$	$a'\Pi q_2$	$b'\Pi q_3$	–	–
q_2	$\Lambda\Pi q_4$	$a\Pi q_2$	$b\Pi q_2$	–	–
q_3	$\Lambda\Pi q_5$	$a\Pi q_3$	$b\Pi q_3$	–	–
q_4	$a\Lambda q_6$	$a\Pi q_4$	$b\Pi q_4$	–	–
q_5	$b\Lambda q_6$	$a\Pi q_5$	$b\Pi q_5$	–	–
q_6	$\Lambda\Lambda q_6$	$a\Lambda q_6$	$b\Lambda q_6$	$a\Pi q_1$	$b\Pi q_1$

Алфавіт зовнішніх символів A в цьому прикладі крім порожньої букви Λ й букв a та b, у яких записується слово для копіювання, містить іще додаткові букви a', b', використовувані в проміжних обчисленнях.

9.3. Обчислення числових функцій на машинах Тьюрінга

Числовою називають функцію $f(x_1, ..., x_n)$, значення якої та значення її аргументів — невід'ємні цілі числа. Розглянемо часткові числові функції, визначені, узагалі кажучи, не для всіх значень аргументів.

Для обчислення числових функцій на машинах Тьюрінга застосовують спеціальне кодування чисел. Наприклад [30], невід'ємне ціле число m можна задати набором з $(m + 1)$ одиниць, який позначатимемо 1^{m+1}: 0 — як 1, 1 — як 11, 2 — як 111 тощо.

Числову функцію $f(x_1, ..., x_n)$ називають *обчислюваною за Тьюрінгом*, якщо існує така машина Тьюрінга T, що для довільних цілих невід'ємних чисел $m_1, m_2, ..., m_n$, якщо $f(m_1, m_2, ..., m_n) = m$, то машина T застосовна до слова $1^{m_1+1}\Lambda 1^{m_2+1}\Lambda ... \Lambda 1^{m_n+1}$ і в заключній конфігурації на якомусь відрізку стрічки буде записано слово 1^{m+1}, а решта комірок (якщо такі є) виявляться порожніми. Якщо ж значення $f(m_1, m_2, ..., m_n)$ не визначено, то машина T незастосовна до слова $1^{m_1+1}\Lambda 1^{m_2+1}\Lambda ... \Lambda 1^{m_n+1}$.

Приклад 9.3. Побудуємо машину Тьюрінга, яка обчислює числову функцію $s(x) = x + 1$. Зовнішній алфавіт $A = \{\Lambda, 1\}$, множина станів $Q = \{q_0, q_1\}$, а команди можна задати так: $q_1 1 \to 1 1 q_1$, $q_1 \Lambda \to 1 1 q_0$. Конфігурації, що виникають у разі обчислення $s(1)$, зображено на рис. 9.5.

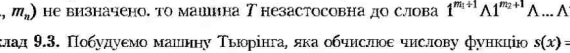


Рис. 9.5

Приклад 9.4. Побудуємо машину Тьюрінга, яка обчислює числову функцію $f(x, y) = x + y$. Зовнішній алфавіт $A = \{\Lambda, 1\}$, множина станів $Q = \{q_0, q_1, q_2, q_3, q_4\}$. У процесі роботи машини головка рухається вправо, і символ Λ між аргументами замінюється на 1. Далі продовжується рух праворуч до першої комірки із символом Λ, після чого головка починає рухатись уліво та стирає дві останні 1 поспіль. Команди можна задати так: $q_1 1 \to 1 1 q_1$, $q_1 \Lambda \to 1 1 q_2$, $q_2 1 \to 1 1 q_2$, $q_2 \Lambda \to \Lambda 1 q_3$, $q_3 1 \to \Lambda 1 q_3$, $q_1 1 \to \Lambda 1 q_0$. Конфігурації, що виникають під час обчислення $f(1, 2)$, зображено на рис. 9.6.

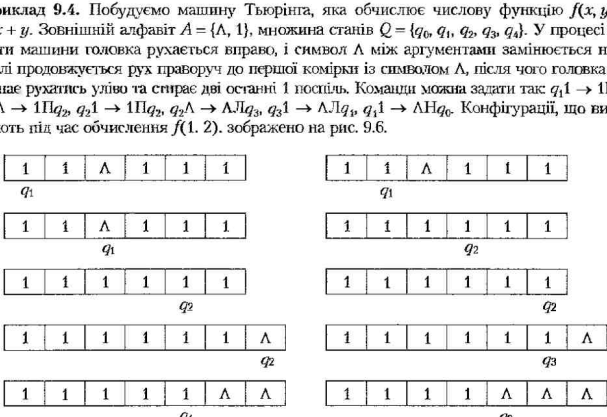


Рис. 9.6

9.4. Теза Тьюрінга. Приклади алгоритмічно нерозв'язних проблем

До цього часу нам удавалося для всіх процедур, які „претендують" на алгоритмічність, тобто *ефективних*, будувати машини Тьюрінга, що їх реалізують. Чи правильно це загалом? Ствердна відповідь на це питання міститься в *тезі Тьюрінга*, яку формулюють так: *будь-який алгоритм можна реалізувати машиною Тьюрінга.*

Довести тезу Тьюрінга неможливо, бо саме поняття алгоритму (ефективної процедури) інтуїтивне (неточне). Це не теорема й не аксіома математичної теорії, а твердження, яке пов'язує теорію з тими об'єктами, для яких її побудовано. За своїм характером теза Тьюрінга нагадує гіпотези фізики про адекватність математичних моделей фізичним явищам і процесам. Підтверджує тезу Тьюрінга, по-перше, математична практика, по-друге — та обставина, що опис алгоритму в термінах будь-якої іншої відомої алгоритмічної моделі можна звести до його опису у вигляді машини Тьюрінга.

Теза Тьюрінга дає змогу, по-перше, замінити неточні (інтуїтивні) твердження про існування ефективних процедур (алгоритмів) точними твердженнями про існування машин Тьюрінга, а, по-друге, глумачити твердження про неіснування машин Тьюрінга як твердження про неіснування алгоритму.

Розглянемо тепер два приклади алгоритмічно нерозв'язних проблем [21, 30].

Проблема самозастосовності. Закодуємо всі машини Тьюрінга символами алфавіту {*, 1} так, щоб за системою команд машини Тьюрінга можна було ефективно побудувати код машини та, навпаки, за кодом можна було ефективно записувати її програму. Це можна зробити, наприклад, так. Нехай T — будь-яка машина Тьюрінга, зовнішній алфавіт якої — $\{\Lambda, a_1, a_2, ..., a_m\}$, а множина внутрішніх станів — $\{q_0, q_1, ..., q_k\}$. Занумеруємо всі символи, що є в командах (окрім →), числами 0, 1, 2, ... так, як це показано в табл. 9.3.

Таблиця 9.3

Символ	Н	Л	П	Λ	a_1	...	a_m	q_0	q_1	...	q_k
Номер	0	1	2	3	4	...	$m+3$	$m+4$	$m+5$	...	$m+4+k$

Число i подаватимемо набором з $i+1$ одиниць. Якщо b — один із символів Н, Л, П, Λ, a_1, ... , a_m, q_0, q_1, ... , q_k, то як $N(b)$ позначимо набір з одиниць, кількість яких відповідає номеру символу b. Наприклад, $N(\text{Н}) = 1$, $N(\text{Л}) = 11$, $N(\text{П}) = 111$, $N(\Lambda) = 1111$, $N(a_1) = 11111$, $N(a_2) = 111111$. Команді $q_i a_i \to a_j D q_j$ поставимо у відповідність слово

$$C(q_i a_i) = N(q_i) * N(a_i) * N(a_j) * N(D) * N(q_j),$$

яке називають *кодом* цієї *команди*. *Кодом машини* T називають слово

$$W(T) = **C(q_1\Lambda)**C(q_1a_1)**...**C(q_1a_m)**C(q_2\Lambda)**C(q_2a_1)**...$$
$$...**C(q_2a_m)**C(q_3\Lambda)**...**C(q_k\Lambda)**...**C(q_ka_m)**$$

(тут у визначеному порядку записано коди всіх команд, відокремлені один від другого двома зірочками).

Розглядатимемо машини Тьюрінга, зовнішній алфавіт яких містить символи * й 1. Нехай T — одна з таких машин. Якщо машина T застосовна до слова $W(T)$, тобто до власного коду, то її називають *самозастосовною*, а ні, то *несамозастосовною*. Кожна машина розглянутого типу або самозастосовна, або несамозастосовна.

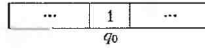


Рис. 9.7

Проблема самозастосовності полягає в тому, що потрібно навести алгоритм для визначення того, чи дана машина Тьюрінга самозастосовна, чи ні. Згідно з тезою Тьюрінга цю задачу можна сформулювати як задачу про побудову машини Тьюрінга, котра застосовна до кодів усіх машин, і заключні конфігурації в разі роботи над кодами самозастосовних і несамозастосовних машин відрізняються. Наприклад, вимагатимемо, щоб у разі роботи над кодом самозастосовної машини заключна конфігурація мала такий вигляд, як на рис. 9.7, а в разі роботи над кодом несамозастосовної машини — як на рис. 9.8. На цих рисунках поза зчитуваним символом можуть бути будь-які символи зовнішнього алфавіту.

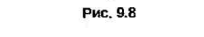


Рис. 9.8

ТЕОРЕМА 9.1. Проблема самозастосовності алгоритмічно нерозв'язна, тобто не існує машини Тьюрінга, яка розв'язує в наведеному вище розумінні цю проблему.

Доведення. Припустимо протилежне. Нехай існує машина L, яка розв'язує проблему самозастосовності. За нею побудуємо машину T, що має такі властивості.

1. Машина T застосовна до кодів несамозастосовних машин.
2. Машина T незастосовна до кодів самозастосовних машин.

Машину T будують так. Усі команди машини L оголошують командами машини T, але заключний стан q_0 машини L оголошують незаключним станом машини T. Будемо вважати заключним станом машини T новий стан q_0'; додамо до системи команд машини T ще такі дві команди:

$$q_0 0 \to 0 \mathrm{H} q_0' \quad \text{і} \quad q_0 1 \to 1 \mathrm{H} q_0.$$

Очевидно, що машина T має зазначені властивості 1 і 2, але вона чи самозастосовна, чи несамозастосовна. Нехай машина T самозастосовна, тобто вона застосовна до свого коду (коду самозастосовної машини), проте це суперечить властивості 2. Якщо ж машина T несамозастосовна, то вона незастосовна до свого коду (коду несамозастосовної машини). Але це суперечить властивості 1. Отже, машина T не може бути ні самозастосовною, ні несамозастосовною. Одержали суперечність. Оскільки машину T побудовано з машини L цілком конструктивно, то машини L не існує.

Проблема зупинки. Серед вимог, яким має задовольняти алгоритм, у підрозділі 9.1 було названо результативність. Найрадикальнішим її формулюванням була б вимога, щоб для будь-якого алгоритму A та даних α можна було визначити, чи приведе робота алгоритму A для початкових даних α до результату. Інакше кажучи, потрібно побудувати такий алгоритм B, що $B(A, \alpha) = 1$, якщо $A(\alpha)$ дає результат, і $B(A, \alpha) = 0$, якщо $A(\alpha)$ не дає результату.

Згідно з тезою Тьюрінга цю задачу можна сформулювати так. За машиною T та словом α розпізнати, чи зупиниться машина T, розпочавши роботу над словом α (тобто розпізнати, чи застосовна машина T до слова α). Отже, потрібно побудувати таку машину P, яка була б застосовна до всіх слів $W(T)\Lambda\alpha$, де T — довільна машина, α — довільне слово. Окрім того, заключні конфігурації машини P мають бути різними залежно від того, чи застосовна машина T до слова α, чи незастосовна.

ТЕОРЕМА 9.2. Проблема зупинки алгоритмічно нерозв'язна, тобто не існує машини Тьюрінга P, яка розв'язує в наведеному вище розумінні названу проблему.

Доведення. Нехай існує машина P, яка розв'язує проблему зупинки, а T — якась машина Тьюрінга. Тоді машина P має бути застосовна до всіх слів $W(T)\Lambda\alpha$, зокрема якщо $\alpha = W(T)$. Отже, машина P має бути застосовна до слова $W(T)\Lambda W(T)$. У разі роботи над словом $W(T)\Lambda W(T)$ вона зупиниться в конфігурації, зображеній на рис. 9.7, якщо машина T застосовна до слова $W(T)$ (тобто якщо машина T самозастосовна), і в конфігурації, зображеній на рис. 9.8, якщо машина T незастосовна до слова $W(T)$ (тобто якщо машина T несамозастосовна).

Нагадаємо, що машина, яка розв'язує проблему самозастосовності, починає працювати зі словом $W(T)$ на стрічці. Отже, щоб із машини P отримати машину, яка розв'язує проблему самозастосовності, використаємо машину T_2 з прикладу 9.2 (уважатимемо, що a — це *, b — 1). Машина T_2 за такої домовленості перетворює будь-яке слово β в алфавіті {*,1} на слово $\beta\Lambda\beta$. Машину Тьюрінга, яка розв'язує проблему самозастосовності, можна одержати з машин T_2 та P: спочатку працює машина T_2, а потім — машина P. Але це суперечить теоремі 9.1. Отже, машини P не існує.

Тлумачачи твердження, пов'язані з алгоритмічною нерозв'язністю, потрібно звернути увагу на те, що в них ідеться про те, що немає єдиного алгоритму, який розв'язує певну проблему. При цьому зовсім не виключено, що її можна розв'язати в часткових випадках, але за допомогою різних процедур для кожного з них. Тому нерозв'язність загальної проблеми зупинки зовсім не знімає потреби доводити збіжність пропонованих алгоритмів, а лише показує, що пошук таких доведень не можна повністю автоматизувати.

Нерозв'язність проблеми зупинки можна інтерпретувати як неіснування загального алгоритму для налагодження програм, котрий за текстом довільної програми й даними для неї міг би визначити, чи зациклиться програма на цих даних. Якщо врахувати зроблене перед цим зауваження, то така інтерпретація не суперечить тому емпіричному факту, що більшість програм удається налагоди-

ти, тобто виявити зациклення, знайти його причину й усунути її. При цьому вирішальну роль відіграють досвід і майстерність програміста.

9.5. Рекурсивні функції

Будь-який алгоритм однозначно ставить у відповідність початковим даним результат (якщо він визначений на них). Тому з кожним алгоритмом однозначно пов'язана функція, значення якої він обчислює. Чи правильне обернене твердження, тобто чи для будь-якої функції існує алгоритм, який обчислює її значення? Дослідження проблеми зупинки для машини Тьюрінга показує, що ні: не існує алгоритму обчислення значення предиката $P(T, \alpha)$, істинного якщо й лише якщо машина Тьюрінга T зупиняється на початкових даних α. Виникає запитання: для яких функцій існують алгоритми обчислення їх значень? Як описати такі алгоритмічні функції, тобто ефективно обчислювані функції?

Дослідження цих питань зумовило створення в 30-х роках XX століття теорії рекурсивних функцій. У ній, як і в теорії алгоритмів узагалі, прийнято конструктивний, фінітний підхід. Основна риса такого підходу полягає в тому, що вся множина об'єктів дослідження (у цьому разі – функцій) будується зі скінченної кількості вихідних об'єктів за допомогою простих операцій, ефективна виконуваність яких достатньо очевидна.

Розглядатимемо числові функції, тобто такі, що набувають значень — так само, як і їх аргументи — із множини $N_0 = \{0, 1, 2, ...\}$.

Назвемо найпростішими такі числові функції:

- *нуль-функцію*: $o(x) = 0$ для кожного x;
- *функцію наступності*: $s(x) = x + 1$ для кожного x;
- *функції вибору аргументів (селективні функції)*: $I_m^{(n)}(x_1, x_2, ..., x_n) = x_m$ для всіх $x_1, x_2, ..., x_n$, $m = 1, 2, ..., n$; $n = 1, 2, ...$.

Функції вибору аргументів — це $I_1^{(2)}(x_1, x_2) = x_1$, $I_1^{(1)}(x_1) = x_1$, $I_2^{(2)}(x_1, x_2) = x_2$, $I_2^{(3)}(x_1, x_2, x_3) = x_2$ та інші.

Тепер уведемо оператори для отримання нових функцій із тих, що вже є.

Оператор суперпозиції. Нехай задано числові функції $f(x_1, ..., x_m)$ і $g_1(x_1, ..., x_n)$, $g_2(x_1, ..., x_n)$, ..., $g_m(x_1, ..., x_n)$, і нехай $h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$. Тоді говорять, що функцію h одержано за допомогою *оператора суперпозиції* з функцій $f, g_1, g_2, ..., g_m$.

Приклад 9.5. Функцію $o(x_1, ..., x_n) = 0$ отримано за допомогою суперпозиції з функцій $o(x)$ та $I_1^{(n)}(x_1, ..., x_n)$:

$$o(x_1, ..., x_n) = o(I_1^{(n)}(x_1, ..., x_n)).$$

Функцію $s_m^{(n)}(x_1, ..., x_n) = x_m + 1$ одержано за допомогою оператора суперпозиції з функцій $s(x)$ та $I_m^{(n)}(x_1, ..., x_n)$:

$$s_m^{(n)}(x_1, ..., x_n) = s((I_m^{(n)}(x_1, ..., x_n)).$$

Оператор примітивної рекурсії. Нехай $g(x_1, ..., x_{n-1})$ і $h(x_1, ..., x_{n-1}, x_n, x_{n+1})$ – дві числові функції, причому $n \geqslant 2$. Означимо третю функцію $f(x_1, ..., x_{n-1}, x_n)$ за допомогою такої схеми:

$$f(x_1, ..., x_{n-1}, 0) = g(x_1, ..., x_{n-1});$$
$$f(x_1, ..., x_{n-1}, y+1) = h(x_1, ... x_{n-1}, y, f(x_1, ..., x_{n-1}, y)), y \geqslant 0.$$

Її називають *схемою примітивної рекурсії для функції* $f(x_1, ..., x_n)$ *за змінними* x_n та x_{n+1}; вона задає *примітивно рекурсивний опис* функції $f(x_1, ..., x_n)$ за допомогою функцій g і h. Говорять також, що функцію f отримано з функцій g й h за допомогою *оператора примітивної рекурсії* (за змінними x_n та x_{n+1}).

У разі примітивно рекурсивного опису функції $f(x)$, залежної від однієї змінної, схема примітивної рекурсії має такий вигляд:

$$\begin{cases} f(0) = a, \\ f(y+1) = h(y, f(y)), \quad y \geqslant 0, \end{cases}$$

де a – константа, $a \in N_0$. У такому разі говорять, що функцію $f(x)$ одержано за допомогою оператора примітивної рекурсії з константи a та функції h.

Приклад 9.6. Функцію $f_+(x, y) = x + y$ можна отримати за допомогою примітивної рекурсії з функцій $I_1^{(1)}(x)$ і $s_3^{(3)}(x, y, z)$. Справді,

$$f_+(x, 0) = x = I_1^{(1)}(x);$$
$$f_+(x, y+1) = f_+(x, y) + 1 = s_3^{(3)}(x, y, f_+(x, y)).$$

Функцію називають *примітивно рекурсивною*, якщо її можна одержати з найпростіших функцій за допомогою скінченної кількості застосувань операторів суперпозиції та примітивної рекурсії.

Отже, функції з прикладів 9.5 і 9.6 примітивно рекурсивні. Клас усіх примітивно рекурсивних функцій позначатимемо $K_{\text{пр}}$.

Продовжимо розглядати приклади примітивно рекурсивних функцій.

Приклад 9.7. Множення $f_*(x, y) = xy$ примітивно рекурсивне:

$$f_*(x, 0) = 0,$$
$$f_*(x, y+1) = f_*(x, y) + x = f_+(x, f_*(x, y)).$$

Приклад 9.8. Піднесення до степеня $f_{\text{exp}}(x, y) = x^y$ примітивно рекурсивне:

$$f_{\text{exp}}(x, 0) = 1;$$
$$f_{\text{exp}}(x, y+1) = x^y x = f_*(x, f_{\text{exp}}(x, y)).$$

Визначимо функцію $x \dot{-} y$ (арифметичне віднімання):

$$x \dot{-} y = \begin{cases} x - y, & \text{якщо } x > y; \\ 0 & \text{у протилежному випадку.} \end{cases}$$

Приклад 9.9. Функція $f(x) = x \dot{-} 1$ примітивно рекурсивна:

$$f(0) = 0; \quad f(y+1) = y.$$

Приклад 9.10. Функція $f(x, y) = x \dot{-} y$ примітивно рекурсивна:

$$f(x, 0) = x; \quad f(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = f(x, y) \dot{-} 1$$

(тут використано функцію з прикладу 9.9).

Приклад 9.11. Функція

$$f(x, y) = |x - y| = (x \dot{-} y) + (y \dot{-} x)$$

примітивно рекурсивна.

Приклад 9.12. Функція

$$\operatorname{sg}(x) = \begin{cases} 0, \text{ якщо } x = 0, \\ 1, \text{ якщо } x \neq 0 \end{cases}$$

примітивно рекурсивна. Схема примітивної рекурсії тут така:

$$\operatorname{sg}(0) = 0;$$
$$\operatorname{sg}(x+1) = 1.$$

Приклад 9.13. Функції

$$\min(x, y) = x \dot{-} (x \dot{-} y),$$
$$\max(x, y) = y + (x \dot{-} y)$$

примітивно рекурсивні.

За допомогою функції sg із прикладу 9.12 та функції $\overline{sg} = 1 \dot{-} \operatorname{sg}$ побудуємо примітивно рекурсивний опис функцій, пов'язаних із діленням.

Приклад 9.14. Функція $r(x, y)$ — остача від ділення y на x — примітивно рекурсивна:

$$r(x, 0) = 0;$$
$$r(x, y+1) = (r(x, y) + 1)\operatorname{sg}(|x - (r(x, y) + 1)|).$$

Зміст другого рядка в схемі такий: якщо число $y + 1$ не ділиться на x, то

$$\operatorname{sg}(|x - (r(x, y) + 1)|) = 1, \quad r(x, y+1) = r(x, y) + 1;$$

якщо ж число $y + 1$ ділиться на x, то

$$r(x, y+1) = \operatorname{sg}(|x - (r(x, y) + 1)|) = 0.$$

Приклад 9.15. Функція $q(x, y) = \lfloor y/x \rfloor$ — частка від ділення числа y на x, тобто ціла частина дробу y/x — примітивно рекурсивна:

$$q(x, 0) = 0;$$
$$q(x, y+1) = q(x, y) + \overline{\operatorname{sg}}(|x - (r(x, y) + 1)|).$$

У другому рядку схеми другий доданок залежить від подільності числа $y + 1$ на x. Якщо воно ділиться на x, то $q(x, y + 1)$ на одиницю більше ніж $q(x, y)$; якщо ні, то $q(x, y + 1) = q(x, y)$.

Оператор мінімізації. Нехай $f(x_1, ..., x_{n-1}, x_n)$, $n \geqslant 1$ — числова функція. Означимо функцію $g(x_1, ..., x_{n-1}, x_n)$ так. Нехай $(m_1, ..., m_{n-1}, m_n)$ — довільний набір цілих невід'ємних чисел. Розглянемо рівняння

$$f(m_1, m_2, ..., m_{n-1}, y) = m_n. \tag{9.1}$$

Якщо воно має розв'язки в невід'ємних цілих числах, то візьмемо мінімальний із цих розв'язків і позначимо його μ_y. Якщо при цьому $f(m_1, ..., m_{n-1}, 0)$, ..., $f(m_1, ..., m_{n-1}, \mu_y - 1)$ визначено, то вважаємо, що

$$g(m_1, m_2, ..., m_{n-1}, m_n) = \mu_y.$$

Якщо ж або рівняння (9.1) не має розв'язків у невід'ємних цілих числах, або хоча б одне зі значень $f(m_1, ..., m_{n-1}, 0)$, ..., $f(m_1, ..., m_{n-1}, \mu_y - 1)$ невизначено, то $g(m_1, m_2, ..., m_{n-1}, m_n)$ невизначено.

Про функцію $g(x_1, ..., x_n)$, побудовану зазначеним способом, говорять, що її одержано з функції $f(x_1, ..., x_{n-1}, x_n)$ *застосуванням оператора мінімізації за змінною* x_n (або за допомогою *мінімізації за* x_n). У такому разі використовують позначення

$$g(x_1, ..., x_{n-1}, x_n) = \mu_y[f(x_1, ..., x_{n-1}, y) = x_n].$$

Оператор мінімізації називають також *μ-оператором*.

Функцію називають *частково рекурсивною*, якщо її можна отримати з найпростіших функцій за допомогою скінченної кількості застосувань операторів суперпозиції, примітивної рекурсії та мінімізації. Клас усіх частково рекурсивних функцій позначимо $K_{чр}$.

Частково рекурсивна функція може бути не всюди визначеною. Усюди визначену частково рекурсивну функцію називають *загальнорекурсивною*. Клас усіх загальнорекурсивних функцій позначатимемо як $K_{зр}$.

Кожна примітивно рекурсивна функція всюди визначена. Для множин $K_{пр}$, $K_{зр}$ та $K_{чр}$ виконуються співвідношення $K_{пр} \subset K_{зр} \subset K_{чр}$, причому всі включення строгі.

Приклад 9.16. Нехай $f(x) = x + 1$. За допомогою оператора мінімізації означимо функцію $g(x) = \mu_y[f(y) = x]$. Рівняння (9.1) набирає вигляду $y + 1 = x$. Отже, функцію $g(x)$ невизначено, якщо $x = 0$, і $g(x) = x - 1$, якщо $x > 0$.

Приклад 9.17. Доведемо, що функція

$$g(x_1, x_2) = \frac{x_1}{1 \dot{-} x_1 x_2} \tag{9.2}$$

частково рекурсивна. Застосуємо оператор мінімізації за змінною x_1 до примітивно рекурсивної функції $f(x_1, x_2) = x_1(1 \dot{-} x_2)$. Тоді $g(x_1, x_2) = \mu_y[f(y, x_2) = x_1]$. Справді, рівняння (9.1) набирає вигляду

$$y(1 \dot{-} x_2) = x_1.$$

Отже,

$$g(x_1, x_2) = \begin{cases} 0, & \text{якщо } x_1 = 0, \\ x_1, & \text{якщо } x_2 = 0, \\ \text{невизначено, якщо } x_1 \neq 0 \text{ й } x_2 \neq 0. \end{cases}$$

або в аналітичному вигляді (9.2).

9.6. Теза Чорча. Зв'язок рекурсивних функцій із машинами Тьюрінга

Поняття частково рекурсивної функції — вичерпна формалізація поняття обчислюваної функції. Це можна подати як тезу А. Чорча (A. Church): *будь-яка функція, обчислювана якимось алгоритмом, частково рекурсивна.*

Теза Чорча — аналог тези Тьюрінга для рекурсивних функцій, але її було сформульовано раніше. Тезу Чорча, як і тезу Тьюрінга, не доводять. З їх зіставлення випливає твердження, яке являє собою теорему, тому її потрібно довести.

ТЕОРЕМА 9.3. Функція обчислювана за Тьюрінгом тоді й лише тоді, коли вона частково рекурсивна.

Доведення цієї теореми досить громіздке. Тут ми коротко опишемо лише його план.

Нехай функція частково рекурсивна. Потрібно довести, що вона обчислювана за Тьюрінгом. Спочатку доводять, що найпростіші функції обчислювані за Тьюрінгом (це досить очевидно), а потім — що оператори суперпозиції, примітивної рекурсії та мінімізації зберігають обчислюваність.

Нехай функція обчислювана за Тюрінгом. Потрібно довести, що вона частково рекурсивна. Таке твердження може видатися дуже слабким для того, щоб говорити про еквівалентність рекурсивних функцій, які мають справу із числами з розширеного натурального ряду, і машин Тьюрінга, які можуть не тільки обчислювати. Проте від будь-яких об'єктів можна перейти до числових за допомогою кодування, до того ж украй простого. З погляду машини, яка не вкладає в символи ніякого змісту, а лише виконує з ними задані операції, немає особливої різниці між числами та „нечислами". Свої елементарні дії — читання, запис, зсув, зміну стану — машина може виконувати за наявності на стрічці як цифр, так і інших символів. Різниця лише в інтерпретації отриманих результатів. Зокрема, нічо не заважає інтерпретувати будь-який зовнішній алфавіт (алфавіт стрічки) $A = \{a_1, a_2, ..., a_m\}$ як множину цифр m-кової системи числення, а слово, записане на стрічці, — як m-ковс число. Тоді будь-яку роботу машини Тьюрінга можна розглядати як перетворення чисел, тобто пошук значень числової функції. Саме таку інтерпретацію тут і беруть до уваги. Докладне доведення полягає в тому, щоб точно описати цю інтерпретацію (її називають *арифметизацією*) та показати, що будь-яке перетворення, реалізоване машиною Тьюрінга, частково рекурсивне, якщо інтерпретувати його як обчислення.

Контрольні запитання та завдання

1. Побудувати машину Тьюрінга, яка перетворює слово α на слово β в алфавіті $\{0, 1\}$:
   а) $\alpha = 1^n$, $\beta = 1^n 01^n$;
   б) $\alpha = 0^n 1^n$, $\beta = (01)^n$;
   в) $\alpha = 1^n$, $\beta = 1^{2n-1}$;
   г) $\alpha = 1^n 01^m$, $\beta = 1^m 01^n$.
2. Побудувати машину Тьюрінга, яка за будь-яким вхідним ланцюжком $0^n 1^m$ визначає, чи виконується умова $n = m$. Чи можна звідси дійти висновку, що машина Тьюрінга може більше, ніж скінченний автомат? Відповідь обґрунтувати.
3. Побудувати машину Тьюрінга, яка кожне слово $x_1x_2 \dots x_{n-1}x_n$ в алфавіті $\{a, b\}$ перетворює на слово $x_nx_{n-1} \dots x_2x_1$.
4. Побудувати машину Тьюрінга, яка обчислює *предикат симетрії* $S(x)$: для будь-якого слова $\alpha = x_1x_2 \dots x_{n-1}x_n$ в алфавіті $\{a, b\}$ $S(\alpha) = 1$, якщо $x_i = x_{n-i+1}$ для всіх $i = 1, 2, \dots, n$, а ні, то $S(\alpha) = 0$.
5. Побудувати машину Тьюрінга, застосовну до слів 1^{3^n} ($n \geqslant 1$) та незастосовну до слів 1^{3^n+m}, де $n \geqslant 1$; $m = 1, 2$.
6. Побудувати машину Тьюрінга, яка обчислює числову функцію
   а) $I_3^{(4)}(x_1, x_2, x_3, x_4) = x_3$.
   б) $x \dot{-} y = \begin{cases} x - y, & \text{якщо } x > y, \\ 0 & \text{у протилежному випадку.} \end{cases}$
   в) $\operatorname{sg}(x) = \begin{cases} 0, & \text{якщо } x = 0, \\ 1, & \text{якщо } x \neq 0. \end{cases}$
   г) $\overline{\operatorname{sg}}(x) = \begin{cases} 1, & \text{якщо } x = 0, \\ 0, & \text{якщо } x \neq 0. \end{cases}$
   д) $f(x, y) = \min(x, y)$.
   е) $f(x) = 2x + 1$.
7. Побудувати машину Тьюрінга, яка обчислює числову функцію $f(x) = \lfloor x/2 \rfloor = m$, якщо $x = 2m$ або $x = 2m + 1$, $m \geqslant 0$.
8. Чи застосовні машини Тьюрінга T_1 і T_2, задані командами

$$T_1\colon q_1 1 \to 1\Pi q_1,\ q_1 0 \to 0\Pi q_0,\ q_1\Lambda \to \Lambda\Pi q_1;$$
$$T_2\colon q_1 1 \to \Lambda\Pi q_1,\ q_1 0 \to 1\Pi q_2,\ q_1\Lambda \to 1\Pi q_0,$$
$$q_2 1 \to \Lambda\Pi q_1,\ q_2 0 \to 0\Pi q_2,\ q_2\Lambda \to 0\Pi q_1,$$

   до таких слів:
   а) 1111111; б) 0110111; в) 111101; г) 001101?

9. Застосувавши оператор примітивної рекурсії за змінними x_2 та x_3 до функцій $g(x_1)$ і $h(x_1, x_2, x_3)$, одержати функцію $f(x_1, x_2)$. Записати функцію $f(x_1, x_2)$ аналітично:

   а) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = x_1 + x_3$;

   б) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = x_1 + x_2$;

   в) $g(x_1) = 2^{x_1}$, $h(x_1, x_2, x_3) = x_3^{x_1}$ (тут уважати, що $0^0 = 1$);

   г) $g(x_1) = 1$, $h(x_1, x_2, x_3) = x_3\,(1 + \operatorname{sg}|x_1 + 2 - 2x_3|)$;

   д) $g(x_1) = x_1$, $h(x_1, x_2, x_3) = (x_3 + 1)\operatorname{sg}\left(1 + \dfrac{x_3}{3}\right)$.

10. Застосувати оператор мінімізації до функції f за змінною x_i. Вислідну функцію подати аналітично:

   а) $f(x_1) = 3$, $i = 1$;

   б) $f(x_1) = \lfloor x_1/2 \rfloor$, $i = 1$;

   в) $f(x_1, x_2) = x_1 \div x_2$, $i = 1, 2$;

   г) $f(x_1, x_2) = I_1^{(2)}(x_1, x_2)$, $i = 2$;

   д) $f(x_1, x_2) = x_1 \dot{-} x_2$, $i = 1, 2$.

11. Застосувавши операцію мінімізації до відповідної примітивно рекурсивної функції, довести, що функція f частково рекурсивна:

   а) $f(x_1) = 2 - x_1$;

   б) $f(x_1) = x_1/2$;

   в) $f(x_1, x_2) = x_1 - 2x_2$.

Комп'ютерний проект

Скласти програму із зазначеними вхідними даними та результатами.

Дано машину Тьюрінга та слово в початковій конфігурації. Знайти слово в заключній конфігурації (уважати, що машина застосовна до слова, заданого в початковій конфігурації).

Розділ 10

Комбінаторні задачі та складність обчислень

- Масові задачі, алгоритми та складність
- Задачі розпізнавання, мови та кодування
- Детерміновані машини Тьюрінга та клас *P*
- Недетерміновані машини Тьюрінга та клас *NP*
- Поліноміальна звідність і *NP*-повні задачі
- Теорема Кука
- Приклади *NP*-повних задач. Доведення *NP*-повноти

У цьому розділі розглянуто проблему складності розв'язування комбінаторних задач, які виникають у дискретній оптимізації, математичному програмуванні, алгебрі, теорії чисел, теорії автоматів, математичній логіці, теорії графів тощо. Започаткував теорію складності С. Кук (S. Cook) у праці „The complexity of theorem-proving procedures", опублікованій 1971 р.

10.1. Масові задачі, алгоритми та складність

У 30-х роках XX століття Тьюрінг дослідив абстрактну машину (див. розділ 9), котра, принаймні щодо обчислень, мала всі можливості сучасних комп'ютерів. Метою Тьюрінга було точно описати межу між тим, що обчислювальна машина може робити, і тим, чого вона не може. Через сорок років Кук розвинув результати Тьюрінга. Йому вдалося поділити задачі на ті, котрі може ефективно розв'язати комп'ютер, і ті, які, у принципі, розв'язні, але потребують для цього так багато часу, що комп'ютер виявляється непридатним для розв'язання всіх „екземплярів" задачі, окрім невеликих. Задачі такого класу називають важкорозв'язними. Навіть у разі зростання швидкодії комп'ютерів на порядки дуже малоймовірно, що нам удасться досягти значних успіхів у точному розв'язанні задач із цього класу. Машини Тьюрінга дають змогу усвідомити принципові можливості програмного забезпечення. Зокрема, за допомогою теорії складності обчислень можна визначити, чи можливо розв'язати задачу „в лоб" і написати відповідну програму для її розв'язання (якщо ця задача не належить до класу важкорозв'язних), чи нам потрібно шукати розв'язок даної важкорозв'язної задачі „в обхід",

застосовуючи наближений, евристичний або якийсь інший метод, котрий може обмежити час, потрібний програмі на відшукання розв'язку.

Ми вже зазначали, що алгоритм застосовують для розв'язання цілого класу задач із різними початковими даними. Під *масовою задачею* (або просто *задачею*) розуміють якесь загальне запитання, на яке потрібно дати відповідь. Задача містить декілька *параметрів*, або вільних змінних, конкретні значення котрих не визначено. Задачу Π задають так.

1. Загальним списком усіх її параметрів.
2. Формулюванням тих властивостей, які має задовольняти відповідь (розв'язок задачі).

Індивідуальну задачу I можна одержати з масової задачі Π, присвоївши всім параметрам задачі Π конкретні значення. Як приклад розглянемо класичну *задачу комівояжера*. Параметри цієї масової задачі — скінченна множина міст $C = \{c_1, c_2, ..., c_m\}$ і віддалей $d(c_i, c_j)$ між кожною парою міст c_i, c_j із C. Розв'язок — це такий упорядкований набір $(c_{s(1)}, c_{s(2)}, ..., c_{s(m)})$ заданої множини міст (перестановка), який мінімізує величину

$$\sum_{i=1}^{m-1} d(c_{s(i)}, c_{s(i+1)}) + d(c_{s(m)}, c_{s(1)}).$$

Це довжина шляху, який починається в місті $c_{s(1)}$, один раз проходить послідовно через усі міста й повертається в $c_{s(1)}$ безпосередньо з останнього міста $c_{s(m)}$.

Приклад 10.1. Індивідуальну задачу комівояжера зображено у вигляді зваженого графа на рис. 10.1. Тут $C = \{c_1, c_2, c_4, c_3\}$, $d(c_1, c_2) = 10$, $d(c_1, c_3) = 5$, $d(c_1, c_4) = 9$, $d(c_2, c_3) = 6$, $d(c_2, c_4) = 9$, $d(c_3, c_4) = 3$.

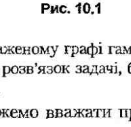


Рис. 10.1

Отже, потрібно знайти в цьому зваженому графі гамільтонів цикл із найменшою довжиною. Послідовність (c_1, c_2, c_4, c_3) — розв'язок задачі, бо відповідний гамільтонів цикл має найменшу можливу довжину — 27.

Для визначеності алгоритм можемо вважати програмою для комп'ютера, написаною формальною мовою. Говорять, що алгоритм *розв'язує* масову задачу Π, якщо він застосовний до будь-якої індивідуальної задачі I з Π й обов'язково видає розв'язок задачі I.

Загалом, нам потрібен „найкращий" алгоритм розв'язання задачі. У найширшому розумінні поняття якості алгоритму пов'язане з усіма обчислювальними ресурсами, потрібними для його роботи. Під „найкращим" алгоритмом розуміють

переважно найшвидший. Оскільки обмеження за часом — це здебільшого головний чинник, від якого залежить придатність конкретного алгоритму для практики, зосередимося на цьому ресурсі.

Час роботи алгоритму зручно виражати у вигляді функції однієї змінної. Ця єдина змінна характеризує „розмір" індивідуальної задачі, тобто обсяг вхідних даних, потрібних для опису задачі. Далі порівняльну складність задачі будемо оцінювати через її розміри.

Опис індивідуальної задачі, який ми формулюємо в термінах входу для комп'ютера, можна розглядати як один скінченний ланцюжок символів скінченного вхідного алфавіту. Будемо вважати, що з кожною масовою задачею пов'язана якась фіксована *схема кодування*, котра відображає індивідуальні задачі у відповідні ланцюжки символів. *Вхідну довжину* індивідуальної задачі **I** з Π визначають як кількість символів у ланцюжку, отриманому застосуванням до задачі **I** схеми кодування для масової задачі Π. Саме цю величину використовують як формальну характеристику розміру індивідуальної задачі.

Часова складність алгоритму відображає потрібні для його роботи витрати часу. Це функція, яка кожній вхідній довжині n ставить у відповідність максимальний (за всіма індивідуальними задачами довжиною n) час, який витрачає алгоритм на розв'язання індивідуальних задач із цією довжиною. Зрозуміло, що цю функцію не буде повністю визначено доти, доки не зафіксовано схему кодування, від якої залежить довжина індивідуальної задачі, і не вибрано обчислювальний пристрій (або його модель), від якого залежить час роботи, проте ці чинники незначно впливають на складність алгоритму. Тому вважатимемо, що зафіксовано якусь конкретну схему кодування для кожної (масової) задачі й вибрано якийсь конкретний обчислювальний пристрій або його модель. Після цього розглядають часову складність алгоритмів відповідно до одержаних вхідних довжин і витрат часу.

Різні алгоритми мають різну часову складність. Нагадаємо, що функція $f(n)$ — це $O(g(n))$, якщо існує така константа c, що $f(n) \leqslant cg(n)$ для всіх значень $n \geqslant n_0$ (тут f і g набувають невід'ємних значень, n_0 — натуральне число). *Поліноміальний алгоритм* (або *алгоритм із поліноміальною часовою складністю*) — це алгоритм, часова складність якого дорівнює $O(p(n))$. Тут $p(n)$ — якийсь поліном, а n — вхідна довжина. Алгоритми, часова складність яких не піддається подібній оцінці, називають *експоненціальними*. Отже, до експоненціальних відносять і алгоритми, складність яких можна оцінити, наприклад, функцією $n^{\log n}$, яка не є експонентою.

Різниця між поліноміальними й експоненціальними алгоритмами дуже помітна, якщо проаналізувати вплив збільшення швидкодії комп'ютера на час роботи алгоритмів. У табл. 10.1, запозиченої з [13], показано, наскільки зростуть розміри задач, розв'язуваних за одну годину машинного часу, якщо завдяки вдосконаленню технології швидкодія комп'ютерів збільшиться в 100 або 1000 разів. Зазначимо, що для функції $f(n) = 2^n$ збільшення швидкості обчислень у 1000 разів зумовлює те, що розмір найбільшої задачі, розв'язної за одну годину, зросте лише на 10, а для функції $f(n) = n^5$ — майже в чотири рази.

Таблиця 10.1

Функція часової складності	Розміри найбільшої задачі, яку можна розв'язати за одну годину		
	на сучасних комп'ютерах	на комп'ютерах, швидкість яких більша в 100 разів	на комп'ютерах, швидкість яких більша в 1000 разів
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.95N_4$
2^n	N_5	$N_5 + 6.64$	$N_5 + 9.97$
3^n	N_6	$N_6 + 4.19$	$N_6 + 6.29$

Нагадаємо, що задачу називають *важкорозв'язною*, якщо немає поліноміального алгоритму її розв'язання. Слід зазначити, що є експоненціальні алгоритми, які добре зарекомендували себе на практиці. Річ у тім, що часову складність означено як міру поведінки алгоритму *в найгіршому випадку*. Насправді може виявитись, що для розв'язання більшості конкретних задач потрібно значно менше часу, і це дійсно так для декількох добре відомих алгоритмів. Наприклад, симплекс-метод для розв'язування задач лінійного програмування має експоненціальну часову складність, але дуже добре працює на практиці. Для задачі лінійного програмування існує й поліноміальний алгоритм, але його майже не використовують. Інший приклад: алгоритм гілок і меж настільки успішно розв'язує задачу про рюкзак, що багато дослідників уважають її „добре розв'язною", хоча такі алгоритми мають експоненціальну часову складність. На жаль, подібні приклади рідкісні. Хоча експоненціальні алгоритми відомі для багатьох задач, небагато з них прийнятні для практичних цілей.

Означення важкорозв'язної задачі, по суті, незалежне від конкретної схеми кодування й моделі обчислювального пристрою, використовуваних для означення часової складності. Поширені схеми кодування, застосовувані на практиці для конкретних задач, відрізняються одна від одної не більше ніж поліноміально. Наведемо дві важливі вимоги до схем кодування.

1. Код будь-якої індивідуальної задачі I має бути „стисненим", тобто не містити надлишкової інформації чи символів.
2. Числа в умові задачі I мають бути у двійковій системі числення (або десятковій, або мати будь-яку іншу основу, але не 1).

Розглядаючи тільки такі схеми кодування, що задовольняють ці умови, можна незалежно від вибору конкретної схеми кодування виявити, чи важкорозв'язна дана (масова) задача.

Таблиця 10.2

Машина 2, яка моделюється	Машина 1, яка моделює		
	1MT	kMT	МДД
Машина Тьюрінга з однією стрічкою (1MT)	–	$O(T(n))$	$O(T(n)\log T(n))$
Машини Тьюрінга з k стрічками (kMT)	$O(T^2(n))$	–	$O(T(n)\log T(n))$
Машина з довільним доступом до пам'яті (МДД)	$O(T^2(n))$	$O(T^2(n))$	–

Аналогічні зауваження можна зробити й щодо вибору моделі обчислювального пристрою [13]. Усі відомі в наш час моделі комп'ютерів (наприклад, однострічкові та багатострічкові машини Тьюрінга, машини з довільним доступом до пам'яті) еквівалентні щодо поліноміальної часової складності. У табл. 10.2 наведено дані щодо часу, потрібного машині 1 для моделювання роботи алгоритму, складність якого на машині 2 — $T(n)$. Є підстави очікувати, що й будь-яка інша модель комп'ютера еквівалентна за складністю всім названим моделям.

10.2. Задачі розпізнавання, мови та кодування

Задля зручності теорію *складності* [13] будують тільки для *задач розпізнавання властивостей*, які мають лише два розв'язки: „так" або „ні". На абстрактному рівні це означає, що задача розпізнавання $\mathcal{P}$ складається просто з двох множин: множини $D_{\mathcal{P}}$ всіх можливих індивідуальних задач і множини $Y_{\mathcal{P}}$ ($Y_{\mathcal{P}} \subset D_{\mathcal{P}}$) індивідуальних задач із відповіддю „так". Стандартна форма опису таких задач складається з двох частин. У першій частині дають опис *умови* задачі в термінах різних компонент — множин, графів, функцій тощо. У другій частині в термінах умови формулюють *запитання*, на яке може бути лише дві відповіді — „так" або „ні". Цей опис очевидним способом задає множини $D_{\mathcal{P}}$ й $Y_{\mathcal{P}}$. Індивідуальна задача належить множині $D_{\mathcal{P}}$ в тому й лише в тому разі, коли її можна отримати зі стандартної форми опису підстановкою конкретних значень в умову. Індивідуальна задача належить множині $Y_{\mathcal{P}}$ тоді й лише тоді, коли відповідь на запитання задачі — „так".

Розглянемо добре відому задачу розпізнавання з теорії графів.

Приклад 10.2. Ізоморфний підграф.

Умова. Дано два графи $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$.

Запитання. Чи містить граф G_1 підграф, ізоморфний графу G_2?

Задачу розпізнавання, яка відповідає задачі комівояжера, можна сформулювати так.

Приклад 10.3. Комівояжер.

Умова. Дано скінченну множину $C = \{c_1, c_2, ..., c_m\}$ міст, віддалі $d(c_i, c_j) \in N$ для кожної пари міст $c_i, c_j \in C$ та межу $B \in N$ (тут N — множина натуральних чисел).

Запитання. Чи існує шлях довжиною не більше B, який починається в місті $c_{s(1)}$, проходить через кожне місто точно один раз і повертається в місто $c_{s(1)}$? Інакше кажучи, чи існує така перестановка $(c_{s(1)}, c_{s(2)}, ..., c_{s(m)})$ елементів множини C, що

$$\sum_{i=1}^{m-1} d\left(c_{s(i)}, c_{s(i+1)}\right) + d\left(c_{s(m)}, c_{s(1)}\right) \leqslant B?$$

Останній приклад також ілюструє важливий метод — побудову задачі розпізнавання з відповідної оптимізаційної задачі.

Задачі розпізнавання мають дуже природний формальний еквівалент, зручний для вивчення методами теорії обчислень. Цей еквівалент — мова. Нагадаємо,

що мова над алфавітом V — це будь-яка підмножина $L \subset V^*$, де V^* — множина всіх скінченних ланцюжків, утворених із символів алфавіту V; ці ланцюжки далі називатимемо *словами*.

Відповідність між задачами розпізнавання та мовами задають за допомогою схем кодування. Схема кодування e задачі $\mathcal{T}$ описує кожну індивідуальну задачу з $\mathcal{T}$ певним словом у якомусь фіксованому алфавіті V. Отже, задача $\mathcal{T}$ та її схема кодування e розбивають слова з множини V^* на три класи: слова, що не є кодами індивідуальних задач із $\mathcal{T}$; слова, що являють собою коди індивідуальних задач із $\mathcal{T}$ з негативною відповіддю на запитання задачі; слова — коди індивідуальних задач із $\mathcal{T}$ з позитивною відповіддю на запитання. Третій клас слів — це, власне, та *мова*, яку ставлять у відповідність задачі $\mathcal{T}$ в разі схеми кодування e й позначають як $L[\mathcal{T}, e]$:

$$L[\mathcal{T}, e] = \{\xi \in V^* \mid V \text{ — алфавіт схеми кодування } e,$$
$$\xi \text{ — код індивідуальної задачі } \mathbf{I} \in Y_{\mathcal{T}} \text{ в разі схеми кодування } e\}.$$

Формально говорять, що якийсь результат справджується для задачі $\mathcal{T}$ в разі схеми кодування e, якщо він правдивий для мови $L[\mathcal{T}, e]$. Якщо обмежитися лише „розумними схемами кодування", то будь-яке нове поняття чи властивість, сформульовані в термінах мови, фактично не залежать від схеми кодування. Це дає змогу неформально говорити про те, що певна властивість виконується чи не виконується для задачі $\mathcal{T}$.

Якщо враховувати кодування, то втрачається зв'язок із формальним поняттям „довжина входу". Тому потрібен параметр, за допомогою якого можна виразити часову складність. Зручно вважати, що з кожною задачею розпізнавання пов'язана незалежна від схеми кодування функція Length: $D_{\mathcal{T}} \to N$, яка „поліноміально еквівалентна" довжині коду індивідуальної задачі, одержаної за будь-якої „розумної" схеми кодування. *Поліноміальну еквівалентність* розуміють так: для будь-якої „розумної" схеми кодування задачі $\mathcal{T}$ існують два поліноми p_1 і p_2 такі, що коли $\mathbf{I} \in D_{\mathcal{T}}$ та слово ξ — код індивідуальної задачі $\mathbf{I}$ в разі кодування e, то $\text{Length}(\mathbf{I}) \leqslant p_1(|\xi|)$ і $|\xi| \leqslant p_2(\text{Length}(\mathbf{I}))$, де $|\xi|$ — довжина слова ξ. Наприклад, у задачі про ізоморфний підграф (див. приклад 10.2) можна записати: $\text{Length}(\mathbf{I}) = |V_1| + |V_2|$, де $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$ — графи, які утворюють розглядувану індивідуальну задачу. У задачі про комівояжера (див. приклад 10.3) можна записати: $\text{Length}(\mathbf{I}) = m + \lceil \log_2 B \rceil + \max\{\lceil \log_2 d(c_i, c_j) \rceil,\ c_i, c_j \in C\}$.

Оскільки будь-які дві „розумні" схеми кодування задачі $\mathcal{T}$ дають поліноміально еквівалентні довжини кодів, то діапазон для вибору функції Length дуже великий, а отримувані результати справджуються для будь-якої функції Length, яка задовольняє сформульовані вище умови. Плідність такого неформального, незалежного від кодування підходу пов'язана з тим, що саме розуміють під „розумною" схемою кодування. Схему кодування вважають розумною, якщо вона достатньо стиснена та її можна декодувати [13]. Зміст поняття „стиснена" пов'язаний з умовами, сформульованими в підрозділі 10.1. Ця властивість потрібна для того, щоб у разі кодування індивідуальної задачі штучно не збільшувалося вхідне слово. Таке збільшення може призвести, наприклад, до того, що експоненціальний алгоритм неприродним способом перетвориться на поліноміальний. Можливість

декодування полягає в тому, що за даною компонентою умови задачі можна навести алгоритм із поліноміальним часом роботи, який би із заданого коду індивідуальної задачі міг відновити опис цієї компоненти.

10.3. Детерміновані машини Тьюрінга та клас *P*

Для формалізації поняття алгоритму потрібно зафіксувати певну модель процесу обчислень. Із цією метою використаємо машину Тьюрінга (точніше, детерміновану однострічкову машину Тьюрінга – скорочено ДМТ), уведену в підрозділі 9.2. Для цілей цього розділу дещо модифікуємо розглянуте раніше означення машини Тьюрінга.

1. В алфавіті зовнішніх символів A явно виділимо підмножину *вхідних символів* V, за допомогою яких записують слово в початковій конфігурації. Отже, $V \subset A$, причому порожній символ $\Lambda \in A \setminus V$.
2. В алфавіті внутрішніх станів Q замість одного заключного стану q_0 виділимо два – q_Y і q_N (від слів Yes – так, No – ні), тобто $Q = \{q_Y, q_N, q_1, q_2, ..., q_k\}$. Стан q_1, як і раніше, уважатимемо початковим. Заключні стани q_Y і q_N не можуть бути в лівій частині жодної команди.
3. Комірки нескінченної стрічки будемо нумерувати цілими числами ..., –2, –1, 0, 1, 2, ...

Вхід для детермінованої машини (програми) – слово $\xi \in V^*$, яке записують на стрічці в комірках із номерами 1, 2, ..., $|\xi|$ по одному вхідному символу в кожній комірці. Як V^* тут позначено множину всіх слів у алфавіті V. Програма розпочинає роботу, перебуваючи в стані q_1; при цьому головка перебуває над коміркою з номером 1. Далі процес обчислень виконується послідовно, крок за кроком; якщо поточний стан $q = q_Y$ або q_N, то процес обчислень закінчується з результатом „так" у разі $q = q_Y$ і „ні" у разі $q = q_N$.

Приклад 10.4. Приклад простої ДМТ-програми M наведено в табл. 10.3. Тут $A = \{0, 1, \Lambda\}$, $V = \{0, 1\}$, $Q = \{q_Y, q_N, q_1, q_2, q_3, q_4\}$.

Таблиця 10.3

Стан	0	1	Λ
q_1	$0\Pi q_1$	$1\Pi q_1$	$\Lambda Л q_2$
q_2	$\Lambda Л q_3$	$\Lambda Л q_4$	$\Lambda Л q_N$
q_3	$\Lambda Л q_Y$	$\Lambda Л q_N$	$\Lambda Л q_N$
q_4	$\Lambda Л q_N$	$\Lambda Л q_N$	$\Lambda Л q_N$

Легко переконатись, що обчислення за програмою M для входу ξ = 10100 після восьми кроків завершується в стані q_Y, тому відповідь – „так". Відповідні конфігурації зображено на рис. 10.2.

Говорять, що програма M із вхідним алфавітом V *приймає* (допускає) слово $\xi \in V^*$ тоді й лише тоді, коли в разі її застосування до вхідного слова ξ вона зупиняється в стані q_Y. Мову L_M, *розпізнавану* програмою M, задають так:

$$L_M = \{\xi \in V^* \mid M \text{ приймає } \xi\}.$$

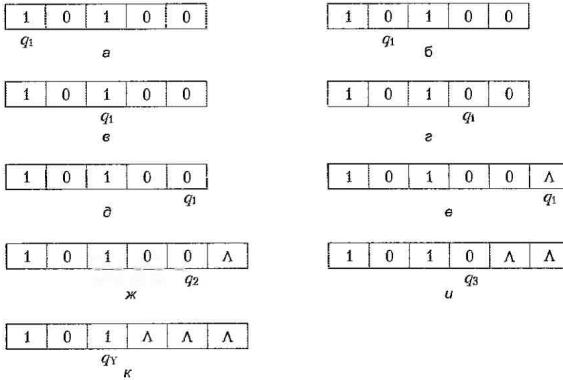


Рис. 10.2

Приклад 10.5. Програма з прикладу 10.4 розпізнає мову

$$\{\xi \in \{0, 1\}^* \mid \text{два останні символи слова } \xi \text{ — нулі}\}.$$

Згідно з означенням розпізнавання мови програма M не має зупинятися на всіх вхідних словах із множини V^*; вона має зупинятися тільки на входах із мови L_M. Якщо слово ξ належить множині $V^* \setminus L_M$, то робота програми M на ньому може закінчитись у стані q_N або тривати без зупинки. Проте ДМТ-програма, яка відповідає нашому розумінню алгоритму, має зупинятися на всіх словах вхідного алфавіту V. Тому можна вважати, що програма з прикладу 10.4 алгоритмічна, бо, почавши роботу на будь-якому слові із символів 0, 1, вона зупиниться.

Відповідність між розпізнаванням мов і розв'язуванням задач розпізнавання задають так. Говорять, що ДМТ-програма M *розв'язує* задачу розпізнавання $\mathcal{T}$ в разі кодування e, якщо вона зупиняється на всіх словах із символів вхідного алфавіту й $L_M = L[\mathcal{T}, e]$. Програма з прикладу 10.4 ілюструє цю відповідність. Розглянемо таку задачу розпізнавання з теорії чисел.

Приклад 10.6. Подільність на 4.

Умова. Дано натуральне число n.

Запитання. Чи існує таке натуральне число m, що $n = 4m$?

У разі стандартного кодування число n можна подати словом із 0 і 1, тобто двійковим записом цього числа. Оскільки натуральне число ділиться на 4 тоді й лише тоді, коли останні дві цифри його двійкового запису — нулі, то програма з прикладу 10.4 розв'язує цю задачу за такого стандартного кодування.

Тепер можна означити поняття часової складності. Час, потрібний ДМТ-програмі M, щоб виконати обчислення для вхідного слова $\xi \in V^*$, – це кількість

кроків до моменту зупинки. Якщо програма M зупиняється для всіх вхідних слів $\xi \in V^*$, то її *часову складність* $T_M : N \to N$ означають так:

$$T_M(n) = \max\{m \mid \text{існує таке слово } \xi \in V^*,\ |\xi| = n, \text{ що для обчислення за програмою } M \text{ на вході } \xi \text{ потрібно } m \text{ кроків}\}.$$

ДМТ-програму називають *поліноміальною*, якщо існує такий поліном p, що для всіх $n \in N$ виконується нерівність $T_M(n) \leqslant p(n)$.

Нарешті, можна формально означити перший важливий клас мов — P:

$$P = \{L \mid \text{існує поліноміальна ДМТ-програма } M, \text{ для якої } L = L_M\}.$$

Говорять, що задача розпізнавання $\mathcal{T}$ належить до класу P в разі схеми кодування e, якщо $L[\mathcal{T}, e] \in P$, тобто існує поліноміальна ДМТ-програма, котра розв'язує задачу $\mathcal{T}$ в разі кодування за схемою e. З урахуванням наведених вище міркувань про еквівалентність „розумних" кодувань ми не будемо наводити конкретні схеми кодування та просто говоритимемо, що (масова) задача розпізнавання $\mathcal{T}$ належить класу P.

Термін „поліноміальний алгоритм" часто будемо використовувати неформально. Формальний еквівалент цього поняття — поліноміальна ДМТ-програма. Однак із врахуванням зазначеної вище еквівалентності реалістичних моделей комп'ютерів щодо поняття „поліноміальний час роботи" формальне означення класу P можна сформулювати в термінах програм для довільної такої моделі, причому в результаті отримаємо один і той самий клас мов. Отже, для доведення того, що певні задачі можна розв'язати за допомогою поліноміального алгоритму, немає потреби заглиблюватись у деталі ДМТ. Насправді ми досліджуємо алгоритми в машинно-незалежному стилі, ніби вони працюють безпосередньо з компонентами індивідуальної задачі (множинами, графами, числами тощо), а не з їх кодами. Тут мається на увазі, що за бажання й наполегливості для будь-якого поліноміального алгоритму можна побудувати відповідну поліноміальну ДМТ-програму.

10.4. Недетерміновані машини Тьюрінга та клас *NP*

У цьому підрозділі введено інший важливий клас мов (тобто задач розпізнавання властивостей) — NP. Перш ніж перейти до формального означення в термінах мов і машин Тьюрінга, пояснимо зміст поняття, на якому ґрунтується означення класу NP.

Розглянемо задачу комівояжера (див. приклад 10.3): за множиною міст, віддалей між ними та межею B виявити, чи існує гамільтонів цикл, довжина якого не перевищує B. Поліноміальний алгоритм для цієї задачі невідомий. Припустимо, що для якоїсь індивідуальної задачі I одержано відповідь „так". Це можна довести, навівши відповідний гамільтонів цикл, і тим самим перевірити відповідне твердження. Більше того, цю процедуру перевірки можна подати у вигляді алгоритму, часова складність якого обмежена поліномом від Length(I).

Подібні властивості має також задача про ізоморфний підграф (див. приклад 10.2). Нехай в індивідуальній задачі I є два графи $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$. Якщо

відповідь на запитання задачі — „так", то це можна довести, указавши підмножини $V' \subset V_1$ і $E' \subset E_1$ та бієкцію $\varphi: V_2 \to V'$ із потрібними властивостями. У цьому разі правильність твердження знову можна легко засвідчити за поліноміальний від Length(I) час, просто перевіривши, що V', E' і φ задовольняють вимоги, сформульовані в задачі. Поняття поліноміальної перевірки дає змогу виділити задачі класу *NP*. Зауважимо, що з можливості перевірки за поліноміальний час не випливає розв'язність за поліноміальний час.

Неформально клас *NP* можна означити за допомогою поняття, яке називають *недетермінованим алгоритмом*. Такий алгоритм складається з двох різних стадій — *угадування* та *перевірки*. Для заданої індивідуальної задачі I на першій стадії просто „вгадують" якусь структуру Σ. Після цього I та Σ разом подають як вхід на стадію перевірки, котра виконується детерміновано та закінчується відповіддю „так" або „ні" чи продовжується нескінченно без зупинки (відповідь „ні" й останню можливість розрізняти необов'язково). Недетермінований алгоритм „розв'язує" задачу розпізнавання π, якщо для довільної індивідуальної задачі $I \in D_\pi$ виконано такі умови.

1. Якщо $I \in Y_\pi$, то існує така структура Σ, угадування якої для входу I зумовлює те, що стадія перевірки, розпочавши роботу з входом (I, Σ), завершується відповіддю „так".
2. Якщо $I \notin Y_\pi$, то не існує такої структури Σ, угадування котрої для входу I забезпечує закінчення стадії перевірки з входом (I, Σ) відповіддю „так".

Наприклад, недетермінований алгоритм розв'язування задачі комівояжера (див. приклад 10.3) можна побудувати так. На стадії вгадування потрібно просто вибрати довільну перестановку елементів множини міст C, а на стадії перевірки — використати поліноміальну процедуру „перевірка доведення" для задачі комівояжера.

Говорять, що недетермінований алгоритм, який розв'язує задачу розпізнавання π, працює вподовж поліноміального часу, якщо є такий поліном p, що для довільної індивідуальної задачі $I \in D_\pi$ знайдеться здогад Σ, який на стадії детермінованої перевірки на вході (I, Σ) зумовить відповідь „так" за час p(Length(I)).

Неформально NP — це клас усіх задач розпізнавання π, котрі в разі „розумного кодування" можна розв'язати недетермінованими (N — nondeterministic) алгоритмами за поліноміальний (P — polynomial) час. Зокрема, задачі комівояжера та про ізоморфний підграф (див. приклади 10.2, 10.3) належать до класу *NP*.

У таких неформальних означеннях поняттям розв'язування слід користуватись обережно. Головне призначення так званого поліноміального недетермінованого алгоритму полягає в поясненні поняття можливості перевірки за поліноміальний час, а не в тому, щоб бути реальним методом розв'язування задач розпізнавання властивостей. Для кожного входу такий алгоритм містить не одне, а декілька можливих обчислень — по одному для кожного можливого здогаду.

Формалізуємо означення класу *NP* в термінах мов і машин Тьюрінга. Формальний еквівалент недетермінованого алгоритму — програма для недетермінованої однострічкової машини Тьюрінга (НДМТ). Вона має декілька варіантів поведінки

після зчитування букви. а одне вхідне слово зумовлює декілька обчислень. Це можна уявити собі так, що машина робить здогади чи використовує довільну кількість паралельних процесорів.

Є різні версії НДМТ. Та, якою ми будемо користуватись [13], має таку саму структуру, як ДМТ; відмінність полягає лише в тому, що НДМТ доповнено *модулем угадування* зі своєю *головкою*, котра може *тільки записувати* на стрічку. Модуль угадування дає інформацію для записування здогаду, і його застосовують лише для цього.

Програму для НДМТ, або НДМТ-програму, означають абсолютно аналогічно до ДМТ-програми. Обчислення НДМТ-програми для входу $\xi \in V^*$, на відміну від обчислення ДМТ-програми, має дві різні стадії.

На першій стадії відбувається вгадування. У початковий момент вхідне слово ξ записано в комірках із номерами 1, 2, ..., $|\xi|$; головку для читання-запису розміщено над коміркою з номером 1, головку модуля вгадування — над коміркою з номером –1, а пристрій керування пасивний. Модуль угадування починає керувати своєю головкою, котра в кожен момент робить один крок. Головка чи записує в комірку під нею одну з букв алфавіту A та зсувається на одну комірку вліво, чи зупиняється (у цьому разі модуль угадування переходить у пасивний стан, а пристрій керування починає роботу в стані q_1). Модуль угадування вирішує, чи продовжити роботу (або перейти в пасивний стан), і яку букву з алфавіту A записати на стрічці, причому робить це цілком довільно. Отже, модуль угадування до моменту закінчення своєї роботи може записати будь-яке слово з множини A^*, а може ніколи не зупинитись.

Друга стадія (перевірки) починається тоді, коли пристрій керування переходить у стан q_1. Починаючи з цього моменту обчислення НДМТ-програми проходять точно за тими самими правилами, що й для ДМТ-програми. Модуль угадування та його головка в обчисленнях більше не зайняті; вони виконали своє завдання — записали на стрічці слово-здогад. Головка читання-запису проглядає це слово в процесі перевірки. Обчислення завершується тоді, коли пристрій керування переходить в один із двох заключних станів (q_Y або q_N). Обчислення називають *приймаючим*, якщо воно зупиняється в стані q_Y. Усі інші обчислення, незалежно від того, закінчуються вони чи ні, називають *неприймаючими*.

Будь-яка НДМТ-програма M може мати нескінченну кількість можливих обчислень для даного входу ξ — по одному для кожного слова-здогаду. Говоритимемо, що НДМТ-програма M *приймає* (*допускає*) вхід ξ, якщо принаймні одне з її обчислень на ньому приймаюче.

Мова, яку розпізнає програма M. — це

$$L_M = \{\xi \in V^* \mid M \text{ приймає } \xi\}.$$

Час прийняття недетермінованою програмою M *слова* $\xi \in L_M$ — це *мінімальна* кількість кроків, виконуваних на стадіях угадування й перевірки до моменту досягнення заключного стану q_Y, де мінімум береться за всіма приймаючими

обчисленнями програми M для входу ξ. *Часова складність НДМТ-програми* M — це функція $T_M : N \to N$, означена так:

$$T_M(n) = \max\{1, m \mid \text{існує слово } \xi \in L_M, |\xi| = n, \text{ таке, що час прийняття } \xi \text{ програмою } M \text{ дорівнює } m\}.$$

Розглядають пару $\{1, m\}$, бо для деяких n, можливо, узагалі немає таких входів довжиною n, які приводять до заключного стану q_Y.

НДМТ-програму називають *НДМТ-програмою з поліноміальним часом роботи*, якщо знайдеться поліном p такий, що $T_M(n) \leqslant p(n)$ для всіх $n \geqslant 1$.

Нарешті, клас *NP формально* означають так:

$$NP = \{L \mid \text{існує така НДМТ-програма } M \text{ із поліноміальним часом роботи, що } L = L_M\}.$$

Задача розпізнавання π належить до класу NP в разі схеми кодування e, якщо $L[\pi, e] \in NP$. Як і для класу P, ми будемо просто говорити, що задача π належить до класу NP, не зазначаючи конкретної схеми кодування. Більше того, оскільки будь-яку реалістичну модель обчислювального пристрою можна доповнити алгоритмом розглянутого модуля вгадування з головкою запису, то наші формальні означення можна переробити для будь-якої іншої стандартної моделі обчислювального пристрою. Позаяк усі такі моделі еквівалентні з точністю до детермінованих поліноміальних обчислень, то всі отримані версії класу NP ідентичні. Отже, можна ототожнити формально означений клас NP з класом усіх задач розпізнавання, котрі „розв'язні" недетермінованими алгоритмами з поліноміальним часом роботи.

10.5. Поліноміальна звідність і *NP*-повні задачі

Очевидно, що $P \subset NP$: будь-яка задача розпізнавання, розв'язна за поліноміальний час детермінованим алгоритмом, розв'язна за поліноміальний час і недетермінованим алгоритмом (стадію вгадування в такому разі просто ігнорують). Головна проблема теорії складності дискретних алгоритмів полягає в доведенні чи спростуванні гіпотези про те, що $P \neq NP$. Якщо гіпотеза справджується, то для розв'язування найважчих задач класу NP поліноміальних алгоритмів не існує.

Нині широко розповсюджена думка, що $P \neq NP$, хоча доведення цієї гіпотези немає. Однак на сучасному рівні знань, мабуть, розумніше працювати за домовленості, що $P \neq NP$. На основі нагромадженого досвіду ми будемо уявляти клас NP так, як це зображено на рис. 10.3, очікуючи (хоча й не з повною впевненістю), що заштрихована область, яка позначає $NP \setminus P$, не порожня.

Якщо $P \neq NP$, то відмінність між класами P та $NP \setminus P$ дуже істотна. Усі задачі з P можна розв'язати поліноміальними алгоритмами, а всі задачі з $NP \setminus P$ важкорозв'язні. Зрозуміло, що доки не доведено, що $P \neq NP$, неможливо довести, що якась конкретна задача належить класу $NP \setminus P$. Тому мета теорії NP-повних задач полягає в доведенні слабших результатів типу „якщо $P \neq NP$, то $\pi \in NP \setminus P$".

Такі умовні результати можна вважати підтвердженням важкорозв'язності з тим самим рівнем упевненості, з яким ми вважаємо, що $P \neq NP$. Головна ідея цього умовного підходу ґрунтується на понятті поліноміальної звідності.

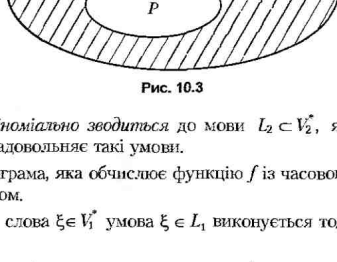


Рис. 10.3

Мова $L_1 \subset V_1^*$ *поліноміально зводиться* до мови $L_2 \subset V_2^*$, якщо існує функція $f: V_1^* \to V_2^*$, яка задовольняє такі умови.

1. Існує ДМТ-програма, яка обчислює функцію f із часовою складністю, обмеженою поліномом.
2. Для будь-якого слова $\xi \in V_1^*$ умова $\xi \in L_1$ виконується тоді й тільки тоді, коли $f(\xi) \in L_2$.

Якщо мова L_1 поліноміально зводиться до мови L_2, то записують $L_1 \propto L_2$ і говорять, що мова L_1 зводиться до мови L_2 (слово „поліноміально" випускають).

Важливість поняття поліноміальної звідності випливає з такої леми.

Лема 10.1. Якщо $L_1 \propto L_2$, то з $L_2 \in P$ випливає, що $L_1 \in P$. (Еквівалентне твердження: з $L_1 \notin P$ випливає, що $L_2 \notin P$.)

Доведення. Нехай V_1 і V_2 — алфавіти мов L_1 і L_2, функція $f: V_1^* \to V_2^*$ реалізує поліноміальну звідність L_1 до L_2; M_f — поліноміальна ДМТ-програма, яка обчислює функцію f, M_2 — поліноміальна ДМТ-програма, яка розпізнає мову L_2. Поліноміальну ДМТ-програму, яка розпізнає мову L_1, можна одержати композицією програм M_f і M_2. До входу $\xi \in V_1^*$ спочатку застосовують програму M_f, щоб побудувати $f(\xi) \in V_2^*$. Після цього до $f(\xi)$ застосовують програму M_2, яка з'ясовує, чи справджується, що $f(\xi) \in L_2$. Оскільки $\xi \in L_1$ тоді й тільки тоді, коли $f(\xi) \in L_2$, то цей опис дає ДМТ-програму, яка розпізнає мову L_1. Те, що час роботи цієї програми обмежений поліномом, одразу випливає з поліноміальності програм M_f і M_2.

Якщо $\mathcal{П}1$ і $\mathcal{П}2$ — задачі розпізнавання, а $e1$ і $e2$ — їхні схеми кодування, то писатимемо $\mathcal{П}1 \propto \mathcal{П}2$, якщо існує поліноміальна звідність мови $L[\mathcal{П}1, e1]$ до мови $L[\mathcal{П}2, e2]$. У подальшому згадки про конкретні схеми кодування будемо випускати. Отже, на рівні задач поліноміальна звідність задачі розпізнавання $\mathcal{П}1$ до задачі розпізнавання $\mathcal{П}2$ означає наявність функції $f: D_{\mathcal{П}1} \to D_{\mathcal{П}2}$, яка задовольняє дві умови.

1. Функція f обчислюється поліноміальним алгоритмом.
2. Для всіх $\mathbf{I} \in D_{\mathcal{П}1}$ умова $\mathbf{I} \in Y_{\mathcal{П}1}$ виконується тоді й лише тоді, коли $f(\mathbf{I}) \in Y_{\mathcal{П}2}$.

Щоб конкретно уявити зміст цього поняття, розглянемо такий приклад.

Приклад 10.7. Гамільтонів цикл.

Умова. Дано граф $G = (V, E)$.

Запитання. Чи правильно, що граф G містить гамільтонів цикл?

Покажемо, що ця задача зводиться до задачі комівояжера (див. приклад 10.3). Для цього потрібно навести функцію f, яка відображає кожну індивідуальну задачу про гамільтонів цикл у відповідну індивідуальну задачу комівояжера та задовольняє дві умови до поліноміальної звідності.

Функцію f означають дуже просто. Нехай $G = (V, E)$, $|V| = m$ — запис умови фіксованої індивідуальної задачі про гамільтонів цикл. Відповідну задачу комівояжера будують так: множина міст C збігається з V; для будь-яких двох міст $v_i, v_j \in C$ віддаль $d(v_i, v_j)$ між ними беремо рівною 1, якщо $\{v_i, v_j\} \in E$, і 2, якщо $\{v_i, v_j\} \notin E$. Межа $B = m$. Як вправу пропонуємо довести (неформально), що ця функція f реалізує звідність і її можна обчислити за поліноміальний час.

Значення леми 10.1 для задач розпізнавання тепер можна проілюструвати на прикладі поліноміальної звідності задачі про гамільтонів цикл до задачі комівояжера. Якщо задачу комівояжера можна розв'язати поліноміальним алгоритмом, то це справджується й для задачі про гамільтонів цикл, а якщо задача про гамільтонів цикл важкорозв'язна, то й задача комівояжера також важкорозв'язна. Отже, лема 10.1 дає змогу інтерпретувати звідність $\mathcal{П}1 \propto \mathcal{П}2$ як твердження, що задача $\mathcal{П}2$ не простіша, ніж задача $\mathcal{П}1$.

Завдяки поняттю звідності можна надати точний зміст твердженню про те, що одна мова не простіша, ніж інша. Найскладніші в цьому розумінні *NP*-повні задачі.

Мову L називають *NP-повною*, якщо виконуються дві властивості.

1. $L \in NP$.
2. $L' \propto L$ для будь-якого $L' \in NP$.

Клас *NP*-повних мов позначають *NPC* (Complete — повний).

Говорячи неформально, задачу розпізнавання $\mathcal{П}$ називають *NP*-повною, якщо $\mathcal{П} \in NP$ та будь-яка інша задача розпізнавання $\mathcal{П}' \in NP$ зводиться до $\mathcal{П}$.

ТЕОРЕМА 10.1. Якщо якась *NP*-повна задача розв'язна за поліноміальний час, то $P = NP$. Інакше кажучи, якщо в класі *NP* існує задача, не розв'язна за поліноміальний час, то й усі *NP*-повні задачі такі.

Доведення проведемо на рівні мов. Нехай L — *NP*-повна мова, яка водночас виявляється розв'язною за поліноміальний час ($L \in NPC$ й $L \in P$). Тоді для будь-якої мови $L' \in NP$ за властивістю 2 означення *NP*-повної мови $L' \propto L$. Отже, за лемою 10.1 $L' \in P$, і перше твердження доведено. Друге твердження теореми — це просто переформулювання першого.

Отже, гіпотеза $P \neq NP$ означає, що *NP*-повні задачі не можна розв'язати за поліноміальний час, тобто вони важкорозв'язні. Більшість експертів уважають, що це дійсно так; передбачуване співвідношення між класами *P*, *NP* та *NPC* наведено на рис. 10.4.

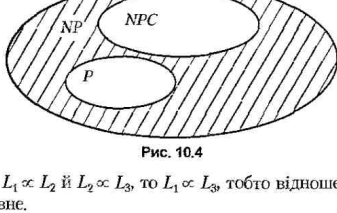


Рис. 10.4

ЛЕМА 10.2. Якщо $L_1 \propto L_2$ й $L_2 \propto L_3$, то $L_1 \propto L_3$, тобто відношення поліноміальної звідності транзитивне.

Для доведення *NP*-повноти потрібно довести, що будь-яка задача з класу *NP* зводиться до якогось „кандидата" на *NP*-повну задачу. Зовсім не зрозуміло, як це можна зробити. Апріорі не очевидно навіть, чи існує хоча б одна *NP*-повна задача.

Наступна теорема, яка безпосередньо випливає з означень і транзитивності відношення $\propto$, показує, що поставлене запитання дуже спрощується, якщо відома принаймні одна *NP*-повна задача.

ТЕОРЕМА 10.2. Якщо мови L_1 і L_2 належать до класу *NP*, мова L_1 *NP*-повна й $L_1 \propto L_2$, то мова L_2 також *NP*-повна.

На рівні задач розпізнавання ця теорема надає простий спосіб доведення *NP*-повноти нової задачі Π, якщо відома хоча б одна *NP*-повна задача (див. підрозділ 10.7).

10.6. Теорема Кука

Першою *NP*-повною задачею стала задача розпізнавання з логіки; її називають *задачею про виконанність* і формулюють так.

Приклад 10.8. Виконанність.

Умова. Дано множину булевих змінних X і формулу F від цих змінних у КНФ.

Запитання. Чи виконана формула F, тобто чи існує інтерпретація, у якій формула F набуває значення Т?

ТЕОРЕМА 10.3 (Кука). Задача про виконанність *NP*-повна.

Доведення. Зазначимо, що КНФ формули F — це кон'юнкція якогось набору диз'юнкцій над множиною змінних X. Позначимо цей набір диз'юнкцій як S. Очевидно, що задача про виконанність міститься в класі *NP*. Недетермінованому алгоритму для її розв'язування достатньо навести інтерпретацію (набір значень істинності на множині змінних) і перевірити, що в цій інтерпретації всі диз'юнкції з набору S виконуються, тобто набувають значення Т. Усе це легко зробити (недетерміновано) за поліноміальний час. Отже, перша вимога, яку мають задовольняти *NP*-повні задачі, справджується.

Для перевірки другої вимоги повернімося до рівня мов, тобто до подання задачі про виконанність мовою $L_{\text{вик}} = L[\text{вик}, e]$ для якоїсь „розумної" схеми кодуван-

ня *е*. Потрібно довести. що для всіх мов $L \in NP$ виконується співвідношення $L \propto L_{\text{вик}}$. Різні мови з класу *NP* можуть дуже різнитись, кількість їх безмежна, тому неможливо зазначити окреме зведення для кожної з них. Однак кожну мову з класу *NP* можна подати в стандартному вигляді, а саме НДМТ-програмою, яка працює поліноміальний час і розпізнає цю мову. Таке подання дає змогу працювати із загальною НДМТ-програмою з поліноміальним часом роботи й отримати загальну звідність розпізнаваної мови до $L_{\text{вик}}$. Якщо спеціалізувати цю загальну звідність до конкретної НДМТ-програми *M*. яка розпізнає мову L_M, то одержимо поліноміальне зведення L_M до $L_{\text{вик}}$.

Спочатку розглянемо НДМТ-програму *M* із поліноміальним часом роботи [13], яка має компоненти *A*, *V*, *Q*, q_1, q_Y, q_N і розпізнає мову $L = L_M$. Окрім того, нехай $p(n)$ – поліном із цілими коефіцієнтами, який обмежує зверху часову складність $T_M(n)$. Можна вважати, що $p(n) \geqslant n$ для всіх $n \in N$. Функцію f_L, яка реалізує загальну звідність, опишемо в термінах *A*, *V*, *Q*, q_1, q_Y, q_N, а також *M* і *p*.

Зручно розглядати функцію f_L як відображення з множини слів у алфавіті *V* в індивідуальні задачі про виконанність, а не в слова в алфавіті *V*, які кодують задачі про виконанність. Справді, технічні деталі, пов'язані з кодуванням, можна легко відновити. Отже, функція f_L має таку властивість, що для всіх входів $\xi \in V^*$ приналежність $\xi \in L$ справджується тоді й лише тоді, коли існує інтерпретація, у якій виконуються всі диз'юнкції з набору $f_L(\xi)$. Головне в побудові функції f_L – використання якогось набору диз'юнкцій для запису твердження, що НДМТ-програма приймає слово ξ, тобто $\xi \in L$.

Якщо програма *M* приймає вхідне слово $\xi \in V^*$, то для нього існує таке приймаюче обчислення програми *M*, що кількість кроків на стадії перевірки та кількість символів у слові-здогалі обмежені величиною $p(n)$, де $n = |\xi|$. У такому обчисленні беруть участь лише комірки з номерами від $-p(n)$ до $p(n)+1$. Це випливає з того, що головка читання–запису починає роботу з комірки з номером 1 і на кожному кроці зсувається не більше ніж на одну комірку. Обчислення на стадії перевірки в кожний момент повністю залежить від вмісту комірок із зазначеними номерами, внутрішнього стану та положення головки читання–запису. Далі, оскільки в обчисленні на стадії перевірки не більше ніж $p(n)$ кроків, то потрібно врахувати не більше ніж $p(n)+1$ моментів. Це дає змогу повністю описати таке обчислення за допомогою поліноміально обмеженої кількості булевих змінних і якогось набору значень істинності на їх множині. Саме для цього призначено множину змінних *X*, використаних в описі функції f_L. Перенумеруємо елементи множин *Q* й *A* так:

$$Q\colon q_1, q_2 = q_Y, q_3 = q_N, q_4, q_5, \ldots, q_r, \text{ де } r = |Q|;$$
$$A\colon a_0 = \Lambda; a_1, a_2, a_3, \ldots, a_v, \text{ де } v = |A| - 1.$$

У подальших міркуваннях буде використано три типи змінних, кожному з яких надано певний зміст відповідно до табл. 10.4. Фразу „момент *i*" використано для скорочення точного виразу „після виконання *i*-го кроку обчислення на стадії перевірки".

Обчислення програми *M* очевидним способом утворює на множині цих змінних певну інтерпретацію (набір значень істинності), якщо домовитись, що в разі

завершення обчислення раніше моменту $p(n)$ конфігурація залишається незмінною в усі моменти після зупинки, тобто зберігаються заключний стан, положення головки та запис на стрічці. У нульовий момент запис на стрічці складається з вхідного слова ξ, яке міститься в комірках із номерами від 1 до n, і слова-здогаду ω в комірках із номерами від -1 до $-|\omega|$. Усі інші комірки, якщо такі є, порожні.

Таблиця 10.4

Змінна	Межі зміни індексів	Зміст, наданий змінній
$Q[i, k]$	$0 \leqslant i \leqslant p(n)$, $1 \leqslant k \leqslant r$	У момент i програма M перебуває в стані q_k
$H[i, j]$	$0 \leqslant i \leqslant p(n)$, $-p(n) \leqslant j \leqslant p(n) + 1$	У момент i головка читання–запису проглядає комірку з номером j
$C[i, j, k]$	$0 \leqslant i \leqslant p(n)$, $-p(n) \leqslant j \leqslant p(n) + 1$, $0 \leqslant k \leqslant v$	У момент i в j-й комірці записано символ a_k

Довільна інтерпретація не обов'язково відповідає якомусь обчисленню, особливо приймаючому. Функції f_L описують, будуючи певний набір диз'юнкцій, які містять наведені в табл. 10.4 змінні. Усі диз'юнкції цього набору мають виконуватись у певній інтерпретації тоді й лише тоді, коли цю інтерпретацію утворено певним приймаючим обчисленням на вході ξ, причому стадія перевірки цього обчислення містить не більше ніж $p(n)$ кроків, а слово-здогад має довжину, що не перевищує $p(n)$. Отже, $\xi \in L \Leftrightarrow$ на вході ξ існує приймаюче обчислення програми $M \Leftrightarrow$ на вході ξ існує приймаюче обчислення програми M, кількість кроків якого не перевищує $p(n)$, а слово-здогад має довжину, що не перевищує $p(n) \Leftrightarrow$ існує інтерпретація, у якій виконуються всі диз'юнкції набору $f_L(\xi)$. Тут символ $\Leftrightarrow$ означає „тоді й лише тоді".

Отже, для функції f_L справджується одна з двох умов з означення поліноміальної звідності. Другу умову, яка полягає в тому, що функцію f_L можна обчислити за поліноміальний час, перевіримо після завершення її опису.

Диз'юнкції індивідуальної задачі $f_L(\xi)$ можна поділити на шість груп, кожна з яких накладає обмеження певного типу на будь-яку інтерпретацію, у якій ці диз'юнкції виконуються. Ці групи наведено в табл. 10.5.

Таблиця 10.5

Група диз'юнкцій	Обмеження
G_1	У будь-який момент i програма M перебуває точно в одному стані
G_2	У будь-який момент i головка читання–запису проглядає точно одну комірку
G_3	У будь-який момент i кожна комірка містить точно один символ з алфавіту A
G_4	У момент 0 обчислення перебуває в початковій конфігурації стадії перевірки для входу ξ
G_5	Не пізніше ніж за $p(n)$ кроків програма M переходить у стан q_Y і, отже, приймає вхід ξ
G_6	Для будь-якого моменту i, $0 \leqslant i \leqslant p(n)$, конфігурацію програми M у момент $i + 1$ можна отримати з конфігурації в момент i виконанням однієї команди програми M

Якщо всі шість груп диз'юнкцій справді задовольняють зазначені в таблиці умови, то інтерпретація, у якій виконуються всі диз'юнкції з набору $f_t(\xi)$, має відповідати потрібному приймаючому обчисленню на вході ξ. Залишається лише навести способи побудови зазначених груп диз'юнкцій.

Група G_1 складається з таких диз'юнкцій:

$$Q[i, 1] \vee Q[i, 2] \vee \ldots \vee Q[i, r], \ 0 \leqslant i \leqslant p(n);$$

$$\overline{Q[i, j]} \vee \overline{Q[i, j']}, \ 0 \leqslant i \leqslant p(n), \ 1 \leqslant j < j' \leqslant r.$$

Перші $p(n) + 1$ із них виконуються тоді й лише тоді, коли в кожний момент i програма M перебуває принаймні в одному стані. Решта $0{,}5(p(n)+1)r(r-1)$ диз'юнкцій виконуються в тому й тільки тому разі, коли в жодний момент i програма M не перебуває більше ніж в одному стані. Отже, група G_1 виконує своє призначення.

Групи G_2 та G_3 будують аналогічно. Група G_2 складається з диз'юнкцій

$$H[i, -p(n)] \vee H[i, -p(n) + 1] \vee \ldots \vee H[i, p(n) + 1], \ 0 \leqslant i \leqslant p(n);$$

$$\overline{H[i, j]} \vee \overline{H[i, j]}, \ 0 \leqslant i \leqslant p(n), \ -p(n) \leqslant j < j' \leqslant p(n) + 1,$$

а група G_3 — з диз'юнкцій

$$C[i, j, 0] \vee C[i, j, 1] \vee \ldots \vee C[i, j, v], \ 0 \leqslant i \leqslant p(n), \ -p(n) \leqslant j \leqslant p(n) + 1;$$

$$\overline{C[i, j, k]} \vee \overline{C[i, j, k']}, \ 0 \leqslant i \leqslant p(n), \ -p(n) \leqslant j \leqslant p(n) + 1, \ 0 \leqslant k < k' \leqslant v.$$

Групи G_4 та G_5 дуже прості — кожна з них містить лише диз'юнкції з рангом 1. Зокрема, група G_4 складається з таких диз'юнкцій:

$$Q[0, 1], H[0, 1], C[0, 0, 0], C[0, 1, k_1], C[0, 2, k_2], \ldots, C[0. n. k_n],$$
$$C[0. n + 1. 0], C[0, n + 2, 0], \ldots C[0, p(n) + 1, 0],$$

де $\xi = a_{k_1} a_{k_2} \ldots a_{k_n}$.

Група G_5 складається з однієї диз'юнкції з рангом 1:

$$Q[p(n), 2].$$

Дещо складніше описати останню групу диз'юнкцій G_6, яка гарантує, що кожну наступну конфігурацію машини одержано з попередньої застосуванням однієї команди програми M. Ця група складається з двох підгруп диз'юнкцій.

Завдяки першій підгрупі якщо головка читання–запису в момент i не проглядає комірку з номером j, то символ у комірці з номером j від моменту i до моменту $i + 1$ не зміниться. Диз'юнкції цієї підгрупи мають такий вигляд:

$$\overline{C[i, j, l]} \vee H[i, j] \vee C[i + 1, j, l],$$

$$0 \leqslant i \leqslant p(n), \ -p(n) \leqslant j \leqslant p(n) + 1, \ 0 \leqslant l \leqslant v.$$

Зафіксуємо довільно момент i, комірку з номером j і символ a_k. Якщо в момент i головка читання–запису не проглядає комірку з номером j і в ній у момент i записано символ a_l, а в момент $i+1$ його там уже немає, то зазначена вище диз'юнкція не виконується (у протилежному випадку вона виконується). Отже, $2(p(n)+1)^2 \times (v+1)$ диз'юнкцій цієї групи виконують своє призначення.

Друга підгрупа диз'юнкцій із групи G_6 гарантує, що перебудова однієї машинної конфігурації в наступну відбувається за однією командою програми M. Нехай у момент i головка читання–запису проглядає комірку з номером j, у якій записано символ a_b, а машина перебуває в стані q_k. Відповідну конфігурацію зображено на рис. 10.5.

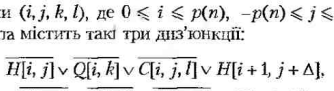


Рис. 10.5

Для кожної четвірки (i, j, k, l), де $0 \leqslant i \leqslant p(n)$, $-p(n) \leqslant j \leqslant p(n)+1$. $1 \leqslant k \leqslant r$, $0 \leqslant l \leqslant v$, ця підгрупа містить такі три диз'юнкції:

$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{C[i,j,l]} \vee H[i+1, j+\Delta],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{C[i,j,l]} \vee Q[i+1, k'],$$
$$\overline{H[i,j]} \vee \overline{Q[i,k]} \vee \overline{C[i,j,l]} \vee C[i+1, j, l'],$$

де значення k'. l', Δ задано так.

Якщо $q_k \in Q \setminus \{q_Y, q_N\}$, то

$$q_k a_l \to a_r D q_{k'}, \quad \Delta = \begin{cases} -1 \text{ для } D = \text{Л}, \\ 0 \text{ для } D = \text{Н}, \\ 1 \text{ для } D = \text{П}, \end{cases}$$

а якщо $q_k \in \{q_Y, q_N\}$, то $k' = k$, $l' = l$, $\Delta = 0$.

Ці $6(p(n))(p(n)+1)r(v+1)$ диз'юнкцій дають потрібне обмеження на набір значень змінних, на якому виконується КНФ $f_L(\xi)$.

Отже, ми показали, як побудувати групи диз'юнкцій G_1–G_6, які виконують своє призначення. Якщо $\xi \in L$, то програма M на вході ξ — приймаюче обчислення довжиною не більшою ніж $p(n)$, і це обчислення задає для зазначеного змісту змінних інтерпретацію, у якій виконуються всі диз'юнкції з $S = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_6$. І навпаки, набір диз'юнкцій S побудовано так, що будь-яка інтерпретація, у якій виконуються всі його диз'юнкції, має відповідати якомусь приймаючому обчисленню програми M на вході ξ. Звідси випливає, що інтерпретація, у якій виконується КНФ $f_L(\xi)$, існує тоді й лише тоді, коли $\xi \in L$.

Залишилося довести, що для будь-якої фіксованої мови L індивідуальну задачу $f_L(\xi)$ можна побудувати за час, обмежений поліномом від $n = |\xi|$. Якщо мову L задано, то можна вибрати якусь НДМТ-програму M, яка розпізнає мову L за час, обмежений поліномом p (немає потреби за поліноміальний час знаходити саму недетерміновану програму, бо слід лише показати, що відображення f_L існує).

Якщо є певна НДМТ-програма M і поліном p, то побудувати множину змінних X і набір диз'юнкцій S трохи складніше, ніж заповнити всі позиції в стандартній (хоча й досить складній) формулі. Поліноміальна обмеженість цього обчислення буде зрозумілою, якщо довести, що величина $\text{Length}(f_L(\xi))$ обмежена поліномом від n: тут Length(I) — довжина слова, яке кодує індивідуальну задачу I в якійсь „розумній" схемі кодування (див. підрозділ 10.2).

Як функцію довжини для задачі про виконанність можна взяти, наприклад, $|X|\cdot|S|$. Жодна з диз'юнкцій не може містити більше ніж $|X|$ літералів (нагадаємо, що літерал — це змінна чи її заперечення). Кількість символів, потрібних для опису кожного літералу, не перевищує $\log(|X|)$, але цією величиною можна знехтувати, бо вона дає поліноміальний внесок у загальну довжину задачі. Оскільки числа r і v зафіксовано заздалегідь, то вони можуть збільшити $|X|$ та $|S|$ лише в сталу кількість разів, тому $|X| = O(p(n)^2)$ та $|S| = O(p(n)^2)$. Звідси випливає, що

$$\text{Length}(f_L(\xi)) = |X|\cdot|S| = O(p(n)^4),$$

тобто ця величина обмежена поліномом від n, що й потрібно було довести.

Ми обґрунтували, що перетворення f_L можна обчислити за алгоритмом із поліноміальною часовою складністю. Отже, для всіх $L \in NP$ функцію f_L можна обчислити за поліноміальний час, і вона відображає мову L у задачу про виконанність (точніше, відображає L у $L_{\text{вип}}$). Звідси випливає твердження теореми, що задача про виконанність NP-повна.

10.7. Приклади *NP*-повних задач. Доведення *NP*-повноти

Якби всі доведення NP-повноти були так само складні, як доведення NP-повноти задачі про виконанність, то список NP-повних задач навряд чи був би таким великим, який він нині. Однак, як уже було зазначено в підрозділі 10.5, якщо відома одна NP-повна задача, то процедура доведення NP-повноти інших задач сильно спрощується. Для доведення NP-повноти задачі $\mathcal{T} \in NP$ достатньо довести, що якусь із відомих NP-повних задач $\mathcal{T}'$ можна звести до $\mathcal{T}$ за поліноміальний час. Отже, у подальшому процес доведення NP-повноти задачі розпізнавання буде складатись із таких кроків.

1. Доведення того, що $\mathcal{T} \in NP$.
2. Доведення, що якась одна відома NP-повна задача $\mathcal{T}'$ зводиться до $\mathcal{T}$ за поліноміальний час.

Задачі, розглянуті в прикладах 10.2 (про ізоморфний підграф), 10.3 (про комівояжера), 10.7 (про гамільтонів цикл) NP-повні. Наведемо інші приклади таких задач.

Приклад 10.9. 3-виконанність.

Умова. Дано булевий вираз у 3-*кон'юнктивній* нормальній формі φ. У таких формулах кожна елементарна диз'юнкція складається рівно з трьох різних літералів, наприклад

$$\varphi = (x_1 \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3).$$

Запитання. Чи виконанна форма φ, тобто чи існує інтерпретація, у якій вона набуває значення Т?

Приклад 10.10. Кліка.

Умова. Дано граф $G = (V, E)$ та натуральне число k, $k \leqslant |V|$.

Запитання. Чи містить граф кліку потужністю k, тобто таку підмножину $V' \subset V$, що $|V'| = k$ та будь-які дві вершини з V' з'єднані ребром з E?

Приклад 10.11. Незалежність.

Умова. Дано граф $G = (V, E)$ та натуральне число k, $k \leqslant |V|$.

Запитання. Чи містить граф G незалежну множину потужністю k, тобто чи існує така підмножина $V' \subset V$, що $|V'| = k$ та жодні дві вершини з V' не з'єднані ребром з E?

Приклад 10.12. Вершинне покриття.

Умова. Дано граф $G = (V, E)$ та натуральне число k, $k \leqslant |V|$.

Запитання. Чи існує в графі G вершинне покриття потужністю k, тобто така підмножина $V' \subset V$, що $|V'| = k$ та для кожного ребра $\{u, v\} \in E$ хоча б одна з вершин u чи v належить V'?

Приклад 10.13. Домінантна множина.

Умова. Дано граф $G = (V, E)$ та натуральне число k, $k \leqslant |V|$.

Запитання. Чи існує в графі G домінантна множина потужністю k, тобто така підмножина $V' \subset V$, що $|V'| = k$ та для всіх вершин $u \in V \setminus V'$ існує така вершина $v \in V'$, що $\{u, v\} \in E$?

Приклад 10.14. Розфарбування вершин графа.

Умова. Дано граф $G = (V, E)$ та натуральне число k, $k \leqslant |V|$.

Запитання. Чи існує розфарбування графа G в k кольорів?

Приклад 10.15. Сума елементів підмножини.

Умова. Дано скінченну множину натуральних чисел A та натуральне число k.

Запитання. Чи існує підмножина $A' \subset A$, сума елементів якої дорівнює k?

Продемонструємо техніку доведення *NP*-повноти на основі поліноміального зведення відомої *NP*-повної задачі до задачі, *NP*-повноту якої хочемо довести. Такий спосіб називають *методом поліноміального зведення*.

ТЕОРЕМА 10.4. Задача про 3-виконанність *NP*-повна.

Доведення. Неважко переконатись, що задача про 3-виконанність належить до класу *NP*. Це випливає з того, що недетермінований алгоритм має вгадати лише інтерпретацію (набір значень істинності змінних задачі) і перевірити за поліноміальний час, чи виконуються в ній усі задані диз'юнкції, кожна з яких складається точно з трьох літералів.

Зведемо задачу про виконанність до задачі про 3-виконанність [13]. Нехай $X = \{x_1, x_2, ..., x_n\}$ — множина змінних. $S = \{d_1, d_2, ..., d_m\}$ — набір елементарних диз'юнкцій, який задає індивідуальну задачу про виконанність. На якійсь множині змінних X' побудуємо такий набір S' диз'юнкцій, кожна з яких складається точно з трьох літералів, що набір S' виконанний тоді й лише тоді, коли виконанний набір S.

Набір S' будують, замінюючи кожну диз'юнкцію $d_j \in S$ певним набором S'_j диз'юнкцій із трьох літералів. Набір S'_j утворюють на множині X заданих змін-

них і на множині X'_j деяких додаткових змінних. Зазначимо, що змінні з X'_j уходять лише до диз'юнкцій з S'_j.

Отже,

$$X' = X \cup \left(\bigcup_{i=1}^{m} X'_j \right) \quad \text{та} \quad S' = \bigcup_{j=1}^{m} S'_j.$$

Залишилося показати, як, виходячи з d_j, побудувати S'_j та X'_j. Нехай $d_j = (l_1 \vee l_2 \vee \ldots \vee l_k)$, де l_i $(i = 1, 2, \ldots, k)$ — літерали на множині X. Спосіб побудови S'_j та X'_j залежить від значення k.

1. Якщо $k = 1$, то $X'_j = \{y_j^1, y_j^2\}$,

$$S'_j = \{(l_1 \vee y_j^1 \vee y_j^2), (l_1 \vee y_j^1 \vee \overline{y_j^2}), (l_1 \vee \overline{y_j^1} \vee y_j^2), (l_1 \vee \overline{y_j^1} \vee \overline{y_j^2})\}.$$

2. Якщо $k = 2$, то $X'_j = \{y_j^1\}$, $S'_j = \{(l_1 \vee l_2 \vee y_j^1), (l_1 \vee l_2 \vee \overline{y_j^1})\}$.
3. Якщо $k = 3$, то $X'_j = \varnothing$, $S'_j = \{d_j\}$.
4. Якщо $k \geqslant 4$, то $X'_j = \{y_j^i \mid 1 \leqslant i \leqslant k-3\}$,

$$S'_j = \{(l_1 \vee l_2 \vee y_j^1)\} \cup \{(\overline{y_j^i} \vee l_{i+2} \vee y_j^{i+1}) \mid 1 \leqslant i \leqslant k-4\} \cup \{(\overline{y_j^{k-3}} \vee l_{k-1} \vee l_k)\}.$$

Тепер залишилося перевірити, що набір диз'юнкцій S' виконанний тоді й лише тоді, коли набір диз'юнкцій S виконанний. Нехай τ — інтерпретація (набір значень істинності змінних із множини X), у якій виконано всі диз'юнкції з множини S (тобто всі вони набувають значення Т). Доведемо, що τ можна продовжити до інтерпретації τ' (набору значень змінних із множини X'), у якій усі диз'юнкції з множини S' виконуються. Оскільки змінні з множини $X' \setminus X$ можна поділити на частини X'_j, змінні з яких уходять лише до диз'юнкції з множини S'_j, то достатньо показати, як можна продовжити інтерпретацію τ на кожну множину X'_j окремо. Після цього в кожному випадку потрібно перевірити, що виконуються всі диз'юнкції у відповідній множині S'_j.

Це можна зробити так. Якщо множини X'_j побудовано як у перших двох випадках, то диз'юнкції з набору S'_j уже виконуються в інтерпретації τ, тому τ можна продовжити на множині X'_j довільно. У третьому випадку $X'_j = \varnothing$, і єдина диз'юнкція в наборі S'_j уже виконується в інтерпретації τ. Залишається лише останній випадок, який відповідає диз'юнкції $(l_1 \vee l_2 \vee \ldots \vee l_k) \in S$, причому $k \geqslant 4$. Позаяк в інтерпретації τ всі диз'юнкції з набору S виконуються, то знайдеться таке натуральне число t, що в наборі τ значення l_t дорівнює Т. Якщо t дорівнює 1 або 2, то в наборі τ' значення y_j^i візьмемо таким, що дорівнює F, $1 \leqslant i \leqslant k-3$. Якщо t дорівнює $k-1$ або k, то в наборі τ' значення y_j^i візьмемо таким, що дорівнює Т, $1 \leqslant i \leqslant k-3$. В інших випадках у наборі τ' значення y_j^i візьмемо таким, що дорівнює Т для $1 \leqslant i \leqslant t-2$ і F для $t-1 \leqslant i \leqslant k-3$. Безпосередньо перевіримо, що в інтерпретації τ' виконуються всі диз'юнкції з набору S'_j; отже, у цій інтерпретації виконуються всі диз'юнкції з набору S'. Навпаки, якщо в інтерпретації τ' виконуються всі диз'юнкції з набору S', то легко перевірити, що її звуження лише на змінні з множини X дасть інтерпретацію, у якій виконуються

всі диз'юнкції з набору S. Отже, S' виконується тоді й лише тоді, коли виконується S.

Щоб переконатись, що ця звідність поліноміальна, достатньо зазначити, що кількість диз'юнкцій із трьох літералів у наборі S' обмежена поліномом від m. Отже, розмір індивідуальної задачі про 3-виконанність обмежений зверху поліномом від розміру відповідної задачі про виконанність. Усі деталі побудови зведення очевидні, тому легко перевірити, що воно поліноміальне.

Обмежена структура задачі про 3-виконанність робить її дуже корисною в разі доведення результатів про *NP*-повноту. Використаємо цю задачу для доведення *NP*-повноти задачі про кліку (див. приклад 10.9).

ТЕОРЕМА 10.5. Задача про кліку *NP*-повна.

Доведення. Спочатку переконаємось, що задача про кліку належить до класу *NP*. Справді, маючи список усіх вершин, що можуть утворювати кліку, можна перевірити наявність ребер, які з'єднують їх, за поліноміальний час.

Доведемо тепер, що задача про 3-виконанність поліноміально звідна до задачі про кліку [50]. Для цього потрібно навести функцію f, котра відображає кожну індивідуальну задачу про 3-виконанність у відповідну індивідуальну задачу про кліку й задовольняє умови поліноміальної звідності. Функцію f означають так. Нехай $F = d_1 \wedge d_2 \wedge \ldots \wedge d_k$, де кожна підформула d_r — диз'юнкція трьох літералів $l_1^{(r)}$, $l_2^{(r)}$ та $l_3^{(r)}$. Побудуємо граф $G = (V, E)$, який має кліку потужністю k тоді й лише тоді, коли формула φ виконанна.

Для кожної диз'юнкції $d_r = (l_1^{(r)} \vee l_2^{(r)} \vee l_3^{(r)})$ із формули φ зобразимо три вершини $v_1^{(r)}, v_2^{(r)}, v_3^{(r)}$. Отже, граф G матиме $3k$ вершин (майбутню кліку буде утворено істинними членами диз'юнкцій). Опишемо ребра графа: дві вершини $v_i^{(r)}$ і $v_j^{(s)}$ з'єднано ребром у графі G, якщо виконано такі умови:

- вершини $v_i^{(r)}$ і $v_j^{(s)}$ належать різним трійкам ($r \neq s$);
- літерали, які відповідають цим вершинам, сумісні, тобто літерал $l_i^{(r)}$ — не заперечення літералу $l_j^{(s)}$.

Граф G легко побудувати за формулою φ за поліноміальний час. На рис. 10.6 зображено граф, який відповідає формулі

$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3).$$

Доведемо, що описана функція f — це дійсно зведення. Спочатку припустимо, що формула φ має інтерпретацію, у якій вона виконується. Тоді кожна диз'юнкція d_r містить хоча б один істинний літерал $l_i^{(r)}$, і кожний такий літерал відповідає вершині $v_i^{(r)}$. Виберемо один із таких літералів для кожної диз'юнкції й отримаємо множину V' із k вершин. Множина V' утворює кліку. Справді, для двох довільних вершин $v_i^{(r)}, v_j^{(s)} \in V'$, де $r \neq s$, обидва відповідні літерали $l_i^{(r)}$ і $l_j^{(s)}$ мають значення T, тобто вони не можуть бути запереченнями один одного. Отже, за побудовою графа $G = (V, E)$ ребро $(v_i^{(r)}, v_j^{(s)})$ належить множині E. У прикладі на рис. 10.6 інтерпретація, у якій формула φ виконується, така: $x_1 = \mathrm{F}$, $x_2 = \mathrm{F}$, $x_3 = \mathrm{T}$. У цій інтерпретації диз'юнкція $(x_1 \vee \neg x_2 \vee \neg x_3)$ виконується за рахунок $\neg x_2$, а $(\neg x_1 \vee x_2 \vee x_3)$ та $(x_1 \vee x_2 \vee x_3)$ — за рахунок x_3. Відповідна кліка

потужністю $k = 3$ складається з вершин, які відповідають $\neg x_2$ з першої диз'юнкції, x_3 з другої та x_3 з третьої. На рисунку ці вершини заштриховано.

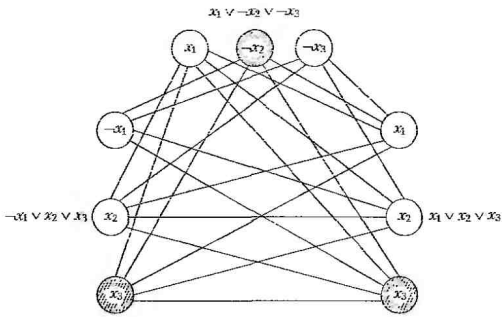


Рис. 10.6

Навпаки, припустимо, що G має кліку V' потужністю k. У кожній трійці вершини не з'єднані ребрами одна з одною, тому кліка V' має точно по одній вершині з кожної трійки. Розглянемо відповідні літерали й оголосимо їх істинними. Сумісність літералів гарантує, що в такому разі не доведеться оголошувати змінну водночас істинною та хибною. Якщо після цього значення деяких змінних усе ще не визначено, виберемо ці значення довільно. На одержаному наборі значень змінних формула φ виконується, бо в кожній диз'юнкції є хоча б один істинний член.

Є задачі, які належать до класу *NP*, проте для їх розв'язання до цього часу не знайдено поліноміальних алгоритмів і невідомо, чи вони *NP*-повні. Найбільш значуща з них задача про ізоморфізм графів.

Приклад 10.16. Ізоморфізм простих графів

Умова. Задано прості графи $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$.

Запитання. Чи ізоморфні графи G_1 і G_2, тобто чи така існує бієкція φ: $V_1 \to V_2$, що

$$\{u, v\} \in E_1 \text{ тоді й лише тоді, коли } \{\varphi(u), \varphi(v)\} \in E_2 \text{ для всіх } u, v \in V_1?$$

Багато практичних задач — це не задачі розпізнавання, тому вони не належать до класу *NP*. Водночас до багатьох із них вдається звести деякі *NP*-повні задачі. У цій ситуації корисним виявляється таке означення. Задачу називають *NP-важкою*, якщо до неї поліноміально зводиться якась *NP*-повна задача.

Теорія *NP*-повноти окрім теоретичного, має й чисто практичне значення. Довівши, що задача *NP*-важка, ми одержуємо достатні причини, щоб відмовитися від пошуку ефективного й точного алгоритму. Можна спрямувати подальші зусилля на отримання наближеного розв'язку.

Контрольні запитання та завдання

1. Задано неорієнтований граф G. Формально визначити задачу відшукання найдовшого циклу в G, який проходить через кожну вершину не більше одного разу. Сформулювати відповідну задачу розпізнавання в термінах ланцюжків.
2. Довести, що задача про вершинне покриття *NP*-повна.
   Вказівка. Звести до неї задачу про кліку.
3. Довести, що задача про суму елементів підмножини *NP*-повна.
   Вказівка. Звести до неї задачу про вершинне покриття.
4. Довести, що задача про гамільтонів цикл *NP*-повна.
   Вказівка. Звести до неї задачу про 3-виконанність.
5. Довести *NP*-повноту задачі розпізнавання з 1.
6. Розглянемо задачу про 2-виконанність.
   Умова. Задано булевий вираз у 2-кон'юнктивній нормальній формі φ. У таких вирazах кожна елементарна диз'юнкція складається рівно з двох різних літералів.
   Запитання. Чи виконанна форма φ , тобто чи існує інтерпретація, у якій формула φ набуває значення T?
   Довести, що задача про 2-виконанність належить до класу *P*.

Термінологічний словник

АВЛ-дерево. Бінарне дерево, у якому висоти двох піддерев кожної з його вершин відрізняються не більше ніж на одиницю.

Алгебра булевих функцій. Множина P_2 всіх булевих функцій разом з уведеною на ній системою операцій.

Алгоритм (в інтуїтивному розумінні). Сукупність правил (програма), для якої дано вказівку, як і в якій послідовності ці правила необхідно застосовувати до початкових даних задачі, щоб одержати її розв'язок. Характеристиками алгоритму є масовість, елементарність кроків, детермінованість кроків, результативність.

Батько вершини *v* в кореневому дереві. Вершина *u* така, що (u, v) є ребром кореневого дерева.

Булеан. Множина, елементами якої є всі підмножини даної множини A, включно з порожньою множиною $\varnothing$ й самою множиною A. Булеан множини A позначають $P(A)$ або 2^A.

Булева змінна. Змінна, яка може приймати два значення: нуль або одиниця.

Булева функція *n* змінних. Функція з E_2^n в E_2, де $E_2 = \{0, 1\}$. Булеву функцію n змінних називають також n-місною.

Вага кореневого дерева. Найбільший рівень вершин кореневого дерева.

Відношення антисиметричне. Відношення R на множині A, яке задовольняє умову: з $(a, b) \in R$ та $(b, a) \in R$ випливає $a = b$.

Відношення бінарне із множини *A* в множину *B*. Підмножина декартового добутку $A \times B$.

Відношення еквівалентності. Рефлексивне, симетричне й транзитивне відношення.

Відношення лінійного порядку. Відношення часткового порядку, у якому кожна пара елементів множини є порівняльною.

Відношення на множині *A*. Бінарне відношення з A в A, тобто підмножина $A \times A$.

Відношення рефлексивне. Відношення R на множині A, яке задовольняє умову: $(a, a) \in R$ для всіх $a \in A$.

Відношення симетричне. Відношення R на множині A, яке задовольняє умову: для всіх $a, b \in A$ з $(a, b) \in R$ випливає $(b, a) \in R$.

Відношення тотального порядку – див. відношення лінійного порядку.

Відношення транзитивне. Відношення R на множині A, яке задовольняє умову: для всіх $a, b, c \in A$ з $(a, b) \in R$ та $(b, c) \in R$ випливає $(a, c) \in R$.

Відношення часткового порядку. Антирефлексивне, симетричне й транзитивне відношення.

Відношення *n*-арне на множинах $A_1, A_2, \ldots, A_n$. Підмножина декартового добутку $A_1 \times \times A_2 \times \ldots \times A_n$.

Вибірка. Скінченна сукупність елементів, відібраних із деякої множини. Залежно від способу відбору, розрізняють вибірки без повторень (відібраний один раз елемент вилучається із множини) та з повтореннями (відібраний елемент не вилучається із множини й може бути відібраним повторно). Залежно від того, задано чи ні порядок елементів, розрізняють вибірки впорядковані й невпорядковані. Вибірку, яка містить r елементів, називають r-вибіркою.

Виконанна формула. Формула логіки висловлювань, яка набуває значення Т (істина) хоча б на одному наборі значень змінних.

Висловлювання. Висловлюванням називають розповідне речення, яке може бути або істинним, або хибним (фальшивим), але не одним та іншим разом. Розділ логіки, який вивчає висловлювання та їхні властивості, називають пропозиційною логікою, або логікою висловлювань.

Внутрішня вершина. Вершина кореневого дерева, яка має синів.

Гамільтонів граф. Граф, який має гамільтонів цикл.

Гамільтонів цикл. Такий цикл $x_0, x_1, ..., x_n, x_0$ у неорієнтованому зв'язному графі, що $x_0, x_1, ..., x_n$ –гамільтонів шлях.

Гамільтонів шлях. Шлях у неорієнтованому зв'язному графі, який містить кожну вершину графа точно один раз.

Гомеоморфні графи. Графи, які можуть бути отримані з одного графа включенням у його ребра нових вершин степеня 2.

Граматика типу 0. Граматика, на продукції якої не накладено жодних обмежень.

Граматика типу 1, або контекстно залежна граматика. Граматика, у якій усі продукції мають вигляд $\xi \to \eta$, де $|\xi| \leqslant |\eta|$, або $\eta = \lambda$.

Граматика типу 2, або контекстно вільна граматика. Граматика, у якій усі продукції у формі $A \to \eta$, де A є нетермінальним символом.

Граматика типу 3, або регулярна граматика. Граматика, у якій усі продукції у формі $A \to aB$, $A \to a$, $S \to \lambda$, де A, B – нетермінальні символи. a – термінальний символ.

Граф. Термін, який у цьому підручнику використовується для довільних графів — неорієнтованих і орієнтованих, з петлями й кратними ребрами (дугами) або без них. Зазначимо, що в літературі термін „граф" часто використовують замість терміна „простий граф".

Декодування. Відновлення повідомлення за його кодом.

Дерево. Зв'язний неорієнтований граф без простих циклів.

Дерево бінарне. Це m-арне дерево в разі $m = 2$.

Дерево впорядковане. Кореневе дерево, у якому сини кожної внутрішньої вершини впорядковані.

Дерево збалансоване. Кореневе дерево, у якому кожний листок має рівень h або $h - 1$, де h – висота дерева.

Дерево з'єднувальне – див. каркас.

Дерево кореневе. Орієнтований граф із виділеною вершиною, яку називають коренем. Ребра в кореневому дереві орієнтовані „від кореня", існує єдиний шлях від кореня до будь-якої іншої вершини.

Дерево пошуку бінарне. Бінарне дерево, у якому кожній вершині присвоєно значення, яке називають ключем. Ключ — це елемент деякої лінійно впорядкованої множини. У кожній вершині ліве піддерево містить лише ключі, менші від ключа цієї вершини, а праве — ключі, більші від ключа вершини.

Дерево прийняття рішень. Кореневе дерево, кожному листку якого поставлено у відповідність рішення зі скінченної множини відомих рішень, а кожній внутрішній вершині v — перевірку умови $P(v)$. Прийняття рішень за допомогою такого дерева полягає в побудові простого шляху від кореневої вершини до листка. Під час побудови шляху виконують перевірку умов у внутрішніх вершинах дерева. Значення умови $P(v)$ у вершині v визначає ребро, яке буде додано в шлях після вершини v. Кінцева вершина побудованого шляху вказує прийняте рішення.

Дерево *m*-арне. Кореневе дерево, у якого кожна внутрішня вершина має не більше ніж m синів.

Дерево *m*-арне повне. Кореневе дерево, у якого кожна внутрішня вершина має точно m синів.

Діаграма Гассе. Графічне подання частково впорядкованої множини.

Ейлерів граф. Граф, який має ейлерів цикл.

Ейлерів цикл. Цикл у неорієнтованому зв'язному графі, який містить кожне ребро графа точно один раз.

Ейлерів шлях. Шлях у неорієнтованому зв'язному графі, який містить кожне ребро графа точно один раз.

Елементарні коди. Коди β_i у схемі кодування.

Жадібний алгоритм. Алгоритм оптимізації, що на кожному кроці здійснює вибір найкращого з можливих варіантів.

Зв'язний граф. Неорієнтований граф, у якому між кожною парою вершин існує шлях.

Ізоморфні графи. Графи, між множинами вершин яких можна встановити взаємно-однозначну відповідність, що зберігає суміжність кратності ребер, петлі й спрямування дуг.

Інваріант. Властивість, яку ізоморфні графи або обидва мають, або обидва не мають.

Інцидентність. Вершину v і ребро e називають інцидентними, коли вершина v є кінцем ребра e.

Каркас. Каркас простого зв'язного графа — це підграф, що є деревом, і містить усі вершини даного графа.

Клас *NP*. — це клас усіх мов або масових задач, що розпізнаються недетермінованими машинами Тьюрінга, у яких час роботи при довільній послідовності недетермінованих виборів поліноміально обмежений.

Клас *P*. Клас, який складається з усіх мов або масових задач, що розпізнаються детермінованими машинами Тьюрінга з поліноміальною залежністю часу роботи від довжини входу.

Кліка. Підмножина вершин простого графа, у якій будь-які дві вершини суміжні.

Код. 1. Код повідомлення. 2. Множина елементарних кодів.

Коди Хемінга. Коди, які мають здатність виправляти одну помилку на n переданих розрядів. Запропоновані Хемінгом у 1950 р.

Код повідомлення. Нехай задано скінченний алфавіт $B = \{b_1, ..., b_q\}$. Через β позначимо слово в алфавіті B і через B^* — множину всіх слів в алфавіті B. Задамо відображення F, яке кожному слову α, $\alpha \in S$, ставить у відповідність слово $\beta = F(\alpha)$, $\beta \in B^*$. Слово β називають кодом повідомлення α.

Кодування. Перехід від слова α до його коду β.

Кодування алфавітне. Кодування, яке задають схемою (або таблицею кодів) σ: $a_i \to \beta_i$, де $a_i \in A$, $\beta_i \in B^*$, $i = 1, ..., r$. Схема σ задає відповідність між буквами алфавіту A і деякими словами в алфавіті B. Вона визначає алфавітне кодування так: кожному слову $\alpha = a_{i_1}a_{i_2}...a_{i_k}$ з S ставиться у відповідність слово $\beta = \beta_{i_1}\beta_{i_2}...\beta_{i_k}$.

Кодування рівномірне. Рівномірне кодування (з параметрами k та n) визначають так. Повідомлення α розбивають на блоки довжини k. Ці блоки розглядають як „букви" деякого алфавіту (таких блоків є, очевидно, r^k, оскільки алфавіт A складається з r букв) і кодують словами в алфавіті B (однакової) довжини n за схемою рівномірного кодування $\sigma_{k,n}$: $\alpha_i \to \beta_i$ ($i = 1, ..., r^k$).

Кортеж. Упорядкований набір елементів. Використовують також терміни вектор, рядок, ланцюжок.

Ланцюжок над алфавітом *V*. Упорядкована сукупність символів алфавіту V.

Ліс. Неорієнтований граф без простих циклів.

Листок. Вершина кореневого дерева, яка не має синів.

Матриця інцидентності. Матриця для подання графа, використовує інцидентність вершин і ребер.

Матриця суміжності. Матриця для подання графа, використовує суміжність вершин.

Метод Фано. Метод побудови коду, близького до оптимального.

Машина Тьюрінга. Одне з уточнень поняття алгоритму, засноване на уявленні про алгоритм як про деякий детермінований пристрій, здатний виконувати в кожний окремий момент лише дуже примітивні операції. Таке уявлення не залишає сумнівів у однозначності алгоритму й елементарності його кроків. Евристика цієї алгоритмічної моделі дуже близька до комп'ютерів.

Метод Хаффмана. Метод побудови оптимального коду.

Множина. Будь-яке зібрання визначених і відмінних один від одного об'єктів нашої інтуїції або інтелекту, яке мислять як єдине ціле.

Множина лінійно впорядкована. Множина із заданим на ній лінійним порядком.

Множина-степінь – див. булеан.

Множина тотально впорядкована – див. множина лінійно впорядкована.

Множина універсальна. Найширша множина, така, що всі множини, які розглядають у певному контексті, є її підмножинами. Позначають U.

Множина частково впорядкована. Множина із заданим на ній частковим порядком.

Мова. Підмножина множини всіх ланцюжків над алфавітом.

Мова, яка розпізнається скінченним автоматом без виходу. Множина всіх ланцюжків, кожний з яких переводить початковий стан автомата в заключний.

Мультиграф. Пара (V, E), де V – скінченна непорожня множина вершин, а E – сім'я невпорядкованих пар різних елементів з V. Застосування терміну „сім'я" замість „множина" означає, що елементи в E (ребра) можуть повторюватись. Ребра, які з'єднують одну й ту саму пару вершин, називають кратними.

Напівстепінь виходу вершини. Кількість дуг, що мають цю вершину початковою.

Напівстепінь входу вершини. Кількість дуг, що мають цю вершину кінцевою.

Незалежна множина. Підмножина вершин простого графа, яка не містить суміжних вершин.

Неорієнтований граф. Термін „неорієнтований граф" або „псевдограф" у цьому підручнику використовується для довільного неорієнтованого графа, який може мати кратні ребра й петлі.

Обхід дерева. Упорядкування вершин дерева.

Обхід бінарного дерева в зворотному порядку (обхід знизу вверх). Упорядкування вершин дерева, визначене рекурсивно як A, B, R – тобто корінь відвідують після обходу піддерев.

Обхід бінарного дерева в прямому порядку (обхід зверху вниз). Упорядкування вершин бінарного дерева, визначене рекурсивно як R, A, B – тобто корінь відвідують до обходу піддерев.

Оптимальний код. Код із мінімальною надлишковістю (має найменшу середню довжину елементарного коду).

Орієнтований граф. Пара (V, E), де V – скінченна непорожня множина вершин, а E – множина впорядкованих пар елементів множини V. Елементи множини E у випадку орієнтованого графа називають дугами. Дугу (v, v) називають петлею.

Орієнтований мультиграф. Пара (V, E), де V – скінченна непорожня множина вершин, а E – деяка сім'я впорядкованих пар елементів з V. Дуги, що повторюються, називають кратними.

Перестановка. Особливий випадок розміщення без повторень, коли в розміщення входять усі елементи.

Перестановка з повтореннями. Перестановка за умови, що не всі елементи різні.

Планарний граф. Граф, який можна зобразити на площині так, що його ребра не перетинатимуться.

Повідомлення. Нехай $A = \{a_1, ..., a_s\}$ — алфавіт, S — деяка підмножина множини A^*: $S \subset A^*$; елементи множини S називають повідомленнями.

Поліноміальне зведення. Мова $L_1 \subset V_1^*$ поліноміально зводиться до мови $L_2 \subset V_2^*$, якщо існує функція $f : V_1^* \to V_2^*$, що задовольняє такі умови: 1) існує ДМТ-програма, яка обчислює f із часовою складністю, обмеженою поліномом; 2) для будь-якого $\xi \in V_1^*$ умова $\xi \in L_1$ виконується тоді й тільки тоді, коли $f(\xi) \in L_2$.

Польський запис — див. префіксний запис.

Польський запис зворотний — див. постфіксний запис.

Породжувальна граматика $G = (V, T, S, P)$. Засіб подання мови. Вона містить алфавіт V, множину термінальних символів T, початковий символ S і множину продукцій P.

Постфіксний запис. Форма запису виразу, яку одержують як результат обходу бінарного дерева цього виразу в зворотному порядку.

Предикат. Предикат (пропозиційна функція) $P(x_1, ..., x_n)$ від n змінних, n-місний предикат на множині D — це функція, визначена на наборах $(a_1, ..., a_n)$ елементів з D зі значеннями в множині {T, F}.

Префіксний запис. Форма запису виразу, яку одержують як результат обходу бінарного дерева цього виразу в прямому порядку.

Продукція граматики. Продукція, або правило перетворення має вигляд $\xi \to \eta$, де ξ та η — ланцюжки над алфавітом V. У разі наявності такого правила ланцюжок ξ може бути замінений ланцюжком η, коли ξ трапляється в ланцюжку мови.

Простий граф. Пара $G = (V, E)$, де V — непорожня скінченна множина вершин, E — множина невпорядкованих пар різних елементів з V. Елементи множини E (невпорядковані пари різних вершин) називають ребрами.

Протиріччя. Формула логіки висловлювань, яка є завжди хибною. Використовують також терміни заперечувана або невиконанна формула.

Псевдограф. Пара (V, E), де V — скінченна непорожня множина вершин, а E — сім'я невпорядкованих пар не обов'язково різних вершин. Ребро $\{v, v\}$ називають петлею.

Рекурентне рівняння. Рівняння, що описує правило для знаходження елементів послідовності через один або декілька попередніх, причому задається відповідна кількість початкових елементів.

Рекурсивні функції. Історично перша формалізація поняття алгоритму. Ця алгоритмічна модель пов'язує поняття алгоритму з найтрадиційнішими поняттями математики — обчисленнями та числовими функціями.

Реляційна модель даних. Модель для подання баз даних, яка ґрунтується на n-арних відношеннях.

Рівень вершини в кореневому дереві. Довжина шляху від кореня до цієї вершини.

Роздільна схема. Схема алфавітного кодування, для якої будь-яке слово, складене з елементарних кодів, єдиним способом розкладається на елементарні коди. Алфавітне кодування з роздільною схемою допускає однозначне декодування.

Розміщення. Розміщення з n елементів по r — це впорядкована r-вибірка з n-елементної множини. Залежно від способу побудови вибірки розміщення є без повторень і з повтореннями.

Розфарбування графа. Таке приписування кольорів вершинам простого графа, що ніякі дві суміжні вершини не набувають однакового кольору.

Син вершини u в кореневому дереві. Будь-яка вершина, для якої вершина u є батьком.

Скінченний автомат без виходу детермінований. Система $M = (S, I, f, s_0, F)$, у якій S — скінченна множина станів, I — скінченний вхідний алфавіт, f: $S \times I \to S$ — функція переходів, визначена на декартовому добутку $S \times I$, $s_0 \in S$ — початковий стан, $F \subset S$ — множина заключних (або приймаючих) станів.

Скінченний автомат без виходу недетермінований. Система $M = (S, I, f, s_0, F)$, у якій S — скінченна множина станів, I — скінченний вхідний алфавіт, f — функція переходів, яка кожній парі стан–вхід ставить у відповідність множину станів, $s_0 \in S$ — початковий стан, $F \subset S$ — множина заключних станів. Єдиною відмінністю між недетермінованим і детермінованим автоматами є тип значень функції переходів f. Для недетермінованого автомата це множина станів (вона може бути й порожньою), а для детермінованого — один стан.

Скінченний автомат із виходом (або автомат Мілі). Система $M = (S, I, O, f, g, s_0)$, у якій S, I, O — скінченні множини, а f: $S \times I \to S$ та g: $S \times I \to O$ — функції, визначені на декартовому добутку $S \times I$. Множину S називають множиною станів, I — вхідним алфавітом, O — вихідним алфавітом, f — функцією переходів, g — функцією виходів, виділений елемент $s_0 \in S$ — початковим станом.

Сполучення. Сполучення з n елементів по r — це невпорядкована r-вибірка з n-елементної множини. Залежно від способу побудови вибірки сполучення є без повторень і з повтореннями.

Степінь вершини. Кількість ребер у неорієнтованому графі, інцидентних цій вершині, причому петлю враховують двічі.

Суміжність. Дві вершини u та v у неорієнтованому графі називають суміжними, якщо $\{u, v\}$ є ребром. Якщо $e = \{u, v\}$ — ребро, то вершини u та v називають його кінцями. Два ребра називають суміжними, якщо вони мають спільний кінець.

Схема з функціональних елементів. Конструкція, що складається з функціональних елементів, з'єднаних за певними правилами. Допустимі з'єднання елементів відповідають суперпозиціям булевих функцій, які цими елементами реалізуються. Конкретна схема з функціональних елементів реалізує певну булеву функцію.

Тавтологія. Формула логіки висловлювань, яка є завжди істинною. Використовують також термін загальнозначуща формула.

Теза Тьюрінга. Будь-який алгоритм може бути реалізований машиною Тьюрінга.

Теза Чорча. Будь-яка функція, що обчислюється деяким алгоритмом, є частково рекурсивною.

Універсум – див. множина універсальна.

Формула. Вираз, що описує складне висловлювання, яке одержують з одного або декількох висловлювань застосуванням логічних операцій. Розглядається п'ять логічних операцій: заперечення, кон'юнкція, диз'юнкція, імплікація та еквівалентність.

Функція, обчислювана за Тьюрінгом. Така числова функція, для якої існує машина Тьюрінга, що її реалізує.

Цикл. Шлях, який починається й закінчується в одній і тій самій вершині.

Цикл простий. Цикл, який жодне ребро не містить більше одного разу.

Числова фувкція. Функція, значеннями якої й значеннями її аргументів є невід'ємні цілі числа.

Шлях. Шлях із вершини u до вершини v в неорієнтованому графі — послідовність одного або більше ребер $e_1, e_2, ..., e_r$, де $e_i = \{x_{i-1}, x_i\}$, $i = 1, 2, ..., r$, $x_0 = u$, $x_r = v$. Шлях із вершини u до вершини v в орієнтованому графі — послідовність однієї або більше дуг $e_1, e_2, ..., e_r$, де $e_i = (x_{i-1}, x_i)$, $i = 1, 2, ..., r$, $x_0 = u$, $x_r = v$.

Шлях простий. Шлях, який жодне ребро не містить більше одного разу.

***NP*-повна задача.** Задача, що належить до класу *NP* і будь-яку задачу з *NP* можна поліноміально звести до неї.

***NP*-важка задача.** Задача, до якої поліноміально зводиться деяка *NP*-повна задача.

Література

1. *Андерсон Д.* Дискретная математика и комбинаторика. — СПб.: Вильямс, 2003. — 958 с.
2. *Ахо А., Хопкрофт Дж., Ульман Дж.* Структуры данных и алгоритмы. — СПб: Вильямс, 2000. — 384 с.
3. *Басакер Р., Саати Т.* Конечные графы и сети. — М.: Наука 1974. — 368 с.
4. *Берж К.* Теория графов и ее применения. — М.: И.Л, 1962. — 319 с.
5. *Биркгоф Г., Барти Т.* Современная прикладная алгебра. — М.: Мир. 1976. — 400 с.
6. *Брауэр В.* Введение в теорию конечных автоматов. — М.: Радио и связь, 1987. — 392 с.
7. *Вирт Н.* Алгоритмы и структуры данных. — М.: Мир. 1989. — 360 с.
8. *Гаврилов Г. П., Сапоженко А. А.* Сборник задач по дискретной математике. — М.: Наука. 1977. — 368 с.
9. *Гладкий А. В.* Формальные грамматики и языки. — М.: Наука. 1973. — 368 с.
10. *Глибовець М. М., Олецький О. В.* Штучний інтелект. — К.: Академія, 2002. — 368 с.
11. *Глушков В. М.* Синтез цифровых автоматов. — М.: Физматгиз, 1962. — 476 с.
12. *Гросс М., Лантен А.* Теория формальных грамматик. — М.: Мир. 1971. — 296 с.
13. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982. — 416 с.
14. *Евстигнеев В. А.* Применение теории графов в программировании. — М.: Наука, 1985. — 352 с.
15. *Емеличев В. А., Мельников О. И., Сарванов В. И, Тышкевич Р. И.* Лекции по теории графов. — М.: Наука, 1990. — 384 с.
16. *Капітонова Ю. В., Кривий С. Л., Летичевський О. А., Луцький Г. М., Печурін М. К.* Основи дискретної математики. — К.: Наукова думка, 2002. — 580 с.
17. *Карпов В. Г., Мощенский В. А.* Математическая логика и дискретная математика. — Минск: Вышэйшая школа, 1977. — 256 с.
18. *Клини С.* Введение в метаматематику. — М.: И.Л. 1957. — 674 с.
19. *Кофман А.* Введение в прикладную комбинаторику. — М.: Наука, 1975. — 480 с.
20. *Кристофидес Н.* Теория графов. Алгоритмический подход. — М.: Мир. 1978. — 432 с.
21. *Кузнецов О. П., Адельсон-Вельский Г. М.* Дискретная математика для инженера. — М.: Энергоатомиздат, 1988. — 480 с.
22. *Кук Д., Бейз Г.* Компьютерная математика. — М.: Наука, 1990. — 384 с.
23. *Липский В.* Комбинаторика для программистов. — М.: Мир, 1988. — 216 с.
24. *Люгер Д.* Искусственный интеллект. — СПб.: Вильямс, 2003. — 864 с.
25. *Майника Э.* Алгоритмы оптимизации на сетях и графах. — М.: Мир. 1981. — 328 с.
26. *Мальцев А. И.* Алгоритмы и рекурсивные функции. — М.: Наука, 1986. — 367 с.
27. *Марков А. А.* Введение в теорию кодирования. — М.: Наука, 1982. — 192 с.
28. *Мендельсон Э.* Введение в математическую логику. — М.: Мир, 1988. — 320 с.

29. *Мошков М. Ю.* Деревья решений. Теория и приложения. — Нижний Новгород: Нижегород. ун-т, 1994. — 175 с.

30. *Мощенский В. А.* Лекции по математической логике. — Минск: БГУ, 1973. — 160 с.

31. *Нікольський Ю. В., Пасічник В. В., Щербина Ю. М.* Дискретна математика. — Львів: Магнолія Плюс, 2005. — 608 с.

32. *Нікольський Ю. В., Щербина Ю. М.* Дослідження ефективності використання дерев прийняття рішень для прогнозування діагнозу в медицині // Вісник НУ „Львівська політехніка". Інформаційні системи та мережі. — 2004. — № 519. — С. 244–253.

33. *Нікольський Ю. В., Щербина Ю. М., Якимечко Р. Я.* Дерева прийняття рішень та їхнє застосування для прогнозування діагнозу в медицині. // Вісник Львів. ун-ту, сер. прикл. матем. та інформатика. — 2003. — Вип. 6. — С. 191–211.

34. *Новиков П. С.* Элементы математической логики. — М.: Наука, 1973. — 400 с.

35. *Новиков Ф. А.* Дискретная математика для программистов. — СПб.: Питер, 2000. — 302 с.

36. *Оре О.* Теория графов. — М.: Мир, 1980. — 336 с.

37. *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. — М.: Мир, 1985. — 510 с.

38. *Пасічник В. В.* Використання частково-визначених булевих функцій для аналізу реляційних моделей баз даних. // Республіканський збірник „Контрольно-вимірювальна техніка". — Львів, 1981. — С. 97–104

39. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы. Теория и практика. — М.: Мир, 1980. — 476 с.

40. *Сергиенко И. В.* Математические модели и методы решения задач дискретной оптимизации. — К.: Наукова думка, 1985. — 381 с.

41. *Столл Р.* Множества. Логика. Аксиоматические теории. — М.: Просвещение, 1968. — 230 с.

42. *Тейз А., Грибомон П., Юлен А., Пирот А. и др.* Логический подход к искусственному интеллекту (От модальной логики к логике баз данных). — М.: Мир, 1998. — 494 с.

43. *Уилсон Р.* Введение в теорию графов. — М.: Мир, 1977. — 207 с.

44. *Харари Ф.* Теория графов. — М.: Мир, 1973. — 300 с.

45. *Холл М.* Комбинаторика. — М.: Мир, 1970. — 234 с.

46. *Хопкрофт Дж., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. — СПб.: Вильямс, 2002. — 528 с.

47. *Цейтлін Г. Є.* Елементи теорії булевих функцій. — К: Техніка, 1973. — 76 с.

48. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. — М.: Наука, 1983. — 256 с.

49. *Яблонский С. В.* Введение в дискретную математику. — М.: Наука, 1986. — 384 с.

50. *Cormen T., Leiserson Ch., Rivest R.* Introduction to Algorithms. The MIT Press, 1999. — 1028 p.

51. *Michaels J., Rosen K.* Applications of Discrete Mathematics. — McGraw-Hill, 1991. — 454 p.

52. *Rosen K.* Discrete Mathematics and Its Applications. — McGraw-Hill, 2002. — 886 p.

Алфавітний покажчик

Д

Е

З

І

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ч

Ш

Навчальне видання

Нікольський Юрій Володимирович,
Пасічник Володимир Володимирович,
Щербина Юрій Миколайович

ДИСКРЕТНА МАТЕМАТИКА

Підручник

Керівник проекту В. П. Пасько
Редактор С. Г. Атдаєва
Коректор Н. М. Тонконог
Комп'ютерна верстка Д. С. Трішенкова

ТОВ «Видавнича група BHV»
Свідоцтво про внесення до Державного реєстру України
суб'єктів видавничої справи
серія ДК №175 від 13.09.2000 р.

Підписано до друку 22.08.06. Формат 70×100 $^1/_{16}$
Папір офсетний. Гарнітура Petersburg. Друк офсетний.
Ум. друк. арк. 29,67. Обл.-вид. арк. 25,34
Наклад 2500 прим. Зам. №30057

Виготовлено в ТОВ «Освітня книга»,
м. Київ, вул. Орловська, 2/7, оф. 6
Свідоцтво про внесення до Державного реєстру
суб'єктів видавничої справи України
серія ДК № 2245 від 26.07.2005 р.