

# Вступ. Дискретні структури

## Предмет «Дискретні структури»

**Дискретна математика** — математика, яка вивчає структури з дискретними елементами, тобто з такими, які можна чітко розрізнати один від одного.



Рис. 1. Дискретна математика та її розгалуження

Сучасна математична теорія ґрунтується на базисі затверджених аксіом та має множину теорем, які одержані за допомогою прийнятої логіки. Таку теорію можна представити як на рис. 2.

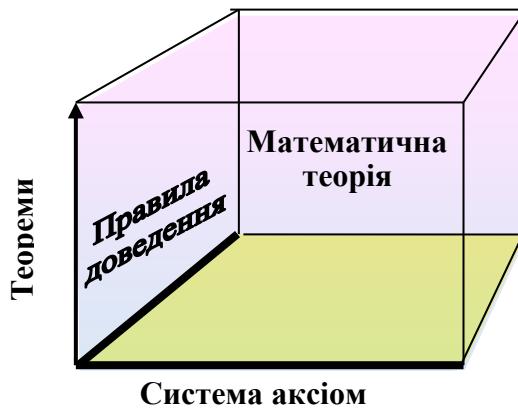


Рис. 2. Структура математичної теорії

Математична теорія, яка ґрунтуюється на множині об'єктів та певній сукупності відношень між цими об'єктами одержала назву математичної структури. Якщо маються на увазі дискретні, конструктивні об'єкти, то маємо дискретну структуру.

Отже, у даному курсі вивчаються дискретні структури. У курсі комп'ютерної дискретної математики вже вивчались такі дискретні структури, як алгебри, зокрема булева алгебра, алгебра перемикальних функцій, а також алгебра множин, дискретна структура автомату. У даному курсі особливе значення приділяється до таких дискретних структур, як графи.

## **Об'єкти.**

Визначення в науці бувають прямі (конструктивні) та непрямі (дескриптивні).

**Пряме визначення об'єкта** — явний опис об'єкта.

Наприклад, арифметичною прогресією називається ряд чисел  $a, a+d, a+2d, \dots$ , де  $a$  і  $d$  — фіксовані числа.

Кілограмом є маса еталону кілограма.

Пряме або конструктивне визначення об'єкту є одночасно і доведенням його існування.

**Непряме визначення** об'єкта — опис, який задає об'єкт як перелік його необхідних властивостей.

Наприклад, арифметична прогресія — це послідовність чисел  $a_1, a_2, a_3, \dots$  для якої різниця  $z = a_{i+1} - a_i$  постійна для усіх  $i = 1, 2, \dots$ .

Непряме визначення об'єкта не доводить його існування.

**Об'єкти** в математиці — це об'єкти різної природи, які відрізняються один від одного найменуваннями, що їм умовно приписуються.

В дискретній математиці, у тому числі в дискретних структурах розглядаються лише конструктивні об'єкти.

**Конструктивні об'єкти** мають дві основні властивості:

- мають пряме визначення,
- повинні чітко, безпомилково відрізнятись один від одного.

Оскільки об'єкт має пряме, тобто, конструктивне визначення, то він завжди існує або може бути побудований (зконструйований) за цим визначенням. Наприклад, багаторозрядне просте число може бути знайдене за алгоритмом решета Ератосфена.

Вимога чіткого виділення даного об'єкта серед інших є основою дискретної математики. Вона дає змогу формально, а не інтуїтивно, виділяти об'єкти та оперувати з ними.

Бувають складні конструктивні об'єкти, які компонуються з простіших об'єктів за прямим визначенням. Це, наприклад слова з літер, багаторозрядні числа, масиви, матриці даних.

## **Множини.**

**Множина** — це сукупність певних об'єктів довільної природи. Наприклад, можна говорити про множину всіх книг у певній бібліотеці, множину літер українського алфавіту. Об'єкти, які складають множину, називають **елементами** цієї множини.

Множина задається декларативно (конструктивно) як перелік її елементів. Нехай множина  $X$  містить елементи  $a, b, c, \dots, k$ . Для запису цього факту застосовують такий вираз:

$$X = \{a, b, c, \dots, k\}.$$

Також множина задається зі вказівкою властивостей елементів або правила їх побудови. Наприклад, множина натуральних чисел менших 100 позначається як:

$$A = \{x \in \mathbb{N} \mid x < 100\}.$$

**Підмножиною  $B$  множини  $A$**  називається множина, усі елементи якої належать до множини  $A$ , хоча можуть бути елементи з  $A$ , які не належать  $B$ . Це позначається як  $B \subset A$ , тобто,  $B = \{x \in A\}$ .

**Потужністю**  $|A|$  множини  $A$  називається число її членів.

Якщо можна класифікувати конструктивні об'єкти за загальним правилом їх конструювання, то множину таких об'єктів називають **ансамблем** конструктивних об'єктів.

Множина натуральних  $\mathbb{N}$  чисел є ансамблем, бо є правило їх побудови.

## Структури

Отже, як було показано вище, математичну теорію можна зобразити як деякий простір, що складається з аксіом, правил доведення (які також вважаються за аксіоми) та теорем, які визначені на їх основі. Така теорія оперує з визначеними математичними об'єктами. Залишилось визначити взаємовідношення між цими об'єктами, аксіомами та теоремами.

З метою розвитку універсального підходу до вивчення математики, Н. Бурбакі у 1948 р. ввів поняття математичної структури.

**Математична структура** — це визначені одна чи кілька множин об'єктів із заданою системою відношень між елементами цих множин. Причому об'єкти можуть бути невизначеними, мати різну природу і відрізнятись один від одного лише за найменуванням, яке надається умовно.

Наприклад, *структура натуральних чисел* визначається як множина  $\mathbb{N}$  елементів, що називаються числами, у якій визначені дві бінарні операції, що називаються додаванням та множенням чисел. Ці операції зіставляють з кожними двома числами  $m$  і  $n$  третє число, яке позначається відповідно через  $m + n$  та через  $m \cdot n$  або  $mn$ . Крім того, у множині  $\mathbb{N}$  виділяється «найперше» число 1 та задається унарна операція «'» переходу до наступного числа  $n' = n + 1$ . У зв'язку з цим структура натуральних чисел  $\mathbb{N}$  позначається як

$$\mathbb{N} = \langle \mathbb{N}; +, \cdot, ' \rangle.$$

У цій структурі операції додавання та множення визначені дескриптивно через списки аксіом цих операцій, які задають їхні основні властивості.

## Відношення

У математичних структурах задаються унарні, бінарні, тернарні і т.д. відношення на певній множині або кількох множинах.

**Бінарне відношення**  $R$  — зв'язує пари елементів з декартового добутку множин  $A \times B$ . **Декартовий добуток** визначається наступним чином:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

Бінарне відношення  $R(x, y)$  вказує певну підмножину у множині пар  $(x, y) \in A \times B$ . Причому відношення вказує усі пари  $(x, y)$  для яких існує відношення  $xRy$  або  $R(x, y)$ .

Бінарне відношення  $E(v_i, v_j)$  яке задане на декартовому квадраті  $(v_i, v_j) \in V \times V, i, j = 1, 2, \dots, |V|$  часто називають **графом**. Тоді граф визначається як дискретна структура  $G = \langle V; E \rangle$ , де  $V$  — множина вершин  $v_i$ , а  $E$  — відношення, яке задає дуги  $(v_i, v_j)$  графа.

Граф, як правило, ілюструється графічним малюнком, в якому вершини представлені жирними точками або кружечками, а дуги — лініями, які з'єднують вершини.

## Історія теорії графів

Першою працею в історії теорії графів вважають статтю Леонарда Ейлера, присвячену задачі про кенігсберзькі мости, яку він опублікував у 1736 році [1]. У місті Кенігсберг було 7 мостів, що з'єднували між собою частини міста, розташовані по обидва боки річки і на двох островах (рис. 3, а). Задача полягала в тому, щоб з'ясувати, чи можливо виконати прогулянку містом таким чином, щоб шлях пройшов через кожен міст рівно один раз. Ейлер зазначив, що вибір шляху в межах будь-якої з частин суходолу не має значення. Важлива лише послідовність перетину мостів.

Таким чином, можна абстрагуватися від зайвих ознак, залишивши тільки частини суходолу та мости як зв'язки між ними. Відповідно, частини суходолу можна зобразити як вершини графу, а мости — як його ребра (рис. 3, б). Ейлер довів, що можливість пройти через кожне ребро графу рівно один раз є лише в разі, якщо кожна вершина (крім тих, що є початком і кінцем маршруту) пов'язана з парною кількістю ребер. У даному випадку всі чотири вершини дотичні до непарної кількості ребер. Лише дві з них можуть бути початком і кінцем маршруту. Таким чином Ейлер довів, що така прогулянка кенігсберзькими мостами не може бути здійснена.



Рис. 3. До задачі про кенігсберзькі мости:

а — план міста; б — граф

Згодом на честь Ейлера такий ланцюг у графі, що проходить кожне ребро один раз, стали називати ейлеровим ланцюгом. Окремий випадок ейлерового ланцюга — ейлерів цикл. Це ейлерів ланцюг, що починається і закінчується в одній і тій же вершині.

Ейлер також представив графом вершини та грані багатогранників. Вершини і ребра багатогранника можна розглядати як вершини і ребра графа. При цьому ми відволікаємося від того, як розташовані елементи багатогранника в просторі, залишаючи лише інформацію про те, які вершини з'єднані ребрами. На рис. 4 показані п'ять способів зобразити один і той же граф тривимірного куба.

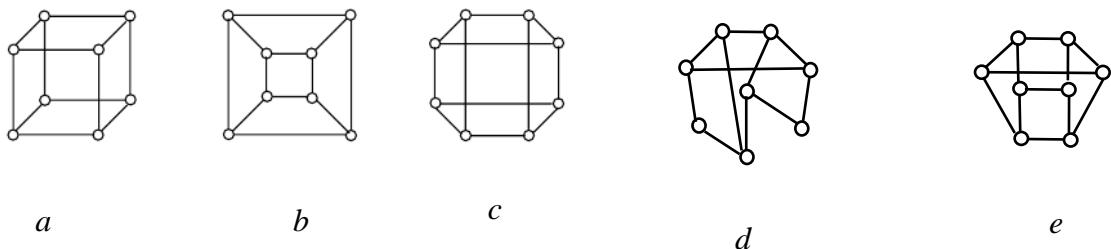


Рис.4.

Формула Ейлера, яка зв'язувала кількість вершин та граней багатогранника. Ейлерова характеристика топологічних поліедрів може бути порахована за формулою:

$$\Gamma - P + V = \chi \text{ де } \Gamma, P \text{ і } V \text{ кількість граней, ребер і вершин відповідно.}$$

Зокрема, для будь якого многогранника справедлива **формула Ейлера**:

$$\Gamma - P + V = 2.$$

Наприклад, характеристика Ейлера для куба на рис. 4 дорівнює 6 — 12 + 8 = 2. Ці знання започаткували таку нову галузь математики, як топологія.

Через сторіччя після формулювання задачі Ейлера Кейлі ввів окремий клас графів – дерева і розвив напрямок – нумерація дерев.

Власне, автором терміну «граф» вважається Д. Д. Сильвестр, який у статті, опублікованій 1878 року в журналі Nature, проводить аналогію між

такою дискретною структурою, як граф і графічним представленням хімічних молекул. Але найбільше поширення термін «граф» одержав після друку книги угорського математика Д. Кьоніга у 1936 р., а потім – книжки Ф. Харарі, яка вийшла у 1969 р.

Однією з найбільш відомих задач у теорії графів є задача чотирьох кольорів: "Чи правда, що на будь-якій карті, області, які накреслені на площині, можуть бути розфарбовані чотирма кольорами таким чином, що будь-які дві області, що мають спільну межу, мають різний колір?" Ця задача була вперше сформульована 1852 р, але її остаточний розв'язок невідомий досі. Найбільших досягнень при її розв'язку було одержано за допомогою комп'ютерного перебору графів.

Теорія графів має потужний апарат розв'язання прикладних задач з найрізноманітніших галузей науки та техніки. Сюди відносяться, наприклад, аналіз та синтез ланцюгів і систем, проектування каналів зв'язку та дослідження процесів передачі інформації, побудова контактних схем і дослідження скінчених автоматів, мережеве планування й управління, дослідження операцій, вибір оптимальних маршрутів і потоків у мережах, моделювання життедіяльності й нервової системи живих організмів, дослідження випадкових процесів та багато інших задач.

Теорія графів тісно пов'язана з такими розділами математики, як теорія множин, теорія матриць, математична логіка й теорія ймовірностей. У всіх цих розділах графи застосовують для зображення різних математичних об'єктів, і в той же час сама теорія графів широко застосовує апарат споріднених розділів математики [1].

## Граф і його елементи

**Графом** називається пара  $G = (V, E)$ , де  $V$  — непорожня множина,  $E \subset V \times V$ .

Елементи  $v_i \in V$  називаються **вершинами** (vertices), а елементи  $e_j \in E$  — **ребрами** (edges).

Якщо множина вершин графу скінчена, то він називається **скінченним графом**. У математиці розглядаються й нескінченні графи, але ми займатися ними не будемо, оскільки в практичних застосуваннях вони трапляються рідко.

Вершини  $u$  і  $v$  графа є **суміжними**, якщо  $(u, v) \in E$ , та є **несуміжними**, якщо  $(u, v) \notin E$ . Якщо  $e_k = (v_i, v_j) \in E$ , то вершини  $v_i$ , і  $v_j$  називають його кінцями. Кажуть, що ребро  $e_k$  з'єднує вершини  $v_i$ , і  $v_j$ .

Для **неорієнтованих графів** пари вершин невпорядковані, так що  $e_k = (v_i, v_j) = (v_j, v_i)$ .

Для **орієнтованих графів** (орграфів) пари вершин  $(v_i, v_j)$  — впорядковані, причому  $v_i$  та  $v_j$  означають відповідно початкову й кінцеву вершини дуги  $e_k$ .

**Петля** при вершині  $v_i$  в орієнтованих та неорієнтованих графах зображується невпорядкованою парою  $(v_i, v_i)$ .

Вершина  $v$  і ребро  $e$  є **інцидентними**, якщо  $e = (u, v)$ , тобто, якщо  $v$  є кінцем ребра, і **неінцидентними** в інакшому випадку. На відміну від суміжності, яка є відношенням між однорідними об'єктами (вершинами), інцидентність — це відношення між різномірними об'єктами (вершинами і ребрами).

Кожне ребро  $e_k \in E$  сполучає пару вершин  $v_i, v_j \in V$ , які є його **кінцями** (**країовими вершинами**). Ребро, крайовими вершинами якого є одна й та ж вершина, називається **петлею**. Ребра з одинаковими крайовими вершинами є паралельними і називаються **кратними**.

Формально розглядаючи, граф — це математична структура  $G = \langle V; R \rangle$ , де  $V = V$ , а  $R$  — бінарне відношення, яке задає суміжність вершин  $v_i \in V$ , тобто  $u R v$  фактично позначає ребро  $(u, v)$ .

## Задання графів

Множини  $V, E$  зручно задавати як множину номерів вершин  $V_G$  і множину пар номерів суміжних вершин  $E_G$ . Наприклад, певний  $(5, 6)$ -граф задається як

$$V_G = \{1, 2, 3, 4, 5\}, E_G = \{\{1, 1\}, \{1, 2\}, \{1, 3\}, \{2, 5\}, \{3, 4\}, \{3, 5\}\}.$$

Часто зустрічається задання графа квадратною **матрицею суміжності**, у клітинці  $i, j$  якої стоїть 1, якщо вершини  $v_i, v_j$  суміжні, і 0 — якщо несуміжні. Для даного прикладу матриця суміжності виглядає так:

$$E = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

За матрицею добре видно, що вершини 1 і 2 суміжні, а вершини 1 і 4 — несуміжні. Матриця суміжності неоріентованого графу завжди симетрична. Неоріентованим ребрам відповідають пари ненульових елементів, симетричних відносно головної діагоналі матриці, дугам — ненульові елементи матриці, а петлям — ненульові елементи головної діагоналі. У стовпчиках і рядках, що відповідають ізольованим вершинам, усі елементи дорівнюють нулю. Елементи матриці простого графу дорівнюють 0 або 1, причому всі елементи головної діагоналі нульові.

Розглядаючи інцидентність вершин і ребер  $(p, q)$ -графу, можна зобразити його **матрицею інцидентності** розміром  $p \times q$ , рядки якої відповідають вершинам, а стовпчики — ребрам.

Для неоріентованого графу елементи цієї матриці визначаються за наступним правилом:  $ij$ -елемент дорівнює 1, якщо вершина  $v_i$  інцидентна ребру  $e_j$ , і дорівнює нулю, якщо  $v_i$  та  $e_j$  не інцидентні. Наприклад, матриця інцидентності графу, наведеного вище, має вигляд:

$$A = \left| \begin{array}{ccccccc|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 5 \end{array} \right|$$

Кожний стовпчик матриці інцидентності містить обов'язково два одиничних елементи. Кількість одиниць у рядку дорівнює степеню відповідної вершини. Нульовий рядок відповідає ізольованій вершині, а нульовий стовпчик — петлі. Слід мати на увазі, що нульовий стовпчик матриці інцидентності лише вказує на наявність петлі, але не містить відомостей про те, з якою вершиною ця петля пов'язана. Тому у випадку, коли важливо ідентифікувати петлю, використовують матрицю суміжності [1].

Графи зручно зображати у вигляді рисунків. Наприклад, на рис. 5 показаний  $(5, 7)$ -граф, заданий вище.

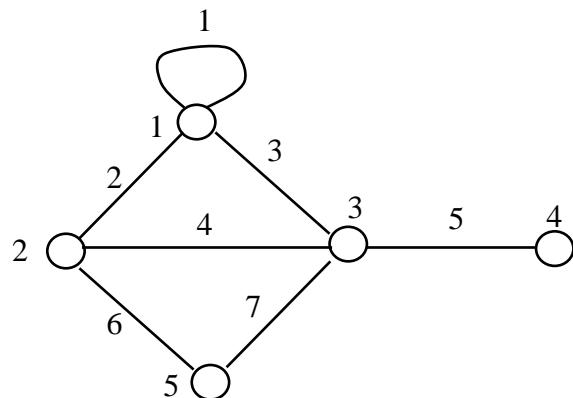


Рис. 5. Граф

З рисунку графу одразу видно, що, наприклад, вершини 1 і 2 суміжні, а вершини 1 і 4 — несуміжні, дуга 1 є петлею, вершина 4 — ізольована, а також те, що вершина 1 інцидентна ребру (1, 3).

## Приклади графів

Граф — це зручна модель для зображення різноманітних відношень, які виникають у людській практиці. Наприклад, нехай маємо структуру  $\langle B; P \rangle$ , яка розглядалась при вивченні родинних відношень між учнями. Тобто, в ній  $A \in B$  є братом  $B$  і  $B \in B$  є братом  $A$ , а  $B \in B$  не є братом  $A$ . Тоді ця структура виражається як граф на рис. 6.

При цьому відношення між елементами  $A, B, B, \Gamma$  є симетричними або комутативними: якщо  $A$  брат  $B$ , то  $B$  — брат  $A$ , що зумовлює використання неорієнтованих ребер.

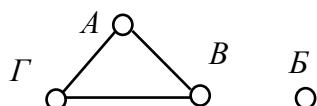


Рис. 6. Граф структури

Ще один спосіб утворення графів з геометричних об'єктів ілюструє рис. 7. На рис. 7,а показано п'ять кіл на площині, що характеризують деякі множини та їх перетини, а праворуч — граф, в якому кожна вершина відповідає одному з цих кіл. У цьому графі дві вершини з'єднані ребром в тому і тільки тому випадку, коли відповідні кола перетинаються. Такі графи називають **графами перетинів**. Взагалі, для будь-якого сімейства множин  $\{S_1, \dots, S_n\}$  можна побудувати граф перетинів з множиною вершин  $\{1, \dots, n\}$ , в якому ребро  $(i, j)$  є тоді і тільки тоді, коли  $i \neq j$  і  $S_i \cap S_j \neq \emptyset$ .

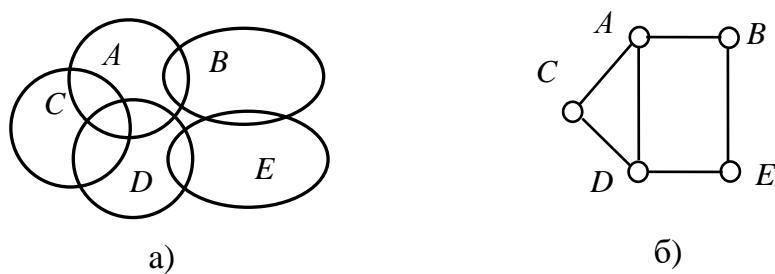


Рис.7

Множину, на якій задано відношення порядку, традиційно зображають графом, який називають **діаграмою Хасе**. Такий граф для прикладу з множиною програмних модулів  $\{a, b, c, d, e, f\}$  і з множиною відношень

$$\{a \prec a, a \prec c, b \prec b, b \prec c, c \prec c, c \prec d, d \prec d, d \prec e, d \prec f, e \prec e, f \prec f\}$$

показано на рис. 8, а. Традиційно в діаграмах Хасе вершини, що стоять у рисунку нижче, передують за відношенням « $\prec$ » тим вершинам, що стоять вище.

Прикладу ґратки з множиною відношень

$$\{a \prec a, a \prec b, b \prec b, b \prec d, a \prec c, c \prec c, c \prec e, d \prec d, d \prec f, e \prec e, e \prec f, f \prec f\}$$

відповідає граф на рис. 8, б.

Нехай маємо структуру ґратки  $\langle B; \prec \rangle$ ,  $B = \{A, B, C, D, E, F\}$ , причому  $A \prec B$ ,  $B \prec C$  та  $C \prec D$ . Наприклад,  $A \prec B$  означає, що учень  $B$  вищий за зростом, ніж учень  $A$ . Тоді така ґратка є лінійною і зображується графом на рис. 8, в.

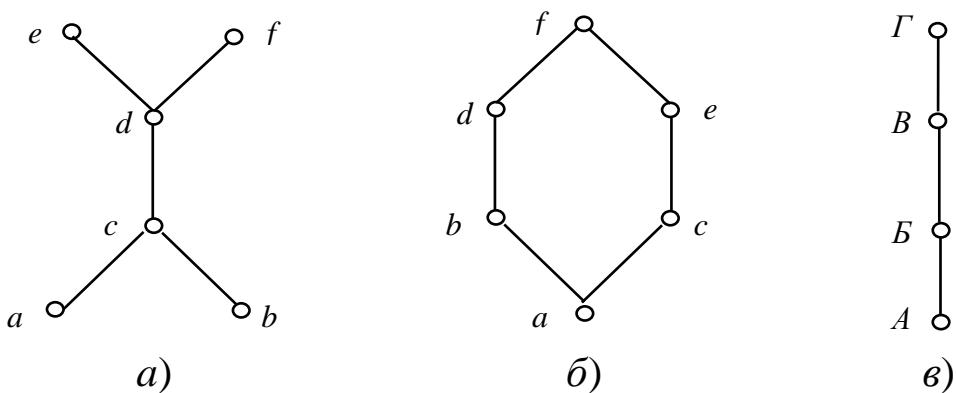


Рис. 8. Діаграми Хасе

## Орієнтовані графи

**Орграфом (орієнтованим графом)** називається пара  $G = (V, E)$ ,  $E \subset V \times V$ , причому ребра  $(u, v) \in E$  є орієнтованими,  $u$  — це початкова, а  $v$  — кінцева вершини ребра, тобто з  $u$  виходить ребро, а в  $v$  — заходить. Тоді відношення суміжності не є комутативним, тобто  $(u, v) \neq (v, u)$ .

Для задання орграфу також користуються матрицею суміжності, у клітинці  $i, j$  якої стоїть 1, якщо вершини  $v_i, v_j$  задають ребро  $(v_i, v_j)$ , і 0 — у

інших випадках. Тобто 1, якщо ребро виходить з  $j$ -ї вершини. Нехай маємо матрицю суміжності:

$$E = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Їй відповідає орграф на рис. 7. Щоб позначити напрямок зв'язку між вершинами орграфу, відповідне ребро позначається стрілкою. Орієнтоване таким чином ребро називають **дугою** [1].

Матриця суміжності орграфу — у загальному випадку несиметрична.

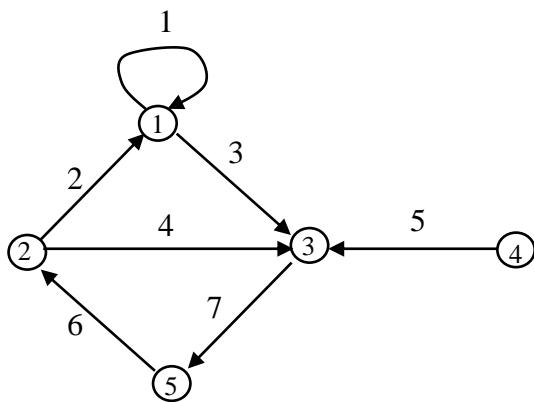


Рис. 9. Орграф

При розгляді орграфів розрізняють **додатну інцидентність** (дуга виходить з вершини) та **від'ємну інцидентність** (дуга заходить у вершину). Розглядаючи інцидентність вершин і ребер  $(p, q)$ -орграфу, можна зобразити його **матрицею інцидентності** розміром  $p \times q$ , рядки якої відповідають вершинам, а стовпчики — ребрам. Для орієнтованого графу елементи цієї матриці визначаються за наступним правилом:  $ij$ -елемент дорівнює 1, якщо  $v_i$  початкова вершина дуги  $e_j$ , і дорівнює -1, якщо  $v_i$  — кінцева вершина дуги і дорівнює нулю, якщо  $v_i$  та  $e_j$  не інцидентні.

Наприклад, матриця інцидентності графу, наведеного на рис. 9, має вигляд:

$$A = \left| \begin{array}{ccccccc|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & -1 & -1 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 5 \end{array} \right|$$

Кожний стовпчик матриці інцидентності містить обов'язково два одиничних елементи. Вони завжди мають різні знаки і дорівнюють відповідно 1 та -1. Кількість одиниць у рядку дорівнює степеню відповідної вершини, причому кількість додатних одиниць визначає додатний степінь, а кількість від'ємних одиниць — від'ємний степінь. Нульовий рядок відповідає ізольованій вершині, а нульовий стовпчик — петлі [1].

Якщо пара вершин з'єднується двома чи більшою кількістю дуг, то такі дуги називають **паралельними**. При цьому, якщо дві дуги відносно даної вершини мають одинаковий напрям, вони є **строго паралельними**, а якщо неоднаковий напрям — **нестрого паралельними**.

Власне, у випадку, якщо паралельні дуги відображують зв'язок між вершинами в обох напрямках, то вони рівноцінні неорієнтованому зв'язку і їх можна замінити на ребро. І навпаки, будь-яке ребро в графі можна замінити на пару нестрого паралельних дуг. Після такої заміни отримуємо граф, який містить і ребра, і дуги. Такий граф називається **змішаним** (рис. 10) [1].

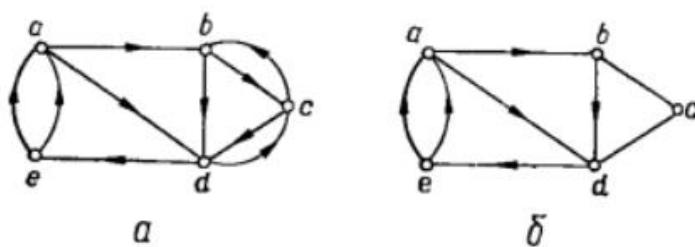


Рис. 10. Орієнтований граф з кратними дугами(а) і змішаний граф (б)

Якщо напрямки всіх дуг орграфу змінити на протилежні, отримаємо орграф, **обернений** до початкового. Якщо напрямки дуг орграфу не враховуються і кожна дуга розглядається як неоріентоване ребро, то він називається **співвіднесеним** (неоріентованим) графом до даного [1].

Оріентовані графи застосовуються, коли зв'язки між об'єктами характеризуються визначеною орієнтацією. Наприклад, на деяких вулицях є лише односторонній автомобільний рух, у з'єднувальних дротах електричного ланцюга задаються додатні напрями струмів, відношення між людьми можуть визначатися підлеглістю чи старшинством. Оріентовані зв'язки також характеризують перехід системи з одного стану в інший. Наприклад, результати зустрічей між командами в спортивних змаганнях, різні відношення між числами (нерівність, подільність) [1].

## Типи графів і їх параметри

Кількість вершин графа  $n = |V|$  називається його **порядком**. Якщо у графа  $m = |E|$  вершин, то  $G$  називають **(n, m)-графом**. Отже,  $i = 1, 2, \dots, n$ ,  $j = 1, 2, \dots, m$ .

Нехай  $V = \{v_1, v_2, \dots, v_p\}$  і  $E = \{e_1, e_2, \dots, e_q\}$  — відповідно множини вершин і ребер  $(p, q)$ -графу. Кожне ребро  $e_k \in E$  сполучає пару вершин  $v_i, v_j \in V$ , які є його **кінцями (крайовими вершинами)**. Для оріентованого ребра (дуги) розрізняють **початкову вершину**, з якої дуга виходить, і **кінцеву вершину**, у яку дуга заходить.

Ребро, крайовими вершинами якого є одна й та сама вершина, називається **петлею**. Ребра з одинаковими крайовими вершинами є паралельними і називаються **кратними**. У загальному випадку граф може містити й **ізольовані вершини**, які не є кінцями ребер і не сполучені ні між собою, ні з іншими вершинами. Наприклад, для  $(5, 6)$ -графа на рис. 9, а  $V = \{v_1, v_2, v_3, v_4, v_5\}$ ;  $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ ; ребра  $e_2$  та  $e_3$  паралельні, ребро  $e_6$  є петлею, а  $v_4$  — ізольована вершина [1].

Кількість ребер, пов'язаних з вершиною  $v_i$  (петля враховується двічі), називають **степенем вершини** і позначають через  $\delta(v_i)$  або  $\deg(v_i)$ . Так, для графу на рис. 11, а  $\delta(v_1) = 1$ ,  $\delta(v_2) = 4$  тощо. Очевидно, степінь ізольованої вершини дорівнює нулю ( $\delta(v_4) = 0$ ). Вершина степеня одиниці називається **кінцевою** або **висячою вершиною** ( $\delta(v_1) = 1$ ).

Легко показати, що в будь-якому графі сума степенів всіх вершин дорівнює подвоєній кількості ребер, а кількість вершин непарного степеня завжди парна. В орграфі розрізняють додатні  $\delta^+(v_i)$  та від'ємні  $\delta^-(v_i)$  степені вершин, які дорівнюють відповідно кількості вихідних із  $v_i$  та входних у  $v_i$  дуг. Наприклад, для вершини  $d$  орграфу (див. рис. 8, а) маємо  $\delta^+(d) = 2$  і  $\delta^-(d) = 3$ . Очевидно, суми додатних і від'ємних степенів всіх вершин орграфу рівні між собою і дорівнюють також кількості всіх дуг [1].

Граф без петель і кратних ребер називають **простим** або **звичайним**. Граф без петель, але з кратними ребрами називається **мультиграфом**. Найбільш загальний випадок графу, коли допускаються петлі й кратні ребра, називають **псевдографом**. Так, граф на рис. 13, б — це мультиграф, а на рис. 11, а — псевдограф.

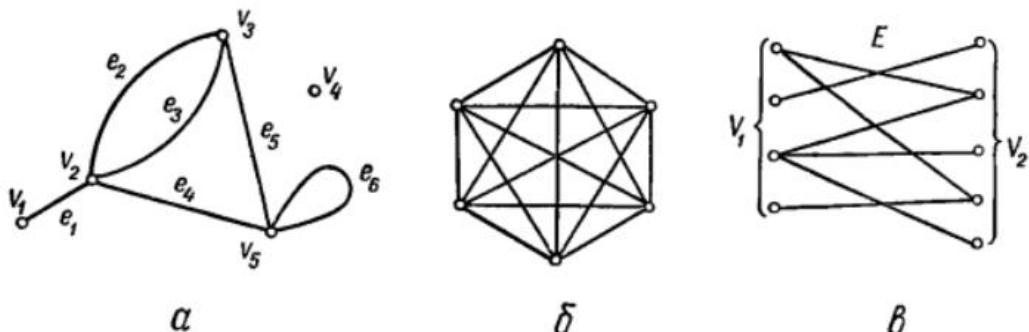


Рис. 11. Типи графів:

а — псевдограф; б — повний граф (шестикутник); в — дводольний граф (біграф).

Якщо граф не має ребер ( $E = \emptyset$ ), то всі його вершини ізольовані ( $V \neq \emptyset$ ), і він називається **порожнім** або **нуль-графом**.

Простий граф, у якому будь-які дві вершини сполучені ребром, називається **повним**. На рис. 11, б наведено приклад повного графу з шістьма вершинами.

Якщо множина вершин  $V$  простого графа допускає таке розбиття на дві неперетинні підмножини  $V_1$  та  $V_2$  ( $V_1 \cap V_2 = \emptyset$ ), що не існує ребер, які сполучають вершини однієї й тієї ж підмножини, то він називається **дводольним** або **біграфом** (рис. 11, в). Орграф вважається **простим**, якщо він не має строго паралельних дуг і петель [1].

Граф, степені всіх вершин якого однакові й дорівнюють  $r$ , називається **однорідним (регулярним)**  $r$ -го **степеня**. Повний граф з  $n$  вершинами завжди однорідний степеня  $n - 1$ , а порожній граф — однорідний степеня 0. Граф третього степеня називають **кубічним**. Він має ряд цікавих властивостей і, зокрема, завжди має парну кількість вершин [1].

## Зважені графи

Подальше узагальнення відображення зв'язків між об'єктами за допомогою графів полягає в приписуванні ребрам і дугам деяких кількісних значень, якісних ознак чи характерних властивостей, що називаються **вагами**. У найпростішому випадку це може бути порядкова нумерація ребер і дуг, яка вказує на черговість при їх розгляді (пріоритет або ієрархія). Вага ребра чи дуги може означати довжину (шляху сполучення), пропускну спроможність (лінії зв'язку), напругу чи струм (електричні кола), кількість набраних очків (турніри), валентність зв'язків (хімічні формули), кількість смуг руху (автомобільні дороги), колір провідника (монтажна схема електронного пристрою), характер відношень між людьми (син, брат, батько, підлеглий, вчитель) тощо [1].

Вагу можна приписувати не лише ребрам і дугам, але й вершинам. Наприклад, вершини, що відповідають населеним пунктам на карті автомобільних доріг, можуть характеризуватися кількістю місць у

кемпінгах, пропускою спроможністю станцій техобслуговування. Взагалі, вага вершини означає будь-яку характеристику відповідного об'єкта (атомна вага елемента в структурній формулі, колір зображеного вершиною предмета, вік людини, тип оператора алгоритму тощо) [1].

Особливого значення для моделювання фізичних систем набули зважені орієнтовані графи, названі **графами потоків сигналів (даних)** або **сигнальними графами**. Вершини сигнального графу ототожнюються з деякими змінними, що характеризують стан системи, а вага кожної вершини означає функцію часу або деякі величини, що характеризують передавання сигналу від однієї вершини до іншої. Вагою дуги найчастіше є ємність буферної пам'яті. Сигнальні графи знаходять широке застосування в теорії ланцюгів і систем, а також в багатьох інших галузях науки і техніки [1].

Для зваженого графу, що не містить кратних ребер, можна узагальнити матрицю суміжності так, що кожний її ненульовий елемент дорівнює вазі відповідного ребра чи дуги. І навпаки, будь-яка квадратна матриця  $n$ -го порядку може бути зображена орграфом з  $n$  вершинами, дуги якого сполучають суміжні вершини та мають ваги, що дорівнюють відповідним елементам матриці. Якщо матриця симетрична, то вона зображується неорієнтованим графом [1].

## Ізоморфізм

На рис.4 та рис. 12 зображені графи, які з геометричної точки зору цілковито різні (перетин ребер, якщо він не відмічений точкою, не є вершиною). Але по суті вони відрізняються лише зображенням, а відношення інцидентності (при відповідному позначенні вершин і ребер) для них одинакові. Графи, для яких зберігається відношення інцидентності, називаються **ізоморфними** [1].

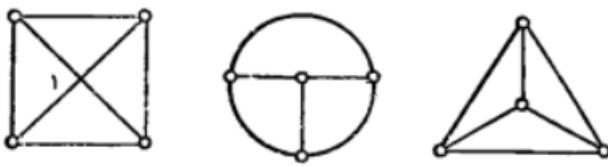


Рис. 12. Ізоморфні графи

Зрозуміло, що матриця інцидентності визначає граф без петель з точністю до ізоморфізму. Зазвичай на її основі можна зобразити різні в геометричному відношенні, але ізоморфні між собою графи, кожний з яких називають *геометричною реалізацією*. Графи які мають однакові зображення і відрізняються лише нумерацією вершин і ребер, не будучи тотожними, є ізоморфними[1]. Отже, граф, який є ізоморфний даному, одержують перестановкою стовпців і/або рядків матриці інцидентності. І навпаки, щоб довести ізоморфність двох графів, слід в одній з матриць інцидентності так переставити рядки і стовпці, щоб ця матриця дорівнювала іншій.

Якщо суттєві властивості графу не пов'язані зі способом його зображення на площині чи нумерацією вершин і ребер, то ізоморфні графи, як правило, не розрізняють між собою [1].

## Маршрути

Нерідко задачі на графах вимагають виділення різних маршрутів, що мають певні властивості й характеристики. **Маршрут** завдовжки  $t$  визначається як послідовність  $t$  ребер графу (не обов'язково різних) таких, що у кожних двох сусідніх ребер одна загальна вершина. Маршрут проходить і через всі вершини, інцидентні його ребрам.

Прикладами маршрутів на графі рис. 11, а можуть слугувати послідовності  $(e_1, e_3, e_2, e_3, e_5)$ ,  $(e_5, e_6, e_4, e_4)$ . Перший маршрут проходить через послідовність вершин  $(v_1, v_2, v_3, v_2, v_3, v_5)$  і сполучає вершини  $v_1$  та  $v_5$ , а другий — через послідовність вершин  $(v_3, v_5, v_5, v_2, v_5)$  і сполучає вершини

$v_3$  та  $v_5$ . **Замкнений маршрут** приводить в ту саму вершину, з якої він почався [1].

Маршрут, у якому всі ребра різні, називається **ланцюгом**, а маршрут, для якого різні всі вершини — **простим ланцюгом**. Замкнений ланцюг називається **циклом**, а замкнений простий ланцюг — **простим циклом**. Так, на графі рис. 11, а  $(e_2, e_5, e_6)$  — ланцюг,  $(e_1, e_2, e_5)$  — простий ланцюг,  $(e_2, e_3, e_4, e_5)$  — цикл,  $(e_2, e_4, e_5)$  — простий цикл [1].

Цикл, який містить всі ребра графу, називається **ейлеровим циклом**, а граф, у якому є такий цикл, називається ейлеровим графом. Тому задача про кенігсберзькі мости зводиться до з'ясування існування такого циклу. Критерій існування ейлерового циклу дуже простий: необхідно, щоб степені всіх вершин були парними.

Простий цикл, який проходить через всі вершини графу, називають **гамільтоновим**. Але для виявлення наявності гамільтонових циклів жодного загального правила не знайдено [1].

Орієнтовані маршрути на орграфі визначаються аналогічно з тією різницею, що початкова вершина кожної наступної дуги маршруту має співпадати з кінцевою вершиною попередньої дуги. Інакше кажучи, рух по маршруту допускається лише в напрямках, вказаних стрілками.

Маршрут, що не містить дуг, які повторюються, називається **шляхом**, а той, що не містить повторюваних вершин — **простим шляхом**. Замкнений шлях називається **контуром**, а простий замкнений шлях — **простим контуром**.

Граф (орграф) називається **циклічним (контурним)**, якщо він містить хоча б один цикл (контур), в іншому випадку він називається **ациклічним (безконтурним)** [1].

Поняття ланцюга й циклу застосовані й до орієнтованих графів. При цьому напрями дуг не враховуються, тобто по суті замість орграфа розглядають неорієнтований співвіднесений йому граф [1].

## Частини графа

Граф  $G' = (V', E')$  є **частиною графу**  $G = (V, E)$ , якщо  $V' \subset V$  і  $E' \subset E$ , тобто граф містить всі вершини та ребра будь-якої його частини. Частина, яка, поряд з деякою підмножиною ребер графу, містить і всі інцидентні їм вершини, називається **підграфом**. Частина, яка поряд з деякою підмножиною ребер графу, містить всі вершини графу ( $V' = V$ ,  $E' \subset E$ ), називається **суграфом**. Розглянуті графи показані на рис. 13 [1].

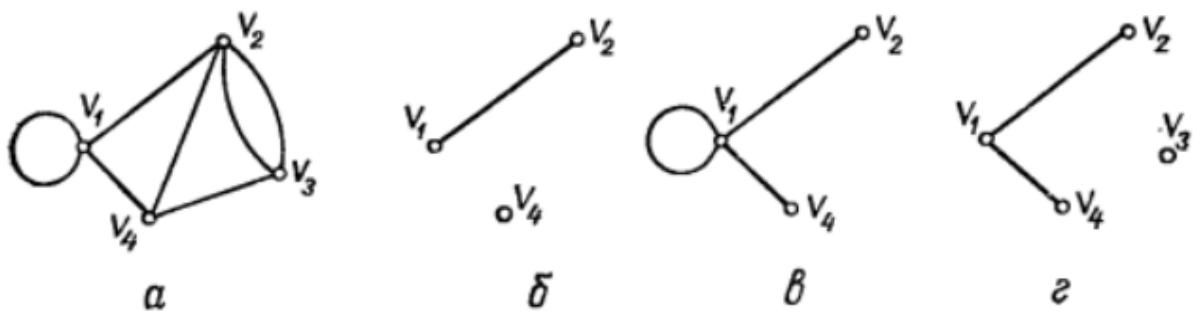


Рис. 13. Граф і його частини:  
а — граф; б — частина графу; в — підграф; г — суграф

Сукупність всіх ребер графу, що не належать його підграфу (разом з інцидентними вершинами), утворює **доповнення підграфу**. Кажуть, що підграфи  $G' = (V', E')$  і  $G'' = (V'', E'')$  **розділені ребрами**, якщо вони не мають спільних ребер ( $E' \cap E'' = \emptyset$ ) і **розділені вершинами**, якщо в них немає спільних вершин ( $V' \cap V'' = \emptyset$ ) [1].

## Зв'язність (connectivity)

Дві вершини графа називають **зв'язаними**, якщо існує маршрут, що поєднує ці вершини [1]. **Зв'язний граф** — це граф, у якому будь-яка пара вершин зв'язана. Очевидно, у зв'язному графі між будь-якими двома вершинами існує простий ланцюг, оскільки з маршруту, що їх сполучає, завжди можна видалити циклічну ділянку, яка проходить через деяку вершину більше ніж один раз (рис. 14, де маршрут між вершинами  $v_i$  та  $v_j$  зображений жирними лініями) [1].

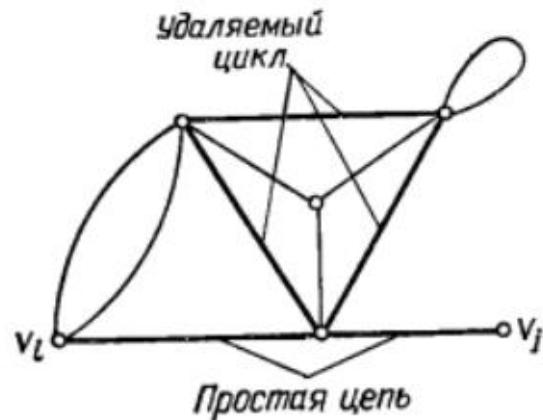


Рис. 14. Зв'язний граф

Якщо граф не зв'язний, то множину його вершин можна єдиним чином розділити на неперетинні підмножини, кожна з яких містить всі сполучені між собою вершини і разом з інцидентними їм ребрами утворює зв'язний підграф. Таким чином, незв'язний граф є сукупністю окремих частин (підграфів), що називаються **компонентами**. На рис. 15 показаний підграф, що складається з трьох компонент (ізольована вершина вважається компонентою) [1].

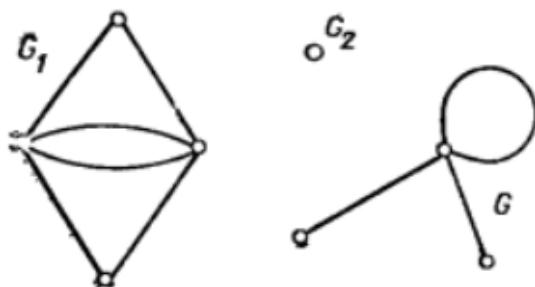


Рис. 15. Незв'язний граф, що складається з трьох компонент  
 $(G_1, G_2, G_3)$

Часто відношення зв'язності ускладнюється додатковими умовами. Граф називають **циклічно зв'язним**, якщо будь-які дві різні вершини містяться в циклі (наприклад, граф на рис. 13, а циклічно зв'язний, а граф на рис. 14 — ні, оскільки вершина  $v_j$  не міститься в жодному циклі з

іншими вершинами). Граф називають ***k*-зв'язним**, якщо будь-яка пара різних вершин сполучена, принаймні *k* ланцюгами, які не мають спільних вершин (крім початкової та кінцевої). Так, граф на рис. 13, а двозв'язний, а на рис. 14 — однозв'язний [1].

**Теорема.** Кількість компонент зв'язності *q* графа з числом вершин *p* і числом ребер *k* оцінюється як

$$p - k \leq q \leq \frac{(p - k)(p - k + 1)}{2}.$$

Зв'язність орієнтованих графів визначається так само, як і для неорієнтованих, тобто, без урахування напрямів дуг. Специфічним для орграфу або змішаного графу є поняття сильної зв'язності. Орграф називають **сильно зв'язним**, якщо для будь-якої пари його вершин  $v_i$  та  $v_j$  існує шлях з  $v_i$  у  $v_j$  та з  $v_j$  у  $v_i$ . Наприклад, граф на рис. 10, а сильно зв'язний.

Граф, що зображує план міста з одностороннім рухом деякими вулицями, повинний бути сильно зв'язним, оскільки в інакшому разі знайшлися б вершини (площі та перехрестя), між якими не можна було б проїхати містом без порушення правил дорожнього руху [1].

## Подільність

Зв'язний граф можна розділити на незв'язні підграфи шляхом видалення з нього деяких вершин і ребер (при видаленні вершин видаляються всі інцидентні їм ребра, а при видаленні ребер вершини зберігаються). Якщо існує така вершина, видалення якої перетворює зв'язний граф (чи компоненту незв'язного графу) в незв'язний, то вона називається **точкою зчленування** (рис. 16, а). Ребро з такими ж властивостями називається **мостом** (рис. 166, б). Зрозуміло, що при наявності моста в графі є, принаймні, дві точки зчленування [1].

Якщо граф зв'язний і не має точок зчленування, то він є **неподільним** (наприклад, граф на рис. 13, а). Якщо ж у графі є хоча б одна точка зчленування, то він є подільним і називається **сепарабельним**. Він

розділяється на блоки, кожний з яких є максимальним неподільним підграфом (на рис. 16, в показані блоки  $B_1, B_2, B_3$  графа рис. 16, б) [1].

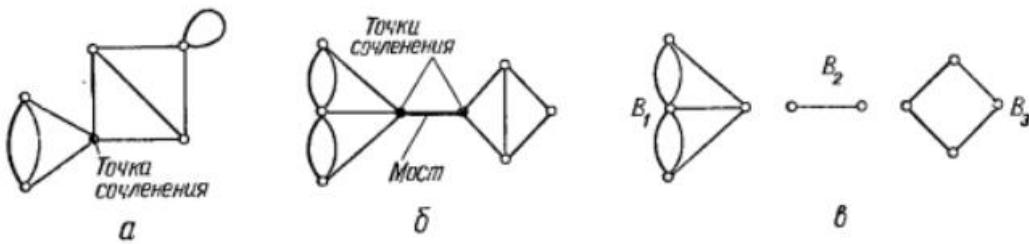


Рис. 16. Подільні графи:

а — з точкою зчленування; б — з мостом; в — блоки  $B_1 — B_3$  графу з мостом

Кожне ребро графа, як і кожна вершина, за винятком точок зчленування, належать лише одному з його блоків. Більш того, тільки одному блоку належить і кожний простий цикл. Звідси випливає, що сукупність блоків графа — це розбиття множин ребер і простих циклів на неперетинні підмножини [1].

У ряді застосувань теорії графів блоки можна розглядати як компоненти. Це зазвичай припустимо, коли сполучення блоків за допомогою точки зчленування несуттєві або коли суттєві властивості графу пов'язані лише з його простими циклами, тобто, контурами. У таких випадках можна розглядати незв'язний граф як зв'язний подільний граф, який утворюється шляхом такого об'єднання компонент, щоб кожна з них була блоком. Це завжди можна зробити, об'єднавши, наприклад, по одній вершині кожного блоку в точку зчленування. Подібні операції використовуються при розгляді графів електричних кіл [1].

## Дерева і ліс

Особливий інтерес становлять зв'язні ациклічні графи, що називаються **деревами**. Дерево на множині  $p$  вершин завжди містить  $q = p - 1$  ребер, тобто мінімальну кількість ребер, необхідну для того, щоб граф був зв'язним. Дійсно, дві вершини сполучаються одним ребром, і для

сполучення кожної наступної вершини з попередніми необхідне ребро, відтак, для сполучення  $p$  вершин необхідно ѹ достатньо  $p - 1$  ребер [1].

При додаванні в дерево ребра утворюється цикл, а при видаленні хоча б одного ребра дерево розпадається на компоненти, кожна з яких також є деревом або ізольованою вершиною. Незв'язний граф, компоненти якого є деревами, називається **лісом** (ліс із  $k$  дерев, що містить  $p$  вершин, має в точності  $p - k$  ребер). Сказане ілюструється на прикладі дерева (рис. 17, а), яке перетворюється в циклічний граф додаванням ребра (рис. 177, б) і розпадається на ліс із двох дерев  $T_1$  і  $T_2$  при видаленні ребра  $e$  (рис. 17, в) [1].

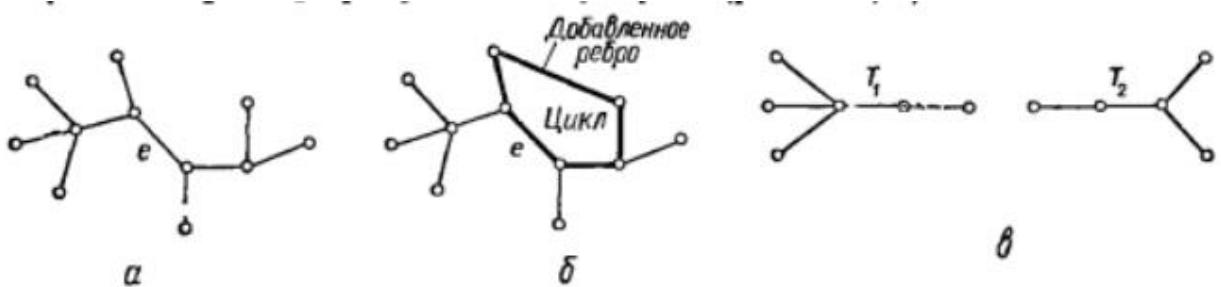


Рис. 17. Дерево (а), утворення циклу при введенні додаткового ребра (б) та ліс, що утворюється після видалення ребра  $e$  (в)

Зазвичай дерева вважаються суттєво різними, якщо вони не ізоморфні. На рис. 18 показані всі можливі різні дерева з шістьма вершинами. Зі збільшенням кількості вершин кількість різних дерев різко зростає (наприклад, при  $p = 20$  їх налічується біля мільйона). Серед різних дерев виділяються два важливих окремих випадки: **послідовне дерево**, що являє собою простий ланцюг, і **зіркове дерево**, у якому одна з вершин (центр) суміжна з усіма іншими вершинами [1].

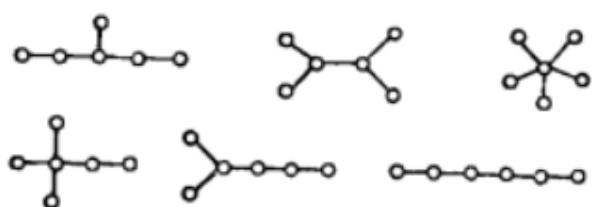


Рис. 18. Суттєво різні дерева з шістьма вершинами



Рис. 19. Прадерево з коренем  $v_0$

Розглядаються також дерева з орієнтованими ребрами (дугами). Орієнтоване дерево називається **прадеревом з коренем  $v_0$** , якщо існує шлях між вершиною  $v_0$  та будь-якою іншою його вершиною (рис. 19). Зрозуміло, що прадерево має єдиний корінь [1].

Досі розглядалися дерева як мінімальні зв'язні графи на множині  $p$  вершин. Важливе значення має й інша точка зору, коли дерева або ліс є частинами деякого графу, тобто утворюються з його ребер. Будь-яка зв'язна сукупність ребер, що не містить контурів, разом з інцидентними їх вершинами утворює **дерево графа** (рис. 20, а). Якщо таке дерево є суграфом (містить всі вершини графу), то воно називається **покривним деревом** або **кістяком** (рис. 20, б). Оскільки петля є найпростішим циклом, що складається з єдиного ребра, то вона не може входити до складу будь-якого дерева графу [1].

Ребра графу, які належать його дереву, називають **гілками**. Якщо дерево покриває граф, то множина ребер графу розбивається на дві підмножини: підмножину гілок та підмножину ребер **доповнення дерева**, що називаються **хордами**. При цьому зв'язний  $(p, q)$ -граф містить  $v = p - 1$  гілок та  $\sigma = q - p + 1$  хорд. Якщо граф незв'язний, то сукупність кістяків  $k$  його компонент утворює **покривний ліс**. У цьому разі  $v = p - k$  і  $\sigma = q - p + k$ .

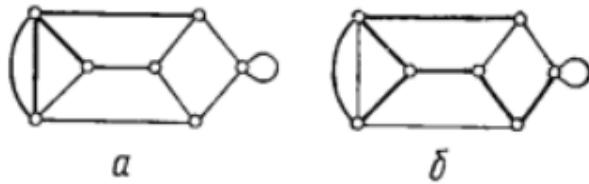


Рис. 20. Дерево як частина графа (виділено жирними лініями):  
а — дерево; б — кістяк (покривне дерево)

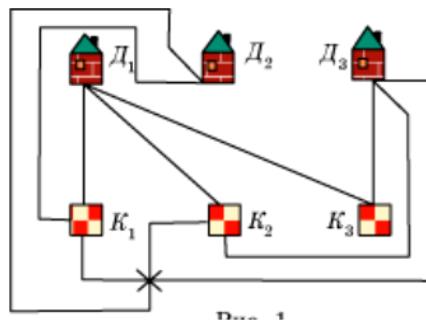
Дерева грають важливу роль у різних прикладних задачах, коли, наприклад, мова йде про зв'язок яких-небудь об'єктів мінімальною кількістю каналів (ліній зв'язку, доріг, комунікацій) з певними властивостями. За допомогою дерева визначається система координат при моделюванні ланцюгів і систем різної фізичної природи. Дерева використовуються в ролі моделей при розгляді ієрархічних систем об'єктів, структурних формул органічних сполук тощо [1].

### Планарність

Граф називають **пласким (планарним)**, якщо існує ізоморфний йому граф (геометрична реалізація), який може бути зображенний на площині без перетину ребер. Наприклад, хоча в одному з графів на рис. 4 ребра перетинаються, ізоморфні йому не мають перетинів, відповідно, він плаский [1].

На рис. 21 показані два непласких графи, що грають фундаментальну роль у теорії планарності та називаються **графами Понтрягіна — Куратовського**. Повний п'ятикутник (рис. 21, а) є простим непласким графом з мінімальною кількістю вершин (повний граф з чотирма вершинами — плаский, а видалення з п'ятикутника хоча б одного ребра також перетворює його в плаский граф). Дводольний граф (рис. 21, б) є моделлю відомої задачі про три будинки та три колодязі: чи можна прокласти від будинків до кожного колодязя дороги так, щоб вони не перетиналися (сусіди, що ворогують, повинні мати можливість

користуватися всіма колодязями, але не хочуть зустрічатися на дорогах)



[1]?

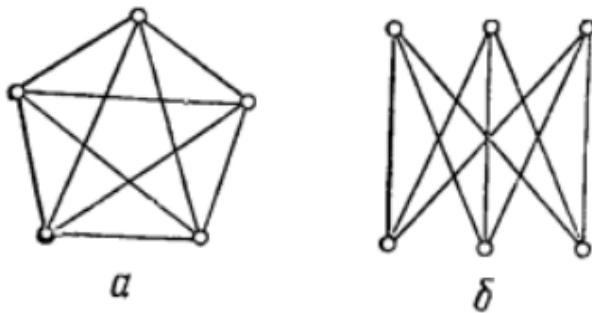


Рис. 21. Графи Понтрягіна — Куратовського:  
а — повний п'ятикутник; б — дводольний граф.

Властивості планарності не порушуються, якщо деяке ребро розбити на два введенням нової вершини другого степеня або замінити два ребра, інцидентні вершині другого степеня, на одне ребро, видаливши цю вершину. Два графи називають **гомеоморфними** (ізоморфними з точністю до вершин другого степеня), якщо після видалення з них вершин другого степеня та об'єднання інцидентних цим вершинам ребер, вони виявляються ізоморфними (рис. 22). Очевидно, граф, гомеоморфний пласкому графу, також плаский [1].

**Теорема:** граф є пласким тоді й тільки тоді, коли він не містить підграфу, гомеоморфного до одного з графів Потрягіна — Куратовського.

Планарність є суттєвою властивістю графів, які моделюють комунікації та зв'язки між об'єктами на площині (дороги між населеними пунктами, водопровідні та газопровідні мережі, лінії передач

електроенергії, між'єднання на друкованих платах електронних пристрій та кристалах інтегральних схем).

Пласкими графами зображуються різні карти, з якими, зокрема, пов'язана відома **проблема чотирьох кольорів**: чи завжди можна розфарбувати ділянки, на які плаский граф поділяє поверхню, так, щоб жодні дві суміжні ділянки не були забарвлені в одинаковий колір і щоб при цьому було використано не більше як чотири кольори?

**Теорема:** для розфарбування карти на площині, в будь-якому разі, достатньо п'яти кольорів.

Необхідність п'яти кольорів для розфарбування карти поки не доведена. Проблема залишається нерозв'язаною, незважаючи на величезні зусилля багатьох видатних математиків, які штурмують її більш ніж століття. При цьому за допомогою комп'ютерного моделювання доведено необхідність і достатність для великої множини графів.

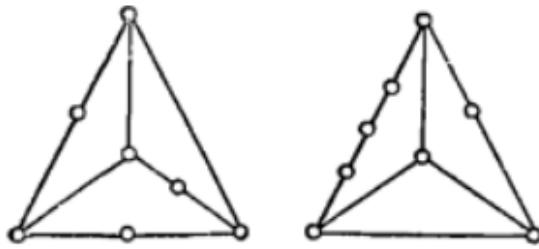


Рис. 22. Гомеоморфні графи

## Графи і відношення

Будь-який орграф  $G(V, E)$  з петлями, але без кратних дуг, задає бінарне відношення  $E$  на множині  $V$  і навпаки. Є повна аналогія між орграфом і бінарним відношенням. Фактично, це один і той же клас об'єктів, тільки описаний різними засобами. Відношення, зокрема, функції є базовим засобом для побудови переважної більшості математичних моделей, що використовуються при вирішенні практичних задач. З іншого боку, графи допускають наочне уявлення у графічному вигляді. Цією обставиною пояснюється широке використання діаграм різного виду, які є

представленням графів або споріднених об'єктів при проектуванні схем і в програмуванні.

Згадаємо види відношень та операції з відношеннями стосовно графів з матрицею суміжності  $E$ , який відповідає відношенню  $R$ .

Граф з петлями на усіх вершинах відповідає рефлексивному відношенню, бо відношення  $R$  на множині  $A$  називається **рефлексивним**, якщо  $\forall a \in A (aRa)$ . Тоді для матриці суміжності  $I \subset E$ .

Навпаки, граф без петель відповідає **антисиметричному** відношенню, для якого  $\forall a \in A (a \bar{R} a)$ . Це означає нульову діагональ у матриці суміжності  $E \cap I = \emptyset$ .

Неоріентований граф відповідає **симетричному** відношенню, тобто,  $\forall a, b \in A (aRb \Rightarrow bRa)$  і  $E = E^T$ .

А орієнтовний граф без нестрого паралельних дуг відповідає **антисиметричному** відношенню, тобто,  $\forall a, b \in A (aRb \wedge bRa \Rightarrow a = b)$ . Унього  $E \cap E^{-1} \subset I$ .

Зміна напрямку всіх дуг відповідає зворотному відношенню. Для такого графу матриця суміжності  $E^{-1}$ .

Повний граф відповідає універсальному відношенню  $U$ .

Рефлексивне, антисиметричне та транзитивне відношення називається **відношенням порядку** (чи просто **порядком**). Йому відповідає граф — діаграма Хасе (за винятком того, що дуги — направлені).

Антисиметричне, антисиметричне та транзитивне відношення називається **відношенням строгого порядку**  $a > b$ .

Відношенню строгого порядку  $R$  відповідає орграф  $G(V, E)$ , в якому  $a > b$  відповідає дуга  $(a, b) \in E$ .

**Теорема.** Якщо відношення  $R$  — це строгое відношення порядку, то орграф  $G(V, E)$  не має контурів.

Доведення — від супротивного. Нехай в  $G$  є контур. Розглянемо довільну дугу  $(a, b)$  у цьому контурі. Тоді маємо  $a > b$ , але з іншого боку, за транзитивністю ланцюжка вершин у контурі маємо  $b > a$ , що протирічить антисиметричності відношення порядку.

Оскільки бінарні відношення визначаються як множини елементів над добутком  $A \times A = A^2$ , то над ними можна виконувати операції об'єднання, перетину і доповнення. Якщо  $R_1, R_2$  — два бінарних відношення, то їх **об'єднання** визначається як:

$$R = R_1 \cup R_2 \Leftrightarrow \forall a, b \in A (a R_1 b \vee a R_2 b).$$

Об'єднанню відношень відповідає об'єднання графів.

Відношення **доповнення** визначається як:

$$\bar{R} \stackrel{\text{def}}{=} \{(a, b) \mid (a, b) \notin R\} \subset A^2.$$

Йому відповідає доповнення графу з матрицею суміжності  $U - E$ .

Нехай  $R_1 \subset A \times C$  — відношення між  $A$  та  $C$ , а  $R_2 \subset C \times B$ , — відношення між  $C$  та  $B$ . **Композицією** двох відношень  $R_1$  та  $R_2$  називається відношення  $R \subset A \times B$  між  $A$  та  $B$  таке, що:

$$R = R_1 \circ R_2 \stackrel{\text{def}}{=} \{(a, b) \mid a \in A \wedge b \in B \wedge \exists c \in C (a R_1 c \wedge c R_2 b)\}.$$

Іншим чином це записується як:

$$a R_1 \circ R_2 b \Leftrightarrow \exists c \in C (a R_1 c \wedge c R_2 b).$$

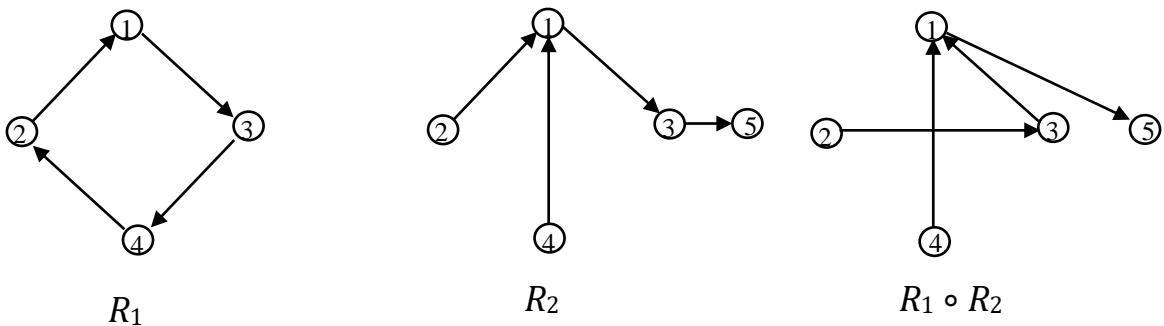


Рис.23

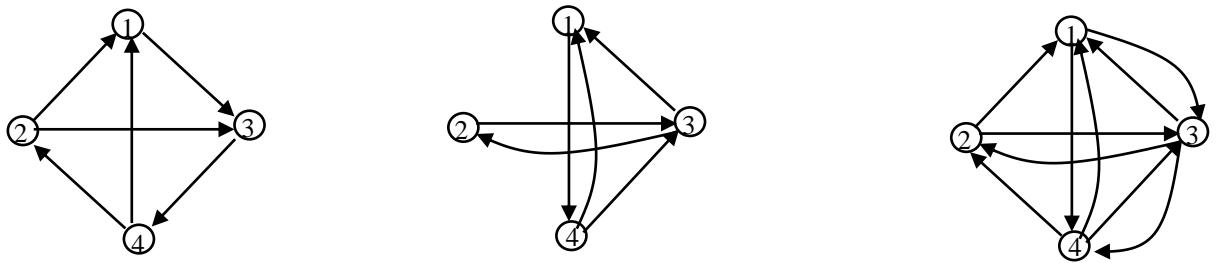
Граф, дуги якого показують, в які вершини веде шлях довжини 1 чи 2, відповідає **транзитивному** відношенню, для якого  $\forall a, b, c \in A (a R b \wedge b R c \Rightarrow a R c)$ .

Тоді такий граф одержується композицією відношення — його другим степенем  $R \cup R \circ R = R \cup R^2$ .

Для транзитивного відношення у матриці суміжності елемент  $e_{i,j}=1$ , якщо з  $i$ -ї вершини у  $j$ -у вершину веде шлях довжиною 1 чи 2. Таку матрицю можна одержати як об'єднання матриці з уміжності та її добутку на себе  $E \cup E \cdot E = E \cup E^2$  за виключенням того, що ненульовий елемент дорівнює кількості маршрутів довжини 2 з однієї вершини у іншу.

Якщо ця інформація не є доречною, то ненульові елементи приводять до одиниці за допомогою **булевого перетворення**:

$$B: X \rightarrow \{0, 1\}; B(x_{i,j}) = 0, \text{ якщо } x_{i,j} = 0, \text{ та } B(x_{i,j}) = 1, \text{ якщо } x_{i,j} \neq 0.$$



$$E = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{vmatrix} \quad E^2 = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \end{vmatrix} \quad E \cup E^2 = \begin{vmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 2 & 0 \end{vmatrix}$$

Рис.24

**Степенем** відношення  $R$  називається його  $n$ -кратна композиція з самим собою і позначається як

$$R^n = \underbrace{R \circ \dots \circ R}_{\tilde{n}}$$

Матрицю суміжності графа відношення  $R^n$  також можна одержати як підведення до степеня матриці графа відношення  $R$ . Елемент  $e_{i,j} \in E^n$  дорівнює кількості маршрутів довжини  $n$ , які ведуть з  $i$ -ї вершини у  $j$ -ту.

## Алгоритми на графах

### Алгоритми перевірки зв'язності

#### Матриця досяжності

**Транзитивне замикання** відношення  $P$  визначається як об'єднання усіх можливих степенів даного відношення:

$$P^\circ \stackrel{\text{def}}{=} P \cup P^2 \cup \dots \cup P^\infty.$$

**Транзитивне і рефлексивне замикання** визначається як:

$$P^* \stackrel{\text{def}}{=} P^\circ \cup I.$$

Граф транзитивного замикання  $P^*$  має дугу  $(v_j, v_i)$ , якщо є якийсь маршрут з вершини  $v_j$  у вершину  $v_i$ .

Згідно з визначенням, матриця суміжності  $R$  графу транзитивного замикання одержується як об'єднання матриць суміжності усіх степенів  $E^k$ . Замість об'єднання степенів матриць суміжності можна виконати їх додавання. Тоді слід виконати корекцію одержаної матриці за допомогою булевого перетворення.

**Теорема.** матриця досяжності  $R$  дорівнює

$$R = B \left( I + \sum_{k=1}^{p-1} E^k \right),$$

де  $B$  — булеве перетворення, а  $I$  — одинична матриця.

Дійсно, якщо вершина  $v_j$  є досяжною з  $v_i$ , то існує простий ланцюг з  $v_j$  у  $v_i$ . Довжина цього маршруту не перебільшує  $p-1$ , оскільки у простому ланцюгу вершини не повторюються. Відповідно, елемент  $i,j$  матриці  $I + E + E^2 + \dots + E^{p-1}$  буде ненульовим, звідки й слідує теорема.

Складність прямого обчислення матриці  $R$  за цією формулою складає  $O(p^4)$ . Матриця досяжності  $R$  може бути обчислена за матрицею суміжності  $E$  за алгоритмом Уоршала за  $O(p^3)$ .

#### Компоненти сильної зв'язності

За матрицею досяжності можна визначити компоненти сильної зв'язності за наступною теоремою.

**Теорема.** Нехай орграф  $G$  має матрицю досяжності  $R$  та матрицю суміжності  $E$ . Тоді:

- 1)  $G$  сильно зв'язаний тоді й тільки тоді, коли  $R = U$ , тобто, матриця, елементами якої є тільки 1.
- 2)  $G$  однобічно зв'язаний тоді й тільки тоді, коли  $B(R + R^T) = U$ , де  $R^T$  – транспонована матриця  $R$ ;
- 3)  $G$  слабо зв'язаний тоді й тільки тоді, коли  $B[(I + E + E^T)^{p-1}] = U$ .

**Теорема.** Нехай орграф  $G$  має матрицю досяжності  $R = (r_{ij})$  та  $R^2 = (s_{ij})$ .

Тоді:

- 1) сильна компонента, яка має вершину  $v_i$ , визначається одиничними елементами в  $i$ -му рядку чи  $i$ -му стовпці поелементного добутку матриць,  $R \bullet R^T$ , який називається **матрицею сильної зв'язності**, де  $R^T$  – транспонована матриця  $R$ ;
- 2) кількість вершин в сильній компоненті, яка має вершину  $v_i$ , дорівнює  $s_{ij}$ .

Доведення слідує з того, що вершина  $v_j$  досяжна з вершини  $v_i$  тоді, коли  $r_{ij} = 1$ . В свою чергу, згідно з умовою зв'язності,  $v_i$  досяжна з вершини  $v_j$  тоді, коли  $r_{ji} = 1$ . Отже,  $v_j, v_i$  взаємно досяжні лише у випадку, коли  $r_{ij} \cdot r_{ji} = 1$ . За визначенням квадрату матриці, величина  $s_{ij}$  дорівнює

$$s_{ij} = \sum_{j=1}^p r_{ij} \cdot r_{ji},$$

причому,  $r_{ij} \cdot r_{ji} = 1$  тоді, коли  $v_j, v_i$  взаємно досяжні. Отже, додавання цих чисел по усіх  $j$  дає кількість вершин  $v_i$ , які є взаємно досяжні.

Усі вершини сильно зв'язного підграфа досяжні одна для одної, тобто, вони перебувають у відношенні сильної зв'язності, яке є симетричним, рефлексивним і транзитивним. Отже, множина вершин сильно зв'язаної компоненти утворює один клас еквівалентності. Кілька таких підграфів – класів еквівалентності пов'язані між собою і утворюють новий граф  $G^*$ , вершини якого відповідають сильним компонентам графа  $G$ .

Орграф  $G^*$ , який називається **конденсацією графа  $G$** , будується наступним чином. Нехай  $K_1, K_2, \dots, K_q$  — сильні компоненти  $G$ . Тоді вибираємо множину вершин  $V(G^*) = \{K_1, K_2, \dots, K_q\}$ , і проводимо дугу від  $K_i$  до  $K_j$  тоді і тільки тоді, коли  $i \neq j$  і для таких вершин  $u \in K_i$  і  $v \in K_j$ , які з'єднані дугою  $(u, v)$  або  $(v, u)$  в  $G$ . На рис. 25 показаний приклад графа з сильними компонентами  $\{v_1, v_4, v_3\}$ ,  $\{v_2, v_5\}$  і  $\{v_6\}$ , яким відповідають вершини конденсованого графа  $\{K_1, K_2, K_3\}$ , відповідно.

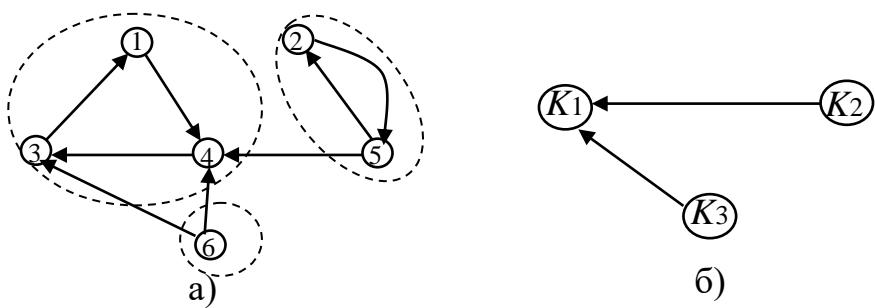


Рис. 25. Граф (а) і його конденсація (б)

**Теорема.** Конденсація графа є ациклічним графом. Доведення виходить з того, що конденсація графа одержана з компонент з сильною зв'язністю, і якщо б через вершини конденсації проходив би цикл, то вони б належали одній компоненті з сильною зв'язністю.

Сукупність вершин  $B$  орграфа  $G$  називається його **вершинною базою** (або базою вершин), якщо кожна вершина, яка не входить в  $B$ , досяжна з деякої вершини в  $B$ , і множина  $B$  є мінімальною. Тут мінімальність  $B$  означає, що ні з якої підмножини  $B$  не можна досягти всіх вершин з решти, що належить  $G$ .

Властивості вершинної бази наступні.

- В орграфі без циклів, у тому числі в конденсації графа, є єдина вершинна база, яка складена з усіх вершин, що не мають вхідних дуг.
- Нехай  $B^*$  — вершинна база конденсації графа  $G$ , тоді однією з багатьох вершинних баз  $B$  графа  $G$  служить множина, в яку входить по одній з вершин графа  $G$ , які входять в вершини вперше бази  $B^*$ .

На основі цих властивостей розроблено наступний алгоритм Кьюніга знаходження множини вершинних баз орграфа.

1. Знаходяться усі сильні компоненти орграфа  $G$ .
2. Будується конденсація  $G^*$  орграфа  $G$ .
3. Знаходиться множина вершин графа конденсації  $B^*$ , у які не входить жодна дуга (вершинна база  $B^*$  конденсації графа  $G^*$ ).
4. З кожної сильної компоненти, яка входить в  $B^*$ , вибирається по одній вершині. Ця множина є вершинною базою  $B$  орграфа  $G$ .

Згідно з алгоритмом, для графа на рис. 25 будуть знайдені дві вершинні бази:  $\{v_2, v_6\}$  і  $\{v_5, v_6\}$ .

Поняття компонент сильної зв'язності, вершинної бази є важливими при дослідженні графів комп'ютерних мереж, їх керованості, продуктивності та надійності.

### Приклад

Є граф, як на рис.26.

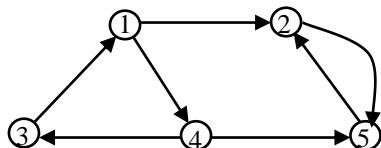


Рис. 26. Приклад ацикличного графа

Знайти:

- 1) всі шляхи довжини 2;
- 2) матрицю досяжності та її квадрат;
- 3) матрицю сильної зв'язності;
- 4) компоненти сильної зв'язності;
- 5) граф конденсації.
- 6) множину вершинних баз.

**Розв'язання.** Для зваженого орграфа, заданого графічно, матриця суміжності

$$E = \begin{vmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$

1) Знайдемо всі шляхи довжини 2. Для цього знаходимо квадрат матриці суміжності:  $E^2$

$$E^2 = \begin{vmatrix} 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

За матрицею  $E^2$ , наприклад, видно, що з вершини  $v_1$  веде 1 такий шлях у вершину  $v_3$  і 2 шляхи у вершину  $v_5$ . Проміжні вершини знаходяться з стовпців матриці  $E$ . Отже, маємо маршрути:  $(v_1, v_4, v_3)$ ,  $(v_1, v_4, v_5)$ ,  $(v_1, v_2, v_5)$ . Ці та решта маршрутів занесено у табл.3.

Табл.3 Маршрути довжини 2 для графа на рис.

Початок	Кінець	Шлях	Початок	Кінець	Шлях
$v_1$	$v_3$	$(v_1, v_4, v_3)$	$v_3$	$v_4$	$(v_3, v_1, v_4)$
$v_1$	$v_5$	$(v_1, v_2, v_5)$	$v_4$	$v_1$	$(v_4, v_3, v_1)$
$v_1$	$v_5$	$(v_1, v_4, v_5)$	$v_4$	$v_2$	$(v_4, v_5, v_2)$
$v_2$	$v_2$	$(v_2, v_5, v_2)$	$v_5$	$v_5$	$(v_5, v_4, v_5)$
$v_3$	$v_2$	$(v_3, v_1, v_2)$			

2) Знайдемо матрицю досяжності  $R(G)$ . Для цього обчислимо

$$E^3 = \begin{vmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$

$$E^4 = \begin{vmatrix} 0 & 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

$$I + E + E^2 + E^3 + E^4 = \begin{vmatrix} 2 & 4 & 1 & 2 & 4 \\ 0 & 3 & 0 & 0 & 2 \\ 2 & 3 & 2 & 1 & 2 \\ 1 & 3 & 2 & 2 & 4 \\ 0 & 2 & 0 & 0 & 3 \end{vmatrix}$$

Застосуємо до нього булеве відображення і одержимо матрицю досяжності:

$$R = B(I + E + E^2 + E^3 + E^4) = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{vmatrix}$$

Квадрат матриці досяжності:

$$R^2 = \begin{vmatrix} 3 & 5 & 3 & 3 & 5 \\ 0 & 2 & 0 & 0 & 2 \\ 3 & 5 & 3 & 3 & 5 \\ 3 & 5 & 3 & 3 & 5 \\ 0 & 2 & 0 & 0 & 2 \end{vmatrix}$$

Аналіз цієї матриці показує, що вершини  $v_1, v_3, v_4$  мають  $s_{ij} = 3$ , тобто, вони взаємно досяжні. Так само вершини  $v_2, v_5$  мають  $s_{ij} = 2$ . Отже, ці вершини, напевне, належать до компонент зв'язності.

3) Використаємо матрицю досяжності для одержання матриці сильної зв'язності графа G. Знайдемо

$$R^T = \begin{vmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

Обчислимо поелементний добуток матриць і одержимо матрицю сильної зв'язності

$$S = R \bullet R^T = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{vmatrix}$$

4) Щоб виділити компоненти сильної зв'язності, переставимо 2-й і 4-й стовпці та рядки одержаної матриці з урахуванням взаємної досяжності вершин:

$$S' = \begin{vmatrix} 1 & 4 & 3 & 2 & 5 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{vmatrix}$$

Як бачимо, одержана матриця є блоково-діагональна з блоками (клітками), які відповідають компонентам сильної зв'язності. Отже, знайдені компоненти сильної зв'язності є  $\{v_1, v_4, v_3\}$  і  $\{v_2, v_5\}$ .

5) Після конденсації вершин, які належать цим компонентам, одержимо граф конденсації як на рис. 27.

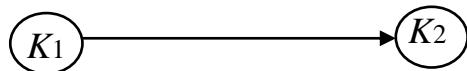


Рис. 27. Граф конденсації

6) Вершинні бази мають по одній вершині, що належить до  $K_1$ , тобто, є три вершинні бази:  $\{v_1\}, \{v_3\}, \{v_4\}$ .

## Алгоритми обходу графа

Виконання будь-якого обходу графа полягає в послідовному відвідуванні його вершин за маршрутом, який відповідає певному алгоритму. Під час обходу виконується дослідження ребер і вершин. Зокрема, на певний час запам'ятовується факт відвідування вершини, яка відмічається як **відвідана**. Вершину, яка ще не відвідана, називається **новою**. В результаті відвідування вершина відмічається як відвідана і залишається такою, поки не будуть дослідженні всі інцидентні їй ребра. Після цього вона відмічається як **закрита**.

## Алгоритм пошуку вшир

Ідея *пошуку вшир* або *BFS-методу* (breadth first search) полягає в тому, щоб відвідувати вершини в порядку їх віддаленості від стартової вершини, яка є заздалегідь обрана. Нехай це вершина  $a$ . Тоді спочатку відвідується сама вершина  $a$ , потім — усі вершини, які є суміжні з  $a$ , тобто ті, що знаходяться від неї на відстані 1, потім вершини, що знаходяться від  $a$  на відстані 2, і т.д.

Розглянемо алгоритм пошуку в ширину із заданою стартовою вершиною  $a$ . Спочатку усі вершини позначаються як нові. Першою відвідується вершина  $a$ , яка позначається як відвідана. Надалі кожен черговий крок починається з вибору деякої відвіданої вершини  $x$ . Ця вершина стає активною. Далі досліджуються ребра, які є інцидентні активній вершині. Якщо таке ребро з'єднує вершину  $x$  з новою вершиною  $y$ , то вершина  $y$  відвідується і відмічається як відвідана.

Коли досліджені усі ребра, які є інцидентні активній вершині, вона перестає бути активною і стає закритою. Після цього вибирається нова активна вершина, і описані дії повторюються. Процес закінчується, коли множина невідвіданих вершин стає порожньою.

Основна особливість алгоритму пошуку вшир, що відрізняє його від інших, полягає в тому, що активною вершиною вибирається та з відвіданих вершин, яка стала відвідана раніше інших. Для реалізації такого правила вибору активної вершини зручно використовувати пам'ять у вигляді черги або FIFO (first in — first out). Схематично процес зміни статусу вершин зображений на рис. 28.

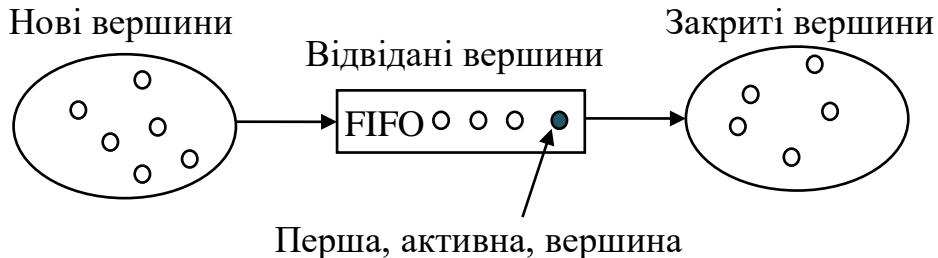


Рис. 28. Множини вершин при пошуку вшир

Нижче наводиться процедура пошуку вшир BFS. Вона приймає на вхід два параметри: ненапрямлений граф  $G$  та початкова вершина  $a$ . Процедура починає обхід графу з вершини  $a$ , додаючи у чергу  $Q$  нові вершини. Робота закінчується, коли черга  $Q$  стає порожньою.

```

BFS( $G, a$ ) {
    BFS[ $i$ ] = 0,  $i \in V$ ; // усі вершини в  $G$  позначити як нові
    BFS[ $a$ ] = 1; // позначити  $a$  як відвідувану
    [ $a$ ] →  $Q$ ; // внести в чергу  $Q$  вершину
     $k = 1$ ;
    while  $Q \neq \emptyset$  {
         $v = Q[1]$ ; //  $v$  – перша вершина в  $Q$ , тобто, активна
        for ( $v, u$ ) ∈  $G$  {
            if BFS[ $u$ ]=0; { // якщо вершина  $u$  – нова, то:
                 $k = k + 1$ ;
                BFS[ $u$ ] =  $k$ ; // позначити  $u$  як відвідувану,
                [ $u$ ] →  $Q$ ; // додати  $u$  в кінець черги
            }
        }
    }
}

```

Тут цикл **for** виконується з вершинами, які є суміжними до даної. При цьому наступна вершина вибирається за порядком, який визначений для даного графа, наприклад, за зростанням номерів вершин або позиції їх назви у словнику, як у наступному прикладі.

Для прикладу виконаємо обхід графу, який зображене на рис. 29, починаючи з вершини  $a$ . Протокол обходу наведено в таблиці 4.

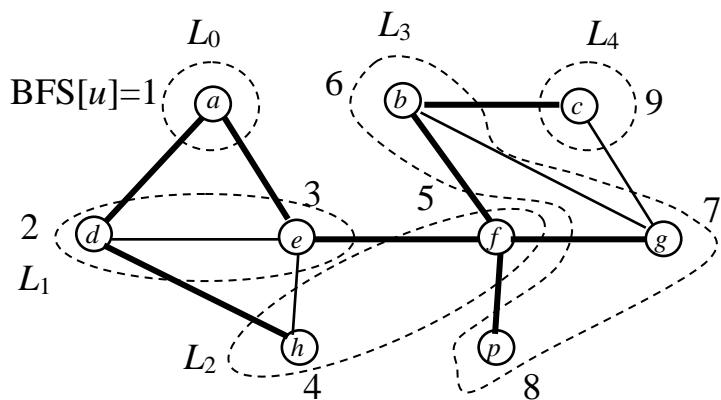


Рис. 29. Граф і його обхід в ширину

Табл. 4. Протокол обходу в ширину графу на рис. 29.

Нові вершини	Активна вершина	Відвідана вершина	BFS[u]	Черга $Q$
$abcdefghp$	—	—	0	$\emptyset$
$bcd\cancel{efghp}$	$a$	—	1	$a$
$bce\cancel{fghp}$	$a$	$d$	2	$da$
$bcf\cancel{ghp}$	$a$	$e$	3	$eda$
$bcf\cancel{ghp}$	$d$	—	—	$ed$
$bcfgp$	$d$	$h$	4	$hed$
$bcfgp$	$e$	—	—	$he$
$bcgp$	$e$	$f$	5	$fhe$
$bcgp$	$h$	—	—	$fh$
$bcgp$	$f$	—	—	$f$
$cgp$	$f$	$b$	6	$bf$
$cp$	$f$	$g$	7	$gbf$
$c$	$f$	$p$	8	$pgbf$
$c$	$b$	—	—	$pgb$
$\emptyset$	$b$	$c$	9	$cpgb$
$\emptyset$	$g$	—	—	$cpg$
$\emptyset$	$p$	—	—	$cp$
$\emptyset$	$c$	—	—	$c$
$\emptyset$	—	—	—	$\emptyset$

Відзначимо деякі властивості процедури BFS.

- Процедура BFS закінчує роботу після скінченного числа кроків. При кожному повторенні циклу while з черги видаляється одна вершина.

Кожна вершина може бути переглянуло не більше одного разу, оскільки відвідуються тільки нові вершини, а в результаті відвідування вершина перестає бути новою. Отже, складність процедури BFS є  $O(m)$ , де  $m$  — кількість ребер в компоненті зв'язності, що містить вершину  $a$ .

2. При виконання процедури BFS будуть відвідані усі вершини лише з однієї компоненти зв'язності, що містить вершину  $a$ . Щоб обійти вершини інших компонент зв'язності слід повторити процедуру, починаючи з будь-якої вершини, яка належить компонентам, що розглядаються. Пошук компонент зв'язності завершується, коли стане пустою множина нових вершин. Такий алгоритм лежить в основі одного з методів вирішення задачі виявлення компонент зв'язності графа. Тоді кількість компонент зв'язності дорівнюватиме числу викликів процедури BFS.

3. Процедура BFS виконується також з напрямленими графами без циклів. Тоді в циклі **for** розглядаються вершини, в які прямують дуги з даної активної вершини.

## BFS-дерево і обчислення відстаней

Інша просте завдання, для вирішення якого можна застосувати пошук в ширину, — це побудова каркасу. Нагадаємо, що каркасом графа називається кістяковий ліс, у якого області зв'язності збігаються з областями зв'язності графа. Каркас зв'язного графа — це кістяк.

Ребра, які досліджуються в процесі обходу графа, можна розділити на дві категорії: якщо ребро з'єднує активну вершину  $x$  з новою вершиною  $y$ , то воно класифікується як **пряме**, в іншому випадку — як **зворотне**. Залежно від розв'язуваної задачі прямі і зворотні ребра можуть піддаватися різній обробці.

**Теорема.** Якщо алгоритм пошуку в ширину застосовується до зв'язного графу, то результуюча множина всіх прямих ребер утворює дерево.

Дійсно, припустимо, що на певному етапі роботи алгоритму виявляється нове пряме ребро  $(x, y)$ , а множина прямих ребер, накопичених до цього кроку, утворює дерево  $F$ .

Тоді вершина  $x$  належить дереву  $F$ , а вершина  $y$  не належить йому. Тому при додаванні до дерева  $F$  ребра  $(x, y)$  зв'язність зберігається, а циклів не з'являється.

Отже, якщо застосувати пошук в ширину до зв'язного графу і запам'ятати усі прямі ребра, то отримаємо кістяк графа. Для довільного графа буде отримано ліс, який також, очевидно, є каркасом.

Кістяк, який буде побудований описаним чином в результаті пошуку в ширину в зв'язковому графі, називається **BFS-деревом**. Його можна розглядати як кореневе дерево з коренем у стартовій вершині  $a$ . BFS-дерево з заданим коренем  $a$ , не є єдиним — воно залежить від того, в якому порядку проглядаються околиці активної вершини.

Кістяк  $T$  зв'язного графа  $G$  з коренем  $a$  називається **геодезичним деревом**, якщо для будь-якої вершини  $x$  шлях з  $x$  в  $a$  в дереві  $T$  є найкоротшим шляхом між  $x$  і  $a$  в графі  $G$ .

**Теорема.** Будь-яке BFS-дерево є геодезичним деревом.

Доведення теореми ґрунтуються на суті алгоритму BFS, який послідовно формує множини вершин, які на 1, 2 і т.д. кроків віддалені від стартової — кореневої вершини. Для графа на рис. 29 це множини  $L_0, L_1, L_2, L_3, L_4$  вершин, які знаходяться на дистанції 0, ..., 4 кроки від кореня.

Отже, ми можемо застосувати пошук вшир для обчислення відстаней від стартової вершини  $a$  до всіх інших вершин графа — потрібно лише в процесі обходу для кожної відвідуваної вершини у визначати відстань від вершини  $u$  до  $a$  в BFS-дереві. Це можна зробити за таким алгоритмом розрахунку відстань  $d(a, x)$  дляожної вершини  $x$  графа.

Спочатку встановлюється  $d(a, a) = 0$  та  $d(a, x) = \infty$  для всіх  $x \neq a$ . Тут нескінченність відстані є ознакою, що вершина — нова у розгляді і

відстань наприкінці залишиться нескінченною, якщо у дану вершину немає маршруту.

Далі під час виконання процедури BFS підраховується відстань:

$$d(a, y) = d(a, x) + 1,$$

де  $x$  — активна вершина.

Якщо після закінчення роботи процедури BFS для усіх компонент зв'язності трапляється  $d(a, x) = \infty$  для деякої вершини  $x$ , це означає, що  $x$  недосяжна з  $a$ , тобто належить іншій компоненті зв'язності.

### Алгоритм пошуку углиб

Другий базовий метод обходу графу — алгоритм пошуку углиб або **DFS-метод** (depth first search). Він, ймовірно, виражає найбільш важливу стратегію обходу графа з огляду на численність застосувань. Ідея цього методу — просуватись вперед в недосліджену область, поки це можливо, а якщо ж навколо все досліджено, відступити на крок назад і шукати нові можливості для просування вперед. Пошук углиб відомий під різними назвами, наприклад «бектрекінг», «пошук з поверненням».

Як і у пошуку вшир, обхід починається з заданої стартовою вершини  $a$ , яка стає активною і єдиною відкритою вершиною. Потім вибирається ребро  $(a, y)$ , яке є інцидентним вершині  $a$  і відвідується вершина  $y$ , яка стає відкритою і активною. Зауважимо, що при пошуку вшир вершина  $a$  залишалася активною до тих пір, поки не були досліжені всі інцидентні їй ребра.

Надалі, як і при пошуку вшир, кожен черговий крок починається з вибору активної вершини з множини відвіданих вершин. Якщо усі ребра, інцидентні активній вершині  $x$ , вже досліжені, вона перетворюється у закриту. В іншому випадку вибирається одне з недосліджених ребер  $(x, y)$ , яке досліджується. Якщо вершина  $y$  нова, то вона відвідується і відмічається як відвідувана.

Головна відмінність від пошуку вшир полягає в тому, що при пошуку углиб як активна вибирається та з відвідуваних вершин, яка була відвідана останньою. Для реалізації такого правила вибору найбільш зручною структурою зберігання множини відвіданих вершин є стек. Стек — це така структура даних, яка видає збережені дані у зворотньому порядку до їх запису у неї, тобто, за правилом: останній прийшов — перший вийшов (last in — first out, LIFO). Отже, відвідувані вершини складаються у стек в тому порядку, в якому вони відвідуються, а за активну вибирається остання вершина у стеку. Схематично структура машини, що виконує алгоритм пошуку углиб показана на рис. 30.

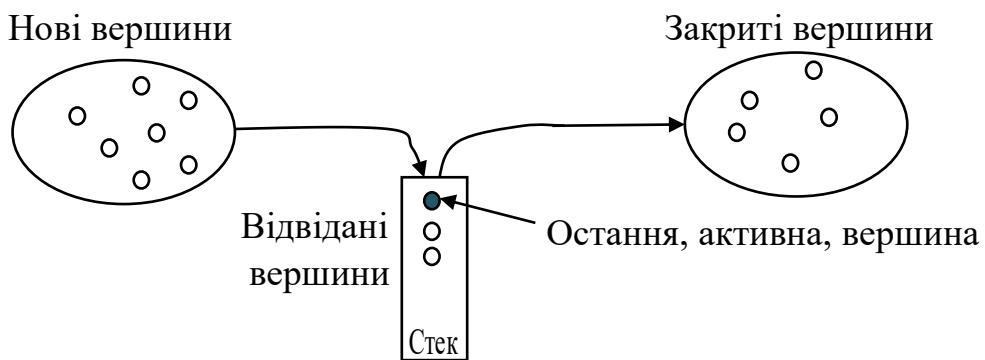


Рис. 30. Схема алгоритму при пошуку углиб

Процедура пошуку углиб DFS відрізняється від процедури пошуку вшир BFS наявністю стека  $S$  замість черги  $Q$ .

```

DFS( $G, a$ ) {
    DFS[ $i$ ] = 0,  $i \in V$ ; // усі вершини в  $G$  позначити як нові
    DFS[ $a$ ] = 1; // позначити  $a$  як відвідану
    [ $a$ ]  $\rightarrow S$ ; // внести вершину в голову  $S$ 
     $k = 1$ ;
    while  $S \neq \emptyset$  {
         $v = \text{top}(S)$ ; //  $v$  — вершина з голови  $S$ , тобто, активна
        for ( $v, u$ )  $\in G$  {
            if DFS[ $u$ ]=0; { // якщо вершина  $u$  — нова, то:
                 $k = k + 1$ ;
                DFS[ $u$ ] =  $k$ ; // позначити  $u$  як відвідану,
                [ $u$ ]  $\rightarrow S$ ; // додати  $u$  в голову стека
            else flush( $S$ ); // видалити вершину з голови стеку
        }
    }
}

```

```

    }
}
}

```

Через  $\text{top}(S)$  позначено верхній елемент стека, тобто, останній елемент, який до нього додано. Функція  $\text{flush}(S)$  видаляє елемент з голови стеку.

Для прикладу виконаємо обхід графу, який зображене на рис. 31, починаючи з вершини  $a$ . Вважаємо, що ребро  $(p,h)$  — відсутнє. Протокол обходу наведено в таблиці 5.

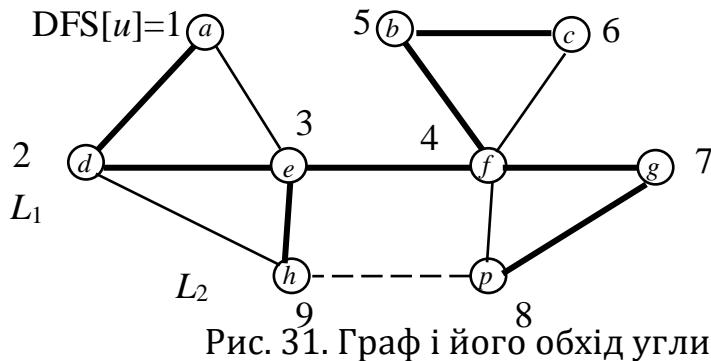


Рис. 31. Граф і його обхід у глиб

Табл. 5. Протокол обходу в ширину графу на рис. 31.

Нові вершини	Активна вершина	Відвідана вершина	BFS[u]	Черга Q
$abcdefghp$	—	—	0	$\emptyset$
$bcedfghp$	$a$	—	1	$a$
$bcefghp$	$d$	$d$	2	$da$
$bcfghp$	$e$	$e$	3	$eda$
$bcghp$	$f$	$f$	4	$feda$
$cghp$	$b$	$b$	5	$bfeda$
$ghp$	$c$	$c$	6	$cbfeda$
$ghp$	$b$	—	—	$bfeda$
$hp$	$f$	—	—	$feda$
$hp$	$g$	$g$	7	$gfeda$
$h$	$p$	$p$	8	$pgfeda$
$h$	$g$	—	—	$gfeda$
$h$	$f$	—	—	$feda$
$h$	$e$	—	—	$eda$
$\emptyset$	$h$	$h$	9	$heda$
$\emptyset$	$e$	—	—	$eda$
$\emptyset$	$d$	—	—	$da$
$\emptyset$	$a$	—	—	$a$
$\emptyset$	—	—	—	$\emptyset$

Властивості алгоритму пошуку в ширину, які відмічені в попередньому розділі, зберігаються і для пошуку в глибину. При виконанні алгоритму в операторі `if` в гілці `then` оператори повторюються  $O(m)$  раз, а у гілці `else` — повторюється  $O(n)$  раз, оскільки кожна вершина може бути видалена зі стека тільки один раз. В цілому, виходить складність  $O(m + n)$ .

### DFS-дерево

Пошук в глибину можна застосувати для знаходження компонент зв'язності графа або для побудови каркаса таким самим чином, як при пошуку в шир. Поняття прямого і зворотного ребра визначаються так само і так само доводиться, що прямі ребра при пошуку углиб утворюють каркас графа. Для зв'язного графа кістяк, який одержується пошуком углиб, називається **DFS-деревом**. DFS-дерево розглядається як кореневе дерево з коренем у стартовій вершині  $a$ . Це дерево має особливі властивості, на використанні яких засновані численні застосування методу пошуку углиб. Розглянемо найбільш важливі з цих властивостей.

Щодо будь-якого кореневого кістяка, усі ребра графа, що не належать дереву, можна розділити на дві категорії. Ребро називається **поздовжнім**, якщо одна з його вершин є предком іншої, в іншому випадку, ребро називається **поперечним**. На рис. 31 невиділені ребра є зворотні ребра, серед них повздовжні ребра це  $(a,e), (d,h), (c,f), (f,p)$ , а поперечне —  $(p,h)$ .

**Теорема.** Нехай  $G$  — зв'язний граф,  $T$  — DFS-дерево графа  $G$ . Тоді щодо  $T$  усі зворотні ребра є поздовжніми.

Згідно з цією теоремою, ребро  $(p,h)$  на рис. 31 має бути відсутнім, або дерево  $T$  було побудоване не за алгоритмом DFS.

### Топологічне сортування

Одним з застосувань пошуку углиб є топологічне сортування.

**Топологічним сортуванням** орієнтованого ациклічного графу  $G = (V, E)$

називається таке лінійне впорядкування усіх його вершин, що якщо граф містить ребро  $(v, u)$ , то вершина  $v$  в ньому знаходиться перед вершиною  $u$ .

Топологічне сортування графу можна розглядати як наявність лінійного порядку між його вершинами (див. рис. 8, в). Воно має багато практичних застосувань, серед яких впорядкування взаємопов'язаних задач у певний розклад. При цьому вершини можна вистроїти вздовж прямої, яка є віссю часу. Наприклад, необхідно визначити порядок виконання задач на одному процесорі. В загальному випадку, топологічне сортування не обов'язково є єдиним. Наприклад, виконання алгоритмів пошуку вшир і углиб дають таке сортування.

**Теорема.** Якщо в орієнтованому графі існує хоча б один цикл, то в ньому не існує топологічного сортування.

Доведення. Припустимо, що в графі існує пара вершин  $v$  та  $u$ , такі що існує орієнтований маршрут з  $v$  до  $u$  та, навпаки, – з  $u$  до  $v$ , тобто, одночасно  $v > u$  та  $u > v$ . Отже, ці вершини включені в єдиний орієнтований цикл. При спробі побудувати топологічне сортування алгоритм побудови натикається на протиріччя  $v > u$  та  $u > v$ , яке не узгоджується з означенням такого сортування.

Графічне представлення топологічного сортування графу на рис. 31 показане на рис. 32.

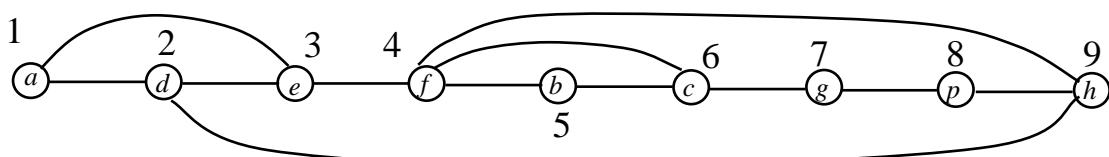


Рис. 32. Результат топологічного сортування графа на рис. 31.

## Алгоритми найкоротшого шляху

*Зважений граф*  $G = (V, E, W)$  складається з множини вершин  $V$ , множини ребер  $E$  і множини вагів  $W$ , що визначає ваги  $w_{ij}$  для ребер  $(i, j) \in E$ . Такий граф часто називають **мережею**.

### 1 Алгоритм Дейкстри

**Найкоротший шлях** — це найкоротший маршрут між будь-якими двома фіксованими парами вершин орієнтованого або неорієнтованого зваженого графа.

Одним з найбільш важливих і корисних алгоритмів є алгоритм Дейкстри, який знаходить найкоротші шляхи у графі. Алгоритм Дейкстри є **жадібним алгоритмом** оптимізації. Це такий алгоритм, який на черговому кроці завжди робить вибір, який виглядає кращим на даний момент. При цьому виконується місцевий оптимальний вибір серед усіх локальних варіантів, який додається до поточного рішення.

Оскільки графи здатні представити багато видів відношень, багато проблем можуть бути задані як проблеми найкоротшого шляху, що робить алгоритм Дейкстри потужним і універсальним інструментом.

#### 1.1 Застосування алгоритму Дейкстри

- Алгоритм Дейкстри застосовується для автоматичного визначення маршрутів між фізичними місцями, наприклад, маршрути на сайтах, таких як Google Maps.

- У мережевих або телекомунікаційних застосунках алгоритм Дейкстри використовується для вирішення проблеми шляху проходження пакетів. Наприклад, в маршрутизації мереж передачі даних мета полягає в тому, щоб знайти шлях для пакетів даних, які проходять через мережу комутації з мінімальною затримкою.

- Він також використовується для вирішення різноманітних проблем найкоротшого шляху, що виникають у плануванні виробництва, робототехніки, транспортування та проектуванні друкованих плат чи мікросхем.

Алгоритм Дейкстри вирішує проблему найкоротшого шляху, щоб знайти найкоротший шлях від заданої вершини  $s$ , яка називається **стартовою вершиною** або початковою вершиною, до всіх інших вершин мережі. Цей алгоритм вирішує тільки проблеми з невід'ємними вагами, тобто,  $w_{ij} \geq 0$  для всіх  $(i, j) \in E$ .

Алгоритм характеризує кожну вершину за своїм станом. **Стан вершини** складається з двох елементів: значення відстані та мітки статусу.

- **Значення відстані**  $l(v)$  вершини — це величина, що представляє оцінку його відстані від вершини  $s$ .
- **Мітка статусу** — це атрибут, який вказує чи значення відстані вершини дорівнює найкоротшій відстані до вершини  $s$  чи ні. Мітка статусу вершини є  $P$ , тобто, постійна, якщо значення його відстані  $l(v)$  дорівнює найкоротшій відстані від вершини  $s$ . В іншому випадку мітка статусу вершини є  $T$ , тобто, вона тимчасова.

При виконанні алгоритму покроково оновлюються стани вершин. На кожному кроці одна вершина позначається як **активна**  $u$ .

#### **Алгоритм Дейкстри:**

Початкові дані: зважений граф  $G = (V, E, W)$ , стартова вершина  $s$ , вершина призначення  $t$ .

Крок 1: Встановити  $l(s)$  і позначити вершину  $s$  міткою постійна. Для всіх вершин  $v \neq s$  призначають тимчасову мітку та  $l(v) = \infty$ . Тобто, стан вершини  $s$  —  $(0, P)$ , а стан інших вершин  $(\infty, T)$ . Призначити вершину  $s$  як активну вершину  $u = s$ .

Крок 2: Якщо  $u$  — вершина з постійною міткою, то для кожного ребра  $e = (u, v)$ , що виходить з  $u$ , якщо  $l(v) > l(u) + w(e)$ ,

то встановити  $l(v) = l(u) + w(e)$ ,

де  $w(e)$  — вага ребра  $e = (u, v)$  і  $v$  є вершиною з міткою тимчасова.

Крок 3: Нехай  $k$  — вершина з міткою тимчасова, для якої  $l(k)$  — мінімальна. Якщо немає такої вершини  $k$ , то приймається рішення, що не

існує найкоротшого шляху від стартової вершини  $s$  до вершини призначення. Інакше перейти до кроку 4.

Крок 4: Зробити мітку вершини  $k$  постійною. Якщо  $k = t$ , то кінець. В іншому випадку встановити  $u = k$ , тобто, призначити цю вершину як активну, а потім перейти до кроку 2.

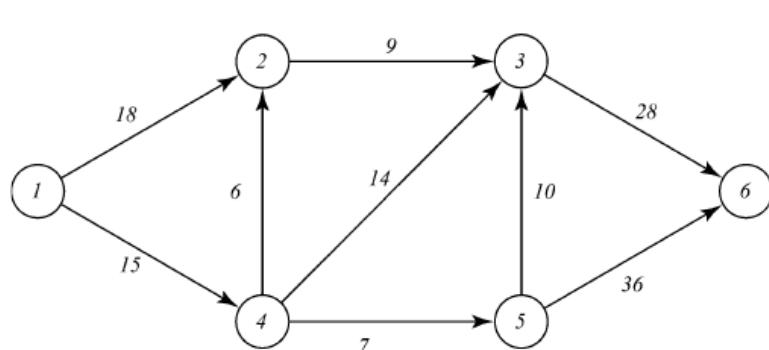
Алгоритм Дейкстри починається з призначення деяких початкових значень для відстаней від вершини  $s$  до решти вершин в графі. Віна діє покроково і на кожному кроці алгоритм покращує значення відстані. На кожному кроці визначається найкоротша відстань від вершини  $s$  до іншого вершини.

### 1.3 Складність алгоритму Дейкстри

Алгоритм Дейкстри вирішує задачу найкоротшого шляху за час  $O(|V|^2)$ . Алгоритм містить зовнішній цикл, який виконується  $|V|$  разів і внутрішній цикл, у якому знаходяться найближчі вершини і оновлені відстані і який виконується  $O(|V|)$ . Тому результуюча часова складність оцінюється як  $O(|V|^2)$ .

#### Приклад.

Виконати алгоритм Дейкстри для графа на рис. 5.1 щоб знайти найкоротший шлях від вершини 1 до вершини 6.



**Fig. 5.1**

Спочатку призначається відстань  $l(1) = 0$  початковій вершині 1 і вона позначається міткою  $P$ . Іншим — не-стартовим вершинам повинні бути присвоєні відстані  $\infty$  та мітки — тимчасова (Таблиця 1).

Таблиця 1

Вершина ( $v$ )	<b>1</b>	2	3	4	5	6
Відстань $l(v)$	<b>0</b>	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Мітка ( $v$ )	<b>P</b>	T	T	T	T	T
Попередник ( $v$ )	—	—	—	—	—	—

Вершини 2 і 4 є суміжними з вершиною 1 і мають мітки тимчасових. У Таблиці 2 оновлені статуси цих вершин, а також позначений їхній попередник — вершина 1.

Таблиця 2. Вершини 2 і 4 суміжні до вершини 1.

Вершина ( $v$ )	1	<b>2</b>	3	<b>4</b>	5	6
Відстань $l(v)$	0	18	$\infty$	15	$\infty$	$\infty$
Мітка ( $v$ )	P	T	T	T	T	T
Попередник ( $v$ )	—	1	—	1	—	—

Далі шукається мінімум шляхів до вершин з тимчасовими мітками.  $\min(18, \infty, 15, \infty, \infty) = 15$ . Знайдена вершина 4 помічається як постійна (таблиця 3).

Таблиця 3. Вершина 4 з мінімальною відстанню стає постійною

Вершина ( $v$ )	1	2	3	<b>4</b>	5	6
Відстань $l(v)$	0	18	$\infty$	<b>15</b>	$\infty$	$\infty$
Мітка ( $v$ )	P	T	T	<b>P</b>	T	T
Попередник ( $v$ )	—	1	—	<b>1</b>	—	—

Вершини, які помічені як тимчасові і суміжні з вершиною 4, це 2, 3 і 5. Для вершини 2  $l(4) + w(4,2) = 15 + 6 = 21$ , але в Таблиці 3 відстань ввершини 2 була 18, тобто, меншою. Отже, ніяких змін не потрібно. У Таблиці 4 оновлені мітки вершин 3 і 5, а також позначено їх попередник — вершина 4.

Таблиця 4 Вершини 2, 3 і 5 — суміжні до вершини 4.

Вершина ( $v$ )	1	2	3	4	5	6
Відстань $l(v)$	0	18	29	15	22	$\infty$
Мітка ( $v$ )	P	T	T	P	T	T
Попередник ( $v$ )	—	1	4	1	4	—

Знову ж таки, шукається мінімум серед вершин з мітками тимчасова.  $\min(18, 29, 22, \infty) = 18$ , отже, вершина 2 стає постійною (Таблиця 5)

Таблиця 5. Тимчасова вершина 2 з мінімальною відстанню стає постійною

Вершина ( $v$ )	1	<b>2</b>	3	4	5	6
Відстань $l(v)$	0	<b>18</b>	29	15	22	$\infty$
Мітка ( $v$ )	$P$	$P$	$T$	$P$	$T$	$T$
Попередник ( $v$ )	—	<b>1</b>	4	1	4	—

Тільки вершина 3 є суміжною з вершиною 2. Оскільки,  $l(2) + w(2,3) = 27$  нова відстань у вершині 3 буде 27, а відповідний попередник стане 2.

Таблиця 6 Сусідньою вершиною для 2 є тільки вершина 3

Вершина ( $v$ )	1	2	<b>3</b>	4	5	6
Відстань $l(v)$	0	18	27	15	22	$\infty$
Мітка ( $v$ )	$P$	$P$	$T$	$P$	$T$	$T$
Попередник ( $v$ )	—	1	2	1	4	—

Оскільки серед вершин, помічених тимчасові, мінімальну відстань має вершина 5, вона стає постійною (Таблиця 7).

Таблиця 7 Вершина 5 стає постійною

Вершина ( $v$ )	1	2	3	4	<b>5</b>	6
Відстань $l(v)$	0	18	27	15	<b>22</b>	$\infty$
Мітка ( $v$ )	$P$	$P$	$T$	$P$	<b><math>P</math></b>	$T$
Попередник ( $v$ )	—	1	2	1	<b>4</b>	—

Вершинами, які є тимчасові і суміжні з вершиною 5, це вершини 3 і 6.

Для вершини 3  $l(5) + w(5,3) = 22 + 10 = 32$ . Але в таблиці 7 відстань для неї була 27. Отже, зміна не потрібна, оскільки  $\min(27, 32) = 27$ . Для вершини 6 оновлюється статус.

Таблиця 8 Суміжні вершини 5 - це 3 і 6

Вершина ( $v$ )	1	2	<b>3</b>	4	5	<b>6</b>
Відстань $l(v)$	0	18	27	15	22	58
Мітка ( $v$ )	$P$	$P$	$T$	$P$	$P$	$T$
Попередник ( $v$ )	—	1	2	1	4	5

Серед вершин, помічених як тимчасові, відстань мінімальна  $\min(27, 58) = 27$ . Отже, вершина 3 була помічена як постійна.

Таблиця 9 Вершина 3 з мінімальною відстанню стає постійною

Вершина ( $v$ )	1	2	<b>3</b>	4	5	<b>6</b>
Відстань $l(v)$	0	18	<b>27</b>	15	22	58
Мітка ( $v$ )	$P$	$P$	<b><math>P</math></b>	$P$	$P$	$T$
Попередник ( $v$ )	—	1	<b>2</b>	1	4	5

Тільки вершина 6 є суміжною з вершиною 3. У таблиці 10 для неї  $l(3) + w(3,6) = 27 + 28 = 55$ , але з таблиці 9,  $\min(55, 58) = 55$ . Отже, нова відстань для вершини 6 дорівнює 55, а оновлений попередник — 3.

Таблиця 10 Вершина 6 є суміжною до вершини 3, і вона стає постійною

Вершина ( $v$ )	1	2	3	4	5	<b>6</b>
Відстань $l(v)$	0	18	27	15	22	55
Мітка ( $v$ )	$P$	$P$	$P$	$P$	$P$	$P$
Попередник ( $v$ )	—	1	2	1	4	5

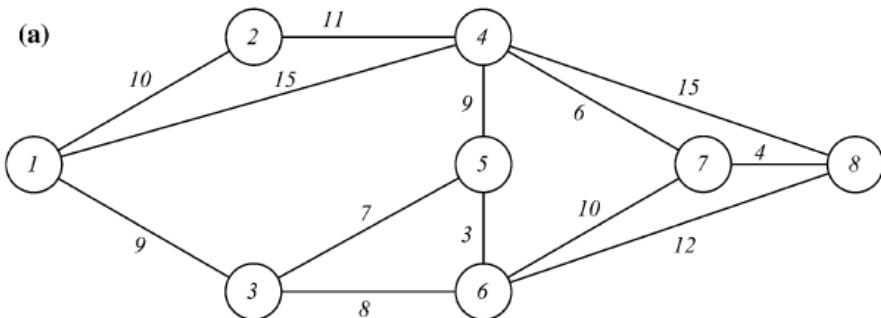
Оскільки вершина призначення  $t = 6$  стає постійною, алгоритм зупиняється. Найкоротша відстань від ввршини 1 до 6 становить 55 одиниць.

Щоб визначити найкоротший маршрут, повертаються від вершини 6 призначення до початкової вершини 1. З таблиць 10, 9 і 5 видно, що попередник 6 дорівнює 3, попередник 3 — 2, а попередник 2 — 1, відповідно. Отже, найкоротший маршрут — 1-2-3-6.

Крім того, можна пересвідчитись з мережі на Рис. 5.1, сума ваг ребер уздовж найкоротшого шляху дорівнює 55.

## 2 Пошук мінімального кістяка

Розглянемо наступний приклад укладання телефонного кабелю в місцевості, як показано на рис. Число на кожному ребрі являє собою відстань між вершина, які з'єднані цим ребром.



### 2.1 Задача мінімального кістяка

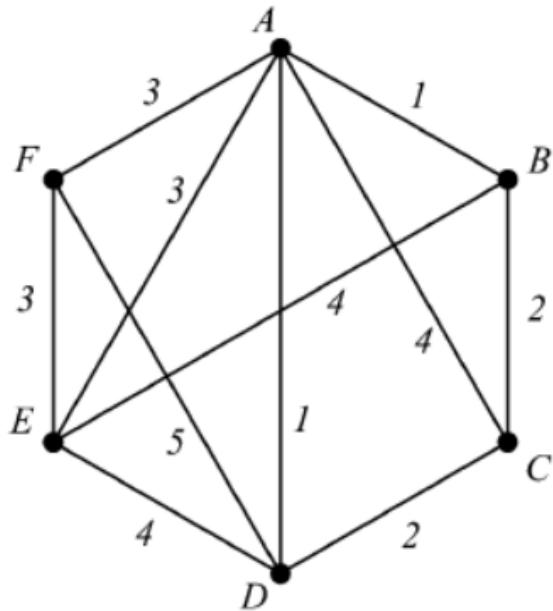
Задача мінімального кістяка полягає в тому, щоб з'єднати вершини мережі набором ребер так, що загальна довжина ребер є мінімальною. У процесі побудови мінімального кістяка слід дбати про те, щоб у ньому не було циклу. Що стосується прикладу прокладання телефонного кабелю, метою є підключення всіх вершин набором ребер, щоб була мінімальна сумарна довжина покладеного телефонного кабелю.

#### Мінімальний кістяк

Нехай  $G$  - зважений граф, в якому кожному ребру  $e$  присвоєно дійсне число  $w(e)$ , яке називається вагою ребра  $e$ . Нехай  $G^*$  — підграф зваженого графа, а вага  $w(G^*) = w(e_1) + \dots + w(e_n)$  — сума ваг усіх ребер підграфа  $G^*$ .

Кістяк  $G_T$  зваженого графа  $G$  називається **мінімальним кістяком**, якщо вага  $w(G_T)$  мінімальною.

Багато задач оптимізації передбачають пошук у відповідному зваженому графі, підграф певного типу з мінімальною вагою. Нехай  $G$  — граф, множина вершин якого є множиною міст, а  $(u, v)$  є ребром тоді і тільки тоді, коли можна прокласти трубопровід, що з'єднує міста  $u$  і  $v$ . У зваженому графі шляхом присвоєння кожному краю вартості побудови відповідного трубопроводу. Наприклад, припустимо, що є шість міст  $A; B; C; D; E; F$  і отримаємо зважений граф  $G$ , як показано на рис.



Відсутність ребра від  $B$  до  $D$  вказує на те, що неможливо побудувати трубопровід від  $B$  до  $D$ . Число (вага), що присвоюється ребру від  $F$  до  $D$ , означає вартість будівництва трубопроводу від  $F$  до  $D$ .

Оскільки проблема полягає в тому, щоб забезпечити кожне місто водою з міста-джерела, ми шукаємо підключений зв'язний граф  $G$ . Крім того, оскільки ми хочемо зробити це найекономічнішим способом, такий підлеглий підграф не повинен мати циклів, оскільки видалення ребра (трубопроводу) з циклу в підграфі-кістяку. Отже, ми шукаємо кістяк графа  $G$ . Більш того, економічний фактор означає, що ми хочемо найдешевший кістяк, тобто кістяк з мінімальною вагою.

Тепер розглянемо два алгоритми — алгоритми Краскала і Пріма — для знаходження мінімального кістяка для зв'язного зваженого графа, у якого вага не є негативною.

## 2.2 Алгоритм Краскала

Нехай  $G = (V, E)$  — зважений граф. шуканий граф  $G_T = \emptyset$ .

Крок 1: Вибрати одне ребро  $e_i$  в  $G$  таким чином, щоб його вага  $w(e_i)$  була мінімальною і видалити його з  $G$ .

Крок 2:

Додати ребро  $e_i$  до  $G_T$ , якщо воно не є хордою, тобто, не замикає цикл.

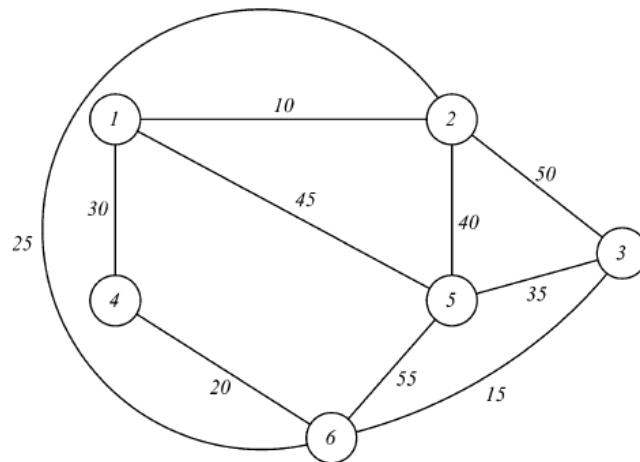
Крок-3:

Стоп, коли  $G = \emptyset$ , інакше перехід на крок 1.  $G_T$  є шуканим кістяком з  $n-1$  ребрами.

Звісно, алгоритм можна перервати раніше, коли буде вибрано  $n-1$  ребер для  $G_T$ . Алгоритм суттєво прискорюється, якщо спочатку ребра відсортувати за зростанням їх ваг за якимось ефективним алгоритмом

сортування. В результаті, складність алгоритму складає, в основному, складність сортування і оцінюється як  $O(m \cdot \log m)$ , де  $m = |E|$ .

**Приклад.** Використовуючи алгоритм Краскала, знайти мінімальний кістяк графа на рис.



Рішення

Ітерація	Знайдене ребро	Вага	$G_T$
1	(1,2)	10	
2	(6,3)	15	
3	(4,6)	20	
4	(2,6)	25	
5	(3,5)	35	

Результатуюча вага  $w(G_T) = 10 + 15 + 20 + 25 + 35 = 105$  одиниць і є мінімальною.

## 2.3 Алгоритм Пріма

Нехай  $G_T$  — дерево, яке послідовно будується за зв'язаним зваженим графом  $G$ . Спочатку  $G_T = \{\emptyset, \emptyset\}$  — пустий граф.

Крок 1: Почати з довільної вершини  $v_0$  в  $G$  і додати її до  $G_T$  так, що  $G_T = \{v_0\}, \emptyset\}$ .

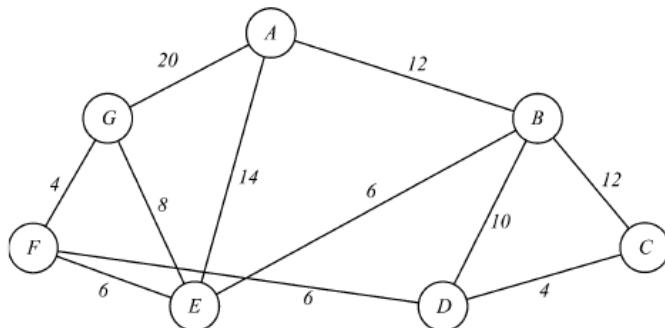
Крок-2: Знайти ребро  $e_{01} = (v_0, v_1) \in G$  серед ребер, які є інцидентні до вершини  $v_0$ , таке, що його вага мінімальна, тобто значення  $w(e_{01})$  — мінімальне. Додати  $v_1, e_1$  до  $G_T$ , тобто  $G_T = \{v_0, v_1\}, \{e_{01}\}\}$ .

Крок 3: Обрати наступне ребро  $e_{ij} = (v_i, v_j)$  таким чином, щоб вершина  $v_i$  знаходилася в  $G_T$ , вершина  $v_j$  не було в  $G_T$ , а вага  $w(e_{ij})$  — мінімальна. Додати  $v_j$  та  $e_{ij}$  до  $G_T$ .

Крок-4: Якщо  $G_T$  містить всі вершини  $G$ , то кінець. Інакше повторити крок-3.

Отже, протягом роботи алгоритму формується дерево, яке розростається, поки не охопить всі вершини початкового графа. На кожному кроці алгоритму до поточного дерева приєднується найлегше з ребер, що з'єднують вершину з побудованого дерева і вершину, що не належить дереву.

**Приклад.** Виконати алгоритм Пріма, щоб знайти мінімальний кістяк графа на рис.

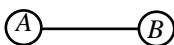
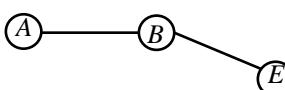
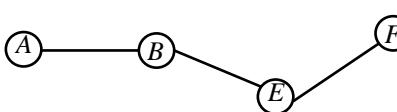
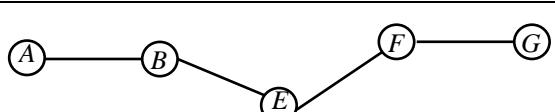
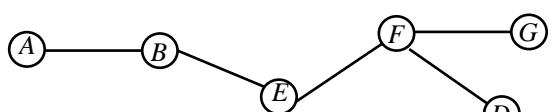
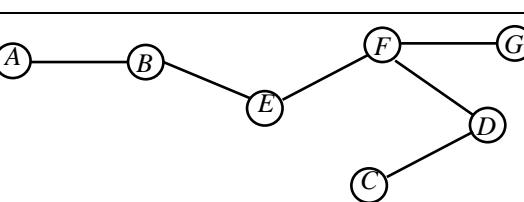


Матриця суміжності, в якій закодовані ваги, така:

$$X(G) = D \begin{bmatrix} A & B & C & D & E & F & G \\ A & 0 & 12 & \infty & \infty & 14 & \infty & 20 \\ B & 12 & 0 & 12 & 10 & 6 & \infty & \infty \\ C & \infty & 12 & 0 & 4 & \infty & \infty & \infty \\ D & \infty & 10 & 4 & 0 & \infty & 6 & \infty \\ E & 14 & 6 & \infty & \infty & 0 & 6 & 8 \\ F & \infty & \infty & \infty & 6 & 6 & 0 & 4 \\ G & 20 & \infty & \infty & \infty & 8 & 4 & 0 \end{bmatrix}$$

Хід побудови кістяка показано у таблиці

### Рішення

Iте-рація	Дії	Вага	$G_T$
1	$G_T = \{\{A\}, \emptyset\}$ .	12	
2	$G_T = \{\{A,B\}, (A,B)\}$ .	6	
3	$G_T = \{\{A,B,E\}, (A,B), (B,E)\}$ .	6	
4	$G_T = \{\{A,B,E,F\}, (A,B), (B,E), (E,F)\}$ .	4	
5	$G_T = \{\{A,B,E,F,G\}, (A,B), (B,E), (E,F), (F,G)\}$ .	4	
6	$G_T = \{\{A,B,E,F,G,D\}, (A,B), (B,E), (E,F), (F,G), (F,D)\}$ .	6	
7	$G_T = \{\{A,B,E,F,G,D,C\}, (A,B), (B,E), (E,F), (F,G), (F,D), (D,C)\}$ .	4	

Мінімальна вага в рядах  $A; B, E; F$  і  $G$  — 4. Але відповідні вершини  $F$  і  $G$  вже присутні в  $T$ . Наступний мінімум — 6 у рядку  $F$ , що відповідає стовпцю вершини  $D$ . Ми включаємо вершину  $D$  і ребро  $(F; D)$  до  $G_T$ .

Загальна вага  $w(G_T) = 12 + 6 + 6 + 4 + 6 + 4 = 38$  одиниць, яка є мінімальною вагою кістяка, що з'являється в рядку №7 таблиці.

## Орієнтовані, впорядковані і бінарні дерева

Орієнтовані (впорядковані) дерева є абстракцією ієрархічних відношень, які дуже часто зустрічаються як у практичному житті, так і в математиці та програмуванні. Дерево (орієнтоване) та ієрархія — це рівновеликі поняття.

### 1. Орієнтовані дерева

Орієнтованим деревом (або *ордеревом*, або *кореневим деревом*) називається орграф з наступними властивостями:

1. Існує єдина вершина  $r$ , напівстепінь заходу якого дорівнює 0,  $d^+(r) = 0$ .

Вона називається коренем ордерева.

2. Напівстепінь заходу всіх решти вершин дорівнює 1.

3. Кожна вершина досяжна з кореня,

Приклад. На рис. наведені діаграми всіх різних орієнтованих дерев з 4 вершинами.



ТЕОРЕМА. Ордерево має наступні властивості:

2. Якщо в ордереві забути орієнтацію дуг, то вийде вільне дерево.
3. В ордереві немає контурів.
4. Для кожної вершини існує єдиний шлях, що веде в цю вершину з кореня.
5. Підграф, що визначається множиною вершин, досяжних з вершини  $v$ , є ордеревом з коренем  $v$  (це ордерево називається піддеревом вершини  $v$ ).
6. Якщо у вільному дереві будь-яку вершину призначити коренем, то вийде ордерево.

НАСЛІДОК Алгоритм пошуку вглиб буде ордерево з коренем в початковій вершині.

Кінцева вершина ордерева називається *листком*. Множина листків називається *кроною*. Шлях з кореня в листок називається *гілкою*. Довжина найбільшої гілки ордерева називається його *висотою*. *Рівень вершини* ордерева — це відстань від кореня до вершини. Сам корінь має рівень 0. Вершини одного рівня утворюють *ярус* ордерева.

Поряд з «рослинною» застосовується ще і «генеалогічна» термінологія. Вершини, досяжні з вершини  $v$ , називаються *нащадками* вершини  $v$  (нащадки однієї вершини утворюють піддерево). Якщо вершина  $v$  є нащадком вершини  $u$ , то вершина  $u$  називається *предком* вершини  $v$ .

Якщо в дереві існує дуга  $(u, v)$ , то вершина  $u$  називається **батьком** вершини  $v$ , а вершина  $v$  називається **сином** вершини  $u$ . Сини одного батька називаються **братами**.

## 2. Еквівалентне визначення ордерева

**Ордерево**  $T$  — це непуста скінчена множина вершин, на якій визначено розбиття, що має наступні властивості:

1. Є один виділений одноелементний блок  $\{r\}$ , що називається коренем даного ордерева.
2. Решта вершин (за винятком кореня) містяться в блоках  $T_1, \dots, T_k$ , кожний з яких, в свою чергу, є ордеревом і називається **піддеревом**.

$$T \stackrel{\text{Def}}{=} \{\{r\}, T_1, \dots, T_k\}.$$

Не важко побачити, що дане визначення еквівалентне до визначення 1. Достатньо побудувати орграф, проводячи дуги від заданого кореня ордерева  $r$  до коренів піддерев  $T_1, \dots, T_k$  і далі повторюючи рекурсивно цей процес для кожного з піддерев.

## 3. Впорядковані дерева

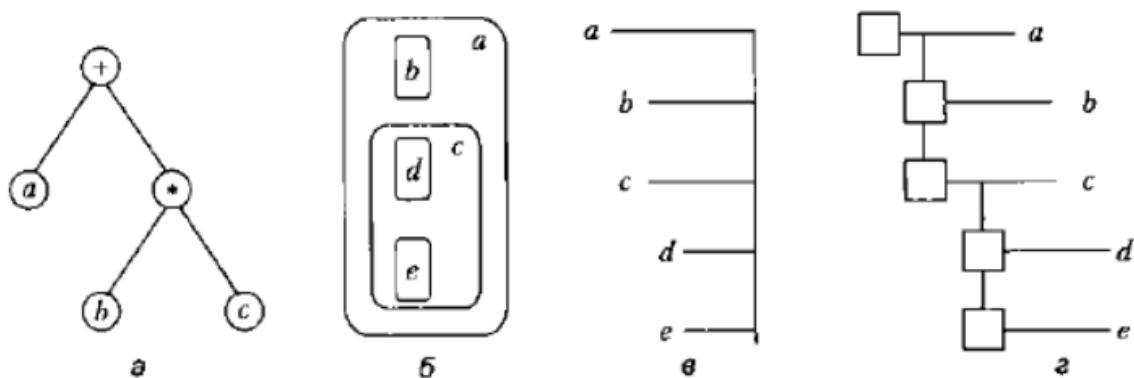
Якщо відносний порядок піддерев  $T_1, \dots, T_k$  в еквівалентному визначенні ордерева фіксований, то ордерево називається **впорядкованим**.

### Приклади

Орієнтовані і впорядковані орієнтовані дерева інтенсивно використовуються в програмуванні:

1. Вирази. Для представлення виразів мов програмування, як правило, використовуються орієнтовані впорядковані дерева. Приклад представлення виразу  $a + b * c$  показано на рис. 9.7, а.
2. Для представлення блочної структури програми і пов'язаної з нею структури областей визначення ідентифікаторів часто використовується орієнтоване дерево (можливо, невпорядковане, оскільки порядок визначення змінних в блоці у більшості мов програмування вважається несуттєвим). На рис. 9.7, б показана структура областей визначення ідентифікаторів  $a, b, c, d, e$ , причому для відображення ієархії використані вкладені області.
3. Для представлення ієархічної структури вкладеності елементів даних та/або операторів управління часто використовується техніка відступів, показана на рис. 9.7, в.
4. Структура вкладеності каталогів і файлів у сучасних операційних системах є впорядкованим орієнтованим деревом. Зазвичай для зображення таких дерев застосовується спосіб, показаний на рис. 9.7, г.

5. Різні «правильні структури дужок» (наприклад  $(a(b)(c(d)(e)))$ ) є орієнтованими впорядкованими деревами.



**Рис. 9.7. Примеры изображения деревьев в программировании**

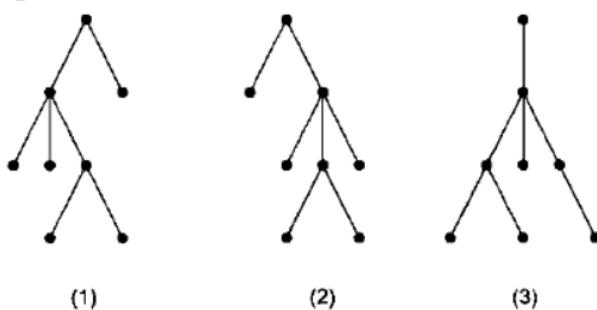
Рис. 9.7. Приклади зображення дерев у програмуванні

Той факт, що більшість систем управління файлами використовує орієнтовані дерева, відображається навіть в термінології, наприклад: «кореневий каталог диска».

#### ЗАУВАЖЕННЯ

Загальноприйнятою практикою при зображення дерев є домовленість про те, що корінь знаходиться згори і всі дуги орієнтовані згори вниз, тому стрілки можна не зображувати. Таким чином, діаграми вільних, орієнтованих і впорядкованих дерев виявляються графично нерозрізnenними, і потребується додаткове уточнення, дерево якого класу зображене на діаграмі. У більшості випадків це зрозуміло з контексту.

**Приклад** На рис. 9.8 наведені три діаграми дерев, які зовні виглядають різними. Як впорядковані дерева вони дійсно всі різні:  $(1) \neq (2)$ ,  $(2) \neq (3)$ ,  $(3) \neq (1)$ . як орієнтовані дерева  $(1) = (2)$ , але  $(2) \neq (3)$ . Як вільні дерева вони всі ізоморфні:  $(1) = (2) = (3)$ .



**Рис. 9.8. Діаграми дерев**

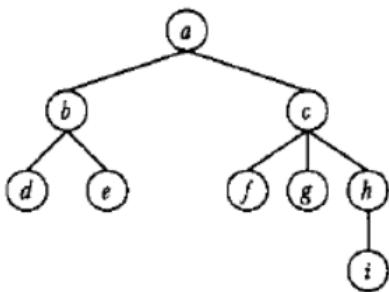
Вказана в підрозділі 2 побудова дає змогу визначити на впорядкованому ордереві єдиний лінійний порядок вершин, узгоджений з вже заданими в дереві впорядкуваннями.

**ТЕОРЕМА.** У впорядкованому ордереві з  $p$  вершинами існує така нумерація вершин числами з діапазону  $1 \dots p$ , що номери потомків більші за номери предків і номери старших братів більші за номери молодших братів.

**ДОВЕДЕННЯ.** Застосуємо до впорядкованого ордерева алгоритм обходу вглиб, починаючи з кореня, причому вершини, суміжні з даною, відвідуються в порядку, що визначається порядком піддерев. Присвоїмо вершинам номери в порядку першого відвідування. Всі вершини отримають унікальні номери з діапазону  $1 \dots p$  і корінь отримає номер 1. Далі, старші брати отримають номери, більші, ніж номери молодших братів за умовою обходу, а нащадки отримають номери більші, ніж номери предків, оскільки алгоритм обходу вглиб відвідує предків раніше, ніж нащадків.

**ЗАУВАЖЕННЯ.** Вказаний порядок обходу часто називають **прямим**.

**Приклад** Вершини впорядкованого ордерева на рис.



при прямому обході отримають наступні номери:  $a — 1, b — 2, c — 5, d — 3, e — 4, f — 6, g — 7, h — 8, i — 9$ .

Побудована в доведенні теореми нумерація не є єдиною нумерацією, що має вказані властивості.

**Приклад** Можна обійти впорядковане ордерево по ярусах. При цьому вершини впорядкованого ордерева на рис. отримають наступні номери:  $a — 1, b — 2, c — 3, d — 4, e — 5, f — 6, g — 7, h — 8, i — 9$ .

#### 4. Бінарні дерева

Бінарне (або двійкове) дерево — це непуста скінчена множина вершин, на якій визначена структура, що має наступні властивості:

1. Є одна виділена вершина  $r$ , що називається коренем даного бінарного дерева.
2. Решта вершин (за винятком кореня) містяться в двох неперетинних множинах (піддеревах) — **лівій** і **правій**, кожна з яких, в свою чергу, або пуста, або є бінарним деревом.

На перший погляд може здатися, що бінарне дерево — це окремий випадок впорядкованого орієнтованого дерева, в якому у кожної вершини не більш як дві суміжні. Але це не так, бінарне дерево не є впорядкованим ордеревом. Річ у тім, що навіть якщо у деякої вершини бінарного дерева є тільки одне непусте піддерево, то все одно відомо, яке саме це піддерево: ліве чи праве.

**Приклад** На рис. 9.9 наведені дві діаграми дерев, які є ізоморфними як впорядковані, орієтовані і вільні дерева, але не ізоморфні як бінарні дерева.



Рис. 9.9. Два різних бінарних дерева

**ЗАУВАЖЕННЯ** Поняття двійкового дерева допускає узагальнення, *m-ковим деревом* називається непуста скінчена множина вершин, яка складається з кореня і *m* неперетинних підмножин, що мають номери 1,..., *m*, кожна з яких в свою чергу, або пуста, або є *m*-ковим деревом. Більша частина тверджень і алгоритмів для двійкових дерев може бути порівняно легко поширені і на *m*-кові дерева (з відповідними модифікаціями).

## Подання дерев у програмах

Обговорення подання дерев ґрунтуються на тих же міркуваннях, що були зазначені при обговоренні подання графів. Крім того, слід підкреслити, що задача представлення дерев у програмі зустрічається значно частіше, ніж задача представлення графів загального виду, а тому методи її розв'язку виявляють ще більший вплив на практику програмування.

### 1. Подання вільних дерев

Для представлення дерев можна використовувати ті ж прийоми, що і для представлення графів загального виду — матриці суміжності й інциденцій, списки суміжності та ін. Але використовуючи особливі властивості дерев, можна запропонувати суттєво більш ефективні подання. Розглянемо наступне подання вільного дерева, відоме як **код Прюфера**.

Припустимо, що вершини дерева  $T(V,E)$  пронумеровані числами з інтервалу 1 ...*p*. Побудуємо послідовність  $A : \text{array}[1 \dots p-1] \text{ of } 1 \dots p$  у відповідності з алгоритмом 1.

**Алгоритм 1.** Побудова коду Прюфера вільного дерева

Початкові дані: Дерево  $T(V,E)$  в будь-якому поданні, вершини дерева пронумеровані числами 1 ...*p* довільним чином.

Результат: Масив  $A : \text{array}[1 \dots p-1] \text{ of } 1 \dots p$  — код Прюфера дерева  $T$ .

**for**  $i=1$  ;  $i \leq p-1$ ;  $i=i+1\{$

$v = \min(k \in V \mid d(k) = 1)$ ; // вибираємо вершину  $v$  — висячу вершину з  
// найменшим номером

$A[i] = T(v)$ ; // заносимо в код номер єдиної вершини, суміжної з  $v$

$V = V - v$  // видаляємо вершину  $v$  з дерева

}

За побудованим кодом можна відновити початкове дерево за допомогою алгоритму 2.

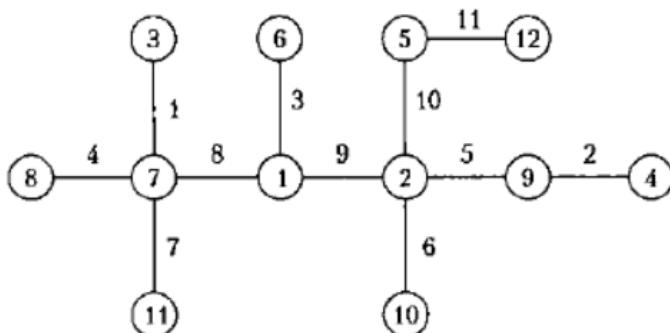


Рис.10. Побудова коду Прюфера

**Приклад.** Для дерева, представленого на рис. 10, код Прюфера 7,9,1,7,2,2,7, 1,2,5,12. На цьому рисунку числа в вершинах — це їх номери, а числа на ребрах вказують порядок, в якому будуть вибиратися висячі вершини і видалятися ребра при побудові коду Прюфера.

**Алгоритм 2.** Розпакування коду Прюфера вільного дерева

Початкові дані: Масив  $A : \text{array}[1..p-1] \text{ of } 1\dots p$  — код Прюфера дерева  $T$ .

Результат: Дерево  $T(V, E)$ , задане множиною ребер  $E$ , вершини дерева пронумеровані числами  $1 \dots p$ .

```

 $E = 0;$  //на початку множина ребер пуста
 $B = 1\dots p;$  //множина невикористаних номерів вершин
for  $i=1$  ;  $i \leq p-1$ ;  $i=i+1\{$ 
     $v = \min(k \in B \mid \forall j \geq i (k \neq A[j]));$  //вибираємо вершину  $v$  —
        //невикористану вершину з найменшим номером,
        //який не зустрічається в залишку коду Прюфера
     $E = E + (v, A[i]);$  // додаємо ребро  $(v, A[i])$ 
     $B = B - v;$  //видаляємо вершину  $v$  зі списку невикористаних
}
```

**ЗАУВАЖЕННЯ.** Код Прюфера — найбільш економне по пам'яті подання дерева. Його можна трохи покращити, якщо помітити, що існує всього одне дерево з двома вершинами, а тому інформацію про «останнє ребро» можна не зберігати, вона відновлюється однозначно.

## 2. Подання бінарних дерев

Всяке вільне дерево можна орієнтувати, призначивши одну з вершин коренем. Всяке ордерево можна довільно впорядкувати. Для нащадків однієї вершини (брать) впорядкованого ордерева визначено відношення старше-молодше (лівіше-правіше). Всяке впорядковане дерево можна представити бінарним деревом, наприклад, провівши правий зв'язок до старшого брата, а левий — до молодшого сина. Таким чином, достатньо розглянути подання в

програмі бінарних дерев. Це спостереження пояснює, чому представленню бінарних дерев в програмах традиційно приділяється особлива увага при навчанні програмування.

**Приклад.** На рис. 11 наведені діаграми впорядкованого і відповідного йому бінарного дерева.

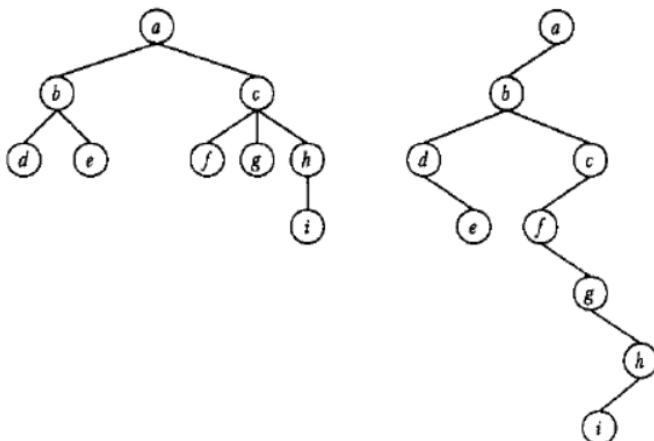


Рис. 11. Впорядковане і бінарне дерева

**ЗАУВАЖЕННЯ.** З даного подання випливає, що множина бінарних дерев взаємно-однозначно відповідає множині впорядкованих лісов впорядкованих одерев. Дійсно, у вказаному поданні одиночному впорядкованому одереву завжди відповідає бінарне дерево, у якого правий зв'язок кореня пустий, а впорядкованому лісу — бінарне дерево, у якого правий зв'язок кореня не пустий.

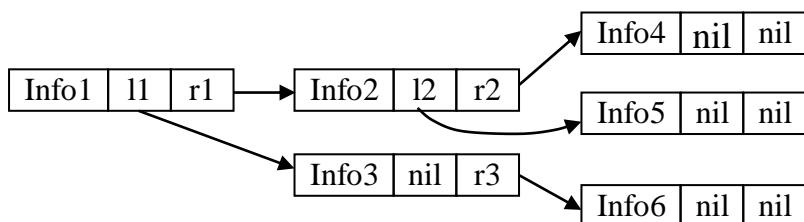
Позначимо через  $n(p)$  об'єм пам'яті, яку займає подання бінарного дерева, де  $p$  — кількість вершин. Наїбільш часто використовуються наступні подання бінарних дерев:

1. *Спискові структури*: кожна вершина представляється записом типу  $N$ , що містить два поля  $(l \ i \ r)$  з вказівниками на ліву і праву вершини і ще одне поле  $i$  для зберігання вказівника на інформацію про вершину. Дерево представляється вказівником на корінь. Тип  $N$  зазвичай визначається наступним чином:

$$N = record \{i : info; l, r: ^N\},$$

де тип  $info$  вважається заданим.

1



Для цього подання  $n(p) = 3p$ .

**ЗАУВАЖЕННЯ.** Оскільки в бінарному дереві, як і в будь-якому іншому,  $n = m - 1$ , то з  $2p$  вказівників, що відводяться для зберігання дуг,  $p+1$  вказівників завжди зберігають значення nil, тобто половина зв'язків не використовується.

**2. Упаковані масиви:** всі вершини розташовуються в масиві, так що всі вершини піддерева даної вершини розташовуються вслід за цією вершиною. Разом з кожною вершиною зберігається індекс вершини, яка є першою вершиною правого піддерева даної вершини. Дерево  $T$  зазвичай визначається наступним чином:

$T : \text{array [1...p]} \text{ of record } \{ i : \text{info}, k : 1...p \}$

де тип  $\text{info}$  вважається заданим. Для цього подання  $n(p) = 2p$ .

**3. Польський запис:** аналогічно, але замість зв'язків фіксується «розмічений степінь» кожної вершини (наприклад, 0 означає, що це листок, 1 — є лівий зв'язок, але немає правого, 2 — є правий зв'язок, але немає лівого, 3 — є обидва зв'язки). Дерево  $T$  визначається наступним чином:

$T : \text{array [1...p]} \text{ of record } \{ i : \text{info}, d : 0...3 \}$

де тип  $\text{info}$  вважається заданим. Для цього подання  $n(p) = 2p$ . Якщо степінь вершини відомий з інформації, що зберігається в самій вершині, то можна не зберігати і степінь. Такий спосіб подання дерев називається польським записом і зазвичай використовується для подання виразів. В цьому випадку подання дерева оказывается найбільш компактним: об'єм пам'яті  $n(p) = p$ .

Приклад. Покажемо, як виглядають в пам'яті бінарні дерева в різних поданнях. В наведеній нижче таблиці — кодування дерева на рис. 11 справа, умовні адреси — це цілі числа, а пустий вказівник — 0.

Адреса	Спискова структура			Упаковані масиви		Польський запис	
	$i$	$l$	$r$	$i$	$k$	$i$	$d$
1	$a$	2	0	$a$	0	$a$	1
2	$b$	3	5	$b$	5	$b$	3
3	$d$	0	4	$d$	4	$d$	2
4	$e$	0	0	$e$	0	$e$	0
5	$c$	6	0	$c$	0	$c$	1
6	$f$	0	7	$f$	7	$f$	2
7	$g$	0	8	$g$	8	$g$	2
8	$h$	9	0	$h$	0	$h$	1
9	$i$	0	0	$i$	0	$i$	0

### **3 Обходи бінарних дерев**

Більшість алгоритмів роботи з деревами основані на обходах. Можливі наступні основні обходи бінарних дерев.

**Прямий** (префіксний, лівий) обхід: потрапити в корінь,  
обійти ліве піддерево,  
обійти праве піддерево.

**Внутрішній** (інфіксний, симетричний) обхід: обійти ліве піддерево,  
потрапити в корінь,  
обійти праве піддерево.

**Кінцевий** (постфіксний, правий) обхід: обійти ліве піддерево,  
обійти праве піддерево,  
потрапити в корінь.

**Приклад** Кінцевий обхід дерева виразу  $a + b^*c$  дає зворотний польський запис цього виразу:  $abc^*+$ .

Польський запис виразів (прямий або зворотний) застосовується в деяких мовах програмування безпосередньо і використовується як внутрішнє подання програм у багатьох трансляторах та інтерпретаторах. Причина полягає в тому, що така форма запису допускає дуже ефективну інтерпретацію (обчислення значення) виразів. Наприклад, значення виразу в зворотному польському запису може бути обчислене при однократному перегляді виразу зліва направо з використанням одного стеку. В таких мовах, як Forth і PostScript, зворотний польський запис використовується як основний.

## **Дерева сортування**

В цьому розділі обговорюється одне конкретне застосування дерев у програмуванні, а саме дерева сортування (які також називають деревами упорядкування). При цьому розглядаються як теоретичні питання, пов'язані, наприклад, з оцінкою висоти дерев, так і практична реалізація алгоритмів, а також цілий ряд прагматичних аспектів застосування дерев сортування і деякі суміжні питання.

### **Асоціативна пам'ять**

В практичному програмуванні для організації зберігання даних і доступу до них часто використовується механізм, який зазвичай називають асоціативною пам'яттю. При використанні асоціативної пам'яті дані поділяються на порції (що називаються *записами*), і з кожним записом асоціюється ключ. **Ключ** — це значення з деякої лінійно впорядкованої множини, а записи можуть мати довільну природу та різні розміри. Доступ до даних здійснюється за значенням ключа, яке зазвичай вибирається простим, компактним і зручним для роботи.

## Приклади

Асоціативна пам'ять використовується в багатьох областях життя:

1. Тлумачний словник: записом є словника стаття, а ключем — заголовок словникової статті.
2. Адресна книга: ключем є ім'я абонента, а записом — адресна інформація (телефон(и), поштова адреса тощо).
3. Банківські рахунки: ключем є номер рахунку, а записом — фінансова інформація.

Таким чином, асоціативна пам'ять має підтримувати принаймні три основні операції:

- 1) додати (ключ, запис);
- 2) знайти (ключ, запис);
- 3) видалити (ключ).

Ефективністьожної операції залежить від структури даних, що використовується для подання асоціативної пам'яті. Ефективність асоціативної пам'яті в цілому залежить від співвідношення частоти виконання різних операцій у даній конкретній програмі.

## Способи реалізації асоціативної пам'яті

Для подання асоціативної пам'яті використовуються наступні основні структури даних:

- 1) невпорядкований масив;
- 2) впорядкований масив;
- 3) дерево сортування — бінарне дерево, кожна вершина якого містить ключ (і вказівник на запис) і має наступну властивість: значення ключа в усіх вершинах лівого піддерева менше, а в усіх вершинах правого піддерева — більше, ніж значення ключа в вершині;
- 4) таблиця розстановки (або хеш-таблиця).

При використанні невпорядкованого масиву алгоритми реалізації операцій асоціативної пам'яті очевидні:

1. Операція «додати (ключ, запис)» реалізується додаванням запису в кінець масиву.
2. Операція «знайти (ключ, запис)» реалізується перевіркою в циклі всіх записів в масиві.
3. Операція «видалити (ключ)» реалізується пошуком запису, який видаляється, а після цього переміщенням останнього запису на місце того, що видаляється.

Для впорядкованого масиву є ефективний алгоритм пошуку, описаний в наступному підрозділі.

## **Алгоритм бінарного (двійкового) пошуку**

При використанні впорядкованого масиву для подання асоціативної пам'яті операція пошуку запису по ключу може бути виконана за час  $O(\log_2 n)$  (де  $n$  — кількість записів) за допомогою наступного алгоритму, відомого як алгоритм бінарного (або двійкового) пошуку.

Вхід: впорядкований масив A: array [l..n] of record {k: key; i: info};  
ключ a : key.

Вихід: індекс запису з шуканим ключем a в масиві A або 0, якщо запису з таким ключем немає.

```
b = 1; //початковий індекс частини масиву для пошуку
e = n; //кінцевий індекс частини масиву для пошуку
while b ≤ e {
    c = (b + e)/2; //індекс елемента, який перевіряється (округлений)
    if A[c].k > a {
        e = z - 1; //продовжуємо пошук в першій половині
    }
    else if A[c].k < a {
        b = z + 1; //продовжуємо пошук в другій половині
    }
    else
        return c; // знайшли шуканий ключ
}
return 0; //шуканого ключа немає в масиві
```

**ОБГРУНТУВАННЯ.** Достатньо зауважити, що на кожному кроці основного циклу шуканий елемент масиву (якщо він є) знаходиться між (включно) елементами з індексами b і e. Оскільки діапазон пошуку на кожному кроці зменшується вдвічі, загальна трудомісткість не перевищує  $\log_2 n$ .

## **Алгоритм пошуку в дереві сортування**

Наступний алгоритм знаходить в дереві сортування вершину з вказаним ключем, якщо він там є.

Алгоритм. Пошук вершини в дереві сортування

Вхід: дерево сортування T, задане вказівником на корінь; ключ a : key.

Вихід: вказівник p на знайдену вершину або nil, якщо в дереві немає такого ключа.

```
p = T; //вказівник на перевірювану вершину
while p ≠ nil {
    if a < p.i {
        p = p.l; // продовжуємо пошук зліва
```

```

        }
    else if a > p.i {
        p := p.r; //продовжуємо пошук справа
    else
        return p; // знайшли вершину
    }
}

return nil; //шуканого ключа немає в дереві

```

Цей алгоритм працює в точній відповідності з визначенням дерева сортування: якщо поточна вершина не шукана, то в залежності від того, менше чи більше шуканий ключ у порівнянні з поточним, треба продовжувати пошук зліва або справа відповідно.

### **Алгоритм вставки в дерево сортування**

Наступний алгоритм вставляє в дерево сортування вершину зі вказаним ключем. Якщо вершина зі вказаним ключем вже є в дереві, то нічого не робиться.

Допоміжна функція NewNode по ключу a буде новий запис:

p.i = a; p.l = nil; p.r = nil;  
і повертає на неї вказівник p.

### **Алгоритм.** Вставка вершини в дерево сортування

Вхід: дерево сортування T, задане вказівником на корінь; ключ a : key.

Вихід: модифіковане дерево сортування T.

```

if T = nil {
    T = NewNode(a); // перша вершина в дереві
    return T;
}
p = T; // вказівник на поточну вершину
while true {
    if a = p.i {
        return T; // в дереві вже є такий ключ
    }
    if a < p.i {
        if p.l = nil{
            p.l := NewNode(a); // створюємо нову вершину
            return T; // і підчеплюємо її до p зліва
        }
    }
}

```

```

        else {
            p:=p.l; //продовж.пошук місця для вставки зліва
        }
    else {
        if a > p.i {
            if p.r = nil {
                p.r: = NewNode(a); //створюємо нову верш.
                return T; // і підчеплюємо її до p справа
            else{
                p: =p.r; // продовжуємо пошук місця для
                // вставки справа
            }
        }
    }
}

```

Алгоритм вставки, по суті, аналогічний алгоритму пошуку: в дереві шукається така вершина, що має вільний зв'язок для підчеплення нової вершини, щоб не порушувалась умова дерева сортування. А саме, якщо новий ключ менше поточного, то або його можна підчепити зліва (якщо лівий зв'язок вільний), або треба знайти зліва підходяще місце. Аналогічно, якщо новий ключ більше поточного.

### **Алгоритм видалення з дерева сортування**

Наступний алгоритм видаляє з дерева сортування вершину з вказаним ключем. Якщо вершини з вказаним ключем немає в дереві, то нічого не робиться. Допоміжні процедури Find і Delete знаходять вершину і видаляють її.

Алгоритм 9.10 Видалення вершини з дерева сортування

Вхід: дерево сортування T, задане вказівником на корінь ; ключ a : key.

Вихід: модифіковане дерево сортування T.

```

Find(T,a,p,q, s); //пошук вершини, яка видаляється, по ключу a,
                    // повернення: вказівник p на вершину, вказівник q на батька,
                    // s=-1 при  p зліва від q, s=1 – якщо справа і s=0 – якщо p –корінь
if p = nil {
    return T; // немає такої вершини — нічого робити не треба
}
if p.r = nil {
    Delete(p,q,p.l,s); // випадок 1, рис. 9.12, зліва
}
else {
    u: =p.r;
    if u.l = nil {
        u.l =p.l ;
        Delete(p, q, u, s); // випадок 2, рис. 9.12, в центрі
    }
}

```

```

        }
    else {
        w = u;
        v = u.l
        while v.l ≠ nil {
            w = v;
            v = v.l
        }
        p.i = v.i ;
        Delete(v,w,v.r, -1); //випадок 3, рис. 9.12, справа
    }
}

```

Видалення вершини виконується перебудовою дерева сортування. При цьому можливі три випадки (не рахуючи тривіального випадку, коли вершини, що видаляється, немає в дереві і нічого робити не треба).

return T

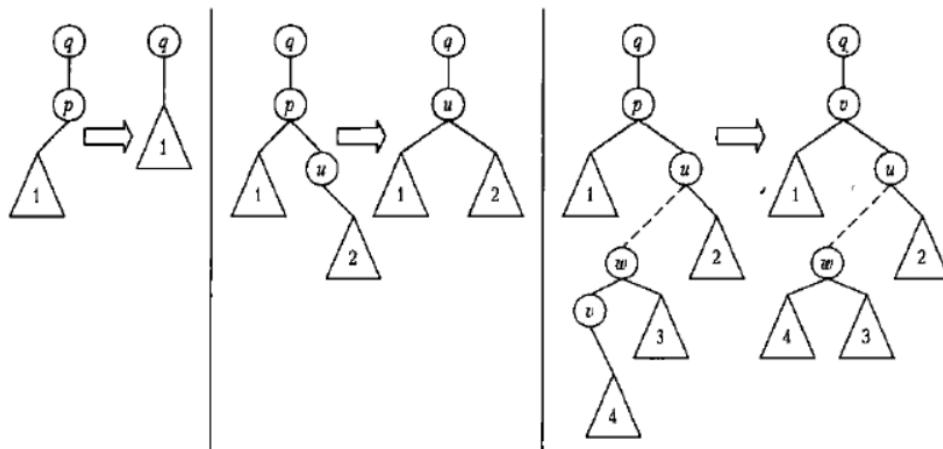


Рис. 9.12. Иллюстрация к алгоритму удаления узла из дерева сортировки

[ 1 ] Правий зв'язок вершини  $p$ , що видаляється, пустий (див. рис. 9.12, зліва). В цьому випадку ліве піддерево 1 вершини  $p$  підчеплюється до батьківської вершини  $q$  з того ж боку, з якого була підчеплена вершина  $p$ . Умова дерева сортування, очевидно, виконується.

[ 2 ] Правий зв'язок вершини  $p$ , що видаляється, не пустий і веде в вершину  $u$ , лівий зв'язок якого пустий (див. рис. 9.12, в центрі). В цьому випадку ліве піддерево 1 вершини  $p$  підчеплюється до вершини  $u$  зліва, а сама вершина  $u$  підчеплюється до батьківської вершини  $q$  з того ж боку, з якого була підчеплена вершина  $p$ . Нескладно перевірити, що умова дерева сортування виконується і в цьому випадку.

[ 3 ] Правий зв'язок вершини  $p$ , що видаляється, не пустий і веде в вершину  $u$ , лівий зв'язок якого не пустий. Оскільки дерево сортування скінченне, можна спуститися від вершини  $u$  до вершини  $v$ , лівий зв'язок якого

пустий (див. рис. 9.12, справа). У цьому випадку виконуються два перетворення дерева. Спочатку інформація у вершині  $v$  замінюється інформацією вершини  $v$ . Оскільки вершина  $v$  знаходитьться в правому піддереві вершини  $r$  ів лівому піддереві вершини  $u$ , маємо  $r.i < v.i < u.i$ . Таким чином, після цього перетворення умова дерева сортування виконується. Далі праве піддерево  $4$  вершини  $v$  підчеплюється зліва до вершини  $w$ , а сама вершина  $v$  видаляється. Оскільки піддерево  $4$  входило в ліве піддерево вершини  $k$ , умова дерева сортування також зберігається.

### Порівняння представлень асоціативної пам'яті

Нехай  $n$  — кількість елементів в асоціативній пам'яті. Тоді складність операцій для різних представлень обмежена згори наступним чином.

Невпорядкований масив	Впорядкований масив	Дерево сортування
Додати $O(T)$	$O(n)$	$O(\log_2(n)) \dots O(n)$
Знайти $O(n)$	$O(\log_2(n))$	$O(\log_2(n)) \dots O(n)$
Видалити $O(n)$	$O(n)$	$O(\log_2(n)) \dots O(n)$

Ефективність операцій з деревом сортування обмежена згори висотою дерева.

### Вирівняні та повні дерева

Бінарне дерево називається **вирівняним**, якщо всі листки знаходяться на одному (останньому) рівні. У вирівняному дереві всі гілки мають одну довжину, рівну висоті дерева, однак вирівняне дерево не завжди є ефективним деревом сортування.

**Приклад** Дерево, що складається з кореня і двох піддерев, ліволінійного і праволінійного, які ведуть до двох листків, є вирівняним, однак таке дерево має висоту  $(p - 1)/2$ .

Бінарне дерево називається **заповненим**, якщо всі вершини, степінь яких менше 2, розташовуються на одному або двох останніх рівнях. Іншими словами, в заповненому дереві  $T$  всі яруси  $D(r,i)$ , крім, можливо, останнього заповнені.

**Приклад** На рис. 9.13 наведені діаграми заповненого (зліва) і незаповненого (справа) дерев.



**Рис. 9.13. Заповнене и незаповнене деревья**

Заповнене дерево має найменшу можливу для даного  $p$  висоту  $h$ .

**ТЕОРЕМА** Для заповненого бінарного дерева  $\log_2(p + 1) - 1 \leq h < \log_2(p + 1)$ .

Вирівняне заповнене бінарне дерево називається **повним**. В повному бінарному дереві всі листки знаходяться на останньому рівні і всі вершини, крім листків, мають напівстепінь виходу 2. Іншими словами, в повному дереві всі яруси заповнені.

**НАСЛІДОК** Повне бінарне дерево висотою  $h$  має  $2^{h+1} - 1$  вершин.

Іноді **повним** називають бінарне дерево, в якому всі нелисткові вершини мають напівстепінь виходу 2, але листки можуть траплятися на будь-якому рівні. Висота дерева, повного в такому розумінні, може змінюватися в широких межах.

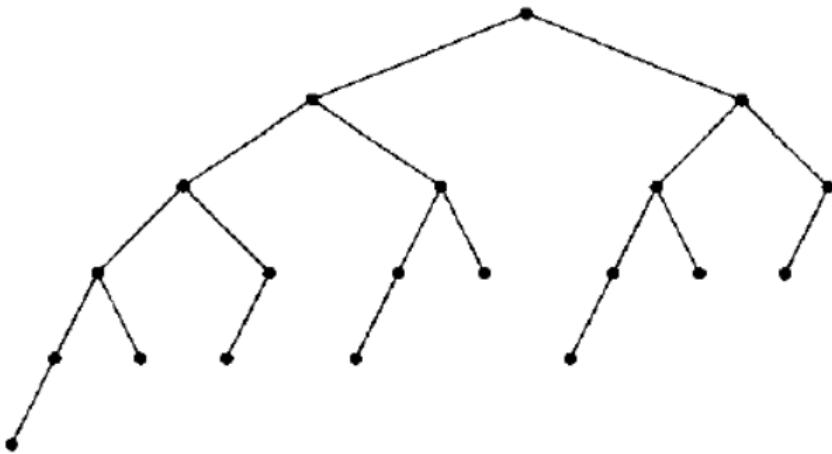
### ЗАУВАЖЕННЯ

Заповнені дерева дають найбільший можливий ефект при пошуку. Однак відомо, що вставка/видалення в заповнене дерево може вимагати повної перебудови всього дерева, і, таким чином, трудомісткість операції в найгіршому випадку складе  $O(p)$ .

### Збалансовані дерева

(Бінарне) дерево називається АВЛ-деревом (Адельсон-Вельський і Ландіс, 1962), або **збалансованим за висотою деревом**, якщо для будь-якої вершини висоти лівого і правого піддерев відрізняються не більш ніж на 1.

**Приклад** На рис. 9.14 наведена діаграма максимально несиметричного збалансованого по висоті дерева, в якому для всіх вершин висота лівого піддерева рівно на 1 більше висоти правого піддерева.



**Рис. 9.14.** Сбалансированное дерево

**ТЕОРЕМА** Для збалансованого по висоті бінарного дерева  $h < 2 \log_2 p$ .

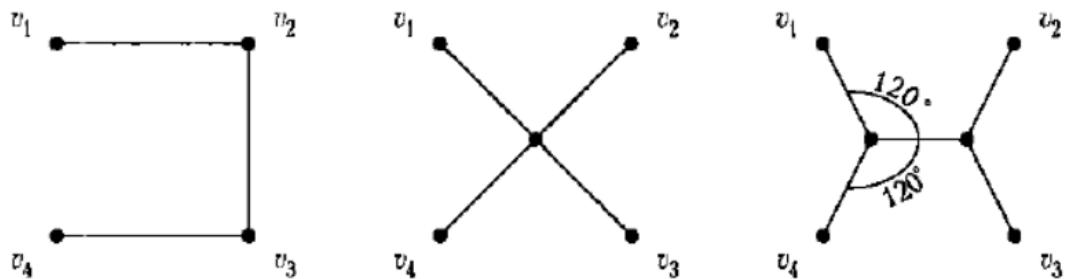
Збалансовані дерева поступаються заповненим деревам по швидкості пошуку (менше ніж у два рази), однак їх перевага полягає в тому, що відомі алгоритми вставки вершин в збалансоване дерево і видалення їх з нього, які зберігають збалансованість і в той же час при перебудові дерева зачіпають тільки скінченну кількість вершин. Тому в багатьох випадках збалансоване дерево виявляється найкращим варіантом подання дерева сортування.

## Дерево Штейнера

Задача про знаходження найкоротшого кістяка належить до числа небагатьох задач теорії графів, які можна вважати повністю розв'язаними. Між тим, якщо змінити умови задачі, на перший погляд навіть незначно, то вона виявляється значно більш важкою.

**Задача Штейнера.** Нехай задана множина міст на карті і треба визначити мінімальний (по сумі відстаней) набір залізничних ліній, який би дав змогу переїхати з будь-якого міста в будь-яке інше. Результатом є **дерево Штейнера**, в якому, на відміну від кістяка, допускається додавання проміжних вершин, що називаються **точками Штейнера**.

Найкоротший кістяк повного графа відстаней між містами не буде розв'язком цієї задачі. На рис. 9.18 наведені, відповідно, діаграми найкоротшого кістяка, наївного «розв'язку» задачі Штейнера і правильного розв'язку для випадку, коли міста розташовані у вершинах квадрата.



**Рис. 9.18.** Кратчайший остов, приближённое и точное решения задачи Штейнера

Задача Штейнера на площині добре досліджена, зокрема, відомо багато важливих властивостей точного розв'язку. Наприклад, відомо, що кожна точка Штейнера має степінь 3 і відрізки в ней зустрічаються під кутом  $120^\circ$ . Тим не менш досі не відомо достатньо ефективних алгоритмів розв'язку даної задачі, і вона лишається предметом інтенсивних досліджень.

## Цикли і розрізи

### Базисні цикли

Розглянемо кістяк  $T$  зв'язаного графа  $G$ . Нехай гілки  $T$  позначені як  $b_1, b_2, \dots, b_{n-1}$ , і нехай хорди  $T$  позначаються як  $c_1, c_2, \dots, c_{m-n+1}$ , де  $m$  — кількість ребер в  $G$ , а  $n$  — кількість вершин у  $G$ . Хоча  $T$  є ациклічним, за теоремою про кістяк і хорди граф  $T \cup c_i$  містить рівно один цикл  $C_i$ . Цей цикл складається з хорди  $c_i$  і тих гілок  $T$ , що лежать в унікальному шляху в  $T$  між кінцевими вершинами  $c_i$ . Цикл  $C_i$  називається **базисним циклом** графа  $G$  по відношенню до хорди  $c_i$  кістяка  $T$ .

Множина усіх  $m - n + 1$  базисних циклів  $C_i$  з  $G$  по відношенню до хорд кістяка  $T$  називається **базисною множиною циклів** графа  $G$  відносно до  $T$ .

Важливою особливістю базисного циклу  $C_i$  є те, що він містить рівно одну хорду, а саме, хорду  $c_i$ . Причому хорда  $c_i$  не міститься в жодному іншому базисному циклі по відношенню до  $T$ . Далі буде показано, що будь-який цикл у графі крім базисного можна представити як комбінацію з базисних циклів, а самі базисні цикли таким способом представити не можна. Це пояснює походження їх назви.

Граф  $G$  і набір основних ланцюгів  $G$  показані на рис. 2.4.

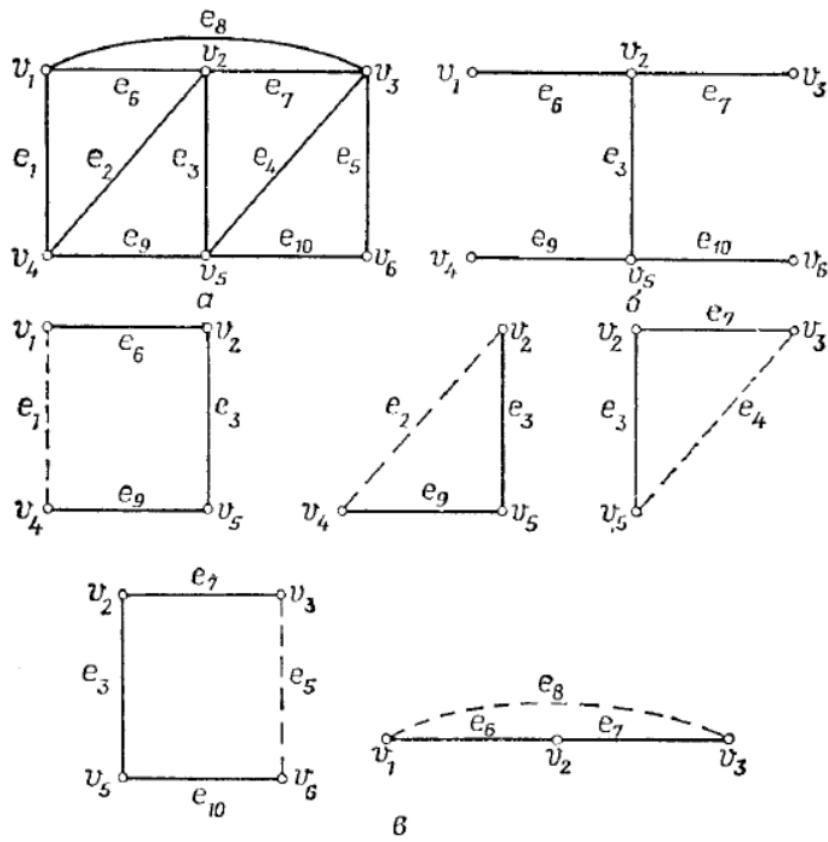


Рис. 2.4. Множество базисних циклов графа  $G$ .  
 а—граф  $G$ ; б—остов  $T$  графа  $G$ ; в—пять базисных циклов графа  $G$  по отношению к  $T$  (хорды обозначены штриховыми линиями).

## 2.5 Розрізувальні множини

До розрізувальної множини  $S$  зв'язного графа  $G$  відноситься така мінімальна множина ребер  $G$ , що видалення їх з графа  $G$  розділяє останній, тобто граф  $G - S$  стає незв'язним.

Наприклад, розглянемо підмножину  $S_1 = \{e_1, e_3, e_7, e_{10}\}$  ребер графа  $G$  (рис. 2.5, а). Після видалення  $S_1$  з  $G$  отримуємо  $G_1 = G - S$  (рис. 2.5, б). Граф  $G_1$  — незв'язний. Крім того, видалення будь-якої власної підмножини  $S_1$  не може перетворити граф  $G$  у незв'язний. Таким чином,  $S_1$  — розрізувальна множина графа  $G$ .

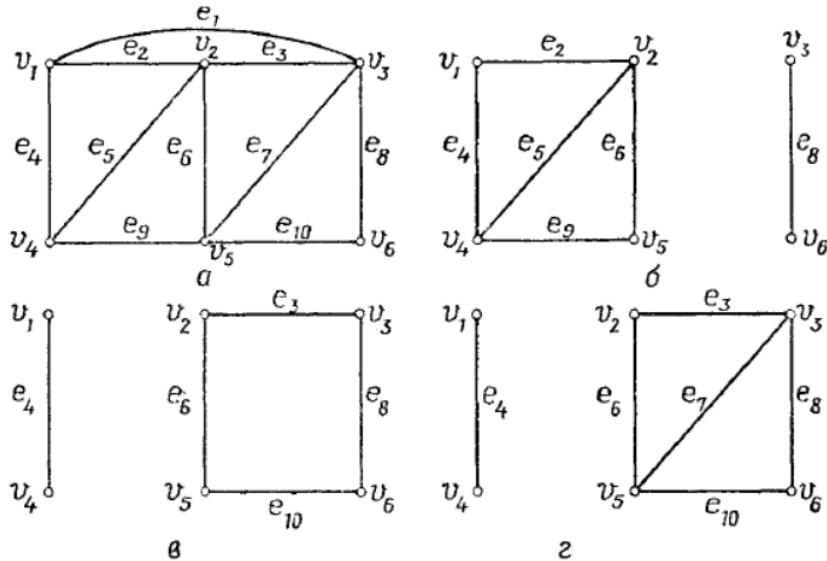


Рис. 2.5. Иллюстрация определения разрезающего множества.

$a$  — граф  $G$ ;  $b$  —  $G_1 = G - S$ ,  $S = \{e_1, e_3, e_7, e_{10}\}$ ;  $c$  —  $G_2 = G - S_2$ ,  $S_2 = \{e_1, e_2, e_5, e_7, e_9\}$ ;  
 $d$  —  $G_3 = G - S'_2$ ,  $S'_2 = \{e_1, e_2, e_5, e_9\}$ .

граф  $G$  в несвязный. Таким образом,  $S_1$  — разрезающее множество графа  $G$ .

Рассмотрим теперь множество  $S_2 = \{e_1, e_2, e_5, e_7, e_9\}$ . Граф  $G_2 = G - S_2$  (рис. 2.5,  $c$ ) — несвязный. Однако множество  $S'_2 = \{e_1, e_2, e_5, e_9\}$ , являющееся собственным подмножеством  $S_2$ , также превращает граф  $G$  в несвязный. Граф  $G_3 = G - S'_2$  показан на рис. 2.5,  $d$ . Следовательно,  $S_2$  не будет разрезающим множеством графа.

Заметим, что по определению разрезающего множества, данного выше, если  $S$  является разрезающим множеством графа  $G$ , то ранги  $G$  и  $G - S$  отличаются по крайней мере на единицу, т. е.  $p(G) - p(G - S) \geq 1$ . В работе [2.1] приводится следующее определение разрезающего множества:

Разрезающим множеством  $S$  связного графа  $G$  является такое минимальное множество ребер  $G$ , что удаление  $S'$  разбивает граф  $G$  на две компоненты, т. е.  $p(G) - p(G - S) = 1$ . Возникает вопрос: эквивалентны ли эти два определения? Ответ — положителен, а доказательство оставлено в качестве упражнения 2.15.

Consider next the set  $S_2 = \{e_1, e_2, e_5, e_7, e_9\}$ . The graph  $G_2 = G - S_2$  shown in Fig. 2.5c is disconnected. However, the set  $S'_2 = \{e_1, e_2, e_5, e_9\}$ , which is a proper subset of  $S_2$ , also disconnects  $G$ . The graph  $G_3 = G - S'_2$  is shown in Fig. 2.5d. Thus  $S_2$  is not a cutset of  $G$ .

Note that, by the definition of a cutset given above, if  $S$  is a cutset of a graph  $G$ , then the ranks of  $G$  and  $G - S$  differ by at least 1; that is,  $p(G) - p(G - S) \geq 1$ .

Seshu and Reed [2.2] define a cutset as follows: A cutset  $S$  of a connected graph  $G$  is a minimal set of edges of  $G$  such that removal of  $S$  connects  $G$  into exactly two components; that is,  $p(G) - p(G - S) = 1$ .

The question now arises whether these two definitions of a cutset are equivalent. The answer is "yes" and its proof is left as an exercise (Exercise 2.15)

## 2.6. Разрез

Определим понятие «разрез» которое связано с понятием «разрезающее множество».

Рассмотрим связный граф  $G$  с множеством вершин  $V$ . Пусть  $V_1$  и  $V_2$  — два таких непересекающихся подмножества множества  $V$ , что  $V = V_1 \cup V_2$ , т. е.  $V_1$  и  $V_2$  не имеют общих вершин, а вместе содержат все вершины множества  $V$ . Тогда множество  $S$  всех тех ребер, которые имеют одну вершину в  $V_1$ , а другую — в  $V_2$ , называется *разрезом* графа  $G$ . Разрез обычно обозначается через  $\langle V_1, V_2 \rangle$ . В работе [2.2] разрез называется *разделителем* (множество ребер, разделяющих множество вершин  $V$ ).

### 2.6 CUTS

We now define the concept of a cut, which is closely to that of a cutset.

Consider a connected graph  $G$  with vertex set  $V$ . Let  $V_1$  and  $V_2$  be two mutually disjoint subsets of  $V$  such that  $V = V_1 \cup V_2$ ; that is,  $V_1$  and  $V_2$  have no common vertices and together contain all the vertices of  $V$ . Then the set  $S$  of all those edges of  $G$  having one end vertex in  $V_1$ , and the other in  $V_2$  is called a cut of  $G$ . This is usually denoted by  $(V_1, V_2)$ . Reed [2.3] refers to a cut as a seg (the set of edges of segregating the vertex set  $V$ ).

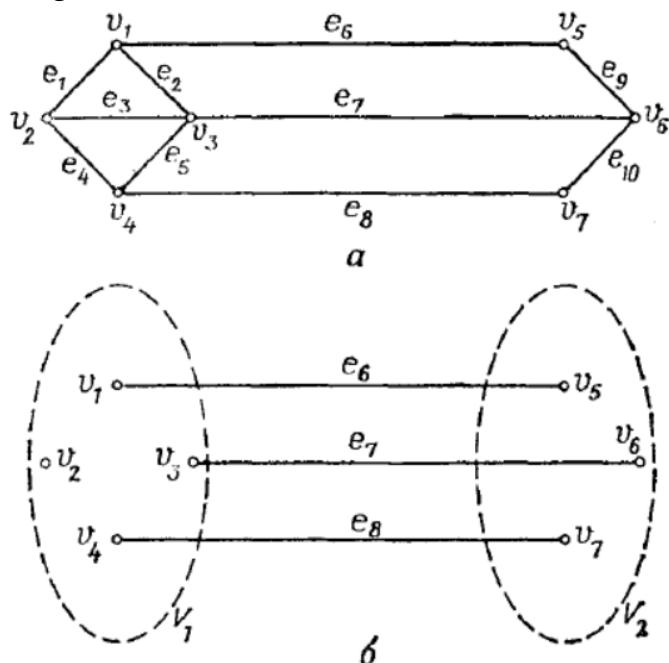


Рис. 2.6. Определение разреза.  
а — граф  $G$ ; б — разрез  $\langle V_1, V_2 \rangle$  графа  $G$ .

В качестве примера рассмотрим граф  $G$ , изображенный на рис. 2.6. Если  $V_1 = \{v_1, v_2, v_3, v_4\}$  и  $V_2 = \{v_5, v_6, v_7\}$ , то разрез  $\langle V_1, V_2 \rangle$  — это множество ребер  $\{e_6, e_7, e_8\}$ .

Отметим, что разрез  $\langle V_1, V_2 \rangle$  графа  $G$  является минимальным множеством ребер, удаление которых из графа  $G$  разбивает исходный граф на два графа  $G_1$  и  $G_2$ , являющиеся порожденными подграфами на множествах вершин  $V_1$  и  $V_2$  соответственно. Графы  $G_1$  и  $G_2$  могут быть несвязными. Если оба эти графа связные, то  $\langle V_1, V_2 \rangle$  по-прежнему минимальное множество ребер, разделяющих граф  $G$  точно на две компоненты. Тогда по определению  $\langle V_1, V_2 \rangle$  — есть разрезающее множество графа  $G$ .

Предположим, что для разрезающего множества  $S$  графа  $G$  величины  $V_1$  и  $V_2$  являются соответственно множествами вершин двух компонент  $G_1$  и  $G_2$  графа  $G - S$ . Тогда  $S$  есть разрез  $\langle V_1, V_2 \rangle$ . Таким образом, имеет место следующая теорема:

For example, for the graph  $G$  shown in Fig. 2.6, if  $V_1 = \{v_1, v_2, v_3, v_4\}$  and  $V_2 = \{v_5, v_6, v_7\}$ , then the cut  $(V_1, V_2)$  of  $G$  is equal to the set  $\{e_6, e_7, e_8\}$  of edges.

Note that the cut  $(V_1, V_2)$  of  $G$  is the minimal set of edges of  $G$  whose removal disconnects  $G$  into two graphs  $G_1$  and  $G_2$ , which are induced subgraphs of  $G$  on the vertex sets  $V_1$  and  $V_2$ .  $G_1$  and  $G_2$  may not be connected. If both these graphs are connected, then  $(V_1, V_2)$  is also the minimal set of edges disconnecting  $G$  into exactly two components. Then, by definition,  $(V_1, V_2)$  is a cutset of  $G$ .

Suppose that for a cutset  $S$  of  $G$ ,  $V_1$  and  $V_2$  are, respectively, the vertex sets of the two components  $G_1$  and  $G_2$  of  $G - S$ . Then  $S$  is the cut  $(V_1, V_2)$ . Thus we have the following theorem.

**Теорема 2.6.** 1. Разрез  $\langle V_1, V_2 \rangle$  связного графа  $G$  есть разрезающее множество графа  $G$ , если соответствующие подграфы  $G_1$  и  $G_2$ , порожденные на множествах вершин  $V_1$  и  $V_2$ , — связные. 2. Если  $S$  — разрезающее множество связного графа  $G$ , а  $V_1$  и  $V_2$  — множества вершин двух компонент  $G - S$ , то  $S = \langle V_1, V_2 \rangle$ .

Любой разрез  $\langle V_1, V_2 \rangle$  в связном графе  $G$  содержит разрезающее множество, так как удаление  $\langle V_1, V_2 \rangle$  из графа  $G$  переводит последний в несвязный граф. Фактически мы можем доказать, что разрез в графе  $G$  является объединением нескольких реберно-непересекающихся разрезающих множеств графа  $G$ . Формально мы утверждаем это в следующей теореме:

**Теорема 2.7.** Разрез в связном графе  $G$  есть объединение нескольких реберно-непересекающихся разрезающих множеств графа  $G$ . Доказательство этой теоремы несложно, поэтому оставим его в качестве упражнения 2.19.

### Theorem 2.6

1. A cut  $(V_1, V_2)$  of a connected graph  $G$  is a cutset of  $G$  if the induced subgraphs of  $G$  on vertex sets  $V_1$  and  $V_2$  are connected.

2. If  $S$  is a cutset of a connected graph  $G$ , and  $V_1$  and  $V_2$  are the vertex sets of the two components of  $G - S$ , then  $S = \langle V_1, V_2 \rangle$ . •

Any cut  $(V_1, V_2)$  in a connected graph  $G$  contains a cutset of  $G$ , since the removal of  $\{V_1, V_2\}$  from  $G$  disconnects  $G$ . In fact, we can prove that a cut in a graph

$G$  is the union of some edge-disjoint cutsets of  $G$ . Formally, we state this in the following theorem.

**Theorem 2.7.** A cut in a connected graph  $G$  is a cutset or union of edge-disjoint cutsets of  $G$ .

Рассмотрим вершину  $v_1$  в связном графе  $G$ . Множество ребер, инцидентных вершине  $v_1$ , образует разрез  $\langle v_1, V - v_1 \rangle$ . Удаление этих ребер разбивает граф  $G$  на два подграфа. Один из них, содержащий только одну вершину  $v_1$ , является связным по определению. Другой является порожденным подграфом  $G'$  графа  $G$  на множестве вершин  $V - v_1$ . Поэтому разрез  $\langle v_1, V - v_1 \rangle$  является разрезающим множеством тогда и только тогда, когда подграф  $G'$  является связным. Но подграф  $G'$  является связным в том и только в том случае, когда  $v_1$  не является точкой сочленения (разд. 1.7). Таким образом, имеем следующую теорему:

The proof of this theorem is not difficult, and it is left as an exercise (Exercise 2.19).

Consider next a vertex  $v_1$  in a connected graph. The set of edges incident on  $v_1$  forms the cut  $(v_1, V - v_1)$ . The removal of these edges disconnects  $G$  into two subgraphs. One of these subgraphs containing only the vertex  $v_1$  is, by definition, connected. The other subgraph is the induced subgraph  $G'$  of  $G$  on the vertex set  $V - v_1$ . Thus the cut  $(v_1, V - v_1)$  is a cutset if and only if  $G'$  is connected. However,  $G'$  is connected if and only if  $v_1$  is not a cut-vertex (Section 1.7). Thus we have the following theorem.

**Теорема 2.8.** Множество ребер, инцидентных вершине  $v$  в связном графе  $G$ , есть разрезающее множество тогда и только тогда, когда  $v$  не является точкой сочленения в графе  $G$ . В качестве примера рассмотрим разделимый граф  $G$ , представленный на рис. 2.7, а.  $v_1$  — точка сочленения графа  $G$ . Подграф графа  $G$ , порожденный на множестве вершин  $V - v_1 = \{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ , изображен на рис. 2.7, б. Этот подграф состоит из трех компонент и не связан. Поэтому ребра, инцидентные точке сочленения  $v_1$ , не образуют разрезающего множества графа  $G$ .

**Theorem 2.8.** The set of edges incident on a vertex  $v$  in a connected graph

$G$  is a cutset of  $G$  if and only if  $v$  is not a cut-vertex of  $G$ . •

For example, consider the separable graph  $G$  shown in Fig. 2.7a in which  $v_1$  is a cut-vertex. The induced subgraph of  $G$  on the vertex set  $V - v_1 = \{v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$  is shown in Fig. 2.7b. This subgraph consists of three components and is not connected. Thus the edges incident on the cut-vertex  $v_1$  do not form a cutset of  $G$ .

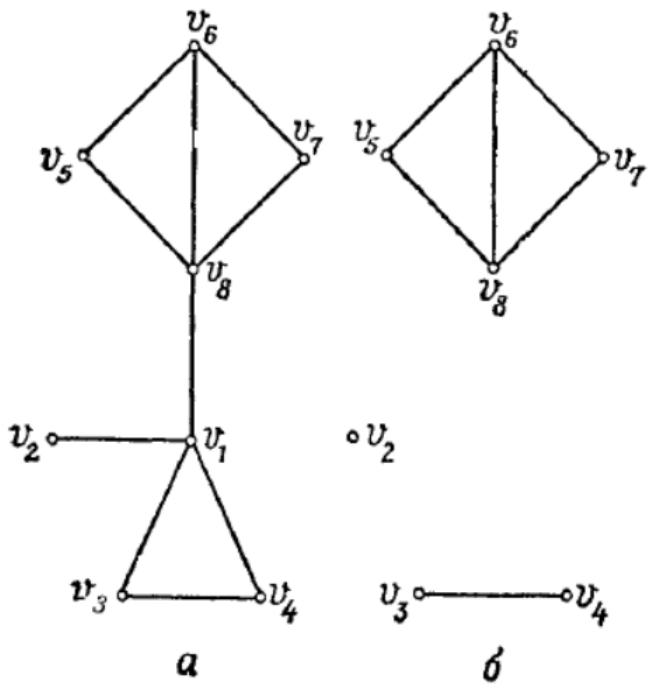


Рис. 2.7. Иллюстрация теоремы 2.8.  
а—граф  $G$ ; б—подграф графа  $G$ , по-  
рожденный на множестве вершин  $\{v_2, v_3,$   
 $v_4, v_5, v_6, v_7, v_8\}$ .

## 2.7. Базисные разрезающие множества

В разд. 2.4 было показано, что остов связного графа можно использовать для получения множества базисных циклов в графе. В данном разделе мы показываем, как с помощью остова можно определить множество базисных разрезающих множеств.

Рассмотрим остов  $T$  связного графа  $G$ . Пусть  $b$  — ветвь  $T$ . Удаление ветви  $b$  разбивает  $T$  на две компоненты  $T_1$  и  $T_2$ . Следует отметить, что  $T_1$  и  $T_2$  являются деревьями графа  $G$ . Пусть  $V_1$  и  $V_2$  обозначают соответственно такие множества вершин  $T_1$  и  $T_2$ , что  $V_1$  и  $V_2$  вместе содержат все вершины графа  $G$ .

Пусть  $G_1$  и  $G_2$  будут соответственно порожденными подграфами графа  $G$  на множествах вершин  $V_1$  и  $V_2$ . Не трудно видеть, что  $T_1$  и  $T_2$  являются остовами  $G_1$  и  $G_2$ . Следовательно, по теореме 2.3  $G_1$  и  $G_2$  — связные графы. В теореме 2.6 доказывается, что разрез  $\langle V_1, V_2 \rangle$  является разрезающим множеством графа  $G$ . Это разрезающее множество называется *базисным множеством* графа  $G$  по отношению к ветви  $b$  остова  $T$  графа  $G$ . Множество всех  $n-1$  базисных разрезающих множеств по отношению к  $n-1$ -ветви остова  $T$  связного графа  $G$  называется *базисным множеством разрезающих множеств* графа  $G$  по отношению к остову  $T$ .

Следует отметить, что разрезающее множество  $\langle V_1, V_2 \rangle$  содержит точно одну ветвь, т. е. ветвь  $b$  остова  $T$ . Все другие ребра  $\langle V_1, V_2 \rangle$  являются хордами  $T$ . Это следует из того, что  $\langle V_1, V_2 \rangle$  не содержит ни одного ребра из остовов  $T_1$  и  $T_2$ . Далее, ветвь  $b$  не присутствует ни в одном базисном разрезающем множестве по отношению к  $T$ . Из этого следует, что множество ребер базисного разрезающего множества нельзя представить в виде кольцевой суммы множеств ребер нескольких или всех базисных разрезающих множеств. В гл. 4 мы покажем, что каждое разрезающее множество графа  $G$  можно представить кольцевой суммой нескольких базисных разрезающих множеств по отношению к остову  $T$  графа  $G$ .

Граф  $G$  и множество его базисных разрезающих множеств представлены на рис. 2.8.

## 2.7 FUNDAMENTAL CUTSETS

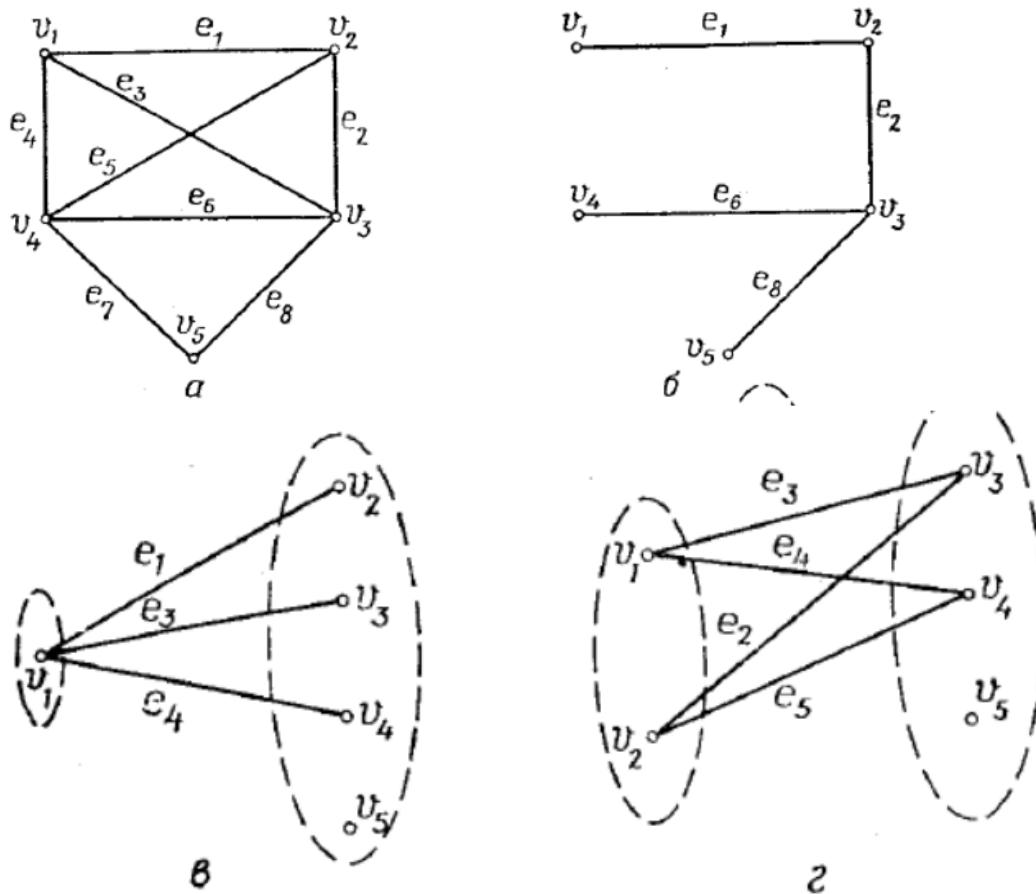
It was shown in Section 2.4 that a spanning tree of a connected graph can be used to obtain a set of fundamental circuits of the graph. We show in this section how a spanning tree can also be used to define a set of fundamental cutsets.

Consider a spanning tree  $\Gamma$  of a connected graph  $G$ . Let  $b$  be a branch of  $\Gamma$ . Now, removal of the branch  $b$  disconnects  $\Gamma$  into exactly two components  $T_1$  and  $T_2$ . Note that  $T_1$  and  $T_2$  are trees of  $G$ . Let  $V_1$  and  $V_2$ , respectively, denote the vertex sets of  $T_1$  and  $T_2$ .  $V_1$  and  $V_2$  together contain all the vertices of  $G$ .

Let  $G_1$  and  $G_2$  be, respectively, the induced subgraphs of  $G$  on the vertex sets  $V_1$  and  $V_2$ . It can be seen that  $T_1$  and  $T_2$  are, respectively, spanning trees of  $G_1$  and  $G_2$ . Hence, by Theorem 2.3,  $G_1$  and  $G_2$  are connected. This, in turn, proves (Theorem 2.6) that the cut  $(V_1, V_2)$  is a cutset of  $G$ . This cutset is known as the fundamental cutset of  $G$  with respect to the branch  $b$  of the spanning tree  $\Gamma$  of  $G$ . The set of all the  $n-1$  fundamental cutsets with respect to the  $n-1$  branches of a spanning tree  $\Gamma$  of a connected graph  $G$  is known as the fundamental set of cutsets of  $G$  with respect to the spanning tree  $\Gamma$ .

Note that the cutset  $(V_j, V_2)$  contains exactly one branch, namely, the branch  $b$  of 7. All the other edges of  $(Vx, V2)$  are links of 7. This follows from the fact that  $(Vx, V2)$  does not contain any edge of 7, or 72. Further, branch  $b$  is not present in any other fundamental cutset with respect to 7. Because of these properties, the edge set of no fundamental cutset can be expressed as the ring sum of the edge sets of some or all of the remaining fundamental cutsets. We show in Chapter 4 that every cutset of a graph  $G$  can be expressed as the ring sum of the fundamental cutsets of  $G$  with respect to a spanning tree  $T$  of  $G$ .

A graph  $G$  and a set of fundamental cutsets of  $G$  are shown in Fig. 2.8.



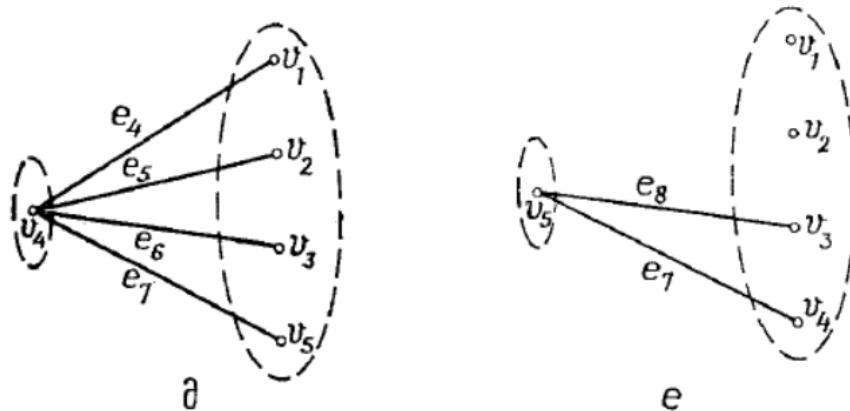


Рис. 2.8. Множество базисных разрезающих множеств графа.

*a*—граф  $G$ ; *б*—остов  $T$  графа  $G$ ; *в*—базисное разрезающее множество по отношению к ветви  $e_1$ ; *г*—базисное разрезающее множество по отношению к ветви  $e_2$ ; *д*—базисное разрезающее множество по отношению к ветви  $e_6$ ; *е*—базисное разрезающее множество по отношению к ветви  $e_8$ .

## 2.8. Остовы, циклы и разрезающие множества

В этом разделе мы обсудим некоторые интересные результаты, связывающие разрезающие множества и циклы с остовами и кодеревьями соответственно. Эти результаты выявляют двойственную природу циклов и разрезающих множеств. Они приводят к альтернативным характеризациям разрезающих множеств и циклов в терминах остовов и кодеревьев соответственно.

Очевидно, что удаление разрезающего множества  $S$  из связного графа  $G$  разрушает все остовы графа  $G$ . Нетрудно показать, что разрезающее множество является минимальным множеством ребер, удаление которого из графа  $G$  вызывает разрушение всех остовов в графе. Однако результат, обратный этому, не очевиден. Первые теоремы данного параграфа затрагивают эти и некоторые другие вопросы, относящиеся к циклам.

**Теорема 2.9.** Разрезающее множество связного графа  $G$  содержит по крайней мере одну ветвь каждого остова графа  $G$ .

*Доказательство.* Предположим, что разрезающее множество  $S$  графа  $G$  не содержит ни одной ветви остова  $T$  графа  $G$ . Тогда граф  $G-S$  будет содержать остов  $T$  и, следовательно, по теореме 2.3  $G-S$  — связный граф. Но это противоречит тому, что  $S$  — разрезающее множество графа  $G$ .

**Теорема 2.10.** Цикл связного графа  $G$  содержит по крайней мере одно ребро каждого кодерева графа  $G$ .

*Доказательство.* Пусть цикл  $C$  графа  $G$  не содержит ни одного ребра кодерева  $T^*$  остова  $T$  графа  $G$ . Тогда граф  $G-T^*$  будет содержать цикл  $C$ . Так как  $G-T^*$  совпадает с остовом  $T$ , то, следовательно, остов  $T$  содержит цикл. Но это противоречит определению остова.

**Теорема 2.11.** Множество ребер  $S$  связного графа  $G$  является разрезающим множеством  $G$  тогда и только тогда, когда  $S$  — минимальное множество ребер, содержащих по крайней мере одну ветвь каждого остова графа  $G$ .

Эта теорема характеризует разрезающее множество в терминах остовов. Сформулируем подобную характеристику для циклов с использованием кодеревьев.

Рассмотрим множество ребер  $C$ , образующее цикл в графе  $G$ . По теореме 2.10 множество  $C$  содержит по крайней мере одно ребро каждого кодерева графа  $G$ . Покажем, что никакое собственное подмножество  $C'$  множества  $C$  не обладает этим свойством. Очевидно, что  $C'$  не содержит цикл. Тогда по теореме 2.4 можно построить остов  $T$ , содержащий подмножество  $C'$ . Кодерево  $T^*$ , соответствующее  $T$ , не имеет общих ребер с  $C'$ . Следовательно, для каждого собственного подмножества  $C'$  множества  $C$  существует по меньшей мере одно кодерево  $T^*$ , не имеющее общих ребер с подмножеством  $C'$ . В самом деле, это утверждение верно для каждого ациклического подграфа. Таким образом, имеем следующую теорему:

**Теорема 2.12.** Циклом связного графа  $G$  является минимальное множество ребер графа, содержащее по крайней мере одно ребро каждого кодерева графа  $G$ . Докажем обратную теорему.

**Теорема 2.13.** Множество ребер  $C$  связного графа  $G$  есть цикл в  $G$ , если оно является минимальным множеством, содержащим по крайней мере одно ребро каждого кодерева графа  $G$ .

Теоремы 2.12 и 2.13 устанавливают, что множество ребер  $C$  связного графа  $G$  будет циклом тогда и только тогда, когда оно является минимальным множеством ребер, содержащим по крайней мере одно ребро каждого кодерева графа  $G$ . Новые характеристики разрезающего множества и цикла, полученные с помощью теорем 2.11—2.13, выявляют двойственную природу понятий «цикл» и «разрезающее множество». Эту двойственность мы подробнее исследуем в гл. 10 при обсуждении теории матроидов.

Следующая теорема связывает циклы и разрезающие множества без использования понятия «дерево».

**Теорема 2.14.** Цикл и разрезающее множество связного графа имеют четное число общих ребер.

## 2.8 SPANNING TREES, CIRCUITS, AND CUTSETS

In this section we discuss some interesting results that relate cutsets and circuits to spanning trees and cospanning trees, respectively. These results will bring out the "dual" nature of circuits and cutsets. They will also lead to alternate characterizations of cutsets and circuits in terms of spanning trees and cospanning trees, respectively.

It is obvious that removal of a cutset  $S$  from a connected graph  $G$  destroys all the spanning trees of  $G$ . A little thought will indicate that a cutset is a minimal set of edges whose removal from  $G$  destroys all the spanning trees of  $G$ . However, the converse of this results is not so obvious. The first few theorems of this section discuss these questions and similar ones relating to circuits.

Theorem 2.9. A cutset of a connected graph  $G$  contains at least one branch of every spanning tree of  $G$ .

Proof. Suppose that a cutset  $S$  of  $G$  contains no branch of a spanning tree  $\Gamma$

of  $G$ . Then the graph  $G - S$  will contain the spanning tree  $T$  and hence, by Theorem 2.3,  $G - S$  is connected. This, however, contradicts that  $S$  is a cutset of  $G$ . •

Theorem 2.10. A circuit of a connected graph  $G$  contains at least one edge of every cospanning tree of  $G$ .

Proof. Suppose that a circuit  $C$  of  $G$  contains no edge of the cospanning tree  $T^*$  of a spanning tree  $T$  of  $G$ . Then the graph  $G - T^*$  will contain the circuit  $C$ . Since  $G - T^*$  is the same as the spanning tree  $T$ , this means that the spanning tree  $T$  contains a circuit. However, this is contrary to the definition of a spanning tree. •

Theorem 2.11. A set  $S$  of edges of a connected graph  $G$  is a cutset of  $G$  if and only if  $S$  is a minimal set of edges containing at least one branch of every spanning tree of  $G$ .

Theorem 2.11 gives a characterization of a cutset in terms of spanning trees. We would like to establish next a similar characterization for a circuit in terms of cospanning trees.

Consider a set  $C$  of edges constituting a circuit in a graph  $G$ . By theorem 2.10,  $C$  contains at least one edge of every cospanning tree of  $G$ . We now show that no proper subset  $C'$  of  $C$  has this property.

It is obvious that  $C$  does not contain a circuit. Hence, by Theorem 2.4, we can construct a spanning tree  $T$  containing  $C$ . The cospanning tree  $T^*$  corresponding to  $T$  has no common edge with  $C$ . Hence for every proper subset  $C'$  of  $C$ , there exists at least one cospanning tree  $T^*$  that has no common edge with  $C'$ . In fact, this statement is true for every acyclic subgraph of a graph. Thus we have the following theorem.

Theorem 2.12. A circuit of a connected graph  $G$  is a minimal set of edges of  $G$  containing at least one edge of every cospanning tree of  $G$ .

The converse of Theorem 2.12 follows next.

Theorem 2.13. The set  $C$  of edges of a connected graph  $G$  is a circuit of  $G$  if it is a minimal set containing at least one edge of every cospanning tree of  $G$ .

Theorem 2.14. A circuit and a cutset of a connected graph have an even number of common edges.

Theorem 2.14 is a very important one. It forms the basis of the orthogonality relationship between cutsets and circuits.

## 6.2. Матрица разрезов

Для определения матрицы разрезов ориентированного графа необходимо каждому разрезу графа присвоить ориентацию.

Рассмотрим ориентированный граф  $G = (V, E)$ . Если  $V_a$  — непустое подмножество множества  $V$ , то напомним (глава 2), что множество дуг, соединяющих вершины  $V_a$  и  $\bar{V}_a$ , является разрезом, обозначаемым  $\langle V_a, \bar{V}_a \rangle$ . Ориентацию  $\langle V_a, \bar{V}_a \rangle$  можно принять как от вершины  $V_a$  к вершине  $\bar{V}_a$ , так и наоборот. Допустим, мы принимаем ориентацию от вершины  $V_a$  к вершине  $\bar{V}_a$ . Тогда говорят, что ориентация дуги из  $\langle V_a, \bar{V}_a \rangle$  соответствует ориентации разреза  $\langle V_a, \bar{V}_a \rangle$ , если дуга ориентирована от вершины из  $V_a$  в вершину из  $\bar{V}_a$ .

*Матрица разрезов*  $Q_c = [q_{ij}]$  графа  $G$  с  $m$  ребрами имеет  $m$  столбцов и столько строк, сколько в этом графе имеется разрезов. Элемент  $q_{ij}$  определяется следующим образом:

$$\begin{cases} \text{Если граф } G \text{ ориентированный, } q_{ij} = & \begin{cases} 1, \text{ если } j\text{-я дуга входит в } i\text{-й разрез и ее ориентация соответствует ориентации разреза;} \\ -1, \text{ если } j\text{-я дуга входит в } i\text{-й разрез и ее ориентация не соответствует ориентации разреза;} \\ 0, \text{ если } j\text{-я дуга не входит в } i\text{-й разрез.} \end{cases} \\ \text{Если граф } G \text{ неориентированный, } q_{ij} = & \begin{cases} 1, \text{ если } j\text{-е ребро входит в } i\text{-й разрез;} \\ 0 \text{ в противном случае.} \end{cases} \end{cases}$$

Строки матрицы  $Q_c$  называют *векторами разрезов*.

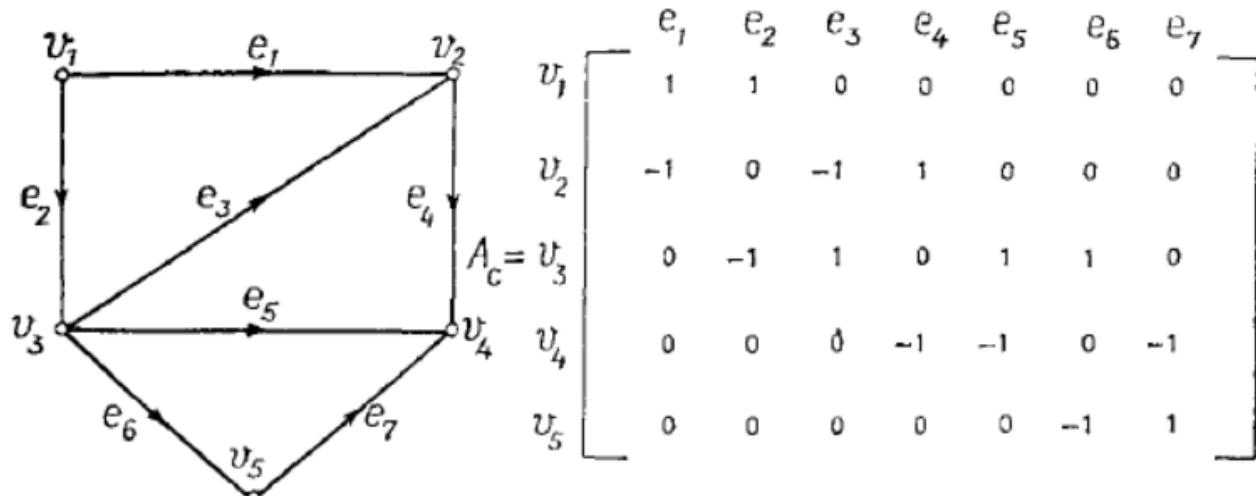


Рис. 6.1.

*a* — ориентированный граф  $G$  и его матрица инциденций;

На рис. 6.2 представлены три разреза ориентированного графа из рис. 6.1, *a*. Штриховыми линиями в каждом случае показана ориентация разреза. Соответствующая этим трем разрезам подматрица  $Q_c$  имеет вид

	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$
Разрез 1	1	0	1	0	1	1	0
Разрез 2	1	1	0	0	0	-1	1
Разрез 3	0	-1	1	0	1	1	0

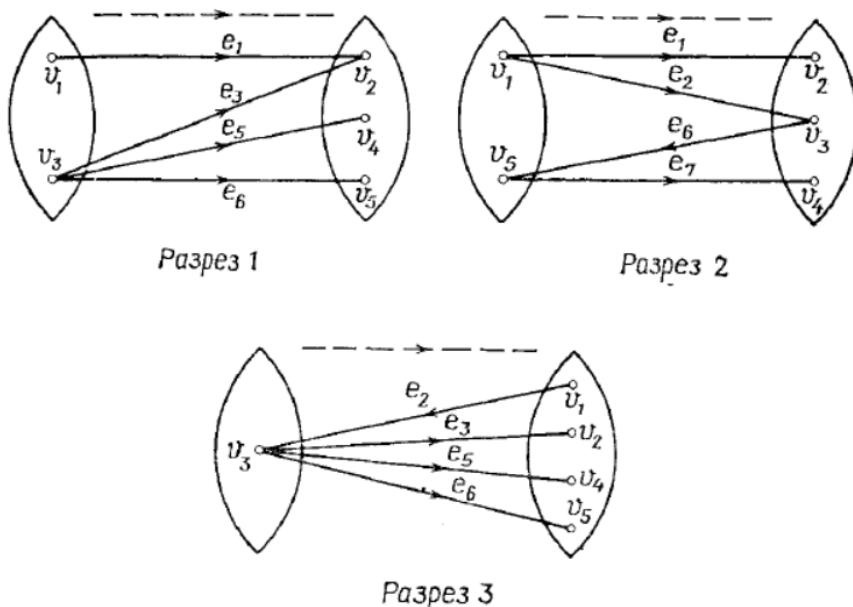


Рис. 6.2. Некоторые разрезы графа на рис. 6.1, *a*.

Соответствующую подматрицу в случае неориентированного графа можно получить, заменив в этой матрице «-1» на «+1».

Рассмотрим произвольную вершину  $v$ . Ненулевые элементы соответствующего вектора инциденций представляют инцидентные вершине  $v$  дуги. Эти дуги образуют разрез  $\langle v, V-v \rangle$ . Если принять ориентацию этого разреза от  $v$  к  $V-v$ , то из определений матриц разрезов и инциденций следует, что строка матрицы  $Q_c$ , соответствующая разрезу  $\langle v, V-v \rangle$ , совпадает со строкой матрицы  $A_c$ , соответствующей вершине  $v$ . Следовательно,  $A_c$  — подматрица матрицы  $Q_c$ .

Покажем, что ранги матриц  $Q_c$  и  $A_c$  одинаковы. Для этого докажем следующую теорему:

**Теорема 6.3.** Всякую строку матрицы разрезов  $Q_c$  можно выразить двумя способами в виде линейной комбинации строк матрицы  $A_c$ . В обоих случаях ненулевые коэффициенты в линейной комбинации равны +1 или -1.

Для иллюстрации теоремы рассмотрим разрез 1 на рис. 6.2. Он разделяет вершины в  $V_a = \{v_1, v_3\}$  и вершины в  $\bar{V}_a$ . Ориентация разреза от  $V_a$  к  $\bar{V}_a$ . Поэтому соответствующий вектор разреза можно выразить следующим образом:  $[1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0] = a_1 + a_3 = -a_2 - a_4 - a_5$ , где  $a_1, a_2, \dots, a_5$  — строки матрицы  $A$  (рис. 6.1, а).

Важным следствием из теоремы 6.3 является то, что  $\text{rank}(Q_c) \leq \text{rank}(A_c)$ . Однако, поскольку  $A_c$  — подматрица матрицы  $Q_c$ , то  $\text{rank}(Q_c) \geq \text{rank}(A_c)$ . Поэтому мы получаем, что  $\text{rank}(Q_c) = \text{rank}(A_c)$ .

Теперь из теоремы 6.2 и следствия 6.2.1 вытекает следующая теорема:

**Теорема 6.4.** Ранг матрицы разрезов  $Q_c$  связного графа  $G$  на  $n$  вершинах равен  $n-1$ , т. е. рангу  $G$ .

*Следствие 6.4.1.* Ранг матрицы разрезов  $Q_c$  графа  $G$  на  $n$  вершинах, имеющего  $p$  компонент, равен  $n-p$ , т. е. рангу этого графа.

Из этих выводов следует, что матрица инциденций  $A_c$  является важной подматрицей матрицы разрезов  $Q_c$ . Определим сейчас другую важную подматрицу матрицы  $Q_c$ .

Нам известно, что остов  $T$  связного графа  $G$  на  $n$  вершинах определяет множество из  $n-1$  базисного разрезающего множества — по одному базисному разрезающему на ветвь дерева  $T$ . Подматрица  $Q_f$ , соответствующая этому  $n-1$ -базисному разрезающему множеству, называется *базисной матрицей разрезающих множеств*  $Q_f$  графа  $G$  по отношению к дереву  $T$ . Пусть  $b_1, b_2, \dots, b_{n-1}$  — ветви дерева  $T$ . Переставим столбцы и строки так, что

- 1)  $i$ -й столбец соответствует ветви  $b_i$  для  $1 \leq i \leq n-1$ ;
- 2)  $i$ -я строка соответствует базисному разрезающему множеству, определяемому  $b_i$ . Если теперь ориентацию базисного разрезающего множества выбрать таким образом, чтобы ей соответствовала ориентация определяющей ветви, тогда матрицу  $Q_f$ , можно представить в следующей удобной форме:

$$Q_f = [U \mid Q_{fc}], \quad (6.8)$$

где  $U$  — единичная матрица порядка  $n-1$ , а столбцы ее соответствуют ветвям дерева  $T$ .

Например, базисная матрица разрезающих множеств  $Q_f$  связного графа на рис. 6.1, *a* по отношению к остову  $T = \{e_1, e_2, e_6, e_7\}$  имеет вид

$$Q_f = \begin{array}{c|ccccc} & e_1 & e_2 & e_6 & e_7 & e_3 & e_4 & e_5 \\ \hline e_1 & 1 & 0 & 0 & 0 & 1 & -1 & 0 \\ e_2 & 0 & 1 & 0 & 0 & -1 & 1 & 0 \\ e_6 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ e_7 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}. \quad (6.9)$$

Из (6.8) следует, что ранг матрицы  $Q_f$  равен  $n-1$ , т. е. рангу матрицы  $Q_c$ . Таким образом, всякий вектор разреза можно выразить в виде линейной комбинации базисных векторов разрезающих множеств.

## 6.2 CUT MATRIX

To define the cut matrix of a directed graph we need to assign an orientation to each cut of the graph.

Consider a directed graph  $G = (V, E)$ . If  $V_a$  is a nonempty subset of  $V$ , then we may recall (Chapter 2) that the set of edges connecting the vertices in  $V_a$  to those in  $V_a$  is a cut, and this cut is denoted as  $(v_{\dots}, v_{\dots})$ . The orientation of  $(V_a, V_a)$  may be assumed to be either from  $V_a$  to  $V_a$  or from  $V_a$  to  $V_a$ . Suppose we assume that the orientation is from  $V_a$  to  $V_a$ . Then the orientation of an edge in  $(V_a, V_a)$  is said to agree with the orientation of the cut  $(V_a, V_a)$  if the edge is oriented from a vertex in  $V_a$  to a vertex in  $V_a$ . The cut matrix  $Q_c = [q_{ij}]$  of a graph  $G$  with  $m$  edges has  $m$  columns and as many rows as the number of cuts in  $G$ . The entry  $q_{ij}$  is defined as follows:

$G$  is directed

1, if the  $i$ -th edge is in the  $j$ -th cut and its orientation agrees with the cut orientation;

-1, if the  $i$ -th edge is in the  $j$ -th cut and its orientation does not agree with the cut orientation;

0, if the  $i$ -th edge is not in the  $j$ -th cut.

$G$  is undirected

-1, if the  $i$ -th edge is in the  $j$ -th cut;

$Q'i \sim \backslash 0$ , otherwise .

A row of  $Q_c$  will be referred to as a cut vector.

Three cuts of the directed graph of Fig. 6.1a are shown in Fig. 6.2. In each case the cut orientation is shown in dashed lines. The submatrix of  $Q_c$  corresponding to these cuts is

e

i e2 e3 «6 «7

Cutl ' 1 0 1 0 1 1 0"

Cut 2 1 1 0 0 0 -1 1

Cut 3 0 -1 1 0 1 1 0

The corresponding submatrix in the undirected case can be obtained by replacing the -l's in the matrix by +l's.

Consider next any vertex  $v$ . The nonzero entries in the corresponding incidence vector represent the edges incident on  $v$ . These edges form the cut  $(v, V - v)$ . If we assume that the orientation of this cut is from  $v$  to  $V - v$ , then we can see from the definitions of cut and incidence matrices that the row in  $Q_c$  corresponding to the cut  $(v, V - v)$  is the same as the row in  $A_c$  corresponding to the vertex  $v$ . Thus  $A_c$  is a submatrix of  $Q_c$ .

Now we proceed to show that the rank of  $Q_c$  is equal to that of  $A_c$ . To do so we need the following theorem.

Theorem 6.3. Each row in the cut matrix  $Q_c$  can be expressed, in two ways, as a linear combination of the rows of the matrix  $A_c$ . In each case the nonzero coefficients in the linear combination are all +1 or all -1.

Proof. Let  $(V_a, V_b)$  be the  $i$ 'th cut in a graph  $G$  with  $\eta$  vertices and  $m$  edges, and let  $a$  be the corresponding cut vector. Let  $V_a = \{v_1, v_2, \dots, v_r\}$  and  $V_b = \{v_{r+1}, v_{r+2}, \dots, v_m\}$ . For  $1 < i < n$ , let  $a_i$  denote the incidence vector corresponding to the vertex  $v_i$ .

We assume, without any loss of generality, that the orientation of  $(V_a, V_b)$  is from  $V_a$  to  $V_b$ , and prove the theorem by establishing that  $q_i = a_1 + a_2 + \dots + a_r = -(a_{r+1} + a_{r+2} + \dots + a_m)$ . (6.5)

Let  $v_p$  and  $v_q$  be the end vertices of the  $k$ th edge,  $1 < k < m$ . Let this edge be oriented from  $v_p$  to  $v_q$  so that

$a_{pk} = 1$

$a_{qk} = -1$ , (6.6)

Now four cases arise.

Case 1  $v_p \in V_a$  and  $v_q \in V_b$ , that is,  $p \leq r$  and  $q > r + 1$ , so that  $q - p = r + 1$ .

Case 2  $v_p \in V_b$  and  $v_q \in V_a$ , that is,  $p > r + 1$  and  $q < r$ , so that  $p - q = r + 1$ .

Case 3  $v_p, v_q \in V_a$ , that is  $p, q \leq r$ , so that  $q - p = 0$ .

Case 4  $v_p, v_q \in V_b$ , that is,  $p, q > r + 1$ , so that  $p - q = 0$ .

It is easy to verify, using (6.6), that the following is true in each of these

four cases.

$$\begin{aligned} <\text{tik} &= (\langle u + \langle 2^* + \dots + \langle r^*) \\ &= -(\langle, + !.^* + \langle r + 2.k + \dots + \text{ank}). \quad (6.7) \end{aligned}$$

Now (6.5) follows since (6.7) is valid for all  $1 < A: < m$ . Hence the theorem. •

To illustrate Theorem 6.3, consider the cut 1 in Fig. 6.2. This cut separates the vertices in  $V_a = \{v_i, i \geq 3\}$  from those in  $V_b$ . The cut orientation is from  $V_a$  to  $V_b$ . So the corresponding cut vector can be expressed as follows:

$$\begin{aligned} [1 0 1 0 1 1 0] &= a_1 + a_3 \\ &= -a_2 -a_4-a_5 \end{aligned}$$

where  $a_1, a_2, \dots, a_5$  are the rows of the matrix  $A$  in Fig. 6.1a.

An important consequence of Theorem 6.3 is that  $\text{rank}(Q_c) \leq \text{rank}(A_c)$ . However,  $\text{rank}(Q_c) \geq \text{rank}(y_{4c})$  because  $A_c$  is a submatrix of  $Q_c$ . Therefore, we get

$$\text{rank}(Q_c) = \text{rank}(A_c).$$

Thus Theorem 6.2 and Corollary 6.2.1, respectively, lead to the following.

**Theorem 6.4.** The rank of the cut matrix  $Q_c$  of an  $\eta$ -vertex connected graph

$G$  is equal to  $\eta - 1$ , the rank of  $G$ . •

**Corollary 6.4.1.** The rank of the cut matrix  $Q_c$  of an  $\eta$ -vertex graph  $G$  with  $\rho$  components is equal to  $\eta - \rho$ , the rank of  $G$ . •

As the above discussions show, the all-vertex incidence matrix  $A_c$  is an important submatrix of the cut matrix  $Q_c$ . Next we identify another important submatrix of  $Q_c$ .

We know that a spanning tree  $T$  of an  $\eta$ -vertex connected graph  $G$  defines a set of  $\eta - 1$  fundamental cutsets—one fundamental cutset for each branch of  $T$ . The submatrix of  $Q_c$  corresponding to these  $\eta - 1$  fundamental cutsets is known as the fundamental cutset matrix  $Q_{fc}$  with respect to  $T$ .

Let  $b_1, b_2, \dots, b_{\eta-1}$  denote the branches of  $T$ . Suppose we arrange the columns and the rows of  $Q_f$  so that

1. For  $1 < i < \eta - 1$ , the  $i$ th column corresponds to the branch  $b_i$ .

2. The  $i$ th row corresponds to the fundamental cutset defined by  $b_i$

If, in addition, we assume that the orientation of a fundamental cutset is so chosen as to agree with that of the defining branch, then the matrix  $Q_f$  can be displayed in a convenient form as follows:

$$Q_f = [U | Q_{fc}] \quad (6.8)$$

where  $U$  is the unit matrix of order  $\eta - 1$  and its columns correspond to the branches of  $T$ .

For example, the fundamental cutset matrix  $Q_f$  for the connected graph of Fig. 6.1a with respect to the spanning tree  $T = \{e_1, e_2, e_6, e_7\}$  is

$$Q_f$$

$$e_2 \ e_6 \ e_7 \ e_3 \ \langle 4$$

ex 1 0 0 0 1 -1 0  
 e2 0 1 0 0 -1 1 0  
 0 0 1 0 0 1 1  
 e7 0 0 0 1 0 1 1  
 (6.9)

It is clear from (6.8) that the rank of  $Q_f$  is equal to  $\eta - 1$ , the rank of  $Q_c$ . Thus every cut vector (which may be a cutset vector) can be expressed as a linear combination of the fundamental cutset vectors.

### 6.3. Цикломатическая матрица

Цикл можно обойти в одном из двух направлений: по часовой или против часовой стрелки. Направление, выбираемое для обхода цикла, определяет его ориентацию. Ориентацию цикла можно указать стрелкой, как на рис. 6.3.

Рассмотрим дугу  $e$  с концевыми вершинами  $v_i$  и  $v_j$ . Пусть дуга  $e$  ориентирована от  $v_i$  к  $v_j$  и входит в цикл  $C$ . Будем говорить, что ориентация дуги  $e$  соответствует ориентации цикла, если вершина  $v_i$

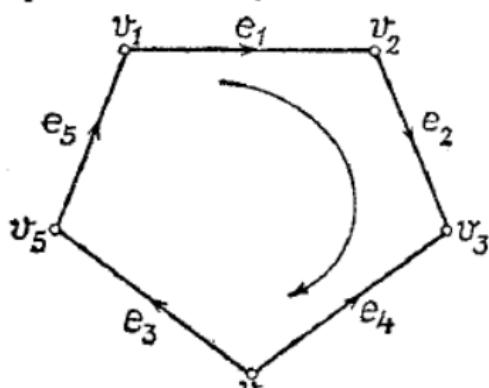


Рис. 6.3.

встречается при обходе цикла  $C$  в направлении, указанном его ориентацией, раньше вершины  $v_j$ . Например, ориентация дуги  $e_1$  в цикле на рис. 6.3 соответствует ориентации цикла, а ориентация дуги  $e_4$  — нет.

**Цикломатическая матрица**<sup>1)</sup>  $B_c = [b_{ij}]$  графа  $G$  с  $m$  ребрами имеет  $m$  столбцов и столько строк, сколько циклов имеется в графе  $G$ . Элемент  $b_{ij}$  определяется следующим образом:

Если граф  $G$  ориентированный,  $b_{ij} =$

- |   |
|---|
| $1$ , если $j$ -я дуга входит в $i$ -й цикл и ее ориентация соответствует ориентации цикла;     |
| $-1$ , если $j$ -я дуга входит в $i$ -й цикл и ее ориентация не соответствует ориентации цикла; |
| $0$ , если $j$ -я дуга не входит в $i$ -й цикл.   |

Если граф  $G$  неориентированный,  $b_{ij} =$

- |   |
|---|
| $1$ , если $j$ -е ребро входит в $i$ -й цикл; |
| $0$ в противном случае.                       |

Строки матрицы  $B_c$  называются *циклическими векторами* графа  $G$ .

В качестве примера снова рассмотрим граф на рис. 6.1, а. На рис. 6.4 представлены три цикла этого графа и их ориентация. Подматрица  $B_c$ , соответствующая этим циклам, имеет вид

	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$
Цикл 1	1	-1	0	1	-1	0	0
Цикл 2	-1	1	1	0	0	0	0
Цикл 3	1	-1	0	1	0	-1	-1

Соответствующая подматрица для неориентированного графа рис. 6.1, б получается заменой в этой матрице «-1» на «+1».

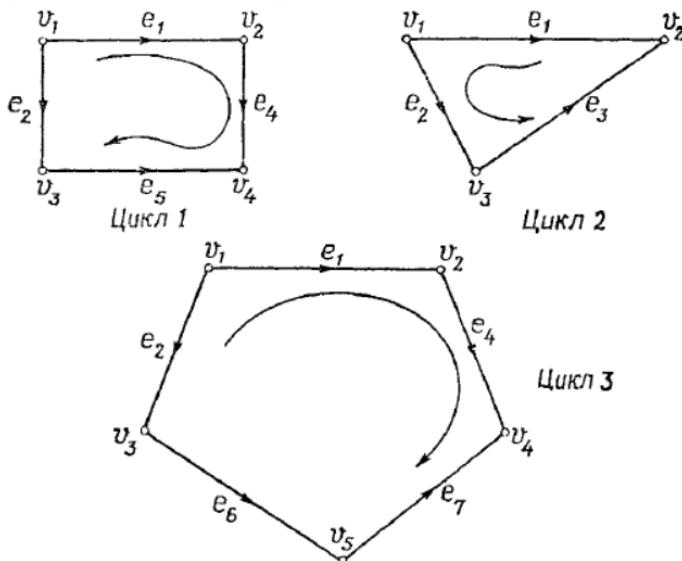


Рис. 6.4. Некоторые циклы графа на рис. 6.1, а.

Сейчас определим подматрицу  $B_c$ , играющую важную роль.

Рассмотрим произвольный остов  $T$  связного графа  $G$ , имеющего  $n$  вершин и  $m$  ребер. Пусть  $c_1, c_2, \dots, c_{m-n+1}$  — хорды  $T$ . Как известно, эти  $m-n+1$ -хорды определяют множество из  $m-n+1$ -базисных циклов. Подматрица  $B_c$ , соответствующая этим базисным циклам, называется *базисной цикломатической матрицей*  $B_f$  графа  $G$  по отношению к остову  $T$ .

Переставим столбцы и строки  $B_f$ , так, что

- 1)  $i$ -й столбец соответствует хорде  $c_i$ ,  $1 \leq i \leq m-n+1$ ;
- 2)  $i$ -я строка соответствует базисному циклу, определяемому хордой  $c_i$ .

Выбирая ориентацию базисного цикла таким образом, чтобы ей соответствовала ориентация определяющей хорды, получим следующее представление матрицы  $B_f$ :

$$B_f = [U \mid B_{f,t}], \quad (6.10)$$

где  $U$  — единичная матрица размерности  $m-n+1$ , столбцы которой соответствуют хордам остова  $T$ .

Например, базисная цикломатическая матрица графа на рис.6.1,а по отношению к остову  $T=\{e_1, e_2, e_6, e_7\}$  имеет вид

$$B_f = \begin{array}{c|ccccccc} e_3 & e_4 & e_5 & e_1 & e_2 & e_6 & e_7 \\ \hline e_3 & 1 & 0 & 0 & -1 & 1 & 0 & 0 \\ e_4 & 0 & 1 & 0 & 1 & -1 & -1 & -1 \\ e_5 & 0 & 0 & 1 & 0 & 0 & -1 & -1 \end{array} \quad (6.11)$$

Из (6.10) следует, что ранг матрицы  $B_f$  равен  $m-n+1$ . Поскольку матрица  $B_f$  — подматрица  $B_c$ , получаем

$$\text{rank}(B_c) \geq m-n+1. \quad (6.12)$$

**Теорема 6.6.** (Соотношение ортогональности.) Если столбцы цикломатической матрицы  $B_c$  и матрицы разрезов  $Q_c$  расположить в одном порядке, то  $B_c Q_c^t = 0$ .

Таким образом, любой циклический вектор можно выразить в виде линейной комбинации базисных циклических векторов. Поэтому  $\text{rank}(B_c) \leq \text{rank}(B_f) = m-n+1$ . Объединяя это неравенство с выражением (6.12), получаем следующую теорему и следствие из нее:

**Теорема 6.7.** Ранг цикломатической матрицы  $B_c$  связного графа  $G$ , имеющего  $n$  вершин и  $m$  дуг, равен  $m-n+1$ , т. е. цикломатическому числу графа  $G$ .

**Следствие 6.7.1.** Ранг цикломатической матрицы  $B_c$  графа  $G$  на  $n$  вершинах и  $m$  дугах с  $p$  компонентами равен  $m-n+p$ , т. е. цикломатическому числу графа  $G$ .

Графы і эл. Цепи

Двумя фундаментальными законами теории электрических цепей являются законы Кирхгофа, которые можно сформулировать следующим образом:

**Закон Кирхгофа для токов (ЗКТ).** Алгебраическая сумма токов, вытекающих из узла, равна нулю.

**Закон Кирхгофа для напряжений (ЗКН).** Алгебраическая сумма напряжений в любом замкнутом контуре равна нулю.

Например, для цепи, показанной на рис. 11.2, а, уравнения ЗКТ и ЗКН приведены ниже:

$$\begin{array}{ll} \text{Уравнения ЗКТ} & \begin{array}{l} \text{Узел } a \quad i_1 - i_5 + i_6 = 0, \\ \text{Узел } c \quad -i_2 + i_4 - i_6 = 0, \\ \text{Узел } b \quad -i_1 + i_2 + i_3 = 0. \end{array} \end{array}$$

$$\begin{array}{ll}
 \text{Уравнения ЗКН} & \text{Контур } \{1, 3, 5\} \quad v_1 + v_3 + v_5 = 0, \\
 & \text{Контур } \{2, 4, 8\} \quad v_2 + v_4 - v_8 = 0, \\
 & \text{Контур } \{1, 6, 2\} \quad -v_1 + v_6 - v_2 = 0.
 \end{array}$$

Для заданной электрической цепи  $N$  задача анализа состоит в том, чтобы определить напряжения и токи в элементах, которые удовлетворяют законам Кирхгофа, и соотношения ток — напряжение, характеризующие различные элементы, образующие цепь.

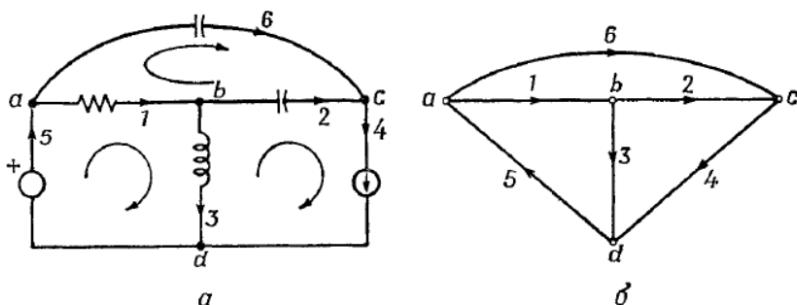


Рис. 11.2. Представление цепи ориентированным графом.  
а — цепь  $N$ ; б — ориентированный граф  $N$ .

Пусть  $T$  является остовом цепи  $N$ , а  $B_f$  и  $Q_f$  обозначают фундаментальную цикломатическую матрицу и матрицу сечений по отношению к  $T$ . Тогда уравнения Кирхгофа для токов и напряжений будут иметь вид

$$Q_f I_e = 0, \quad (11.1)$$

$$B_f V_e = 0. \quad (11.2)$$

Предположим, что  $I_e$  и  $V_e$  разделены следующим образом:

$$I_e = \begin{bmatrix} I_c \\ I_t \end{bmatrix} \text{ и } V_e = \begin{bmatrix} V_c \\ V_t \end{bmatrix},$$

где векторы, отвечающие хордам и ветвям остова  $T$ , отличаются подстрочными индексами  $c$  и  $t$  соответственно. Тогда выражения (11.1) и (11.2) можно записать в виде

$$[Q_{fc} \quad U] \begin{bmatrix} I_c \\ I_t \end{bmatrix} = 0, \quad (11.3)$$

$$[U \quad B_{ft}] \begin{bmatrix} V_c \\ V_t \end{bmatrix} = 0. \quad (11.4)$$

Заметим, что [выражение (6.13)]

$$Q_{fc} = -B_{ft}^t.$$

Сначала рассмотрим выражение (11.3). Из него получаем

$$I_t = -Q_{fc} I_c = B_{ft}^t I_c.$$

Таким образом, ток  $I_e$  можно представить в виде

$$I_e = \begin{bmatrix} I_c \\ I_t \end{bmatrix} = \begin{bmatrix} U \\ B_{ft}^t \end{bmatrix} I_c = B_f^t I_c.$$

Исходя из выражения (11.4), аналогично можно показать, что

$$V_e = Q_f^t V_t. \quad (11.8)$$

В результате имеем следующую теорему:

**Теорема 11.1.** 1. Все токи, текущие через элементы электрической цепи  $N$ , можно выразить линейной комбинацией хордовых токов, т. е. токов, связанных с хордами остова  $N$ .

2. Все напряжения на элементах электрической цепи  $N$  можно выразить линейной комбинацией напряжений на ветвях, т. е. напряжений, связанных с ветвями остова цепи  $N$ .

Для иллюстрации выражений (11.7) и (11.8) рассмотрим цепь  $N$ , представленную на рис. 11.2.

Матрицами  $B_f$  и  $Q_f$  по отношению к остову  $T$ , состоящему из элементов 1, 4 и 5, будут матрицы

$$B_f = \begin{array}{c|ccc|cc} 2 & 3 & 6 & 1 & 4 & 5 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array},$$

$$Q_f = \begin{array}{ccc|ccc} 2 & 3 & 6 & 1 & 4 & 5 \\ \hline -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 \\ -1 & -1 & -1 & 0 & 0 & 1 \end{array}.$$

Тогда  $I_e$  и  $V_e$  можно выразить следующим образом:

$$\begin{bmatrix} i_2 \\ i_3 \\ i_6 \\ i_1 \\ i_4 \\ i_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_2 \\ i_3 \\ i_6 \end{bmatrix},$$

$$\begin{bmatrix} v_2 \\ v_3 \\ v_6 \\ v_1 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_4 \\ v_5 \end{bmatrix}.$$

Рекомендована література

1. Сигорский

2. ...