МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
"ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"

С. П. Іглін, Ю. І. Зайцев, Ю. Б. Решетняк

Теорія графів на базі MATLAB

Навчальний посібник
для студентів інформаційних спеціальностей
усіх форм навчання
вищих навчальних закладів

Затверджено
редакційно-видавничою
радою університету
протокол № 03 від 26.10.2022 р.

Харків
НТУ "ХПІ"
2023

УДК 519.17
І 26

Рецензенти:
М. В. Сидоров, д-р ф.-м. наук, професор, Харківський національний університет радіоелектроніки;
Ю. І. Руднєв, к-т ф.-м. наук, доцент, Харківський національний університет ім. В. Н. Каразіна.

Рекомендовано Вченою Радою Національного технічного університету "Харківський політехнічний інститут" протокол № 09 від 24 листопада 2022 р.

Іглін, С. П.
І 26 Теорія графів на базі MATLAB: навч. посіб. для студентів інформаційних спеціальностей усіх форм навчання вищих навчальних закладів / С. П. Іглін, Ю. І. Зайцев, Ю. Б. Решетняк. — Харків: "НТМТ" , 2023. — 236 с.

ISBN 978-617-578-345-0

Розглядаються деякі задачі теорії графів. Там, де це доцільно, вони зводяться до задач лінійного програмування: звичайного, цілочисельного або бінарного. В інших випадках розглядаються класичні алгоритми. Наведено приклади. Усі розділи орієнтовані на широке застосування математичного пакету MATLAB.

Для студентів, аспірантів, викладачів, наукових працівників, що використовують теорію графів.

Іл. 112. Табл. 3. Бібліогр.: 14 назв.

УДК 519.17

ISBN 978-617-578-345-0

© С. П. Іглін, Ю. І. Зайцев,
Ю. Б. Решетняк, 2023 р.

Зміст

Вступ 5
1. Введення в теорію графів 10
1.1. Основні означення 10
1.2. Як задати граф 18
1.3. Матричні представлення графів 27
1.4. Запитання для перевірки 43
2. Пакування, покриття, домінуючі множини, кліки 45
2.1. Реберне пакування 45
2.2. Вершинне пакування 53
2.3. Реберне покриття 57
2.4. Вершинне покриття 61
2.5. Двоїстість задач про пакування та покриття 65
2.6. Домінуюча множина ребер 67
2.7. Домінуюча множина вершин 72
2.8. Повний підграф. 77
2.9. Запитання для перевірки 78
3. Правильна розфарбовка графів 81
3.1. Мінімальна правильна розфарбовка вершин графа . . . 81
3.2. Мінімальна правильна розфарбовка ребер графа 86
3.3. Запитання для перевірки 91
4. Мінімальні остовні дерева 92
4.1. Жадібні алгоритми та матроїди 92
4.2. Мінімальне остовне дерево (МОД) 98
4.3. Запитання для перевірки 109
5. Цикли та коцикли 110
5.1. Ейлерові цикли 110
5.2. Гамільтонові цикли 117
5.3. Фундаментальна система циклів 119
5.4. Фундаментальна система коциклів 130
5.5. Запитання для перевірки 141

6. Сильно зв'язані компоненти орграфа 142
6.1. Бінарні відношення 142
6.2. Сильно зв'язані компоненти 143
6.3. Визначення сильно зв'язаних компонент 147
6.4. Часткове упорядкування сильно зв'язаних компонент . . 157
6.5. Запитання для перевірки 165
7. Найкоротший шлях 166
7.1. Постановка задачі про найкоротший шлях 166
7.2. Алгоритм Дейкстри 166
7.3. Алгоритм Флойда-Воршола 176
7.4. Метричні характеристики графа 180
7.5. Запитання для перевірки 183
8. Мережеві задачі 185
8.1. Мережі 185
8.2. Максимальний потік у мережі 189
8.3. Двоїста задача — мінімальний розріз 193
8.4. Мережеві діаграми PERT 197
8.5. Запитання для перевірки 201
9. Ізоморфізм графів 202
9.1. Постановка задачі 202
9.2. Інваріанти графів 204
9.3. Запитання для перевірки 214
10. Індивідуальні домашні завдання 215
10.1. Зміст завдання 215
10.1.1. Задачі на простому графі 215
10.1.2. Задачі на орграфі 215
10.2. Варіанти завдань 217
Предметний покажчик 232
Література 235

Вступ

На рівні наших буденних уявлень граф — це будь-які об'єкти (зазвичай їх малюють у вигляді точок або кружків), поєднані лініями чи стрілками. Зокрема, у вигляді графів можна представити електричні схеми, маршрути перевезень, схеми взаємозв'язків підрозділів підприємства, грошові та ресурсні потоки, системи керування різними об'єктами тощо. З давніх часів люди помітили, що графи мають спільні властивості незалежно від того, який реальний об'єкт вони представляють. На основі вивчення цих властивостей і виникла наука під назвою "Теорія графів". У процесі її розвитку з'ясувалося, що вона тісно пов'язана з іншими розділами математики: теорією множин і комбінаторним аналізом. Тому в технічних ЗВО зазвичай теорію графів поряд з теорією множин, комбінаторикою, топологією та деякими іншими розділами математики вивчають у курсі під назвою "Дискретна математика".

З іншого боку, багато задач теорії графів формулюються як задачі лінійного програмування: звичайного, цілочисельного чи бінарного. Це дозволяє розглядати окремі розділи теорії графів у предметах "Методи оптимізації" та "Дослідження операцій". Ці лекції були написані як раз для курсу "Математичні методи дослідження операцій" для студентів НТУ "ХПІ" , що навчаються за напрямом підготовки "Інформатика" , спеціалізаціями "Інформаційні технології проектування" та "Комп'ютерна графіка". Але зараз цей курс також викладається студентам напряму підготовки "Прикладна математика" як частина предмету "Дискретні структури і структури даних". Розділ "Теорія графів" читається після розділу "Лінійне програмування". Там, де це доцільно, задачі теорії графів зводяться до задач лінійного програмування. В інших випадках розглядаються класичні алгоритми.

Для докладнішого ознайомлення з теорією графів можна скористатися підручниками [5, 6, 12–14].

Деякі алгоритми розглядаються на сайті [4]. Попередні відомості про лінійне програмування можна знайти в книгах [2, 3].

Важливою особливістю сучасного розвитку інформаційних технологій є поява потужних математичних пакетів, що дозволяють максимально спростити процес підготовки задачі, її розв'язання та аналізу отриманих результатів. Застосовуючи такі програмні продукти, як Maple, Mathcad, Mathematica чи MATLAB, багато математичних задач можна розв'язати досить швидко та просто. Зокрема, в цій книзі для розв'язання задач теорії графів використовується MATLAB. Для ознайомлення з цим пакетом можна скористатися чисельними підручниками, наприклад, [8], та офіційною інформацією на сайті [9].

Раніше в MATLAB не було готових пакетів для розв'язання задач на графах, тому користувачі намагалися усунути цей недолік, створю-

чи свої функції та розміщуючи їх в інтернеті. Зокрема, один з авторів цієї книги розробив Graph Theory Toolbox для MATLAB та розмістив його на сайті [10]. Але час плине, MATLAB розвивається та покращується, і в останніх релізах MATLAB з'явився вбудований інструментарій для розв'язання задач на графах. Проте деякі потрібні функції в ньому відсутні. Тому в цій книзі розглядаються як вбудовані в MATLAB функції для роботи з графами, так і додаткові функції з Graph Theory Toolbox.

Для роботи з цією книгою потрібен MATLAB, встановлений на комп'ютері користувача. Бажано встановлювати одну з останніх версій. Зокрема, всі функції Graph Theory Toolbox створювалися автором в релізі MATLAB-2021b. Деякі функції цього пакету вимагають для своєї роботи також встановлення інструментарію для розв'язання задач оптимізації Optimization Toolbox.

Для полегшення перехресних посилань усі формули, означення, теореми, рисунки тощо мають подвійну нумерацію, що включає номер розділу. Кінець прикладу, означення, формулювання чи доведення теореми позначений ось таким квадратиком: □.

У псевдокодах алгоритмів нумерація індексів у масивах починається з одиниці. Самі індекси записуються в круглих дужках: `A(i,j)`. Символ присвоювання — знак рівності. Тіла циклів та умовних операторів відокремлюються службовими словами `begin` та `end`.

Фрагменти коду MATLAB та текстові результати розрахунків, що виводяться у командне вікно MATLAB, у цій книзі мають такий вигляд:

```
Це - зразок області введення команд MATLAB.
```

```
Це - зразок області друку результатів роботи
у MATLAB Command Window.
```

Вони набрані `ось таким` або `ось таким` шрифтом і на світло-сірому фоні. Графічні результати, якщо вони є, розміщуються в книзі автоматично на зручних місцях компоновувачем LaTeX (не обов'язково відразу після текстових) та нумеруються, як і всі рисунки, наскрізною нумерацією в кожному розділі.

Для отримання довідки про будь-яку команду, функцію чи конструкцію мови програмування MATLAB треба ввести в командному вікні команду `help` з потрібним запитом. Ось, наприклад, як виглядає довідка про функцію `intlinprog`, що використовується в деяких функціях Graph Theory Toolbox.

```
>> help intlinprog
 intlinprog Mixed integer linear programming.

    X = intlinprog(f,intcon,A,b) attempts to solve problems
        of the form

    min f'*x    subject to:  A*x  <= b
     x                       Aeq*x = beq
                             lb <= x <= ub
                             x(i) integer, where i is
                             in the index vector intcon
                             (integer constraints)

    X = intlinprog(f,intcon,A,b) solves the problem with
    integer variables in the intcon vector and linear
    inequality constraints A*x <= b.
    intcon is a vector of positive integers indicating
    components of the solution X that must be integers.
    For example, if you want to constrain X(2) and
    X(10) to be integers, set intcon to [2,10].

    X = intlinprog(f,intcon,A,b,Aeq,beq) solves the problem
    above while additionally satisfying the equality
    constraints Aeq*x = beq. (Set A=[] and b=[] if
    no inequalities exist.)

    X = intlinprog(f,intcon,A,b,Aeq,beq,LB,UB) defines
    a set of lower and upper bounds on the design variables,
    X, so that the solution is in the range LB <= X <= UB.
    Use empty matrices for LB and UB if no bounds exist.
    Set LB(i) = -Inf if X(i) is unbounded below;
    set UB(i) = Inf if X(i) is unbounded above.

    X = intlinprog(f,intcon,A,b,Aeq,beq,LB,UB,X0) sets
    the initial point to X0.

    X = intlinprog(f,intcon,A,b,Aeq,beq,LB,UB,X0,OPTIONS)
    minimizes with the default optimization parameters
    replaced by values in OPTIONS, an argument created
    with the OPTIMOPTIONS function.
    See OPTIMOPTIONS for details.
```

```
X = intlinprog(PROBLEM) finds the minimum for PROBLEM.
PROBLEM is a structure with the vector 'f' in PROBLEM.f,
the integer constraints in PROBLEM.intcon,
the linear inequality constraints in PROBLEM.Aineq and
PROBLEM.bineq, the linear equality constraints
in PROBLEM.Aeq and PROBLEM.beq,
the lower bounds in PROBLEM.lb,
the upper bounds in PROBLEM.ub,
the initial point in PROBLEM.x0,
the options structure in PROBLEM.options,
and solver name 'intlinprog' in PROBLEM.solver.

[X,FVAL] = intlinprog(f,intcon,A,b,...) returns
the value of the objective function at X: FVAL = f'*X.

[X,FVAL,EXITFLAG] = intlinprog(f,intcon,A,b,...) returns
an EXITFLAG that describes the exit condition.
Possible values of EXITFLAG and the corresponding exit
conditions are:
  3  Optimal solution found with poor constraint
     feasibility.
  2  Solver stopped prematurely. Integer feasible point
     found.
  1  Optimal solution found.
  0  Solver stopped prematurely.
     No integer feasible point found.
 -1  Solver stopped by an output function
     or plot function.
 -2  No feasible point found.
 -3  Root LP problem is unbounded.
 -9  Solver lost feasibility probably due to
     ill-conditioned matrix.

[X,FVAL,EXITFLAG,OUTPUT] = intlinprog(f,A,b,...)
returns a structure OUTPUT containing information
about the optimization process. OUTPUT includes
the number of integer feasible points found and
the final gap between internally calculated bounds
on the solution. See the documentation for
a complete description.
```

```
See also linprog.

Documentation for intlinprog
```

Довідку з будь-якої функції MATLAB можна також отримати в інтернеті, ввівши в рядку пошукової системи браузера слова MATLAB та ім'я функції. Перше посилання веде на сайт [9], на сторінку з докладною інформацією про потрібну функцію.

У книзі використовуються такі позначення. Ребра чи дуги пишуться або з одним індексом: e_k, e_2, або з двома — номерами поєднуваних вершин: e_{ij}, $e_{2,5}$.

Множини записуються, як правило, великими літерами: V, а їхні елементи — малими: $v_i \in V$. Нумерація елементів множини (у тому числі вершин, ребер, дуг) починається з одиниці.

Матриці позначаються великими жирними літерами: $\boldsymbol{E}$, а вектори — малими жирними літерами: $\boldsymbol{p}$. Індексація елементів матриць та векторів починається з одиниці.

Щоб працювати з цією книгою, завантажте з сайту [10] Graph Theory Toolbox та збережіть його на своєму комп'ютері в якійсь теці, доступній MATLAB. Потім запустіть MATLAB, відкрийте редактор-налагоджувач та занесіть туди команди з кожної області введення. Збережіть та запускайте на розрахунок.

1. Введення в теорію графів

1.1. Основні означення

Рис. 1.1. Звичайний (а) та орієнтований (б) графи

Зображаються графи так, як показано на рис. 1.1. На рис. 1.1, *а* показаний звичайний (неорієнтований) граф, а на рис. 1.1, *б* — орієнтований (орграф). Але це тільки ілюстрація. Розглянемо деякі означення теорії графів з точки зору теорії множин. Це найзагальніший підхід до вивчення графів та їхніх властивостей.

Означення 1.1. *Граф* — це сукупність двох множин:

$$G = (V, E)\,, \tag{1.1}$$

де V — множина (вона називається основною), а E — множина двохелементних підмножин множини V. □

Означення 1.2. Елементи основної множини V називаються *вершинами* (vertex, vertices). □

У MATLAB для вершин використовується позначення *вузли* (node, nodes).

Будемо позначати вершини графа $v_1, v_2, \ldots, v_n$.

Означення 1.3. Кількість вершин графа G (потужність множини V) називається *розміром графа* (англ. порядок графа: the order of a graph). □

Розмір графа зазвичай позначається буквою $n : |V| = n$. Тут функція $|\ldots|$ — кількість елементів множини.

Означення 1.4. Елементи множини E називаються *ребрами* (edge). □

Як правило, ребра позначають своїми номерами: $e_1, e_2, \ldots, e_m$ або номерами поєднуваних вершин: $e_{ij}, e_{2,5}$.

Означення 1.5. Кількість ребер графа G (потужність множини E) називається *потужністю графа* (англ. розмір графа: the size of a graph). □

Потужність графа зазвичай позначають m: $|E| = m$. За основними аксіомами теорії множин однакові елементи множини E вважаються одним елементом, тому у відповідності до означення 1.1 кратних ребер у графі не може бути. За тими ж аксіомами у кожного ребра повинні бути дві різні вершини, оскільки дві однакові вершини — це все одно, що одна вершина, а елементи множини E обов'язково повинні бути двохелементними підмножинами елементів множини V. Тому петель у графі за означенням 1.1 теж немає. Приклад: з точки зору теорії множин граф, зображений на рис. 1.1, *а* — це сукупність множин $V = \{v_1, v_2, v_3, v_4\}$ та $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}, \{v_3, v_4\}\}$. Для нього $n = 4, m = 5$. Порядок нумерації вершин (елементів множини V) не має значення, оскільки за аксіомами теорії множин елементи будь-якої множини вважаються неупорядкованими. Так само не має значення й порядок нумерації ребер (елементів множини E) і вершин у ребрі (елементів множин e_1, e_2, e_3, e_4, e_5). Тому будь-який граф за означенням 1.1 — неорієнтований. Для графа з рис. 1.1, *а* множини V, E та їхні елементи показані на рис. 1.2.

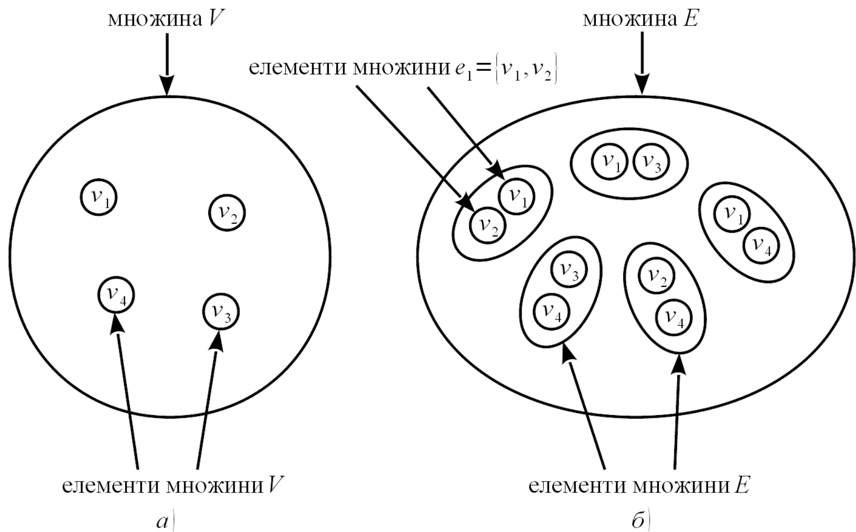


Рис. 1.2. Множини V (а) та E (б) графа G

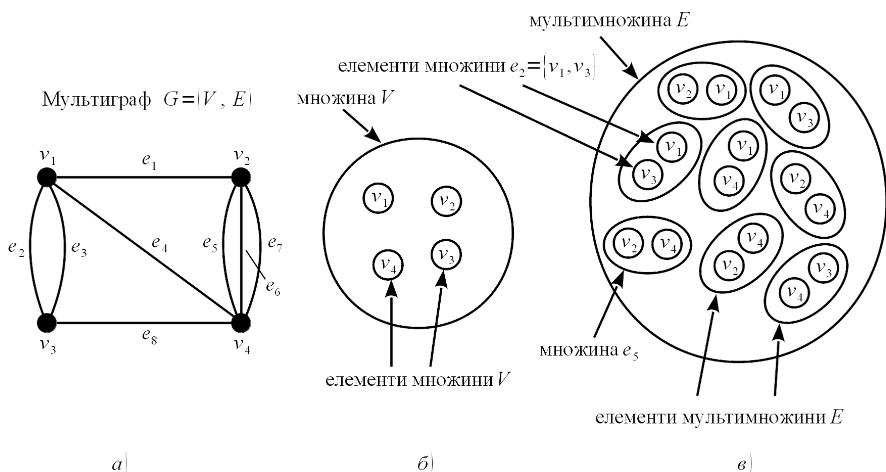


Рис. 1.3. Мультиграф G (а), його множина V (б)
та мультимножина E (в)

Але реальне життя значно ширше за означення 1.1. Інколи доводиться розглядати графи з кратними ребрами та петлями.

Означення 1.6. *Мультиграфом* (multigraph) G називається сукупність (1.1) множини V (основна множина) та мультимножини E двохелементних підмножин множини V. □

У цьому означенні E вже є мультимножиною, тобто в ній можуть бути повторювані елементи, які вважаються різними. Це відповідає кільком ребрам, що поєднують одну й ту саму пару вершин (кратним ребрам). Але елементи E, як і раніше, залишаються двохелементними множинами, тому кожне ребро повинно поєднувати дві різні вершини — петель немає. На рис. 1.3 показаний мультиграф (*а*), його множина V (*б*) та мультимножина E (*в*).

Означення 1.7. *Псевдографом* (pseudograph) G називається сукупність (1.1) множини V (основна множина) та мультимножини E двохелементних мультипідмножин множини V. □

У відповідності до цього означення допускаються не тільки однакові елементи множини E (кратні ребра), але й однакові елементи у кожній підмножині e_k, тобто ребро може з'єднувати вершину саму з собою. Такі ребра називаються петлями (a loop). Зрозуміло, що петлі у псевдографі також можуть бути кратними. На рис. 1.4 показаний псевдограф (*а*), його множина V (*б*) та мультимножина E (*в*).

Розглянемо ще одне узагальнення графа.

Означення 1.8. *Гіперграфом* (hypergraph) G називається сукуп-

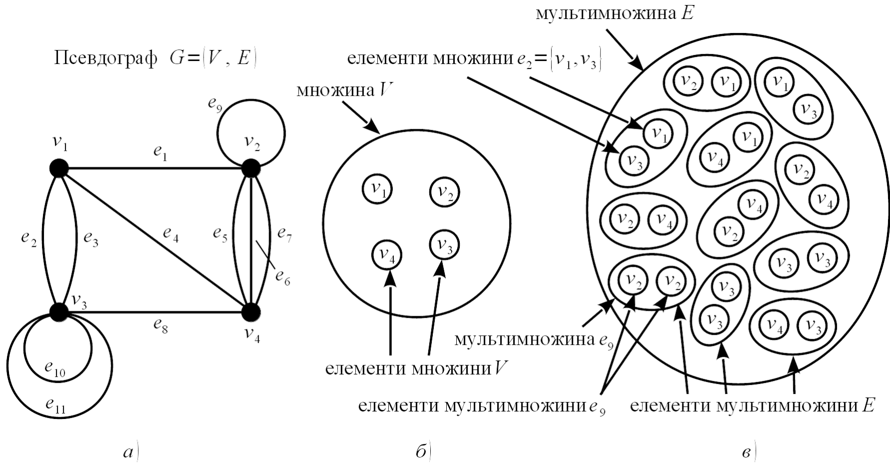


Рис. 1.4. Псевдограф G (а), його множина V (б) та мультимножина E (в)

ність (1.1) множини V (основна множина) та мультимножини E непустих підмножин множини V (не обов'язково двохелементних). □

Згідно цього означення ребра гіперграфа (вони так і називаються — *гіперребра*, hyperedge) можуть з'єднувати не тільки одну чи дві, а й будь-яку кількість вершин. Означення 1.8 припускає кратні гіперребра, у т. ч. й петлі. На рис. 1.5 наведений приклад гіперграфа. Тут для опису петель достатньо одноелементних підмножин множини V. У гіперграфа на цьому рисунку є гіперребра, що поєднують три вершини. Вони позначені лініями, що з'єднуються маленькими точками. Але можуть бути й гіперребра, що поєднують чотири, п'ять або взагалі будь-яку кількість вершин з наявних у V.

Означення 1.9. Ребро (гіперребро) e_k називається *інцидентним* (incident) до вершини v_i, якщо v_i є одним з кінців e_k. □

Означення 1.10. Ребро (гіперребро) e_k називається *суміжним* (adjacent) до гіперребра e_l, якщо існує вершина v_i, інцидентна до них обох. □

Наприклад, на рис. 1.5 ребро e_{12} є суміжним до всіх інших ребер.

Означення 1.11. Дві вершини v_i та v_j називаються *суміжними* (adjacent), якщо існує ребро (гіперребро), інцидентне до них обох. □

Означення 1.12. Граф (та всі його узагальнення) G називається графом (або мультиграфом, тощо) *зі зваженими вершинами* (graph with weighted vertices), якщо задане відображення множини V на множину дійсних чисел:

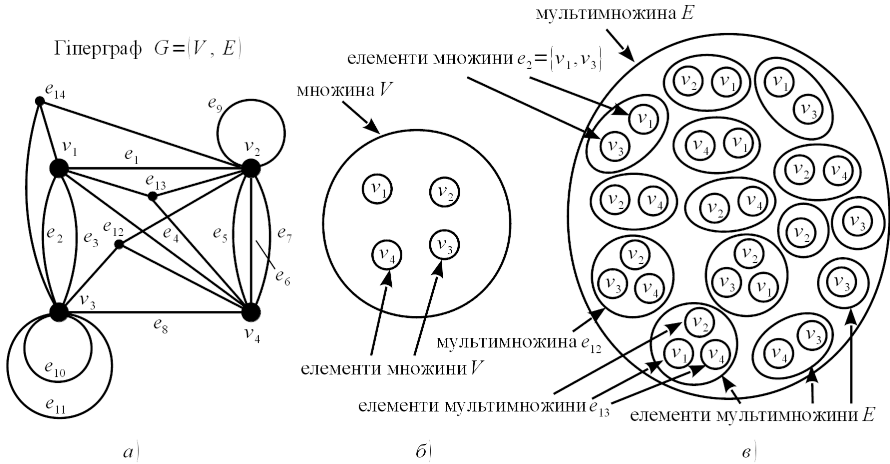


Рис. 1.5. Гіперграф G (а), його множина V (б) та мультимножина E (в)

$$\varphi :\ V \to \mathbb{R}. \tag{1.2}$$

Дійсні числа $b_i = \varphi(v_i)$, що характеризують кожну вершину, називаються в цьому випадку *вагами вершин* (weight of vertex). □

Означення 1.13. Граф (та всі його узагальнення) G називається графом *зі зваженими ребрами* (graph with weighted edges), якщо задане відображення множини E на множину дійсних чисел:

$$\psi :\ E \to \mathbb{R}. \tag{1.3}$$

Дійсні числа $c_k = \psi(e_k)$, що характеризують кожне ребро, називаються в цьому випадку *вагами ребер* (weight of edge). □

Зокрема, якщо ваги вершин або ребер — натуральні числа (або числа будь-якої зліченної множини), ми можемо взяти набір фарб, перенумерувати їх у відповідності до цих чисел, та сказати, що ми провели *розфарбовку вершин або ребер графа* (vertex or edges coloring).

Розглянемо деякі різновиди графів, що часто зустрічаються в застосуваннях.

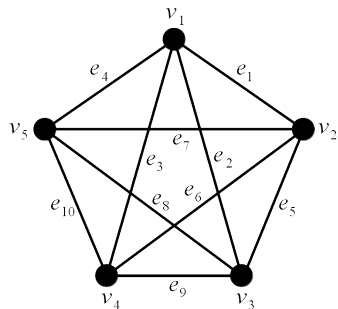


Рис. 1.6. Кліка K_5

Означення 1.14. Граф (у сенсі означення 1.1) O_n з n вершинами називається *пустим графом* (empty graph) або *нуль-графом* (null graph), якщо в ньому немає ребер: $m = 0.$ □

Означення 1.15. Граф (у сенсі означення 1.1) K_n з n вершинами називається *повним графом* (complete graph) або *клікою* (clique), якщо кожну пару його вершин поєднує ребро. □

Легко довести, що потужність кліки K_n:

$$m = \frac{n\,(n-1)}{2}. \tag{1.4}$$

Дійсно: кожна з n вершин з'єднується з будь-якою іншою з $(n-1)$, тому загальна кількість кінців ребер дорівнює $n\,(n-1)$. Але у кожного ребра є два кінці, звідсіля й отримуємо формулу (1.4). На рис. 1.6 показана кліка K_5.

Означення 1.16. Граф G називається *дводолевим* (bipartite graph), якщо множина його вершин розбивається на 2 підмножини V і W, що не перетинаються, такі, що будь-яке ребро e_k з'єднує вершину з V з вершиною з W. □

Дводолеві графи зазвичай позначають як сукупність трьох множин:

$$D = (V, W, E)\,. \tag{1.5}$$

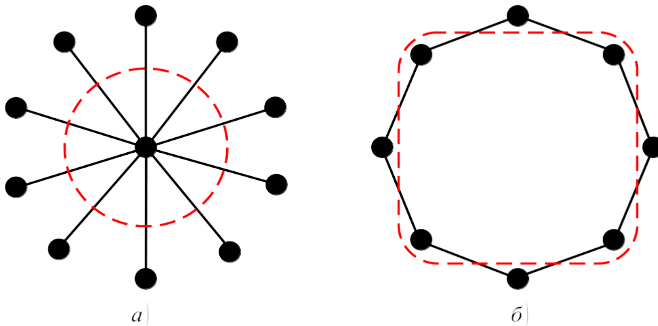


Рис. 1.7. Дводолеві графи: зірка (а) та багатокутник з парною кількістю вершин (б)

Прикладами дводолевих графів є зірка з будь-якою кількістю променів та багатокутник із парною кількістю вершин (рис. 1.7). Розбиття множини вершин на дві частини показане тут червоною штриховою лінією. Але зазвичай вершини дводолевих графів зображають на двох вертикалях або горизонталях, як на рис. 1.8, *б*.

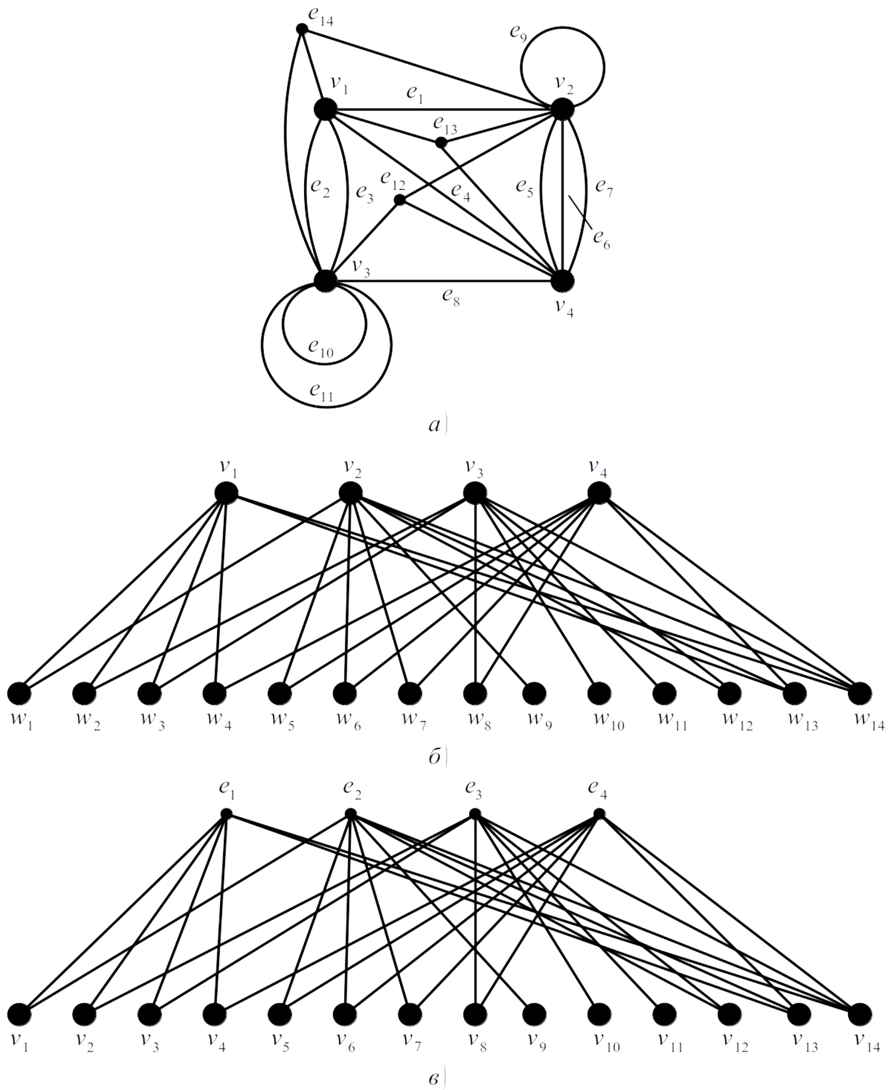


Рис. 1.8. Гіперграф (а), відповідний до нього дводолевий граф (б) та двоїстий гіперграф (в)

Дводолеві графи часто використовуються в застосуваннях. Наприклад, задачі призначення, розподілу, плани перевезень описуються дводолевими графами (працівники та роботи, джерела ресурсів та споживачі, склади та магазини). Але й у теоретичних дослідженнях нам доводиться стикатися з ними. Зокрема, будь-якому гіперграфу можна співставити у взаємно однозначну відповідність (бієкцію) дводолевий граф. Для цього достатньо вершини цього гіперграфа віднести до першої долі V, а кожному *гіперребру* поставити у відповідність *вершину* з другої долі W. Тепер з'єднуємо кожну вершину $v_i \in V$ з тими вершинами у W, які відповідають гіперребрам початкового гіперграфа, інцидентним до v_i.

Означення 1.17. Дводолевий граф $D = (V', W', E')$ називається *відповідним* (incidence) до гіперграфа $G = (V, E)$, якщо $|V'| = |V|$, $|W'| = |E|$, а ребро $e'_k = \left\{v'_i, v'_j\right\} \in E'$ тоді й тільки тоді, коли у гіперграфі G вершина v_i інцидентна до ребра e_j. □

Приклад 1.1. Побудуємо дводолевий граф, що буде відповідним до гіперграфа з рис. 1.5. На рис. 1.8, *а* показаний заданий гіперграф, а на 1.8, *б* — відповідний до нього дводолевий граф. У заданого гіперграфа $n = 4$, $m = 14$, тому в першій долі відповідного дводолевого графа буде 4 вершини, а у другій — 14. □

У заданому гіперграфі v_1 інцидентна до $e_1, e_2, e_3, e_4, e_{13}$ та e_{14}, тому у відповідному дводолевому графі з'єднуємо v_1 з $w_1, w_2, w_3, w_4, w_{13}$ та w_{14}. Так само вчиняємо і з іншими вершинами. Ми бачимо, що за гіперграфом G відповідний до нього дводолевий граф будується однозначно. І навпаки, за будь-яким дводолевим графом однозначно відновлюється початковий гіперграф, до якого цей дводолевий є відповідним. Відновлення можна провести, наприклад, так. У відповідному дводолевому графі $|V| = 4$, тому рисуємо 4 вершини початкового гіперграфа. Переглядаємо кожну вершину другої долі W: це буде ребро відновлюваного гіперграфа. Вершина w_1 суміжна з v_1 і v_2, і т. ч. ребро e_1 гіперграфа буде з'єднувати v_1 і v_2, і т. д. Якщо w_9 суміжна лише з однією вершиною v_2, то у відновлюваному початковому гіперграфі буде петля e_9 у вершині v_2. Вершина w_{12} суміжна відразу з трьома вершинами долі V: v_2, v_3 і v_4 — отже, у початковому гіперграфі з'явиться гіперребро $e_{12} = \{v_2, v_3, v_4\}$.

Що буде, якщо ми поміняємо місцями долі V та W у відповідному дводолевому графі, а потім спробуємо відновити початковий гіперграф? Ми отримаємо гіперграф, двоїстий до початкового.

Означення 1.18. Гіперграф $G' = \{V', E'\}$ називається *двоїстим* (dual) по відношенню до гіперграфа $G = \{V, E\}$, якщо $|V'| = |E|$, $|E'| = |V|$, і гіперребро e'_j є інцидентним до вершини v'_i тоді й тільки тоді, коли гіперребро e_i є інцидентним до вершини v_j. □

Приклад 1.2. Продовження прикладу 1.1. Побудуємо гіперграф,

двоїстий до гіперграфа з рис. 1.5. Ми вже побудували дводолевий граф, відповідний до нього (рис. 1.8, *б*). Поміняємо місцями долі V та W (це можна зробити в умі). Відновимо тепер гіперграф з 14 вершинами та 4 гіперребрами. Гіперребро e_1 буде з'єднувати вершини $v_1, v_2, v_3, v_4, v_{13}$ та v_{14}. Так само будуємо й інші гіперребра. Результат показаний на рис. 1.8, *в*. Вочевидь, гіперграф, двоїстий до двоїстого, співпадає з початковим. Тобто ми можемо казати про пару взаємно двоїстих гіперграфів. □

Означення 1.19. Орієнтований граф, або *орграф* (digraph) G – це сукупність двох множин V і E:

$$G = (V, E)\,, \tag{1.6}$$

де V – основна множина (вершини), а E – множина упорядкованих двохелементних підмножин множини V. Ці підмножини називаються *дугами* (arc) або *стрілками* (arrow). □

Приклад орграфа – на рис. 1.1, *б*. Як і для графів, для орграфів можна ввести до розгляду кратні дуги та петлі (можливо, теж кратні). Орграф та його узагальнення можуть також мати зважені або розфарбовані вершини та (або) дуги. А ось про орієнтовані гіперграфи я взагалі не чув.

1.2. Як задати граф

Граф та його узагальнення — це сукупність двох множин V та E. Їх і треба задати для розв'язання різних задач на графах.

Почнемо з вершин. Вершини є елементами множини V. Будемо нумерувати вершини натуральними числами від 1 до n. Тоді для опису вершин достатньо задати одне натуральне число n (а, може, і його не треба задавати, як ми побачимо далі). Цього цілком достатньо для розв'язання задач теорії графів. Але для рисування треба знати ще й координати вершин. З координатами можна вчинити так: задати масив дійсних чисел розміром $n \times 2$ або $n \times 3$. У першому стовпці цього масиву задаємо абсциси вершин, а у другому — ординати. Якщо зручніше рисувати граф у просторі (як якусь аксонометричну проекцію), то у третьому стовпці задаємо аплікати. Якщо координати не задавати, можна рисувати граф з лінійним розташуванням вершин, як на рис. 1.8, *в*. Або зображати вершини графа на колі у вершинах правильного n-кутника.

Множину E найзручніше задавати у вигляді списку ребер або дуг — масиву натуральних чисел розміром $m \times 2$, у кожному рядку якого записані номери вершин, що з'єднуються відповідним ребром або дугою. Наприклад, для графа з рис. 1.1, *а* та орграфа з рис. 1.1, *б* список ребер (дуг) має вигляд:

```
1   2
```

```
1  3
1  4
2  4
3  4
```

Для неорієнтованого графа елементи кожного рядка можна переставляти, а для орграфа — ні: будемо вважати, що кожна стрілка спрямована від вершини з номером, що знаходиться у першому стовпці, до вершини, номер якої вказаний у другому стовпці. При такому заданні графів немає проблем з кратними ребрами (дугами) та петлями. Якщо є кратне ребро, то відповідний рядок повторюється потрібну кількість разів. Для петлі у рядку буде два однакових числа.

Саме так у MATLAB і задаються графи та орграфи. Це — найпростіший варіант: треба задати номери вершин-початків та номери вершин-кінців кожного ребра або дуги. Наприклад, можна задати два вектори-рядки або вектори-стовпці однакової довжини з натуральних чисел. Кількість вершин при цьому визначається максимальним номером вершини. Якщо є висячі вершини з номерами, більшими за максимальне число в ребрах (дугах), то треба ще явно задати кількість вершин. Якщо ребра (дуги) зважені, ще додатково задаємо вектор з вагами ребер (дуг). Так само задаємо й ваги вершин.

Ось кілька прикладів створення та рисування графів та орграфів.

Приклад 1.3. Створимо простий граф, що містить 11 вершин та 22 ребра. Виведемо в командне вікно MATLAB інформацію про нього. Нарисуємо граф найпростішими засобами (рис. 1.9).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
disp("Граф:")
G = graph(s,t) % незважений граф
disp("Ребра:")
G.Edges % ребра графа
fprintf("Кількість ребер m = %d\n",numedges(G))
disp("Вершини:")
G.Nodes % вершини графа
fprintf("Кількість вершин n = %d\n",numnodes(G))
figure % нове вікно фігури
plot(G) % нарисували граф
title("Найпростіше рисування графа")
```

```
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("SimplePaintGraph","-dpng") % зберегли рисунок у файл
```

```
Граф:
G =
  graph with properties:

    Edges: [22×1 table]
    Nodes: [11×0 table]
Ребра:
ans =
  22×1 table
    EndNodes
    ________
     1     2
     1     3
     1     5
     1     6
     2     3
     2     4
     3     4
     3     6
     3     7
     4     7
     5     6
     5     8
     6     7
     6     8
     6    11
     7    10
     7    11
     8     9
     8    11
     9    10
     9    11
    10    11
Кількість ребер m = 22
Вершини:
ans =
  11×0 empty table
```

```
Кількість вершин n = 11
```

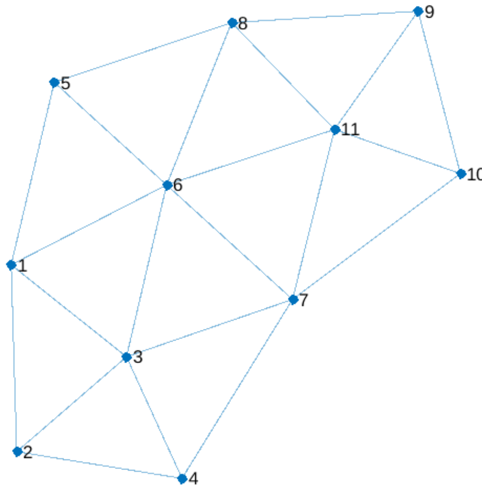


Рис. 1.9. Найпростіше рисування графа засобами MATLAB

У MATLAB використовується моделе-орієнтований підхід (МОП) — своєрідна реалізація ідей об'єктно-орієнтованого програмування (ООП). Поняття "модель" в MATLAB ідентичне до понять "клас" в C++, C#, Java або "об'єкт" у Pascal. У моделі є поля та методи. Доступ до полів здійснюється так само, як і в інших мовах програмування:

```
MyRes = MyModel.MyField;
```

Але методи викликаються дещо інакше. В інших мовах програмування ми використовуємо такий синтаксис:

```
[MyRes =] MyModel.MyMethod[(MyArgs)];
```

А в моделях MATLAB методи викликаються так:

```
[MyRes =] MyMethod(MyModel[, MyArgs]);
```

В моделях MATLAB підтримуються інкапсуляція, наслідування та поліморфізм. Але ці властивості майже ніколи не використовуються, оскільки в MATLAB є велика кількість вбудованих готових моделей для різноманітних задач, зокрема і для графів та орграфів.

У цьому прикладі ми бачимо, що команда graph є конструктором екземпляра моделі графа з ідентифікатором G. Ця модель містить поля G.Edges та G.Nodes з інформацією про ребра та вершини. Відповідні поля є таблицями (ще один приклад моделі) зі своїми полями та методами. А команди numedges, numnodes та plot — це методи моделі graph.

Зверніть увагу: у неорієнтованому графі номери вершин кожного ребра упорядковуються в порядку зростання, а потім і самі ребра теж упорядковуються в порядку зростання номерів першої та другої вершин.

Метод plot за умовчанням рисує граф, виходячи з його структури, так, щоб він, на думку авторів, виглядав привабливіше. □

Приклад 1.4. Додамо до графа з попереднього прикладу 1.3 ваги ребер, задамо координати вершин та нарисуємо в цих координатах. Позначимо на ребрах їхні ваги (рис. 1.10).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 ...
  2 2 5 4 5]; % ваги ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t,w); % граф зі зваженими ребрами
disp("Зважені ребра:")
G.Edges % зважені ребра графа
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,...
  "EdgeLabel",G.Edges.Weight) % нарисували граф
title("Граф зі зваженими ребрами")
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("GraphWithWeightedEdges","-dpng") % зберегли рисунок
```

```
Зважені ребра:
ans =
  22×2 table
    EndNodes    Weight

    --------    ------
     1     2      5
     1     3      2
```

```
         1          5            3
         1          6            2
         2          3            5
         2          4            5
         3          4            2
         3          6            5
         3          7            2
         4          7            3
         5          6            1
         5          8            5
         6          7            1
         6          8            2
         6         11            3
         7         10            3
         7         11            2
         8          9            5
         8         11            2
         9         10            5
         9         11            4
        10         11            2
```

Рис. 1.10. Граф зі зваженими ребрами

Бачимо, що в таблиці `G.Edges`, крім поля `G.Edges.EndNodes`, з'явилося ще одне поле `G.Edges.Weight` з вагами ребер.

Додаткові параметри рисування задаються в методі `plot` у вигляді пар параметр-значення. □

Приклад 1.5. Створити та нарисувати орграф зі зваженими вершинами (рис. 1.11).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки дуг
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці дуг
d = [2 3 3 4 1 2 3 3 5 1 5]; % ваги вершин
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
disp("Орграф:")
G = digraph(s,t) % орграф
disp("Дуги:")
G.Edges % дуги орграфа
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",d) % нарисували орграф
title("Орграф зі зваженими вершинами")
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("DigraphWithWeightedNodes","-dpng") % зберегли рисунок
```

```
Орграф:
G =
  digraph with properties:

    Edges: [22×1 table]
    Nodes: [11×0 table]
Дуги:
ans =
  22×1 table
    EndNodes
    ________
     1     3
     1     5
     1     6
     2     1
     2     3
     2     4
     3     4
     3     6
     3     7
     4     7
     5     6
     5     8
```

```
     6     7
     6     8
     6    11
     7    10
     7    11
     8     9
     8    11
    10     9
    11     9
    11    10
```

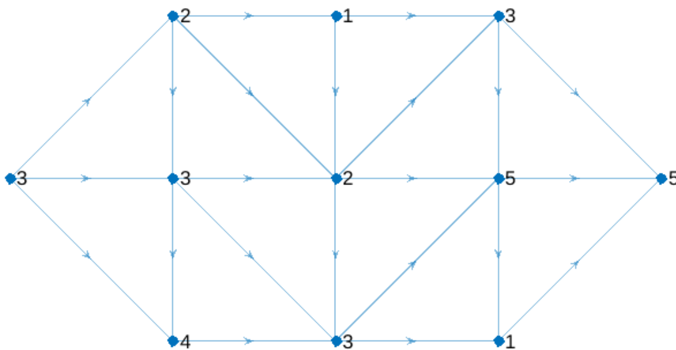


Рис. 1.11. Орграф зі зваженими вершинами

Тут `digraph` — конструктор моделі орграфа. Зрозуміло, що в дугах орграфа номери вершин переставляти не можна. Але самі дуги можна упорядкувати. □

Ми розглянули, як задаються графи та орграфи. Мультиграфи та псевдографи задаються так само. Але гіперграфи задавати так не вдасться. Для них треба використовувати масив розміром не $m \times 2$, а $m \times e_{\max}$, де $e_{\max}$ — максимальна кількість вершин, які може з'єднувати гіперребро. Тоді коротші рядки доповнюються нулями. Наприклад, вхідна інформація про гіперграф з рис. 1.8, *в* буде такою:

```
1   2   3   4  13  14   0   0
1   5   6   7   9  12  13  14
2   3   8  10  11  12  14   0
4   5   6   7   8  12  13   0
```

У такій інформації про гіперграф немає двозначностей, оскільки ми нумеруємо вершини числами від 1 до n, а вершини с номером 0 немає.

У багатьох мовах програмування, зокрема, і в MATLAB, є можливість задавати масиви (списки, структури тощо) з елементами різних типів. Зокрема, з рядками різної довжини. Тому, якщо є така можливість, можете задавати вхідну інформацію про гіперграф з рис. 1.8, *в* так:

```
1  2  3  4 13 14
1  5  6  7  9 12 13 14
2  3  8 10 11 12 14
4  5  6  7  8 12 13
```

Зауважимо, що наведений вище набір даних однозначно характеризує також і гіперграф з рис. 1.8, *а*, який є двоїстим до гіперграфа з рис. 1.8, *в*. Для кожної вершини (рядка) тут вказані номери інцидентних до неї ребер. І навпаки: матриця списку гіперребер для гіперграфа з рис. 1.8, *а* є такою:

```
1  2  0
1  3  0
1  3  0
1  4  0
2  4  0
2  4  0
2  4  0
3  4  0
2  0  0
3  0  0
3  0  0
2  3  4
1  2  4
1  2  3
```

або такою (без нулів, з рядками змінної довжини):

```
1  2
1  3
1  3
1  4
2  4
2  4
2  4
3  4
2
3
3
```

```
2  3  4
1  2  4
1  2  3
```

І цей набір даних також однозначно характеризує гіперграф, показаний на рис. 1.8, *в*, що є двоїстим до гіперграфа з рис. 1.8, *а*. Тут теж для кожної вершини (рядка) вказані номери інцидентних до неї ребер. Тому інформацію про гіперграф (точніше, про пару взаємно двоїстих гіперграфів) можна задавати й у вигляді масиву з n рядків, у кожному з яких перелічені номери ребер, інцидентних до відповідної вершини, і коротші рядки за необхідності доповнювати нулями.

Вбудованого механізму для роботи з гіперграфами в MATLAB наразі немає, але на сайті [10] є пакети, створені користувачами. Можете ними скористатися.

1.3. Матричні представлення графів

Для розв'язання задач зручно представити граф у матричному вигляді. Розглянемо деякі матриці, що характеризують граф.

Означення 1.20. *Матрицею інцидентності* (incidence matrix) гіперграфа G називається булівська (з елементами true та false) або бінарна (з елементами 1 та 0) матриця $\boldsymbol{A}$ розміром $n \times m$, кожен елемент якої $a_{ik} = \text{true}$ (або $a_{ik} = 1$) тоді й тільки тоді, коли вершина v_i інцидентна до ребра e_k. $\square$

Для простого графа у кожному стовпці матриці інцидентності буде рівно дві одиниці, а решта нулі, і всі стовпці будуть різними. Так, для графа з рис. 1.1, *а* матриця інцидентності має вигляд:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{1.7}$$

У матриці інцидентності мультиграфа з'являться однакові стовпці. Для псевдографів у стовпцях, що відповідають петлям, буде лише одна одиниця (або двійка, якщо це обумовлено конкретною задачею). У кожному стовпці матриці інцидентності гіперграфа може бути скільки завгодно одиниць. Ось якою є, наприклад, матриця інцидентності гіперграфа з рис. 1.5, *а*:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}. \tag{1.8}$$

Її розміри 4×14. Оскільки v_1 інцидентна до $e_1, e_2, e_3, e_4, e_{13}$ та e_{14}, то в першому рядку одиничними будуть елементи з номерами 1, 2, 3, 4, 13 та 14. Так само заповнюємо й інші рядки.

Якщо побудувати матрицю інцидентності для двоїстого гіперграфа, показаного на рис. 1.8, *в*, то неважко помітити, що вона буде транспонованою до матриці інцидентності початкового гіперграфа (1.8). Можна довести відповідну теорему — вона майже очевидна з означення 1.18.

Як бачимо, за заданим списком ребер ця матриця будується однозначно. Псевдокод побудови матриці інцидентності графа (мультиграфа, псевдографа) за заданим масивом E розміром $m \times 2$ зі списком ребер виглядає так.

```
for k=1 step 1 to m do {переглядаємо рядки масиву ребер E}
begin {for k}
  for i=1 step 1 to n do
  begin {for i}
    A(i,k)=0 {обнуляємо стовпчик матриці інцидентності}
  end {for i}
  A(E(k,1),k)=1 {записуємо дві одиниці на потрібні місця}
  A(E(k,2),k)=1 {для гіперграфа замість двох одиниць буде цикл}
end {for k}
```

Зворотна задача теж розв'язується однозначно з точністю до порядку вершин у кожному гіперребрі. Алгоритм виглядає так.

1. Додаємо всі рядки матриці інцидентності $\boldsymbol{A}$. В отриманому рядку кожне число — це кількість вершин, інцидентних до відповідного гіперребра. Найбільше з цих чисел — це $e_{\max}$.
2. Описуємо масив E для списку ребер розміром $m \times e_{\max}$. Обнуляємо його.
3. Переглядаємо кожен стовпчик матриці інцидентності $\boldsymbol{A}$, і заповнюємо відповідний рядок масиву E номерами одиничних елементів.

Псевдокод третього кроку цього алгоритму виглядає так.

```
for k=1 step 1 to m do {стовпці матриці інцидентності}
begin {for k}
  j=1 {номер стовпця масиву E: 1, 2, ..., emax}
  for i=1 step 1 to n do {ел-ти стовпця матриці інцидентності}
  begin {for i}
    if (A(i,k)>0.5) {ненульовий елемент стовпця}
    then
    begin {if-then}
      E(k,j)=i
      j=j+1
```

```
    end {if-then}
  end {for i}
end {for k}
```

У MATLAB матриця інцидентності графа та орграфа (див. далі означення 1.24) будується за допомогою методу `incidence`. Він повертає розріджену матрицю орграфа, а її модуль буде матрицею інцидентності графа чи мультиграфа. Для псевдографів цей метод не працює.

Приклад 1.6. Створити мультиграф (рис. 1.12) та нарисувати його матрицю інцидентності (рис. 1.13).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10 2 2 4]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9 3 4 2]; % кінці ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений мультиграф
disp("Ребра мультиграфа:")
G.Edges % ребра мультиграфа
A = abs(incidence(G)); % матриця інцидентності мультиграфа
figure % нове вікно фігури
plot(G,"XData",x,"YData",y) % нарисували мультиграф
title("Мультиграф")
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MultiGraph","-dpng") % зберегли рисунок у файл
figure % нове вікно фігури
imagesc(A) % поверхня
colormap(flipud(hot(256))) % чорно-біла палітра
axis("equal") % однаковий масштаб уздовж осей координат
xlim([0.5 numedges(G)+0.5]) % границі вздовж Ox
ylim([0.5 numnodes(G)+0.5]) % границі вздовж Oy
title("Матриця інцидентності мультиграфа")
print("IncMatrMultiGraph","-dpng") % зберегли рисунок у файл
```

```
Ребра мультиграфа:
ans =
  25×1 table
    EndNodes
    --------
```

```
     1     2
     1     3
     1     5
     1     6
     2     3
     2     3
     2     4
     2     4
     2     4
     3     4
     3     6
     3     7
     4     7
     5     6
     5     8
     6     7
     6     8
     6    11
     7    10
     7    11
     8     9
     8    11
     9    10
     9    11
    10    11
```

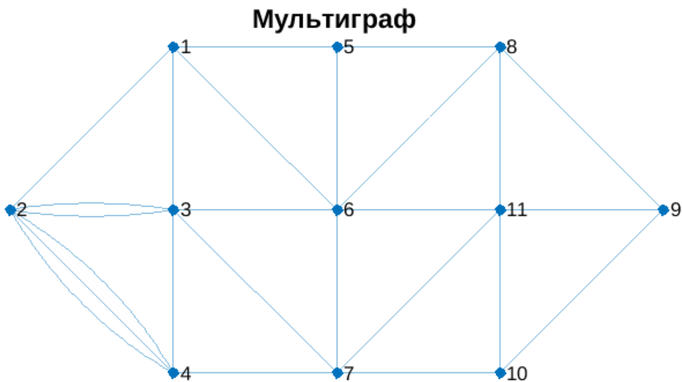


Рис. 1.12. Мультиграф

У матриці інцидентності на рис. 1.13 чорним кольором позначені одиниці, а білим — нулі. □

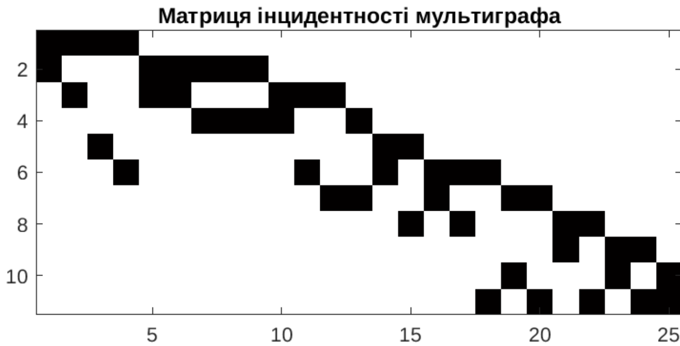


Рис. 1.13. Матриця інцидентності мультиграфа

У кожному стовпці матриці інцидентності простого графа або мультиграфа рівно дві одиниці. А у кожному рядку — стільки одиниць, скільки ребер є інцидентними до цієї вершини.

Означення 1.21. *Ступенем вершини* (degree of a vertex) d_i називається кількість ребер, інцидентних до неї. □

Ступінь вершини v_i є сумою елементів i-го рядка матриці інцидентності $\boldsymbol{A}$. Оскільки елементи рядка — це нулі та одиниці, можна обчислювати d_i як суму квадратів елементів i-го рядка цієї матриці, або як скалярний добуток i-го рядка самого на себе:

$$d_i = (\boldsymbol{a}_i, \boldsymbol{a}_i)\,, \tag{1.9}$$

де $\boldsymbol{a}_i$ — i-й рядок матриці інцидентності $\boldsymbol{A}$, що розглядається як вектор-стовпчик.

Розглянемо тепер скалярний добуток двох різних рядків матриці інцидентності $\boldsymbol{A}$. Для простого графа у кожному стовпці цієї матриці рівно дві одиниці, і однакових стовпців немає. Тому, якщо записати один під одним два різні рядки матриці $\boldsymbol{A}$, то одиниця під одиницею може зустрітися не більше одного разу, а може й зовсім не зустрітися. Отже, скалярний добуток i-го рядка на j-й може дорівнювати або 1, або 0. Він буде дорівнювати 1, якщо в i-му та j-му рядках є одиниця під одиницею, тобто є ребро, що поєднує v_i з v_j. Щоб обчислити скалярні добутки всіх рядків на всі, простіше за все помножити матрицю інцидентності $\boldsymbol{A}$ на транспоновану до неї. Наприклад, для матриці (1.7) простого графа з рис. 1.1, *а* результат множення буде таким:

$$\boldsymbol{A}\boldsymbol{A}^T = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 1 & 1 & 3 \end{pmatrix}. \tag{1.10}$$

На головній діагоналі — ступені вершин, а позадіагональні елементи дорівнюють 1, якщо відповідні вершини суміжні, і 0, якщо ні.

Означення 1.22. *Матрицею суміжності вершин* (adjacency matrix) простого графа G називається булівська (з елементами true та false) або бінарна (з елементами 0 та 1) матриця $\boldsymbol{B}$ розміром $n \times n$, кожен елемент якої $b_{ij} = \text{true}$ (або $b_{ij} = 1$) тоді й тільки тоді, коли v_i є суміжною з v_j. На головній діагоналі ставляться нулі. □

Якщо v_i суміжна з v_j, то й v_j є суміжною з v_i. Тому матриця суміжності вершин буде симетричною. Ось як вона виглядає для графа з рис. 1.1, *а*:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}. \tag{1.11}$$

Порівнюючи формули (1.10) та (1.11), бачимо, як пов'язані між собою матриці інцидентності та суміжності вершин:

$$\boldsymbol{B} = \boldsymbol{A}\boldsymbol{A}^T - \boldsymbol{D}, \tag{1.12}$$

де $\boldsymbol{D}$ — діагональна матриця зі ступенями вершин на головній діагоналі.

Як і матриця інцидентності, матриця суміжності вершин однозначно будується за масивом зі списком ребер. Якщо є в наявності матриця інцидентності, можна скористатися формулою (1.12). Якщо ж її немає і вона не потрібна, можна відразу будувати матрицю суміжності вершин `B` за допомогою наступного алгоритму.

```
for i=1 step 1 to n do
begin {for i}
  for j=1 step 1 to n do
  begin {for j}
    B(i,j)=0 {обнуляємо матрицю суміжності вершин}
  end {for j}
end {for i}
for k=1 step 1 to m do
```

```
begin {for k}
  B(E(k,1),E(k,2))=1
  B(E(k,2),E(k,1))=1
end {for k}
```

Тут E — масив зі списком ребер розміром $m \times 2$.

У MATLAB для побудови матриці суміжності вершин графа, мультиграфа чи псевдографа є метод adjacency. В ньому кратні ребра та петлі ігноруються.

Приклад 1.7. Створити псевдограф (рис. 1.14) та нарисувати його матрицю суміжності вершин (рис. 1.15).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10 2 2 4 6 6 8]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9 3 4 2 6 6 8]; % кінці ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений псевдограф
B = adjacency(G); % матриця суміжності вершин псевдографа
figure % нове вікно фігури
plot(G,"XData",x,"YData",y) % нарисували псевдограф
title("Псевдограф")
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("PseudoGraph","-dpng") % зберегли рисунок у файл
figure % нове вікно фігури
imagesc(B) % поверхня
colormap(flipud(hot(256))) % чорно-біла палітра
axis("equal") % однаковий масштаб уздовж осей координат
xlim([0.5 numnodes(G)+0.5]) % границі вздовж Ox
ylim([0.5 numnodes(G)+0.5]) % границі вздовж Oy
title("Матриця суміжності вершин псевдографа")
print("AdjVerMatrPseudoGraph","-dpng") % зберегли рисунок
```

Зворотна задача теж розв'язується однозначно з точністю до нумерації ребер та вершин у ребрі. Алгоритм дуже простий: переглядаємо всі елементи матриці $\boldsymbol{B}$ над головною діагоналлю. Якщо побачимо ненульовий елемент, то його індекси (номер рядка та стовпця) — це буде черговий рядок масиву зі списком ребер. Ось реалізація цього алгоритму на псевдокоді.

```
k=1 {номер рядка масива E}
```

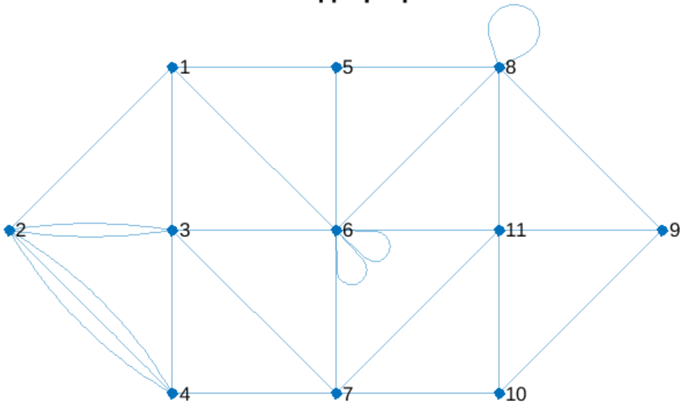


Рис. 1.14. Псевдограф

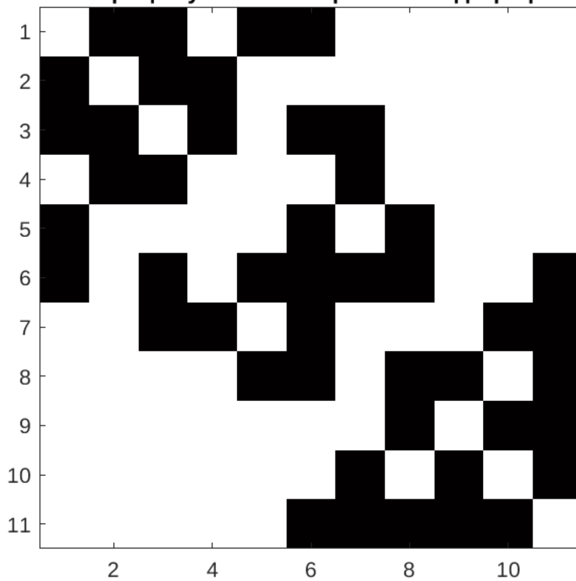


Рис. 1.15. Матриця суміжності вершин псевдографа □

```
for i=1 step 1 to n-1 do
begin {for i}
  for j=i+1 step 1 to n do
  begin {for j}
    if (B(i,j)>0.5) {ненульовий ел-нт матриці суміжності вершин}
    then
    begin {if-then}
      E(1,k)=i
      E(2,k)=j
      k=k+1
    end {if-then}
  end {for j}
end {for i}
```

Якщо це потрібно для якоїсь конкретної задачі, для мультиграфів замість одиниць можна ставити кількість ребер (тоді матриця вже не буде ані булівською, ані бінарною). Для псевдографів на головній діагоналі будуть не нулі, а кількість петель. Для зважених графів замість одиниць можна задавати ваги відповідних ребер.

Якщо у дводолевому графі спочатку перенумерувати вершини однієї долі (r вершин), а потім другої (s вершин), то матриця суміжності вершин буде мати структуру:

$$\boldsymbol{B} = \begin{pmatrix} \mathbf{0} & \boldsymbol{B}_{rs} \\ \boldsymbol{B}_{rs}^T & \mathbf{0} \end{pmatrix}, \tag{1.13}$$

де $\mathbf{0}$ — нульова матриця відповідних розмірів, $\boldsymbol{B}_{rs}$ — прямокутна матриця суміжності вершин з різних долей розміром $r \times s$.

Розглянемо тепер скалярні добутки не рядків, а стовпців матриці інцидентності $\boldsymbol{A}$. Для простого графа у кожному стовпці $\boldsymbol{A}$ рівно дві одиниці, тому скалярний добуток кожного стовпця самого на себе дорівнює двом. Однакових стовпців (кратних ребер) у простому графі немає, тому, якщо порівняти два різні стовпці $\boldsymbol{A}$ з двома одиницями в кожному, то співпадінь одиниць може бути не більше одного, а може й взагалі не бути. Таке співпадіння двох одиниць буде тоді, коли є вершина, інцидентна обом ребрам, тобто ребра суміжні. Щоб знайти скалярні добутки всіх пар стовпців, треба помножити $\boldsymbol{A}^T$ на $\boldsymbol{A}$. Наприклад, для матриці (1.7) простого графа з рис. 1.1, *а* результат множення буде таким:

$$
\boldsymbol{A}^T\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \end{pmatrix}. \tag{1.14}
$$

На головній діагоналі знаходяться двійки, а позадіагональні елементи дорівнюють 1, якщо відповідна пара ребер суміжна, і 0, якщо ні.

Означення 1.23. *Матрицею суміжності ребер* (edge-adjacency matrix) простого графа G називається булівська (з елементами true та false) або бінарна (з елементами 1 та 0) матриця $\boldsymbol{C}$ розміром $m \times m$, кожен елемент якої $c_{kl} = \text{true}$ (або $c_{kl} = 1$) тоді й тільки тоді, коли ребро e_k є суміжним з e_l. На головній діагоналі ставляться нулі. □

Ця матриця також є симетричною. Для графа з рис. 1.1, *а* вона має вигляд:

$$
\boldsymbol{C} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \tag{1.15}
$$

Порівнюючи формули (1.14) та (1.15), бачимо зв'язок між матрицями інцидентності та суміжності ребер:

$$
\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{A} - 2\boldsymbol{E}. \tag{1.16}
$$

Якщо є в наявності матриця інцидентності $\boldsymbol{A}$, то побудувати матрицю суміжності ребер $\boldsymbol{C}$ можна (однозначно) за формулою (1.16). Якщо ж її немає і вона не потрібна, можна відразу однозначно будувати матрицю суміжності ребер $\boldsymbol{C}$ за допомогою такого алгоритму.

```
for k=1 step 1 to m do
begin {for k}
  for l=1 step 1 to m do
  begin {for l}
    C(k,l)=0 {обнуляємо матрицю суміжності ребер}
  end {for l}
end {for k}
for k=1 step 1 to m-1 do
 begin {for k}
  for l=k+1 step 1 to m do
```

```
  begin {for l}
    if ( (E(k,1)=E(l,1)) or (E(k,2)=E(l,2)) or
      (E(k,1)=E(l,2)) or (E(k,2)=E(l,1)) )
    then
    begin {if-then}
      C(k,l)=1
      C(l,k)=1
    end {if-then}
  end {for l}
end {for k}
```

Як і раніше, тут `E` — масив зі списком ребер розміром $m \times 2$.

Але за матрицею суміжності ребер $\boldsymbol{C}$ відновити масив зі списком ребер неможливо: у матриці $\boldsymbol{C}$ немає інформації про те, з якого саме кінця ребро e_k є суміжним до ребра e_l.

У MATLAB немає вбудованого методу для побудови матриці суміжності ребер, але реалізація формули (1.16) для простих графів не викликає труднощів.

Приклад 1.8. Для простого графу з прикладу 1.3 створити та нарисувати його матрицю суміжності ребер (рис. 1.16).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
A = abs(incidence(G)); % матриця інцидентності графа
C = A'*A-2*speye(numedges(G)); % матриця суміжності ребер
figure % нове вікно фігури
imagesc(C) % поверхня
colormap(flipud(hot(256))) % чорно-біла палітра
axis("equal") % однаковий масштаб уздовж осей координат
xlim([0.5 numedges(G)+0.5]) % границі вздовж Ox
ylim([0.5 numedges(G)+0.5]) % границі вздовж Oy
title("Матриця суміжності ребер графа")
print("AdjEdgeMatrGraph","-dpng") % зберегли рисунок у файл
```

Тут чорні клітинки — одинці, а білі — нулі. □

Розглянемо тепер матриці, що характеризують орграфи.

Означення 1.24. *Матрицею інцидентності* (incidence matrix) орграфа G називається матриця $\boldsymbol{A}$ розміром $n \times m$, кожен елемент якої

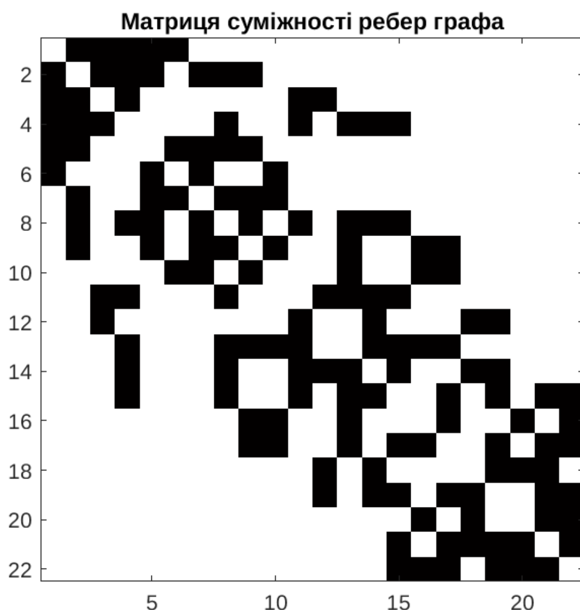


Рис. 1.16. Матриця суміжності ребер графа

$a_{ik} = 1$ тоді й тільки тоді, коли з вершини v_i виходить дуга e_k, $a_{ik} = -1$ тоді й тільки тоді, коли у вершину v_i входить дуга e_k, а в інших випадках $a_{ik} = 0$. □

Ось як виглядає матриця інцидентності для орграфа з рис. 1.1, *б*:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{pmatrix}. \tag{1.17}$$

Для орграфа з кратними дугами у цій матриці будуть повторювані стовпці. Якщо є петлі, можна, наприклад, ставити одну одиницю у відповідному стовпці.

Як і для звичайних (неорієнтованих) графів, ця матриця однозначно будується за масивом `E` зі списком дуг розміром $m \times 2$. Відповідний алгоритм виглядає так.

```
for k=1 step 1 to m do {переглядаємо рядки масиву дуг E}
begin {for k}
  for i=1 step 1 to n do
  begin {for i}
```

```
    A(i,k)=0 {обнуляємо стовпчик матриці інцидентності}
  end {for i}
  A(E(k,1),k)=1 {хвіст дуги}
  A(E(k,2),k)=-1 {вістря дуги}
end {for k}
```

Зворотна задача теж розв'язується однозначно з точністю до нумерації дуг. Ось реалізація цього алгоритму.

```
for k=1 step 1 to m do {стовпці матриці інцидентності}
begin {for k}
  for i=1 step 1 to n do {ел-ти стовпця матриці інцидентності}
  begin {for i}
    if (A(i,k)>0.5) {хвіст (кінець) дуги}
    then
    begin {if-then}
      E(k,1)=i
    end {if-then}
    else if (A(i,k)<-0.5) {вістря (початок) дуги}
    then
    begin {if-then}
      E(k,2)=i
    end {if-then}
  end {for i}
end {for k}
```

У MATLAB метод `incidence` повертає матрицю інцидентності орграфа. Для графа чи мультиграфа треба просто взяти її модуль (див. означення 1.20 та приклад 1.6).

Приклад 1.9. Для орграфу з прикладу 1.5 (рис. 1.11) створити та нарисувати його матрицю інцидентності (рис. 1.17).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки дуг
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці дуг
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = digraph(s,t); % орграф
A = incidence(G); % матриця інцидентності орграфа
imagesc(A) % поверхня
colormap(gray(256)) % сіра палітра
```

```
axis("equal") % однаковий масштаб уздовж осей координат
xlim([0.5 numedges(G)+0.5]) % границі вздовж Ox
ylim([0.5 numnodes(G)+0.5]) % границі вздовж Oy
title("Матриця інцидентності орграфа")
print("IncMatrDiGraph","-dpng") % зберегли рисунок у файл
```

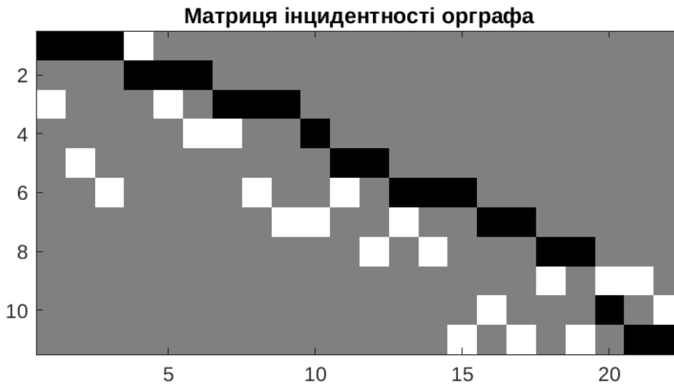


Рис. 1.17. Матриця інцидентності орграфа

Чорні клітинки тут — числа -1, білі — 1, а сірі — 0. □

Означення 1.25. *Матрицею суміжності вершин* (adjacency matrix) орграфа G називається матриця $\boldsymbol{B}$ розміром $n \times n$, кожен елемент якої $b_{ij} = 1$ тоді й тільки тоді, коли з v_i виходить дуга у v_j, $b_{ij} = -1$ тоді й тільки тоді, коли у v_i входить дуга з v_j, і $b_{ij} = 0$ в усіх інших випадках. □

Неважко помітити, що матриця суміжності вершин орграфа буде кососиметричною. Для орграфа рис. 1.1, *б* вона має вигляд:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}. \tag{1.18}$$

Як і для графа, ця матриця будується однозначно за масивом зі списком дуг:

```
for i=1 step 1 to n do
begin {for i}
  for j=1 step 1 to n do
  begin {for j}
    B(i,j)=0 {обнуляємо матрицю суміжності вершин}
  end {for j}
```

```
end {for i}
for k=1 step 1 to m do
begin {for k}
  B(E(k,1),E(k,2))=1
  B(E(k,2),E(k,1))=-1
end {for k}
```

Зворотна задача теж розв'язується однозначно з точністю до нумерації дуг:

```
k=1 {номер рядка масива E}
for i=1 step 1 to n-1 do
begin {for i}
  for j=i+1 step 1 to n do
  begin {for j}
    if (B(i,j)>0.5) {хвіст (кінець) дуги}
    then
    begin {if-then}
      E(1,k)=i
      E(2,k)=j
      k=k+1
    end {if-then}
    else if (B(i,j)<-0.5) {вістря (початок) дуги}
    then
    begin {if-then}
      E(1,k)=j
      E(2,k)=i
      k=k+1
    end {if-then}
  end {for j}
end {for i}
```

У MATLAB метод `adjacency` повертає для орграфа матрицю досяжності (див. далі означення 6.8): $b_{ij} = 1$, коли з v_i виходить дуга у v_j, і $b_{ij} = 0$ в усіх інших випадках. Щоб отримати матрицю суміжності за означенням 1.25, треба ще відняти від отриманої матриці транспоновану до неї.

Приклад 1.10. Для орграфу з прикладу 1.5 (рис. 1.11) створити та нарисувати його матрицю суміжності вершин (рис. 1.18).

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки дуг
```

```
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці дуг
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = digraph(s,t); % орграф
B1 = adjacency(G); % 1/2 матриці суміжності вершин орграфа
B = B1-B1'; % матриця суміжності вершин орграфа
imagesc(B) % поверхня
colormap(gray(256)) % сіра палітра
axis("equal") % однаковий масштаб уздовж осей координат
xlim([0.5 numnodes(G)+0.5]) % границі вздовж Ox
ylim([0.5 numnodes(G)+0.5]) % границі вздовж Oy
title("Матриця суміжності вершин орграфа")
print("AdjVerMatrDiGraph","-dpng") % зберегли рисунок у файл
```

Рис. 1.18. Матриця суміжності вершин орграфа

Як і в попередньому прикладі 1.10, чорні клітинки — це -1, білі — 1, а сірі — 0. □

Далі, в наступних главах, будуть застосовуватися ще деякі матриці. А зараз розглянемо, як змінюються матриці інцидентності, суміжності вершин та суміжності ребер при перенумерації вершин та (або) ребер чи

дуг.

Почнемо з матриці інцидентності $\boldsymbol{A}$. При перенумерації ребер (дуг) треба переставити її стовпці, а при перенумерації вершин — рядки. З лінійної алгебри відомо, що для перестановки стовпців треба матрицю $\boldsymbol{A}$ помножити на матрицю перестановок $\boldsymbol{T}_E$ розміром $m \times m$. У кожному рядку матриці $\boldsymbol{T}_E$ повинна бути одна одиниця (на тому місці, куди треба переставити цей стовпчик), а решта — нулі. Так само, якщо треба переставити рядки матриці $\boldsymbol{A}$, створюємо матрицю перестановок $\boldsymbol{T}_V$ розміром $n \times n$ за тими самими правилами, транспонуємо її та множимо $\boldsymbol{T}_V^T$ на $\boldsymbol{A}$. Нова матриця інцидентності $\boldsymbol{A}_1$ буде обчислюватися так:

$$\boldsymbol{A}_1 = \boldsymbol{T}_V^T \boldsymbol{A} \boldsymbol{T}_E. \tag{1.19}$$

Матриця суміжності вершин $\boldsymbol{B}$ змінюється лише при перенумерації вершин: переставляються і рядки, і стовпці. Перенумерація ребер її не змінює. Формула для обчислення нової матриці суміжності вершин $\boldsymbol{B}_1$ буде такою:

$$\boldsymbol{B}_1 = \boldsymbol{T}_V^T \boldsymbol{B} \boldsymbol{T}_V. \tag{1.20}$$

І, нарешті, матриця суміжності ребер $\boldsymbol{C}$ змінюється лише при перенумерації ребер: переставляються і рядки, і стовпці. Перенумерація вершин її не змінює. Формула для зміни така ж сама:

$$\boldsymbol{C}_1 = \boldsymbol{T}_E^T \boldsymbol{C} \boldsymbol{T}_E. \tag{1.21}$$

1.4. Запитання для перевірки

1. Що таке (простий) граф?
2. Що таке вершини графа? ребра?
3. Що таке розмір графа? його потужність?
4. Які вершини та ребра називаються зваженими? розфарбованими?
5. Що таке мультиграф?
6. Що таке псевдограф?
7. Що таке гіперграф?
8. Що таке орграф?
9. Як задати граф, мультиграф, псевдограф, орграф у MATLAB?
10. Як нарисувати граф, мультиграф, псевдограф, орграф у MATLAB?
11. Що таке кліка?
12. Який граф називається дводолевим?
13. Який гіперграф називається двоїстим до заданого?
14. Як будується матриця інцидентності графа?
15. Як відновлюється список ребер графа за його матрицею інцидентності?

16. Як побудувати матрицю інцидентності графа, мільтиграфа, орграфа в MATLAB?
17. Як будується матриця суміжності вершин графа?
18. Як відновлюється список ребер графа за його матрицею суміжності вершин?
19. Як побудувати матрицю суміжності вершин графа, мільтиграфа, орграфа в MATLAB?
20. Як пов'язані між собою матриці інцидентності та суміжності вершин?
21. Як будується матриця суміжності ребер графа?
22. Як пов'язані між собою матриці інцидентності та суміжності ребер?
23. Як побудувати матрицю суміжності ребер графа, мільтиграфа в MATLAB?
24. Як будується матриця інцидентності орграфа?
25. Як відновлюється список дуг орграфа за його матрицею інцидентності?
26. Як будується матриця суміжності вершин орграфа?
27. Як відновлюється список дуг орграфа за його матрицею суміжності вершин?
28. Як змінюється матриця інцидентності графа чи орграфа при перенумерації його вершин? ребер (дуг)?
29. Як змінюється матриця суміжності вершин графа чи орграфа при перенумерації його вершин? ребер (дуг)?
30. Як змінюється матриця суміжності ребер графа при перенумерації його вершин? ребер?

2. Пакування, покриття, домінуючі множини, кліки

У цьому розділі будуть розглянуті сім задач: вершинні та реберні пакування, покриття та домінуючі множини (це $2 \times 3 = 6$ задач) і задача про максимальний повний підграф (максимальну кліку). Усі ці задачі формулюються як задачі бінарного лінійного програмування (БЛП).

2.1. Реберне пакування

Означення 2.1. *Реберним пакуванням* (edge packing) у графі $G = (V, E)$ називається підмножина попарно несуміжних ребер $E_1 \subset E$. Інша назва реберного пакування: *паросполучення* (matching). Паросполучення називається *максимальним за включенням*, якщо воно не є підмножиною паросполучення з більшою кількістю ребер. Паросполучення називається *максимальним*, якщо воно містить у собі максимально можливу кількість ребер. □

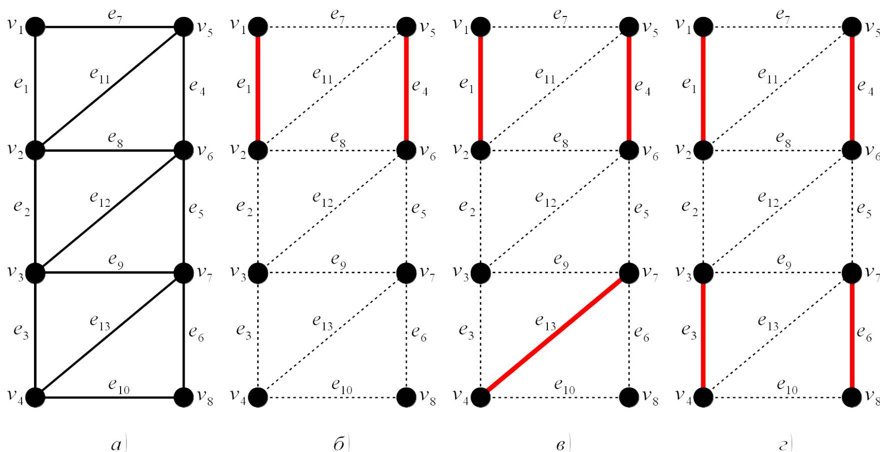


Рис. 2.1. Граф (а), його паросполучення (б), максимальне за включенням паросполучення (в) та максимальне паросполучення (г)

На рис. 2.1, *а* представлений граф з $n = 8$ вершинами та $m = 13$ ребрами. Одне з його паросполучень — це множина ребер $\{e_1, e_4\}$, що не є суміжними. Воно виділене на рис. 2.1, *б*. Це паросполучення не є максимальним за включенням: до нього можна долучити ребро e_{13}, як показано на рис. 2.1, *в*. До множини ребер $\{e_1, e_4, e_{13}\}$ вже нічого додати не можна: будь-яке інше ребро є суміжним з якимось із цих ребер. Тому паросполучення $\{e_1, e_4, e_{13}\}$ є максимальним за включенням. Але воно не є макси-

мальним: замість одного ребра e_{13} до множини $\{e_1, e_4\}$ можна долучити два ребра: e_3 та e_6 (рис. 2.1, *г*). Паросполучення $\{e_1, e_4, e_3, e_6\}$ вже буде максимальним: для 8 вершин більше ніж 4 несуміжних ребра вибрати неможливо.

Означення 2.2. Паросполучення називається *довершеним* (perfect matching), якщо його ребра є інцидентними до всіх вершин графа. □

Довершене паросполучення графа є водночас і його реберним покриттям (див. далі підрозділ 2.3). Його можна побудувати (якщо взагалі можна) лише для графа з парною кількістю вершин, і кількість ребер у ньому дорівнює $\frac{n}{2}$. Тому можна сказати й так: якщо у графі з парною кількістю вершин знайдене паросполучення з $\frac{n}{2}$ ребрами, то це паросполучення буде довершеним.

Однією з задач теорії графів є знаходження в ньому максимального паросполучення. Якщо ребра графа зважені, то узагальненням буде задача про максимальне зважене паросполучення. В ній треба знайти паросполучення з максимальною загальною вагою ребер.

Прикладом такої задачі є розбиття колективу людей на пари: екіпажі, бригади тощо. Якщо кожне ребро означає можливість спільної роботи, то маємо незважену задачу: створити максимально можливу кількість працездатних бригад. Якщо ж ребра зважені, то зазвичай вага ребра означає продуктивність цієї бригади. В цьому випадку формування колективу з максимальною загальною продуктивністю є задачею про максимальне зважене паросполучення.

Для розв'язання задачі про максимальне (зважене) паросполучення є багато методів. Але, якщо ми маємо ефективну процедуру розв'язання задачі БЛП, то найпростіший спосіб розв'язання задачі про максимальне (зважене) паросполучення — це зведення її до задачі БЛП. Розглянемо, як це зробити.

Введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Назвемо цей вектор *асоційованим* з ребрами графа E, оскільки кожна координата e_k вектора $\boldsymbol{e}$ буде характеризувати відповідне ребро. Якщо ребро e_k входить у шукане паросполучення, то асоційована з ним змінна e_k буде приймати значення 1, а якщо ні — то 0. Тоді загальну кількість ребер, що входять у паросполучення, можна записати у вигляді:

$$z = \sum_{k=1}^{m} e_k = (\mathbf{1}, \boldsymbol{e}), \tag{2.1}$$

де $\mathbf{1}$ — вектор-стовпчик з одиниць відповідної розмірності. В задачі про максимальне паросполучення цю величину треба максимізувати за умови, що всі змінні e_k можуть приймати значення тільки 0 або 1:

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1,m}; \end{cases} \tag{2.2}$$

і серед ребер немає суміжних. Остання вимога означає, що для кожної вершини існує не більше одного ребра, інцидентного до неї. Перейдемо від ребер e_k до асоційованих з ними змінних e_k. Маємо: для кожної вершини v_i сума змінних e_k, асоційованих з ребрами, інцидентними до v_i, не повинна перевищувати одиниці.

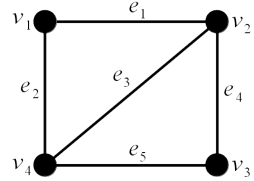


Рис. 2.2. Приклад графа

Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. До вершини v_1 є інцидентними ребра e_1 та e_2; тому сума змінних e_1 та e_2 не повинна перевищувати одиниці. До вершини v_2 є інцидентними ребра e_1, e_3 та e_4; тому сума змінних e_1, e_3 та e_4 також не повинна перевищувати одиниці тощо:

$$\begin{array}{ll} v_1: & e_1 + e_2 \leq 1; \\ v_2: & e_1 + e_3 + e_4 \leq 1; \\ v_3: & e_4 + e_5 \leq 1; \\ v_4: & e_2 + e_3 + e_5 \leq 1. \end{array} \tag{2.3}$$

Якщо скористатися матрицею інцидентності, яка для графа з рис. 2.2 має вигляд:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \tag{2.4}$$

то умова відсутності суміжних ребер записується так:

$$\boldsymbol{Ae} \leq \boldsymbol{1}. \tag{2.5}$$

Тут векторна нерівність означає одночасне виконання нерівностей за всіма координатами. Отже, маємо задачу БЛП:

$$\begin{cases} z = (\boldsymbol{1}, \boldsymbol{e}) \to \max; \\ \boldsymbol{Ae} \leq \boldsymbol{1}; \\ e_k = 0 \vee 1; \ k = \overline{1,m}. \end{cases} \tag{2.6}$$

В задачі про максимальне зважене паросполучення треба максимізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами ребер як $\boldsymbol{c}$. Тоді замість (2.6) будемо мати задачу БЛП, що відрізняється від (2.6) тільки цільовою функцією:

$$
\begin{cases}
z = (\boldsymbol{c}, \boldsymbol{e}) \to \max; \\
\boldsymbol{A}\boldsymbol{e} \le \boldsymbol{1}; \\
e_k = 0 \vee 1;\ k = \overline{1, m}.
\end{cases}
\tag{2.7}
$$

Алгоритм (2.6-2.7) реалізований в методі `maxmatch`, що міститься в Graph Theory Toolbox. Для графа `G` він повертає список ребер, включених до максимального паросполучення (вектор-рядок). Якщо граф зі зваженими ребрами, то розв'язується задача (2.7), а якщо ні — то (2.6).

Приклад 2.1. Для графа з прикладу 1.4 (рис. 1.10) знайти максимальні незважене та зважене паросполучення.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 5]; % ваги
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
mm = maxmatch(G); % розв'язуємо незважену задачу
MaxSize = length(mm); % кількість ребер
EdgeWidth = ones(numedges(G),1); % тонкі лінії ребер
EdgeWidth(mm) = 5; % товсті лінії ребер паросполучення
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"LineWidth",EdgeWidth) % рисуємо
title("Кількість ребер у максимальному паросполученні " + ...
  MaxSize)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MaxMatch","-dpng") % зберегли рисунок у файл
G = graph(s,t,w); % зважений граф
mm = maxmatch(G); % розв'язуємо зважену задачу
MaxWeight = sum(G.Edges.Weight(mm)); % максимальна вага
EdgeWidth = ones(numedges(G),1); % тонкі лінії ребер
EdgeWidth(mm) = 5; % товсті лінії ребер паросполучення
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"EdgeLabel",G.Edges.Weight, ...
  "LineWidth",EdgeWidth) % нарисували
title("Вага максимального зваженого паросполучення " + ...
  MaxWeight)
axis("equal") % однаковий масштаб уздовж осей координат
```

```
axis("off") % прибрали осі
print("WeightedMaxMatch","-dpng") % зберегли рисунок у файл
```

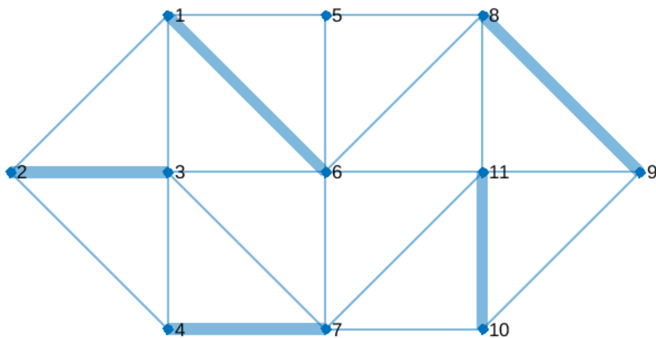


Рис. 2.3. Максимальне паросполучення

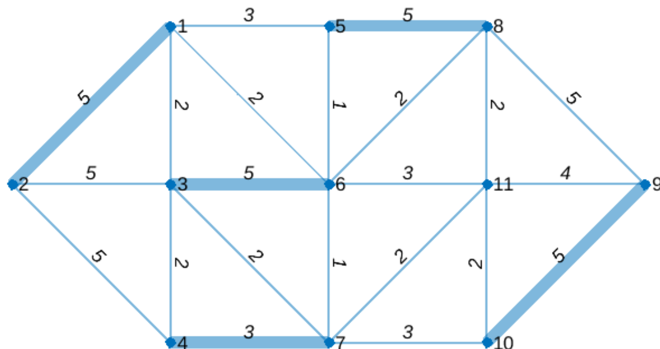


Рис. 2.4. Максимальне зважене паросполучення

На рис. 2.3 показаний розв'язок незваженої задачі, а на рис. 2.4 — зваженої. Мітки ребер у зваженій задачі — ваги ребер. □

Інший приклад: максимальне зважене паросполучення на дводольному графі є задачею про призначення.

Приклад 2.2. У табл. 2.1 показана матриця продуктивностей для задачі про призначення п'яти працівників на шість робіт, не більше одного працівника на кожну роботу та не більше однієї роботи на кожного

Табл. 2.1. Матриця продуктивностей

$i\backslash j$	1	2	3	4	5	6
1	4	6	5	2	8	13
2	14	11	6	5	11	14
3	14	2	8	2	4	14
4	3	2	9	5	6	12
5	13	11	7	10	4	10

працівника. Треба так розподілити працівників на роботи, щоб отримати максимальний прибуток.

У цій задачі нам треба знайти максимальне зважене паросполучення на повному дводолевому графі $K_{5,6}$. Ваги всіх $5 \times 6 = 30$ ребер задані в табл. 2.1.

```
% вхідні дані - матриця продуктивностей
C = [ 4  6  5  2  8 13; ...
     14 11  6  5 11 14; ...
     14  2  8  2  4 14; ...
      3  2  9  5  6 12; ...
     13 11  7 10  4 10];
% кінець вхідних даних

% друк вхідних даних
disp("Задача про призначення")
[m,n] = size(C); % кількість працівників і робіт
fprintf("Кількість працівників = %d\n",m)
fprintf("Кількість робіт = %d\n",n)
disp("Матриця продуктивностей:")
fprintf("i\\j ") % початок 1-го рядка
fprintf("%3.0f  ",1:n) % 1-й рядок
fprintf("\n") % кінець 1-го рядка
for i=1:m % наступні рядки
  fprintf("%2.0f",i) % початок рядка
  fprintf("%5.0f",C(i,:)) % рядок
  fprintf("\n") % кінець рядка
end
disp("================================")
```

```
% формуємо вхідні дані для графа
s = reshape(repmat(1:m,n,1),m*n,1); % працівники
t = repmat((1:n)',m,1)+m; % роботи
w = reshape(C',m*n,1); % ваги ребер

G = graph(s,t,w); % граф зі зваженими ребрами
N = numnodes(G); % кількість вершин; N=m+n
M = numedges(G); % кількість ребер; M=m*n
mm = maxmatch(G); % max зважене паросполучення
MaxWeight = sum(G.Edges.Weight(mm)); % max продуктивність

% друкуємо результати розв'язання задачі
disp("Розподіл працівників:")
disp("Працівник  Робота  Продуктивність")
fprintf("  %3.0f  ---> %3.0f       %5.0f\n",...
  (G.Edges{mm,:}-repmat([0 m 0],length(mm),1))')
disp("====================================")
fprintf("MAX продуктивність  = %5.0f\n",MaxWeight)

% дані для рисування
x = [-ones(m,1); ones(n,1)]*2; % координати вершин
y = [(2:-4/(m-1):-2), 2:-4/(n-1):-2]';
% працівники ліворуч, роботи праворуч
EdgeWidth = ones(numedges(G),1)/2; % тонкі лінії ребер
EdgeWidth(mm) = 4; % товсті лінії ребер паросполучення
NodeLabel = cell(1,N); % для міток вершин
for k=1:m % номер працівника
  NodeLabel{k} = ['\itv\rm_{' num2str(k) '}'];
end
for k=1:n % номер роботи
  NodeLabel{m+k} = ['\itw\rm_{' num2str(k) '}'];
end
EdgeLabel = cell(1,M); % для міток ребер
for k=1:M % усі мітки пусті
  EdgeLabel{k} = '';
end
for k=1:length(mm) % мітки ребер паросполучення - ваги
  EdgeLabel{mm(k)} = [num2str(G.Edges.Weight(mm(k))) ...
    '                                                    ' ...
    '                                                    '];
```

```
    % додали в кінці пробіли, щоб змістити мітку ребра ліворуч
end

% рисуємо
figure("Position",[100 100 600 600]) % нове вікно фігури
plot(G,... % рисуємо граф
  "XData",x,"YData",y,... % координати вершин
  "LineWidth",EdgeWidth,... % товщини ліній ребер
  "NodeLabel",NodeLabel,... % мітки вершин
  "EdgeLabel",EdgeLabel,... % мітки ребер
  "MarkerSize",6) % розмір міток вершин
title("MAX продуктивність = " + MaxWeight) % заголовок
axis("equal") % однаковий масштаб вдовж осей координат
axis("off") % прибрали оси
print("MaxAssign","-dpng") % зберегли рисунок у файл
```

```
Задача про призначення
Кількість працівників = 5
Кількість робіт = 6
Матриця продуктивностей:
i\j   1    2    3    4    5    6
 1    4    6    5    2    8   13
 2   14   11    6    5   11   14
 3   14    2    8    2    4   14
 4    3    2    9    5    6   12
 5   13   11    7   10    4   10
================================
Розподіл працівників:
Працівник  Робота  Продуктивність
    1   --->   6          13
    2   --->   5          11
    3   --->   1          14
    4   --->   3           9
    5   --->   2          11
===================================
MAX продуктивність  =    58
```

На рис. 2.5 показаний дводолевий граф — розв'язок задачі про призначення. Ліва доля — працівники, права — роботи, товсті лінії — призначення. □

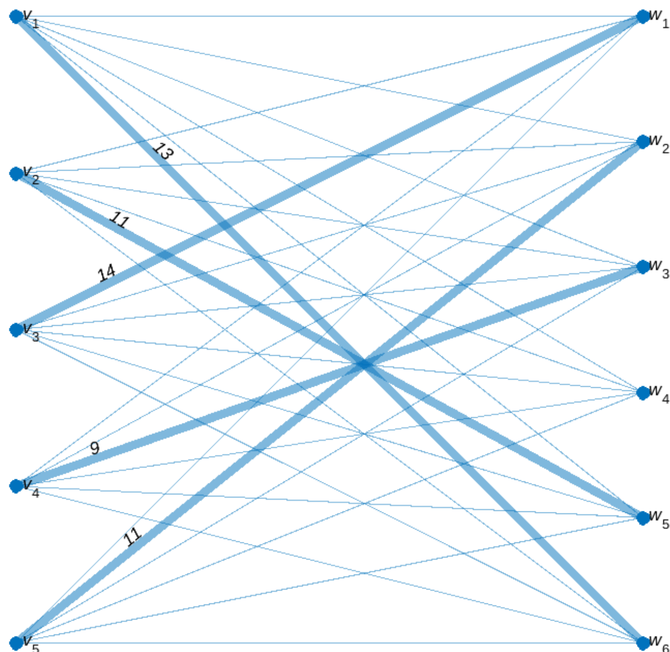


Рис. 2.5. Розв'язок задачі про призначення

2.2. Вершинне пакування

Означення 2.3. *Вершинним пакуванням* (vertex packing) у графі $G = (V, E)$ називається підмножина попарно несуміжних вершин $V_1 \subset V$. Інші назви вершинного пакування: *незалежна множина вершин* (independent set), *внутрішньо стійка множина вершин* (stable set). Незалежна множина вершин називається *максимальною за включенням*, якщо вона не є підмножиною незалежної множини вершин з більшою кількістю вершин. Незалежна множина вершин називається *максимальною*, якщо вона містить у собі максимально можливу кількість вершин. □

На рис. 2.6, *а* представлений граф з $n = 8$ вершинами та $m = 13$ ребрами, той самий, що й на рис. 2.1, *а*. Одна його вершина v_1 утворює незалежну множину вершин (рис. 2.6, *б*). Ця незалежна множина не є максимальною за включенням: до неї можна долучити вершину v_7, як показано на рис. 2.6, *в*. До множини вершин $\{v_1, v_7\}$ вже нічого додати не можна: будь-яка інша вершина є суміжною або з v_1, або з v_7. Тому незалежна множина вершин $\{v_1, v_7\}$ є максимальною за включенням.

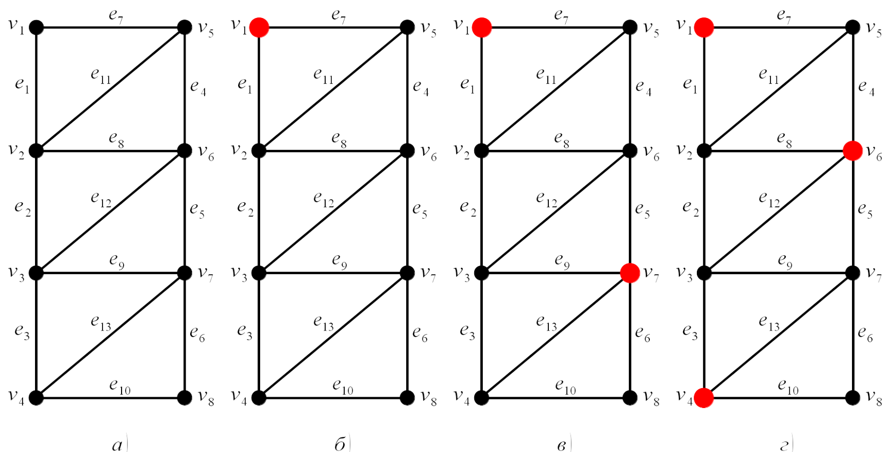


Рис. 2.6. Граф (а), його незалежна множина вершин (б), максимальна за включенням незалежна множина вершин (в) та максимальна незалежна множина вершин (г)

Але вона не є максимальною: замість однієї вершини v_7 до множини $\{v_1\}$ можна долучити дві вершини: v_6 та v_4 (рис. 2.6, *г*). Незалежна множина вершин $\{v_1, v_4, v_6\}$ вже буде максимальною: знайти чотири попарно несуміжні вершини в цьому графі нам не вдасться.

Однією з задач теорії графів є знаходження в ньому максимальної незалежної множини вершин. Якщо вершини графа зважені, то узагальненням буде задача про максимальну зважену незалежну множину вершин. У ній треба знайти множину вершин з максимальною загальною вагою.

Приклад: є колектив експертів (вершини графа); вага вершини — це рівень компетентності експерта. Ребра графа — зв'язки між експертами. Треба обрати з цього колективу підгрупу незалежних одне від одного експертів із максимальним загальним рівнем компетентності.

Зведемо задачу про максимальну незалежну множину вершин до задачі БЛП. Для цього введемо до розгляду вектор-стовпчик $\boldsymbol{v}$ довжиною n. Цей вектор буде асоційованим з вершинами: якщо якась вершина v_i міститься у незалежній множині, то відповідна змінна v_i буде приймати значення 1, а якщо ні — то 0. Тобто координати вектора $\boldsymbol{v}$ є цілочисельними та можуть приймати лише одне з двох значень — 0 або 1:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1, n}. \end{cases} \tag{2.8}$$

Треба максимізувати загальну кількість вершин:

$$z = \sum_{i=1}^{n} v_i = (\mathbf{1}, \boldsymbol{v}) \tag{2.9}$$

за умови, що серед них немає суміжних. Ця вимога означає, що для кожного ребра існує не більше однієї вершини з незалежної множини, інцидентної до нього. Якщо перейти від вершин v_i до асоційованих з ними змінних v_i, то маємо: для кожного ребра e_k сума змінних v_i, асоційованих з вершинами, інцидентними до e_k, не повинна перевищувати одиниці. Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. Ребро e_1 є інцидентним до вершин v_1 і v_2; тому сума змінних v_1 і v_2 не може перевищувати одиниці. Ребро e_2 є інцидентним до вершин v_1 і v_4; тому сума змінних v_1 і v_4 також не повинна перевищувати одиниці тощо:

$$\begin{array}{ll} e_1: & v_1 + v_2 \le 1; \\ e_2: & v_1 + v_4 \le 1; \\ e_3: & v_2 + v_4 \le 1; \\ e_4: & v_2 + v_3 \le 1; \\ e_5: & v_3 + v_4 \le 1. \end{array} \tag{2.10}$$

Якщо скористатися матрицею інцидентності (2.4), то умова відсутності суміжних вершин записується так:

$$\boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}. \tag{2.11}$$

Отже, маємо задачу БЛП:

$$\begin{cases} z = (\mathbf{1}, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}. \end{cases} \tag{2.12}$$

В задачі про максимальну зважену незалежну множину вершин треба максимізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами вершин $\boldsymbol{b}$. Тоді замість (2.12) будемо мати задачу БЛП, що відрізняється від (2.12) тільки цільовою функцією:

$$\begin{cases} z = (\boldsymbol{b}, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}. \end{cases} \tag{2.13}$$

У Graph Theory Toolbox є метод `maxindset`, що реалізує алгоритм (2.12-2.13). Для графа `G` він повертає список вершин, включених до максимальної незалежної множини (вектор-рядок). Якщо вершини графа зважені, то розв'язується задача (2.13), а якщо ні — то (2.12).

Приклад 2.3. Для графа зі зваженими вершинами з прикладу 1.5 (рис. 1.11 — там орграф, тому треба замінити дуги ребрами) знайти максимальні незважену та зважену незалежні множини вершин.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
d = [2 3 3 4 1 2 3 3 5 1 5]; % ваги вершин
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
ms = maxindset(G); % розв'язуємо незважену задачу
MaxSize = length(ms); % розмір незалежної множини вершин
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин з незалежної множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"MarkerSize",NodeSize) % рисуємо
title("Кількість вершин у максимальній незалежній " + ...
  "множині = " + MaxSize) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MaxIndSet","-dpng") % зберегли рисунок у файл
G = graph(s,t); % незважений граф
G.Nodes.Weight = d'; % додали ваги вершин
ms = maxindset(G); % розв'язуємо зважену задачу
MaxWeight = sum(G.Nodes.Weight(ms)); % вага
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин з незалежної множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",G.Nodes.Weight, ...
  "MarkerSize",NodeSize) % нарисували
title("Загальна вага вершин у максимальній незалежній " + ...
  "множині = " + MaxWeight) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("WeightedMaxIndSet","-dpng") % зберегли рисунок у файл
```

На рис. 2.7 показаний розв'язок незваженої задачі, а на рис. 2.8 — зваженої. У незваженій задачі мітки вершин — це їхні номери, а у зваженій — ваги. □

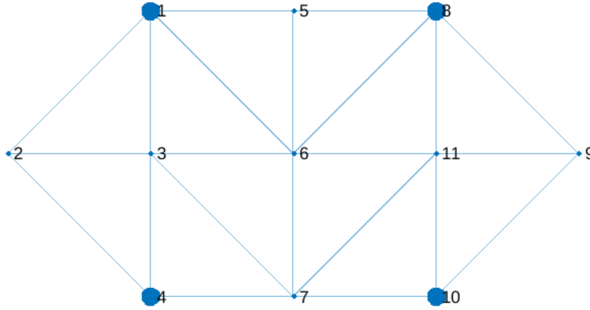


Рис. 2.7. Максимальна незалежна множина вершин

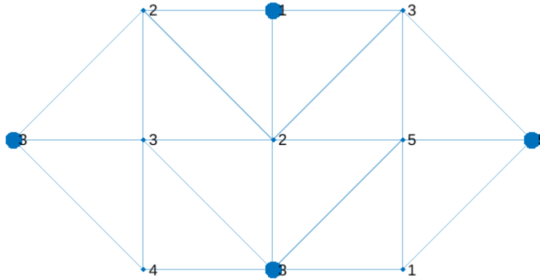


Рис. 2.8. Максимальна зважена незалежна множина вершин

2.3. Реберне покриття

Означення 2.4. *Реберним покриттям* графа $G = (V, E)$ (line-covering, edge covering) називається підмножина ребер $E_1 \subset E$, що є інцидентними до всіх вершин графа. Реберне покриття називається *мінімальним за включенням*, якщо будь-яка його підмножина з меншою кількістю ребер не є реберним покриттям. Реберне покриття називається *мінімальним*, якщо воно містить у собі мінімально можливу кількість ребер. □

На рис. 2.9, *а* показаний граф з $n = 8$ вершинами та $m = 13$ ребрами (той самий приклад, що й раніше). Одне з його реберних покриттів — це $\{e_5, e_7, e_8, e_{10}, e_{11}, e_{12}, e_{13}\}$ з рис. 2.9, *б*. Його ребра є інцидентними до всіх вершин. Це покриття не є мінімальним за включенням: з нього можна вилучити, наприклад, ребра e_5 та e_8 (рис. 2.9, *в*). Ті ребра, що залишилися: $\{e_7, e_{10}, e_{11}, e_{12}, e_{13}\}$, також є інцидентними до всіх вершин. Отже, це реберне покриття. Якщо вилучити з покриття $\{e_7, e_{10}, e_{11}, e_{12}, e_{13}\}$ ще будь-яке ребро, якась з вершин стане ізольованою. Отже, покриття

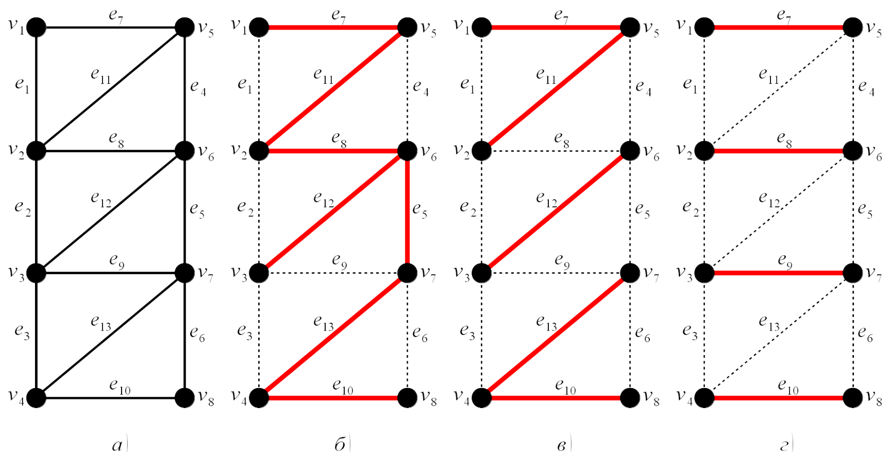


Рис. 2.9. Граф (а), його реберне покриття (б), мінімальне за включенням реберне покриття (в) та мінімальне реберне покриття (г)

$\{e_7, e_{10}, e_{11}, e_{12}, e_{13}\}$ є мінімальним за включенням. Але це не мінімальне покриття: можна створити покриття не з п'яти, а з чотирьох ребер, які будуть інцидентними до всіх вершин. Одне з таких мінімальних реберних покриттів $\{e_7, e_8, e_9, e_{10}\}$ представлене на рис. 2.9, *г*. Для цього прикладу знайдене мінімальне реберне покриття є водночас довершеним паросполученням (див. підрозділ 2.1).

Однією з класичних задач теорії графів є знаходження мінімального реберного покриття. Якщо ребра зважені, можна ставити задачу про мінімальне зважене реберне покриття: знайти реберне покриття не з найменшою кількістю ребер, а з найменшою загальною вагою. Приклад з військової справи. Є опорні пункти супротивника (вершини графа), пов'язані між собою комунікаціями (ребра). Якщо знищення комунікації знищує також обидва опорні пункти, інцидентні до неї, то маємо задачу: як знищити всю оборону супротивника, завдавши найменшу кількість ударів по комунікаціях? Інколи знищувати одні комунікації легше (вага відповідного ребра менша), а інші важче (більша вага). Тоді маємо зважену задачу: найменшою ціною знищити всю оборону супротивника.

Розглянемо формулювання задачі про мінімальне (зважене) реберне покриття як задачі БЛП. Як і в задачі про паросполучення (див. підрозділ 2.1), введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Його координати є асоційованими з ребрами графа E. Якщо ребро e_k входить у шукане реберне покриття, то асоційована з ним змінна e_k буде приймати значення 1, а якщо ні — то 0. Тоді загальну кількість ребер, що входять у реберне покриття, можна записати у вигляді:

$$t = \sum_{k=1}^{m} e_k = (\mathbf{1}, \boldsymbol{e}) . \tag{2.14}$$

В задачі про мінімальне реберне покриття цю величину треба мінімізувати за умови, що всі змінні e_k можуть приймати значення тільки 0 або 1:

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1, m}; \end{cases} \tag{2.15}$$

і ребра покривають усі вершини (тобто є інцидентними до всіх вершин). Ця вимога означає, що для кожної вершини існує принаймні одне ребро з шуканого покриття, інцидентне до неї. Тобто: для кожної вершини v_i сума змінних e_k, асоційованих з ребрами, інцидентними до v_i, не повинна бути меншою за одиницю. Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. До вершини v_1 є інцидентними ребра e_1 та e_2; тому сума змінних e_1 та e_2 не повинна бути меншою за одиницю. До вершини v_2 інцидентними є ребра e_1, e_3 та e_4; тому сума змінних e_1, e_3 та e_4 також не повинна бути меншою за одиницю тощо. Як бачимо, нерівності-обмеження в цій задачі відрізняються від нерівностей-обмежень задачі про паросполучення (2.3) лише знаком:

$$\begin{array}{ll} v_1: & e_1 + e_2 \geq 1; \\ v_2: & e_1 + e_3 + e_4 \geq 1; \\ v_3: & e_4 + e_5 \geq 1; \\ v_4: & e_2 + e_3 + e_5 \geq 1. \end{array} \tag{2.16}$$

З використанням матриці інцидентності (2.4) система (2.16) записується у вигляді:

$$\boldsymbol{Ae} \geq \mathbf{1}. \tag{2.17}$$

Отже, маємо задачу БЛП:

$$\begin{cases} t = (\mathbf{1}, \boldsymbol{e}) \rightarrow \min; \\ \boldsymbol{Ae} \geq \mathbf{1}; \\ e_k = 0 \vee 1; \; k = \overline{1, m}. \end{cases} \tag{2.18}$$

В задачі про мінімальне зважене реберне покриття треба мінімізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами ребер $\boldsymbol{c}$. Тоді замість (2.18) будемо мати задачу БЛП, що відрізняється від (2.18) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{c}, \boldsymbol{e}) \rightarrow \min; \\ \boldsymbol{Ae} \geq \mathbf{1}; \\ e_k = 0 \vee 1; \; k = \overline{1, m}. \end{cases} \tag{2.19}$$

Цей алгоритм реалізований у Graph Theory Toolbox у вигляді метода `minedgecover`. Для графа `G` він повертає вектор-рядок з номерами ребер, включених у мінімальне реберне покриття. Якщо ребра графа зважені, розв'язується задача (2.19), а якщо ні — то (2.18).

Приклад 2.4. Для графа з прикладу 2.1 знайти мінімальні незважене та зважене реберні покриття.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 5]; % ваги
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
mc = minedgecover(G); % розв'язуємо незважену задачу
MinSize = length(mc); % мінімальний розмір
EdgeWidth = ones(numedges(G),1); % товщини ліній ребер
EdgeWidth(mc) = 5; % товщини ліній ребер покриття
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"LineWidth",EdgeWidth) % рисуємо
title("Кількість ребер у мінімальному покритті = " + MinSize)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinEdgeCover","-dpng") % зберегли рисунок у файл
G = graph(s,t,w); % зважений граф
mc = minedgecover(G); % розв'язуємо зважену задачу
MinWeight = sum(G.Edges.Weight(mc)); % мінімальна вага
EdgeWidth = ones(numedges(G),1); % товщини ліній ребер
EdgeWidth(mc) = 5; % товщини ліній ребер покриття
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"EdgeLabel",G.Edges.Weight, ...
  "LineWidth",EdgeWidth) % нарисували
title("Вага мінімального реберного покриття = " + MinWeight)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("WeightedMinEdgeCover","-dpng") % зберегли рисунок
```

На рис. 2.10 показаний розв'язок незваженої задачі, а на рис. 2.11 — зваженої. Мітки ребер у зваженій задачі — ваги ребер. □

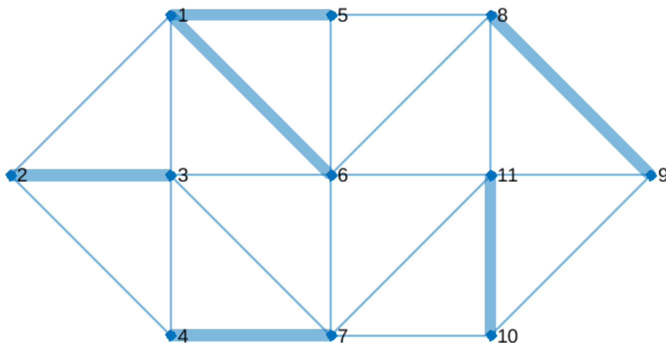


Рис. 2.10. Мінімальне реберне покриття

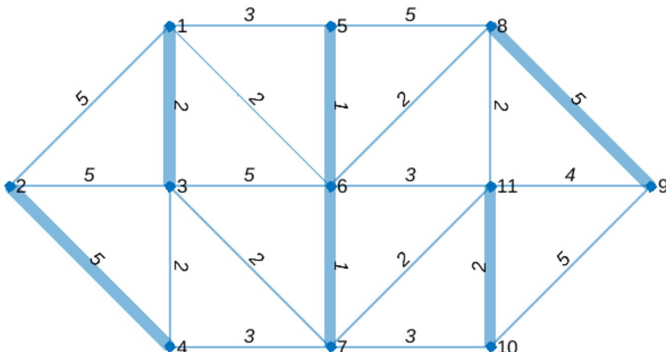


Рис. 2.11. Мінімальне зважене реберне покриття

2.4. Вершинне покриття

Означення 2.5. *Вершинним покриттям*, або (інша назва) *трансверсальною множиною вершин* графа $G = (V, E)$ (vertex covering, transversal set) називається підмножина вершин $V_1 \subset V$, що є інцидентними до всіх ребер. Вершинне покриття називається *мінімальним за включенням*, якщо будь-яка його підмножина з меншою кількістю вершин не є вершинним покриттям. Вершинне покриття називається *мінімальним*, якщо воно містить у собі мінімально можливу кількість вершин. □

Якщо з множини V вершин графа, показаного на рис. 2.12, *а*, вилучити вершину v_4, то ті вершини, що залишилися $\{v_1, v_2, v_3, v_5, v_6, v_7, v_8\}$, утворюють вершинне покриття (рис. 2.12, *б*): вони покривають усі реб-

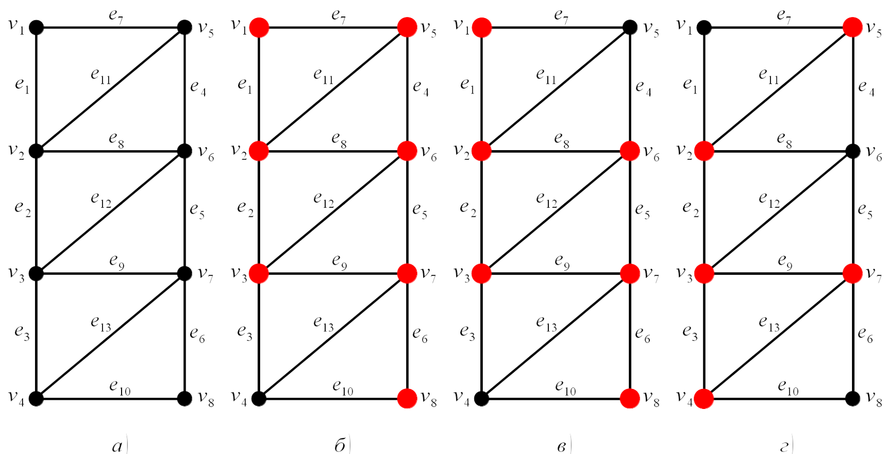


Рис. 2.12. Граф (а), його вершинне покриття (б), мінімальне за включенням вершинне покриття (в) та мінімальне вершинне покриття (г)

ра. Це покриття не є мінімальним за включенням: з нього можна вилучити v_5, і воно залишиться покриттям (рис. 2.12, *в*), оскільки вершини $\{v_1, v_2, v_3, v_6, v_7, v_8\}$ також покривають усі ребра. Покриття $\{v_1, v_2, v_3, v_6, v_7, v_8\}$ вже буде мінімальним за включенням: якщо з нього вилучити ще хоч одну вершину, то якесь ребро не буде покриватися. Але це покриття не є мінімальним: можна створити покриття не з шести, а з п'яти вершин $\{v_2, v_3, v_4, v_5, v_7\}$, що є інцидентними до всіх ребер (рис. 2.12, *г*). Покриття $\{v_2, v_3, v_4, v_5, v_7\}$ буде одним з мінімальних вершинних покриттів для цього графа: для нього не існує множини з чотирьох вершин, які були б інцидентними до всіх ребер.

В задачі про мінімальне вершинне покриття треба його знайти. Якщо вершини зважені, можна ставити задачу про мінімальне зважене вершинне покриття: знаходження множини вершин мінімальної загальної ваги, що є інцидентними до всіх ребер.

Приклад — військова задача. Є опорні пункти супротивника (вершини графа) та комунікації між ними (ребра). Знищення опорного пункту знищує й усі комунікації, що від нього тягнуться. Треба знищити мінімальну кількість опорних пунктів, перервавши всі комунікації. Для зваженої задачі додатково задається вартість знищення кожного опорного пункту. Треба знищити всі комунікації за мінімальну ціну.

Сформулюємо задачу про мінімальне (зважене) вершинне покриття як задачу БЛП. Для цього введемо до розгляду вектор-стовпчик $\boldsymbol{v}$ довжиною n. Його координати асоційовані з вершинами: якщо якась вершина v_i входить до вершинного покриття, то відповідна змінна v_i буде приймати

значення 1, а якщо ні — то 0. Тобто координати вектора $\boldsymbol{v}$ є цілочисельними та можуть приймати лише одне з двох можливих значень — 0 або 1:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1, n}. \end{cases} \tag{2.20}$$

Треба мінімізувати загальну кількість вершин:

$$t = \sum_{i=1}^{n} v_i = (\mathbf{1}, \boldsymbol{v}) \tag{2.21}$$

за умови, що ці вершини покривають усі ребра. Ця вимога означає, що для кожного ребра існує не менше однієї вершини з цього покриття, інцидентної до нього. Якщо перейти від вершин v_i до асоційованих з ними змінних v_i, то маємо: для кожного ребра e_k сума змінних v_i, асоційованих з вершинами, інцидентними до e_k, не повинна бути меншою за одиницю. Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. Ребро e_1 є інцидентним до вершин v_1 і v_2; тому сума змінних v_1 і v_2 не може бути меншою за одиницю. Ребро e_2 є інцидентним до вершин v_1 і v_4; тому сума змінних v_1 і v_4 також не повинна бути меншою за одиницю тощо. Нерівності-обмеження відрізняються від відповідних обмежень задачі про вершинне пакування (2.10) лише знаком:

$$\begin{aligned} e_1: &\quad v_1 + v_2 \geq 1; \\ e_2: &\quad v_1 + v_4 \geq 1; \\ e_3: &\quad v_2 + v_4 \geq 1; \\ e_4: &\quad v_2 + v_3 \geq 1; \\ e_5: &\quad v_3 + v_4 \geq 1. \end{aligned} \tag{2.22}$$

Якщо скористатися матрицею інцидентності (2.4), то ці нерівності записуються так:

$$\boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}. \tag{2.23}$$

Отже, маємо задачу БЛП:

$$\begin{cases} t = (\mathbf{1}, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}; \\ v_i = 0 \vee 1; \ i = \overline{1, n}. \end{cases} \tag{2.24}$$

В задачі про мінімальне зважене вершинне покриття треба мінімізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами вершин $\boldsymbol{b}$. Тоді замість (2.24) будемо мати задачу БЛП, що відрізняється від (2.24) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{b}, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}. \end{cases} \tag{2.25}$$

Цю задачу для графа G розв'язує метод minvercover з інструментарію Graph Theory Toolbox. Він повертає вектор-рядок з номерами вершин, включених у мінімальне вершинне покриття. Якщо вершини графа зважені, то розв'язується задача (2.25), а якщо ні — то (2.24).

Приклад 2.5. Для графа зі зваженими вершинами з прикладу 2.3 знайти мінімальні незважене та зважене вершинні покриття.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
d = [2 3 3 4 1 2 3 3 5 1 5]; % ваги вершин
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
mc = minvercover(G); % розв'язуємо незважену задачу
MinSize = length(mc); % розмір min вершинного покриття
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(mc) = 8; % розміри міток вершин покриття
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"MarkerSize",NodeSize) % рисуємо
title("Розмір мінімального вершинного покриття = " + MinSize)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinVerCover","-dpng") % зберегли рисунок у файл
G = graph(s,t); % незважений граф
G.Nodes.Weight = d'; % додали ваги вершин
mc = minvercover(G); % розв'язуємо зважену задачу
MinWeight = sum(G.Nodes.Weight(mc)); % min вага
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(mc) = 8; % розміри міток вершин
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",G.Nodes.Weight, ...
  "MarkerSize",NodeSize) % нарисували
title("Вага мінімального зваженого вершинного " + ...
  "покриття = " + MinWeight) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
```

```
axis("off") % прибрали осі
print("WeightedMinVerCover","-dpng") % зберегли рисунок
```

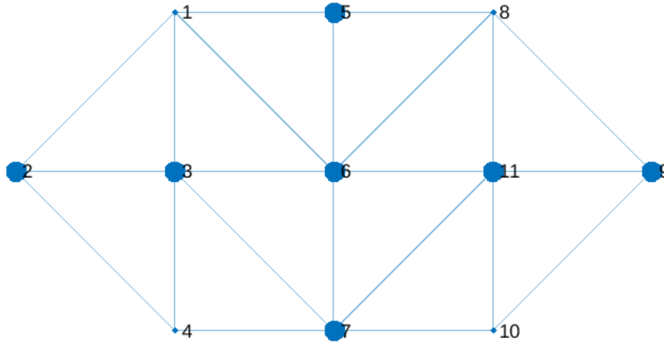


Рис. 2.13. Мінімальне вершинне покриття

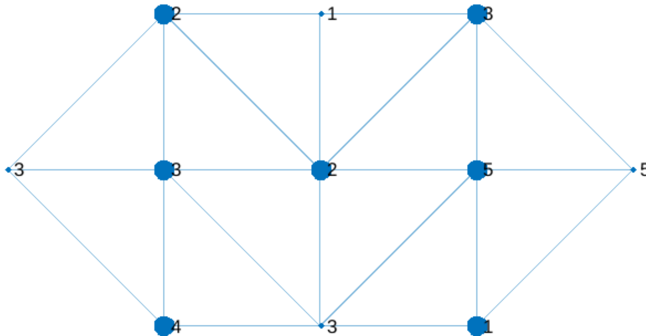


Рис. 2.14. Мінімальне зважене вершинне покриття

На рис. 2.13 показаний розв'язок незваженої задачі, а на рис. 2.14 — зваженої. У незваженій задачі мітки вершин — це їхні номери, а у зваженій — ваги. □

2.5. Двоїстість задач про пакування та покриття

Порівняємо між собою незважені задачі про пакування та покриття (2.6, 2.12, 2.18, 2.24). Зведемо їх до однієї таблиці 2×2 (2.26). Перший

рядок — пакування, другий — покриття, перший стовпчик — реберні, другий — вершинні.

$$
\begin{array}{lcc}
 & \text{Реберні} & \text{Вершинні} \\
\text{Пакування} & \left\{\begin{array}{l} z = (\mathbf{1}, \boldsymbol{e}) \to \max; \\ \boldsymbol{Ae} \le \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1, m}. \end{array}\right. & \left\{\begin{array}{l} z = (\mathbf{1}, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T\boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}. \end{array}\right. \\
 & \nwarrow\!\nearrow\ \swarrow\!\searrow & \\
\text{Покриття} & \left\{\begin{array}{l} t = (\mathbf{1}, \boldsymbol{e}) \to \min; \\ \boldsymbol{Ae} \ge \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1, m}. \end{array}\right. & \left\{\begin{array}{l} t = (\mathbf{1}, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T\boldsymbol{v} \ge \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}. \end{array}\right.
\end{array}
\tag{2.26}
$$

Якщо замінити умови бінарності змінних e_k та v_i на менш жорсткі умови невід'ємності: $\forall e_k \ge 0$; $\forall v_i \ge 0$; то ми отримаємо дві пари взаємно двоїстих задач лінійного програмування. Вони відмічені стрілками. Для таких пар має місце низка теорем двоїстості. Зокрема, якщо існує розв'язок однієї з задач, то існує й розв'язок іншої, і в цьому випадку $z_{\max} = t_{\min}$. Спираючись на структуру матриці інцидентності $\boldsymbol{A}$ (вона складається з нулів та одиниць), ми можемо бути впевнені, що розв'язки всіх чотирьох задач за умови невід'ємності змінних напевно існують. Тому рівності $z_{\max} = t_{\min}$ для обох пар задач виконуються. Позначимо це число:

$$z_{\max} = t_{\min} = \omega_{\text{opt}}. \tag{2.27}$$

У наших задачах (2.6, 2.12, 2.18, 2.24) області допустимих розв'язків звужені у порівнянні з невід'ємними змінними: усі змінні є лише бінарними, тобто можуть приймати значення тільки 0 або 1. Тому $z_{\max}$, можливо, і не досягне значення ω_{opt}, а буде менше за нього, а $t_{\min}$, можливо, буде більшим за ω_{opt}. Як наслідок, для кожної пари задач з (2.26) маємо нерівність:

$$z_{\max} \le t_{\min}. \tag{2.28}$$

Це означає: в одному й тому ж графі кількість ребер у максимальному паросполученні не може перевищувати кількості вершин у мінімальному вершинному покритті. Так само: в одному й тому ж графі кількість вершин у їхній максимальній незалежній множині не може бути більшою за кількість ребер у мінімальному реберному покритті.

Розглянемо тепер відповідні зважені задачі (2.7, 2.13, 2.19, 2.25). Як би ми не зважували ребра, при побудові паросполучення вимога несуміжності ребер залишається. Тому кількість ребер у зваженому макси-

мальному паросполученні ніколи не буде більшою, ніж у незваженому. Так само, при побудові вершинного покриття, якими б не були ваги вершин, вершини з покриття все одно повинні покривати всі ребра. Тому кількість вершин у зваженому мінімальному вершинному покритті ніколи не буде меншою, ніж у незваженому. Отже, і в зважених задачах ми можемо зробити такий саме висновок: кількість ребер у максимальному зваженому паросполученні ніколи не буде перевищувати кількості вершин у мінімальному зваженому вершинному покритті. Але нерівність (2.28) у зважених задачах у загальному випадку вже не має місця, оскільки тепер $z_{\max}$ і $t_{\min}$ обчислюються через ваги ребер і вершин, а не через їхню кількість.

Такі саме висновки можна зробити й для другої пари двоїстих задач: кількість вершин у максимальній зваженій незалежній множині не може перевищувати кількості ребер у мінімальному зваженому реберному покритті (а загальна вага може).

2.6. Домінуюча множина ребер

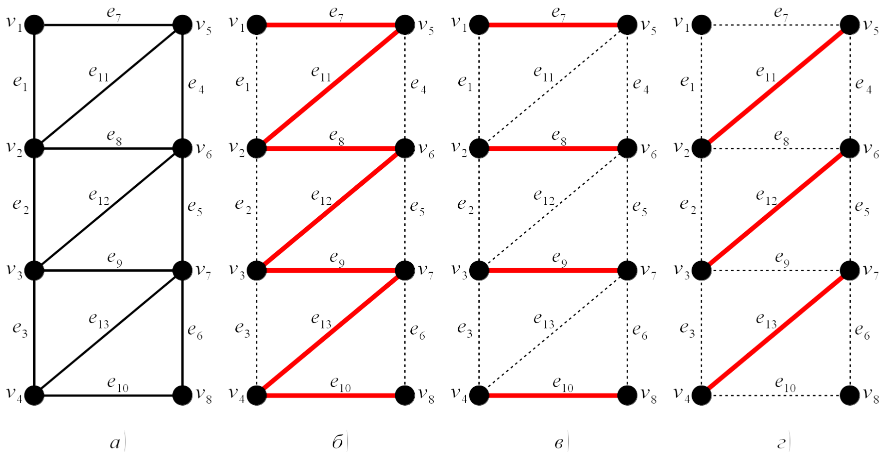


Рис. 2.15. Граф (а), його домінуюча множина ребер (б), мінімальна за включенням домінуюча множина ребер (в) та мінімальна домінуюча множина ребер (г)

Означення 2.6. *Домінуючою множиною ребер* (edge dominating set) у графі $G = (V, E)$ називається така підмножина ребер $E_1 \subset E$, що будь-яке ребро графа або належить до E_1, або є суміжним з ребром із E_1. Якщо вважати, що кожне ребро є суміжним із самим собою, то домінуюча множина ребер — це підмножина ребер E_1, до ребер якої суміжні всі ребра

графа. Інша назва: *зовнішньо стійка множина ребер* (edge external stability set). Домінуюча множина ребер називається *мінімальною за включенням*, якщо будь-яка її підмножина з меншою кількістю ребер не є домінуючою. Домінуюча множина ребер називається *мінімальною*, якщо вона містить у собі мінімально можливу кількість ребер. □

На рис. 2.15, *а* показаний граф з $n = 8$ вершинами та $m = 13$ ребрами (той самий приклад, що й раніше). Одна з його домінуючих множин ребер — $\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}\}$ з рис. 2.15, *б*. Її ребра є суміжними з усіма ребрами. Ця домінуюча множина не є мінімальною за включенням: з неї можна вилучити, наприклад, ребра e_{11}, e_{12} та e_{13} (рис. 2.15, *в*). Ті ребра, що залишилися: $\{e_7, e_8, e_9, e_{10}\}$, також є суміжними з усіма ребрами. Отже, це домінуюча множина ребер. Якщо вилучити з неї ще будь-яке ребро, якесь із ребер не охопиться суміжністю. Отже, домінуюча множина ребер $\{e_7, e_8, e_9, e_{10}\}$ є мінімальною за включенням. Але це не мінімальна домінуюча множина: можна створити домінуючу множину не з чотирьох, а з трьох ребер, що будуть суміжними з усіма ребрами. Одна з таких мінімальних домінуючих множин $\{e_{11}, e_{12}, e_{13}\}$ представлена на рис. 2.15, *г*.

Однією з задач теорії графів є знаходження мінімальної домінуючої множини ребер. Якщо ребра зважені, можна ставити задачу про мінімальну зважену домінуючу множину ребер: знайти домінуючу множину не з найменшою потужністю, а з найменшою загальною вагою.

Приклад з військової справи. Є опорні пункти супротивника (вершини графа), пов'язані між собою комунікаціями (ребра). Якщо знищення комунікації знищує також усі суміжні комунікації, то маємо задачу: як знищити всі комунікації супротивника, завдавши найменшу кількість ударів по комунікаціях? Інколи знищувати одні комунікації легше (вага відповідного ребра менша), а інші важче (більша вага). Тоді маємо зважену задачу: найменшою ціною знищити всі комунікації супротивника.

Розглянемо формулювання задачі про мінімальну (зважену) домінуючу множину ребер як задачу БЛП. Як і в попередніх задачах, введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Його координати є асоційованими з ребрами графа E. Якщо ребро e_k входить у шукану домінуючу множину, то асоційована з ним змінна e_k буде приймати значення 1, а якщо ні — то 0. Тоді загальну кількість ребер, що входять у домінуючу множину, можна записати у вигляді:

$$t = \sum_{k=1}^{m} e_k = (\mathbf{1}, \boldsymbol{e})\,. \tag{2.29}$$

В задачі про мінімальну домінуючу множину ребер цю величину треба мінімізувати за умови, що всі змінні e_k можуть приймати значення тільки 0 або 1:

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1, m}; \end{cases} \tag{2.30}$$

і ребра є суміжними до всіх ребер. Ця вимога означає, що для кожного ребра або воно саме, або якесь суміжне з ним входить до домінуючої множини. Тобто: для кожного ребра e_k сума змінних e_k та всіх e_l, суміжних з e_k, не повинна бути меншою за одиницю. Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. До ребра e_1 є суміжними ребра e_2, e_3 та e_4; тому сума змінних e_1, e_2, e_3 та e_4 не повинна бути меншою за одиницю. До ребра e_2 є суміжними ребра e_1, e_3 та e_5; тому сума змінних e_2, e_1, e_3 та e_5 також не повинна бути меншою за одиницю тощо. Ось як виглядає ця система обмежень-нерівностей для графа з рис. 2.2:

$$\begin{array}{ll} e_1: & e_1 + e_2 + e_3 + e_4 \geq 1; \\ e_2: & e_2 + e_1 + e_3 + e_5 \geq 1; \\ e_3: & e_3 + e_1 + e_2 + e_4 + e_5 \geq 1; \\ e_4: & e_4 + e_1 + e_3 + e_5 \geq 1; \\ e_5: & e_5 + e_2 + e_3 + e_4 \geq 1. \end{array} \tag{2.31}$$

Якщо скористатися матрицею суміжності ребер, яка для графа з рис. 2.2 має вигляд:

$$\boldsymbol{C} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \tag{2.32}$$

то умова суміжності всіх ребер з ребрами домінуючої множини записується так:

$$(\boldsymbol{C} + \boldsymbol{E})\,\boldsymbol{e} \geq \mathbf{1}, \tag{2.33}$$

де $\boldsymbol{E}$ — одинична матриця. Отже, маємо задачу БЛП:

$$\begin{cases} t = (\mathbf{1}, \boldsymbol{e}) \to \min; \\ \boldsymbol{C}_1 \boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1;\; k = \overline{1, m}, \end{cases} \tag{2.34}$$

де

$$\boldsymbol{C}_1 = \boldsymbol{C} + \boldsymbol{E} \tag{2.35}$$

— матриця суміжності ребер з одиницями на головній діагоналі.

В задачі про мінімальну зважену домінуючу множину ребер треба мінімізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами ребер $\boldsymbol{c}$. Тоді замість (2.34) будемо мати задачу БЛП, що відрізняється від (2.34) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{c}, \boldsymbol{e}) \to \min; \\ \boldsymbol{C}_1 \boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1; \; k = \overline{1, m}. \end{cases} \tag{2.36}$$

У Graph Theory Toolbox є метод `mindomedgeset`, який для графа `G` повертає вектор-рядок з номерами ребер, включених у мінімальну домінуючу множину. Якщо ребра графа зважені, розв'язується задача (2.36), а якщо ні — то (2.34).

Приклад 2.6. Для графа з прикладу 2.4 знайти мінімальні незважену та зважену домінуючі множини ребер.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 5]; % ваги
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
ms = mindomedgeset(G); % розв'язуємо незважену задачу
MinSize = length(ms); % мінімальний розмір
EdgeWidth = ones(numedges(G),1); % товщини ліній ребер
EdgeWidth(ms) = 5; % товщини ліній ребер домінуючої множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"LineWidth",EdgeWidth) % рисуємо
title("Кількість ребер у мінімальній домінуючій " + ...
  "множині = " + MinSize) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinDomEdgeSet","-dpng") % зберегли рисунок у файл
G = graph(s,t,w); % зважений граф
ms = mindomedgeset(G); % розв'язуємо зважену задачу
MinWeight = sum(G.Edges.Weight(ms)); % мінімальна вага
EdgeWidth = ones(numedges(G),1); % товщини ліній ребер
EdgeWidth(ms) = 5; % товщини ліній ребер домінуючої множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"EdgeLabel",G.Edges.Weight, ...
```

```
  "LineWidth",EdgeWidth) % нарисували
title("Вага ребер мінімальної домінуючої множини = " + ...
  MinWeight) % заголовк
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("WeightedMinDomEdgeSet","-dpng") % зберегли рисунок
```

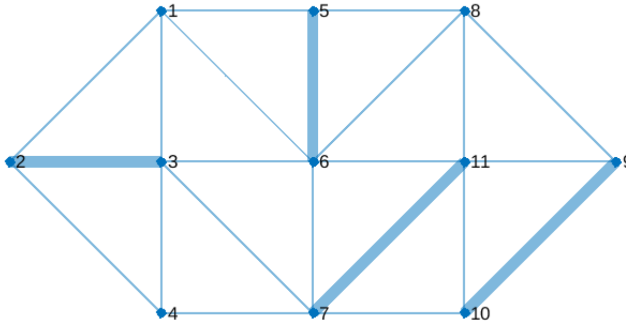


Рис. 2.16. Мінімальна домінуюча множина ребер

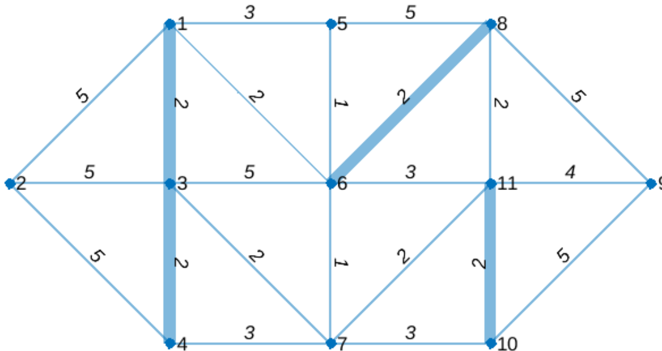


Рис. 2.17. Мінімальна зважена домінуюча множина ребер

На рис. 2.16 показаний розв'язок незваженої задачі, а на рис. 2.17 — зваженої. Мітки ребер у зваженій задачі — ваги ребер. □

Зауважимо, що двоїста до (2.34) задача в силу симетрії матриці $\boldsymbol{C}_1$ буде мати вигляд:

$$\begin{cases} z = (\mathbf{1}, \boldsymbol{e}) \to \max; \\ \boldsymbol{C}_1 \boldsymbol{e} \le \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1, m}. \end{cases} \tag{2.37}$$

Двоїстими змінними тут будуть не вершини, а теж ребра. Сенс цієї задачі: знайти таку підмножину ребер максимальної потужності або ваги E_1, щоб кожне інше ребро, що не входить до E_1, було суміжним не більш ніж з одним ребром з E_1 (а може, й взагалі не було суміжним з ребрами з E_1). Тобто між кожною парою ребер з E_1 повинно бути не менше двох кроків уздовж ребер.

2.7. Домінуюча множина вершин

Означення 2.7. *Домінуючою множиною вершин* (vertex dominating set) у графі $G = (V, E)$ називається така підмножина вершин $V_1 \subset V$, що будь-яка вершина графа або належить до V_1, або є суміжною з вершиною із V_1. Якщо вважати, що кожна вершина є суміжною із самою собою, то домінуюча множина вершин — це підмножина вершин V_1, до вершин якої суміжні всі вершини графа. Інша назва: *зовнішньо стійка множина вершин* (vertex external stability set). Домінуюча множина вершин називається *мінімальною за включенням*, якщо будь-яка її підмножина з меншою кількістю вершин не є домінуючою. Домінуюча множина вершин називається *мінімальною*, якщо вона містить мінімально можливу кількість вершин. □

На рис. 2.18, *а* показаний граф з $n = 8$ вершинами та $m = 13$ ребрами (той самий приклад, що й раніше). Одна з його домінуючих множин вершин — це $\{v_1, v_3, v_6, v_8\}$ з рис. 2.18, *б*. Її вершини є суміжними з усіма вершинами. Ця домінуюча множина не є мінімальною за включенням: з неї можна вилучити, наприклад, вершину v_6 (рис. 2.18, *в*). Ті вершини, що залишилися: $\{v_1, v_3, v_8\}$, також є суміжними з усіма вершинами. Отже, це домінуюча множина вершин. Якщо вилучити з неї ще будь-яку вершину, якась із вершин не охопиться суміжністю. Отже, домінуюча множина вершин $\{v_1, v_3, v_8\}$ є мінімальною за включенням. Але це не мінімальна домінуюча множина: можна створити домінуючу множину не з трьох, а з двох вершин, що будуть суміжними з усіма вершинами: це $\{v_4, v_5\}$. Вона представлена на рис. 2.18, *г*.

Однією з класичних задач теорії графів є знаходження мінімальної домінуючої множини вершин. Якщо вершини зважені, можна ставити задачу про мінімальну зважену домінуючу множину вершин: знайти домінуючу множину не з найменшою кількістю вершин, а з найменшою їх загальною вагою.

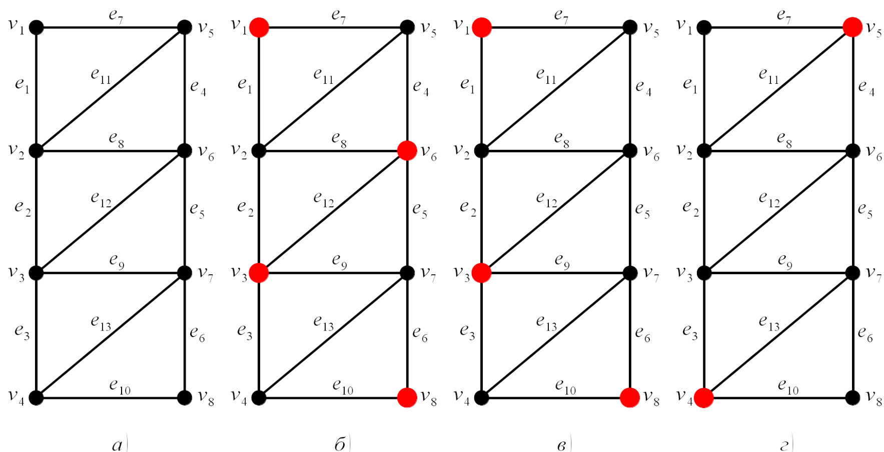


Рис. 2.18. Граф (а), його домінуюча множина вершин (б), мінімальна за включенням домінуюча множина вершин (в) та мінімальна домінуюча множина вершин (г)

Приклад з військової справи. Є опорні пункти супротивника (вершини графа), пов'язані між собою комунікаціями (ребра). Якщо знищення опорного пункту знищує також усі суміжні опорні пункти, то маємо задачу: як знищити всі опорні пункти супротивника, завдавши найменшу кількість ударів по опорних пунктах? Інколи знищувати одні опорні пункти легше (вага відповідної вершини менша), а інші важче (більша вага). Тоді маємо зважену задачу: найменшою ціною знищити всі опорні пункти супротивника.

Розглянемо формулювання задачі про мінімальну (зважену) домінуючу множину вершин як задачу БЛП. Як і в попередніх задачах, введемо до розгляду вектор-стовпчик $\boldsymbol{v}$ довжиною n. Його координати є асоційованими с вершинами графа V. Якщо вершина v_i входить у шукану домінуючу множину, то асоційована з нею змінна v_i буде приймати значення 1, а якщо ні — то 0. Тоді загальну кількість вершин, що входять у їх домінуючу множину, можна записати у вигляді:

$$t = \sum_{i=1}^{n} v_i = (\mathbf{1}, \boldsymbol{v}) \,. \tag{2.38}$$

В задачі про мінімальну домінуючу множину вершин цю величину треба мінімізувати за умови, що всі змінні v_i можуть приймати значення тільки 0 або 1:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1, n}, \end{cases} \tag{2.39}$$

і її вершини є суміжними до всіх вершин. Ця вимога означає, що для кожної вершини або вона сама, або якась суміжна з нею входить до домінуючої множини. Тобто: для кожної вершини v_i сума змінних v_i та всіх v_j, суміжних з v_i, не повинна бути меншою за одиницю. Так, для графа з рис. 2.2 ця система нерівностей-обмежень виглядає наступним чином. До вершини v_1 є суміжними вершини v_2 та v_3; тому сума змінних v_1, v_2 та v_3 не повинна бути меншою за одиницю. До вершини v_2 суміжними є вершини v_1, v_3 та v_4; тому сума змінних v_2, v_1, v_3 та v_4 також не повинна бути меншою за одиницю тощо. Ось як виглядає ця система обмежень-нерівностей для графа з рис. 2.2:

$$\begin{array}{ll} v_1 : & v_1 + v_2 + v_3 \geq 1; \\ v_2 : & v_2 + v_1 + v_3 + v_4 \geq 1; \\ v_3 : & v_3 + v_1 + v_2 + v_4 \geq 1; \\ v_4 : & v_4 + v_2 + v_3 \geq 1. \end{array} \tag{2.40}$$

Якщо скористатися матрицею суміжності вершин, яка для графа з рис. 2.2 має вигляд:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \tag{2.41}$$

то умова суміжності всіх вершин з вершинами домінуючої множини записується так:

$$(\boldsymbol{B} + \boldsymbol{E})\, \boldsymbol{e} \geq \mathbf{1}, \tag{2.42}$$

де $\boldsymbol{E}$ — одинична матриця. Отже, маємо задачу БЛП:

$$\begin{cases} t = (\mathbf{1}, \boldsymbol{v}) \to \min; \\ \boldsymbol{B}_1 \boldsymbol{v} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1, n}, \end{cases} \tag{2.43}$$

де

$$\boldsymbol{B}_1 = \boldsymbol{B} + \boldsymbol{E} \tag{2.44}$$

— матриця суміжності вершин з одиницями на головній діагоналі.

В задачі про мінімальну зважену домінуючу множину вершин треба мінімізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з вагами вершин $\boldsymbol{b}$. Тоді замість (2.43) будемо мати задачу БЛП, що відрізняється від (2.43) тільки цільовою функцією:

$$
\begin{cases}
t = (\boldsymbol{b}, \boldsymbol{v}) \to \min; \\
\boldsymbol{B}_1 \boldsymbol{v} \geq \mathbf{1}; \\
v_i = 0 \vee 1;\ i = \overline{1, n}.
\end{cases}
\tag{2.45}
$$

В інструментарії Graph Theory Toolbox є метод `mindomverset`, який для графа `G` повертає вектор-рядок з номерами вершин, включених у мінімальну домінуючу множину. Якщо вершини графа зважені, розв'язується задача (2.45), а якщо ні — то (2.43).

Приклад 2.7. Для графа зі зваженими вершинами з прикладу 2.5 знайти мінімальні незважену та зважену домінуючі множини вершин.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
d = [2 3 3 4 1 2 3 3 5 1 5]; % ваги вершин
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
ms = mindomverset(G); % розв'язуємо незважену задачу
MinSize = length(ms); % розмір мінімальної домінуючої множини
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин домінуючої множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"MarkerSize",NodeSize) % рисуємо
title("Розмір мінімальної домінуючої множини " + ...
  "вершин = " + MinSize) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinDomVerSet","-dpng") % зберегли рисунок у файл
G = graph(s,t); % незважений граф
G.Nodes.Weight = d'; % додали ваги вершин
ms = mindomverset(G); % розв'язуємо зважену задачу
MinWeight = sum(G.Nodes.Weight(ms)); % вага
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин домінуючої множини
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",G.Nodes.Weight, ...
  "MarkerSize",NodeSize) % рисуємо
title("Вага мінімальної домінуючої множини вершин = " + ...
  MinWeight) % заголовок
```

```
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("WeightedMinDomVerSet","-dpng") % зберегли рисунок
```

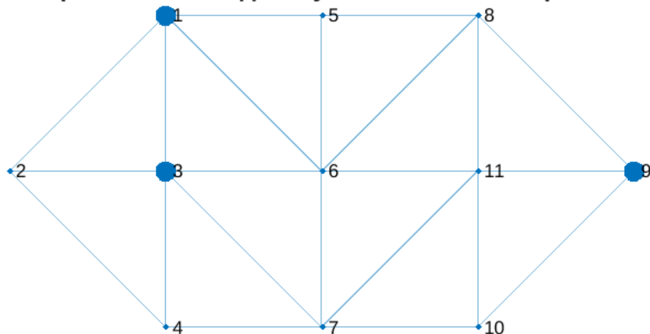


Рис. 2.19. Мінімальна домінуюча множина вершин

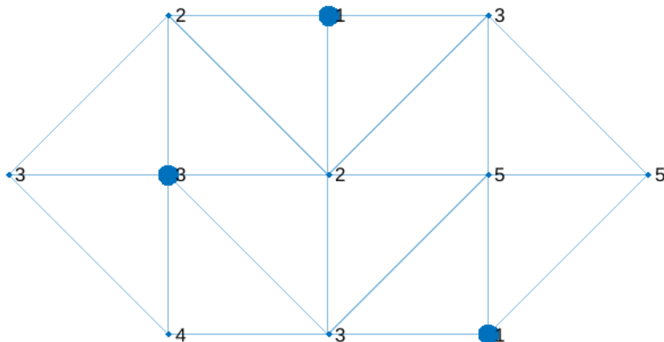


Рис. 2.20. Мінімальна зважена домінуюча множина вершин

На рис. 2.19 показаний розв'язок незваженої задачі, а на рис. 2.20 — зваженої. У незваженій задачі мітки вершин — це їхні номери, а у зваженій — ваги. □

Зауважимо, що двоїста до (2.43) задача в силу симетрії матриці $\boldsymbol{B}_1$ буде мати вигляд:

$$\begin{cases} z = (\mathbf{1}, \boldsymbol{v}) \to \max; \\ \boldsymbol{B}_1 \boldsymbol{v} \leq \mathbf{1}; \\ v_i = 0 \vee 1; \; i = \overline{1, n}. \end{cases} \tag{2.46}$$

Двоїстими змінними тут будуть не ребра, а теж вершини. Сенс цієї задачі: знайти таку підмножину вершин максимальної потужності або ваги V_1, щоб кожна інша вершина, що не входить до V_1, була суміжна не більш ніж з однією вершиною з V_1 (а може, й взагалі не була б суміжною з вершинами з V_1). Тобто між кожною парою вершин з V_1 повинно бути не менше двох проміжних вершин, і, як наслідок, не менше трьох кроків уздовж ребер.

2.8. Повний підграф

Означення 2.8. *Повний підграф* (complete subgraph) у заданому графі $G = (V, E)$ — це підмножина вершин $V_1 \subset V$, що є взаємно суміжними. Інша назва: *кліка* (clique). Кліка називається *максимальною за включенням*, якщо вона не є підмножиною кліки більшого розміру. Кліка називається *максимальною*, якщо вона складається з максимально можливої кількості вершин. □

Якщо вершини графа незважені, ставиться задача знаходження максимальної кліки. Для графа зі зваженими вершинами можна ставити задачу знаходження максимальної зваженої кліки: підмножини взаємно суміжних вершин максимальної загальної ваги.

Кліка є поняттям, протилежним до незалежної множини вершин. Тому й задача про максимальну (зважену) кліку в графі $G = (V, E)$ еквівалентна до задачі про максимальну (зважену) незалежну множину вершин для графа $\overline{G} = \left(V, \overline{E}\right)$, де $\overline{E}$ — ребра, яких немає в графі G, тобто та множина ребер, що доповнює граф G до кліки. Отже, щоб звести задачу про максимальну кліку до задачі БЛП, треба:

1. побудувати граф $\overline{G}$, тобто знайти всі ребра, яких немає в G;
2. розв'язати для графа $\overline{G}$ задачу (2.12) про максимальну незалежну множину вершин або (2.13) про максимальну зважену незалежну множину вершин.

Цю задачу розв'язує метод `maxcompsub`, що міститься у пакеті Graph Theory Toolbox. Він для заданого графа `G` повертає вектор-рядок з номерами вершин, включених у максимальну кліку. Якщо вершини графа зважені, знаходиться максимальна зважена кліка, а якщо ні — то просто максимальна кліка.

Приклад 2.8. Для графа зі зваженими вершинами з прикладу 2.3 знайти максимальні незважену та зважену кліки.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
```

```
  10 11 10 9 9 9]; % кінці ребер
d = [2 3 3 4 1 2 3 3 5 1 5]; % ваги вершин
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
ms = maxcompsub(G); % максимальна незважена кліка
MaxSize = length(ms); % розмір кліки
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин кліки
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"MarkerSize",NodeSize) % рисуємо
title("Кількість вершин у максимальній кліці = " + MaxSize)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MaxCompSub","-dpng") % зберегли рисунок у файл
G = graph(s,t); % незважений граф
G.Nodes.Weight = d'; % додали ваги вершин
ms = maxcompsub(G); % максимальна незважена кліка
MaxWeight = sum(G.Nodes.Weight(ms)); % вага кліки
NodeSize = ones(numnodes(G),1)*2; % розміри міток вершин
NodeSize(ms) = 8; % розміри міток вершин кліки
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",G.Nodes.Weight, ...
  "MarkerSize",NodeSize) % рисуємо
title("Загальна вага вершин у максимальній кліці = " + ...
  MaxWeight) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("WeightedMaxCompSub","-dpng") % зберегли рисунок
```

На рис. 2.21 показаний розв'язок незваженої задачі, а на рис. 2.22 — зваженої. У незваженій задачі мітки вершин — це їхні номери, а у зваженій — ваги. □

2.9. Запитання для перевірки

1. Що таке паросполучення? максимальне за включенням? максимальне? зважене?
2. Як у MATLAB розв'язується задача про максимальне (зважене) паросполучення?
3. Як у MATLAB розв'язується задача про призначення?
4. Що таке незалежна множина вершин? максимальна за включенням? максимальна? зважена?

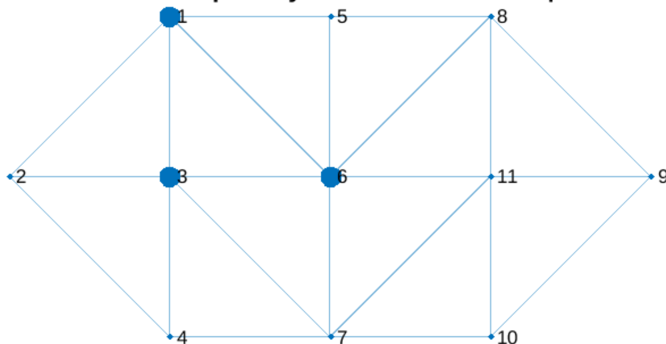


Рис. 2.21. Максимальна кліка

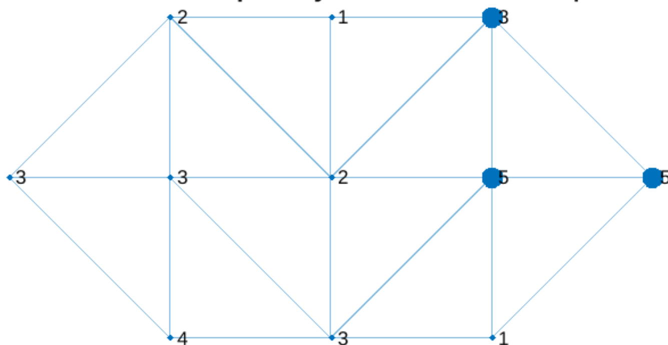


Рис. 2.22. Максимальна зважена кліка

5. Як у MATLAB розв'язується задача про максимальну (зважену) незалежну множину вершин?
6. Що таке реберне покриття? мінімальне за включенням? мінімальне? зважене?
7. Як у MATLAB розв'язується задача про мінімальне (зважене) реберне покриття?
8. Що таке вершинне покриття? мінімальне за включенням? мінімальне? зважене?
9. Як у MATLAB розв'язується задача про мінімальне (зважене) вершинне покриття?
10. Які задачі лінійного програмування називаються взаємно двоїстими?
11. Що таке домінуюча множина ребер? мінімальна за включенням?

мінімальна? зважена?

12. Як у MATLAB розв'язується задача про мінімальну (зважену) домінуючу множину ребер?
13. Що таке домінуюча множина вершин? мінімальна за включенням? мінімальна? зважена?
14. Як у MATLAB розв'язується задача про мінімальну (зважену) домінуючу множину вершин?
15. Що таке кліка? максимальна за включенням? максимальна? зважена?
16. Як у MATLAB розв'язується задача про максимальну (зважену) кліку?

3. Правильна розфарбовка графів

У цьому розділі будуть розглянуті дві задачі: про мінімальну правильну розфарбовку вершин та ребер графа. Обидві вони формулюються як задачі цілочисельного лінійного програмування (ЦЛП).

3.1. Мінімальна правильна розфарбовка вершин графа

Означення 3.1. Граф $G = (V, E)$ називається графом з *розфарбованими вершинами* (colored vertices), якщо задане відображення множини вершин V на множину натуральних чисел:

$$\varphi : \ V \to \mathbb{N}. \ \square \tag{3.1}$$

Натуральні числа можна вважати номерами кольорів за якоюсь таблицею. Наприклад, колір номер 1 — це жовтий, 2 — блакитний, 3 — червоний, 4 — чорний тощо. Тоді кожна вершина фарбується у якийсь колір. Замість множини натуральних чисел $\mathbb{N}$ можна використовувати будь-яку зліченну множину чисел: цілі, раціональні, алгебраїчні тощо. Але ми будемо для зручності позначати кольори натуральними числами — їхніми номерами.

Означення 3.2. Розфарбовка вершин графа називається *правильною* (regular vertex coloring), якщо суміжні вершини фарбуються різними кольорами. *Мінімальною правильною розфарбовкою вершин* графа (minimum regular vertex coloring) називається правильна розфарбовка мінімальною кількістю фарб. Кількість фарб у мінімальній правильній розфарбовці називається *хроматичним числом* графа (chromatic number), та позначається $\chi(G)$. $\square$

У двох крайніх випадках розв'язок задачі про мінімальну правильну розфарбовку вершин є тривіальним. Так, для пустого графа O_n (без ребер) усі вершини можуть бути пофарбовані однією фарбою, тому його хроматичне числом $\chi(O_n) = 1$. А у кліці K_n кожну вершину треба фарбувати своїм кольором, тому тут $\chi(K_n) = n$.

Спробуємо розв'язати цю задачу за допомогою *жадібного алгоритму* (greedy algorithm), який ми докладніше розглянемо у наступному розділі 4. Для цього знайдемо у графі максимальну незалежну множину вершин. Ці вершини є несуміжними між собою, тому їх можна пофарбувати одним кольором. Пофарбуємо їх кольором номер 1. Вилучимо з графа ці вершини та інцидентні до них ребра. В тому графі, що залишився, знову знайдемо максимальну незалежну множину вершин та пофарбуємо їх у колір номер 2, і т. д. до повного вичерпання вершин. Такий алгоритм, безумовно, буде давати правильну розфарбовку: суміжні вершини завжди

будуть пофарбовані різними кольорами. Але чи завжди ця правильна розфарбовка буде мінімальною?

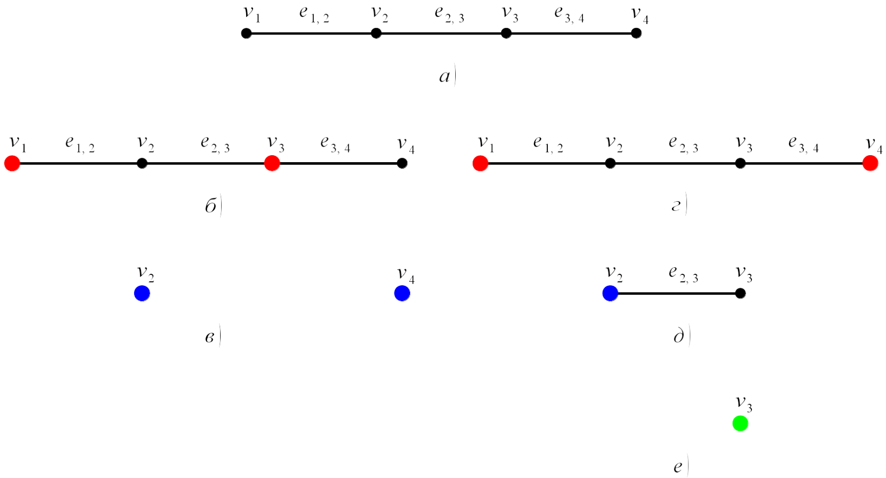


Рис. 3.1. Жадібний алгоритм не завжди дає мінімальну правильну розфарбовку вершин графа

Поглянемо на рис. 3.1. На рис. 3.1, *а* показаний граф з $n = 4$ вершинами та $m = 3$ ребрами. Якщо взяти максимальну незалежну множину вершин $\{v_1, v_3\}$ та надати цим вершинам колір номер 1 (на рис. 3.1, *б* це червоний), то на наступному кроці від графа залишаться лише вершини v_2 та v_4, які не є суміжними, бо ми видалили ребра, інцидентні до v_1 та v_3. Цим вершинам v_2 та v_4 можна надати колір номер 2 (на рис. 3.1, *в* це синій). Як бачимо, в цьому випадку задача розв'язана вірно: маємо два кольори у мінімальній правильній розфарбовці. Менше не буває, тому що в графі є ребра.

Але, якщо взяти ось таку максимальну незалежну множину вершин: $\{v_1, v_4\}$, а це теж правильний результат розв'язання задачі про максимальну незалежну множину вершин (рис. 3.1, *г*), то правильна розфарбовка вже не буде мінімальною. Дійсно: після вилучення вершин v_1, v_4 та інцидентних до них ребер отримаємо граф, показаний на рис. 3.1, *д*. В ньому тільки одній вершині, наприклад, v_2, можна надати колір номер 2, а вершину v_3 вже треба фарбувати кольором номер 3 (рис. 3.1, *е*). Як бачимо, жадібний алгоритм не завжди дає правильний розв'язок задачі про мінімальну правильну розфарбовку вершин.

Розглянемо алгоритм, що дає правильний розв'язок. Сформулюємо нашу задачу як задачу ЦЛП. Номер фарби — це натуральне число, тому введемо до розгляду цілочисельні змінні v_i, асоційовані з вершинами.

Усього маємо n таких змінних, і кожна з них може приймати значення від 1 до n (або від 0 до $n-1$, якщо це дозволяє мова програмування): це номер фарби вершини v_i. В найгіршому випадку кожна вершина буде пофарбована у свій колір, тому максимальне значення кожної змінної v_i — це n:

$$\begin{cases} v_i \in \{\overline{1,n}\}; \\ i = \overline{1,n}. \end{cases} \tag{3.2}$$

Нам треба мінімізувати максимальне v_i:

$$t = \max_{i=\overline{1,n}} v_i \to \min . \tag{3.3}$$

Зручніше за все ввести для цього до розгляду ще одну додаткову змінну v_0 (теж цілочисельну), і пов'язати її з усіма іншими v_i системою лінійних нерівностей:

$$\begin{cases} v_i \leq v_0; \\ i = \overline{1,n}. \end{cases} \tag{3.4}$$

Тоді цільова функція — це:

$$t = v_0 \to \min . \tag{3.5}$$

У задачі про мінімальну правильну розфарбовку суміжні вершини повинні мати різні кольори (номери фарб). Це означає: для кожного ребра $e_{ij} \in E$ змінні v_i та v_j, що відповідають вершинам, інцидентним до ребра e_{ij}, повинні відрізнятися хоча б на одиницю:

$$\begin{cases} |v_i - v_j| \geq 1; \\ \forall e_{ij} \in E. \end{cases} \tag{3.6}$$

Ці обмеження (3.6) — нелінійні: в них є модуль. Перехід від кожної нерівності з модулем (3.6) до двох нерівностей без модулів:

$$\begin{cases} \left[\begin{array}{l} v_i - v_j \geq 1; \\ v_j - v_i \geq 1; \end{array} \right. \\ \forall e_{ij} \in E \end{cases} \tag{3.7}$$

теж нічого не дає, оскільки маємо не систему, а об'єднання нерівностей (треба, щоб виконувалася або одна, або інша нерівність). Але сформулювати (3.6) як систему лінійних нерівностей все ж таки можливо. Механізм для цього описаний в [11]. Щоб це зробити, зауважимо спочатку, що змінні v_i відрізняються одна від одної не більш ніж на $n-1$, оскільки максимальний номер фарби — це n, а мінімальний — 1. Тому насправді замість (3.7) ми маємо:

$$\begin{cases} \left[\begin{array}{l} 1 \le v_i - v_j \le n-1; \\ 1 \le v_j - v_i \le n-1; \end{array} \right. \\ \forall e_{ij} \in E; \end{cases} \tag{3.8}$$

при цьому праві нерівності автоматично виконуються в силу (3.2). Тепер введемо до розгляду бінарні змінні e_{ij}, асоційовані з ребрами, яким дозволимо приймати лише одне з двох можливих значень: 0 або 1:

$$\begin{cases} e_{ij} = 0 \vee 1; \\ \forall e_{ij} \in E. \end{cases} \tag{3.9}$$

Розглянемо систему нерівностей (вже не об'єднання, а саме систему):

$$\begin{cases} v_i - v_j - ne_{ij} \le -1; \\ v_j - v_i + ne_{ij} \le n-1; \\ \forall e_{ij} \in E. \end{cases} \tag{3.10}$$

Якщо змінна e_{ij} приймає значення 0, то система (3.10) дає другу пару нерівностей з (3.8), а якщо 1 — то першу.

Отже, маємо таку задачу ЦЛП. Необхідно мінімізувати функцію t (3.5), яка формально залежить від $n+m+1$ змінних $v_0, v_1, \dots, v_n, \forall e_{ij}$, але фактично до неї входить тільки v_0. На змінні v_i накладені обмеження (3.4) — усього n обмежень-нерівностей. Для кожної змінної e_{ij} та відповідних до неї v_i та v_j повинні виконуватися по два обмеження (3.10) — усього $2m$ обмежень-нерівностей. Усі змінні v_i — цілочисельні та можуть приймати значення від 1 до n (3.2). Усі змінні e_{ij} — бінарні: вони можуть приймати значення тільки 0 або 1 (3.9).

Цей алгоритм реалізований у методі `minvercolor`, що включений до пакету Graph Theory Toolbox. Для графа `G` він повертає вектор-рядок з номерами фарб його вершин.

Приклад 3.1. Знайти мінімальну правильну розфарбовку вершин узагальненого графа Петерсена з 16 вершинами та 35 ребрами.

```
t = 0:4; % значення параметру
x = [5*sin(2*pi*t/5) 4*sin(2*pi*(t-0.5)/5) ...
  2*sin(2*pi*(t-0.5)/5) 0]; % координати вершин
y = [5*cos(2*pi*t/5) 4*cos(2*pi*(t-0.5)/5) ...
  2*cos(2*pi*(t-0.5)/5) 0];
s = [1 7 2 8 3 9 4 10 5 6 1 2 3 4 5 1 2 3 4 5 6 7 ...
  8 9 10 11 12 13 14 15 1 2 3 4 5]; % початки ребер
t = [7 2 8 3 9 4 10 5 6 1 10 6 7 8 9 12 13 14 15 11 14 ...
  15 11 12 13 12 13 14 15 11 16 16 16 16 16]; % кінці ребер
G = graph(s,t); % створили граф
```

```
vc = minvercolor(G); % розв’язали задачу
MaxColor = max(vc); % найменша кількість кольорів
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",vc,"MarkerSize",9,...
  "NodeColor",cm(round((vc-1)/(MaxColor-1)*255+1),:))
title("Мінімальна кількість фарб = " + MaxColor)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinVerColor","-dpng") % зберегли малюнок у файл
```

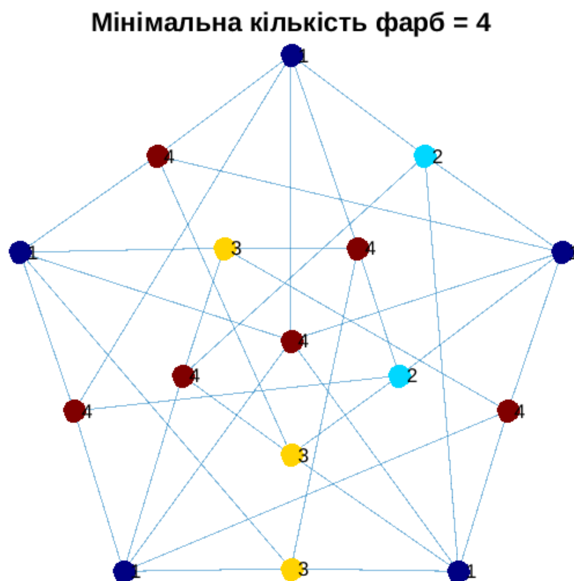


Рис. 3.2. Мінімальна правильна розфарбовка
вершин графа Петерсена

На рис. 3.2 показаний результат роботи цього алгоритму: мінімальна правильна розфарбовка чотирма фарбами одного з узагальнених графів Петерсена. □

Зауважимо, що обмеження (3.10) є досить складними для сучасних програм розв’язання задач ЦЛП. Для великих графів такі задачі можуть обчислюватися тривалий час або навіть зависнути. В цьому випадку доводиться обмежуватися жадібним алгоритмом поступового вилучення максимальних незалежних множин вершин. Інколи він теж дає правильний результат. Але так буває не завжди.

3.2. Мінімальна правильна розфарбовка ребер графа

Означення 3.3. Граф $G = (V, E)$ називається графом з *розфарбованими ребрами* (colored edges), якщо задане відображення множини ребер E на множину натуральних чисел:

$$\psi: \; E \to \mathbb{N}. \; \square \tag{3.11}$$

Натуральні числа можна вважати номерами кольорів за якоюсь таблицею. Тоді кожне ребро фарбується у якийсь колір. Як і при фарбуванні вершин, будемо для зручності позначати кольори натуральними числами — їхніми номерами.

Означення 3.4. Розфарбовка ребер графа називається *правильною* (regular edge coloring), якщо суміжні ребра фарбуються різними кольорами. *Мінімальною правильною розфарбовкою ребер* графа (minimum regular edge coloring) називається правильна розфарбовка мінімальною кількістю фарб. Кількість фарб у мінімальній правильній розфарбовці ребер графа називається *хроматичним індексом* графа (chromatic index), та позначається $\chi'(G)$. $\square$

Якщо максимальний ступінь вершин графа позначити як δ, то, вочевидь, хроматичний індекс $\chi'(G)$ не може бути меншим за δ. Дійсно: всі ребра, що є інцидентними до вершини з максимальним ступенем, повинні бути розфарбовані різними кольорами, а таких ребер як раз δ. З іншого боку, має місце теорема Візінга, доведення якої можна знайти, наприклад, у [13].

Теорема 3.1. Теорема Візінга (Vizing). Для простого графа його хроматичний індекс $\chi'(G) \leq \delta + 1$. $\square$

Згідно з теоремою Візінга всі прості графи можна розбити на два класи за хроматичним індексом: клас δ та клас $\delta + 1$. На жаль, визначення, до якого саме класу належить конкретний граф, є досить складною задачею.

Зауважимо, що теорема Візінга має місце лише для простих графів. Навіть для мультиграфів вона не завжди виконується. Наприклад, для трикутника з подвоєними ребрами $\delta = 4$, а для правильної розфарбовки ребер потрібно шість фарб.

Спробуємо розв'язати задачу про мінімальну правильну розфарбовку ребер за допомогою жадібного алгоритму (greedy algorithm). Для цього знайдемо у графі максимальне паросполучення. Його ребра є несуміжними між собою, тому їх можна пофарбувати одним кольором. Пофарбуємо їх кольором номер 1. Вилучимо з графа ці ребра. В тому графі, що залишився, знову знайдемо максимальне паросполучення та пофарбуємо його ребра у колір номер 2, і т. д. до повного вичерпання ребер. Такий

алгоритм, безумовно, буде давати правильну розфарбовку: суміжні ребра завжди будуть пофарбовані різними кольорами. Але чи завжди ця правильна розфарбовка буде мінімальною?

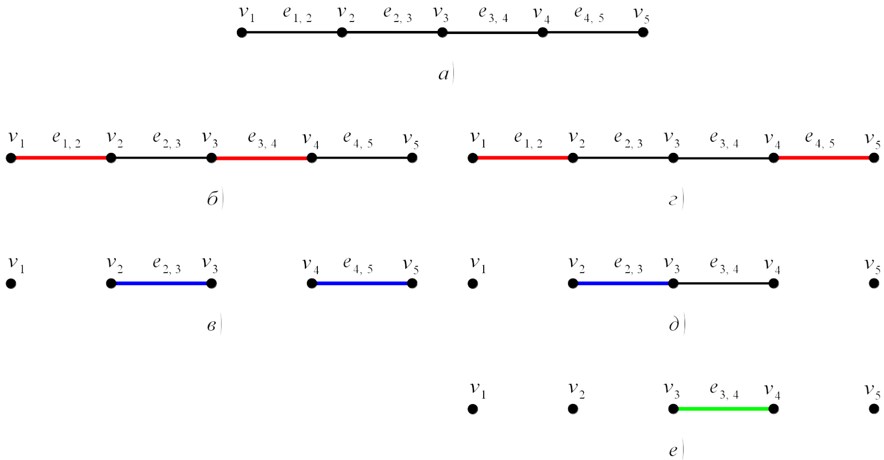


Рис. 3.3. Жадібний алгоритм не завжди дає мінімальну правильну розфарбовку ребер графа

Поглянемо на рис. 3.3. На рис. 3.3, *а* показаний граф з $n = 5$ вершинами та $m = 4$ ребрами. Якщо взяти максимальне паросполучення $\{e_{1,2}, e_{3,4}\}$ та пофарбувати його ребра у колір номер 1 (на рис. 3.3, *б* це червоний), то на наступному кроці у графі залишаться лише ребра $e_{2,3}$ та $e_{4,5}$, які не є суміжними. Ці ребра можна пофарбувати в колір номер 2 (на рис. 3.3, *в* це синій). Як бачимо, в цьому випадку задача розв'язана вірно: маємо два кольори у мінімальній правильній розфарбовці. Менше бути не може, тому що максимальний ступінь вершин у цьому графі дорівнює 2.

Але якщо взяти максимальне паросполучення $\{e_{1,2}, e_{4,5}\}$, а це теж правильний результат розв'язання задачі про максимальне паросполучення (див. рис. 3.3, *г*), то правильна розфарбовка вже не буде мінімальною. Дійсно, після вилучення ребер $e_{1,2}$ та $e_{4,5}$ отримаємо граф на рис. 3.3, *д*. В ньому тільки одному ребру, наприклад, $e_{2,3}$, можна надати колір номер 2, а ребро $e_{3,4}$ вже треба фарбувати кольором номер 3 (рис. 3.3, *е*). Як бачимо, тут, як і в випадку розфарбовки вершин, жадібний алгоритм не завжди дає правильний результат.

Розглянемо алгоритм, який дає правильний розв'язок нашої задачі. Запишемо її як задачу ЦЛП. Номер фарби — це натуральне число, тому введемо до розгляду цілочисельні змінні e_k, асоційовані з ребрами. Усього

маємо m таких змінних, і кожна з них може приймати значення від 1 до m (або від 0 до $m-1$, якщо так зручніше для програмування): це номер фарби ребра e_k. В найгіршому випадку кожне ребро буде мати свій колір, тому максимальне значення кожної змінної e_k — це m:

$$\begin{cases} e_k \in \{1, m\}\,; \\ k = \overline{1,m}. \end{cases} \tag{3.12}$$

Насправді за теоремою Візінга можна стверджувати, що кожна e_k не буде перевищувати $\delta + 1$. Нам треба мінімізувати максимальне e_k:

$$t = \max_{k=\overline{1,m}} e_k \to \min\,. \tag{3.13}$$

Зручніше за все ввести для цього до розгляду ще одну додаткову змінну e_0 (теж цілочисельну), і пов'язати її з усіма іншими e_k системою лінійних нерівностей:

$$\begin{cases} e_k \le e_0; \\ k = \overline{1,m}. \end{cases} \tag{3.14}$$

Тоді цільова функція — це:

$$t = e_0 \to \min\,. \tag{3.15}$$

У задачі про мінімальну правильну розфарбовку суміжні ребра повинні мати різні кольори (номери фарб). Це означає: для кожної вершини $v_i \in V$ змінні e_k, що відповідають ребрам, інцидентним до цієї вершини, повинні відрізнятися хоча б на одиницю:

$$\begin{cases} |e_{i1} - e_{i2}| \ge 1; \\ \forall v_i \in V; \end{cases} \tag{3.16}$$

де e_{i1}, e_{i2} — будь-яка пара ребер, інцидентних до вершини v_i. Якщо позначити ступінь вершини v_i (кількість інцидентних до неї ребер) як d_i, то усього для кожної вершини v_i буде $p_i = \frac{d_i(d_i-1)}{2}$ таких нерівностей. Як і в задачі про правильну розфарбовку вершин, ці нелінійні нерівності (3.16) можна звести до лінійних, якщо ввести до розгляду p_i бінарних змінних для кожної вершини v_i:

$$\begin{cases} v_{ij} = 0 \vee 1; \\ j = \overline{1,p_i}; \\ \forall v_i \in V. \end{cases} \tag{3.17}$$

Тоді маємо систему нерівностей:

$$
\begin{cases}
e_{i1} - e_{i2} - mv_{ij} \leq -1; \\
e_{i2} - e_{i1} + mv_{ij} \leq m - 1; \\
j = \overline{1, p_i}; \\
\forall v_i \in V.
\end{cases} \tag{3.18}
$$

Якщо змінна v_{ij} приймає значення 0, то система (3.18) дає нерівність (3.16) в один бік, а якщо 1 — то в інший. За теоремою Візінга у цих формулах замість m можна поставити $\delta + 1$.

Отже, маємо таку задачу ЦЛП. Необхідно мінімізувати функцію t (3.15), що формально залежить від $1 + m + p_1 + p_2 + \ldots + p_n$ змінних: $e_0, e_1, \ldots, e_m$, $\forall v_{ij}$, але фактично до неї входить тільки e_0. На змінні e_k накладені обмеження (3.14) — усього m обмежень-нерівностей. Для кожної змінної v_{ij} та відповідних e_{i1} і e_{i2} повинні виконуватися по 2 обмеження (3.18) — усього $2(p_1 + p_2 + \ldots + p_n)$ обмежень-нерівностей. Усі змінні e_k — цілочисельні та можуть приймати значення від 1 до m (3.12). Усі змінні v_{ij} — бінарні: вони можуть приймати тільки значення 0 або 1 (3.17).

Метод `minedgecolor`, що міститься в пакеті Graph Theory Toolbox, реалізує цей алгоритм. Для графа `G` він повертає вектор-рядок з номерами фарб його ребер.

Приклад 3.2. Знайти мінімальну правильну розфарбовку ребер графа з прикладу 2.8.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
ec = minedgecolor(G);  % розв'язали задачу
MaxColor = max(ec); % найменша кількість кольорів
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",{},...
  "EdgeLabel",ec,"LineWidth",3,...
  "EdgeColor",cm(round((ec-1)/(MaxColor-1)*255+1),:))
title("Мінімальна кількість фарб = " + MaxColor)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinEdgeColor","-dpng") % зберегли рисунок у файл
```

На рис. 3.4 показаний результат роботи цього алгоритму: мінімальна

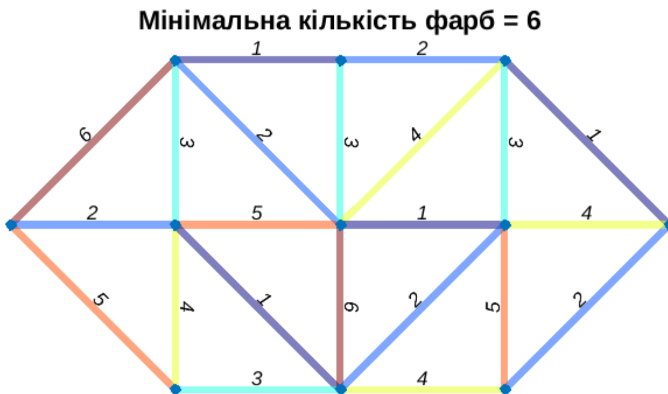


Рис. 3.4. Мінімальна правильна розфарбовка ребер графа

правильна розфарбовка ребер графа шістьма фарбами. □

Слід зауважити, що обмежень (3.18) буде дуже багато, і вони є досить складними для сучасних алгоритмів розв'язання задачі ЦЛП. Тому для графів великих розмірів та потужностей можна скористатися наближеним жадібним алгоритмом послідовного вилучення паросполучень. При цьому теорема Візінга допоможе зрозуміти, чи правильний розв'язок ми отримали. Якщо ребра графа розфарбувалися δ фарбами, то розв'язок точно правильний: хроматичний індекс $\chi'(G)$ не може бути меншим за δ. Якщо для розфарбовки потрібно $\delta + 1$ фарб, то розв'язок, можливо, теж правильний, але не обов'язково. І, нарешті, якщо виявиться, що треба більш ніж $\delta + 1$ фарб, то це точно неправильно, бо протирічить теоремі Візінга. В цьому випадку жадібний алгоритм не підходить.

У Graph Theory Toolbox є метод `minedgecolorapprox`. Він для графа `G` реалізує жадібний алгоритм для знаходження правильної розфарбовки, яка не завжди буде мінімальною. Повертає вектор-рядок з номерами фарб його ребер.

Приклад 3.3. Знайти мінімальну правильну розфарбовку ребер графа з прикладу 3.3 за допомогою жадібного алгоритму.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
```

```
G = graph(s,t); % незважений граф
ec = minedgecolorapprox(G);  % розв’язали задачу
MaxColor = max(ec); % найменша кількість кольорів
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",{},...
  "EdgeLabel",ec,"LineWidth",3,...
  "EdgeColor",cm(round((ec-1)/(MaxColor-1)*255+1),:))
title("Мінімальна кількість фарб = " + MaxColor)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinEdgeColorApprox","-dpng") % зберегли рисунок у файл
```

На рис. 3.5 показаний результат роботи цього алгоритму: правильна розфарбовка ребер графа, яка не є мінімальною: в ній сім фарб замість шести. □

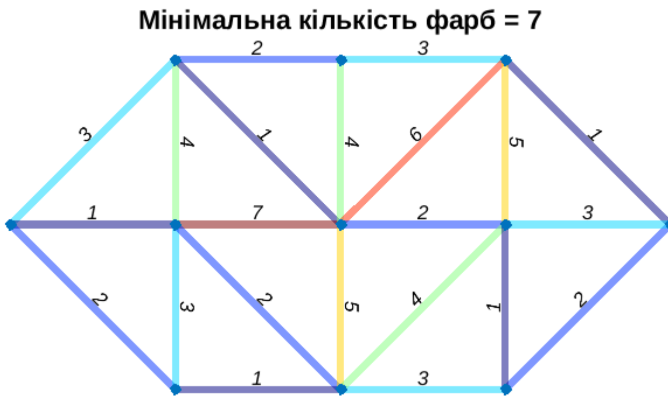


Рис. 3.5. Правильна розфарбовка ребер графа, знайдена за допомогою жадібного алгоритму

3.3. Запитання для перевірки

1. Що таке розфарбовка вершин графа? правильна розфарбовка? мінімальна правильна розфарбовка? хроматричне число?
2. Що таке розфарбовка ребер графа? правильна розфарбовка? мінімальна правильна розфарбовка? хроматричний індекс?
3. Як формулюється теорема Візінга?
4. Чи завжди працює жадібний алгоритм у задачах розфарбовки?
5. Як звести задачі правильної розфарбовки вершин та ребер до задачі ЦЛП?

4. Мінімальні остовні дерева

У цьому розділі буде розглянута задача побудови мінімального остовного дерева (МОД) та доведено, що для її розв'язання можна застосовувати жадібний алгоритм.

4.1. Жадібні алгоритми та матроїди

Означення 4.1. *Жадібний алгоритм* (greedy algorithm) — це алгоритм вибору найкращого варіанту на кожному кроці розв'язання задачі. □

Такий алгоритм не завжди призводить до найкращого (оптимального) розв'язку. Типовий приклад: метод мінімальної вартості для побудови початкового плану перевезень у транспортній задачі. Побудований цим методом план може виявитися не оптимальним, і знадобиться перекидання вантажу за циклом. Інший приклад ми розглянули в попередньому розділі: спроба правильно розфарбувати вершини графа шляхом поступового вилучення максимальних незалежних множин вершин не завжди дає мінімальну правильну розфарбовку. Ось ще приклад: задача про призначення трьох працівників на чотири роботи, або задача про максимальне зважене паросполучення на повному дводолевому графі $K_{3,4}$ з $|V| = 3$; $|W| = 4$; $|E| = 12$. Він показаний на рис. 4.1.

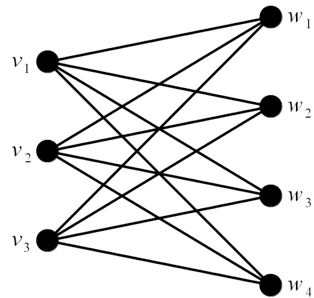


Рис. 4.1. Приклад задачі про призначення

Будемо нумерувати ребра подвійними індексами: перша цифра — номер вершини з V, а друга — з W. Нехай матриця ваг ребер (матриця продуктивностей) має вигляд, показаний в табл. 4.1.

Табл. 4.1. Матриця продуктивностей

$i\backslash j$	1	2	3	4
1	8	7	7	6
2	7	7	5	8
3	8	6	6	7

Скористаємося жадібним алгоритмом. Ми бачимо, що перший працівник найкраще впорається з першою роботою, тому включимо до паросполучення ребро $e_{1,1}$. Другого працівника спрямовуємо на четверту

роботу, оскільки саме на ній його продуктивність є максимальною: додаємо ребро $e_{2,4}$. Тепер у третього працівника залишився невеликий вибір: або друга, або третя роботи з однаковою продуктивністю 6. Оберемо, наприклад, $e_{2,3}$. Отже, побудовано максимальне за включенням зважене паросполучення $\{e_{1,1}, e_{2,4}, e_{3,2}\}$ з загальною вагою 22. Але цей розв'язок — не найкращий. Можна взяти інше паросполучення $\{e_{1,2}, e_{2,4}, e_{3,1}\}$ з загальною вагою 23, та довести, що це дійсно буде максимальне зважене паросполучення. Як бачимо, тут жадібний алгоритм не спрацював: після побудови початкового плану методом максимальної продуктивності знадобилося ще й перекидання працівників з роботи на роботу (як у транспортній задачі ми перекидали вантаж за циклом).

Наступний приклад. Треба знайти максимальне значення лінійної функції n змінних $z = (\boldsymbol{c}, \boldsymbol{x}) \to \max$ на перестановці P_n значень її аргументів $\boldsymbol{x}$. Скажімо, такої:

$$z = 4x_1 - 3x_2 - 6x_3 + 2x_4 \to \max, \tag{4.1}$$

де вектор аргументів (x_1, x_2, x_3, x_4) — одна из перестановок чисел $(2, 3, 4, 5)$. На які місця у векторі $\boldsymbol{x}$ треба поставити числа 2, 3, 4, 5, щоб максимізувати функцію (4.1)? Скористаємося жадібним алгоритмом. Ми бачимо, що коефіцієнт біля x_1 максимальний: він дорівнює 4. Тому надамо змінній x_1 максимальне з наявних значень: 5. Наступний за величиною коефіцієнт — це 2 біля x_4, тому змінній x_4 надамо максимальне з решти значень 4. Далі змінній x_2 надаємо значення 3, а x_3 — 2. Цей процес показаний на рис. 4.2. Отримаємо $z = 7$. Можна перебрати всі $P_4 = 24$ перестановки та переконатися, що дійсно знайдене максимальне значення. Доведення цього факту дуже просте: для кожної пари коефіцієнтів та значень аргументів має місце співвідношення:

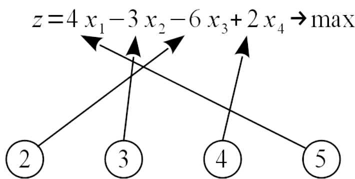


Рис. 4.2. Максимум лінійної функції на перестановці значень її аргументів

$$\begin{aligned} &(c_i > c_k) \cap (x_i > x_k) \Rightarrow c_i x_i + c_k x_k > c_i x_k + c_k x_i : \\ &c_i x_i + c_k x_k - c_i x_k - c_k x_i = c_i (x_i - x_k) - c_k (x_i - x_k) = \\ &= (c_i - c_k)(x_i - x_k) > 0. \end{aligned} \tag{4.2}$$

Чому ж в останньому прикладі жадібний алгоритм привів до успіху, а в попередніх ні? Відповідь на це питання дає структура області допустимих значень. В останньому прикладі, коли на черговому етапі розв'язання задачі ми забирали одне значення з тих, що залишилися, ми забирали тільки його, а інші залишалися недоторканими. У попередньому

ж прикладі, коли на першому етапі ми забрали ребро $e_{1,1}$, то ми забрали насправді не тільки його. Ми позбавили себе можливості на наступних етапах забирати всі ребра, інцидентні до v_1 і w_1. А саме $e_{1,2}$ та $e_{3,1}$, як виявляється, входять в одне з максимальних зважених паросполучень.

Структури, на яких працює жадібний алгоритм, називаються в математиці матроїдами. Існує десь біля півсотні різних означень матроїдів. Класичним вважається означення матроїда через незалежні множини.

Означення 4.2. *Матроїд* (matroid) $M = (S, I)$ — це пара двох множин: S — скінченна множина, що називається *носієм матроїда* (ground set), а I — деяка множина підмножин елементів S, що називається *сімейством незалежних множин* (independent sets), тобто $I \subseteq 2^S$. При цьому для незалежних множин, що входять в I, повинні виконуватися такі аксіоми.

1. Серед елементів I повинна бути пуста підмножина: $\emptyset \in I$;
2. Якщо якась множина A елементів S належить до I, то й будь-яка її власна підмножина також повинна належати до I: $(\forall A \in I) \cap (\forall B \subset A) : \ B \in I$;
3. Якщо є дві множини A і B елементів S, що належить до I, і при цьому в A більше елементів, ніж у B, то серед елементів A, яких немає у B, повинен знайтися хоча б один такий елемент x, що його об'єднання з B також буде належати до I: $(\forall A, B \in I) \cap (|A| > |B|) : \exists x \in (A \setminus B) : B \cup \{x\} \in I$. □

У прикладі з лінійною функцією носієм матроїда буде множина усіх можливих упорядкованих пар (c_i, x_k), де c_i — це коефіцієнти функції (4.1), тобто одне з чисел 4, −3, −6, 2, а x_k — можливі значення аргументів, тобто одне з чисел 2, 3, 4, 5. Усього в носії буде $|S| = 4 \times 4 = 16$ елементів:

$$
\begin{aligned}
S = \{&(c_1, x_1), (c_1, x_2), (c_1, x_3), (c_1, x_4), \\
&(c_2, x_1), (c_2, x_2), (c_2, x_3), (c_2, x_4), \\
&(c_3, x_1), (c_3, x_2), (c_3, x_3), (c_3, x_4), \\
&(c_4, x_1), (c_4, x_2), (c_4, x_3), (c_4, x_4)\}.
\end{aligned}
\tag{4.3}
$$

Сімейство незалежних множин I створимо з елементів S таким чином: з кожного рядка формули (4.3) оберемо не більше одного елемента, і їхня сукупність (вона містить не більше чотирьох елементів S), і буде одним з елементів множини I. Наприклад, можна з першого рядка (4.3) обрати елемент (c_1, x_3), з другого — (c_2, x_2), з третього — нічого, з четвертого — (c_4, x_2). Тоді підмножина елементів S: $((c_1, x_3), (c_2, x_2), (c_4, x_2))$ буде одним з елементів I. Також для виконання умови 1 з означення 4.2 додамо до I пусту множину $\emptyset$, тобто з кожного рядка (4.3) не обираємо жодного елемента. Матроїд, який ми побудували, називається *матроїдом розбиттів* (partition matroid). В загальному випадку для отримання матроїда розбиттів треба здійснити розбиття множини S на деякі непусті

підмножини, та обрати з кожної підмножини не більше заданої кількості елементів. Ми розбили S на чотири підмножини з різними c_i, та обрали з кожної підмножини не більше одного елемента.

Неважко довести, що отримана структура $M = (S, I)$ дійсно є матроїдом, тобто що для неї виконуються всі три умови з означення 4.2. Зокрема, аксіома 3 доводиться шляхом порівняння c_i у множинах A і B. Оскільки в B менше елементів, ніж в A, то у множині $A \setminus B$ напевно буде елемент з таким c_i, якого немає в B. Саме його й долучаємо до B, і тоді елемент $B \cup \{x\} \in I$.

Означення 4.3. *Базою (базисом) матроїда* (base, basis of matroid) називається будь-який максимальний за включенням елемент множини I. □

Якщо додати до бази ще будь-який елемент з S, ми ніколи не отримаємо незалежної множини. Ми отримаємо множину, яка називається *залежною* (dependent set), вона до I не входить. Будемо позначати бази матроїда $M = (S, I)$ як $B_1, B_2, \ldots$, а сукупність усіх баз $\mathcal{B}$. Оскільки бази — це теж незалежні множини, то $\mathcal{B} \subset I$. Для баз матроїда мають місце такі властивості.

Властивість 4.1. Множина $\mathcal{B}$ є непустою. □

Доведення. Сімейство незалежних множин I є непустим: за першою аксіомою в означенні матроїда 4.2 до нього входить принаймні пуста множина $\varnothing$. Тому серед незалежних множин напевно є множини з максимальною за включенням кількістю елементів. □

Властивість 4.2. Якщо $B_1, B_2 \in \mathcal{B}$, і $B_1 \neq B_2$, то $B_1 \not\subseteq B_2$ і $B_2 \not\subseteq B_1$. □

Доведення очевидне: бази є максимальними за включенням і, якщо вони різні, то не можуть бути підмножинами одна одної. □

Властивість 4.3. Усі бази мають однакову потужність (кількість елементів): $|B_1| = |B_2| = \ldots$. □

Доведення цієї властивості будується від протилежного. Припустимо, що $|B_2| < |B_1|$. Тоді за третьою аксіомою в означенні 4.2: $\exists x \in (B_1 \setminus B_2)$ такий, що $B_2 \cup \{x\} \in I$. А це означає, що B_2 не є максимальною за включенням, і, отже, не є базою. □

Властивість 4.4. Якщо $B_1, B_2 \in \mathcal{B}$, то $\forall x \in B_1 : \exists y \in B_2$ такий, що $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. □

Доведення. За другою аксіомою матроїда 4.2 маємо: $\forall x \in B_1 : (B_1 \setminus \{x\}) \in I$. Звідсіля за властивістю рівнопотужності баз 4.3: $|B_2| > |B_1 \setminus \{x\}|$. Тоді за третьою аксіомою матроїда 4.2: $\exists y \in B_2$ такий, що $(B_1 \setminus \{x\}) \cup \{y\} \in I$. А оскільки $|(B_1 \setminus \{x\}) \cup \{y\}| = |B_1|$, і B_1 — база, то $(B_1 \setminus \{x\}) \cup \{y\}$ теж є базою: $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$, що й треба було

довести. □

Властивість 4.5. Якщо є дві бази: $B_1, B_2 \in \mathcal{B}$, то $\forall x \in (B_1 \setminus B_2)$: $\exists y \in (B_2 \setminus B_1)$ такий, що $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. □

Для доведення цієї властивості потрібні поняття про замикання незалежної множини, яке ми не вивчали. Тому дамо лише пояснення до цієї властивості. Якщо взяти будь-який елемент x, який є у базі B_1, але якого немає у базі B_2, то для нього обов'язково знайдеться елемент y з бази B_2, якого немає у базі B_1, такий, що вилучення x з бази B_1 з наступним приєднанням y утворює якусь базу. Ця властивість називається властивістю обміну елементами між базами.

Означення 4.4. *Рангом матроїда* (rank of matroid) називається потужність будь-якої його бази. □

Побудований нами на множині (4.4) матроїд розбиттів має ранг 4, бо кожна з його баз містить 4 елементи — по одному з кожного рядка формули (4.4). Усього наш матроїд має $|B| = 4^4 = 256$ баз.

Загальна постановка задачі є такою. Нехай задана скінченна множина S та деяке сімейство I її підмножин. Наприклад, це може бути носій і сімейство незалежних множин матроїда. Для кожного елемента $x \in S$ задана його вага: якась функція $w(x)$. Вага множини $X \subseteq S$ визначається як сума ваг усіх елементів X. Треба знайти множину A найбільшої ваги, що належить I.

Жадібний алгоритм для розв'язання цієї задачі полягає в сортуванні та послідовному відборі елементів з S (СПВ). Ось його схема.

```
Сортуємо елементи S=(x1,x2,...,xm) у порядку спадання ваг:
w(x1)>=w(x2)>=...>=w(xm).
A=(пуста множина)
for k=1 step 1 to m do {переглядаємо елементи множини S}
begin {for k}
  if (A or xk) in I {додавання нового елементу залишає нас в I}
  begin {if}
    A=(A or xk) {додаємо цей елемент}
  end {if}
end {for k}
```

У нашому прикладі вага кожного елемента — це добуток c_i на x_k:

$$w(c_i, x_k) = c_i x_k. \tag{4.4}$$

Всі елементи носія S (4.3) мають такі ваги:

$$
\begin{aligned}
S = \{&8, 12, 16, 20, \\
&-6, -9, -12, -15, \\
&-12, -18, -24, -30, \\
&4, 6, 8, 10\}\,.
\end{aligned}
\tag{4.5}
$$

Для розв'язання задачі (4.1) нам треба знайти таку базу, в якій усі x_k різні, та яка має максимальну вагу. Застосуємо жадібний алгоритм. Сортуємо рядки в порядку зростання значень c_i:

$$
\begin{aligned}
&c_1 = 4 : 8, 12, 16, 20, \\
&c_4 = 2 : 4, 6, 8, 10, \\
&c_2 = -3 : -6, -9, -12, -15, \\
&c_3 = -6 : -12, -18, -24, -30.
\end{aligned}
\tag{4.6}
$$

З першого рядка обираємо найбільше значення: $w_{1,4} = 20$. З другого рядка обираємо теж найбільше значення, але з іншого стовпця, тобто при іншому значенні x_k: $w_{4,3} = 8$. Наступні c_i від'ємні, тому з третього рядка обираємо *найменше* значення серед стовпців для x_1, x_2: це $w_{2,2} = -9$. І, нарешті, в четвертому рядку залишається $w_{3,1} = -12$. Отримали для цільової функції: $z_{\max} = 20 + 8 - 9 - 12 = 7$, і це значення досягається на перестановці аргументів $(x_4, x_2, x_1, x_3) = (5, 3, 2, 4)$.

Отже, в задачі про максимум лінійної функції на перестановці значень її аргументів жадібний алгоритм спрацював, а в попередніх прикладах — ні. Чи є випадковим цей факт? Виявляється, ні. Одна з основних теорем теорії матроїдів стверджує наступне.

Теорема 4.1. Теорема Радо-Едмондса. Нехай на елементах x носія S матроїда $M = (S, I)$ задана вагова функція $w(x)$. Нехай також $A \in I$ — множина максимальної ваги серед незалежних множин X потужності k. Серед усіх елементів x, що $x \notin A$, але таких, що $A \cup \{x\} \in I$, оберемо елемент з максимальною вагою. Тоді $A \cup \{x\}$ буде множиною максимальної ваги серед усіх незалежних множин X потужності $k + 1$. □

Доведення. Розглянемо $B \in I$ — множину максимальної ваги серед незалежних множин X потужності $k + 1$. З третьої аксіоми матроїда 4.2 маємо, що $\exists y \in (B \setminus A) : \ A \cup \{y\} \in I$. Тоді мають місце такі нерівності:

$w(A \cup \{y\}) = w(A) + w(y) \le w(B) \ \Rightarrow \ w(A) \le w(B) - w(y)\,;$

$w(B \setminus \{y\}) = w(B) - w(y) \le w(A) \ \Rightarrow \ w(A) \ge w(B) - w(y)\,.$

Оскільки ми обмежили $w(A)$ з обох боків однією й тією самою величиною $w(B) - w(y)$, то ці величини однакові: $w(A) = w(B) - w(y)$. Звідсіля маємо: $w(A \cup \{y\}) = w(A) + w(y) = w(B)$. Отже, якщо об'єднати множину A з x — елементом максимальної ваги з тих, що $A \cup \{x\} \in I$, то отримаємо множину максимальної ваги серед незалежних множин X потужності $k + 1$. □

Якщо застосовувати цю теорему поступово, поки не дійдемо до бази, то й отримаємо базу максимальної ваги, що свідчить про правильність жадібного алгоритму для матроїда.

Зрозуміло, що так само можна будувати й базу мінімальної ваги: треба додавати на кожному кроці елемент з мінімальною вагою. Як бачимо, досить складна теорія матроїдів пояснює, чому формула (4.3) має місце. Більш докладну інформацію про матроїди можна знайти в інтернеті. А ми розглянемо одну важливу задачу теорії графів, яка теж є задачею на матроїді, і для її розв'язання можна буде використовувати жадібний алгоритм.

4.2. Мінімальне остовне дерево (МОД)

Будемо розглядати графи та мультиграфи, але не псевдографи. Тобто допускаються кратні ребра, але не петлі.

Означення 4.5. *Шлях, або маршрут* (path) у графі $G = (V, E)$ — це послідовність вершин та ребер виду $v_1 e_1 v_2 e_2 \dots v_k$, у якій сусідні елементи є інцидентними. □

Означення 4.6. Шлях називається *простим* (simple path), якщо кожна вершина зустрічається в ньому лише один раз. □

У простому шляху немає перетинів (вершин, що повторюються) і, як наслідок, не може бути повторюваних ребер. Протилежне твердження не має місця: ребра у шляху можуть бути унікальними, але вершини повторюватися у точках перетину; і такий шлях вже не простий.

Означення 4.7. Граф $G = (V, E)$ називається *зв'язним* (connected graph), якщо існує шлях із будь-якої його вершини у будь-яку іншу. □

Досі всі приклади, що ми розглядали — це були зв'язні графи. У незв'язних графах можна виділити окремі зв'язні компоненти. У крайньому випадку, коли у графа взагалі немає ребер: $E = \emptyset$, у нього буде n компонент — по одній вершині у кожній.

Розглянемо зв'язний граф (або мультиграф) G. Якщо з нього видалити деякі ребра, він може залишится зв'язним, а може й розпастися на окремі компоненти. Скільки (за максимумом) ребер та які з них можна видалити, щоб граф залишився зв'язним? Якщо ребра графа зважені, то можна поставити задачу так: як видалити максимальну кількість ребер максимальної ваги, щоб залишився зв'язний граф із загальною мінімальною вагою ребер?

Спочатку розглянемо простішу, незважену задачу: яка мінімальна кількість ребер необхідна для зв'язності графа з n вершинами?

Теорема 4.2. У зв'язному графі $G = (V, E)$ виконується: $m \geq n - 1$. Тобто мінімально можлива кількість ребер зв'язного графа є $n - 1$. □

Доведення. Коли $n = 1$, казати про зв'язність взагалі немає сенсу: одну вершину нема з чим поєднувати. Можна сказати, що одна вершина пов'язана сама з собою нульовою кількістю ребер. Дві вершини можна поєднати одним ребром, і граф стане зв'язним. Третю вершину можна під'єднати за допомогою ще одного, вже другого ребра. За індукцією: нехай теорема має місце для $n = k$, тобто граф з k вершинами та $k - 1$ ребрами зв'язний. Наступну, $(k + 1)$-у вершину можна приєднати до нього за допомогою ще одного, k-го ребра. Отриманий граф з $k + 1$ вершинами та k ребрами теж буде зв'язним. За індукцією теорема доведена. □

Ця теорема показує, скільки (мінімум) ребер треба залишити у зв'язному графі з n вершинами, щоб він залишився зв'язним: $n - 1$.

Означення 4.8. Зв'язний граф $G = (V, E)$ з n вершинами та $n - 1$ ребрами називається *остовним деревом* (spanning tree). □

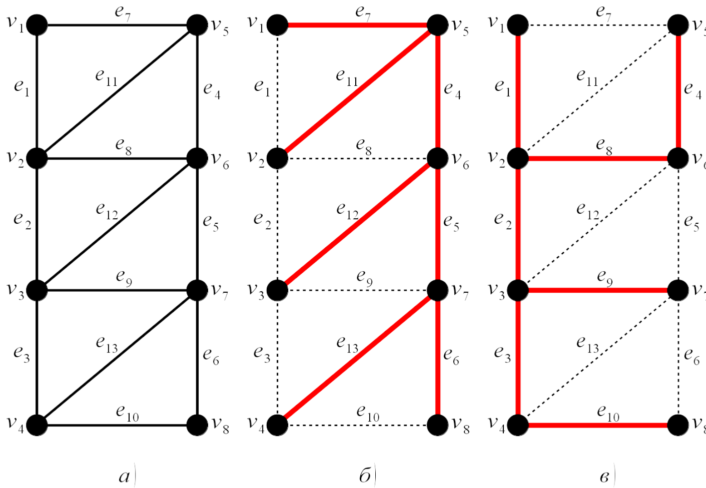


Рис. 4.3. Граф (а) та деякі з його остовних дерев (б, в)

На рис. 4.3 показаний граф (*а*) та деякі з його остовних дерев (*б*, *в*).

Якщо зв'язний граф $G = (V, E)$ не є остовним деревом, то, вочевидь, з нього можна видалити деякі ребра так, щоб ті ребра, що залишилися $E_1 \subset E$, разом з множиною вершин V утворювали б остовне дерево $G_1 = (V, E_1)$. Остовне дерево має цікаві властивості, які ми зараз розглянемо. Але спочатку ще деякі означення.

Означення 4.9. Шлях $v_1 e_1 v_2 e_2 \ldots v_k$ називається *циклом* (cycle), якщо в ньому початкова та кінцева вершини співпадають. □

Означення 4.10. Цикл називається *простим* (simple cycle), якщо

у шляху, що його визначає, кожна проміжна вершина зустрічається лише один раз. □

Простий цикл — це цикл без самоперетинів. Якщо першу та останню вершини вважати за одну, то в ньому немає вершин, що повторюються, а, отже, немає й повторюваних ребер. Вочевидь, з будь-якого непростого циклу можна виділити принаймні два різних простих цикли, а з непростого шляху — принаймні один простий цикл.

Тепер розглянемо властивості остовних дерев.

Властивість 4.6. В остовному дереві $G_1 = (V, E_1)$ зв'язного графа $G = (V, E)$ немає циклів. □

Доведення. Від протилежного: якщо у G_1 є хоча б один цикл, то можна без порушення зв'язності видалити одне з ребер цього циклу: шлях з будь-якої вершини до будь-якої іншої можна організувати вздовж тієї частини циклу, що залишилася. Значить, у G_1 більше, ніж $n-1$ ребер: видалення з нього одного ребра не порушує зв'язності. Отже, G_1 не є остовним деревом. □

Властивість 4.7. Навпаки, якщо у зв'язному графі немає циклів, він є остовним деревом. □

Доведення. Теж від протилежного: нехай зв'язний граф $G = (V, E)$ не є остовним деревом: $m > n - 1$. Оскільки граф зв'язний, то серед m ребер точно є такі $(n-1)$ ребер, що утворюють остовне дерево. Тоді кожне ребро з решти додає інший шлях між якоюсь парою вершин, тобто утворює цикл. □

Властивість 4.8. В остовному дереві G_1 зв'язного графа існує лише один шлях з кожної вершини в кожну іншу. □

Доведення. Від протилежного: якщо б із деякої вершини v_i існували б два різних шляхи у v_j, то існував би цикл $v_i \dots v_j \dots v_i$, що протирічить властивості 4.7. □

Властивість 4.9. Навпаки, якщо у зв'язному графі $G = (V, E)$ існує лише один шлях між кожною парою вершин, то такий граф є остовним деревом. □

Доведення. Від протилежного: якщо зв'язний граф не є остовним деревом, то за властивістю 4.7 у ньому є хоча б один цикл, а значить, існує більш ніж один шлях між якоюсь парою вершин. □

За допомогою властивостей 4.6-4.9 ми встановили еквівалентність тверджень:

- остовне дерево — це зв'язний граф з $|E| = |V| - 1$;
- остовне дерево — це зв'язний граф без циклів;
- остовне дерево – це зв'язний граф, у якому існує лише один шлях з будь-якої вершини у будь-яку іншу.

Зазвичай у застосуваннях стає задача знаходження остовного дерева мінімальної ваги (в подальшому — МОД, мінімальне остовне дерево). Така задача виникає, наприклад, під час проектування ліній електромереж: для прокладання ліній обирають ділянки з мінімальною вартістю робіт, що забезпечують зв'язність мережі.

Дослідимо можливості застосування жадібного алгоритму для знаходження МОД. Ми покажемо, що знаходження МОД — це задача на матроїді, тобто жадібний алгоритм тут дійсно працює. Для цього введемо ще деякі означення.

Означення 4.11. *Деревом* (tree) називається остовне дерево, побудоване на деякій підмножині вершин $V_1 \subset V$ та інцидентних до всіх них ребрах. □

Означення 4.12. *Лісом* (forest) називається множина дерев, що не перетинаються. □

На рис. 4.4 показаний приклад лісу, утвореного двома деревами, що містять ребра, та ізольованими вершинами, які теж вважаються деревами. У кожному дереві з n_1 вершин міститься $n_1 - 1$ ребер.

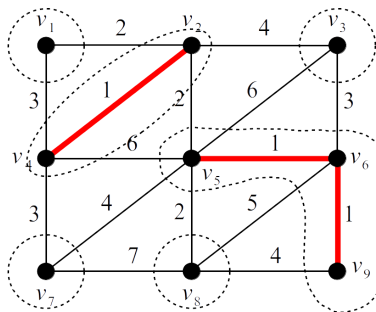


Рис. 4.4. Ліс та його дерева

Поглянемо на проблему побудови МОД як на задачу на матроїді. За носій матроїда S (див. означення 4.2) візьмемо множину усіх ребер графа. Розглянемо будь-яку підмножину ребер, у якій немає циклів, тобто будь-яке дерево або ліс, включно з пустою множиною. Перевіримо систему таких підмножин на незалежність згідно означення 4.2. Пусту множину ми включили до системи, тому аксіома 1 виконується. Якщо з дерева або лісу вилучити одне або кілька ребер, то циклів не утвориться. Залишиться дерево або ліс, який теж є у нашій системі підмножин. Отже, аксіома 2 теж виконується. Залишилося довести виконання аксіоми 3. Сформулюємо її у вигляді теореми.

Теорема 4.3. Нехай у графі G є два дерева або ліси A та B, причому в A більше ребер, ніж у B: $|A| > |B|$. Тоді серед ребер, які є в A, але яких немає у B, є таке ребро, що долучення його до B залишає B деревом або лісом, тобто не утворює у ньому циклів. □

Доведення. Зауважимо спочатку, що, оскільки $|A| > |B|$, то у меншому дереві (лісі) B буде менше, ніж $n - 1$ ребер. Отже, B не може бути остовним деревом і, т. ч., має більше однієї компоненти зв'язності (ізольо-

вана вершина — це теж компонента зв'язності). Чи існує серед ребер A таке ребро (якого немає у B), яке поєднує дві якісь компоненти зв'язності у B? Припустимо, що ні. Якщо це так, то будь-яка компонента зв'язності A цілком (усіма своїми вершинами, але не обов'язково ребрами) входить до якоїсь компоненти зв'язності B. Якщо у цій компоненті зв'язності B було k вершин і, відповідно, $k-1$ ребер (це дерево, циклів немає), то й у відповідній компоненті зв'язності A буде не більше вершин і, відповідно, не більше ребер. Додавши ці результати за усіма компонентами зв'язності B, дійдемо до висновку, що у дереві (лісі) A не більше ребер, ніж у B, що протирічить умові. Отже, наше припущення було хибним, і насправді серед ребер A є хоча б одне ребро, що поєднує компоненти зв'язності у B. Це ребро не утворює циклів у B, залишаючи його деревом (лісом). □

Висновок із теореми 4.3: множина усіх дерев та лісів графа є незалежною множиною матроїда, побудованого на носії — множині його ребер. Цей матроїд так і називається: графовий матроїд.

Означення 4.13. *Графовий матроїд* (graphic matroid) — це матроїд, побудований на носії S — множині усіх ребер графа з незалежними множинами — деревами та лісами. □

Залишилося ввести вагову функцію на ребрах, і можна застосовувати жадібний алгоритм. У нашому випадку вагова функція — це просто вага ребра. І для побудови МОД можна застосовувати жадібний алгоритм, додаючи поступово ребра мінімальної ваги.

Розглянемо два класичні алгоритми побудови МОД. В їх описах будемо позначати ребро, що поєднує вершини v_i та v_j, як $e_{i,j}$, тобто двома індексами поєднуваних вершин.

Алгоритм Пріма (Prim's algorithm). Він показаний на рис. 4.5. Біля вершин проставлені їхні номери, а біля ребер – ваги. У нашому графі $n = 9$, $m = 16$, тому в побудованому МОД повинно залишитися лише 8 ребер з 16. Остовне дерево повинно поє'днувати всі вершини, у т. ч. й v_1. Тому візьмемо перше ребро мінімальної ваги, що виходить з вершини v_1. Таким ребром буде $e_{1,2}$: його вага 2 менша за вагу 3 ребра $e_{1,4}$. Отже, починаємо побудову МОД з нього (*а*). Далі переглядаємо всі ребра, суміжні до вже побудованого фрагменту МОД, тобто до $e_{1,2}$. Всього таких ребер 4 (одне у вершині v_1 та три у v_2). Обираємо з них ребро мінімальної ваги. Мінімальна вага серед цих чотирьох ребер — 1, вона у ребра $e_{2,4}$. Його й приєднуємо до МОД, що будується (*б*). Продовжуємо переглядати ребра, суміжні до вже побудованого дерева (фрагменту МОД) і такі, що не утворюють у ньому циклів. Беремо всі ребра, інцидентні до v_1, v_2 та v_4 (крім $e_{1,4}$ — його брати не можна, бо утворюється цикл), та обираємо з них ребро мінімальної ваги. Мінімальна вага 2 — у ребра $e_{2,5}$, його й приєднуємо (*в*). Продовжуємо процес. Тепер у нас з'явилась можливість долучити до МОД ребро $e_{5,6}$ вагою 1 (*г*), а потім — $e_{6,9}$ теж з вагою 1

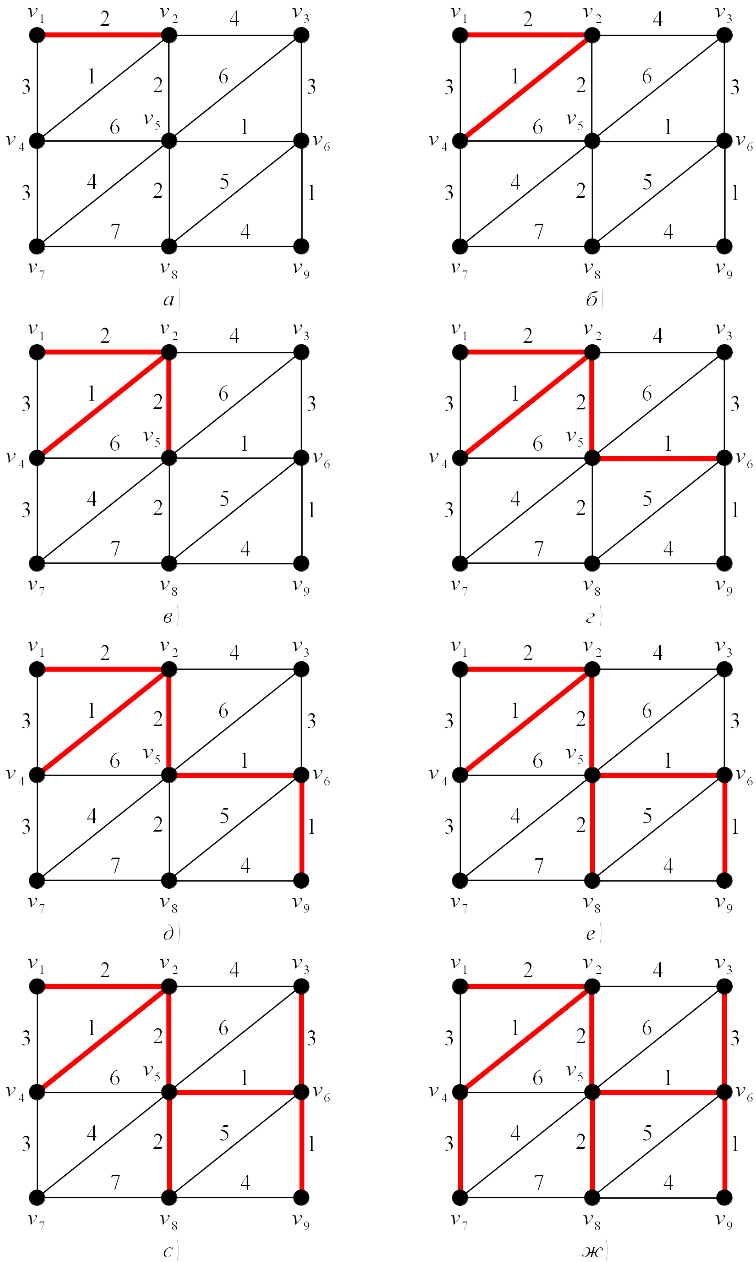


Рис. 4.5. Алгоритм Пріма

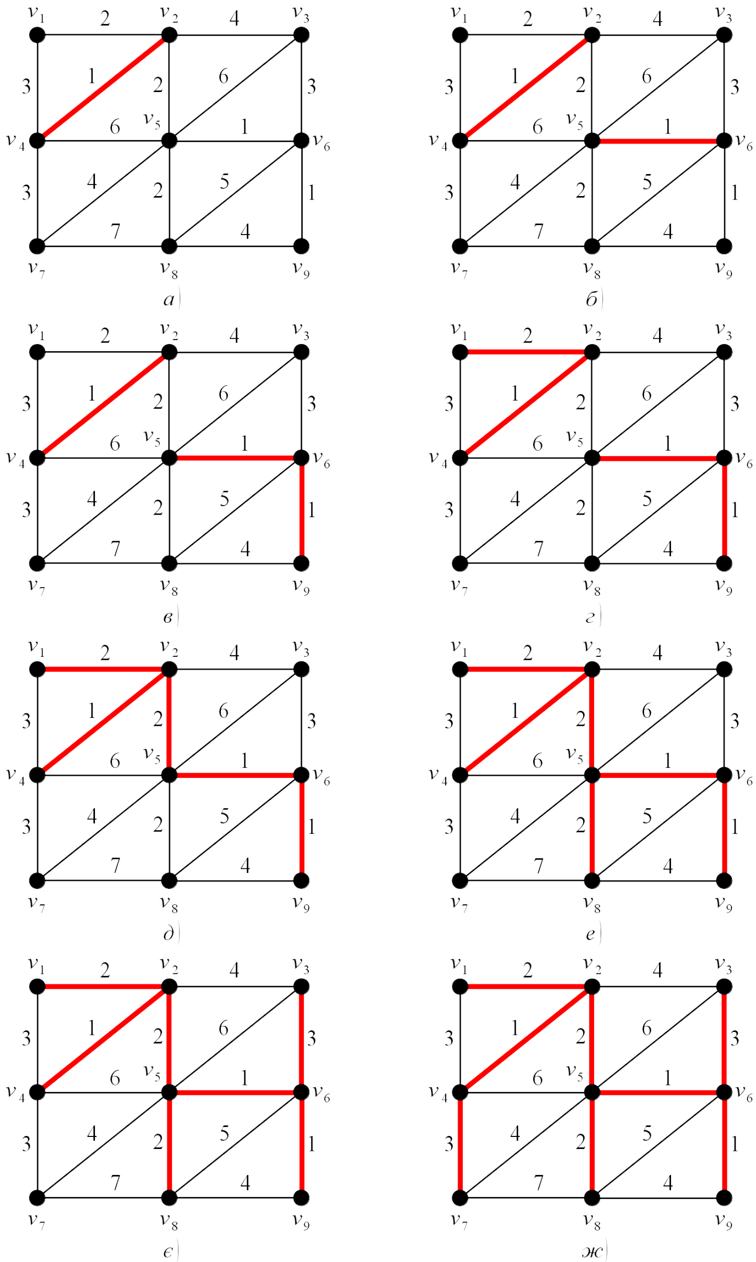


Рис. 4.6. Алгоритм Краскала

(*д*). Серед ребер, що залишилися, мінімальна вага 2 — у ребра $e_{5,8}$, його також можна долучити до МОД (*е*). І, нарешті, приєднуємо ребра $e_{3,6}$ та $e_{4,7}$ ваги 3 (*є*, *ж*). Отримали 8 ребер — МОД побудоване. Його вага 15.

Алгоритм Краскала (Kruskal's algorithm). Цей алгоритм показаний на рис. 4.6. Основна його відмінність від алгоритму Пріма — ми будемо додавати в МОД, що будується, не обов'язково суміжні ребра. Головне, щоб не утворювалися цикли. Переглядаємо всі ребра мінімальної ваги 1. Починаємо з $e_{2,4}$ — воно зустрілося першим (*а*). Наступне ребро ваги 1 — це $e_{5,6}$, і воно не утворює циклів — додаємо його (*б*). Далі перевіряємо $e_{6,9}$: циклів немає, долучаємо його (*в*). Всі ребра з вагою 1 вичерпані. Переходимо до ребер з наступною мінімальною вагою 2. Ребро $e_{1,2}$ можна приєднати (*г*), $e_{2,5}$ — також (*д*), $e_{5,8}$ теж підходить (*е*). Далі — ребра з вагою 3. Ребро $e_{1,4}$ не годиться (утворюється цикл), а ось $e_{3,6}$ підходить (*є*), і $e_{4,7}$ — також (*ж*). В результаті отримали те же саме МОД ваги 15, що й за алгоритмом Пріма.

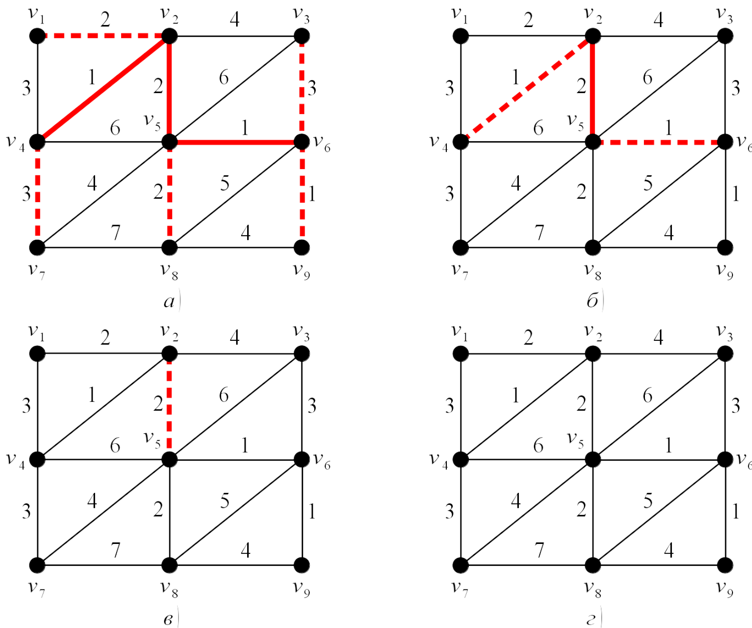


Рис. 4.7. Перевірка на відсутність циклів (немає циклів)

Як бачимо, ці алгоритми відрізняються порядком долучення ребер. В алгоритмі Пріма ми весь час приєднували суміжні ребра, тобто нарощували МОД, залишаючи його зв'язним. В алгоритмі Краскала будуються окремі частини МОД (дерéва), які потім поєднуються. Алгоритм Краска-

ла — це класична схема жадібного алгоритму: долучаємо ребра у порядку зростання ваги, якщо вони не утворюють циклу, тобто залишають нас у множині дерев та лісів. У алгоритмі Пріма є додаткова умова для долучення ребер: їхня суміжність з тим деревом, що вже побудоване. На мою думку, алгоритм Пріма більш придатний для розв'язання задачі вручну, а алгоритм Краскала — для програмування.

При програмуванні алгоритмів Пріма та Краскала треба на кожному кроці перевіряти створену підмножину ребер на незалежність, тобто на наявність або відсутність циклів. Ця перевірка може здійснюватися, наприклад, за таким алгоритмом.

1. Обчислюємо у побудованому дереві (лісі) ступені всіх вершин. Для цього можна, наприклад, додати всі стовпці або рядки матриці суміжності вершин, або додати всі стовпці матриці інцидентності.
2. Перевіряємо, чи є вершини ступеня 1 (висячі вершини).
3. Якщо такі вершини є, вилучаємо всі інцидентні до них ребра, і йдемо на крок 1. Якщо ж таких вершин немає, то йдемо на крок 4.
4. Перевіряємо: якщо залишилися вершини тільки ступеня 0 — циклів немає. Якщо ж залишилися вершини ступеня 2 або більше, то є цикли.

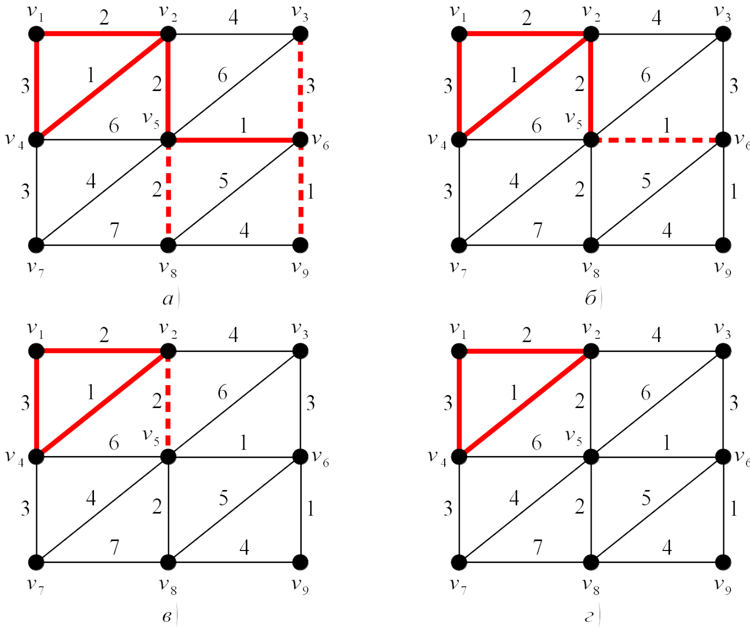


Рис. 4.8. Перевірка на відсутність циклів (є цикли)

Ось як виглядає алгоритм перевірки на наявність циклів для ребер з рис. 4.6, *ж*. На рис. 4.7, *а* показані ці ребра. Висячі вершини: v_1, v_3, v_7, v_8, v_9. Інцидентні до них ребра відмічені на рис. 4.7, *а* червоними штриховими лініями. Вилучаємо їх (рис. 4.7, *б*). Перевіряємо знову: тут висячі вершини v_4, v_6, інцидентні до них ребра теж позначені червоними штриховими лініями. Після їхнього вилучення маємо рис. 4.7, *в*, а ще після одного кроку — пустий граф на рис. 4.7, *г*. Ребер немає — дійсно мали дерево, циклів не було.

Якщо б на останньому кроці алгоритму Краскала (рис. 4.6, *є*) ми б додали замість ребра $e_{4,7}$ ребро $e_{1,4}$, то перевірка на наявність циклу показана на рис. 4.8. Кроки вилучення ребер, інцидентних до висячих вершин, такі ж самі. На останньому етапі немає вершин ступеня 1, але є три вершини ступеня 2, що свідчить про наявність циклу. Отже, ребро $e_{1,4}$ не можна включати до МОД, що будується.

У MATLAB є метод (вбудована функція) `minspantree`, який для графа `G` будує мінімальне остовне дерево та повертає його у вигляді графа `T`. Якщо ваги ребер задані, знаходиться мінімальне остовне дерево, а якщо ні, то всі ваги вважаються одиничними, і знаходиться перше остовне дерево, що зустрінеться. Необов'язкові вхідні параметри у вигляді пари `Name,Value` визначають додаткові параметри: алгоритм (Пріма чи Краскала), початкову вершину в алгоритмі Пріма, можливість побудови лісу для незв'язного графа. Крім самого остовного дерева `T`, можна повернути вектор-рядок довжини n з номерами вузлів-попередників для кожного вузла при побудові остовного дерева пошуком у ширину.

Приклад 4.1. Для графа з прикладу 2.6 знайти незважене та зважене мінімальні остовні дерева.

```
s = [2 2 2 1 3 1 1 3 3 4 5 6 5 6 6 7 7 ...
  8 11 8 11 10]; % початки ребер
t = [1 3 4 3 4 5 6 6 7 7 6 7 8 8 11 11 ...
  10 11 10 9 9 9]; % кінці ребер
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 5]; % ваги
x = [1 0 1 1 2 2 2 3 4 3 3]; % координати вершин
y = [1 0 0 -1 1 0 -1 1 0 -1 0];
G = graph(s,t); % незважений граф
T = minspantree(G); % незважена задача
figure % нове вікно фігури
p1 = plot(G,"XData",x,"YData",y); % нарисували граф
highlight(p1,T); % додали остовне дерево
axis("equal") % однаковий масштаб уздовж осей координат
axis('off') % прибрали осі
```

```
title("Остовне дерево, кількість ребер = " + numedges(T))
print("SpanTree","-dpng") % зберегли рисунок у файл
G = graph(s,t,w); % створили зважений граф
T = minspantree(G); % зважена задача
figure % нове вікно фігури
p2 = plot(G,"XData",x,"YData",y,...
  "EdgeLabel",G.Edges.Weight); % нарисували граф
highlight(p2,T); % додали мінімальне остовне дерево
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
title("Мінімальне остовне дерево, вага = " + ...
  sum(T.Edges.Weight)) % заголовок
print("MinSpanTree","-dpng") % зберегли рисунок у файл
```

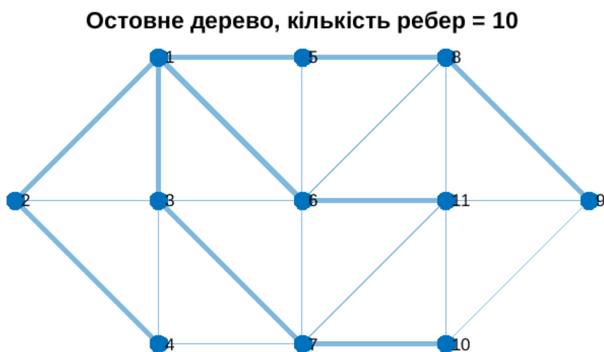


Рис. 4.9. Остовне дерево

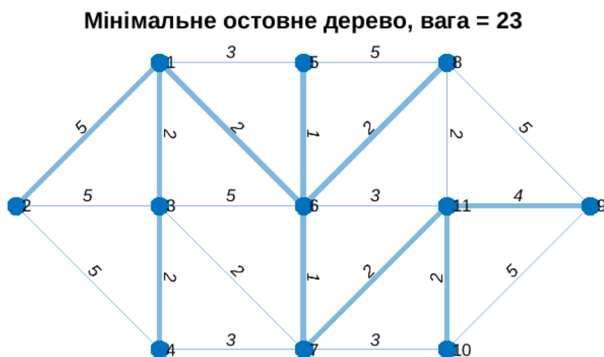


Рис. 4.10. Мінімальне остовне дерево

На рис. 4.9 показаний розв'язок незваженої задачі, а на рис. 4.10 — зваженої. У незваженій задачі ми не задавали початкову вершину для побудови остовного дерева, тому за умовчанням була взята v_1, і проводився пошук "у ширину". Мітки ребер у зваженій задачі — ваги ребер. □

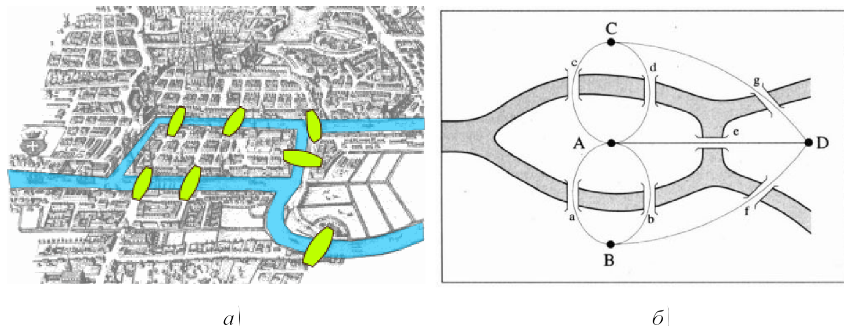
4.3. Запитання для перевірки

1. Що таке матроїд? носій матроїда? незалежна множина матроїда? залежна множина матроїда? база матроїда?
2. Що таке мінімальне остовне дерево? дерево? ліс?
3. Як працює алгоритм Пріма?
4. Як працює алгоритм Краскала?
5. Чому для побудови мінімального остовного дерева можна застосовувати жадібний алгоритм?
6. Як будується мінімальне остовне дерево в MATLAB?

5. Цикли та коцикли

У цьому розділі будуть розглянуті ейлерові та гамільтонові цикли, незалежні системи циклів (базис у комбінаторному просторі циклів) і незалежні системи коциклів (розрізів). Буде показано, що дві останні задачі — це задачі на матроїді, і для пошуку незалежних систем можна використовувати МОД. Для ейлерових та гамільтонових циклів будуть наведені деякі теореми.

5.1. Ейлерові цикли



а) *б*)

Рис. 5.1. Мости Кьоніґсберга

Місто Кьоніґсберг (Königsberg) розташовано на берегах річки Праґоля (Pregel River) та на двох островах. У класичній задачі про сім мостів Кьоніґсберга, поставленій Леонардом Ейлером (Leonhard Euler) у 1735 році (рис. 5.1), треба обійти усі сім мостів по одному разу кожен. З точки зору теорії графів ми тут маємо мультиграф з $n = 4$, $m = 7$, і треба побудувати цикл, що проходить один і тільки один раз через кожне ребро.

Означення 5.1. Цикл у зв'язному графі чи мультиграфі, що проходить один і тільки один раз через кожне ребро, називається *ейлеровим* (Eulerian cycle). Граф (мультиграф), у якому існує ейлерів цикл, називається *ейлеровим* (Eulerian multigraph). □

Зауважимо, що додавання петель до мультиграфа не змінює його ейлеровість. Дійсно: якщо граф є ейлеровим, то під час обходу ейлерового циклу, коли ми опинилися у вершині, де є петля чи кілька петель, можна їх обійти та продовжити рух уздовж ейлерового циклу. Але кратні ребра суттєво впливають на ейлеровість. Додавання чи вилучення паралельного ребра може надати мультиграфові ейлеровості або, навпаки, позбавити. Тому будемо розв'язувати задачу про ейлерів цикл для мультиграфів, але не для псевдографів.

Далі, суттєвим для ейлеровості є зв'язність мультиграфа. Якщо у мультиграфа є кілька компонент зв'язності, то побудувати ейлерів цикл неможливо: ми ніколи не обійдемо усі ребра, оскільки не зможемо перейти з однієї компоненти зв'язності до іншої. Тому далі розглядатимемо лише зв'язні мультиграфи.

Інколи у мультиграфі побудувати ейлерів цикл неможливо, але можна побудувати тільки маршрут (шлях), що обходить усі ребра по одному разу, але не замикається.

Означення 5.2. Шлях у зв'язному граф чи мультиграфі, що проходить один і тільки один раз через кожне ребро, називається *ейлеровим* (Eulerian path). Граф (мультиграф), у якому існує ейлерів шлях, але не існує ейлерового циклу, називається *напівейлеровим* (semi-Eulerian multigraph). □

Від напівейлерового мультиграфа до ейлерового — лише один крок (ребро). Якщо додати до напівейлерового мультиграфу ще одне ребро (таке, що поєднує початкову та кінцеву вершини ейлерового шляху), то шлях замкнеться і стане циклом. А напівейлерів граф перетвориться на ейлерів.

Чи є якісь ознаки, за якими можна визначити ейлеровість чи напівейлеровість мультиграфа та побудувати ейлерів цикл чи шлях? Так.

Теорема 5.1. Необхідна умова ейлеровості. Якщо зв'язний граф (або мультиграф) $G = (V, E)$ є ейлеровим, то усі його вершини мають парний ступінь. □

Доведення. Нехай мультиграф $G = (V, E)$ є ейлеровим. Візьмемо будь-яку його вершину v_i. Оскільки у графі існує ейлерів цикл, він напевно проходить через усі вершини, зокрема й через v_i. Підемо з v_i уздовж ейлерового циклу, вилучаючи з мультиграфу пройдені ребра. Ступінь v_i спочатку зменшиться на 1, ступені наступних вершин зменшаться на 2 (коли ми проходимо вершину, вилучаються два ребра), і в кінці ейлерового циклу ступінь v_i зменшиться ще на 1. В результаті всі ребра вилучені — ступені всіх вершин є 0. Оскільки під час проходження ейлерового циклу ступені вершин зменшувалися на 2 (може, й кілька разів), то з самого початку вони були парними. □

Від протилежного: якщо у мультиграфі є вершини непарної кратності, він не може бути ейлеровим Так, у мультиграфа з рис. 5.1, *б* ступені вершин 3, 3, 3, 5 — усі непарні. Отже, побудувати ейлерів цикл через сім мостів Кьоніґсберга неможливо. А якщо б були парні?

Теорема 5.2. Достатня умова ейлеровості. Якщо усі вершини зв'язного мультиграфа $G = (V, E)$ мають парний ступінь, то цей мультиграф є ейлеровим. □

Доведення. Нехай у мультиграфі $G = (V, E)$ усі вершини мають

парний ступінь. Візьмемо v_0 — довільну вершину мультиграфа. Оскільки v_0 не є ізольованою, можна побудувати шлях з початком у v_0. Побудуємо його, обираючи ребра $e_1, e_2, \ldots$ довільно, і зупинившись у момент, коли шлях не можна продовжити (тобто всі ребра, інцидентні до вершини, в яку ми потрапили, вже включені у шлях). Оскільки кількість ребер у мультиграфі є скінченною, ми зупинимося через скінченну кількість кроків. Нехай нами побудований шлях $e_1, e_2, \ldots, e_k$, що проходить через вершини $v_0, v_1, \ldots v_k$. Доведемо, що $v_k = v_0$, тобто що ми побудували цикл (поки що не ейлерів, а просто якийсь цикл).

Припустимо, що $v_k \neq v_0$. Тоді, проходячи весь шлях від v_0 до v_k, ми якусь кількість разів, скажімо, l разів, входили у вершину v_k, і $l-1$ разів з неї виходили, причому всі ребра, за якими ми входили та виходили, різні. Отже, у побудованому шляху рівно $2l-1$ ребер, інцидентних до v_k. Оскільки ступінь вершини v_k парний, існує інцидентне до v_k ребро, що не входить у побудований шлях, що суперечить нашій побудові. Отже, $v_k = v_0$.

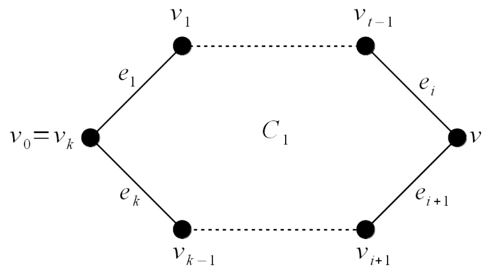


Рис. 5.2. До доведення теореми 5.2

Позначимо цикл, утворений ребрами $e_1, e_2, \ldots, e_k$, через C_1 (рис. 5.2). Якщо він ейлерів, то побудову закінчено. Нехай C_1 не ейлерів, тобто в мультиграфі G_1, отриманому з мультиграфа G вилученням ребер $\{e_1, e_2, \ldots, e_k\}$, є ще ребра. Серед цих ребер точно є ребро, інцидентне до вершини з циклу C_1, тому що в протилежному випадку C_1 був би компонентою зв'язності мультиграфа G, що не співпадає з усім мультиграфом, а це суперечить зв'язності G. Нехай f_1 — ребро мультиграфа G_1, інцидентне до якоїсь вершини v_i з циклу C_1. У мультиграфі G_1, так само як і у початковому мультиграфі G, ступеня усіх вершин парні. Дійсно, при видаленні ребер $e_1, e_2, \ldots, e_k$ ступінь кожної з вершин $v_0, v_1, \ldots v_k$ зменшилася на парне число, а ступеня решти вершин не змінилися.

Розглянемо компоненту зв'язності мультиграфа G_1, що містить ребро f_1. Усередині цієї компоненти зв'язності можна, почавши з вершини v_i, побудувати цикл $\overline{C}_2$, що складається з ребер $f_1, f_2, \ldots, f_m$ (рис. 5.3), у

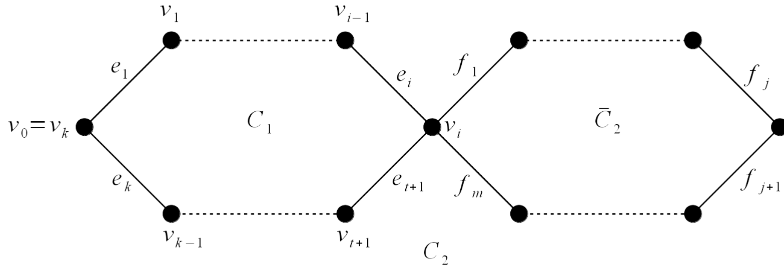


Рис. 5.3. До доведення теореми 5.2

той самий спосіб, що був побудований і цикл C_1. Тоді послідовність ребер $e_1, e_2, \ldots, e_i, f_1, f_2, \ldots, f_m, e_{i+1}, \ldots, e_k$ також утворює цикл (позначимо його C_2), і цей цикл містить більше ребер, ніж C_1. Якщо цикл C_2 ейлерів, то побудову закінчено. В іншому випадку повторюємо описаний вище алгоритм, розвинувши цикл C_2 до циклу C_3, і т. д. Оскільки кількість ребер у G є скінченною, на якомусь кроці черговий побудований цикл виявиться ейлеровим. □

У напівейлерових мультиграфах є рівно дві вершини непарного ступеня: початкова та кінцева в ейлеровому шляху. Отже, алгоритм перевірки мультиграфа на ейлерівість чи напівейлерівість виглядає так.

Перевірка мультиграфа на ейлеровість чи напівейлеровість.

1. Знаходимо вектор ступенів вершин. Для цього додаємо рядки чи стовпці матриці суміжності вершин $\boldsymbol{B}$ розміром $n \times n$, або додаємо стовпці матриці інцидентності $\boldsymbol{A}$ розміром $n \times m$.
2. Перевіряємо: якщо всі координати цього вектора парні, маємо ейлерів мультиграф; якщо дві координати непарні, а решта парні — маємо напівейлерів мультиграф; в інших випадках мультиграф не є ані ейлеровим, ані напівейлеровим.

Найпростіший, але не найшвидший алгоритм знаходження ейлерового циклу чи шляху — це алгоритм Фльорі (Fleury's Algorithm). Ось його схема. На вхід подається ейлерів чи напівейлерів мультиграф, на виході отримуємо список ребер у порядку обходу.

Алгоритм Фльорі.

1. Покладемо поточний мультиграф початковому G, покладемо поточний список ребер S пустим. Оберемо будь-яку вершину в ейлеровому графі або вершину з непарним ступенем у напівейлеровому.
2. Оберемо довільне, з урахуванням обмеження (див. нижче) ребро e поточного мультиграфа, інцидентне до поточної вершини.
3. Призначимо поточною другу вершину, інцидентну до e.
4. Видалимо e з поточного мультиграфа G та внесемо його у список S.

5. Якщо у поточному мультиграфі G ще залишилися ребра, повертаємося на крок 2.

Обмеження: якщо ступінь поточної вершини в поточному мультиграфі більше за 1, не можна обирати ребро, видалення якого з поточного мультиграфа збільшить число компонент зв'язності в ньому.

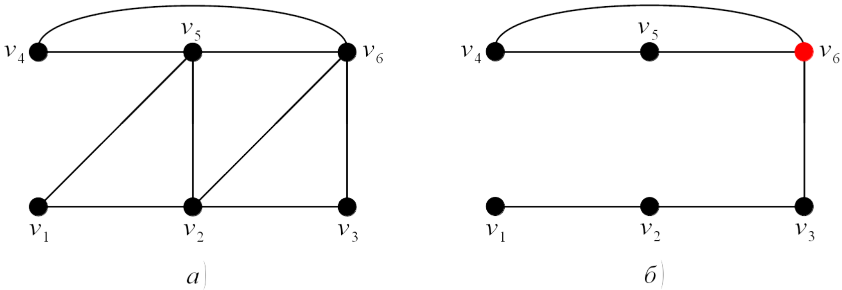


Рис. 5.4. Ейлерів граф (а) та побудова ейлерового циклу (б)

На прикладі рис. 5.4 розглянемо, як працює алгоритм Фльорі. Граф на рис. 5.4, *а* ейлерів: усі його вершини мають парний ступінь. Припустимо, що на першому кроці ми обрали вершину v_1. Ступінь цієї вершини $2 > 1$. Вилучення як ребра $e_{1,5}$, так і ребра $e_{1,2}$ не збільшує кількість компонент зв'язності, тому на цьому кроці обмеження ніяк не позначається; нехай обрано, наприклад, ребро $e_{1,5}$. На двох наступних ітераціях обмежень на вибір теж не виникає; нехай обрані ребра $e_{5,2}$ та $e_{2,6}$. Тоді поточним графом стає граф, зображений на рис. 5.4, *б* (поточна вершина — v_6). На наступній ітерації не можна обрати ребро $e_{6,3}$ через обмеження: ступінь вершини v_6 дорівнює 3, а вилучення ребра $e_{6,3}$ збільшує кількість компонент зв'язності. Тому обираємо якесь інше ребро, наприклад, $e_{6,5}$. Подальший вибір ребер тепер однозначний (поточна вершина завжди буде мати ступінь 1), так що в підсумку буде побудований такий ейлерів цикл: $e_{1,5} \to e_{5,2} \to e_{2,6} \to e_{6,5} \to e_{5,4} \to e_{4,6} \to e_{6,3} \to e_{3,2} \to e_{2,1}$.

Найелегантніша реалізація алгоритму Фльорі, на мою думку, описана в [1]. На цьому сайті наведена рекурсивна процедура, що знаходить ейлерів цикл або шлях. Вона використовує та змінює два глобальні параметри: матрицю суміжності вершин графа `B` розміром $n \times n$ типу (1.10) та одновимірний масив `L`, в який вона записує номери вершин у порядку їхнього обходу. Цей масив перед викликом процедури повинен бути пустим. У процедури один вхідний параметр — номер останньої вершини в шуканому циклі або шляху. Для ейлерового графа можна задавати останньою будь-яку вершину, а для напівейлерового — одну з двох вершин з непарним ступенем. Ось запис цього алгоритму на псевдокоді. В ньому використовується процедура `push(L,v)`, що доповнює масив `L` новим еле-

ментом v в кінці.

```
procedure Fleury(v) {v - номер останньої вершини}
begin {Fleury}
  for i=1 step 1 to n do {переглядаємо всі вершини}
  begin {for i}
    if (B(v,i)>0) {з яких є ребро до вершини номер v}
    begin {if}
      B(v,i)=0 {вилучаємо це ребро}
      B(i,v)=0 {з матриці суміжності вершин}
      call Fleury(i) {шукаємо ейлерів шлях до вершини i}
    end {if}
  end {for i}
  push(L,v) {додаємо до списку вершин L останню вершину v}
end {Fleury}
```

Саме цей алгоритм було покладено в основу методу `eulerianpath`, що міститься в пакеті Graph Theory Toolbox. Для заданого графа `G` він повертає два вектори-рядки з натуральних чисел: порядок проходження вершин та порядок проходження ребер. Якщо граф не є ані ейлеровим, ані напівейлеровим, повертаються пусті масиви. Цей метод працює лише з простими графами.

Приклад 5.1. Побудувати ейлерів шлях або цикл для заданих графів.

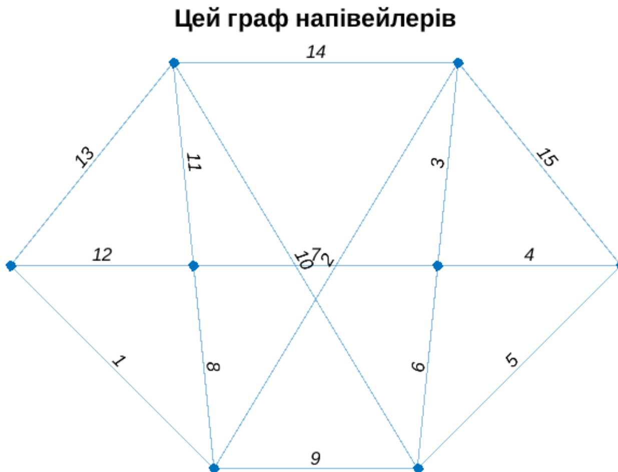


Рис. 5.5. Ейлерів шлях

```
s = [1 1 1 2 3 2 3 4 5 6 5 6 7 2 4]; % початки ребер
t = [2 3 4 3 4 5 6 7 6 7 8 8 8 7 5]; % кінці ребер
x = [0 0.8 0.9 1 2.2 2.1 2 3]; % координати вершин
y = [1 2 1 0 2 1 0 1];
G = graph(s,t); % створили граф
G = simplify(G); % спростили граф
[Vlist,Elist] = eulerianpath(G); % ейлерів шлях (цикл)
[~,isel] = sort(Elist); % номери ребер у порядку обходу
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",{},...
  "EdgeLabel",isel) % нарисували граф з мітками ребер
if isempty(Elist) % граф не ейлерів
  title("Цей граф не ейлерів") % заголовок малюнку
elseif (Vlist(1)==Vlist(end))
  title("Цей граф ейлерів") % заголовок малюнку
else
  title("Цей граф напівейлерів") % заголовок малюнку
end
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("EulerianPath","-dpng") % зберегли рисунок у файл
s = [1 3 4 5 3 6 2 8 3 9 9 10 10 7 8 6 20 28 11 21 19 8 ...
  10 11 7 14 15 12 13 13 14 26 15 16 27 12 13 20 17 17 19 ...
  19 21 17 18 18 19 20 21 22 23 24 22 21]; % початки ребер
t = [2 2 3 4 5 1 7 2 8 4 5 4 5 6 7 9 28 15 16 18 23 9 ...
  9 6 12 9 10 11 12 14 26 13 14 27 11 17 18 15 16 18 18 ...
  20 16 22 22 23 24 25 22 21 24 25 24 23]; % кінці ребер
x = [repmat(0:4,1,5) 2.5 0.3 3.7]; % координати вершин
y = [4 4 4 3.7 4 3 3.3 3.3 3 3 2 2 2 2 2 1 1 1 1 1 0 0 ...
  0.3 0 0 1.7 1.5 1.5];
G = graph(s,t); % створили граф
G = simplify(G); % спростили граф
[Vlist,Elist] = eulerianpath(G); % ейлерів шлях (цикл)
[~,isel] = sort(Elist); % номери ребер у порядку обходу
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",{},...
  "EdgeLabel",isel) % нарисували граф з мітками ребер
if isempty(Elist) % граф не ейлерів
  title("Цей граф не ейлерів") % заголовок малюнку
elseif (Vlist(1)==Vlist(end))
  title("Цей граф ейлерів") % заголовок малюнку
```

```
else
  title("Цей граф напівейлерів") % заголовок малюнку
end
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("EulerianCycle","-dpng") % зберегли рисунок у файл
```

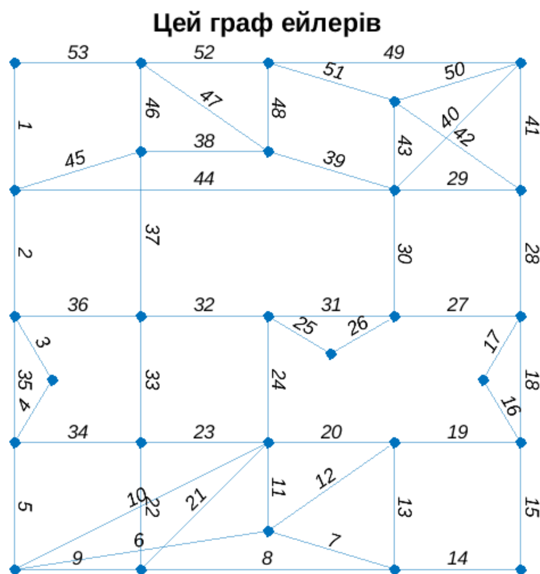


Рис. 5.6. Ейлерів цикл

На рис. 5.5 показаний ейлерів шлях у напіейлеровому графі, а на рис. 5.6 — ейлерів цикл в ейлеровому. Мітки ребер на обох рисунках — порядок обходу ребер. □

5.2. Гамільтонові цикли

Розглянемо тепер гамільтонові цикли. Їхня відмінність від ейлерових полягає в тому, що треба по одному разу пройти не ребра, а вершини.

Означення 5.3. Цикл, що проходить через кожну вершину графа один і тільки один раз, називається *гамільтоновим* (Hamiltonain cycle). Граф називається *гамільтоновим* (Hamiltonain graph), якщо в ньому існує гамільтонів цикл. □

Означення 5.4. Шлях, що проходить через кожну вершину графа один і тільки один раз, називається *гамільтоновим* (Hamiltonain path).

Граф називається *напівгамільтоновим* (semi-Hamiltonain graph), якщо в ньому існує гамільтонів шлях, але не існує гамільтонового циклу. □

Наприклад, граф на рис. 5.7, *а* є гамільтоновим: у ньому є гамільтонів цикл $v_1 \to v_2 \to v_3 \to v_8 \to v_4 \to v_9 \to v_{12} \to v_{11} \to v_7 \to v_6 \to v_{10} \to v_5 \to v_1$. А граф на рис. 5.7, *б* лише напівгамільтонів: його гамільтонів шлях $v_1 \to v_2 \to v_3 \to v_4 \to v_5$.

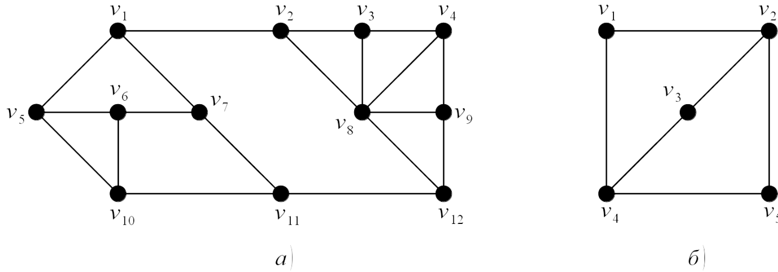


Рис. 5.7. Гамільтонів (а) та напівгамільтонів (б) графи

На відміну від ейлерових графів, задача визначення гамільтоновості графа та знаходження гамільтонового циклу є складною у тому сенсі, що не існує швидких (поліноміальних відносно n та m) алгоритмів її розв'язання. Можна лише стверджувати, що, по-перше, гамільтонів граф є зв'язним, і, по-друге, наявність петель та кратних ребер не впливає на гамільтоновість. Тому гамільтонові цикли (шляхи) шукають лише на простих графах.

Наведемо без доведення дві теореми, які надають достатні умови гамільтоновості.

Теорема 5.3. Теорема Оре (Ore). Нехай $G = (V, E)$ — простий зв'язний граф з $n > 2$. Якщо для будь-якої пари несуміжних вершин v, w сума їхніх ступенів не менша за n: $\deg v + \deg w \geq n$, то такий граф є гамільтоновим. □

Теорема 5.4. Теорема Дірака (Dirac). Нехай $G = (V, E)$ — простий зв'язний граф з $n > 2$. Якщо ступінь будь-якої його вершини v є не меншим за $\frac{n}{2}$: $\deg v \geq \frac{n}{2}$, то такий граф є гамільтоновим. □

Зауважимо, що ці теореми дають лише достатню, але не необхідну умову гамільтоновості. Існують гамільтонові графи, для яких ці теореми не виконуються. Наприклад, шестикутник: для нього $n = 6$; $m = 6$; ступінь кожної вершини $\deg v_i = 2 < 3$; сума ступенів будь-якої пари несуміжних вершин $\deg v + \deg w = 4 < 6$; але, вочевидь, цей граф гамільтонів: є гамільтонів цикл уздовж шести його вершин.

5.3. Фундаментальна система циклів

Можливо, ви вивчали електротехніку, і вам доводилося розраховувати мережі за законами Кірхгофа. Електрична мережа може бути представлена у вигляді графа (чи мультиграфа) з n вершинами та m ребрами. Рівняння 1-го закону Кірхгофа — це баланс струмів у вузлах. Загальний баланс струмів в усіх вузлах дорівнює нулю, тому з n рівнянь 1-го закону незалежними будуть лише $n-1$. Щоб знайти струми в усіх m гілках мережі (ребрах графа), потрібні ще $m-n+1$ незалежних рівнянь. Їх дає 2-й закон Кірхгофа: сумарне падіння напруг у кожному замкненому контурі дорівнює сумі ЕРС у цьому контурі. Тому треба знайти такі $m-n+1$ контурів, щоб рівняння 2-го закону Кірхгофа для них були незалежними. Аналізуючи структуру рівнянь 2-го закону Кірхгофа, бачимо, що це виконується, якщо кожен з контурів буде відрізнятися від будь-якого іншого хоча б однією гілкою. Тому, розв'язуючи відповідну задачу на графі, ми повинні побудувати $m-n+1$ простих циклів, кожен з яких відрізняється від будь-якого іншого хоча б одним ребром.

Пригадаємо та уточнимо означення простого циклу. У попередній главі ми визначили шлях та його окремий випадок цикл як послідовність інцидентних вершин та ребер. Але ця інформація є надлишковою: щоб однозначно визначити будь-який шлях (у т. ч. й цикл), достатньо задати тільки послідовність ребер. Більше того, якщо нас цікавлять лише прості шляхи та цикли, то не важливий навіть порядок ребер. Тому будемо задавати простий цикл як деяку підмножину множини ребер. А порядок завжди можна відновити. Алгоритм такого відновлення простий, як доміно: беремо будь-яке ребро, шукаємо, яке з ним суміжне, потім шукаємо, яке з цим другим суміжне і т. д. Оскільки ребра утворюють простий цикл, то останнє ребро буде суміжним до першого з іншого кінця.

Наприклад, прості цикли графа з рис. 5.8, *а* — це підмножини з множини його ребер: $\{e_1, e_3, e_4\}$, $\{e_2, e_3, e_5\}$, $\{e_1, e_2, e_4, e_5\}$, які показані відповідно на рис. 5.8, *бвг*.

Але не будь-яка підмножина множини ребер E є простим циклом. Так, у цьому ж прикладі $\{e_1, e_2\}$ чи $\{e_1, e_2, e_4\}$ не є циклами. Тому ми будемо розглядати тільки такі підмножини множини ребер E, які дійсно є простими циклами. Будемо позначати їх C_1, C_2 тощо.

У нашому приклад у графа з рис. 5.8, *а* є лише 3 простих цикли: $C_1 = \{e_1, e_3, e_4\}$; $C_2 = \{e_2, e_3, e_5\}$; $C_3 = \{e_1, e_2, e_4, e_5\}$. Розглянемо властивості простих циклів та визначимо дії над ними.

Властивість 5.1. Множина ребер, що утворює простий цикл, не є пустою. □

Доведення є очевидним: у циклі повинні бути ребра, інакше це не цикл. □

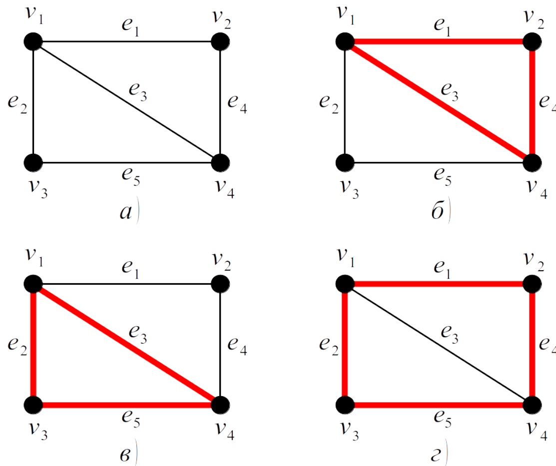


Рис. 5.8. Граф (а) та його прості цикли (б, в, г)

Властивість 5.2. Якщо множина ребер C_i є простим циклом, то будь-яка її власна підмножина не є циклом. □

Доведення. Якщо ми видалимо з простого циклу хоча б одно ребро, то він розімкнеться та вже не буде циклом. □

Введемо тепер над підмножинами операцію додавання за логікою XOR (виключного або): з двох підмножин C_i та C_j формуємо нову підмножину, включаючи до неї тільки неповторювані елементи з C_i та C_j.

Означення 5.5. *Прямою сумою* (direct sum) двох простих циклів C_i та C_j називається підмножина ребер $C_i \oplus C_j$, до якої входять ребра, які є або тільки в C_i, або тільки в C_j. □

Чи буде пряма сума простих циклів простим циклом? Інколи так, а інколи й ні. Перевіримо, наприклад, що буде при додаванні простих циклів графа з рис. 5.8. У нас є $C_1 = \{e_1, e_3, e_4\}$; $C_2 = \{e_2, e_3, e_5\}$; $C_3 = \{e_1, e_2, e_4, e_5\}$. Додаємо:

- $C_1 \oplus C_2 = \{e_1, e_2, e_4, e_5\} = C_3$;
- $C_1 \oplus C_3 = \{e_2, e_3, e_5\} = C_2$;
- $C_2 \oplus C_3 = \{e_1, e_3, e_4\} = C_1$.

Як бачимо, в цьому прикладі пряма сума двох будь-яких простих циклів дає третій, також простий цикл. Але поглянемо на рис. 5.9.

На рис. 5.9, *а* показаний граф, а на рис. 5.9, *бв* — два його прості цикли: $C_1 = \{e_{2,3},\ \ e_{2,4}, e_{3,6}, e_{4,5}, e_{5,6}\}$ та $C_2 = \{e_{2,3}, e_{2,5}, e_{3,6}, e_{5,8}, e_{6,9}, e_{8,9}\}$.

У результаті їхнього прямого додавання отримаємо підмножину ребер: $\{e_{2,4}, e_{2,5}, e_{4,5}, e_{5,6}, e_{5,8}, e_{6,9}, e_{8,9}\}$, яка не є простим циклом. Це видно

й на рис. 5.9, *г*, і з перевірки властивості 5.2. У цієї множини є власні підмножини $\{e_{2,4}, e_{2,5}, e_{4,5}\}$ та $\{e_{5,6}, e_{5,8}, e_{6,9}, e_{8,,9}\}$, які є простими циклами. Оскільки властивість 5.2 не виконується, то отримана множина $C_1 \oplus C_2$ не є простим циклом.

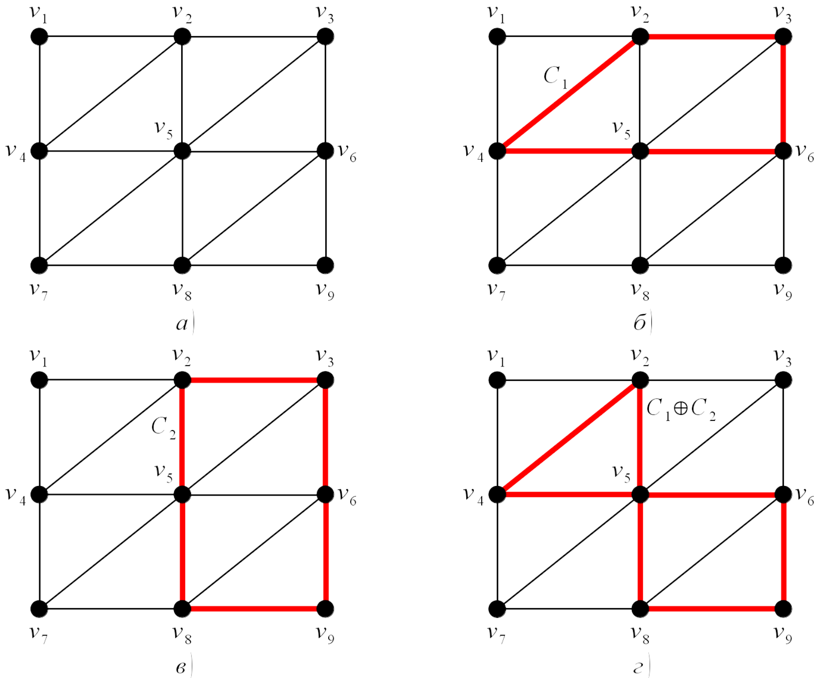


Рис. 5.9. Граф (а), його прості цикли (б, в) та їхня пряма сума (г)

Але можна стверджувати, що $C_1 \oplus C_2$ містить у собі, як підмножину, якийсь простий цикл.

Властивість 5.3. Якщо C_i та C_j — прості цикли, то існує простий цикл $C_k \subseteq C_i \oplus C_j$. □

Доведення. Додамо до ребер кожного простого циклу C_i та C_j усі вершини, інцидентні до його ребер (тобто ті, що знаходяться між ребрами). Оскільки цикли прості, то всі ці вершини мають ступінь 2 як у C_i, так і у C_j. При об'єднанні всіх ребер $C_i \cup C_j$ отримаємо або простий граф, або мультиграф, тому що деякі ребра з C_i та C_j можуть повторюватися (не більш ніж удвічі). При цьому ступені вершин додаються, і будуть або $0 + 0 = 0$, або $0 + 2 = 0$, або $2 + 2 = 4$. Далі за логікою XOR вилучаємо кратні ребра. При вилученні двох кратних ребер ступінь обох інцидентних до них вершин зменшується на 2. Т. ч., у $C_i \oplus C_j$ будуть вершини

лише парного ступеня: або 0, або 2, або 4. Такий граф (або його частина — підграф) є ейлеровим: в ньому існує цикл (не обов'язково простий) що по одному разу обходить усі ребра. А з будь-якого циклу завжди можна виділити простий цикл. □

Розглянемо цикли з точки зору теорії матроїдів. У попередній главі ми будували графовий матроїд $M = (S, I)$ на носії S — множині ребер графа. Незалежними множинами (елементами I) такого матроїда є всі можливі дерева та ліси, а базами — остовні дерева. З теорії остовних дерев відомо, що додавання до остовного дерева (рис. 5.10, *а*) ще одного ребра (рис. 5.10, *б*) утворює рівно один простий цикл. Крім цього циклу, в отриманій підмножині ребер, можливо, будуть ребра, що не входять у цикл — такі собі "хвости"(рис. 5.10, *в*). Після видалення цих хвостів отримаємо простий цикл (рис. 5.10, *г*).

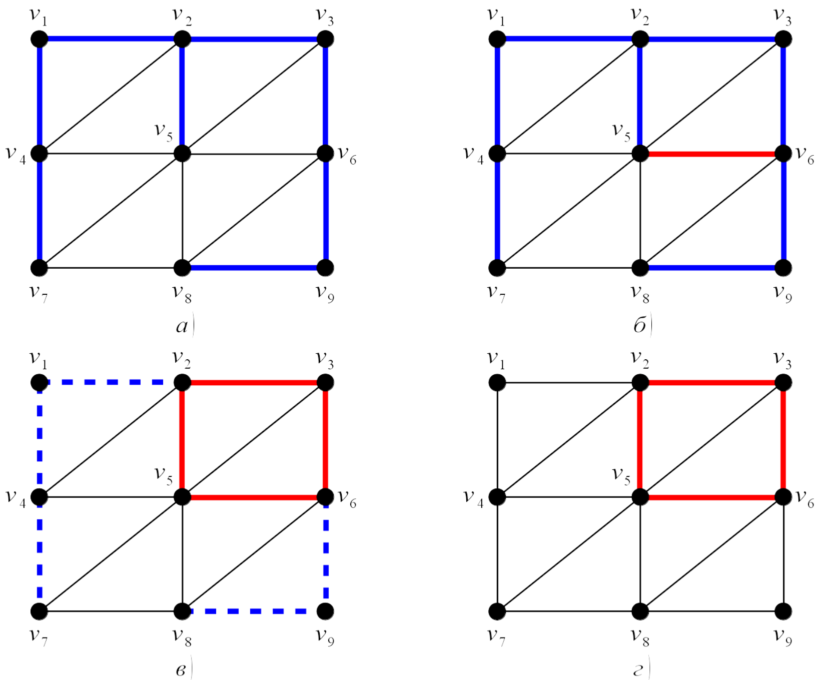


Рис. 5.10. Додавання до остовного дерева (а) ребра (б) утворює множину ребер (в), що містить простий цикл (г)

Цим діям відповідають такі означення теорії матроїдів.

Означення 5.6. *Залежна множина матроїда* (dependent set of matroid) — це підмножина елементів носія S, що не є незалежною. □

Означення 5.7. *Цикл матроїда* (circuit, cycle of matroid) — мінімальна за включенням залежна множина матроїда. □

Залежна множина будується з незалежної додаванням нових елементів носія (ребер графа). При цьому інколи ми будемо отримувати нову незалежну множину, а інколи — залежну. Щоб напевно отримати залежну множину, треба додати ребро до незалежної множини максимальної потужності, тобто до бази матроїда — остовного дерева. Після цього ми отримаємо залежну множину, але ще не цикл. Треба з цієї залежної множини так видаляти елементи, щоб множина залишилася залежною (циклом). Алгоритм такого видалення був нами розглянутий у попередньому розділі 4, див. рис. 4.7 та 4.8.

Множину циклів матроїда можна визначити через властивості, які ми вже довели: це властивості 5.1, 5.2 та 5.3. Ось формулювання цього означення.

Означення 5.8. Множина $\mathcal{C}$ є множиною всіх простих циклів матроїда, якщо виконуються такі аксіоми.

1. Жоден з елементів $\mathcal{C}$ не є пустою множиною.
2. Жоден з елементів $\mathcal{C}$ не є власною підмножиною іншого елемента $\mathcal{C}$.
3. Якщо $C_1, C_2 \in \mathcal{C}$ та $e \in C_1 \cap C_2$, то $\exists C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$. □

Це дає можливість визначити матроїд не тільки через незалежні множини носія, як ми це зробили у попередньому розділі, а й через залежні множини, мінімальні за включенням, тобто через прості цикли.

Означення 5.9. *Матроїд* $M = (S, \mathcal{C})$ — це пара двох множин: S — носія матроїда, та $\mathcal{C}$ — сімейства залежних множин: непустих підмножин елементів S, для яких мають місце аксіоми з означення 5.8. □

Чи можна серед усіх простих циклів обрати якусь мінімальну їхню множину так, щоб усі інші прості цикли можна було б утворити за властивістю 5.3? Множину $\mathcal{C}$ усіх простих циклів графа можна розглядати як деякий комбінаторний простір. Тоді той мінімальний набір циклів, з яких можна побудувати всі інші, доречно назвати базисом цього простору.

Означення 5.10. *Базис у просторі циклів* (*цикловий базис*, *фундаментальна система циклів*, cycle basis, fundamental cycles) — це мінімальна за кількістю елементів підмножина простих циклів, з яких можна побудувати будь-який інший простий цикл за властивістю 5.3. Кількість циклів у цьому базисі називається *циклічним рангом* (cyclic rank) графа. □

У звичайних лінійних просторах базисні елементи є лінійно-незалежними. Для перевірки цього факту треба побудувати їхню лінійну комбінацію та спробувати знайти для неї ненульові коефіцієнти. У комбінаторних просторах ситуація дещо інша: ми не вводили до розгляду операцію множення на скаляр. Абстрактну алгебру ми не вивчали, і у нас є тільки

операція додавання з наступним виділенням підмножини. Тому й означення лінійної незалежності тут буде дещо іншим.

Означення 5.11. Прості цикли називаються *незалежними* (independent cycles), якщо вони відрізняються один від одного хоча б одним ребром. □

Сформулюємо кілька теорем про незалежні прості цикли.

Теорема 5.5. У системі незалежних простих циклів жоден з них не може бути побудований з інших шляхом додавання за логікою XOR та вибірки підмножини (тобто за властивістю 5.3). □

Доведення. Візьмемо будь-який цикл C_0 з нашої системи незалежних простих циклів. У ньому обов'язково є ребро, якого немає в жодному іншому циклі системи. Тому, як би ми не об'єднували множини ребер з інших циклів, і що б ми не обирали з цих об'єднань, все одно відсутнє ребро ніколи не з'явиться. Тому побудувати C_0 ніколи не вдасться. □

Висновок: щоб побудувати будь-який простий цикл C_0 за властивістю 5.3, треба мати принаймні таку систему незалежних простих циклів, в якій є всі ребра з C_0.

Теорема 5.6. (Зворотна до 5.5). Якщо деяка система простих циклів не є незалежною, то принаймні один з них можна побудувати з інших за властивістю 5.3. □

Доведення. Згідно означення 5.11 у залежній системі циклів є хоча б один, кожне ребро якого є у якомусь іншому циклі. Позначимо цей цикл C_0. Об'єднуючи ребра інших циклів, отримаємо множину ребер, що містить C_0. □

Ця теорема стверджує таке. Якщо у нас є якийсь простий цикл C_0, всі ребра якого входять в інші прості цикли, то C_0 завжди можна побудувати з цих інших за властивістю 5.3. Дійсно, поглянемо на рис. 5.9. Тут є два простих цикли: $C_1 = \{e_{2,3}, e_{2,4}, e_{3,6}, e_{4,5}, e_{5,6}\}$ та $C_2 = \{e_{2,3}, e_{2,5}, e_{3,6}, e_{5,8}, e_{6,9}, e_{8,9}\}$. Розглянемо простий цикл $C_0 = \{e_{2,3}, e_{2,5}, e_{3,6}, e_{5,6}\}$. Він є залежним з C_1 та C_2, оскільки всі його ребра містяться або в C_1, або в C_2. Чи можна побудувати його з C_1 та C_2 за властивістю 5.3? Теорема 5.6 стверджує, що так. Це можна зробити, наприклад, у такий спосіб. Спочатку будуємо $C_3 = \{e_{2,4}, e_{2,5}, e_{4,5}\} \subset C_1 \oplus C_2$, а потім знаходимо $C_0 = \{e_{2,3}, e_{2,5}, e_{3,6}, e_{5,6}\} = C_1 \oplus C_3$. На другому кроці нам навіть не потрібно вибирати підмножину: пряма сума C_1 та C_3 відразу дає потрібний нам простий цикл C_0.

Теорема 5.7. У зв'язному графі $G = (V, E)$ з $|V| = n$, $|E| = m$ існує принаймні $m - n + 1$ незалежних простих циклів. □

Доведення. Це доведення є конструктивним, тобто дає алгоритм побудови системи з $m-n+1$ незалежних простих циклів. Побудуємо будь-

яке остовное дерево графа G. До нього увійдуть $n-1$ ребер, і не увійдуть решта $m-n+1$ ребер. При додаванні кожного з них до остовного дерева утвориться рівно один простий цикл — усього маємо $m-n+1$ простих циклів. У кожному з них є принаймні одне ребро, якого немає в інших: це те ребро, що ми додали до остовного дерева для утворення простого циклу. Тому отримані $m-n+1$ простих циклів є незалежними. □

Ця теорема дає нам дуже зручний алгоритм побудови $m-n+1$ незалежних простих циклів: треба взяти остовне дерево та додати до нього одне ребро. При цьому утвориться рівно один простий цикл з "хвостами" (як медуза зі щупальцями). Відрізавши "хвости", отримаємо простий цикл. Потім повторюємо цю процедуру з іншим відкинутим ребром — отримаємо інший простий цикл, і т. д. — усього будемо мати $m-n+1$ незалежних простих циклів. Але чи достатньо їх? Чи можна сказати, що будь-який інший простий цикл у графі можна буде побудувати з них за властивістю 5.3? Відповідь на це питання дає наступна теорема.

Теорема 5.8. Будь-який простий цикл зв'язного графа $G=(V,E)$ з $|V|=n$, $|E|=m$ є залежним з тими $m-n+1$ незалежними простими циклами, що були побудовані у теоремі 5.7. □

Доведення. Кожен з незалежних простих циклів, що ми побудували, утворюється з якихось ребер остовного дерева та однієї хорди (так називають ребра, викинуті при побудові остовного дерева). Розглянемо будь-який інший простий цикл C_0, якого немає у нашій системі незалежних простих циклів. У ньому не можуть бути ребра лише з остовного дерева, бо в того немає циклів. Тому у C_0 є щонайменше дві хорди (а, може, й більше), та якась кількість ребер з остовного дерева. Будуємо C_0. Додамо за логікою XOR ті цикли системи, в яких є хорди, що входять у C_0. Всі інші ребра є спільними, бо входять у остовне дерево. Викинувши "хвости"отримаємо C_0. □

За цією теоремою ми можемо стверджувати, що побудовані за допомогою остовного дерева $m-n+1$ незалежних простих циклів дійсно утворюють базис у просторі циклів: будь-який інший простий цикл будується з них шляхом прямого додавання з наступною вибіркою згідно властивості 5.3. Як кажуть, циклічний ранг зв'язного графа дорівнює кількості хорд будь-якого остовного дерева.

Розглянемо як приклад граф з рис. 5.9, *а*. У нього $n=9$ вершин та $m=16$ ребер. Згідно з теоремами 5.7, 5.8 він має $m-n+1=8$ незалежних простих циклів. Для такого простого графа ми й так їх бачимо на рис. 5.9, *а*: це 8 малих трикутників: $C_1=\{e_{1,2},e_{1,4},e_{2,4}\}$; $C_2=\{e_{2,4},e_{2,5},e_{4,5}\}$; $C_3=\{e_{2,3},e_{2,5},e_{3,5}\}$ тощо. Але при розв'язанні задачі на комп'ютері незалежні цикли можуть бути й іншими: все залежить від того, яке остовне дерево використовується для їхньої побудови. Якщо, наприклад, взяти остовне дерево з рис. 5.10, *а*, то отримаємо незалежні

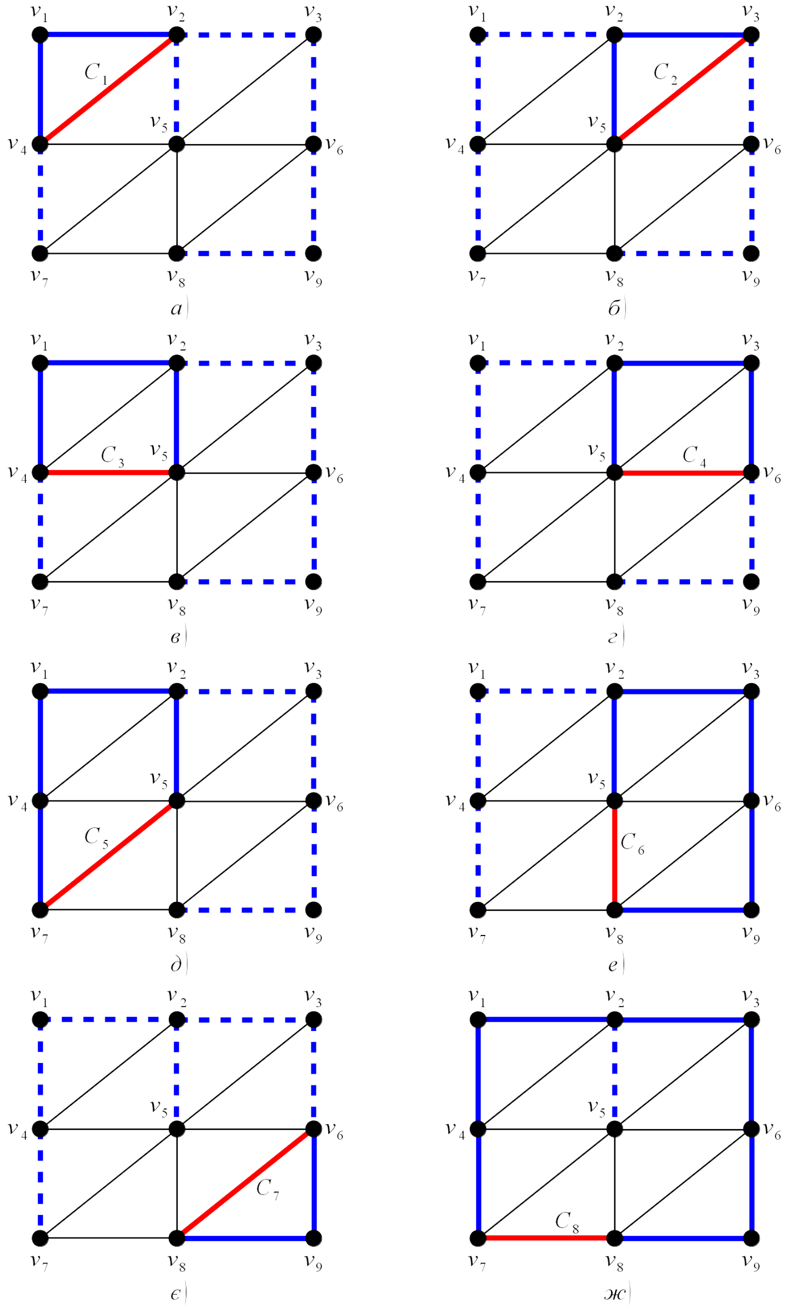


Рис. 5.11. Система незалежних простих циклів

цикли, представлені на рис. 5.11. Червоним кольором показано те ребро, що додається до остовного дерева (хорда), а штриховими лініями — "хвости", що видаляються для отримання простого циклу. Отримана система простих циклів дійсно є незалежною: у кожному з них є ребро, якого немає в інших циклах, воно нарисоване червоним кольором. Оскільки знайдених циклів $m - n + 1 = 8$, то вони утворюють базис: більше незалежних простих циклів побудувати не можна.

Це як у звичайному лінійному просторі L_n: базис може бути не тільки натуральним $\{\boldsymbol{e}_1, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$, а й будь-якою системою лінійно незалежних векторів $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_n\}$. Тільки у нас цикли можуть множитися лише на 0 або 1, а додаються за логікою XOR.

У MATLAB є метод (вбудована функція) `cyclebasis`, яка для графа `G` знаходить усі незалежні цикли та повертає інформацію про них у вигляді двох вихідних параметрів:

1. `cycles` — масив-стовпчик комірок, у якому `cycles{k}` містить номери вершин, включених до k-го циклу (вектори-рядки);
2. `edgecycles` — масив-стовпчик комірок, у якому `edgecycles{k}` містить номери ребер, включених до k-го циклу (вектори-рядки).

Приклад 5.2. Для напівейлерового графу з прикладу 5.1 знайти фундаментальну систему циклів та нарисувати їх.

```
s = [1 1 1 2 3 2 3 4 5 6 5 6 7 2 4]; % початки ребер
t = [2 3 4 3 4 5 6 7 6 7 8 8 8 7 5]; % кінці ребер
x = [0 0.8 0.9 1 2.2 2.1 2 3]; % координати вершин
y = [1 2 1 0 2 1 0 1];
G = graph(s,t); % створили граф
[cycles,edgecycles] = cyclebasis(G); % незалежні цикли
Cc = size(cycles,1); % кількість циклів
for k=1:Cc % рисуємо цикли
  figure % нове вікно фігури
  h = plot(G,"XData",x,"YData",y, ...
    "NodeLabel",{}); % нарисували граф
  highlight(h,"Edges",edgecycles{k},...
    "EdgeColor",'r',"LineWidth",5) % позначили ребра циклу
  highlight(h,cycles{k},"NodeColor","b",...
    "MarkerSize",8) % позначили вершини циклу
  title("Цикл № " + k) % заголовок рисунку
  axis("equal") % однаковий масштаб уздовж осей координат
  axis("off") % прибрали осі
  print("Cycle" + k,"-dpng") % зберегли рисунок у файл
end
```

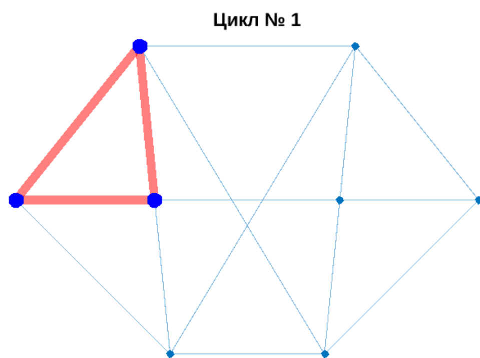


Рис. 5.12. Цикл №1

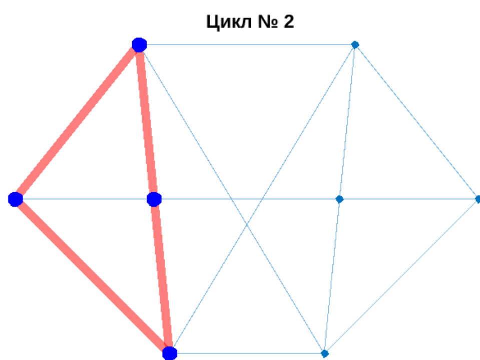


Рис. 5.13. Цикл №2

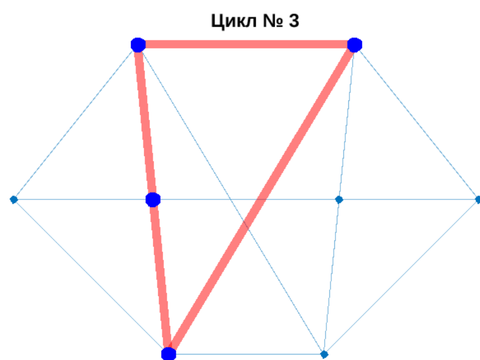


Рис. 5.14. Цикл №3

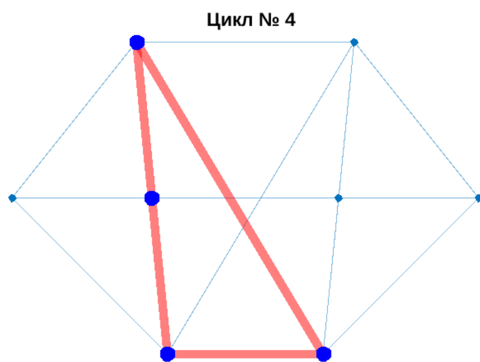


Рис. 5.15. Цикл №4

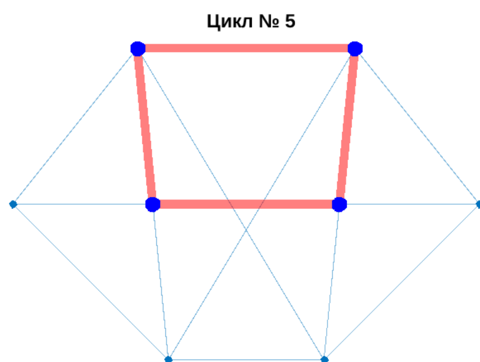


Рис. 5.16. Цикл №5

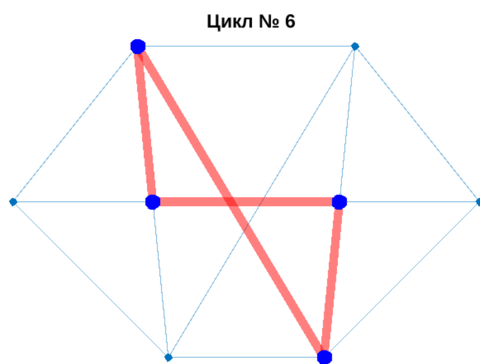


Рис. 5.17. Цикл №6

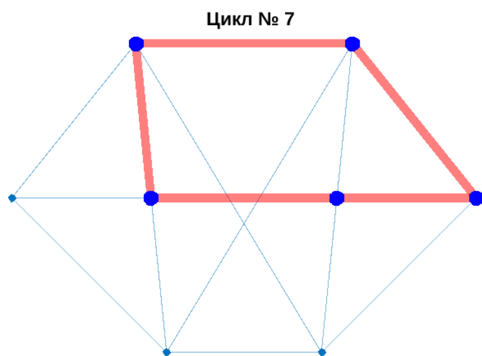


Рис. 5.18. Цикл №7

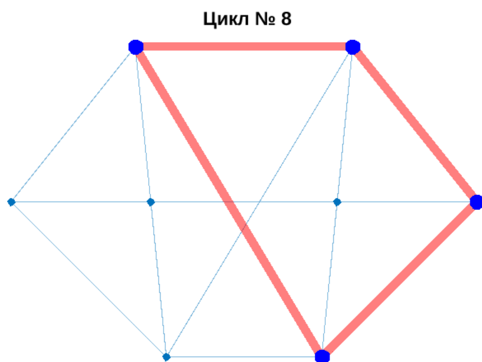


Рис. 5.19. Цикл №8

На рис. 5.12-5.19 показані всі $m - n + 1 = 15 - 8 + 1 = 8$ незалежних циклів цього графа. □

Ми розглянули побудову будь-якої системи незалежних циклів. Значно складнішою є задача побудови незалежної множини циклів мінімальної загальної ваги, якщо ребра графа зважені. Ця задача була розв'язана Хортоном [7].

5.4. Фундаментальна система коциклів

Яку мінімальну кількість ребер треба видалити зі зв'язного графа, щоб він втратив свою зв'язність? Якщо у графа є висяча вершина (вершина ступеня 1), то, вочевидь, її можна ізолювати, видалив єдине ребро, що поєднує її з рештою вершин графа. Тобто в цій задачі досить видалити

лише одне ребро. А якщо треба так розрізати граф, щоб деяка підмножина вершин V_1 разом з інцидентними лише до них ребрами стала однією компонентою зв'язності, а решта вершин V_2 разом з інцидентними лише до них ребрами — іншою? Ця задача теж досить просто розв'язується. Нехай задане розбиття множини вершин на дві підмножини, що не перетинаються: $V = V_1 \oplus V_2$. Переглядаємо всі ребра та дивимось: якщо обидві вершини ребра належать до V_1, залишаємо ребро у 1-й компоненті. Якщо обидві вершини ребра належать до V_2, залишаємо ребро у 2-й компоненті. А ось якщо одна з вершин ребра належить до V_1, а інша до V_2, то таке ребро треба вилучити для розрізання графа.

Означення 5.12. *Розріз* (cut) — це множина ребер, видалення якої зі зв'язного графа робить його незв'язним. *Коцикл* (cocycle) — це розріз з мінімальною за включенням кількістю ребер. □

Наведений вище алгоритм будує коцикл, бо вилучається лише мінімально необхідна кількість ребер: тільки ті, у яких один кінець належить до V_1, а інший — до V_2.

У математиці префікс "ко" означає якусь двоїстість, симетричність чи протилежність до поняття, що є коренем слова. Розглянемо, як це виглядає в циклах та коциклах. Почнемо з властивостей.

Властивість 5.4. Множина ребер, що утворює коцикл, не є пустою (як і в циклі). □

Доведення є очевидним: не видалили ребер — граф не роз'єднався. □

Властивість 5.5. Якщо множина ребер K_i є коциклом, то будь-яка її власна підмножина не є коциклом (як і в циклі). □

Доведення. Коцикл — це мінімальна за включенням множина ребер, що роз'єднує граф на дві частини. Тому, якщо ми видалимо з коциклу хоча б одно ребро, то він вже не буде роз'єднувати граф, тобто вже не буде коциклом. □

Поки що маємо повне співпадіння з властивостями циклів 5.1, 5.2. Підемо далі. Введемо над коциклами дію додавання за логікою XOR (виключного або): з двох підмножин ребер K_i та K_j формуємо нову підмножину, включаючи до неї тільки неповторювані елементи з K_i та K_j (як для циклів).

Означення 5.13. *Прямою сумою* (direct sum) двох коциклів K_i та K_j називається підмножина ребер $K_i \oplus K_j$, до якої входять ребра, які є або тільки в K_i, або тільки в K_j. □

Як і для циклів, пряма сума двох коциклів може або відразу давати коцикл, або містити його. Наприклад, у граф з рис. 5.20, *а* є коцикли $K_1 = \{e_1, e_2, e_3\}$; $K_2 = \{e_2, e_3, e_4\}$; $K_3 = \{e_1, e_4\}$. Вони показані червоними лініями на рис. 5.20, *бвг*. Зеленою лінією відмічений відповідний розріз на

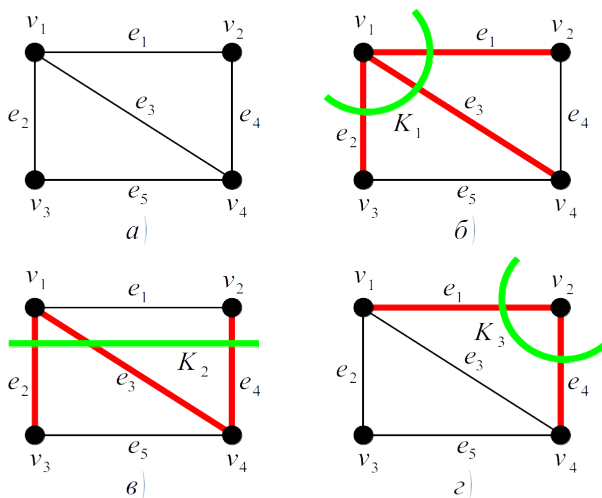


Рис. 5.20. Граф (а) та його коцикли (б, в, г)

дві частини. Додаємо:

- $K_1 \oplus K_2 = \{e_1, e_4\} = K_3$;
- $K_1 \oplus K_3 = \{e_2, e_3, e_4\} = K_2$;
- $K_2 \oplus K_3 = \{e_1, e_2, e_3\} = K_1$.

У цьому прикладі пряма сума двох будь-яких коциклів відразу дає третій коцикл.

Але це не завжди так. Наприклад на рис. 5.21, *аб* показані (синіми лініями) два коцикли $K_1 = \{e_{1,2}, e_{2,4}, e_{2,5}, e_{3,5}, e_{5,6}, e_{6,8}, e_{8,9}\}$ та $K_2 = \{e_{2,3}, e_{3,5}, e_{4,7}, e_{5,6}, e_{5,7}, e_{5,8}\}$. Відповідні розрізи намальовані червоними лініями. Але їхня пряма сума $K_1 \oplus K_2 = \{e_{1,2}, e_{2,3}, e_{2,4}, e_{2,5}, e_{4,7}, e_{5,7}, e_{5,8}, e_{6,8}, e_{8,9}\}$, зображена на рис. 5.21, *в*, не є коциклом: вона розділяє граф не на дві, а на три частини. Це краще видно на рис. 5.21, *г*, де жирними лініями показані ті ребра, яких немає в $K_1 \oplus K_2$. Щоб отримати з $K_1 \oplus K_2$ коцикл, треба вилучити якісь ребра. Наприклад, якщо вилучити $\{e_{1,2}, e_{2,3}, e_{2,4}, e_{2,5}\}$, то отримаємо коцикл $\{e_{4,7}, e_{5,7}, e_{5,8}, e_{6,8}, e_{8,9}\}$. Звідсіля — наступна властивість.

Властивість 5.6. Якщо K_i та K_j — коцикли, то існує коцикл $K_k \subseteq K_i \oplus K_j$. $\square$

Доведення. Нехай коцикл K_1 розрізає вершини графа на підмножини V_1 та V_2, а K_2 — на W_1 та W_2, як показано на рис. 5.22. Ребра K_1 — це горизонтальні та нахилені лінії, а ребра K_2 — вертикальні та нахилені лінії. При обчисленні $K_1 \oplus K_2$ спільні ребра (сині нахилені лінії) вилуча-

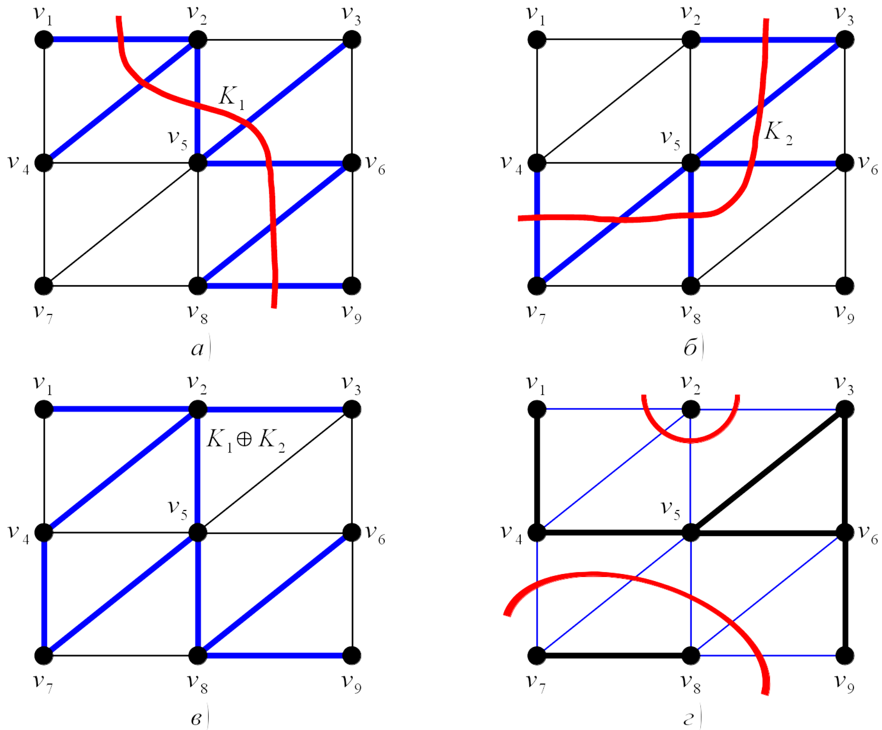


Рис. 5.21. Два коцикли (а, б) та їхня пряма сума (в), що містить два коцикли (г)

ються, а залишаються лише ті ребра, що входять тільки в один з коциклів (червоні вертикальні та горизонтальні лінії). Отже, ребра, що входять до $K_1 \oplus K_2$, є розрізами між підмножинами вершин:

$$\begin{array}{lcl} \{V_1 \cap W_1\} & \leftrightarrow & \{V_1 \cap W_2\}\,; \\ \{V_1 \cap W_1\} & \leftrightarrow & \{V_2 \cap W_1\}\,; \\ \{V_1 \cap W_2\} & \leftrightarrow & \{V_2 \cap W_2\}\,; \\ \{V_2 \cap W_1\} & \leftrightarrow & \{V_2 \cap W_2\}\,. \end{array} \tag{5.1}$$

При цьому ті підмножини вершин, що розташовані на рис. 5.22 вздовж діагоналі, можуть залишитися нерозрізаними, деякі можуть виявитися пустими, але у будь-якому випадку ребрами $K_1 \oplus K_2$ граф розрізається на дві, три або чотири частини. А з цих ребер завжди можна вибрати такі, що розрізають граф лише на дві частини. □

Наприклад, на рис. 5.21 маємо:

- $V_1 = \{v_1, v_4, v_5, v_7, v_8\}$;

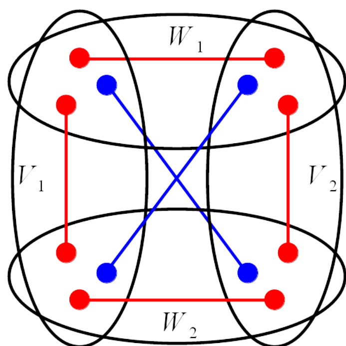


Рис. 5.22. До доведення властивості 5.6

- $V_2 = \{v_2, v_3, v_6, v_9\}$;
- $W_1 = \{v_1, v_2, v_4, v_5\}$;
- $W_2\, \{v_3, v_6, v_7, v_8, v_9\}$.

Підмножини вершин, що розрізаються ребрами $K_1 \oplus K_2$:

- $\{V_1 \cap W_1\} = \{v_1, v_4, v_5\}$;
- $\{V_1 \cap W_2\} = \{v_7, v_8\}$;
- $\{V_2 \cap W_1\} = \{v_2\}$;
- $\{V_2 \cap W_2\} = \{v_3, v_6, v_9\}$.

Підмножини $\{V_1 \cap W_1\}$ та $\{V_2 \cap W_2\}$ об'єдналися (сині лінії, що вилучені на рис. 5.22), а між $\{V_1 \cap W_2\}$ та $\{V_2 \cap W_1\}$ взагалі не було ребер: вони залишилися роз'єднаними.

Інший приклад — рис. 5.20. Тут для K_1:

- $V_1 = \{v_1\}$;
- $V_2 = \{v_2, v_3, v_4\}$.

Для K_2:

- $W_1 = \{v_1, v_2\}$;
- $W_2 = \{v_3, v_4\}$.

Підмножини вершин, що розрізаються ребрами $K_1 \oplus K_2$:

- $\{V_1 \cap W_1\} = \{v_1\}$;
- $\{V_1 \cap W_2\} = \{\varnothing\}$;
- $\{V_2 \cap W_1\} = \{v_2\}$;
- $\{V_2 \cap W_2\} = \{v_3, v_4\}$.

Підмножини $\{V_1 \cap W_1\}$ та $\{V_2 \cap W_2\}$ об'єдналися, у $\{V_1 \cap W_2\}$ немає вершин — залишилися дві частини.

Коцикли, як і цикли, є залежними множинами матроїда, який так і називається: матроїд коциклів. Для коциклів мають місце тобто властивості 5.4, 5.5, 5.6. А, отже, і означення 5.9.

Переходимо тепер до визначення базису у комбінаторному просторі

коциклів, тобто такої мінімальної кількості коциклів, з яких можна побудувати всі інші за властивістю 5.6. Матроїдна структура нам у цьому допоможе.

Означення 5.14. *Базис у просторі коциклів* (*коцикловий базис*, *фундаментальна система коциклів*, cocycle basis, fundamental cocycles) — це мінімальна за кількістю елементів підмножина коциклів, з яких можна побудувати будь-який інший коцикл за властивістю 5.6. Кількість коциклів у цьому базисі називається *коциклічним рангом* (cocycle rank) графа. □

Означення 5.15. Коцикли називаються *незалежними* (independent cocycles), якщо вони відрізняються один від одного хоча б одним ребром. □

Наприклад, на рис. 5.20 коцикли K_1, K_2 та K_3 є залежними: кожен з них є прямою сумою інших двох. У той же час будь-які два з них є незалежними, бо відрізняються хоча б одним ребром. Але взяти два з них і сказати, що це базис, ми не можемо: за їх допомогою ми ніколи не розріжемо v_3 та v_4, тому що ребро e_5, яке їх поєднує, не входить в жоден коцикл.

Теорема 5.9. У системі незалежних коциклів жоден з них не може бути побудований з інших шляхом додавання за логікою XOR та вибірки підмножини (тобто за властивістю 5.6). □

Доведення таке саме, як і для теореми 5.5. □

Теорема 5.10. (Зворотна до 5.9). Якщо деяка система коциклів не є незалежною, то принаймні один з них можна побудувати з інших за властивістю 5.6. □

Доведення таке саме, як і для теореми 5.6. □

Теорема 5.11. У зв'язному графі $G = (V, E)$ з $|V| = n$, $|E| = m$ існує принаймні $n - 1$ незалежних коциклів. □

Доведення. Як для циклів, доведення є конструктивним, тобто дає алгоритм побудови системи з $n-1$ незалежних коциклів. Побудуємо будь-яке остовное дерево графа G. До нього увійдуть $n-1$ ребер. При видаленні якогось ребра з остовного дерева граф розіб'ється на дві компоненти зв'язності. Далі переглядаємо всі ребра та обираємо ті з них, у яких одна вершина входить до однієї компоненти зв'язності, а друга — до іншої. Обрані ребра й утворюють коцикл. До нього, зокрема, входить і те ребро, що ми видалили з остовного дерева. Усього, т. ч., маємо $n - 1$ коциклів. У кожному з них є принаймні одне ребро, якого немає в інших: це те ребро, що ми вилучили з остовного дерева для розбиття графа на дві компоненти зв'язності. Тому отримані $n - 1$ коциклів є незалежними. □

У доведенні описаний алгоритм побудови $n-1$ незалежних коциклів.

Але чи достатньо їх? Чи можна сказати, що будь-який інший коцикл у графі можна буде побудувати з них за властивістю 5.6? Як і для циклів, можна стверджувати, що так. Про це каже остання теорема у цій главі.

Теорема 5.12. Будь-який коцикл зв'язного графа $G = (V, E)$ з $|V| = n$, $|E| = m$ є залежним з тими $n-1$ незалежними коциклами, що були побудовані у теоремі 5.11. □

Доведення. Кожен з незалежних коциклів, що ми побудували, утворюється з рівно одного ребра остовного дерева та якихось хорд, причому хорди обираються однозначно. Розглянемо будь-який інший коцикл K_0, якого немає у нашій системі незалежних коциклів. У ньому не можуть бути лише хорди, тому що остовне дерево залишає граф зв'язним. Тому у K_0 є щонайменше два ребра остовного дерева (а може, й більше), та якась кількість хорд. Будуємо K_0. Додамо за логікою XOR ті коцикли системи, в яких є ребра з остовного дерева, що входять у K_0. Всі інші ребра є спільними, бо є хордами дерева. Викинувши зайве, отримаємо K_0. □

За цією теоремою ми можемо стверджувати, що $n-1$ незалежних коциклів дійсно утворюють базис у просторі коциклів: будь-який інший коцикл будується з них шляхом прямого додавання з наступною вибіркою згідно до властивості 5.6. Як кажуть, коциклічний ранг зв'язного графа дорівнює кількості ребер будь-якого його остовного дерева.

Розглянемо як приклад граф з рис. 5.9, *а*. У нього $n = 9$ вершин та $m = 16$ ребер. Згідно з теоремами 5.11, 5.12 він має $n - 1 = 8$ незалежних коциклів. Якщо взяти остовне дерево з рис. 5.10, *а*, то отримаємо незалежні коцикли, представлені на рис. 5.23. Штриховою синьою лінією показане те ребро, що видаляється з остовного дерева та розділяє граф на дві компоненти зв'язності. У червоний колір пофарбовані ребра, що разом з видаленим ребром остовного дерева (синя штрихова лінія) утворюють коцикл. Цей коцикл розрізає граф на дві частини, лінія розрізання позначена зеленим кольором. Отримана система коциклів дійсно є незалежною: у кожному з них є ребро, якого немає в інших коциклах: воно намальоване синьою штриховою лінією. Оскільки знайдених коциклів $n-1 = 8$, то вони утворюють базис: більше незалежних коциклів побудувати не можна.

Фундаментальну систему коциклів у MATLAB будує метод `cocyclebasis`, що міститься в інструментарії Graph Theory Toolbox. Для графа `G` він знаходить базис у комбінаторному просторі коциклів і повертає інформацію про нього в структурі `cocycles` з такими полями:

- `V1` — масив-стовпчик комірок, у якому `V1{k}` містить номери вершин, включених до першої частини (вектори-рядки);
- `V2` — масив-стовпчик комірок, у якому `V2{k}` містить номери вершин, включених до другої частини (вектори-рядки);
- `Edges` — масив-стовпчик комірок, у якому `Edges{k}` містить номери ребер, що утворюють розріз (вектори-рядки).

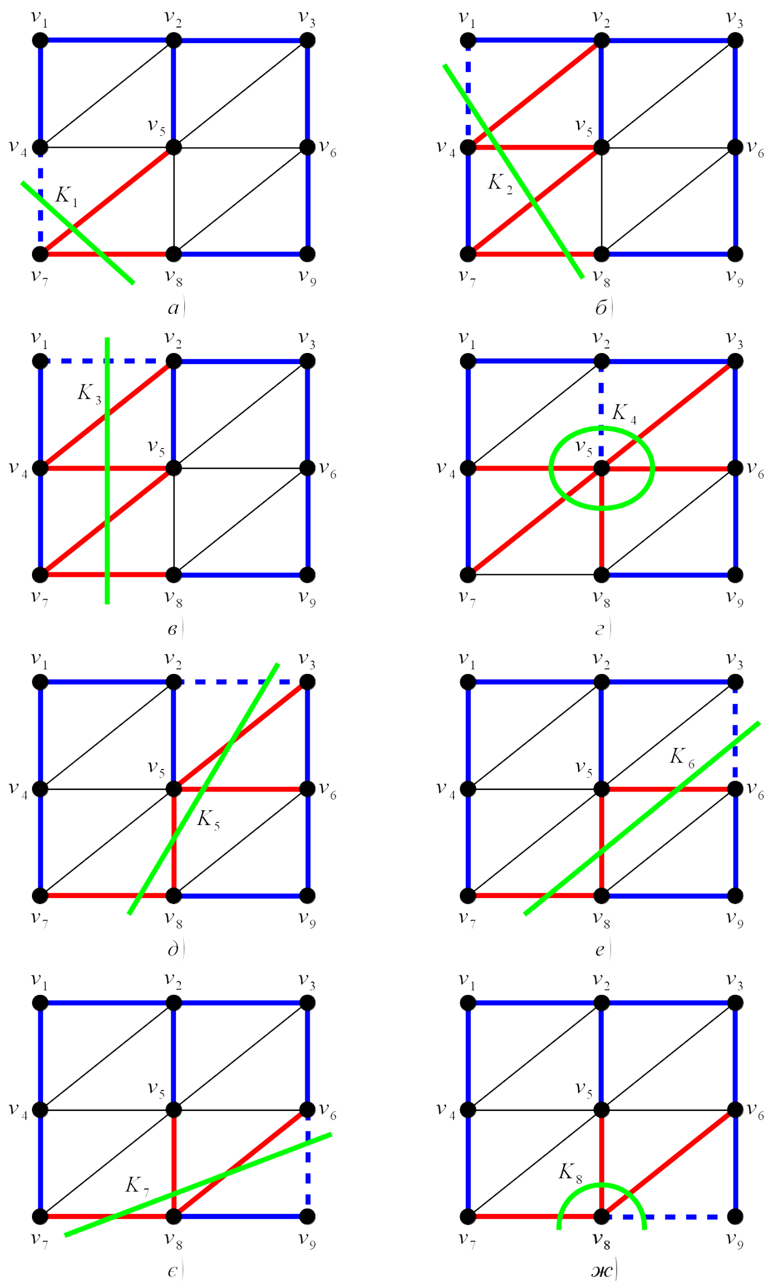


Рис. 5.23. Система незалежних коциклів

Приклад 5.3. Для напівейлерового графу з прикладів 5.1 та 5.2 знайти фундаментальну систему коциклів та нарисувати їх.

```
s = [1 1 1 2 3 2 3 4 5 6 5 6 7 2 4]; % початки ребер
t = [2 3 4 3 4 5 6 7 6 7 8 8 8 7 5]; % кінці ребер
x = [0 0.8 0.9 1 2.2 2.1 2 3]; % координати вершин
y = [1 2 1 0 2 1 0 1];
G = graph(s,t); % створили граф
cocycles = cocyclebasis(G); % знайшли його коцикли
Cc = size(cocycles.Edges,1); % кількість коциклів
for k=1:Cc % малюємо коцикли
  figure % нове вікно фігури
  h = plot(G,"XData",x,"YData",y, ...
    "NodeLabel",{}); % нарисували граф
  highlight(h,"Edges",cocycles.Edges{k},...
    "EdgeColor","k","LineWidth",5) % позначили ребра коциклу
  highlight(h,cocycles.V1{k},"NodeColor","b",...
    "MarkerSize",8) % позначили вершини 1-ї частини
  highlight(h,cocycles.V2{k},"NodeColor","r",...
    "MarkerSize",8) % позначили вершини 2-ї частини
  title("Коцикл № " + k) % заголовок малюнку
  axis("equal") % однаковий масштаб уздовж осей координат
  axis("off") % прибрали осі
  print("Cocycle" + k,"-dpng") % зберегли рисунок у файл
end
```

На рис. 5.24-5.30 показані всі $n - 1 = 8 - 1 = 7$ незалежних коциклів цього графа. □

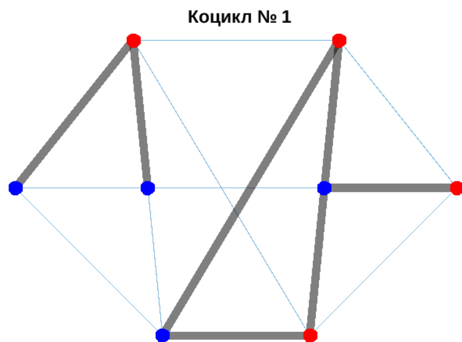


Рис. 5.24. Коцикл №1

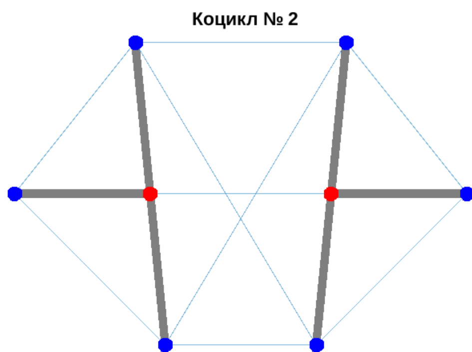


Рис. 5.25. Коцикл №2

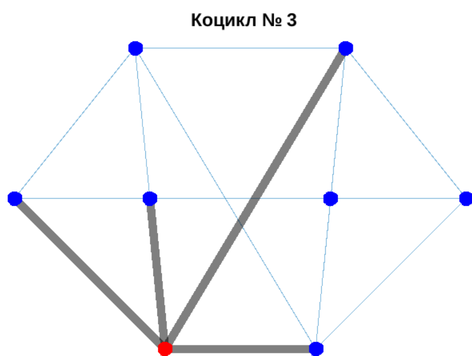


Рис. 5.26. Коцикл №3

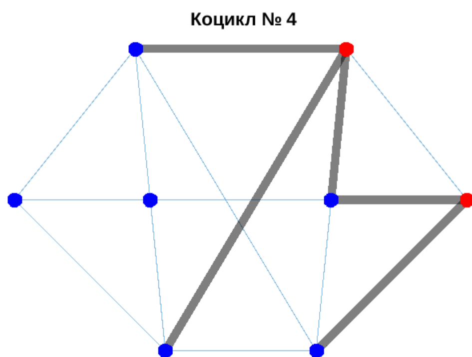


Рис. 5.27. Коцикл №4

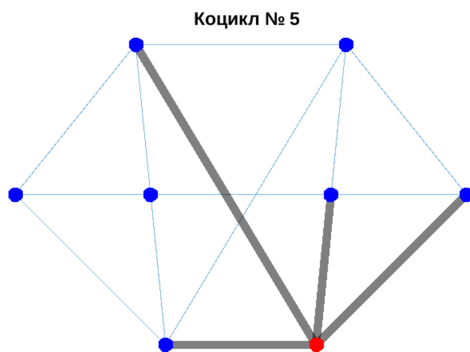


Рис. 5.28. Коцикл №5

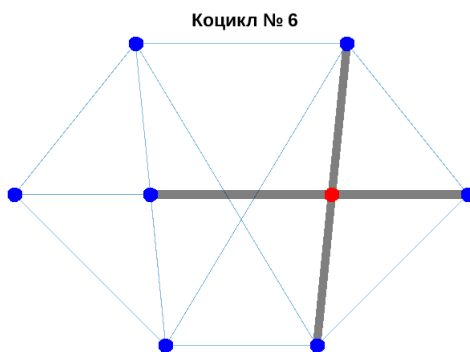


Рис. 5.29. Коцикл №6

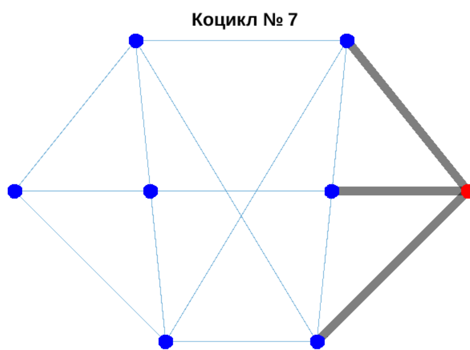


Рис. 5.30. Коцикл №7

Як для циклів, тут можна поставити задачу знаходження базису найменшої ваги.

5.5. Запитання для перевірки

1. Який граф називається ейлеровим? напівейлеровим?
2. Які ви знаєте необхідні та достатні умови ейлеровості? напівейлеровості?
3. Які ви знаєте алгоритми побудови ейдерового циклу (шляху)?
4. Як побудувати ейлерів цикл (шлях) у MATLAB?
5. Який граф називається гамільтоновим? напівгамільтоновим?
6. Які ви знаєте достатні умови гамільтоновості?
7. Які прості цикли називаються незалежними? скільки їх? як їх знайти?
8. Як побудувати фундаментальну систему циклів у MATLAB?
9. Які коцикли називаються незалежними? скільки їх? як їх знайти?
10. Як побудувати фундаментальну систему коциклів у MATLAB?

6. Сильно зв'язані компоненти орграфа

Переходимо до орграфів. Перша задача, яку ми розглянемо — це розбиття орграфа на сильно зв'язані компоненти та їхнє часткове упорядкування. Для цього треба поглянути на орграф з точки зору бінарних відношень.

6.1. Бінарні відношення

Означення 6.1. *Декартовим добутком* (Cartesian product) $A \times B$ двох множин A і B називається множина усіх можливих упорядкованих пар елементів, в яких перший елемент $a \in A$, а другий $b \in B$:

$$A \times B = \{(a, b) : (a \in A) \cap (b \in B)\} . \square \tag{6.1}$$

Для скінченних множин з $|A| = n$, $|B| = m$ елементи $A \times B$ повністю заповнюють клітинки прямокутної таблиці розміром $n \times m$. Прикладом декартового добутку для однозначних додатних цілих чисел є добре відома зі школи таблиця множення.

Означення 6.2. *Бінарним відношенням* (binary relation) називається підмножина декартового добутку якоїсь множини A саму на себе. $\square$

Бінарне відношення зручно позначати знаком між елементами множини. Наприклад, на множинах натуральних, цілих, раціональних та дійсних чисел визначені бінарні відношення $=, \neq, >, <, \geq, \leq$. Тому запис $5 \geq 3$ фактично означає, що елемент декартового добутку $(5, 3)$ належить бінарному відношенню "більше або дорівнює": $(5, 3) \in \geq$. Але запис $5 \geq 3$ зручніший, ніж $(5, 3) \in \geq$, тому зазвичай користуються саме ним. Зворотне відношення не має місця, тому кажуть, що $(3, 5) \notin \geq$ (елемент декартового добутку $(3, 5)$ не належить до бінарного відношення "більше або дорівнює").

З цієї точки зору множину дуг орграфа можна розглядати як бінарне відношення на множині вершин, яке ми назвемо досяжністю (approachability). Коли ми кажемо, що з вершини v_i досягається вершина v_j, то це еквівалентно тому, що існує дуга з v_i у v_j, або бінарне відношення $v_i \to v_j$. І навпаки, вираз "вершина v_j є досяжною з вершини v_i" означає існування дуги у v_j з v_i, або бінарне відношення $v_j \leftarrow v_i$. Будемо вважати, що обидва ці вирази та відношення еквівалентні між собою.

Розглянемо класифікацію бінарних відношень. Тут для їхнього позначення використовується посміхайлик ☺, щоб не прив'язуватися до жодного конкретного бінарного відношення. Елементи множини A позначені a або $a_1, a_2, a_3, \ldots$.

Означення 6.3. Бінарне відношення ☺ $\subseteq A \times A$ на A називається:

- *рефлексивним* (reflexive), якщо $a☺a$ має місце $\forall a \in A$;
- *транзитивним* (transitive), якщо $\forall a_1, a_2, a_3 \in A$ з виконання $a_1☺a_2$ та $a_2☺a_3$ випливає, що $a_1☺a_3$;
- *повним* (total), якщо $\forall a_1, a_2 \in A$ обов'язково виконується або $a_1☺a_2$, або $a_2☺a_1$;
- *антисиметричним* (antisymmetric), якщо з одночасного виконання умов $a_1☺a_2$ та $a_2☺a_1$ слідує, що $a_1 = a_2$ (елементи еквівалентні в сенсі деякого означення, яке ще треба ввести до розгляду);
- *асиметричним* (asymmetric), якщо з виконання $a_1☺a_2$ слідує, що $a_2☺a_1$ не виконується ніколи;
- *симетричним* (symmetric), якщо з виконання $a_1☺a_2$ слідує, що тоді обов'язково виконується $a_2☺a_1$;
- *передупорядкованим*, або *квазіупорядкованим* (preorder, quasiorder), якщо воно рефлексивне та транзитивне;
- *еквівалентним* (equivalence), якщо воно рефлексивне, транзитивне та симетричне;
- *частково упорядкованим* (partial order), якщо воно рефлексивне, транзитивне та антисиметричне;
- *повністю упорядкованим* (total order), якщо воно рефлексивне, транзитивне, повне та антисиметричне. □

Деякі автори розрізняють передупорядковані та квазіупорядковані бінарні відношення. Вони називають бінарне відношення передупорядкованим, якщо воно рефлексивне, транзитивне та повне; і квазіупорядкованим, якщо воно тільки рефлексивне та транзитивне.

Перевірте самостійно, якими є бінарні відношення $=$, $\neq$, $>$, $<$, $\geq$, $\leq$ на множині дійсних чисел.

6.2. Сильно зв'язані компоненти

Розглянемо, які з властивостей бінарних відношень виконуються для орграфів та бінарного відношення $\to$ (досяжність) на множині його вершин V.

Рефлексивність. Навіть якщо у вершині v немає петель, ми можемо вважати, що, якщо ми вже знаходимося у вершині v, то ми її досягли. Тому будемо вважати, що $v \to v$ виконується $\forall v$, і, отже, бінарне відношення $\to$ є рефлексивним.

Транзитивність. Якщо з v_1 досягається v_2, а з v_2 — v_3, то ми можемо стверджувати, що v_3 є досяжною з v_1 (вже не за один, а за два переходи вздовж дуг). Отже, бінарне відношення $\to$ є транзитивним.

Повнота. Не обов'язково! Наприклад, на рис. 6.1, *а* $v_1 \to v_2$, але, навпаки, $(v_2, v_1) \notin \to$.

Антисиметричність. Для перевірки цієї вимоги треба ввести по-

няття еквівалентних вершин.

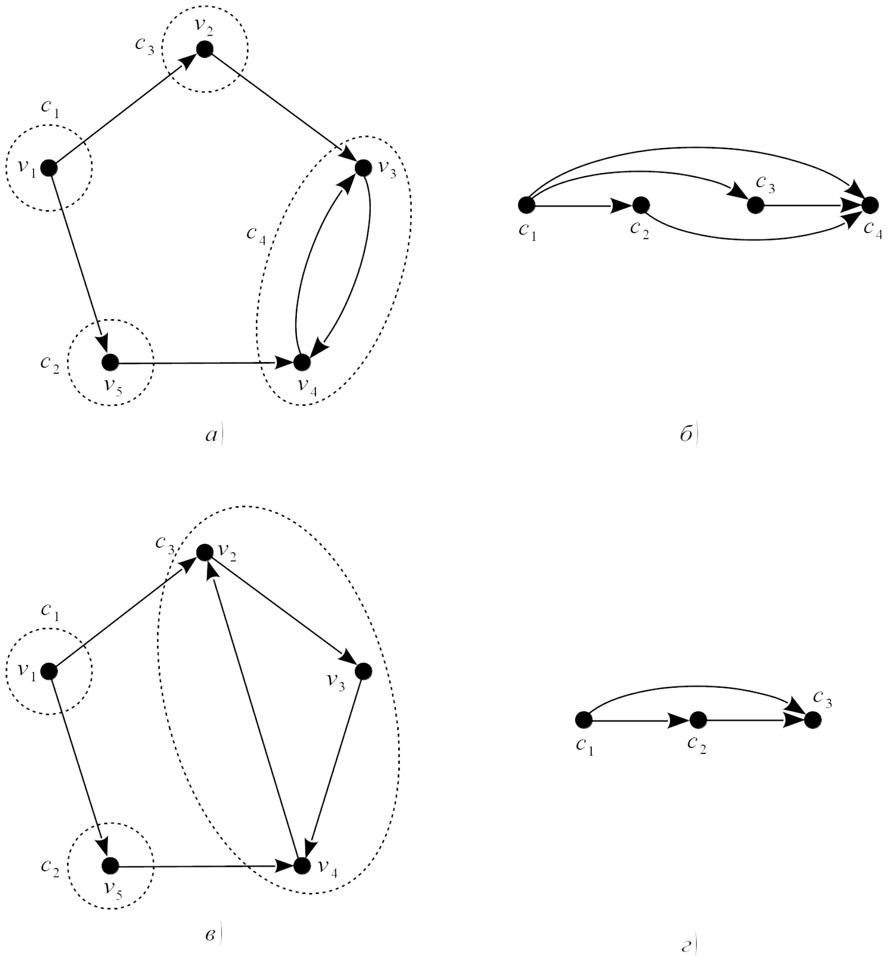


Рис. 6.1. Сильно зв'язані компоненти орграфа

Означення 6.4. Вершини орграфа v_i та v_j називаються *сильно зв'язаними* (strongly connected), якщо існують шляхи вздовж дуг з v_i до v_j та з v_j до v_i. $\square$

Будемо вважати сильно зв'язані вершини еквівалентними в сенсі взаємної досяжності. Тоді бінарне відношення $\to$ є антисиметричним.

Асиметричність. Ця вимога не виконується: існують орграфи, у яких є взаємно досяжні (сильно зв'язані) вершини.

Симетричність. Теж не виконується, оскільки можуть існувати орграфи, у яких $v_i \to v_j$, але $(v_j, v_i) \notin \to$. Приклад такого орграфа — на рис. 6.1, *а*.

Отже, множина вершин орграфа з введеним на ній бінарним відношенням $\to$ (досяжність) та поняттям еквівалентності (взаємна досяжність, сильна зв'язаність) є *частково упорядкованою*, оскільки вона рефлексивна, транзитивна та антисиметрична.

З теорії множин відомо, що, ввівши поняття еквівалентності, ми можемо розбити будь-яку множину (у нашому випадку множину вершин орграфа) на класи еквівалентності, в кожен з яких потрапляють лише еквівалентні (у нас взаємно досяжні) вершини. Інший важливий висновок теорії множин: для частково упорядкованих множин їхні класи еквівалентності також частково упорядковуються.

Означення 6.5. *Компонентою сильної зв'язності* орграфа (strongly connected component) називається підмножина вершин, що утворює клас еквівалентності за досяжністю. □

На рис. 6.1, *а* показаний як приклад орграф з п'ятьма вершинами. Вони розбиваються на чотири компоненти сильної зв'язності, які обведені замкненими штриховими лініями. На рис. 6.1, *б* зображено їхнє часткове упорядкування: класи c_i розташовані лінійно, і відношення досяжності між ними (стрілки) спрямовані тільки праворуч. Але упорядкування тут дійсно часткове: для класів c_2 та c_3 бінарне відношення $\to$ не визначене в жоден бік. Їх можна було б поміняти на рисунку місцями, і при цьому упорядкування збереглося б.

Означення 6.6. *Орієнтований цикл* (орцикл, oriented cycle) — це послідовність дуг орграфа, в якій кожна наступна виходить з вершини, в яку входить попередня, а остання входить у ту вершину, з якої виходить перша. □

Означення 6.7. Орграф називається *ациклічним* (acyclic digraph), якщо в ньому відсутні орцикли. □

У кожній компоненті сильної зв'язності орграфа є принаймні один орцикл. Дійсно: у сильно зв'язаній компоненті з кожної вершини можна дістатися до кожної іншої, а потім повернутися до початкової, тому кожна пара вершин входить до якогось орциклу. І навпаки, орграф, утворений із сильно зв'язаних компонент, є ациклічним. Якщо б у нього були орцикли, то можна було б об'єднати кілька компонент сильної зв'язності в одну, як на рис. 6.1, *в*.

Означення 6.8. *Матрицею досяжності* (connectivity matrix) орграфа G називається булівська або бінарна матриця $\boldsymbol{R}$ розміром $n \times n$, кожен елемент якої $r_{ij} = \text{true}$ (або $r_{ij} = 1$) тоді й тільки тоді, коли з v_i виходить дуга у v_j, і $r_{ij} = \text{false}$ (або $r_{ij} = 0$) в усіх інших випадках. На

головній діагоналі можна ставити 0 або 1 в залежності від постановки задачі. □

Матриця досяжності відрізняється від матриці суміжності вершин орграфа тільки відсутністю −1. У MATLAB матриця досяжності орграфа будується за допомогою методу `adjacency` (див. приклад 1.10).

Цю матрицю можна задавати також для орграфа компонент сильної зв'язності. У цьому випадку вона буде мати розміри $N \times N$, де N — кількість компонент сильної зв'язності ($N \leq n$). Часткове упорядкування компонент сильної зв'язності полягає в тому, що треба так перенумерувати компоненти, щоб нижче головної діагоналі залишилися лише 0 (false). Так, для часткового упорядкування на рис. 6.1, *б* ця матриця має вигляд:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.2}$$

Тут на головній діагоналі проставлені одиниці, щоб підкреслити рефлексивність бінарного відношення $\rightarrow$. Якщо ми тепер замінимо дугу $e_{4,3}$ на $e_{4,2}$ (рис. 6.1, *в*), то замість чотирьох компонент сильної зв'язності отримаємо тільки три. Їхнє упорядкування з часткового перетворюється вже на повне (рис. 6.1, *г*), а його матриця досяжності буде мати вигляд:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}. \tag{6.3}$$

На основі розбиття множини вершин на сильно зв'язані компоненти та їхнього часткового упорядкування можна розв'язувати різні задачі. Цей алгоритм можна використовувати, наприклад, у соціологічних дослідженнях. За його допомогою легко виявляються лідери та групи впливу в різних об'єднаннях громадян: робочих колективах, шкільних класах, студентських групах, керівництві політичних партій, клубах за інтересами.

Інша цікава проблема, що може бути розв'язана на основі цієї задачі — це стягування орграфа та зменшення кількості його вершин. Взаємно досяжні вершини об'єднуються в одну, а дуги між ними видаляються. Далі можна ставити задачу мінімізації кількості дуг з тим, щоб залишити отримане часткове упорядкування тощо. Зараз ми розглянемо лише два перші етапи: розбиття множини вершин на сильно зв'язані компоненти та часткове упорядкування цих компонент.

6.3. Визначення сильно зв'язаних компонент

Нехай ми побудували матрицю досяжності орграфа $\boldsymbol{R}$ за означенням 6.8. У ній кожен елемент $r_{ij} = \text{true}$ тоді й тільки тоді, коли в орграфі є дуга з v_i у v_j. На головній діагоналі ставимо true (рефлексивність). Припустимо, що дуги з якоїсь v_i у якусь v_j немає: $r_{ij} = \text{false}$. Як тепер визначити, чи можемо ми дістатися з вершини v_i до вершини v_j за два кроки? Вочевидь, це можливо, якщо існує якась третя вершина v_k така, що є дуги з v_i у v_k і водночас з v_k у v_j, тобто існує якесь $k \in [1, n]$, для якого $(r_{ik} = \text{true}) \wedge (r_{kj} = \text{true})$. Щоб це перевірити, треба взяти всі елементи i-го рядка матриці $\boldsymbol{R}$ та перемножити їх за булівськими правилами на відповідні елементи j-го стовпчика цієї ж матриці, а потім скласти отримані булівські змінні. Якщо хоча б при одному якомусь k було $(r_{ik} = \text{true}) \wedge (r_{kj} = \text{true})$, то в результаті отримаємо true. А якщо немає жодної вершини v_k, для якої виконується $(r_{ik} = \text{true}) \wedge (r_{kj} = \text{true})$, то в результаті буде false. Так можна перевірити будь-яку пару вершин, взявши потрібні рядки та стовпці матриці $\boldsymbol{R}$. Але ж поелементне множення кожного рядка на кожний стовпчик з наступним додаванням — це просто множення матриць!

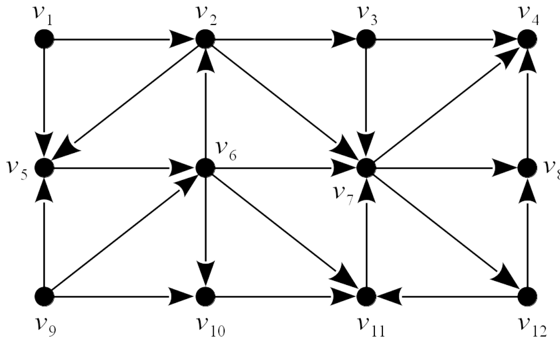


Рис. 6.2. Приклад орграфа

Отже, приходимо до висновку: матриця $\boldsymbol{R}^2$ є матрицею досяжності за два кроки. При цьому, оскільки на головній діагоналі ми поставили true, то в $\boldsymbol{R}^2$ досяжність за один крок теж не втрачається. Адже можна вважати, що один крок — це насправді два: перший крок у цю саму вершину, а другий у потрібну (або навпаки). Отже, $\boldsymbol{R}^2$ є матрицею досяжності орграфа за один або два кроки. За індукцією: $\boldsymbol{R}^3$ є матрицею досяжності за один, два або три кроки, $\boldsymbol{R}^4$ — за 1, 2, 3 або 4 кроки тощо. Тобто можна просто множити матрицю $\boldsymbol{R}$ саму на себе потрібну кількість разів. Де ж зупинитися? Будь-який орцикл не може мати більше, ніж $\min(m, n)$

дуг. Це й є максимальний теоретично можливий ступінь матриці $\boldsymbol{R}$. Але орцикли можуть мати й менший розмір. Тому не обов'язково треба множити матрицю $\boldsymbol{R}$ саму на себе $\min(m, n)$ разів. Можна просто перевіряти: якщо після чергового множення матриця не змінилася: $\boldsymbol{R}^{k+1} = \boldsymbol{R}^k$, то далі підвищувати ступінь немає сенсу.

Можна ще більше прискорити цей процес, якщо на кожному кроці множити $\boldsymbol{R}^k$ не на $\boldsymbol{R}$, а на саму себе, тобто на $\boldsymbol{R}^k$. Тоді ми послідовно обчислюємо не $\boldsymbol{R}^2$, $\boldsymbol{R}^3$, $\boldsymbol{R}^4$, ..., а $\boldsymbol{R}^2$, $\boldsymbol{R}^4$, $\boldsymbol{R}^8$, ...

Розглянемо як приклад орграф на рис. 6.2. У нього $n = 12$; $m = 23$. Його матриця досяжності:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.4}$$

Обчислюємо $\boldsymbol{R}^2$:

$$\boldsymbol{R}^2 = \boldsymbol{R} \cdot \boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.5}$$

Оскільки $\boldsymbol{R}^2 \neq \boldsymbol{R}$, продовжуємо підводити до квадрату:

$$\boldsymbol{R}^4 = \boldsymbol{R}^2 \cdot \boldsymbol{R}^2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.6}$$

Знову маємо: $\boldsymbol{R}^4 \neq \boldsymbol{R}^2$. Ще раз підводимо до квадрату:

$$\boldsymbol{R}^8 = \boldsymbol{R}^4 \cdot \boldsymbol{R}^4 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.7}$$

Тут вже є співпадіння $\boldsymbol{R}^8 = \boldsymbol{R}^4$. Далі множити матрицю саму на себе немає сенсу: результат буде таким самим. Отже, ми отримали матрицю досяжності за будь-яку кількість кроків, тобто матрицю транзитивної досяжності. Позначимо її буквою $\boldsymbol{D}$:

$$
\boldsymbol{D} = \boldsymbol{R}^8 = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{pmatrix}. \qquad (6.8)
$$

Описані вище дії можна реалізувати, наприклад, у вигляді функції так, як показано нижче. Ця функція за заданою матрицею досяжності `R` будує транзитивну матрицю досяжності `D`. Для її роботи потрібні три такі допоміжні функції:

- `B = SquareMatrix(A)` — множить саму на себе квадратну булівську матрицю `A` та повертає результат — квадратну булівську матрицю `B` такого самого розміру;
- `B = CopyMatrix(A)` — копіює квадратну булівську матрицю `A` та повертає результат — квадратну булівську матрицю `B` такого самого розміру;
- `Eq = CompareMatrix(A,B)` — порівнює квадратні булівські матриці `A` і `B` та повертає результат — булівську змінну `Eq`, яка дорівнює true, якщо матриці `A` і `B` співпадають, і false, якщо ні.

Реалізація цих функцій залежить від конкретної мови програмування. Ось алгоритм побудови матриці транзитивної досяжності на псевдокоді.

```
function D = GetTransAppr(R) {будує матрицю транз. досяжності D}
begin {GetTransAppr}
  repeat {множимо матрицю досяжності саму на себе}
  begin {repeat}
    D = SquareMatrix(R)
    Eq = CompareMatrix(R,D)
    R = CopyMatrix(D)
  end {repeat}
  while (not Eq) {доти, доки вона змінюється}
  return D {повертаємо знайдену матрицю D}
end {GetTransAppr}
```

В отриманій матриці транзитивної досяжності $\boldsymbol{D}$ буде $d_{ij} = d_{ji} =$

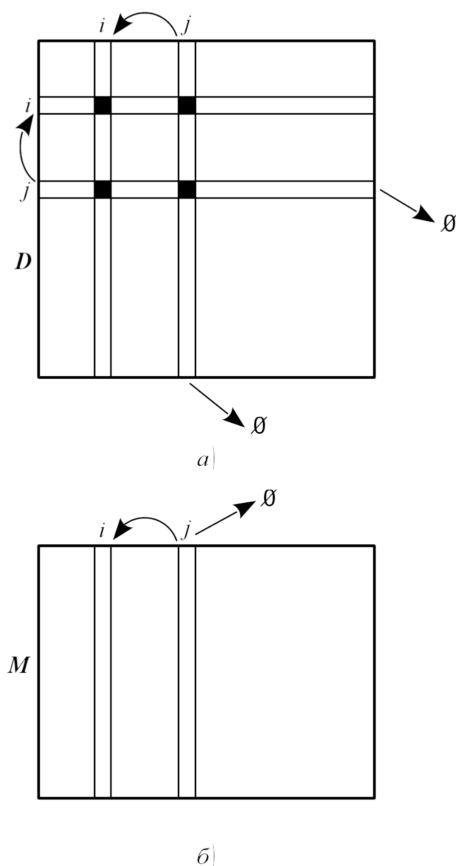


Рис. 6.3. Об'єднання вершин сильно зв'язаної компоненти

true тоді й тільки тоді, коли v_i та v_j є взаємно досяжними (тобто потрапляють в одну компоненту сильної зв'язності). Нам треба об'єднати їх в один клас еквівалентності за досяжністю. Таке об'єднання полягає в тому, що всі вершини, як були досяжні з v_i або v_j, будуть досяжними і з їхнього об'єднання. І навпаки: отримане об'єднання буде досяжним з усіх вершин, з яких були досяжні і v_i, і v_j. Простіше за все цього домогтися шляхом додавання (об'єднання) відповідних рядків та стовпців матриці $\boldsymbol{D}$, як показано на рис. 6.3, *а*. Як тільки ми виявимо у матриці $\boldsymbol{D}$ пару симетричних істинних елементів $d_{ij} = d_{ji} = \text{true}$, треба виконати такі дії:

- додати до i-го стовпчика матриці $\boldsymbol{D}$ j-й стовпчик, і видалити j-й стовпчик;
- додати до i-го рядка матриці $\boldsymbol{D}$ j-й рядок, і видалити j-й рядок.

Після такої операції матриця досяжності зменшить свої розміри на одиницю. Далі ми шукаємо наступну пару симетричних істинних елементів і повторюємо процес. Так робимо доти, доки вдається знайти пару $d_{ij} = d_{ji} = \text{true}$. Якщо всі такі пари вичерпані, ітераційний процес закінчується: всі взаємно досяжні вершини об'єднані в класи еквівалентності за досяжністю, тобто у сильно зв'язані компоненти.

Але це ще не все. Нам треба десь зберігати номери вершин, що входять до тієї чи іншої компоненти сильної зв'язності. Для цього перед ітераціями створимо ще одну матрицю $\boldsymbol{M}$ — одиничну булівську матрицю розміром $n \times n$. У кожному стовпці істинні елементи будуть відповідати номерам вершин, що входять у цю компоненту. Перед початком ітерацій у нас є n компонент, у кожну з яких входить лише одна вершина, тому початкова $\boldsymbol{M}$ — одинична. Як тільки знаходиться пара $d_{ij} = d_{ji} = \text{true}$, то крім описаних вище двох дій, виконуємо ще й третю:

- додати до i-го стовпчика матриці $\boldsymbol{M}$ j-й стовпчик, і видалити j-й стовпчик.

У результаті цієї дії у i-му стовпчику матриці $\boldsymbol{M}$ будуть збережені номери вершин, які об'єднуються в одну компоненту сильної зв'язності. Ця дія показана на рис. 6.3, *б*.

Додавання та викидання стовпців та рядків можна реалізувати у різний спосіб. Тут усе залежить від мови програмування. Якщо можна швидко та ефективно реалізувати викидання рядка та (або) стовпчика з матриці безпосередньо, слід так і робити. Якщо ж такі дії виконуються тривалий час, то, можливо, ефективнішим буде застосування множення на відповідну матрицю деформування-додавання-стискання (її інколи називають прокрустовою матрицею). Наприклад, у матриці $\boldsymbol{D}$ з (6.8): $d_{2,5} = d_{5,2} = \text{true}$. Прокрустова матриця $\boldsymbol{T}$ будується так: береться одинична матриця, і у ній до другого стовпчика додається п'ятий, а потім п'ятий стовпчик видаляється. Тобто вона буде мати розмір 12×11:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.9}$$

На перший погляд здається, що цей спосіб не швидше попереднього, бо теж треба викидати стовпчик матриці. Але прокрустова матриця будується заздалегідь, за заданими розмірами, і фактично викидати стовпчик після побудови матриці не треба.

З використанням прокрустової матриці $\boldsymbol{T}$ (6.9) об'єднання v_2 та v_5 в одну компоненту сильної зв'язності (реалізація дій з рис. 6.3) виглядає так:

$$\boldsymbol{D}_1 = \boldsymbol{T}^T \boldsymbol{D} \boldsymbol{T} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}; \qquad (6.10)$$

$$\boldsymbol{M}_1 = \boldsymbol{M} \boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (6.11)$$

Після першого кроку матриця $\boldsymbol{D}_1$ стала матрицею досяжності компонент сильної зв'язності, збережених у матриці $\boldsymbol{M}_1$. Далі дивимося тепер вже на матрицю $\boldsymbol{D}_1$: чи є в ній $d_{ij} = d_{ji} = \text{true}$? Є така пара: це також $d_{2,5} = d_{5,2} = \text{true}$, тільки два та п'ять тут вже нові номери. Тому знову або додаємо та викидаємо рядки та стовпчики (якщо це простіше), або будуємо прокрустову матрицю $\boldsymbol{T}_1$. Тепер вона вже буде мати розмір 11×10. Вона утворюється з одиничної додаванням п'ятого стовпчика до другого та викиданням п'ятого стовпчика:

$$
\boldsymbol{T}_1 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}. \qquad (6.12)
$$

Подальші дії цієї ітерації:

$$
\boldsymbol{D}_2 = \boldsymbol{T}_1^T \boldsymbol{D}_1 \boldsymbol{T}_1 = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1
\end{pmatrix}; \qquad (6.13)
$$

$$
\boldsymbol{M}_2 = \boldsymbol{M}_1 \boldsymbol{T}_1 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}. \qquad (6.14)
$$

Як бачимо, об'єдналися v_2, v_5 та v_6. Далі можна об'єднувати за $d_{5,9} = d_{9,5} = \text{true}$, і т. д. Після останнього кроку отримаємо такі матриці $\boldsymbol{D}$ та $\boldsymbol{M}$ (нижні індекси в них опускаємо):

$$\boldsymbol{D} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}; \quad (6.15)$$

$$\boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \quad (6.16)$$

Усе, що можна, об'єднали. У матриці $\boldsymbol{D}$ більше немає жодної пари елементів, для яких $d_{ij} = d_{ji} = \text{true}$. Матриця $\boldsymbol{D}$ стала матрицею досяжності вже не окремих вершин, а компонент сильної зв'язності. А у матриці $\boldsymbol{M}$ бачимо, які саме вершини об'єдналися у сильно зв'язані компоненти: це $\{v_2, v_5, v_6\}$ та $\{v_7, v_{11}, v_{12}\}$. Усі інші вершини залишилися в однині: кожна з них утворює свою компоненту сильної зв'язності.

Реалізація описаних вище дій — у наступному алгоритмі. В ньому наведена процедура, яка задану матрицю транзитивної досяжності вершин `D` стягує до матриці транзитивної досяжності компонент сильної зв'язності. Крім того, вона перетворює одиничну булівську матрицю `M` на матрицю компонент сильної зв'язності. Допоміжні процедури та функції, які потрібні:

- `(i,j) = GetPairTrueElem(A)` — повертає номер рядка і стовпчика пари симетричних істинних елементів булівської матриці `A`; якщо таких елементів немає, повертаються нулі;
- `AddDelRow(A,i,j)` — додає у булівській матриці `A` до `i`-го рядка `j`-й та видаляє `j`-й рядок;
- `AddDelCol(A,i,j)` — додає у булівській матриці `A` до `i`-го стовпчика `j`-й та видаляє `j`-й стовпчик.

Ось сам алгоритм:

```
procedure GetStrongComp(D,M) {будує комп-и сильної зв'язності}
begin {GetStrongComp}
  (i,j) = GetPairTrueElem(D) {пара взаємно досяжних вершин}
  while (i>0) {поки є така пара}
  begin {while}
    AddDelRow(D,i,j)
    AddDelCol(D,i,j)
    AddDelCol(M,i,j)
    (i,j) = GetPairTrueElem(D) {пара взаємно досяжних вершин}
  end {while}
end {GetStrongComp}
```

У MATLAB є вбудований метод `conncomp`, що знаходить компоненти сильної зв'язності орграфа. Для звичайного графа він знаходить компоненти зв'язності. Метод повертає такі вихідні дані:

- `bins` — вектор-рядок довжини n (кількість вершин); кожен його елемент містить номер компоненти зв'язності для цієї вершини;
- `binsizes` — вектор-рядок довжини, що дорівнює кількості компонент зв'язності; кожен його елемент містить розмір компоненти зв'язності.

Приклад 6.1. Знайти та нарисувати компоненти сильної зв'язності заданого орграфа.

```
s = [1 3 4 5 6 2 8 3 9 9 10 7 8 8 10 11 7 ...
  13 14 15 12 13 13 14 15 16 12 13 20 17 17 18 19 19 ...
  21 17 18 22 18 19 20 21 22 23 24 24]; % початки дуг
t = [2 2 3 4 1 7 2 8 4 5 5 6 7 9 9 6 12 ...
  8 9 10 11 12 14 13 14 11 17 18 15 16 18 17 18 20 ...
  16 22 22 18 23 24 25 22 21 24 23 25]; % кінці дуг
x = reshape(repmat(0:4,1,5),25,1); % абсциси вершин
y = reshape(repmat(4:-1:0,5,1),25,1); % ординати вершин
G = digraph(s,t); % створили орграф
[b,bs] = conncomp(G); % знаходимо комп. сильної зв'язності
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",b,"MarkerSize",6, ...
  "NodeColor",cm(round((b-1)/(length(bs)-1)*255+1),:))
title("Кількість компонент сильної зв'язності = " + ...
  length(bs)) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
```

```
print("Conncomp","-dpng") % зберегли рисунок у файл
```

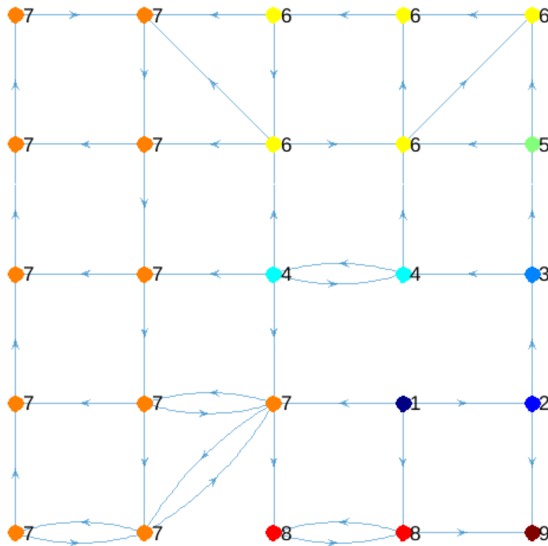


Рис. 6.4. Компоненти сильної зв'язності орграфа

На рис. 6.4 біля кожної вершини показаний номер її компоненти зв'язності. □

6.4. Часткове упорядкування сильно зв'язаних компонент

Задача лінійного упорядкування об'єктів є цікавою сама по собі. Найпростіший та очевидний шлях її розв'язання — це порівнювати кожен елемент з будь-яким іншим. Такий алгоритм вимагає $\frac{n(n-1)}{2}$ операцій порівняння. Можна також додавати по одному елементу у вже упорядковану послідовність. Якщо частина елементів упорядкована, то новий елемент, що додається, не обов'язково порівнювати з усіма попередніми. Ми можемо порівнювати його з першим, другим тощо, поки не знайдемо його місце у послідовності. У найгіршому випадку будемо мати ті самі $\frac{n(n-1)}{2}$ операцій порівняння, але в середньому їх буде менше. В ідеальному випадку взагалі після першого ж порівняння новий елемент відразу стане на своє місце. Але й цей алгоритм не найкращий: можна застосувати дискретний варіант методу ділення навпіл. Коли в упорядковану послідовність

додається новий елемент, його в першу чергу порівнюють з середнім елементом послідовності (або з одним з двох середніх, якщо їхня кількість парна). Тим самим ми визначаємо половину, в якій і буде знаходитися новий елемент. Далі процес ділення цієї половини навпіл продовжується доти, доки положення нового елемента, що додається, не буде визначене однозначно.

На жаль, у нашому випадку ці алгоритми не будуть працювати, оскільки упорядкування у нас тільки часткове. Деякі сильно зв'язані компоненти не є порівнюваними. Але, як виявляється, нам взагалі не треба порівнювати між собою сильно зв'язані компоненти, оскільки результат порівняння у нас вже є: це матриця досяжності компонент $\boldsymbol{D}$. Нам треба тільки розташувати компоненти у потрібному порядку. Порядок визначається тим, що в матриці $\boldsymbol{D}$ не повинно бути істинних елементів нижче головної діагоналі. Тому, якщо такий елемент d_{ij} виявиться, виконуємо такі дії:

- міняємо місцями i-й та j-й стовпці матриці $\boldsymbol{D}$;
- міняємо місцями i-й та j-й рядки матриці $\boldsymbol{D}$;
- міняємо місцями i-й та j-й стовпці матриці $\boldsymbol{M}$.

У результаті ми замінимо місцями i-у та j-у компоненти сильної зв'язності та відповідним чином скоригуємо матрицю досяжності.

Цей процес треба продовжувати доти, доки у матриці $\boldsymbol{D}$ є істинні елементи нижче головної діагоналі. Як і для стягування матриць, для перестановки рядків та стовпців можна їх міняти місцями безпосередньо, а можна скористатися матрицею перестановок. Наприклад, у матриці $\boldsymbol{D}$ у формулі (6.5) є $d_{7,2} = \text{true}$. Треба поміняти місцями другий та сьомий стовпці та рядки в матриці $\boldsymbol{D}$, а також другий та сьомий стовпці в матриці $\boldsymbol{M}$. Створюємо матрицю перестановок $\boldsymbol{T}$: беремо одиничну матрицю та міняємо в ній другий та сьомий стовпці:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.17}$$

Тепер за допомогою відомих формул лінійної алгебри типу (1.19-1.21) переставляємо рядки та стовпці, як потрібно:

$$\boldsymbol{D}_1 = \boldsymbol{T}^T \boldsymbol{D} \boldsymbol{T} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}; \qquad (6.18)$$

$$\boldsymbol{M}_1 = \boldsymbol{M} \boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \qquad (6.19)$$

Перевіряємо далі. У матриці $\boldsymbol{D}_1$ є істинний елемент нижче головної діагоналі: $d_{7,3} = \text{true}$. Матриця перестановки третього та сьомого стовпців:

$$\boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (6.20)$$

Відповідна перестановка рядків та стовпців:

$$
\boldsymbol{D}_2 = \boldsymbol{T}_1^T \boldsymbol{D}_1 \boldsymbol{T}_1 = \begin{pmatrix}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1
\end{pmatrix}; \qquad (6.21)
$$

$$
\boldsymbol{M}_2 = \boldsymbol{M}_1 \boldsymbol{T}_1 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{pmatrix}. \qquad (6.22)
$$

Як бачимо, одиниці нижче головної діагоналі у матриці $\boldsymbol{D}$ поступово переповзають вгору-праворуч. Через ще кілька кроків ми отримаємо таке (нижні індекси опущені):

$$
\boldsymbol{D} = \begin{pmatrix}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}; \qquad (6.23)
$$

$$
\boldsymbol{M} = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}. \tag{6.24}
$$

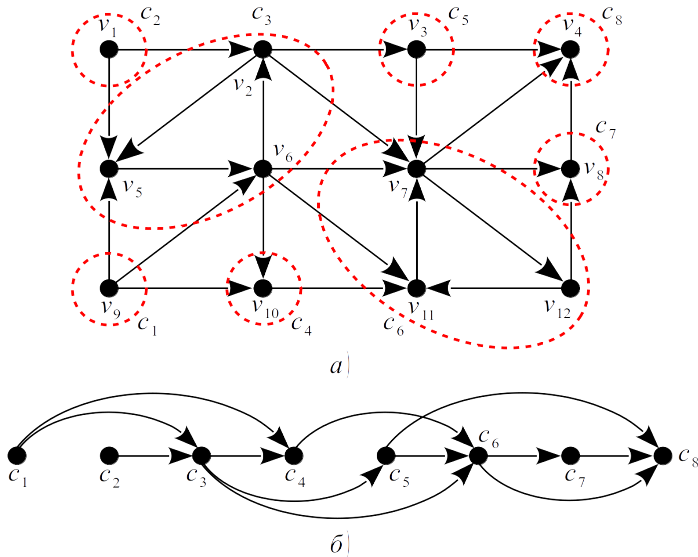


Рис. 6.5. Сильно зв'язані компоненти орграфа (а)
та їхнє часткове упорядкування (б)

У матриці $\boldsymbol{M}$ (6.24) — 8 стовпців. Це означає, що 12 вершин орграфа розбилися на 8 компонент сильної зв'язності. У першу компоненту потрапила тільки одна вершина v_9, у другу — теж тільки одна v_1, у третій — три вершини: v_2, v_5 та v_6, і т. д. Компоненти сильної зв'язності у матриці $\boldsymbol{M}$ перенумеровані за своїм частковим упорядкуванням. Саме часткове упорядкування записане у матриці $\boldsymbol{D}$ (6.23). Одиниця означає досяжність з якоїсь компоненти до якоїсь, а нуль — відсутність досяжності. Зокрема,

$d_{1,2} = 0$ означає, що з першої компоненти $\{v_9\}$ друга компонента $\{v_1\}$ не досягається. Так само, з четвертої компоненти $\{v_{10}\}$ не досягається п'ята $\{v_3\}$. В усіх інших випадках компонента з більшим номером завжди досягається з компоненти з меншим номером. На рис. 6.5, *а* показаний наш орграф з рис. 6.2, на якому позначені компоненти сильної зв'язності, а на рис. 6.5, *б* — їхнє часткове упорядкування.

Щоб не затіняти, на рис. 6.5, *б* показані лише реальні дуги між компонентами, а не транзитивне упорядкування, яке записане у формулі (6.23). Тобто, наприклад у (6.23) $d_{1,5} = 1$, що означає досяжність компоненти c_5 з c_1. На рис. 6.5, *б* ця досяжність теж є, але через c_3.

Видно, що між компонентами c_1 та c_2 упорядкування відсутнє, так само як і між c_4 та c_5.

Реалізація часткового упорядкування компонент сильної зв'язності — в алгоритмі, що наведений нижче. Для його роботи потрібні такі додаткові функції та процедури:

- `(i,j) = GetTrueDownElem(A)` — для булівської квадратної матриці `A` повертає номер рядка та стовпця істинного елемента нижче головної діагоналі (`i>j`); якщо таких елементів немає, повертає нулі;
- `ChangeRows(A,i,j)` — у булівській матриці `A` міняє місцями рядки з номерами `i` та `j`;
- `ChangeCols(A,i,j)` — у булівській матриці `A` міняє місцями стовпці з номерами `i` та `j`.

Ось як виглядає цей алгоритм.

```
procedure PartOrder(D,M) {часткове упорядкування компонент}
begin {PartOrder}
  (i,j) = GetTrueDownElem(D) {неупорядкована компонента}
  while (i>0) {поки є такі}
  begin {while}
    ChangeRows(D,i,j)
    ChangeCols(D,i,j)
    ChangeCols(M,i,j)
    (i,j) = GetTrueDownElem(D) {неупорядкована компонента}
  end {while}
end {PartOrder}
```

Описаний вище алгоритм може застосовуватися й до визначення компонент зв'язності звичайного (неорієнтованого) графа. Для цього треба додати до орграфа m дуг, зворотних до кожної наявної. Тоді кожна пара досяжних в один бік вершин орграфа стане взаємно досяжною парою вершин, тобто буде входити в одну компоненту сильної зв'язності орграфа. І всі тепер вже взаємно досяжні вершини теж увійдуть до однієї компоненти сильної зв'язності. Отже, знайдені компоненти сильної зв'язності

доповненого зворотними дугами орграфа будуть співпадати з компонентами зв'язності графа.

Розглянутий у прикладі 6.1 метод `conncomp` знаходить сильно зв'язані компоненти орграфа та упорядковує їх, але не повертає матрицю транзитивної досяжності (6.23). Щоб отримати ще й її, можна скористатися методом `orderstrongcomp`, що міститься в пакеті Graph Theory Toolbox. Цей метод для орграфа `G` повертає ті ж самі вихідні параметри `bins` та `binsizes`, що й метод `conncomp`, і додатково повертає третій вихідний параметр `ord`, що містить матрицю транзитивної досяжності $\boldsymbol{D}$ (6.23).

Приклад 6.2. Для орграфа з рис. 6.5 знайти та нарисувати компоненти сильної зв'язності, а також побудувати матрицю транзитивної досяжності цих компонент.

```
s = [1 2 3 1 2 6 2 3 7 8 5 6 7 9 9 6 6 11 ...
  7 12 9 10 12]; % початки дуг
t = [2 3 4 5 5 2 7 7 4 4 6 7 8 5 6 10 11 7 ...
  12 8 10 11 11]; % кінці дуг
x = [0 1 2 3 0 1 2 3 0 1 2 3]; % абсциси вершин
y = [2 2 2 2 1 1 1 1 0 0 0 0]; % ординати вершин
G = digraph(s,t); % створили орграф
[b,bs,ord] = orderstrongcomp(G); % розв'язали задачу
disp("Сильно зв'язані компоненти:")
disp(" N    вершини")
for k1=1:length(bs)
  fprintf("%2.0f    ",k1)
  fprintf("%d  ",find(b==k1))
  fprintf("\n")
end
fprintf("\nЧасткове упорядкування компонент:\n  ")
fprintf("%3.0f",1:size(ord,2))
fprintf("\n")
for k1=1:size(ord,1) % матриця часткового упорядкування
  fprintf("%2.0f ",k1)
  fprintf(" %1.0f ",ord(k1,:))
  fprintf("\n")
end
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",b,"MarkerSize",6, ...
  "NodeColor",cm(round((b-1)/(length(bs)-1)*255+1),:))
title("Кількість компонент сильної зв'язності = " + ...
```

```
  length(bs)) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("Orderstrongcomp","-dpng") % зберегли рисунок у файл
```

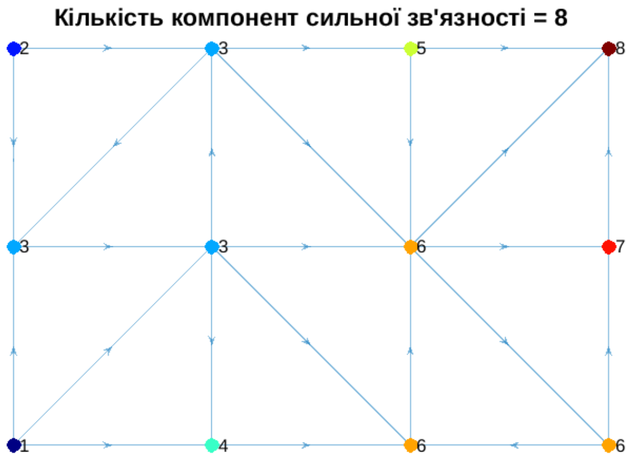


Рис. 6.6. Компоненти сильної зв'язності орграфа з рис. 6.5

```
Сильно зв'язані компоненти:
 N    вершини
 1    9
 2    1
 3    2  5  6
 4    10
 5    3
 6    7  11  12
 7    8
 8    4

Часткове упорядкування компонент:
    1  2  3  4  5  6  7  8
 1  1  0  1  1  1  1  1  1
 2  0  1  1  1  1  1  1  1
 3  0  0  1  1  1  1  1  1
 4  0  0  0  1  0  1  1  1
 5  0  0  0  0  1  1  1  1
```

```
  6   0   0   0   0   0   1   1   1
  7   0   0   0   0   0   0   1   1
  8   0   0   0   0   0   0   0   1
```

На рис. 6.6 біля кожної вершини показаний номер її компоненти зв'язності. Як бачимо, отримані результати співпадають з (6.23) та даними з рис. 6.5 □

6.5. Запитання для перевірки

1. Класифікуйте бінарні відношення $>$, $<$, $\geq$, $\leq$, $=$ та $\neq$ на множинах цілих, раціональних та дійсних чисел.
2. Яким є бінарне відношення $\to$ (досягається) з точки зору його класифікації?
3. Як будується матриця транзитивної досяжності?
4. Як знаходяться компоненти сильної зв'язності?
5. Як проводиться часткове упорядкування компонент сильної зв'язності?
6. Що працює швидше на вашій мові програмування: алгоритм безпосереднього вилучення рядка (стовпця) матриці з коригуванням її розмірів чи алгоритм множення на прокрустову матрицю?
7. Як у MATLAB знайти компоненти сильної зв'язності орграфа та їхнє часткове упорядкування?

7. Найкоротший шлях

У цьому розділі ми розглянемо задачу знаходження найкоротшого шляху в орграфі зі зваженими дугами. Будуть описані два найпоширеніших алгоритми: Дейкстри (E. W. Dijkstra) та Флойда-Воршола (R. W. Floyd, S. Warshall). Також ми навчимося обчислювати ексцентриситети вершин, знаходити радіус та діаметр графа, визначати центральні та периферійні вершини.

7.1. Постановка задачі про найкоротший шлях

Нехай заданий орграф $G = (V, E)$. У попередній главі ми розв'язали задачу: чи можна з вершини v_i потрапити у v_j, рухаючись тільки за стрілками? Зараз ми розглянемо узагальнення цієї задачі: якщо таких шляхів кілька, то який з них найкоротший (має мінімальну кількість дуг)? Для орграфа зі зваженими дугами можна сформулювати задачу так: якщо існує декілька шляхів з v_i у v_j, то який з них має мінімальну загальну вагу?

Ці питання — різні постановки задач про найкоротший шлях (the shortest path problem). Як і в інших задачах, які ми розв'язували раніше, незважений орграф можна розглядати як окремий випадок зваженого, якщо всім дугам приписати одиничні ваги. З іншого боку, можна узагальнити й зважену задачу: доповнити орграф до зваженої кліки відсутніми дугами, приписавши їм нескінченні ваги. Тоді формально можна визначати найкоротший шлях від будь-якої вершини до будь-якої іншої. Якщо результатом буде нескінченність, то це означає, що реальний шлях відсутній.

Будемо вважати, що всі ваги дуг у цій задачі невід'ємні. Це суттєве обмеження: якщо раптом в орграфі виявиться орцикл з загальною від'ємною вагою, то задача мінімізації втратить сенс: можна буде весь час кружляти вздовж цього орциклу, зменшуючи загальну вагу до $-\infty$.

Ми розглянемо два найпоширеніших алгоритми розв'язання задачі про найкоротший шлях. В одному з них (алгоритмі Дейкстри) знаходяться найкоротші шляхи від заданої вершини до всіх інших. Другий алгоритм (Флойда-Воршола) розв'язує загальнішу задачу: будує матрицю найкоротших шляхів з будь-якої вершини у будь-яку.

7.2. Алгоритм Дейкстри

У цьому алгоритмі задається конкретна стартова вершина v_s, і знаходяться довжини найкоротших шляхів з неї до всіх інших вершин. Алгоритм будується на поступовому розростанні шляхів від v_s до всіх інших вершин уздовж дуг мінімальної ваги. Орграф, що будується на кожному

кроці, при цьому стає схожим на зростаючий кристал або тріщину. Опишемо кроки цього алгоритму, а потім сформулюємо та доведемо теорему про коректність алгоритму Дейкстри. Вхідним параметром є матриця ваг дуг між кожною парою вершин $\boldsymbol{B}$ розміром $n \times n$. Якщо якась дуга відсутня, відповідний елемент матриці $\boldsymbol{B}$ будемо задавати як нескінченність. На виході отримуємо вектор $\boldsymbol{d}$ довжини n, в координатах якого — найкоротші шляхи з заданої вершини v_s до всіх інших. Крім того, нам знадобиться булівський масив $\boldsymbol{u}$ довжини n, в якому ми будемо помічати, чи переглянули ми вже якусь вершину, чи ні. Перед початком роботи алгоритму ми знаходимось у вершині v_s, і жодна вершина ще не переглянута:

$$\begin{cases} u_i = \text{false}; \\ i = \overline{1, n}; \end{cases} \tag{7.1}$$

а у векторі $\boldsymbol{d}$ тільки одна s-а координата дорівнює 0 (це найкоротший шлях від s-ї вершини до самої себе), а всі інші координати є нескінченними:

$$\begin{cases} d_i = 0; & i = s; \\ d_i = \infty; & i \neq s. \end{cases} \tag{7.2}$$

Сам алгоритм Дейкстри містить n ітерацій. На кожній ітерації виконуються такі кроки.

Крок 1. Серед ще не переглянутих вершин (тобто тих, для яких $u_i = \text{false}$) обираємо вершину з найменшим значенням координати вектора $\boldsymbol{d}$. Нехай її номер буде i, тобто обрана вершина v_i. Якщо таких вершин декілька, обираємо будь-яку. Зрозуміло, що на першій ітерації в силу (7.2) завжди буде обиратися стартова вершина v_s. Помічаємо вершину v_i як переглянуту: надаємо $u_i = \text{true}$.

Крок 2. Для всіх вершин, що ще не переглянуті, перераховуємо довжину найкоротшого шляху від v_s до кожної з них. Нехай, наприклад, вершина v_t ще не переглянута. У матриці $\boldsymbol{B}$ елемент b_{it} — це довжина шляху від v_i до v_t. А у векторі $\boldsymbol{d}$ його координати d_i та d_t — це поточні (на цій ітерації) довжини найкоротших шляхів з v_s до v_i та з v_s до v_t. Так ось: якщо шлях від v_s до v_t через v_i виявиться коротшим за той, що є на цій ітерації, його й треба буде взяти за шлях від v_s до v_t:

$$\begin{cases} d_t = \min\left(d_t, d_i + b_{it}\right); \\ \forall t : u_t = \text{false}. \end{cases} \tag{7.3}$$

Дія (7.3) називається *релаксацією* (relaxation) непереглянутих вершин. На цьому поточна ітерація закінчується. Якщо є ще не переглянуті вершини, йдемо на наступну ітерацію. Якщо ж усі n вершин переглянуті, закінчуємо роботу алгоритму.

Теорема 7.1. Якщо всі ваги дуг невід'ємні, то вектор $\boldsymbol{d}$, отриманий у результаті роботи алгоритму Дейкстри, є вектором найкоротших шляхів з вершини v_s в усі інші вершини. □

Доведення. Доведемо, що після кожної ітерації найкоротші шляхи до переглянутих вершин вже не змінюються. Якщо це так, то після всіх n ітерацій дійсно матимемо найкоротші шляхи з v_s до всіх вершин.

Доведення будуватимемо за індукцією. Для першої ітерації це твердження очевидне: для стартової вершини v_s маємо: $d_s = 0$, що й є довжиною найкоротшого шляху до неї. Нехай тепер це твердження виконано для всіх попередніх ітерацій, тобто для всіх вже переглянутих вершин. Доведемо, що тоді воно не порушується й після поточної ітерації.

Нехай v_i — вершина, що обрана на поточній ітерації, тобто вершина, яку алгоритм збирається помітити як переглянуту. Доведемо, що d_i дійсно буде дорівнювати довжині найкоротшого шляху з v_s до v_i, яку ми позначимо l_i.

Розглянемо найкоротший шлях (позначимо його P) від v_s до v_i. Оскільки цей шлях починається у переглянутій вершині v_s, то його можна розбити на дві частини: P_1, що складається тільки з переглянутих вершин (принаймні початкова вершина v_s буде в P_1), та решта шляху P_2 (вона теж може включати переглянуті вершини, але починається обов'язково з непереглянутої; принаймні v_i ще не переглянута). Нехай остання вершина P_1 є v_p, а перша вершина P_2 є v_q. Для v_p, за нашим припущенням, відповідна координата вектора $\boldsymbol{d}$ дорівнює довжині найкоротшого шляху з v_s до v_p: $d_p = l_p$.

Доведемо спочатку, що й для v_q: $d_q = l_q$. На одній із попередніх ітерацій ми обирали вершину v_p та виконували релаксацію до неї. Оскільки в силу самого вибору вершини v_q найкоротший шлях з v_s до v_q дорівнює найкоротшому шляху з v_s до v_p плюс вага дуги b_{pq}, то при здійсненні релаксації з v_p величина d_q дійсно стане дорівнювати l_q.

Ваги дуг невід'ємні, тому довжина найкоротшого шляху l_q (як тільки що ми довели, вона дорівнює d_q) не може перевищувати довжини l_i найкоротшого шляху до вершини v_i. Враховуючи, що $l_i \leq d_i$ (адже алгоритм Дейкстри не може знайти коротші шляхи, ніж це взагалі можливо) маємо співвідношення:

$$d_q = l_q \leq l_i \leq d_i. \tag{7.4}$$

З іншого боку, оскільки і v_q, і v_i — непереглянуті вершини, а на поточній ітерації була обрана саме v_i, а не v_q, то маємо іншу нерівність:

$$d_q \geq d_i. \tag{7.5}$$

З цих двох нерівностей випливає, що $d_q = d_i$, а тоді з (7.4, 7.5) отримаємо таку рівність:

$$d_i = l_i; \tag{7.6}$$

що й треба було довести. □

Відновлення найкоротшого шляху. Після закінчення роботи алгоритму Дейкстри ми маємо вектор $\boldsymbol{d}$, координати якого — довжини найкоротших шляхів з v_s до кожної вершини. Як за цими даними відновити сам найкоротший шлях з v_s до якоїсь конкретної, заданої вершини v_t, тобто послідовність вершин? Для такого відновлення треба мати ще один вектор $\boldsymbol{p}$ довжини n, який ми назвемо вектором предків. У кожній його координаті p_i буде зберігатися номер вершини, що передує вершині з номером v_i у найкоротшому шляху до v_i. Зрозуміло, що $p_s = 0$ (у початкової вершини немає предків). Тут використовується адитивність найкоротшого шляху: якщо з найкоротшого шляху до v_i вилучити останню вершину, то отримаємо найкоротший шлях до вершини, номер якої зберігається у p_i. Отже, якщо ми будемо мати такий вектор предків, то найкоротший шлях P до вершини v_t зручно будувати з кінця, додаючи у шлях наступну вершину — предка попередньої, аж доти, доки предком не стане 0:

$$P = \{v_t; p(v_t); p(p(v_t)); p(p(p(v_t))); \ldots; v_s\}. \tag{7.7}$$

Сам вектор $\boldsymbol{p}$ будується дуже просто: при кожній успішній релаксації, тобто коли при порівнянні буде $d_i + b_{it} < d_t$, ми записуємо, що предком v_t наразі буде вершина з номером i:

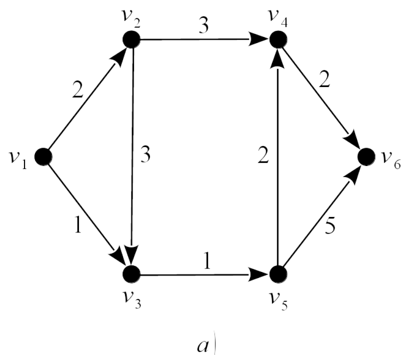
$$p_t = p(t) = i. \tag{7.8}$$

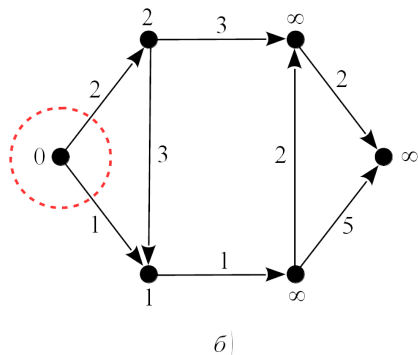
На наступних ітераціях предок v_t може змінитися, якщо буде знайдений ще коротший шлях.

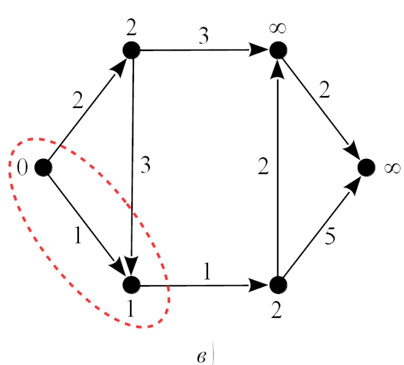
Продемонструємо роботу алгоритму Дейкстри на орграфі з рис. 7.1, *а*. У нього $n = 6$; $m = 8$. Спочатку будемо знаходити довжини найкоротших шляхів з v_1 в усі інші вершини, тобто $s = 1$. Потім відновимо найкоротший шлях з v_1 у v_6.

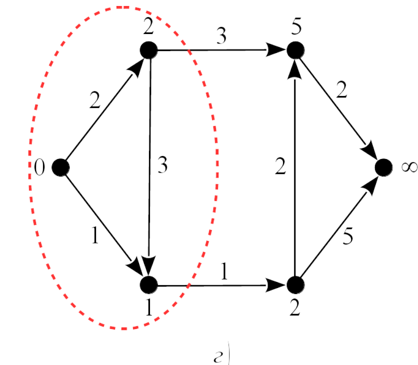
Матриця ваг дуг може містити будь-які елементи на головній діагоналі: вони ніде в алгоритмі не використовуються. Візьмемо нескінченності:

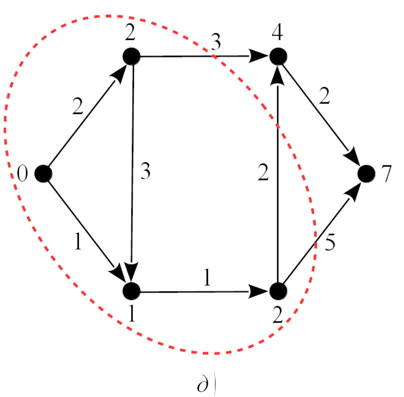
$$\boldsymbol{B} = \begin{pmatrix} \infty & 2 & 1 & \infty & \infty & \infty \\ \infty & \infty & 3 & 3 & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & 2 & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}. \tag{7.9}$$











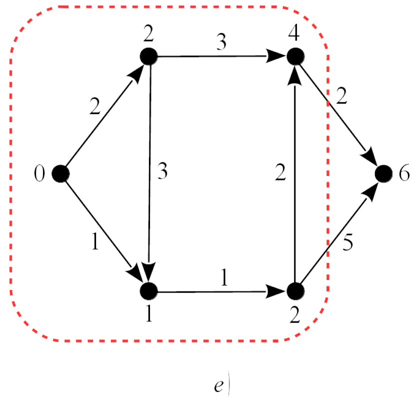


Рис. 7.1. Алгоритм Дейкстри

Перед початком ітерацій заповнюємо вектори (булівські змінні у векторі $\boldsymbol{u}$ позначаємо 0 або 1):

$$\begin{aligned} \boldsymbol{d} &= \{\ 0 \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty\ \}; \\ \boldsymbol{u} &= \{\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ \}; \\ \boldsymbol{p} &= \{\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ \}. \end{aligned} \tag{7.10}$$

Ітерація 1. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ має $d_1 = 0$, тому помічаємо як переглянуту першу вершину: $u_1 = 1$. Для всіх інших, непереглянутих вершин, проводимо релаксацію:

- вершина 2: $d_1 + b_{1,2} = 0 + 2 = 2 < d_2 = \infty \Rightarrow$ є зменшення: $d_2 = 2$; $p_2 = 1$;
- вершина 3: $d_1 + b_{1,3} = 0 + 1 = 1 < d_3 = \infty \Rightarrow$ є зменшення: $d_3 = 1$; $p_3 = 1$;
- вершина 4: $d_1 + b_{1,4} = 0 + \infty = \infty = d_4 = \infty \Rightarrow$ немає зменшення: d_4 та p_4 без змін;
- вершина 5: $d_1 + b_{1,5} = 0 + \infty = \infty = d_5 = \infty \Rightarrow$ немає зменшення: d_5 та p_5 без змін;
- вершина 6: $d_1 + b_{1,6} = 0 + \infty = \infty = d_6 = \infty \Rightarrow$ немає зменшення: d_6 та p_6 без змін.

Результат першої ітерації:

$$\begin{aligned} \boldsymbol{d} &= \{\ 0 \quad 2 \quad 1 \quad \infty \quad \infty \quad \infty\ \}; \\ \boldsymbol{u} &= \{\ 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ \}; \\ \boldsymbol{p} &= \{\ 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0\ \}. \end{aligned} \tag{7.11}$$

Він показаний на рис. 7.1, *б*. Біля кожної вершини проставлена відповідна координата вектора $\boldsymbol{d}$, а переглянуті вершини обведені червоною лінією. Є ще непереглянуті вершини — продовжуємо.

Ітерація 2. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ має $d_3 = 1$; тому помічаємо як переглянуту третю вершину: $u_3 = 1$. Для всіх інших непереглянутих вершин проводимо релаксацію:

- вершина 2: $d_3 + b_{3,2} = 1 + \infty = \infty > d_2 = 2 \Rightarrow$ немає зменшення: d_2 та p_2 без змін;
- вершина 4: $d_3 + b_{3,4} = 1 + \infty = \infty = d_4 = \infty \Rightarrow$ немає зменшення: d_4 та p_4 без змін;
- вершина 5: $d_3 + b_{3,5} = 1 + 1 = 2 < d_5 = \infty \Rightarrow$ є зменшення: $d_5 = 2$; $p_5 = 3$;
- вершина 6: $d_3 + b_{3,6} = 1 + \infty = \infty = d_6 = \infty \Rightarrow$ немає зменшення: d_6 та p_6 без змін.

Результат другої ітерації:

$$\begin{aligned} \boldsymbol{d} &= \{\ 0 \quad 2 \quad 1 \quad \infty \quad 2 \quad \infty\ \}; \\ \boldsymbol{u} &= \{\ 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\ \}; \\ \boldsymbol{p} &= \{\ 0 \quad 1 \quad 1 \quad 0 \quad 3 \quad 0\ \}. \end{aligned} \tag{7.12}$$

Він показаний на рис. 7.1, *в*. Є ще непереглянуті вершини — продовжуємо.

Ітерація 3. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ мають $d_2 = d_5 = 2$. Обираємо, наприклад, d_2. Помічаємо як переглянуту другу вершину: $u_2 = 1$. Для всіх інших непереглянутих вершин проводимо релаксацію:

- вершина 4: $d_2 + b_{2,4} = 2 + 3 = 5 < d_4 = \infty \Rightarrow$ є зменшення: $d_4 = 5$; $p_4 = 2$;
- вершина 5: $d_2 + b_{2,5} = 2 + \infty = \infty > d_5 = 2 \Rightarrow$ немає зменшення: d_5 та p_5 без змін;
- вершина 6: $d_2 + b_{2,6} = 2 + \infty = \infty = d_6 = \infty \Rightarrow$ немає зменшення: d_6 та p_6 без змін.

Результат третьої ітерації:

$$\begin{aligned} \boldsymbol{d} &= \{\ 0 \quad 2 \quad 1 \quad 5 \quad 2 \quad \infty\ \}; \\ \boldsymbol{u} &= \{\ 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0\ \}; \\ \boldsymbol{p} &= \{\ 0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 0\ \}. \end{aligned} \tag{7.13}$$

Він показаний на рис. 7.1, *г*. Є ще непереглянуті вершини — продовжуємо.

Ітерація 4. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ має $d_5 = 2$. Помічаємо п'яту вершину як переглянуту: $u_5 = 1$. Для решти непереглянутих вершин проводимо релаксацію. Перевіряємо:

- вершина 4: $d_5 + b_{5,4} = 2 + 2 = 4 < d_4 = 5 \Rightarrow$ є зменшення: $d_4 = 4$; $p_4 = 5$;
- вершина 6: $d_5 + b_{5,6} = 2 + 5 = 7 < d_6 = \infty \Rightarrow$ є зменшення: $d_6 = 7$; $p_6 = 5$.

Результат четвертої ітерації:

$$\begin{aligned} \boldsymbol{d} &= \{\ 0 \quad 2 \quad 1 \quad 4 \quad 2 \quad 7\ \}; \\ \boldsymbol{u} &= \{\ 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0\ \}; \\ \boldsymbol{p} &= \{\ 0 \quad 1 \quad 1 \quad 5 \quad 3 \quad 5\ \}. \end{aligned} \tag{7.14}$$

Він показаний на рис. 7.1, *д*. Є ще непереглянуті вершини — продовжуємо.

Ітерація 5. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ має $d_4 = 4$. Помічаємо четверту вершину як переглянуту: $u_4 = 1$. Для непереглянутої вершини v_6 проводимо релаксацію:

- вершина 6: $d_4 + b_{4,6} = 4 + 2 = 6 < d_6 = 7 \Rightarrow$ є зменшення: $d_6 = 6$; $p_6 = 4$.

Результат п'ятої ітерації:

$$\begin{array}{l} \boldsymbol{d} = \{\ 0 \quad 2 \quad 1 \quad 4 \quad 2 \quad 6\ \}; \\ \boldsymbol{u} = \{\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0\ \}; \\ \boldsymbol{p} = \{\ 0 \quad 1 \quad 1 \quad 5 \quad 3 \quad 4\ \}. \end{array} \tag{7.15}$$

Він показаний на рис. 7.1, *е*. Залишилася остання непереглянута вершина v_6 — проводимо останню ітерацію.

Ітерація 6. Мінімальне значення серед непереглянутих координат вектора $\boldsymbol{d}$ має $d_6 = 6$. Помічаємо v_6 як переглянуту: $u_6 = 1$. Непереглянутих вершин немає — закінчуємо ітерації. Остаточний результат:

$$\begin{array}{l} \boldsymbol{d} = \{\ 0 \quad 2 \quad 1 \quad 4 \quad 2 \quad 6\ \}; \\ \boldsymbol{u} = \{\ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\ \}; \\ \boldsymbol{p} = \{\ 0 \quad 1 \quad 1 \quad 5 \quad 3 \quad 4\ \}. \end{array} \tag{7.16}$$

У векторі $\boldsymbol{d}$ — найкоротші довжини шляхів від v_1 до кожної іншої вершини. Відновлюємо найкоротший шлях від v_1 до, наприклад, v_6. Кінцева вершина цього шляху — це v_6. Попередні: $p_6 = 4 \Rightarrow v_4$; $p_4 = 5 \Rightarrow v_5$; $p_5 = 3 \Rightarrow v_3$; $p_3 = 1 \Rightarrow v_1$; $p_1 = 0$ — все. Маємо найкоротший шлях: $v_1 \to v_3 \to v_5 \to v_4 \to v_6$. Його довжина дорівнює $b_{1,3} + b_{3,5} + b_{5,4} + b_{4,6} = 1 + 1 + 2 + 2 = 6$.

Алгоритм Дейкстри може бути реалізований у вигляді функції, що потребує двох вхідних параметрів. Перший — це матриця ваг дуг `B` розміром $n \times n$ з нескінченностями на головній діагоналі, така, як (7.9). Другий — це номер початкової (стартової) вершини `s`. Повертає функція два одновимірні масиви `d` та `p` довжини n. У масиві `d` повертаються довжини найкоротших шляхів від v_s до усіх інших вершин, а у масиві `p` — номери вершин-предків для кожної вершини. Текст цієї функції на псевдокоді наведений нижче. Потрібна додаткова функція `n=GetRows(A)`, що повертає кількість рядків матриці `A`.

```
(d,p) = function Dijkstra(B,s) {алгоритм Дейкстри}
begin {Dijkstra}
  n = GetRows(B) {кількість вершин}
  for i=1 step 1 to n do {заповнюємо масиви}
  begin {for i}
    d(i) = infinity {довжини найкоротших шляхів}
    u(i) = false {переглянуті вершини}
    p(i) = 0 {номери вершин-предків}
  end {for i}
  d(s) = 0 {починаємо з вершини номер s, довжина шляху до неї 0}
```

```
  for i=1 step 1 to n do {ітерації алгоритму Дейкстри}
  begin {for i}
    dmin = infinity {для пошуку мінімуму}
    jmin = 0
    for j=1 step 1 to n do {вершина з найменшим значенням d}
    begin {for j}
      if ( (not u(j)) and (d(j)<dmin) ) {ще не переглянуту}
      then
      begin {if then}
        dmin = d(j) {найменше значення d}
        jmin = j {номер цієї вершини}
      end {if then}
    end {for j}
    u(jmin) = true {тепер ця вершина переглянута}
    for j=1 step 1 to n do {релаксації}
    begin {for j}
      if ( (not u(j)) and (d(jmin)+B(jmin,j)<d(j)) )
      then {є релаксація}
      begin {if then}
        d(j) = d(jmin)+B(jmin,j) {нове значення шляху}
        p(j) = jmin {предок цієї вершини}
      end {if then}
    end {for j}
  end {for i}
end {Dijkstra}
```

Для знаходження найкоротших шляхів у MATLAB є вбудовані методи `shortestpath` та `shortestpathtree`. Перший знаходить найкоротший шлях між заданою парою вершин, а другий — найкоротші шляхи від однієї початкової вершини до кількох чи всіх кінцевих, або навпаки: від кількох чи всіх початкових до однієї кінцевої. Якщо ваги дуг (ребер) не задані, вони вважаються одиничними.

Вхідними параметрами для методу `shortestpath` є граф чи орграф `G`, номери початкової та кінцевої вершин і додаткові параметри (зокрема, й алгоритм). Вихідні параметри методу `shortestpath`:

- `p` — вектор-рядок з номерів вершин у найкоротшому шляху;
- `d` — скаляр з довжиною найкоротшому шляху;
- `e` — вектор-рядок з номерів дуг (ребер) у найкоротшому шляху.

У методі `shortestpathtree` вхідні параметри такі самі, тільки початкові або кінцеві вершини можна задавати вектором. Вихідні параметри у методі `shortestpathtree` є такими:

- `TR` — орграф з деревом найкоротших шляхів;
- `d` — вектор з довжинами найкоротших шляхів;

- `e` — булівський масив, елементи якого показують, чи включене відповідне ребро з `G` в `TR`.

Приклад 7.1. Для орграфа, подібного до графу та орграфу з прикладів 1.4 та 1.5, побудувати найкоротший шлях з крайньої лівої до крайньої правої вершини, і знайти довжини найкоротших шляхів від крайньої лівої вершини до всіх інших.

```
s = [1 1 1 2 3 2 2 3 3 4 5 6 5 6 6 ...
  7 7 8 9 8 9 10]; % початки дуг
t = [2 3 4 3 4 5 6 6 7 7 6 7 8 8 9 9 ...
  10 9 10 11 11 11]; % кінці дуг
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 4]; % ваги дуг
x = [0 1 1 1 2 2 2 3 3 3 4]; % координати вершин
y = [1 2 1 0 2 1 0 2 1 0 1];
s0 = 1; % початок шляху
t0 = 11; % кінець шляху
G = digraph(s,t,w); % створили орграф зі зваженими дугами
[p,d,e] = shortestpath(G,s0,t0); % найкоротший шлях
NodeSize = ones(numnodes(G),1)*4; % розміри міток вершин
NodeSize(p) = 7; % розміри міток вершин на найкоротшому шляху
EdgeWidth = ones(numedges(G),1); % товщини ліній дуг
EdgeWidth(e) = 4; % товщини ліній дуг найкоротшого шляху
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"MarkerSize",NodeSize, ...
  "LineWidth",EdgeWidth,"EdgeLabel",G.Edges.Weight);
  % нарисували орграф і найкоротший шлях
title("Найкоротший шлях від вершини " + s0 + ...
  " до вершини " + t0 + " дорівнює " + d)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("ShortestPath1","-dpng") % зберегли рисунок у файл
[~,d] = shortestpathtree(G,s0); % найкоротші шляхи
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",d, ...
  "EdgeLabel",G.Edges.Weight); % орграф і найкоротші шляхи
title("Найкоротші шляхи від вершини " + s0)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("ShortestPath2","-dpng") % зберегли рисунок у файл
```

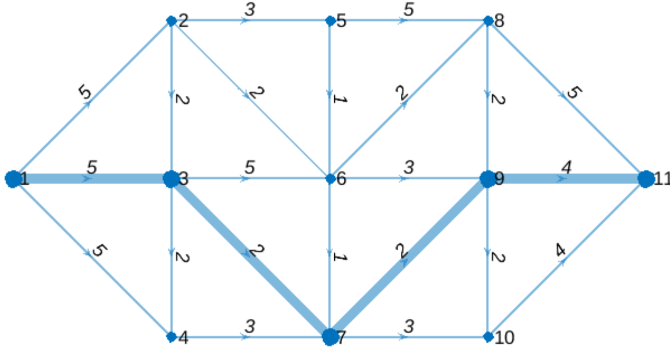


Рис. 7.2. Найкоротший шлях від v_1 до v_{11}

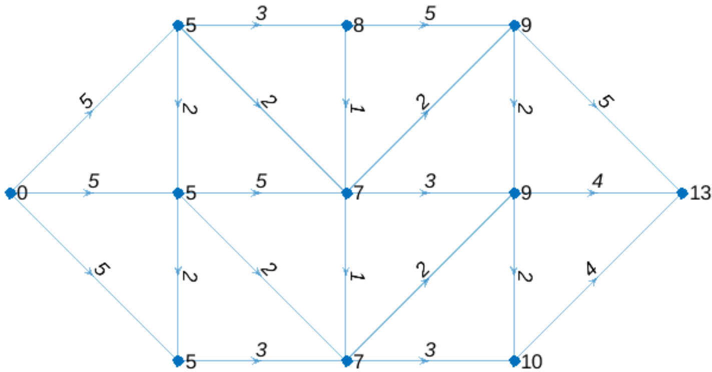


Рис. 7.3. Довжини найкоротших шляхів від v_1 до всіх інших вершин

На рис. 7.2 біля кожної вершини показаний її номер, а на рис. 7.3 — найкоротша відстань до неї від v_1. Мітки ребер — довжини шляхів між відповідними вершинами. □

7.3. Алгоритм Флойда-Воршола

У цьому алгоритмі будується матриця найкоротших шляхів між усіма парами вершин орграфа. На вході треба задати матрицю `B` розміром $n \times n$ з вагами дуг, як і в алгоритмі Дейкстри. На головній діагоналі можуть бути будь-які числа, алгоритм їх не використовує. Але для модифікації матриці предків (див. далі) зручно взяти нескінченності. Весь ал-

горитм процедури вміщується в один потрійний цикл. Його запис на псевдокоді — нижче. Як і в алгоритмі Дейкстри, потрібна додаткова функція `n=GetRows(A)`, що повертає кількість рядків матриці `A`.

```
procedure FloydWarshall(B) {процедура алгоритму Флойда-Воршола}
begin {FloydWarshall}
  n = GetRows(B)
  for k=1 step 1 to n do
  begin {for k}
    for i=1 step 1 to n do
    begin {for i}
      for j=1 step 1 to n do
      begin {for j}
        B(i,j) = min(B(i,j),B(i,k)+B(k,j))
      end {for j}
    end {for i}
  end {for k}
end {FloydWarshall}
```

Сенс цього потрійного циклу показаний на рис. 7.4: для кожної проміжної вершини v_k ми перевіряємо всі можливі початкові вершини v_i та кінцеві v_j. Якщо при кожній такій перевірці сума ваг дуг $b_{ik} + b_{kj}$ буде меншою за b_{ij}, замінюємо довший шлях b_{ij} коротшим $b_{ik} + b_{kj}$. Ця операція називається операцією трикутника.

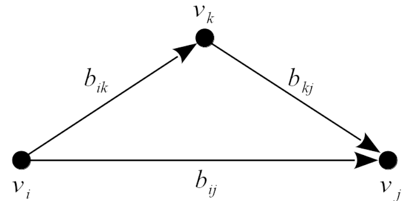


Рис. 7.4. Релаксація трикутника

Означення 7.1. *Операцією (релаксацією) трикутника* (triangle relaxation) для фіксованої вершини v_k називається операція:

$$\begin{cases} b_{ij} = \min\left(b_{ij}, b_{ik} + b_{kj}\right); \\ i = \overline{1,n}; i \neq k; \\ j = \overline{1,n}; j \neq k. \end{cases} \tag{7.17}$$

де допускається $i = j$. □

Зауважимо, що в наведеній вище процедурі алгоритмі Флойда-Воршола ми не перевіряли виконання умов $i \neq k$ та $j \neq k$. В цьому немає потреби, оскільки ми задавали на головній діагоналі матриці $\boldsymbol{B}$ нескінченності.

Отже, алгоритм Флойда-Воршола полягає в проведенні операції трикутника для кожної вершини. Виявляється, цього цілком достатньо!

Теорема 7.2. Якщо всі ваги дуг невід'ємні, то після одноразового проведення операції трикутника для всіх вершин матриця $\boldsymbol{B}$ стає матрицею найкоротших шляхів. □

Доведення. Доведемо за індукцією, що після кроку номер s матриця $\boldsymbol{B}$ стане матрицею найкоротших шляхів з будь-якої вершини у будь-яку вздовж вершин з номерами $k \leq s$. Почнемо з $s = 1$. Якщо провести операцію трикутника тільки для $k = 1$, то шляхи з будь-якої вершини у будь-яку через v_1 будуть найкоротшими. Значить, для $s = 1$ теорема має місце. Припустимо що теорема вірна для $s = r - 1$: шляхи з будь-якої вершини у будь-яку через вершини з номерами від 1 до $r - 1$ мінімальні за довжиною. Виконаємо операцію трикутника ще й для проміжної вершини номер r: $b_{ij} = \min(b_{ij}, b_{ir} + b_{rj})$. Доведемо, що і в цьому випадку шлях з будь-якої вершини у будь-яку, що проходить через проміжні вершини v_1, v_2, ..., v_r, буде найкоротшим. Якщо реальний найкоротший шлях насправді не проходить через нову вершину v_r, то у (7.17) мінімальним буде перше число b_{ij}, і в цьому випадку теорема вірна, оскільки додавання нової проміжної вершини v_r нічого не змінює. Якщо ж реальний найкоротший шлях пройде через нову вершину v_r, то b_{ij} буде замінене на меншу величину $b_{ir} + b_{rj}$. Але кожен з доданків b_{ir} та b_{rj} є найкоротшим шляхом через усі проміжні вершини v_1, v_2, ..., v_{r-1}, тому $b_{ir} + b_{rj}$ дійсно буде найкоротшим шляхом з v_i у v_j через проміжні вершини v_1, v_2, ..., v_r. Отже, за індукцією теорема доведена. □

Відновлення найкоротшого шляху. В алгоритмі Дейкстри ми формували вектор предків $\boldsymbol{p}$ довжиною n. В алгоритмі Флойда-Воршола знаходяться найкоротші шляхи не від однієї вершини до всіх інших, а від кожної до всіх інших. Тому тут треба формувати вже не вектор, а матрицю предків $\boldsymbol{P}$ розміром $n \times n$. Кожен її елемент p_{ij} — це предок вершини v_j у найкоротшому шляху зі стартової вершини v_i. Ця матриця формується так само, як і вектор $\boldsymbol{p}$ в алгоритмі Дейкстри: якщо на якомусь кроці дійсно є зменшення довжини шляху: $b_{ik} + b_{kj} < b_{ij}$, то треба запам'ятати номер k у відповідному місці матриці $\boldsymbol{P}$. Модифікований алгоритм Флойда-Воршола на псевдокоді буде виглядати так, як показано нижче. Вхідними параметрами процедури є матриця ваг дуг `B` та незаповнена матриця предків `P`. В результаті роботи процедури вони змінюються, і на виході отримуємо: у матриці `B` — довжини найкоротших шляхів, у матриці `P` — номери вершин-предків. Використовується додаткова функція `n=GetRows(A)`, що повертає кількість рядків матриці `A`.

```
procedure FloydWarshall(B,P) {модиф. алгоритм Флойда-Воршола}
begin {FloydWarshall}
  n = GetRows(B) {кількість вершин}
  for i=1 step 1 to n do {заповнюємо матрицю предків}
  begin {for i}
```

```
    for j=1 step 1 to n do
    begin {for j}
      P(i,j) = i {перший рядок усі 1, другий - усі 2 тощо}
    end {for j}
  end {for i}
  for k=1 step 1 to n do {основний 3-й цикл Флойда-Воршола}
  begin {for k}
    for i=1 step 1 to n do
    begin {for i}
      for j=1 step 1 to n do
      begin {for j}
        if ( B(i,k)+B(k,j)<B(i,j) ) {є релаксація}
        then
        begin {if then}
          B(i,j) = B(i,k)+B(k,j)
          P(i,j) = P(k,j)
        end (if then}
      end {for j}
    end {for i}
  end {for k}
  for i=1 step 1 to n do {модифікуємо матрицю предків}
  begin {for i}
    for j=1 step 1 to n do
    begin {for j}
      if ( B(i,j)=infinity ) {немає шляху між vi та vj}
      then
      begin {if then}
        P(i,j) = 0 {немає предка}
      end (if then}
    end {for j}
  end {for i}
end {FloydWarshall}
```

Щоб останній подвійний цикл (модифікація матриці предків) провівся коректно, треба на діагоналі матриці $\boldsymbol{B}$ задавати нескінченності.

У MATLAB є вбудований метод `distances`, який для заданого графа чи орграфа `G` повертає квадратну матрицю найкоротших відстаней між усіма парами його вершин. Якщо шлях відсутній, повертається нескінченність. Вага дуги (ребра) розглядається як довжина шляху між відповідними вершинами. Для незважених графів вона вважається одиничною. Але матриця предків у цьому методі не обчислюється. Тому для побудови самóго найкоротшого шляху користуйтеся методом `shortestpath` з прикладу 7.1.

Приклад 7.2. Для орграфа з рис. 7.2 знайти матрицю найкоротших шляхів між кожною парою вершин.

```
s = [1 1 1 2 3 2 2 3 3 4 5 6 5 6 6 ...
  7 7 8 9 8 9 10]; % початки дуг
t = [2 3 4 3 4 5 6 6 7 7 6 7 8 8 9 9 ...
  10 9 10 11 11 11]; % кінці дуг
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 2 5 4 4]; % ваги дуг
x = [0 1 1 1 2 2 2 3 3 3 4]; % координати вершин
y = [1 2 1 0 2 1 0 2 1 0 1];
G = digraph(s,t,w); % зважений орграф
D = distances(G); % відстані між усіма парами вершин
fprintf("Мінімальні відстані між усіма " + ...
  "вершинами орграфа:\n    ")
fprintf("%5.0f",1:size(D,2))
fprintf("\n")
for k1=1:size(D,1) % матриця відстаней
  fprintf("%4.0f ",k1)
  fprintf(" %3.0f ",D(k1,:))
  fprintf("\n")
end
```

```
Мінімальні відстані між усіма вершинами орграфа:
        1    2    3    4    5    6    7    8    9   10   11
   1    0    5    5    5    8    7    7    9    9   10   13
   2  Inf    0    2    4    3    2    3    4    5    6    9
   3  Inf  Inf    0    2  Inf    5    2    7    4    5    8
   4  Inf  Inf  Inf    0  Inf  Inf    3  Inf    5    6    9
   5  Inf  Inf  Inf  Inf    0    1    2    3    4    5    8
   6  Inf  Inf  Inf  Inf  Inf    0    1    2    3    4    7
   7  Inf  Inf  Inf  Inf  Inf  Inf    0  Inf    2    3    6
   8  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0    2    4    5
   9  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0    2    4
  10  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0    4
  11  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0
```

На діагоналі отриманої матриці — нулі: найменша відстань від кожної вершини до самої себе нульова. □

7.4. Метричні характеристики графа

Якщо доповнити m стрілок орграфа ще m стрілками протилежного напрямку з такими самими вагами, як і у наявних, то за допомогою,

наприклад, алгоритму Флойда-Воршола можна отримати матрицю найкоротших відстаней між будь-якою парою вершин у простому графі. У мультиграфі автоматично будуть обиратися ребра найменшої ваги, а петлі взагалі не впливають на довжину шляху, тому ми тут кажемо саме про простий граф.

Означення 7.2. *Відстанню* (the distance) між вершинами v_i та v_j у простому графі називається довжина найкоротшого шляху від v_i до v_j в орграфі з $2m$ дугами, що утворюється з простого графа заміною кожного ребра на дві дуги протилежного напрямку з такими самими вагами. □

Наприклад, якщо в орграфі з рис. 7.1, *а* замінити дуги ребрами, то матриця найкоротших відстаней $\boldsymbol{B}$ матиме такий вигляд (на головній діагоналі ставимо нулі):

$$\boldsymbol{B} = \begin{pmatrix} 0 & 2 & 1 & 4 & 2 & 6 \\ 2 & 0 & 3 & 3 & 4 & 5 \\ 1 & 3 & 0 & 3 & 1 & 5 \\ 4 & 3 & 3 & 0 & 2 & 2 \\ 2 & 4 & 1 & 2 & 0 & 4 \\ 6 & 5 & 5 & 2 & 4 & 0 \end{pmatrix}. \tag{7.18}$$

Ця матриця є симетричною. Її елемент b_{ij} — це найкоротша відстань між вершинами v_i та v_j. Так, наприклад, серед усіх вершин найвіддаленішою від v_1 буде v_6, тому що $d_{1,6} = 6$, а це найбільший елемент першого рядка. Усі інші вершини розташовані ближче до v_1, ніж v_6. Так само, максимальний елемент четвертого рядка — це $b_{4,1} = 4$, тому найвіддаленішою від v_4 буде v_1.

Означення 7.3. *Ексцентриситетом вершини* (eccentricity of the vertex) зв'язного графа v_i називається найдовша відстань від v_i до інших вершин. □

Ексцентриситет кожної вершини легко відновлюється за матрицею відстаней $\boldsymbol{B}$: це найбільший елемент кожного рядка (або стовпця). Для нашого графа вектор ексцентриситетів вершин є таким:

$$\boldsymbol{\varepsilon} = \begin{pmatrix} 6 & 5 & 5 & 4 & 4 & 6 \end{pmatrix}. \tag{7.19}$$

Означення 7.4. *Радіусом графа* (radius of a graph) називається найменший з ексцентриситетів вершин. Вершини з мінімальним ексцентриситетом, який дорівнює радіусу, називаються *центральними* (central vertex). □

Означення 7.5. *Діаметром графа* (diameter of a graph) називається найбільший з ексцентриситетів вершин. Вершини з максимальним ексцентриситетом, який дорівнює діаметру, називаються *периферійними* (peripheral vertex). □

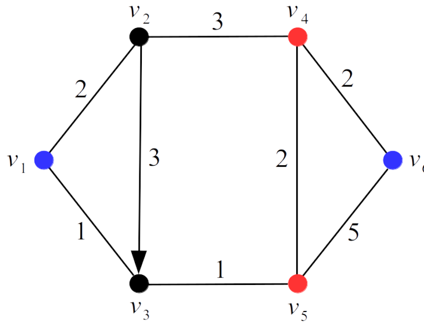


Рис. 7.5. Центральні (червоні) та периферійні (сині) вершини графа

У нашому прикладі центральними є вершини v_4 та v_5, а периферійними — v_1 та v_6. Вони показані на рис. 7.5 відповідно червоним та синім кольорами.

Приклад 7.3. Для псевдографа зі зваженими ребрами знайти ексцентриситети його вершин, радіус, діаметр, визначити центральні та периферійні вершини.

```
s = [1 3 4 5 6 2 8 3 9 9 10 7 8 8 10 11 ...
  7 13 14 15 12 13 13 14 5 15 16 12 13 ...
  20 17 17 18 19 19 5 21 17 18 22 18 19 ...
  20 21 22 23 10 24 24]; % початки ребер
t = [2 2 3 4 1 7 2 8 4 5 5 6 7 9 9 6 12 ...
  8 9 10 11 12 14 13 5 14 11 17 18 15 16 ...
  18 17 18 20 5 16 22 22 18 23 24 25 22 ...
  21 24 10 23 25]; % кінці ребер
w = [1 2 3 4 5 6 7 1 9 8 7 6 5 4 3 2 1 2 ...
  3 4 5 6 7 8 10 9 8 7 6 5 4 3 2 1 2 8 3 ...
  4 5 6 7 8 9 8 7 6 8 5 4]; % ваги ребер
x = reshape(repmat(0:4,1,5),25,1); % абсциси вершин
y = reshape(repmat(4:-1:0,5,1),25,1); % ординати вершин
G = graph(s,t,w); % граф зі зваженими ребрами
D = distances(G); % відстані між усіма парами вершин
Ecc = max(D); % ексцентриситети вершин
Rad = min(Ecc); % радіус графа
Diam = max(Ecc); % діамент графа
Cv = find(abs(Ecc-Rad)<10*eps); % центральні вершини
Pv = find(abs(Ecc-Diam)<10*eps); % периферійні вершини
```

```
figure % нове вікно фігури
h = plot(G,"XData",x,"YData",y,"EdgeLabel",G.Edges.Weight, ...
  "NodeLabel",Ecc); % нарисували граф
highlight(h,Cv,"NodeColor","r", ...
  "MarkerSize",8) % позначили центральні вершини
highlight(h,Pv,"NodeColor","b", ...
  "MarkerSize",8) % позначили периферійні вершини
title("Ексцентриситети вершин") % заголовок рисунку
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("Eccentricity","-dpng") % зберегли рисунок у файл
```

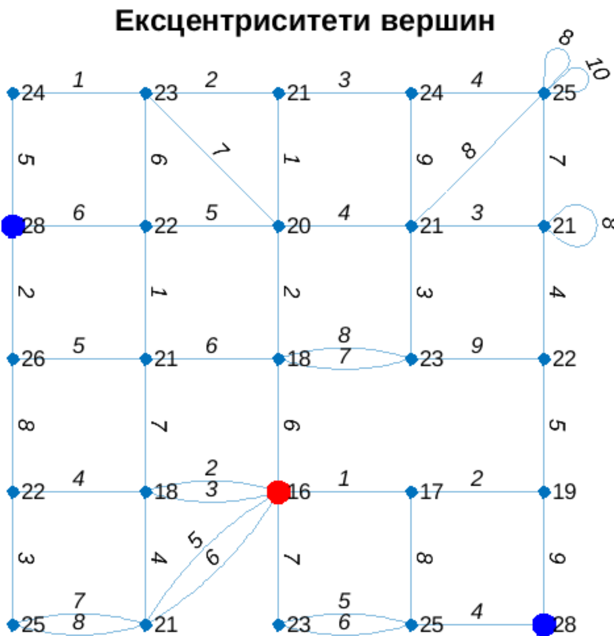


Рис. 7.6. Ексцентриситети вершин псевдографа

На рис. 7.6 мітками ребер є їхні ваги (відстані між вершинами), а вершин — їхні ексцентриситети. Червоним кольором відмічені центральні вершини, а синім — периферійні. □

7.5. Запитання для перевірки

1. Як ставиться задача про найкоротший шлях?

2. Як працює алгоритм Дейкстри? Як у ньому відновити найкоротший шлях?
3. Як реалізований алгоритм Дейкстри у MATLAB?
4. Як працює алгоритм Флойда-Воршола? Як у ньому відновити найкоротший шлях?
5. Як реалізований алгоритм Флойда-Воршола у MATLAB?
6. Що таке ексцентриситет вершини графа?
7. Що таке радіус графа? діаметр графа?
8. Які вершини графа називаються центральними? периферійними?
9. Як у MATLAB визначити ексцентриситети вершин? радіус графа? його діаметр? знайти центральні та периферійні вершини?

8. Мережеві задачі

Наразі ми розглянемо три задачі на мережах: знаходження максимального потоку в мережі з обмеженими пропускними здатностями, знаходження мінімального розрізу в ній та створення плану керування проектом.

8.1. Мережі

Означення 8.1. *Мережею* (a network) називається ациклічний орграф, у якого є:

- *початкова* (стартова, джерело) вершина (source) v_s, з якої тільки виходять дуги, і з якої досяжні всі інші вершини;
- *кінцева* (хвостова, стік) вершина (tail, sink) v_t, в яку тільки входять дуги, і яка є досяжною з будь-якої іншої вершини. □

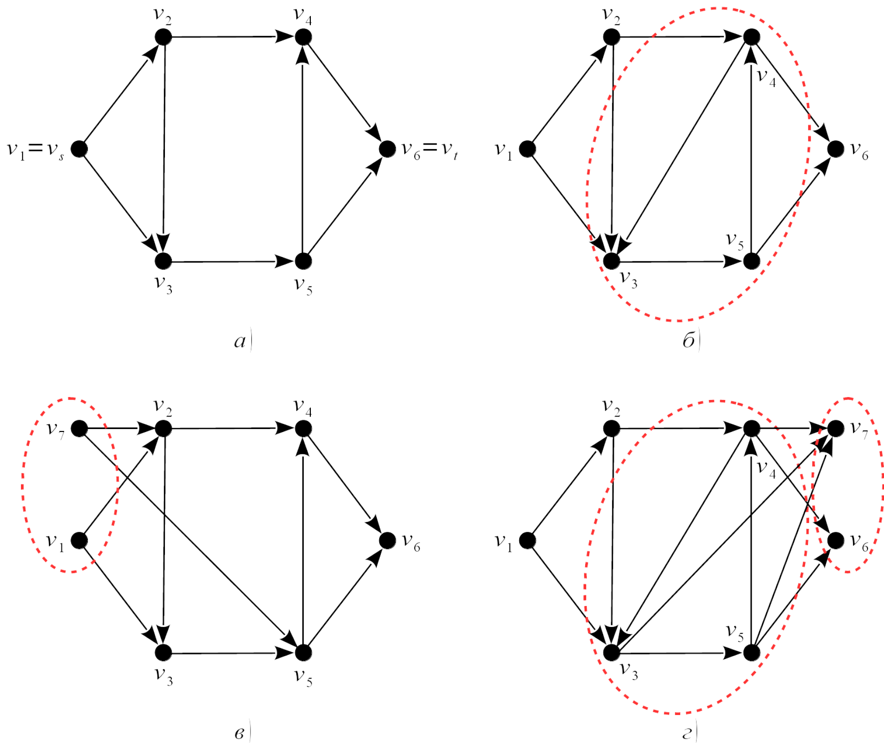


Рис. 8.1. Мережа (а) та орграфи, що не є мережами (б, в, г)

На рис. 8.1, *а* зображений орграф, який є мережею: він ациклічний, і в ньому є початкова та кінцева вершини. А на рис. 8.1, *бвг* показані орграфи, які не є мережами. У орграфа з рис. 8.1, *б* є орцикл, він обведений червоною лінією. У орграфа на рис. 8.1, *в* не існує стартової вершини, а на рис. 8.1, *г* показаний орграф, у якого є орцикл та немає кінцевої вершини.

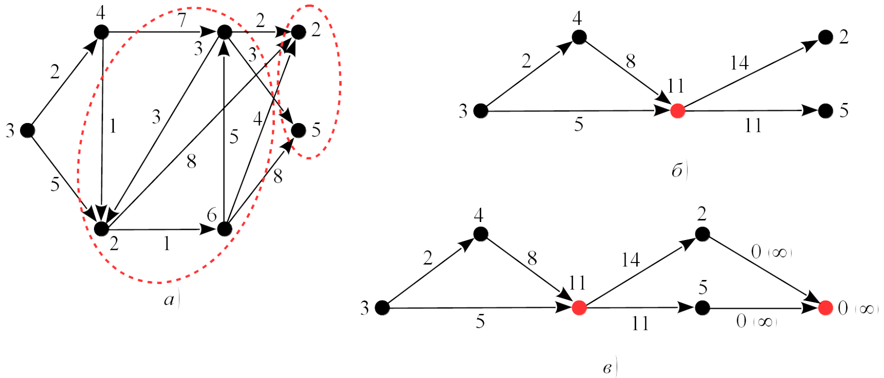


Рис. 8.2. Перетворення орграфа на мережу

Щоб перетворити будь-який орграф на мережу, треба перш за все позбутися орциклів. Для цього знаходимо сильно зв'язані компоненти орграфа (див. главу 6), та стягуємо вершини кожної компоненти сильної зв'язності в одну. Отриманий орграф компонент сильної зв'язності буде вже ациклічним і, як наслідок, частково упорядкованим (рис. 6.4). Якщо початковий орграф був зі зваженими вершинами та (або) дугами, то дії над вагами залежать від сенсу конкретної задачі. Інколи ваги треба додавати, інколи брати найбільшу тощо. На рис. 8.2, *а* показаний орграф з рис. 8.1, *г* зі зваженими вершинами та дугами, а на рис. 8.2 — той самий орграф з стягнутим орциклом. Ваги вершин компоненти сильної зв'язності тут додаються. Ваги дуг, що з'єднують компоненту сильної зв'язності з іншими вершинами, теж додаються. А дуги всередині компоненти сильної зв'язності просто зникають. Що робити з їхніми вагами — залежить від конкретної задачі.

Тепер треба перевірити, чи є у нашого ациклічного орграфа початкова та (або) кінцева вершини. Якщо є кілька початкових вершин, між якими відсутнє упорядкування, треба додати ще одну (фіктивну) вершину, з якої провести дуги у наші початкові. Наприклад, у орграфа на рис. 8.2, *в* немає початкової вершини у сенсі означення 8.1, бо v_1 та v_7 не упорядковані. Тому додаємо до орграфа ще одну вершину v_8 та дуги $e_{8,1}$ та $e_{8,7}$. Так само і з кінцевою вершиною. Якщо вона відсутня (рис. 8.2, *б*),

додаємо її та проводимо дуги до неї з усіх останніх вершин або компонент сильної зв'язності, неупорядкованих між собою. Таке додавання показане на рис. 8.2, *в*. Ваги доданих вершин та дуг можна покласти нулю або нескінченності в залежності від сенсу конкретної задачі.

Приклад 8.1. В орграфі з рис. 6.2 видалити верхню праву дугу (v_8, v_4), зважити дуги та створити з цього орграфа мережу. Ваги дуг між компонентами сильної зв'язності додати, для дуг з та до додаткових фіктивних вершин задати великі ваги.

```
s = [1 2 3 1 2 6 2 3 7 5 6 7 9 9 6 6 ...
  11 7 12 9 10 12]; % початки дуг
t = [2 3 4 5 5 2 7 7 4 6 7 8 5 6 10 11 ...
  7 12 8 10 11 11]; % кінці дуг
w = [4 3 5 4 2 3 6 5 4 5 6 3 4 5 9 4  ...
  3 4 5 6 8 6]; % ваги дуг
x = [0 1 2 3 0 1 2 3 0 1 2 3]; % абсциси вершин
y = [2 2 2 2 1 1 1 1 0 0 0 0]; % ординати вершин
wst = 1e5; % вага фіктивних дуг, якщо вони потрібні
G = digraph(s,t,w); % створили зважений орграф
[b,bs,ord] = orderstrongcomp(G); % сильно зв'язані комп.
figure % нове вікно фігури
cm = colormap("jet"); % зберегли палітру
plot(G,"XData",x,"YData",y,"NodeLabel",b,"MarkerSize",6, ...
  "NodeColor",cm(round((b-1)/(length(bs)-1)*255+1),:), ...
  "EdgeLabel",G.Edges.Weight) % намалювали
title("Кількість компонент сильної зв'язності = " + ...
  length(bs)) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("CreateNet1","-dpng") % зберегли рисунок у файл
stn = b(G.Edges.EndNodes); % замість номерів вершин
% номери компонент
GN = digraph(stn(:,1),stn(:,2),G.Edges.Weight);
% створили орграф мережі
GN = simplify(GN,"sum"); % спростили з додаванням ваг дуг
xn = zeros(1,length(bs)); % задаємо координати вершин
% мережі для рисування
yn = xn;
for ks=1:length(bs) % для кожної комп. сильної зв'язності
  ls = find(b==ks); % номери вершин цієї компоненти
  xn(ks) = mean(x(ls)); % середнє арифм. координат вершин
```

```
    yn(ks) = mean(y(ls));
  end
  rxn = 0.2*range(xn); % для рисування фіктивних вершин
  sv = find(ord(1,:)<0.5); % нульові елементи 1-го рядка
  if ~isempty(sv) % є такі - треба додавати початкову вершину
    ns = numnodes(GN)+1; % номер початкової вершини
    GN = addedge(GN,ns,1,wst); % додали зважену дугу
    for ks=1:length(sv) % додаємо інші дуги
      GN = addedge(GN,ns,sv(ks),wst);
    end
    xn(end+1) = min(xn)-rxn; % початкова вершина - лівіше
    yn(end+1) = mean(yn);
  end
  tv = find(ord(:,end)<0.5); % нульові елементи ост. стовпця
  if ~isempty(tv) % є такі - треба додавати кінцеву вершину
    nt = numnodes(GN)+1; % номер кінцевої вершини
    GN = addedge(GN,length(bs),nt,wst); % додали дугу
    for kt=1:length(nt) % додаємо інші дуги
      GN = addedge(GN,tv(kt),nt,wst);
    end
    xn(end+1) = max(xn)+rxn; % кінцева вершина - правіше
    yn(end+1) = mean(yn);
  end
  figure % нове вікно фігури
  plot(GN,"XData",xn,"YData",yn,"MarkerSize",6, ...
    "EdgeLabel",GN.Edges.Weight) % нарисували
  title("Мережа") % заголовок
  axis("equal") % однаковий масштаб уздовж осей координат
  axis("off") % прибрали осі
  print("CreateNet2","-dpng") % зберегли рисунок у файл
```

На рис. 8.3 показане розбиття заданого орграфа на сильно зв'язані компоненти. Деякі з них містять більше однієї вершини, тобто всередині них є орцикли. Такі компоненти замінюємо однією вершиною. Ваги дуг до та від них додаємо. Далі перевіряємо: чи доступні з першої вершини всі наступні? Якщо ні, то додаємо фіктивну початкову вершину та проводимо з неї дуги до потрібних вершин з заданими вагами. Так само робимо з кінцевими вершинами. Якщо кінцева вершина доступна не з усіх попередніх, додаємо фіктивну кінцеву вершину з потрібними дугами до неї. Отримана мережа показана на рис. 8.4. □

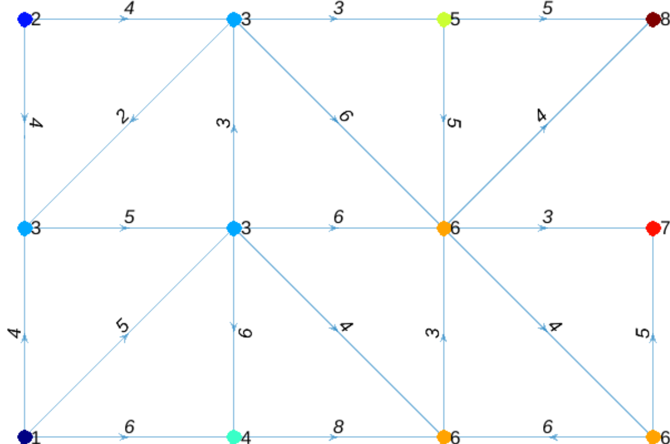


Рис. 8.3. Компоненти сильної зв'язності орграфа

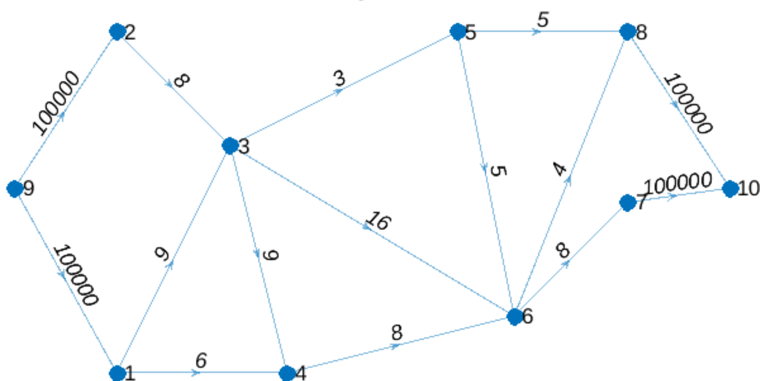


Рис. 8.4. Мережа, створена з орграфа

8.2. Максимальний потік у мережі

Розглянемо таку задачу. На пункті v_s є якийсь ресурс (нафта, товар, гроші, електроенергія тощо), який треба за максимумом пропустити крізь мережу (трубопровідну, транспортну, банківську, електричну тощо) на пункт v_t за умови, що окремі гілки мережі можуть пропустити ресурсу не більше, ніж це в них закладено, а у вузлах мережі ресурс не накопичується: скільки прийшло, стільки ж і вийшло. Це й є *задача про*

максимальний потік (a maximum flow problem) у мережі з обмеженими пропускними здатностями. Сформулюємо математичну постановку цієї задачі.

Нехай є мережа $G = (V, E)$: ациклічний орграф з початковою вершиною v_s та кінцевою v_t. Дуги — зважені: невід'ємну вагу дуги e_k будемо позначати c_k. У задачі про максимальний потік вага кожної дуги розглядається як обмеження на її пропускну здатність. Якщо для створення мережі ми додавали до ациклічного орграфу фіктивну початкову та (або) кінцеву вершини, додані дуги повинні мати нескінченну пропускну здатність. Це означає, що ресурс у необмеженій кількості може потрапляти на реальні початкові вершини, та у необмеженій кількості зніматися з реальних кінцевих вершин.

Для кожної дуги e_k введемо до розгляду асоційовану з нею змінну, яку теж позначимо e_k. Її сенс: це реальна величина ресурсу, що пройшов через цю дугу (потік у дузі). Цей потік e_k є невід'ємним та не може перевищувати пропускної здатності c_k у цій дузі:

$$\begin{cases} 0 \le e_k \le c_k; \\ k = \overline{1, m}. \end{cases} \tag{8.1}$$

У задачі про максимальний потік, як слідує з її назви, треба максимізувати сумарний потік, що виходить з джерела v_s за умови, що в усіх проміжних вершинах (крім v_s та v_t) сума вхідних та вихідних потоків повинна дорівнювати нулю. З загального балансу потоків слідує, що потік у v_t буде відрізнятися від потоку у v_s тільки знаком: адже ані в дугах, ані у проміжних вершинах нічого не затримується!

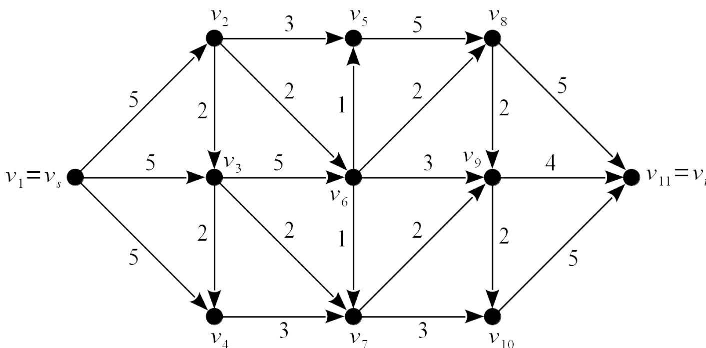


Рис. 8.5. Приклад мережі

Ця задача припускає формулювання її як задачі лінійного програмування. Ми вже ввели до розгляду змінні e_k та обмеження на них за

пропускними здатностями (8.1). Ще треба задати умови балансу ресурсу у кожній проміжній вершині. В цьому нам допоможе матриця інцидентності орграфа (1.9). Нагадаємо: матриця інцидентності орграфа $\boldsymbol{A}$ — це матриця розміром $n \times m$. Кожен її елемент $a_{ik} = 1$, якщо з вершини v_i виходить дуга e_k; $a_{ik} = -1$, якщо у вершину v_i входить дуга e_k; в інших випадках $a_{ik} = 0$. Наприклад, для орграфа мережі з рис. 8.5 матриця інцидентності має вигляд (8.2). Рядки в ній дуже широкі, і матриця не вміщується в сторінку, тому записана у транспонованому вигляді.

$$\boldsymbol{A}^T = \begin{pmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1
\end{pmatrix}. \qquad (8.2)$$

У кожному стовпці цієї матриці є по одній одиниці та по одній -1, тому сума усіх рядків — нульовий рядок. Це відповідає загальному балансу потоків.

Для матриці інцидентності орграфа $\boldsymbol{A}$ позначимо:

- $\boldsymbol{a}_s$ — її s-й рядок (вектор-рядок); в ньому є тільки одиниці, тому що з джерела дуги тільки виходять;
- $\boldsymbol{a}_t$ — її t-й рядок (також вектор-рядок); в ньому є тільки -1, тому що у стік дуги тільки входять;
- $\boldsymbol{A}_{st}$ — матриця $\boldsymbol{A}$ з викинутими з неї s-м та t-м рядками.

Тоді цільова функція, що максимізується — це загальний потік з джерела. Вона може бути записана так:

$$z = \boldsymbol{a}_s \boldsymbol{e} \to \max; \qquad (8.3)$$

де $\boldsymbol{e}$ — вектор-стовпчик змінних e_k. А умова балансу потоків в усіх проміжних вершинах записується у вигляді обмеження-рівності:

$$\boldsymbol{A}_{st} \boldsymbol{e} = \boldsymbol{0}. \qquad (8.4)$$

Разом з обмеженнями (8.1) вирази (8.3-8.4) визначають задачу лінійного програмування, причому навіть не бінарного, а звичайного, оскільки потоки в дугах можуть бути й дрібними.

Для орграфів у MATLAB є метод `maxflow`, що знаходить максимальний потік та мінімальний розріз, див. далі підрозділ 8.3. Вхідними параметрами для нього є орграф `G`, номери початкової та кінцевої вершин і, за необхідності, алгоритм. Вихідні параметри:

- `mf` — величина максимального потоку (скаляр);
- `GF` — орграф, створений з орграфа `G`, в якому залишені лише дуги з ненульовими потоками. Потоки в дугах повертаються в полі `GF.Edges.Weight`.

Якщо ваги дуг орграфа `G` не задані, вони вважаються одиничними.

Приклад 8.2. Для орграфа з рис. 8.5 розв'язати задачу про максимальний потік.

```
s = [1 1 1 2 3 2 2 3 3 4 6 6 5 6 6 7 7 ...
  8 9 8 9 10]; % початки дуг
t = [2 3 4 3 4 5 6 6 7 7 5 7 8 8 9 9 10 ...
  9 10 11 11 11]; % кінці дуг
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 ...
  2 2 5 4 4]; % ваги дуг
x = [0 1 1 1 2 2 2 3 3 3 4]; % координати вершин
y = [1 2 1 0 2 1 0 2 1 0 1];
s0 = 1; % джерело мережі
t0 = 11; % стік мережі
G = digraph(s,t,w); % створили орграф мережи
[mf,GF] = maxflow(G,s0,t0); % розв'язали задачу
figure % нове вікно фігури
h = plot(G,"XData",x,"YData",y,"NodeLabel",{}); % орграф
hold("on") % будемо додавати рисунок
plot(GF,"XData",x,"YData",y,"EdgeLabel",GF.Edges.Weight,...
  "EdgeColor","b","LineWidth",4,"ArrowSize",15) % не-0 дуги
hold("off") % закінчили рисування
title("Потоки в дугах. Максимальний потік = " + mf)
```

```
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MaxFlow","-dpng") % зберегли рисунок у файл
```

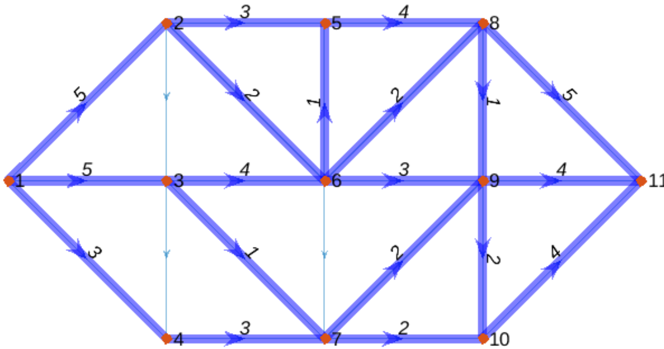


Рис. 8.6. Розв'язання задачі про максимальний потік у мережі

На рис. 8.6 показані результат розв'язання задачі про максимальний потік. При цьому деякі дуги навантажуються повністю: в них $e_k = c_k$; деякі — частково: в них $e_k < c_k$; а деякі виявилися зайвими: потік у них нульовий. □

8.3. Двоїста задача — мінімальний розріз

Як відомо, у будь-якої задачі лінійного програмування є двоїста до неї. Побудуємо задачу, двоїсту до (8.3-8.1-8.4). У нас є m обмежень-нерівностей типу "$\leq$" (праві частини (8.1)) та $n-2$ обмежень-рівностей (8.4). Отже, у двоїстій задачі перші m змінних будуть невід'ємними, а останні $n-2$ довільного знаку. Позначимо змінні двоїстої задачі $\boldsymbol{y}$ — вектор довжиною $m+n-2$. Перші його m координат асоційовані з обмеженнями в дугах (8.1), тобто фактично з дугами, а останні $n-2$ — з обмеженнями у проміжних вершинах (8.4), тобто з усіма вершинами, крім v_s та v_t.

Далі, у початковій задачі всі змінні $e_k \geq 0$, і отже, у двоїстій задачі всі обмеження будуть нерівностями типу "$\geq$". Матриця коефіцієнтів обмежень початкової задачі утворюється з одиничної матриці $\boldsymbol{E}$ для правих частин (8.1) та матриці $\boldsymbol{A}_{st}$ для рівностей (8.4). Транспонуємо її:

$$\begin{pmatrix} \boldsymbol{E} \\ \boldsymbol{A}_{st} \end{pmatrix}^T = \begin{pmatrix} \boldsymbol{E} & \boldsymbol{A}_{st}^T \end{pmatrix} \tag{8.5}$$

— тим самим отримаємо матрицю коефіцієнтів обмежень двоїстої задачі розміром $m \times (m + n - 2)$. Праві частини для обмежень-нерівностей двоїстої задачі — це коефіцієнти цільової функції початкової задачі: транспонований у стовпчик вектор $\boldsymbol{a}_s$. І навпаки, коефіцієнти цільової функції двоїстої задачі — це праві частини обмежень початкової задачі. У нас це вектор $\boldsymbol{c}$ обмежень на пропускні здатності з (8.1), доповнений $n-2$ нулями з (8.4). Отже, задача, двоїста до задачі про максимальний потік, має вигляд:

$$\begin{array}{l} t = \left(\begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{0} \end{pmatrix}, \boldsymbol{y} \right) \to \min; \\ \begin{pmatrix} \boldsymbol{E} & \boldsymbol{A}_{st}^T \end{pmatrix} \boldsymbol{y} \geq \boldsymbol{a}_s^T; \\ \begin{cases} y_k \geq 0; \\ k = \overline{1, m}; \end{cases} \\ \begin{cases} (y_k \geq 0) \cup (y_k < 0); \\ k = \overline{m+1, m+n-2}. \end{cases} \end{array} \tag{8.6}$$

З'ясуємо сенс цієї задачі на прикладі орграфа з рис. 8.5, 8.6. Результат розв'язання задачі (8.6) $t_{\min} = z_{\max} = 13$; як і очікували за теоремами двоїстості. Значення координат вектора $\boldsymbol{y}$ на оптимальному розв'язку наведені в останньому стовпчику табл. 8.1. Для порівняння до цієї таблиці занесені також координати вектора $\boldsymbol{e}$ на оптимальному розв'язку (розв'язок початкової задачі) та обмеження на пропускну здатність $\boldsymbol{c}$. Позначені також номери дуг та номери поєднуваних вершин. Для y_k з останніми $n-2$ номерами у табл. 8.1 наведені лише номери вершин, асоційованих з цими змінними.

Означення 8.2. *Розріз* (cut-set) орграфа мережі — це розбиття частково упорядкованої множини його вершин V на дві підмножини S та T, такі, що $v_s \in S$, $v_t \in T$, а всі дуги між S і T спрямовані з S у T. *Пропускна спроможність розрізу* — це сума пропускних спроможностей усіх дуг, що поєднують S і T. *Потік через розріз* – це сума потоків у дугах, що поєднують S і T. □

Перші m змінних y_k чудово ілюструють теореми двоїстості. У тих дугах, де потік не насичений: $(e_k < c_k)$, відповідна двоїста змінна $y_k = 0$. А там, де потік повністю насичує дугу: $(e_k = c_k)$, відповідна двоїста змінна може бути й більше нуля: $y_k \geq 0$. Про цьому, якщо задача невироджена, то в насичених дугах буде $y_k > 0$. У нас задача вироджена, тому і в деяких насичених дугах буде $y_k = 0$. Якщо в орграфі видалити насичені та зайві дуги, то він розпадеться на дві (для невироджених задач) або кілька (для вироджених) незв'язаних частин. Реальний потік через ці дуги дорівнює $t_{\min} = z_{\max} = 13$, а всі дуги в ньому є або насиченими, або зайвими.

Означення 8.3. Розріз називається *мінімальним* (minimum cut-

Табл. 8.1. Розв'язок задач (8.3-8.1-8.4) та (8.6)

k	v_i	v_j	c_k	e_k	y_k	k	v_i	y_k
1	1	2	5	5	1	23	2	0
2	1	3	5	5	1	24	3	0
3	1	4	5	3	0	25	4	-1
4	2	3	2	0	0	26	5	0
5	3	4	2	0	0	27	6	0
6	2	5	3	3	0	28	7	0
7	2	6	2	2	0	29	8	0
8	3	6	5	4	0	30	9	0
9	3	7	2	1	0	31	10	0
10	4	7	3	3	1			
11	6	5	1	1	0			
12	6	7	1	0	0			
13	5	8	5	4	0			
14	6	8	2	2	0			
15	6	9	3	3	0			
16	7	9	2	2	0			
17	7	10	3	2	0			
18	8	9	2	1	0			
19	9	10	2	2	0			
20	8	11	5	5	0			
21	9	11	4	4	0			
22	10	11	5	4	0			

set), якщо після видалення зайвих дуг потік через нього дорівнює його пропускній спроможності. □

Розріз, який знайдений з розв'язання двоїстої задачі, якраз і є мінімальним. Отже, задача, двоїста до задачі про максимальний потік, є задачею про мінімальний розріз.

Теорема 8.1. Теорема Форда-Фалкерсона (Ford-Fulkerson). Максимальний потік у мережі дорівнює пропускній спроможності її мінімального розрізу після вилучення зайвих дуг. □

Доведення є наслідком теорем двоїстості задач лінійного програмування: $t_{\min} = z_{\max}$. □

Як відомо, двоїсту задачу можна безпосередньо не розв'язувати, а знаходити її розв'язок за розв'язком початкової, використовуючи теореми двоїстості. У нашому випадку ми можемо будувати мінімальний розріз (чи один з мінімальних розрізів, якщо задача вироджена) за потоками в дугах. Робиться це так. Викидаємо з мережі пусті ($e_k = 0$) та насичені ($e_k = c_k$) дуги. Ці викинуті дуги утворюють розріз (коцикл) або містять у собі коцикл, якщо задача вироджена. Перевіряємо, на які компоненти зв'язності розбивається відповідний неорієнтований граф після викидання пустих та насичених дуг. Якщо таких компонент дві, то задача невироджена: всі викинуті дуги утворюють мінімальний розріз. Якщо ж таких компонент буде більше двох, залишаємо тільки якийсь один розріз на дві компоненти.

Метод `maxflow` у MATLAB, який ми розглянули в підрозділі 8.2, розв'язує також і задачу про мінімальний розріз. Достатньо вказати в ньому ще два вихідні параметри:

- `cs` — вектор-рядок з номерами вершин до розрізу;
- `ct` — вектор-рядок з номерами вершин після розрізу.

Приклад 8.3. Доповнити приклад 8.2 розв'язанням задачі про мінімальний розріз.

```
s = [1 1 1 2 3 2 2 3 3 4 6 6 5 6 6 7 7 ...
  8 9 8 9 10]; % початки дуг
t = [2 3 4 3 4 5 6 6 7 7 5 7 8 8 9 9 10 ...
  9 10 11 11 11]; % кінці дуг
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 ...
  2 2 5 4 4]; % ваги дуг
x = [0 1 1 1 2 2 2 3 3 3 4]; % координати вершин
y = [1 2 1 0 2 1 0 2 1 0 1];
s0 = 1; % джерело мережі
t0 = 11; % стік мережі
```

```
G = digraph(s,t,w); % створили орграф мережи
[mf,GF,cs,ct] = maxflow(G,s0,t0); % розв'язали задачу
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",{}); % заданий орграф
hold("on") % будемо додавати рисунок
h = plot(GF,"XData",x,"YData",y, ...
  "EdgeLabel",GF.Edges.Weight, ...
  "EdgeColor","b","LineWidth",4,"ArrowSize",15); % не-0 дуги
hold("off") % закінчили рисування
highlight(h,cs,"NodeColor","b","MarkerSize",8) % до розрізу
highlight(h,ct,"NodeColor","r","MarkerSize",8) % після нього
title("Потоки в дугах. Максимальний потік = " + mf)
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("MinCutSet","-dpng") % зберегли рисунок у файл
```

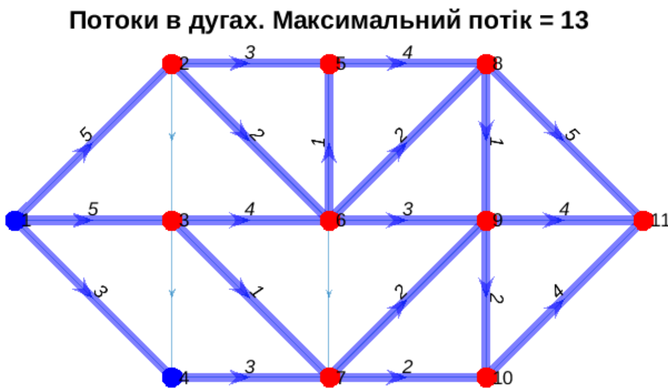


Рис. 8.7. Розв'язання задачі про мінімальний розріз у мережі

На рис. 8.7 синім кольором відмічені вершини до розрізу, а червоним — після. □

8.4. Мережеві діаграми PERT

Існує принаймні два варіанти розшифрування скорочення PERT: одне — це Program Evaluation and Review Technique, а друге — Project Evaluation and Research Task. Але у будь-якому випадку мережева діаграма PERT — це орграф мережі, що допомогає керувати складним проектом з великою кількістю робіт, які треба проводити послідовно або паралельно.

Діаграма PERT є множиною точок-вершин (події, event) разом з дугами, що їх поєднують (роботи, task). Дуги є зваженими: невід'ємна вага

дуги — це час виконання роботи. Будь-яка вершина є подією, що характеризує закінчення одних робіт (вхідні дуги) та початок інших (вихідні дуги). Так у діаграмі реалізується вимога, що до жодної з робіт не можна було б приступити до того, як будуть виконані всі роботи, що їй передують згідно з технологією реалізації проекту. Тому у діаграмі не повинно бути орциклів. Початок процесу реалізації проекту — це v_s, а закінчення — v_t. Всі інші вершини повинні мати і вхідні, і вихідні дуги. Якщо це не так, додаємо фіктивні v_s та (або) v_t з нульовими вагами дуг від (до) них. Отже, діаграма PERT будується не на будь-якому орграфі, а саме на мережі.

Які характеристики проекту зазвичай нас цікавлять? Перш за все, треба визначити загальну тривалість проекту. Розглянемо всі можливі шляхи від v_s до v_t. Сума ваг дуг у кожному шляху — це його тривалість. Найбільша тривалість визначає час виконання всього проекту, мінімально можливий при зафіксованих характеристиках дуг мережі. Відповідний шлях — критичний (critical path), тобто саме від тривалості його робіт залежить загальна тривалість проекту, хоча при змінюванні тривалості будь-яких робіт проекту критичним може стати й інший шлях.

Визначення критичного шляху, на якому затримки робіт неприпустимі, та його довжини й є першою задачею розрахунку характеристик мережі PERT. Цю задачу можна розв'язати за допомогою алгоритму Флойда-Воршола знаходження найкоротшого шляху. Треба тільки змінити знаки у ваг дуг. Алгоритм Флойда-Воршола визначався лише для невід'ємних ваг дуг, але він чудово працює і з від'ємними. Його потрійний цикл не вносить жодних складнощів у розрахунки при від'ємних вагах дуг.

Після роботи алгоритму Флойда-Воршола в s-у рядку матриці $\boldsymbol{B}$ будуть міститися моменти часу настання кожної події (треба тільки змінити знак знову на плюс). Визначення моменту настання кожної події — це друга задача розрахунку характеристик мережі PERT. Тепер ми знаємо, коли повинна розпочатися за планом та чи інша робота: це момент-подія, що передує цій роботі (вершина — початок дуги).

А чи можемо ми з якихось причин затримати якусь роботу без затримки часу виконання всього проекту? Вочевидь, якщо робота знаходиться на критичному шляху, то ні. Для інших робіт затримки можливі, і це не призведе до затримки здачі проекту. Обчислення часу можливої затримки кожної роботи — це третя задача розрахунку характеристик мережі PERT. Як їх знайти? Нехай момент настання якоїсь події v_j — це t_j; момент настання події v_i, що їй передує — це t_i, а робота, що їх поєднує, займає всього e_k одиниць часу. Щоб встигнути до моменту t_j, ми можемо розпочати її виконувати не в момент часу t_i, а пізніше, в момент $t_j - e_k$, якщо цей момент настає пізніше, ніж t_i. Тобто можлива затримка цієї роботи на $t_j - t_i - e_k$ одиниць часу. Загальна формула має вигляд:

$$\begin{cases} d_k = t_j - t_i - e_k; \\ k = \overline{1, m}; \end{cases} \tag{8.7}$$

де e_k — дуга-робота, що прямує від вершини-події v_i до вершини-події v_j.

За цією формулою обчислюємо можливий час затримки кожної роботи. На критичному шляху всі затримки нульові, але є нульові затримки також і для деяких інших робіт. Це означає, що можливі затримки на наступних роботах, якщо ця буде виконана без затримки. Якщо ж цю некритичну роботу затримати, то зменшиться час можливої затримки наступних робіт.

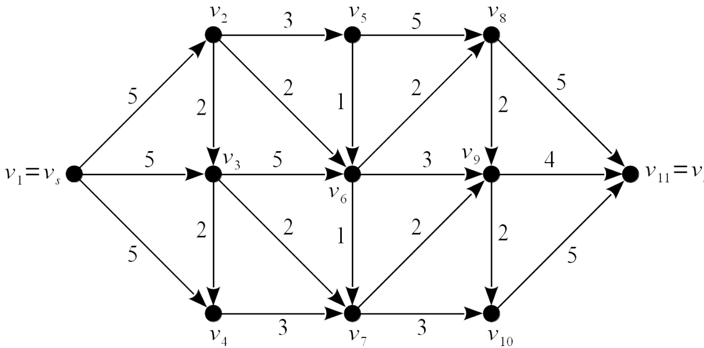


Рис. 8.8. Схема проекту

У Graph Theoty Toolbox є метод `pert`. Вхідним параметром для нього є орграф мережі `G` зі зваженими дугами. Ваги дуг розглядаються як тривалості робіт. Якщо вони не задані, то вважаються одиничними. Вихідним параметром є структура `pres` з полями:

- `pres.lcp` — тривалість критичного шляху (скаляр);
- `pres.cpn` — критичний шлях (вектор-рядок з номерами вершин-подій);
- `pres.cpe` — критичний шлях (вектор-рядок з номерами дуг-робіт);
- `pres.st` — вектор-рядок довжини `n=numnodes(G)` з моментами настання кожної події;
- `pres.td` — вектор-рядок довжини `m=numedges(G)` з моментами можливої затримки кожної роботи.

Приклад 8.4. Для проекту, схема якого показана на рис. 8.8, знайти та нарисувати характеристики діаграми PERT.

Моменти настання подій. Критичний шлях = 22

Рис. 8.9. Моменти настання кожної події

```
s = [1 1 1 2 3 2 2 3 3 4 5 6 5 6 6 7 7 8 ...
  9 8 9 10]; % події-початки
t = [2 3 4 3 4 5 6 6 7 7 6 7 8 8 9 9 10 9 ...
  10 11 11 11]; % події-кінці
w = [5 5 5 2 2 3 2 5 2 3 1 1 5 2 3 2 3 2 ...
  2 5 4 4]; % тривалості робіт
x = [0 1 1 1 2 2 2 3 3 3 4]; % координати вершин
y = [1 2 1 0 2 1 0 2 1 0 1];
G = digraph(s,t,w); % створили орграф мережи PERT
pres = pert(G); % розв’язали задачу
EdgeWidth = ones(numedges(G),1); % товщини ліній робіт
EdgeWidth(pres.cpe) = 4; % товщини ліній на крит. шляху
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",pres.st, ...
  "EdgeLabel",G.Edges.Weight,"LineWidth",EdgeWidth, ...
  "ArrowSize",12) % нарисували
title("Моменти настання подій. Критичний шлях = " + ...
  pres.lcp) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("Pert1","-dpng") % зберегли рисунок у файл
figure % нове вікно фігури
plot(G,"XData",x,"YData",y,"NodeLabel",pres.st, ...
  "EdgeLabel",pres.td,"LineWidth",EdgeWidth, ...
  "ArrowSize",12) % нарисували
title("Можливі затримки робіт. Критичний шлях = " + ...
```

```
   pres.lcp) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
print("Pert2","-dpng") % зберегли рисунок у файл
```

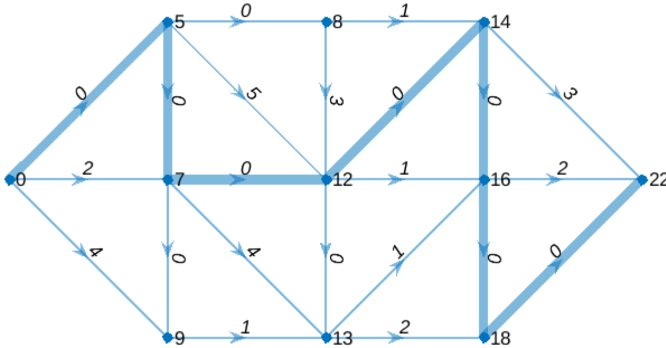


Рис. 8.10. Можливий час затримки кожної роботи

На рис. 8.9 товстою лінією виділений критичний шлях. Його довжина дорівнює 22. Біля кожної вершини-події на цьому рисунку проставлене число t_i — момент настання цієї події за планом, а біля кожної дуги — тривалість відповідної роботи. На рис. 8.10 біля кожної роботи проставлений можливий час її затримки. □

8.5. Запитання для перевірки

1. Чим мережа відрізняється від орграфа?
2. Як перетворити орграф на мережу за допомогою MATLAB?
3. Як формулюється задача про максимальний потік у мережі з обмеженими пропускними здатностями?
4. Як розв'язується задача про максимальний потік у MATLAB?
5. Яка задача є двоїстою до задачі про максимальний потік? Як вона формулюється?
6. Як розв'язується задача про мінімальний розріз у MATLAB?
7. Як визначається критичний шлях у мережі PERT?
8. Як визначаються можливі затримки робіт у мережі PERT?
9. Як знаходяться критичний шлях, моменти настання подій та можливі затримки робіт у MATLAB?

9. Ізоморфізм графів

9.1. Постановка задачі

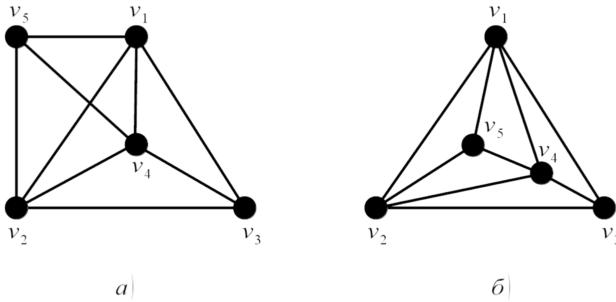


Рис. 9.1. Два різні зображення одного й того самого графа

За означенням у графі, а також в усіх його узагальненнях координати вершин не задані. Як його рисувати на площині — не має значення. Ми його рисуємо так, як зручніше. Наприклад, на рис. 9.1, *а* та 9.1, *б* зображений один і той самий граф.

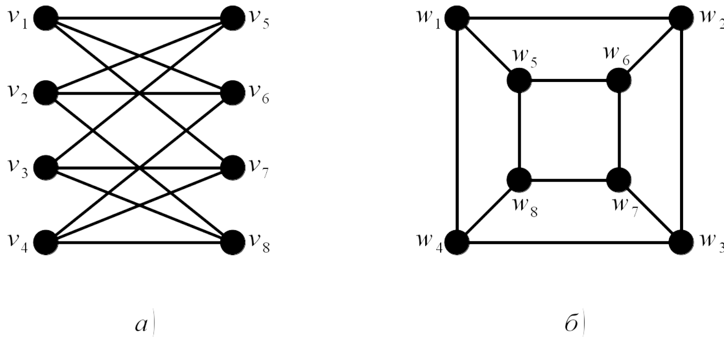


Рис. 9.2. Два різні представлення одного й того самого графа

Але якщо не тільки інакше нарисувати граф, а ще й в іншому порядку перенумерувати вершини, то можна й не побачити, що насправді граф не змінився. Так, графи $G = (V, E)$ та $H = (W, F)$ з рис. 9.2, *а* та 9.2, *б* насправді є двома представленнями одного й того самого графа, якщо їхні вершини перенумеровані таким чином:

$$\begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ w_1 & w_6 & w_8 & w_3 & w_5 & w_2 & w_4 & w_7 \end{pmatrix}. \qquad (9.1)$$

Після такої перенумерації (підстановки) всі 12 ребер поєднують ті самі вершини:

$$\begin{pmatrix} E \\ F \end{pmatrix} = \begin{pmatrix} e_{1,5} & e_{1,6} & e_{1,7} & e_{2,5} & e_{2,6} & e_{2,8} & e_{3,5} & e_{3,7} & e_{3,8} & e_{4,6} & e_{4,7} & e_{4,8} \\ f_{1,5} & f_{1,2} & f_{1,4} & f_{6,5} & f_{6,2} & f_{6,7} & f_{8,5} & f_{8,4} & f_{8,7} & f_{3,2} & f_{3,4} & f_{3,7} \end{pmatrix}. \quad (9.2)$$

Саме такі графи й називаються ізоморфними.

Означення 9.1. *Ізоморфізмом двох графів* (graph isomorphism) $G = (V, E)$ та $H = (W, F)$ називається бієкція (взаємно однозначна відповідність) між множинами вершин V та W:

$$f : V \leftrightarrow W \quad (9.3)$$

така, що будь-які дві вершини v_i та v_j графа G є суміжними тоді й тільки тоді, коли $f(v_i)$ та $f(v_j)$ суміжні в H. □

Існування ізоморфізму позначається: $G \simeq H$.

В цьому означенні йдеться про прості незважені графи. Але його можна узагальнити й на мультиграфи, псевдографи, гіперграфи, орграфи. Якщо ребра зважені, то ізоморфізм вимагає також збереження ваги відповідних ребер. Єдине, що не зберігається, так це порядок (нумерація) ребер. А він і не потрібен: будь-яка множина, у т. ч. й множина ребер E, є неупорядкованою.

Ми будемо розглядати лише найпростіший випадок: прості незважені графи. Основна задача ізоморфізму є такою. Для двох заданих графів $G = (V, E)$ та $H = (W, F)$ треба з'ясувати, чи є вони ізоморфними, тобто чи існує бієкція (9.3). Якщо існує, треба її знайти: побудувати (9.1).

Зазвичай ми нумеруємо вершини натуральними числами від 1 до n. Тоді графи є ізоморфними, якщо існує перестановка $p(i)$ чисел від 1 до n така, що вершини з номерами $p(i)$ та $p(j)$ будуть суміжними в графі H тоді й тільки тоді, коли вершини з номерами i та j суміжні в графі G. Здавалося б, у чому проблема? Треба перевірити всі $P_n = n!$ перестановок і знайти потрібну. Але це неможливо за реальний час. Дійсно, нехай, наприклад, $n = 100$ (невеликий граф). Треба перевірити $P_{100} = 100! \approx 9.33 \times 10^{157}$ перестановок. Нехай у нашому розпорядженні є комп'ютер з об'ємом пам'яті розміром з земну кулю: $R = 6378.1\,\mathrm{km}$, $V = 1.087 \times 10^{21}\,\mathrm{m}^3$. А розмір елемента пам'яті нехай буде мінімально можливим: $10^{-8}\,\mathrm{m}$, як у атома водню. Об'єм такого елемента складає $10^{-24}\,\mathrm{m}^3$, а всього у нашому комп'ютері буде тоді 1.087×10^{45} елементів пам'яті. Час однієї операції візьмемо мінімально можливим: це той час, який світло проходить відстань в один атом водню $10^{-8}\,\mathrm{m}$ зі своєю швидкістю $299792458\,\frac{\mathrm{m}}{\mathrm{s}}$. Цей час становить $3.34 \times 10^{-17}\,\mathrm{s}$. Виходить, що за 1 секунду в одному елементі пам'яті можна здійснити 3×10^{16} операцій.

При абсолютному розпаралелюванні операції в усіх елементах відбуваються водночас, і загальна кількість операцій за 1 секунду на нашому суперкомп'ютері тоді складе 3.258×10^{61}. Якщо навіть вважати, що перевірка однієї перестановки здійснюється за одну операцію, то для перебирання всіх перестановок все одно потрібно буде $2.86 \times 10^{96}\,\mathrm{s} = 9.08 \times 10^{88}$ років, що перевищує час існування Всесвіту.

Отже, перебирання перестановок є неефективним. Треба скорочувати кількість операцій. На жаль, поки ще не відомі алгоритми розв'язання задачі ізоморфізму, поліноміальні за часом відносно n та m. В інтернеті нещодавно промайнуло повідомлення, що, начебто, такий алгоритм створено, але підтвердження цьому немає. Проте інколи є можливість швидко довести неізоморфність графів, що перевіряються. Справа в тому, що є величини, які не змінюються при переході до ізоморфного графа. Кажуть, що такі величини є інваріантними відносно бієкції (9.3). Розглянемо їх.

9.2. Інваріанти графів

Означення 9.2. *Інваріантом графа* (graph invariant) називається числова величина (скалярна, векторна, матрична), що не змінюється при переході до ізоморфного графа. □

З означення випливає, що рівність інваріантів є лише необхідною, але не достатньою умовою ізоморфізму. Тобто якщо інваріанти не співпадають, то графи точно не ізоморфні. А якщо співпадають, то можуть бути ізоморфними, а можуть і не бути. Це дозволяє при дослідженні на ізоморфізм відкидати напевно неізоморфні графи.

Почнемо з найпростішого. Якщо у графів $G_1 = (V_1, E_1)$ та $G_2 = (V_2, E_2)$ різна кількість вершин або ребер, вони не можуть бути ізоморфними, тому що тоді побудувати бієкцію (9.3) неможливо.

Інваріант 9.1. $G_1 \simeq G_2 \Longrightarrow n_1 = n_2$. □

Інваріант 9.2. $G_1 \simeq G_2 \Longrightarrow m_1 = m_2$. □

Так, у ізоморфних графів з рис. 9.2 $n_1 = n_2 = 8$; $m_1 = m_2 = 12$. Але, якщо $n_1 = n_2$ та (або) $m_1 = m_2$, це не означає, що графи ізоморфні. На рис. 9.3 зображені неізоморфні графи: у лівого є два підграфи-трикутники K_3, а у правого немає. Але у кожного з них $n_1 = n_2 = 6$; $m_1 = m_2 = 8$.

Обчислимо в графах G_1 та G_2 ексцентриситети всіх вершин (7.19). Оскільки ізоморфні графи відрізняються один від одного лише нумерацією вершин, то в векторах ексцентриситетів ізоморфних графів повинна бути однакова кількість одиниць, двійок, трійок тощо. Якщо координати векторів ексцентриситетів упорядкувати (наприклад, у порядку зростання), то вони повинні просто співпадати. Позначимо упорядковані вектори

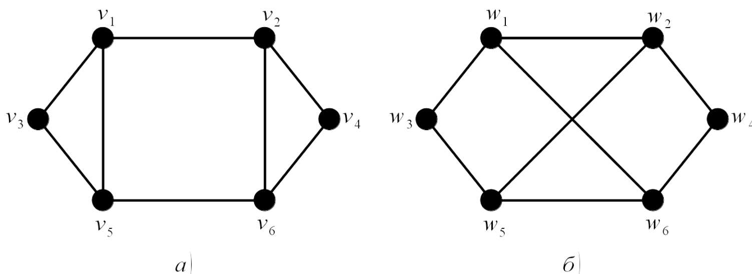


Рис. 9.3. Неізоморфні графи з однаковими розмірами та потужностями

ексцентриситетів графів G_1 та G_2 як $\boldsymbol{\varepsilon}(G_1)$ та $\boldsymbol{\varepsilon}(G_2)$. Тоді маємо наступний інваріант:

Інваріант 9.3. $G_1 \simeq G_2 \Longrightarrow \boldsymbol{\varepsilon}(G_1) = \boldsymbol{\varepsilon}(G_2)\,.\;\square$

Як наслідок, повинні співпадати найменші координати цих векторів (радіуси графів) та найбільші (діаметри):

Інваріант 9.4. $G_1 \simeq G_2 \Longrightarrow \operatorname{rad}(G_1) = \operatorname{rad}(G_2)\,.\;\square$

Інваріант 9.5. $G_1 \simeq G_2 \Longrightarrow \operatorname{diam}(G_1) = \operatorname{diam}(G_2)\,.\;\square$

Так, для ізоморфних графів з рис. 9.2 ексцентриситети усіх вісьмох вершин однакові та дорівнюють 3. Тобто і діаметри, і радіуси цих графів теж дорівнюють 3. На жаль, для обох неізоморфних графів з рис. 9.3 упорядковані вектори ексцентриситетів теж співпадають: вони складаються з чотирьох двійок та двох трійок.

Вже на цьому етапі, якщо виявиться, що $\boldsymbol{\varepsilon}(G_1) = \boldsymbol{\varepsilon}(G_2)$, можна спробувати відшукати перестановку ізоморфізму (9.1), перебираючи не всі $n!$ перестановок, а лише $n_1! \times n_2! \times n_2! \times \ldots \times n_k!$, де k — максимальний ексцентриситет вершини. Тобто можна окремо переставляти між собою вершини з ексцентриситетом 1, з ексцентриситетом 2 тощо. Це суттєво зменшить кількість обчислювань. Наприклад, для графа з $n = 50$ пряме перебирання дає $50! \approx 3.04 \times 10^{64}$ перестановок. Якщо ж, наприклад, упорядкований вектор ексцентриситетів $\boldsymbol{\varepsilon}(G) = \begin{pmatrix} 3 & 3 & 4 & 4 & 4 & 5 & 5 & 7 & 7 & 8 \end{pmatrix}$, то треба перебрати всього лише $(3!)^2 \times (4!)^3 \times (5!)^2 \times (7!)^2 \times 8! \approx 7.34 \times 10^{21}$ перестановок, що значно менше.

Наступний інваріант. Обчислимо відстань від кожної вершини v_i до кожної іншої v_j: $d(v_i, v_j)$, а потім знайдемо їхню суму, яка називається *індексом Вінера* (Wiener index):

$$W(G) = \sum_{\forall i,j} d(v_i, v_j)\,. \tag{9.4}$$

Якщо графи ізоморфні, то між множинами відстаней (як і між множинами вершин) існує бієкція. Як наслідок, сума відстаней не змінюється.

Інваріант 9.6. $G_1 \simeq G_2 \Longrightarrow W(G_1) = W(G_2)$. □

Матриці відстаней між будь-якою парою вершин графів з рис. 9.2 не співпадають:

$$\boldsymbol{D}_G = \begin{pmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 1 & 3 \\ 2 & 0 & 2 & 2 & 1 & 1 & 3 & 1 \\ 2 & 2 & 0 & 2 & 1 & 3 & 1 & 1 \\ 2 & 2 & 2 & 0 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 3 & 0 & 2 & 2 & 2 \\ 1 & 1 & 3 & 1 & 2 & 0 & 2 & 2 \\ 1 & 3 & 1 & 1 & 2 & 2 & 0 & 2 \\ 3 & 1 & 1 & 1 & 2 & 2 & 2 & 0 \end{pmatrix};$$

$$\boldsymbol{D}_H = \begin{pmatrix} 0 & 1 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 2 & 2 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 3 & 2 & 1 & 2 \\ 1 & 2 & 1 & 0 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 0 & 1 & 2 & 1 \\ 2 & 1 & 2 & 3 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 2 & 2 & 1 & 0 & 1 \\ 2 & 3 & 2 & 1 & 1 & 2 & 1 & 0 \end{pmatrix}, \tag{9.5}$$

але суми елементів над головною діагоналлю для них однакові: $W(G) = W(H) = 48$. На жаль, те ж саме маємо і для неізоморфних графів з рис. 9.3. Їхні матриці відстаней такі:

$$\boldsymbol{D}_G = \begin{pmatrix} 0 & 1 & 1 & 2 & 1 & 2 \\ 1 & 0 & 2 & 1 & 2 & 1 \\ 1 & 2 & 0 & 3 & 1 & 2 \\ 2 & 1 & 3 & 0 & 2 & 1 \\ 1 & 2 & 1 & 2 & 0 & 1 \\ 2 & 1 & 2 & 1 & 1 & 0 \end{pmatrix}; \quad \boldsymbol{D}_H = \begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 1 \\ 1 & 0 & 2 & 1 & 1 & 2 \\ 1 & 2 & 0 & 3 & 1 & 2 \\ 2 & 1 & 3 & 0 & 2 & 1 \\ 2 & 1 & 1 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 & 1 & 0 \end{pmatrix}, \tag{9.6}$$

і суми елементів над головною діагоналлю для них співпадають: $W(G) = W(H) = 23$.

Обчислимо вектор (наприклад, вектор-рядок, хоча це не має значення) ступенів вершин. Найпростіше це зробити, додавши стовпці матриці інцидентності або суміжності вершин. Якщо, наприклад, виявиться, що у графі G_1 є 8 вершин ступеня 5, а у графі G_2 є 12 таких вершин, то ніякої бієкції (9.3) встановити між вершинами G_1 та G_2 не вдасться: для

чотирьох вершин ступеня 5 у G_2 не знайдеться відповідних у G_1. Отже, необхідною умовою ізоморфізму є співпадіння кількості нулів, одиниць, двійок, трійок тощо у векторах ступенів вершин досліджуваних графів. Якщо ці вектори упорядкувати (наприклад, у порядку зростання), та позначити упорядковані вектори як $\boldsymbol{p}(G_1)$ та $\boldsymbol{p}(G_2)$, то ці вектори повинні просто співпадати, як і вектори ексцентриситетів.

Інваріант 9.7. $G_1 \simeq G_2 \Longrightarrow \boldsymbol{p}(G_1) = \boldsymbol{p}(G_2)\,.\;\square$

Матриці суміжності вершин для графів з рис. 9.2 мають вигляд:

$$\boldsymbol{B}_G = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$\boldsymbol{B}_H = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}, \qquad (9.7)$$

а упорядковані вектори ступенів вершин, як і упорядковані вектори ексцентриситетів, складаються лише з трійок:

$$\begin{aligned} \boldsymbol{p}(G) &= \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}; \\ \boldsymbol{p}(H) &= \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}. \end{aligned} \qquad (9.8)$$

Для графів з рис. 9.3 їхні матриці суміжності вершин такі:

$$\boldsymbol{B}_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}; \quad \boldsymbol{B}_H = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \qquad (9.9)$$

а упорядковані вектори ступенів вершин мають вигляд:

$$\boldsymbol{p}(G)=\left(\begin{array}{cccccc}2 & 2 & 3 & 3 & 3 & 3\end{array}\right);\quad \boldsymbol{p}(H)=\left(\begin{array}{cccccc}2 & 2 & 3 & 3 & 3 & 3\end{array}\right), \tag{9.10}$$

тобто теж однакові, хоча ці графі не є ізоморфними.

Як і у випадку з вектором ексцентриситетів, вектор ступенів вершин також можна використовувати для зменшення кількості перестановок, які треба перебирати. Тобто можна окремо переставляти вершини ступеня 1, окремо ступеня 2 тощо.

Можна також поєднати дослідження за ступенями вершин та ексцентриситетами. Для цього треба розбити множини вершин обох порівнюваних графів на класи еквівалентності таким чином, що в один клас потрапляли вершини з однаковим ступенем та однаковим ексцентриситетом. Потім можна порівнювати ці класи. Якщо, наприклад, у графі G_1 є 4 вершини ступеня 2 з ексцентриситетом 3, а у графі G_2 таких вершин 5, то, зрозуміло, такі графи точно не є ізоморфними. Якщо ж є повне співпадіння кількостей вершин у кожному класі еквівалентності, можна спробувати переставити вершини окремо у кожному класі.

Наприклад, для графів з рис. 9.2 ми не отримаємо спрощень, тому що в них і вектори ступенів вершин, і вектори ексцентриситетів складаються лише з трійок. А для графів з рис. 9.3 вектори ексцентриситетів та ступенів вершин (неупорядковані) є такими:

$$\begin{array}{l}\boldsymbol{\varepsilon}(G)=\left(\begin{array}{cccccc}2 & 2 & 3 & 3 & 2 & 2\end{array}\right);\ \boldsymbol{\varepsilon}(H)=\left(\begin{array}{cccccc}2 & 2 & 3 & 3 & 2 & 2\end{array}\right);\\ \boldsymbol{p}(G)=\left(\begin{array}{cccccc}3 & 3 & 2 & 2 & 3 & 3\end{array}\right);\ \boldsymbol{p}(H)=\left(\begin{array}{cccccc}3 & 3 & 2 & 2 & 3 & 3\end{array}\right).\end{array} \tag{9.11}$$

У обох цих графів є по 4 вершини ступеня 3 та ексцентриситету 2 і по 2 вершини ступеня 2 та ексцентриситету 3. При пошуку ізоморфізму (9.3) можна окремо пробувати порівнювати $\{v_3, v_4\}$ з $\{w_3, w_4\}$, і окремо $\{v_1, v_2, v_5, v_6\}$ з $\{w_1, w_2, w_5, w_6\}$.

Зі ступенями вершин пов'язаний ще один інваріант: *індекс Рандича* (Randič index), який використовується в математичній хімії та хемоінформатиці. Треба знайти ступені кожної вершини $\deg(v_i)$; потім для кожного ребра $e_{ij}=\{v_i,\, v_j\}$ обчислити величини $\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}$ та додати їх:

$$r(G)=\sum_{\forall e_{ij}\in E}\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}. \tag{9.12}$$

Це й є індекс Рандича. Якщо структура графа не змінюється, то незалежно від нумерації вершин у кожного ребра графа G є свій "двійник" у ізоморфному графі H з такими самими ступенями вершин на його кінцях.

Інваріант 9.8. $G_1 \simeq G_2 \Longrightarrow r(G_1)=r(G_2)$. □

Неважко усно порахувати, що для обох графів з рис. 9.2 індекс Рандича дорівнює 4. Але і для обох неізоморфних графів з рис. 9.3 цей індекс теж однаковий: $r(G) = r(H) = \frac{2\sqrt{6}+4}{3} \approx 2.966$.

Наступний інваріант — це хроматичне число $\chi(G)$, або кількість кольорів у мінімальній правильній розфарбовці (див. главу 3).

Інваріант 9.9. $G_1 \simeq G_2 \Longrightarrow \chi(G_1) = \chi(G_2)$. □

Наприклад, граф на рис. 9.2, *а* правильно фарбується двома фарбами: ліва доля однією, а права — іншою. У графі з рис. 9.2, *б* однією фарбою можна пофарбувати вершини w_1, w_3, w_6, w_8; а другою — чотири інші. І, нарешті, за допомогою цього інваріанту ми можемо розрізнити неізоморфні графи з рис. 9.3! Для правильної розфарбовки графа з рис. 9.3, *а* потрібно три фарби, бо в ньому є трикутники K_3. А вершини графа з рис. 9.3, *б* правильно фарбуються лише двома фарбами. Для w_2, w_3, w_6 обираємо першу фарбу, а для w_1, w_4, w_5 — другу.

Розглянемо наступний інваріант, пов'язаний з матрицями суміжності вершин (9.7) або (9.9). При бієкції (9.3) вершини перенумеровуються, що відповідає перестановці рядків та стовпців матриці суміжності вершин. З лінійної алгебри відомо, що для перестановки стовпців якоїсь матриці треба помножити її на квадратну матрицю перестановок $\boldsymbol{T}$. У першому рядку $\boldsymbol{T}$ буде одиниця на тому місці, куди треба перенести перший стовпчик, у другому рядку — одиниця на тому місці, куди треба перенести другий, і т. д., а всі інші елементи $\boldsymbol{T}$ нульові. Так, матриця перестановки (9.1) має вигляд:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{9.13}$$

Множення $\boldsymbol{B}_G$ на $\boldsymbol{T}$ переставляє стовпці у $\boldsymbol{B}_G$ згідно з (9.1):

$$\boldsymbol{B}_G\boldsymbol{T} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{9.14}$$

Як бачимо, 1-й стовпчик залишився 1-м, 2-й став 6-м, 3-й — 8-м тощо. Для перестановки рядків тепер треба помножити транспоновану матрицю $\boldsymbol{T}^T$ на результат (9.14). Рядки у $\boldsymbol{B}_G$ будуть переставлені у такому самому порядку, як і раніше стовпці:

$$\boldsymbol{T}^T\boldsymbol{B}_G\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{9.15}$$

Після такої перестановки стовпців і рядків ми отримали $\boldsymbol{B}_H$:

$$\boldsymbol{B}_H = \boldsymbol{T}^T \boldsymbol{B}_G \boldsymbol{T}. \tag{9.16}$$

Але матриця $\boldsymbol{T}$ є ортогональною: норми всіх стовпців одиничні, а скалярні добутки двох будь-яких різних стовпців є нулями. Для такої матриці $\boldsymbol{T}^{-1} = \boldsymbol{T}^T$, і має місце формула:

$$\boldsymbol{B}_H = \boldsymbol{T}^{-1} \boldsymbol{B}_G \boldsymbol{T}. \tag{9.17}$$

Саме за такою формулою змінюється й матриця лінійного оператора при переході до іншого базису. У нашому випадку матриці $\boldsymbol{B}_G$ та $\boldsymbol{B}_H$ симетричні, а $\boldsymbol{T}$ ортогональна. То ж формула (9.17) дає зміну матриці симетричного лінійного оператора при переході до іншого ортонормованого базису (ОНБ). Причому тут матриця $\boldsymbol{T}$ описує перехід не до будь-якого ОНБ, а перехід з перенумерацією осей координат у E_n згідно з бієкцією (9.3). З теорії лінійних операторів відомо, що при зміні базису не змінюються характеристичний поліном його матриці $D(\lambda)$ і, відповідно, його корені (тобто спектр оператора). Отже, характеристичний поліном матриці суміжності вершин (та його корені) є інваріантом.

Інваріант 9.10. $G_1 \simeq G_2 \Longrightarrow D_1(\lambda) = D_2(\lambda)$. $\square$

На практиці легше перевіряти коефіцієнти характеристичного поліному (цілі числа).

Для обох графів з рис. 9.2:

$$D(\lambda) = |\boldsymbol{B}_G - \lambda \boldsymbol{E}| = |\boldsymbol{B}_H - \lambda \boldsymbol{E}| = \lambda^8 - 12\lambda^6 + 30\lambda^4 - 28\lambda^2 + 9. \tag{9.18}$$

А для неізоморфних графів з рис. 9.3 вони різні:

$$\begin{cases} D_1(\lambda) = |\boldsymbol{B}_G - \lambda \boldsymbol{E}| = \lambda^6 - 8\lambda^4 - 4\lambda^3 + 12\lambda^2 + 8\lambda; \\ D_2(\lambda) = |\boldsymbol{B}_H - \lambda \boldsymbol{E}| = \lambda^6 - 8\lambda^4 + 4\lambda^2. \end{cases} \tag{9.19}$$

З іншого боку, є неізоморфні графи, для яких $D(\lambda)$ співпадають. Наприклад, графи на рис. 9.4 напевно неізоморфні: у них різні ступені вершин. Але характеристичні поліноми в них однакові:

$$D(\lambda) = |\boldsymbol{B}_G - \lambda \boldsymbol{E}| = |\boldsymbol{B}_H - \lambda \boldsymbol{E}| = -\lambda^5 + 4\lambda^3. \tag{9.20}$$

Існують і інші інваріанти, у т. ч. й достатні. Якщо, наприклад, виписати в рядок елементи матриці суміжності вершин $\boldsymbol{B}$ над головною діагоналлю: $b_{12}, b_{13}, b_{23}, b_{14}, b_{24}, b_{34}, \ldots, b_{1,n}, \ldots, b_{n-1,n}$, то отримаємо послідовність нулів та одиниць, яку можна розглядати як двійкове число. Це число після переведення в десяткову систему координат називається

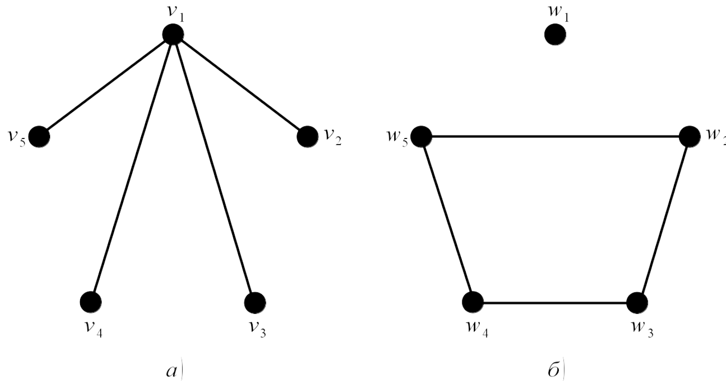


Рис. 9.4. Неізоморфні графи з однаковими характеристичними поліномами

кодом матриці $\boldsymbol{B}$ та позначається $\mu(G)$. При різній нумерації вершин, тобто при різних перестановках рядків і стовпців $\boldsymbol{B}$, код може бути більшим або меншим. Так ось, найменше та найбільше значення коду: $\mu_{\min}(G)$ та $\mu_{\max}(G)$ є достатніми інваріантами при заданому конкретному n.

Інваріант 9.11. $G_1 \simeq G_2 \iff$
$\iff (\mu_{\min}(G_1) = \mu_{\min}(G_2)) \wedge (\mu_{\max}(G_1) = \mu_{\max}(G_2)) \wedge (n_1 = n_2)$. □

На жаль, задача знаходження перестановки, яка надає коду матриці $\boldsymbol{B}$ мінімальне та максимальне значення, нічим не легша за звичайне перебирання перестановок.

Як бачимо, задача перевірки графів на ізоморфізм є досить складною. У MATLAB є два вбудовані методи для розв'язання задачі ізоморфізму: `isisomorphic` та `isomorphism`. У кожному з них вхідними параметрами є два графи (мультиграфи, псевдографи, орграфи), а також додаткові параметри, що визначають ідентифікацію вершин.

Метод `isisomorphic` повертає булівську змінну, що визначає: ізоморфні графи чи ні. Метод `isomorphism` повертає два вихідні параметри: вектори-стовпчики з перестановками вершин та ребер, якщо графи ізоморфні. Для неізоморфних графів повертаються пусті масиви.

Приклад 9.1. Знайти ізоморфізм графів з з рис. 9.2.

```
s1 = [1 1 1 2 2 3 3 4 5 5 7 7]; % початки ребер
t1 = [2 4 5 3 6 4 7 8 6 8 6 8]; % кінці ребер
x1 = [1 4 4 1 2 3 3 2]; % координати вершин
y1 = [4 4 1 1 3 3 2 2];
```

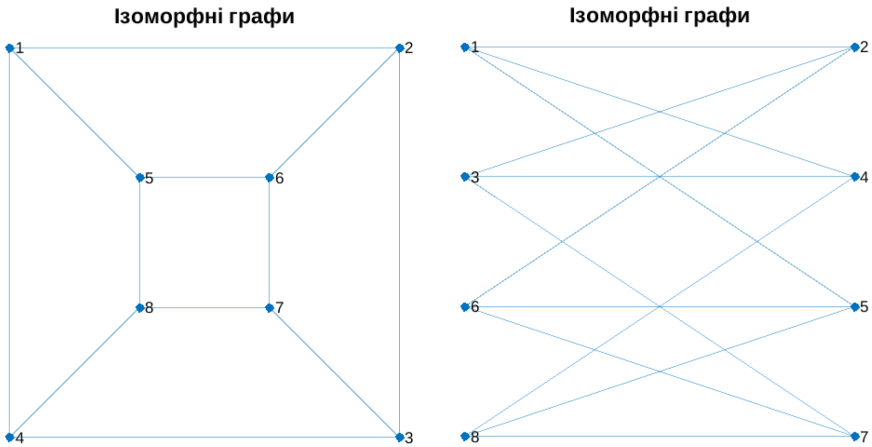


Рис. 9.5. Ізоморфізм двох графів

```
s2 = [1 1 1 2 2 2 3 3 3 4 4 4]; % початки ребер
t2 = [5 6 7 5 6 8 5 7 8 6 7 8]; % кінці ребер
x2 = [1 1 1 1 4 4 4 4]; % координати вершин
y2 = [4 3 2 1 4 3 2 1];
G1 = graph(s1,t1); % створили перший граф
G2 = graph(s2,t2); % створили другий граф
p = isomorphism(G1,G2); % знаходимо ізоморфізм
if isempty(p)
  t = "Не ізоморфні графи";
else
  t = "Ізоморфні графи";
end
figure("Position",[100 100 960 420]) % нове вікно фігури
subplot(1,2,1) % ліва частина
plot(G1,"XData",x1,"YData",y1) % перший граф
title(t) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
subplot(1,2,2) % права частина
[s,is] = sort(p); % номери вершин
plot(G2,"XData",x2,"YData",y2,"NodeLabel",is) % другий граф
title(t) % заголовок
axis("equal") % однаковий масштаб уздовж осей координат
axis("off") % прибрали осі
```

```
print("Isomorphism","-dpng") % зберегли рисунок у файл
```

На рис. 9.5 показані два ізоморфні графи. Вершини G_2 перенумеровані у відповідності до G_1. □

9.3. Запитання для перевірки

1. Що таке ізоморфізм?
2. Що таке перестановка?
3. Що таке інваріант графа?
4. Що таке індекс Вінера?
5. Що таке індекс Рандича?
6. Що таке характеристичний поліном?
7. Як розв'язати задачу ізоморфізму в MATLAB?

10. Індивідуальні домашні завдання

10.1. Зміст завдання

Вхідними даними є орграф зі зваженими вершинами та дугами, він використовується в ІДЗ-2. Якщо замінити дуги ребрами (тобто прибрати стрілки), то отримаємо простий граф зі зваженими вершинами та ребрами, він використовується для ІДЗ-1.

Перенумеруйте вершини. Задайте координати вершин (приблизно, це потрібно лише для рисування). Перенумеруйте ребра (дуги). Задайте структуру графа (орграфа), тобто визначте, які вершини поєднує кожне ребро (дуга). Збережіть координати вершин, ваги вершин, структуру графа (орграфа), ваги ребер (дуг) у якомусь текстовому файлі або електронній таблиці. За допомогою MATLAB, сайту [4] або будь-якого іншого ресурсу розв'яжіть та нарисуйте розв'язки таких задач.

10.1.1. Задачі на простому графі

1. Максимальне зважене паросполучення (реберне пакування).
2. Максимальна зважена незалежна множина вершин (вершинне пакування).
3. Мінімальне зважене реберне покриття.
4. Мінімальне зважене вершинне покриття.
5. Мінімальна зважена домінуюча (зовнішньо стійка) множина ребер.
6. Мінімальна зважена домінуюча (зовнішньо стійка) множина вершин.
7. Максимальний зважений повний підграф (кліка).
8. Мінімальна правильна розфарбовка вершин.
9. Мінімальна правильна розфарбовка ребер.
10. Мінімальне зважене остовне дерево.
11. Фундаментальна система циклів.
12. Фундаментальна система коциклів (розрізів).
13. Ексцентриситети вершин, радіус та діаметр графа, центральні та периферійні вершини.
14. Перевірте граф на ейлеровість; якщо він ейлерів (напівейлерів), знайдіть ейлерів цикл (шлях).

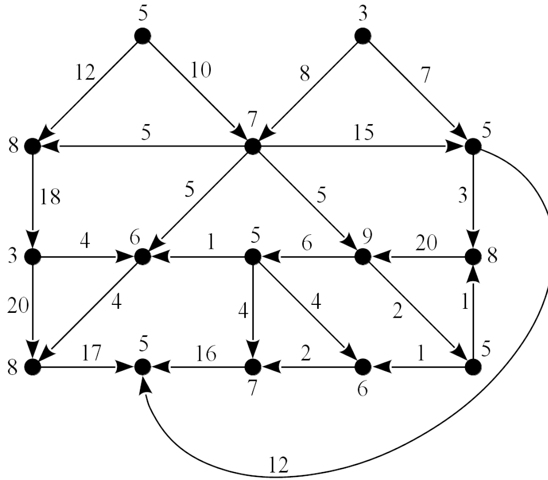
10.1.2. Задачі на орграфі

1. Сильно зв'язані компоненти та їхнє часткове упорядкування.
2. Найкоротший шлях від якоїсь вершини з якоїсь першої компоненти сильної зв'язності до якоїсь вершини з якоїсь останньої компоненти сильної зв'язності (вершини та компоненти оберіть самі).

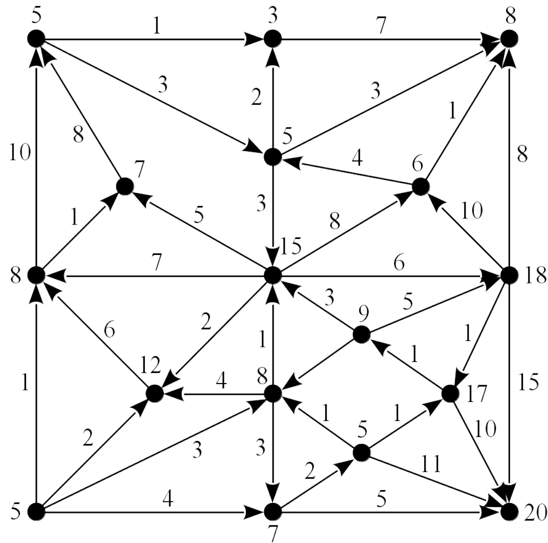
Створіть з орграфа мережу. Для цього стягніть кожну компоненту сильної зв'язності до однієї вершини. Ваги дуг, що йдуть до компоненти, додаються; що йдуть від компоненти — теж. За необхідності додайте фіктивну початкову вершину та (або) кінцеву. У побудованій мережі розв'яжіть такі задачі.

3. Максимальний потік (вага дуги — обмеження на її пропускну здатність, у фіктивних дугах пропускна здатність необмежена).
4. Мінімальний розріз (вага дуги — обмеження на її пропускну здатність, у фіктивних дугах пропускна здатність необмежена).
5. Критичний шлях, моменти настання кожної події та можливі затримки робіт у мережі PERT (вага дуги — це тривалість роботи; у фіктивних дугах тривалість роботи нульова).

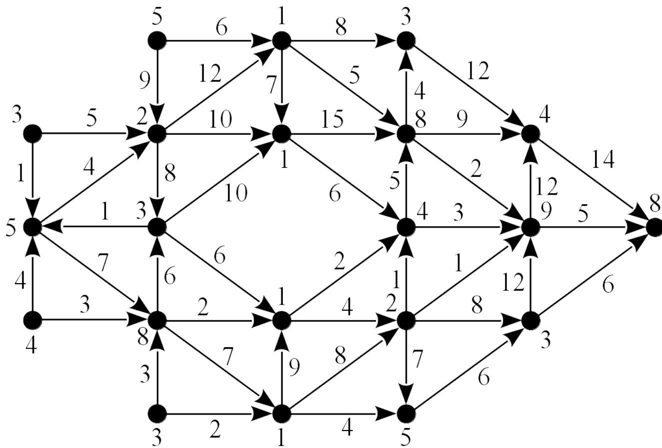
10.2. Варіанти завдань



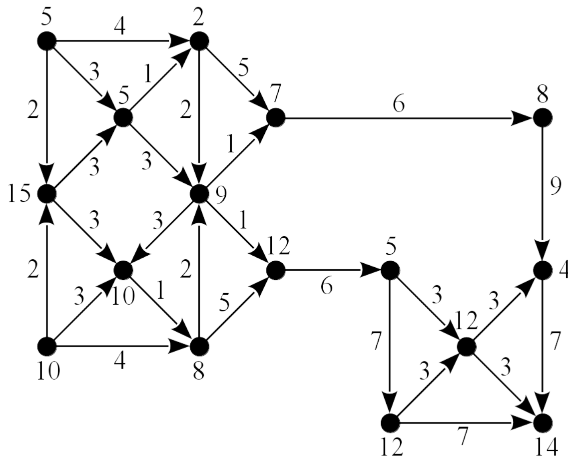
Варіант 1



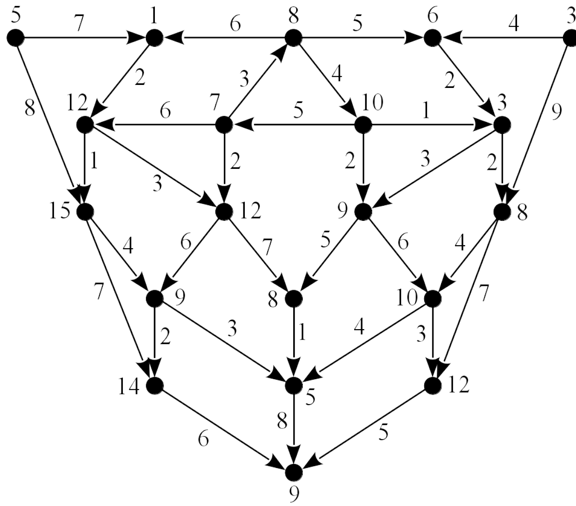
Варіант 2



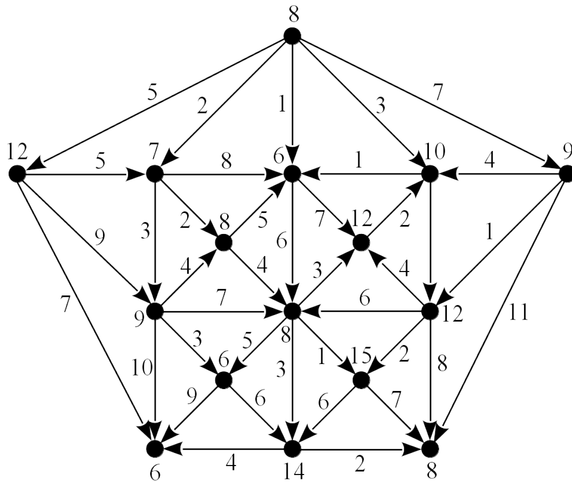
Варіант 3



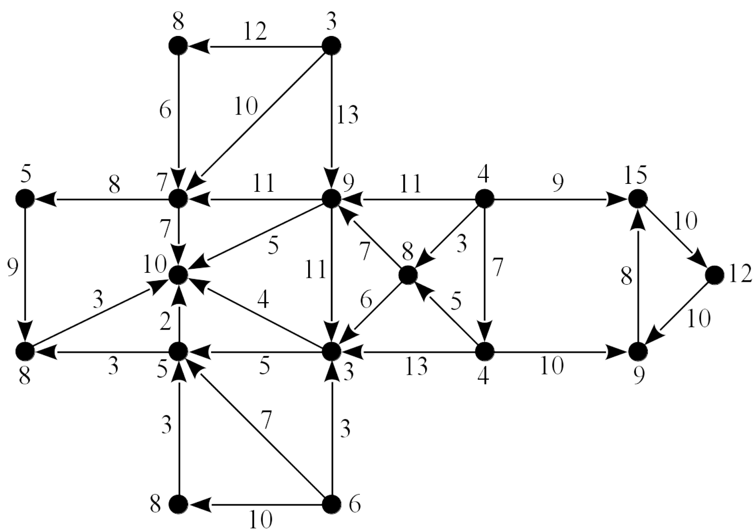
Варіант 4



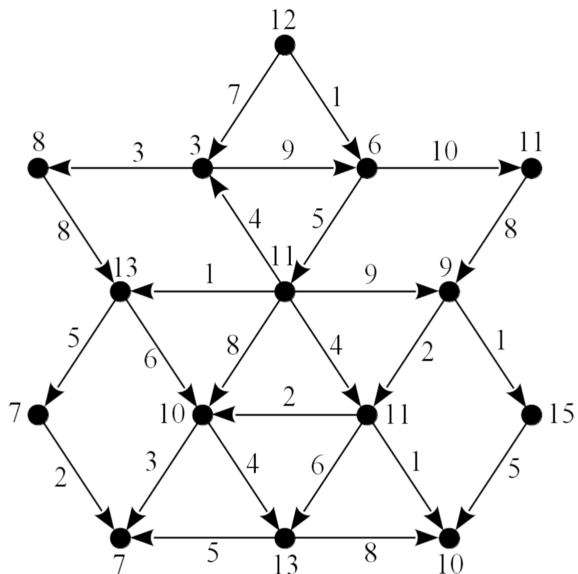
Варіант 5



Варіант 6



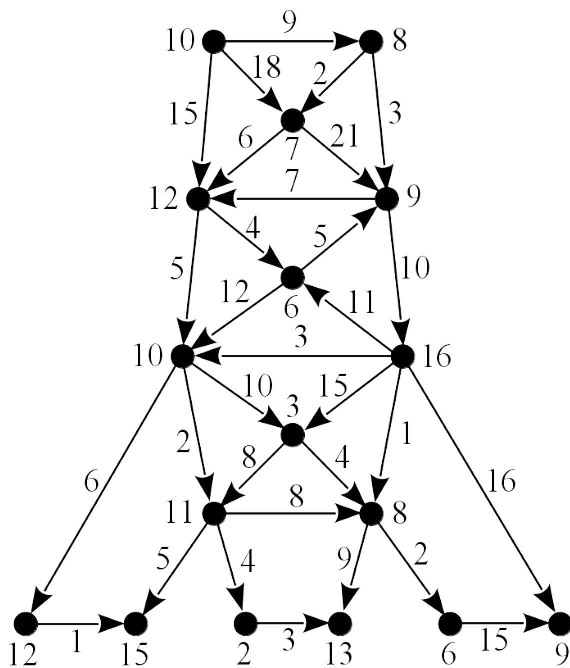
Варіант 7



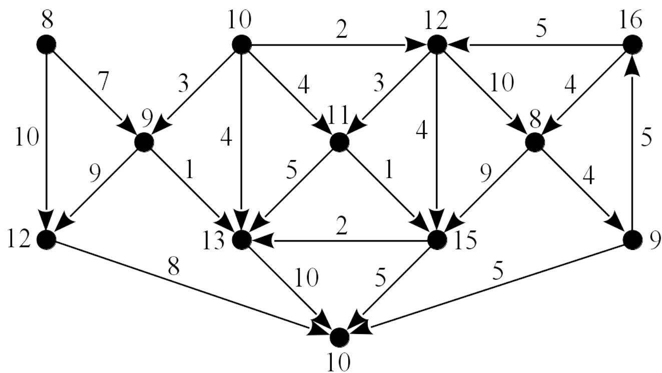
Варіант 8

Варіант 9

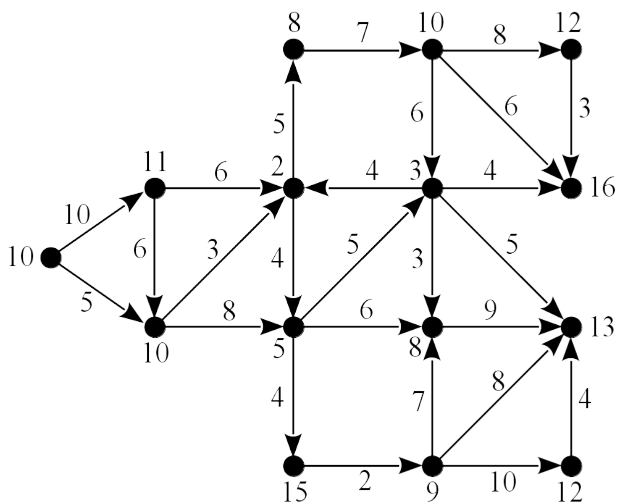
Варіант 10



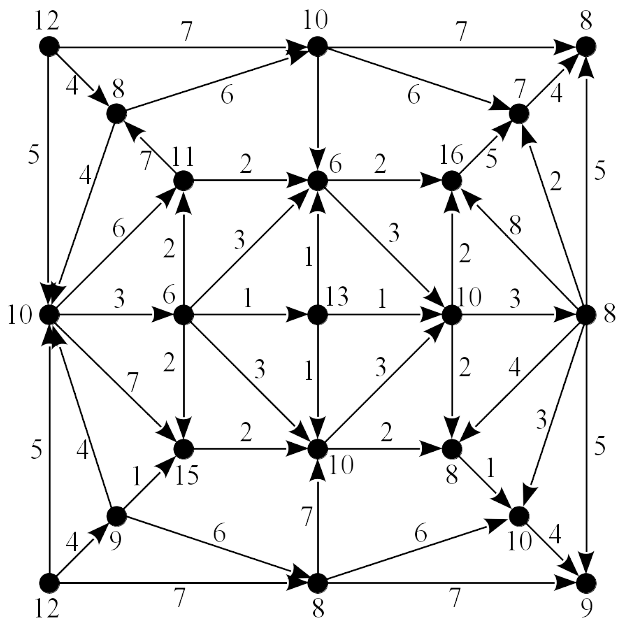
Варіант 11



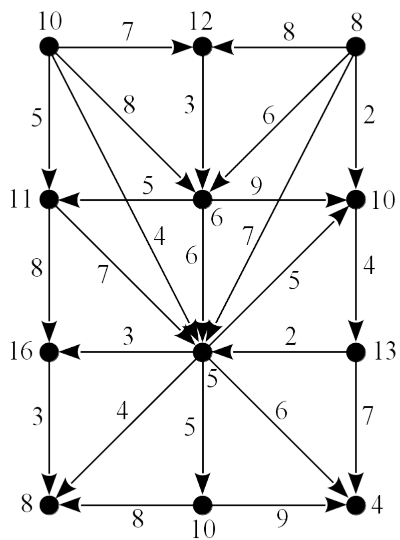
Варіант 12



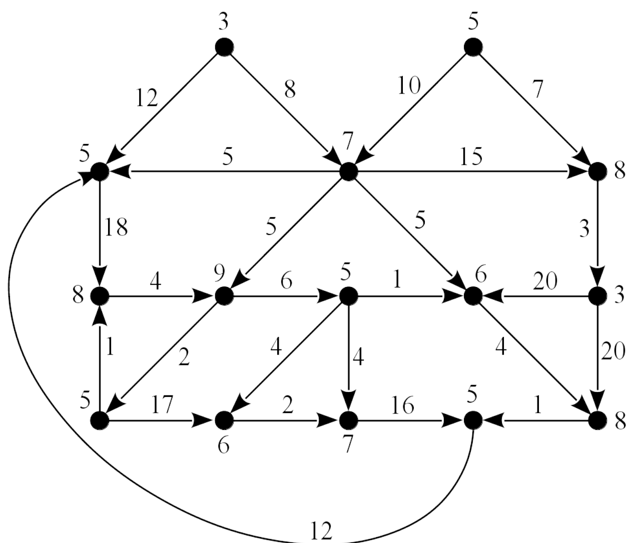
Варіант 13



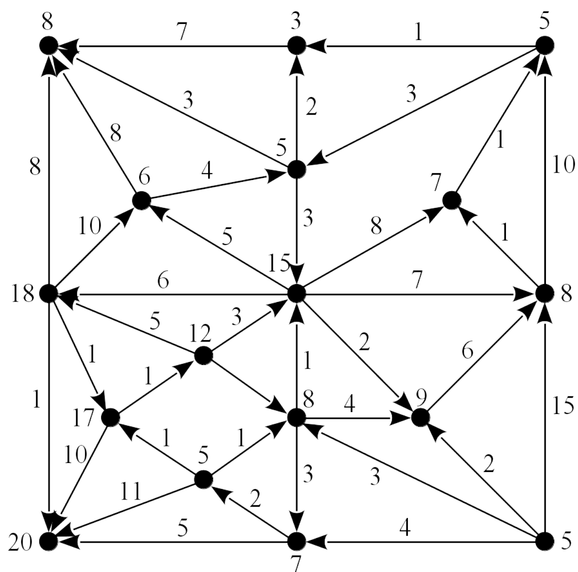
Варіант 14



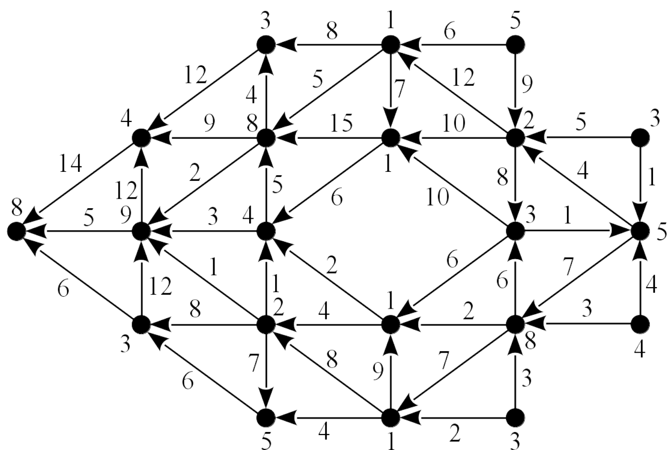
Варіант 15



Варіант 16



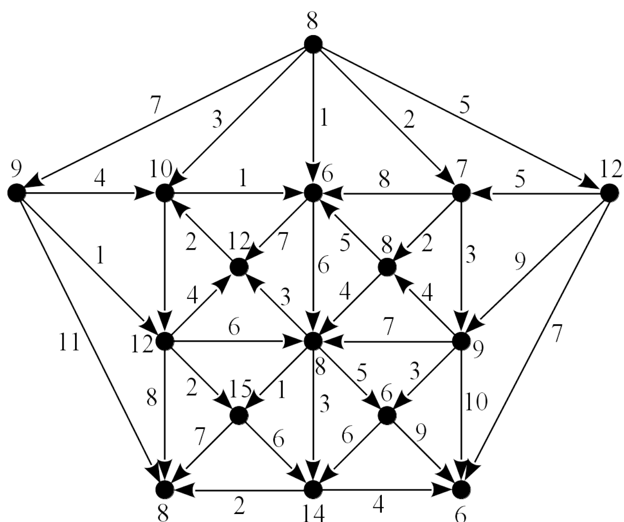
Варіант 17



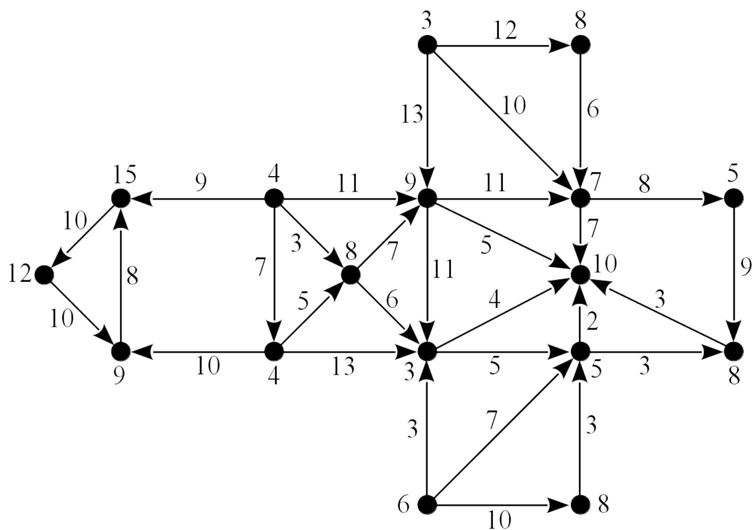
Варіант 18

Варіант 19

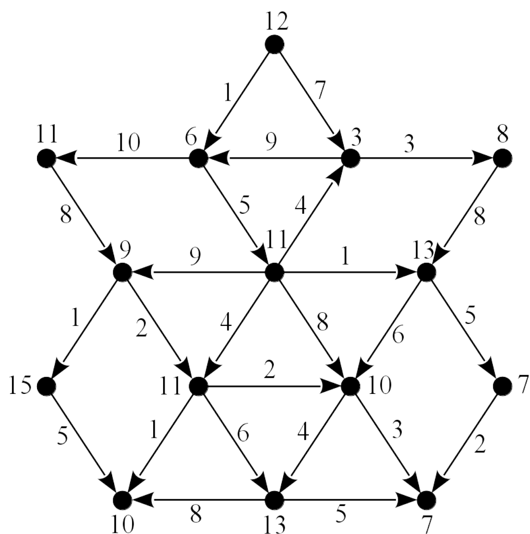
Варіант 20



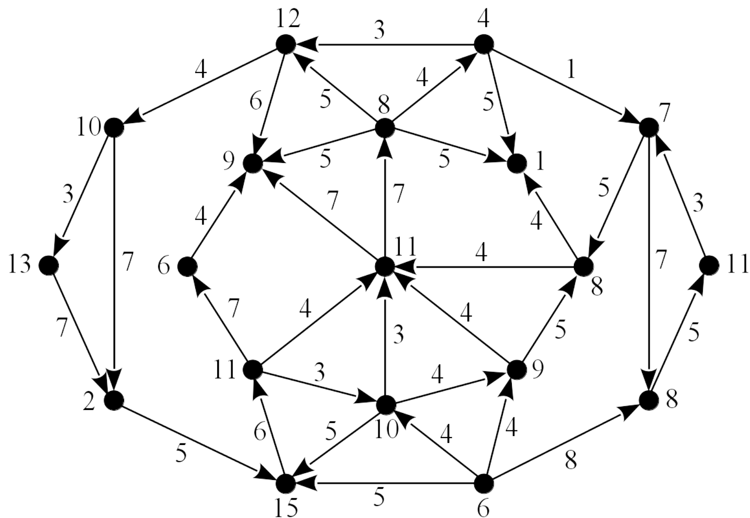
Варіант 21



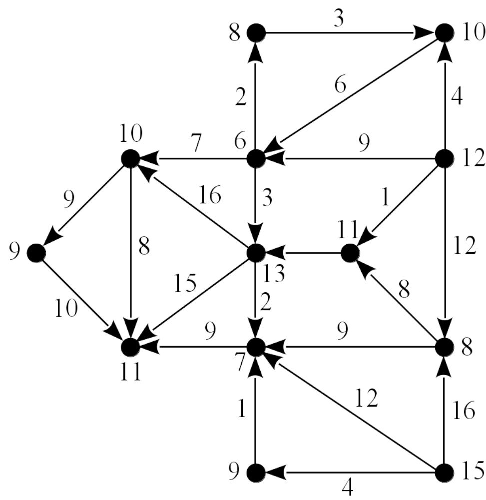
Варіант 22



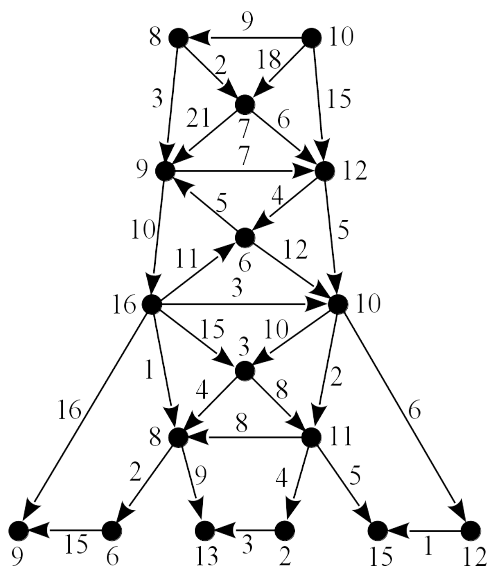
Варіант 23



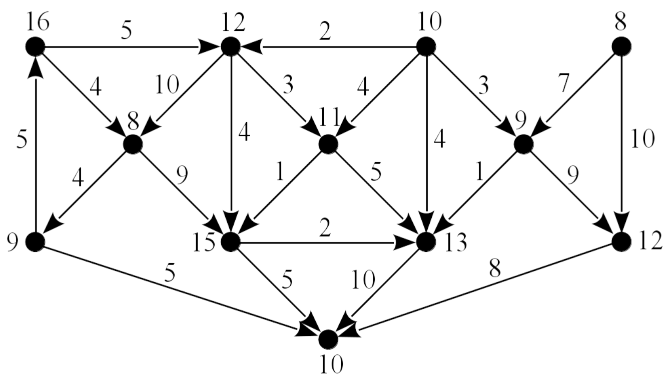
Варіант 24



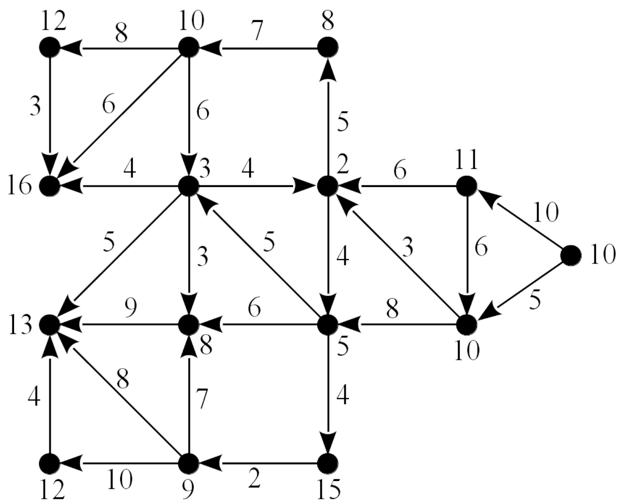
Варіант 25



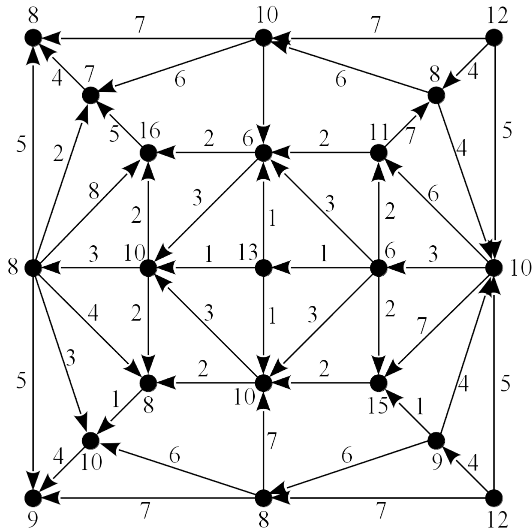
Варіант 26



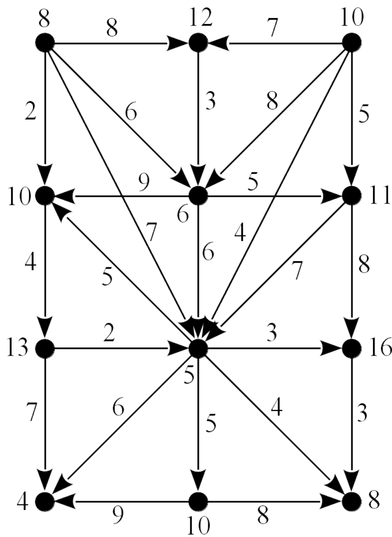
Варіант 27



Варіант 28



Варіант 29



Варіант 30

Предметний покажчик

A

acyclic digraph ациклічний орграф 145
adjacency matrix матриця суміжності вершин 32
adjacent суміжний 13
antisymmetric антисиметричне 143
approachability досяжність 142
arc дуга 18
arrow стрілка 18
asymmetric асиметричне 143

B

base, basis of matroid база (базис) матроїда 95
binary relation бінарне відношення 142
bipartite graph дводолевий граф 15

C

Cartesian product декартів добуток 142
central vertex центральна вершина 181
chromatic index хроматичний індекс 86
chromatic number хроматичне число 81
circuit of matroid цикл матроїда 123
clique кліка 15, 77
cocycle basis коцикловий базис 135
cocycle rank коциклічний ранг 135
cocycle коцикл 131
colored edges розфарбовані ребра 86
colored vertices розфарбовані вершини 81
coloring розфарбовка 14
complete graph повний граф 15
complete subgraph повний підграф 77
connected graph зв'язний граф 98
connectivity matrix матриця досяжності 145
covering (edge) реберне покриття 57
covering (vertex) вершинне покриття 61
critical path критичний шлях 198
cut розріз 131
cut-set розріз 194
cycle basis цикловий базис 123
cycle of matroid цикл матроїда 123
cycle цикл 99
cyclic rank циклічний ранг 123

D

degree of a vertex ступінь вершини 31
dependent set (of matroid) залежна множина матроїда 95
dependent set of matroid залежна множина матроїда 122
diameter of a graph діаметр графа 181
digraph орграф 18
Dijkstra algorithm алгоритм Дейкстри 166
Dirac theorem теорема Дірака 118
direct sum пряма сума 120, 131
distance відстань 181
dominating set (edge) домінуюча множина ребер 67
dominating set (vertex) домінуюча множина вершин 72
dual hypergraph двоїстий гіперграф 17

E

eccentricity of a vertex ексцентриситет вершини 181
edge ребро 10
edge-adjacency matrix матриця суміжності ребер 36
empty graph пустий граф 15

equivalence еквівалентне 143
Eulerian cycle ейлерів цикл 110
Eulerian multigraph ейлерів мультиграф 110
Eulerian path ейлерів шлях 111
event подія 197
external stability set (edge) зовнішньо стійка множина ребер 68
external stability set (vertex) зовнішньо стійка множина вершин 72

F

Fleury's Algorithm алгоритм Фльорі 113
Floyd-Warshall algorithm алгоритм Флойда-Воршола 176
Ford-Fulkerson theorem теорема Форда-Фалкерсона 196
forest ліс 101
fundamental cocycles фундаментальна система коциклів 135
fundamental cycles фундаментальна система ціклів 123

G

graph граф 10
graphic matroid графовий матроїд 102
greedy algorithm жадібний алгоритм 81, 86, 92
ground set (of matroid) носій матроїда 94

H

Hamiltonain cycle гамільтонів цикл 117
Hamiltonain graph гамільтонів граф 117
Hamiltonain path гамільтонів шлях 117
hyperedge гіперребро 13
hypergraph гіперграф 12

I

incidence matrix матриця інцидентності 27
incidence відповідний 17
incident інцидентний 13
independent cocycles незалежні коцикли 135
independent cycles незалежні цикли 124
independent set (of vertex) незалежна множина вершин 53
independent sets (of matroid) незалежні множини матроїда 94
invariant інваріант 204
isomorphism ізоморфізм 203

K

Kruskal's algorithm алгоритм Краскала 105

L

line-covering реберне покриття 57
loop петля 12

M

matching паросполучення 45
matroid матроїд 94, 123
maximum flow problem задача про максимальний потік 190
minimum cut-set мінімальний розріз 194
minimum regular edge coloring мінімальна правильна розфарбовка ребер 86
minimum regular vertex coloring мінімальна правильна розфарбовка вершин 81
multigraph мультиграф 12

N

network мережа 185
node вузол 10
null graph нуль-граф 15

O

order of a graph розмір графа 10
Ore theorem теорема Оре 118
oriented cycle орієнтований цикл 145

P

packing (edge) реберне пакування 45
packing (vertex) вершинне пакування 53
partial order частково упорядковане 143
partition matroid матроїд розбиттів 94
path шлях, маршрут 98
perfect matching довершене паросполучення 46
peripheral vertex периферійна вершина 181
PERT 197
preorder передупорядковане 143
Prim's algorithm алгоритм Пріма 102
pseudograph псевдограф 12

Q

quasiorder квазіупорядковане 143

R

radius of a graph радіус графа 181
Rado-Edmonds theorem теорема Радо-Едмондса 97
Randič index індекс Рандича 208
rank of matroid ранг матроїда 96
reflexive рефлексивне 143
regular edge coloring правильна розфарбовка ребер 86
regular vertex coloring правильна розфарбовка вершин 81
relaxation релаксація 167

S

semi-Eulerian multigraph напівейлерів мультиграф 111
semi-Hamiltonain graph напівгамільтонів граф 118
shortest path найкоротший шлях 166
simple cycle простий цикл 99
simple path простий шлях 98
sink кінцева вершина 185
size of a graph потужність графа 11
source початкова вершина 185
spanning tree остовне дерево 99
stable set (of vertex) внутрішньо стійка множина вершин 53
strongly connected component компонента сильної зв'язності 145
strongly connected сильно зв'язані 144
symmetric симетричне 143

T

tail кінцева вершина 185
task робота 197
total order повністю упорядковане 143
total повне 143
transitive транзитивне 143
transversal set (vertex) трансверсальна множина вершин 61
tree дерево 101
triangle relaxation релаксація трикутника 177

V

vertex вершина 10
Vizing theorem теорема Візінга 86

W

weight вага 14
weighted зважений 13
Wiener index індекс Вінера 205

Література

1. Алгоритм Фльорі для знахождения циклу Ейлера в графі. — http://www.cyberforum.ru/c-beginners/thread1139533.html.
2. Бартіш М. Я. Дослідження операцій. Частина 1. Лінійні моделі / М. Я. Бартіш, І. М. Дудзяний. — Львів : Видавничий центр ЛНУ імені Івана Франка, 2007. — 168 с.
3. Бартіш М. Я. Дослідження операцій. Частина 2. Алгоритми оптимізації на графах / М. Я. Бартіш, І. М. Дудзяний. — Львів : Видавничий центр ЛНУ імені Івана Франка, 2007. — 120 с.
4. Іглін С. П. Персональна сторінка. — http://iglin.epizy.com.
5. Berge C. The Theory of Graphs and It's Applications / C. Berge. — NY : Wiley, 1962. — 247 p.
6. Harary F. Graph Theory / F. Harary. — London : Addison-Wesley, 1969. — 280 p.
7. Horton J. D. A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph / J. D. Horton // SIAM Journal on Computing. — 1987. — Vol. 16(2). — P. 358–366.
8. Hunt B. R. A Guide to MATLAB for Beginners and Experienced Users / B. R. Hunt, R. L. Lipsman, J. M. Rosenberg. — London : Cambridge University Press, 2001. — 346 p.
9. Mathworks. — 2022. — https://www.mathworks.com/.
10. Mathworks File Exchange Central. — https://www.mathworks.com /matlabcentral/fileexchange/.
11. Miller C. E. Integer Programming Formulation of Traveling Salesman Problems / C. E. Miller, A. W. Tucker, R. A. Zemlin // Journal of the ACM. — 1960. — Vol. 7. — P. 326–329.
12. Ore O. Theoty of Graphs / O. Ore. — NY : AMS, 1967. — 270 p.
13. Swamy M. N. S. Graphs, Networks and Algorithms / M. N. S. Swamy, K. Thulasiraman. — NY : Wiley, 1992. — 480 p.
14. Tutte W. T. Graph Theory / W. T. Tutte. — London : Cambridge University Press, 1988. — 360 p.

Навчальне видання

ІГЛІН Сергій Петрович
ЗАЙЦЕВ Юрій Іванович
РЕШЕТНЯК Юрій Борисович

ТЕОРІЯ ГРАФІВ НА БАЗІ MATLAB

Навчальний посібник
для студентів інформаційних спеціальностей
усіх форм навчання
вищих навчальних закладів

Відповідальний за випуск *О. Б. Ахієзер*
В авторській редакції

План 2022 р., поз. 99

Підп. до друку 27.10.2022. Формат 60 × 90 1/16. Папір офсетний.
Гарнітура Cambrina. Ум. друк. арк. 14,75.
Наклад 100 прим. Зам. № 77 – 2022/2510. Ціна договірна.

Видавництво "НТМТ".
Свідоцтво про внесення до Державного реєстру видавництв ДК № 1748 від 15.04.2004 р.
61072, м. Харків, вул. Дерев'янка, б. 6, к. 83. Тел. (095)249-39-96