Розділ 1

Введення в теорію графів

1.1 Основні означення

Зображаються графи так, як показано на мал. 1.1. На мал. 1.1а показаний звичайний (неорієнтований) граф, а на мал. 1.1б — орієнтований (оргграф). 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* (graph) G — це сукупність двох множин V і E:

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

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

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

Будемо позначати вершини графа v_1, v_2, …, 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 позначають буквою n: $\big|V\big| = n$. Тут функція $\big|\ldots\big|$ — кількість елементів множини.

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

Як правило, ребра позначають буквами e_1, e_2, …, e_m.

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: $\big|E\big| = m$. За основними аксіомами теорії множин одинакові елементи множини E вважаються одним елементом, тому у відповідності з означенням 1.1 кратних ребер у графі не може бути. За тими ж аксіомами у кожного ребра повинні бути дві різні вершини, т. я. дві одинакові

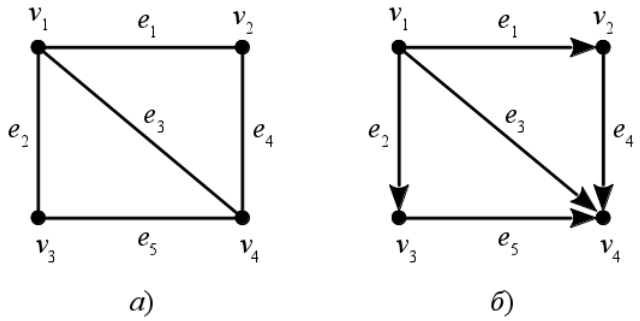


Рис. 1.1: Звичайний (а) та орієнтований (б) графи

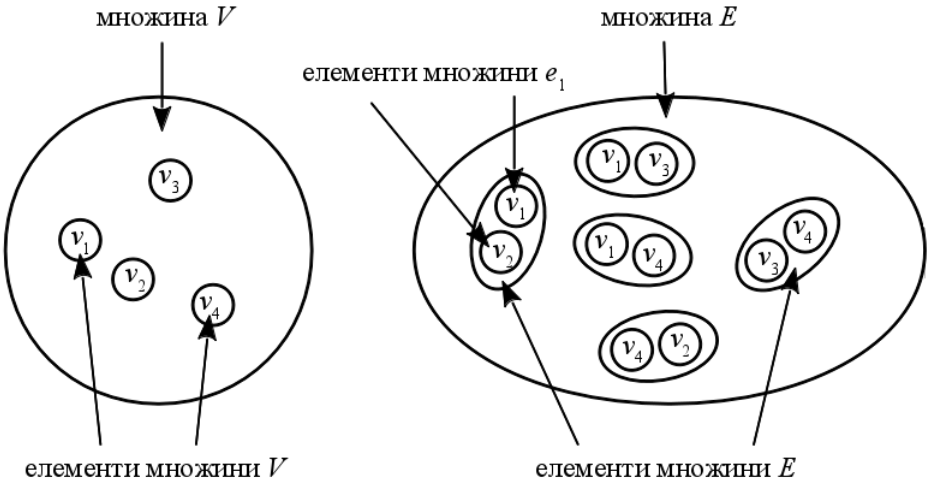


Рис. 1.2: Множини V (а) та E (б) графа G

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 мал. 1.1а — 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, т. я. елементи множини вважаються неупорядкованими. Так само не має значення і 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 мал. 1.1а множини V, E та їхні елементи показані на мал. 1.2. Але реальне життя значно ширше 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 мал. 1.3 зліва показаний мультиграф, а справа — його множина V та мультимножина E.

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 одинакові елементи множини E (кратні ребра), але й одинакові 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 мал. 1.4 показаний псевдограф, його множина V та мультимножина E.

Означення 1.8. *Гіперграфом* (hypergraph) G називається сукупність (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, але й будь-яку кількість вершин. Означення 1.1 припускає кратні гіперребра, у т. ч. й петлі. На мал. 1.5 наведений приклад гіперграфа. Тут для опису петель достатньо 1-елементних підмножин множини V. У гіперграфа на цьому малюнку є гіперребра, що з'єднують 3 вершини. Вони позначені лініями, з'єднаними точками. Але можуть бути й гіперребра, що поєднують 4, 5 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. ♢

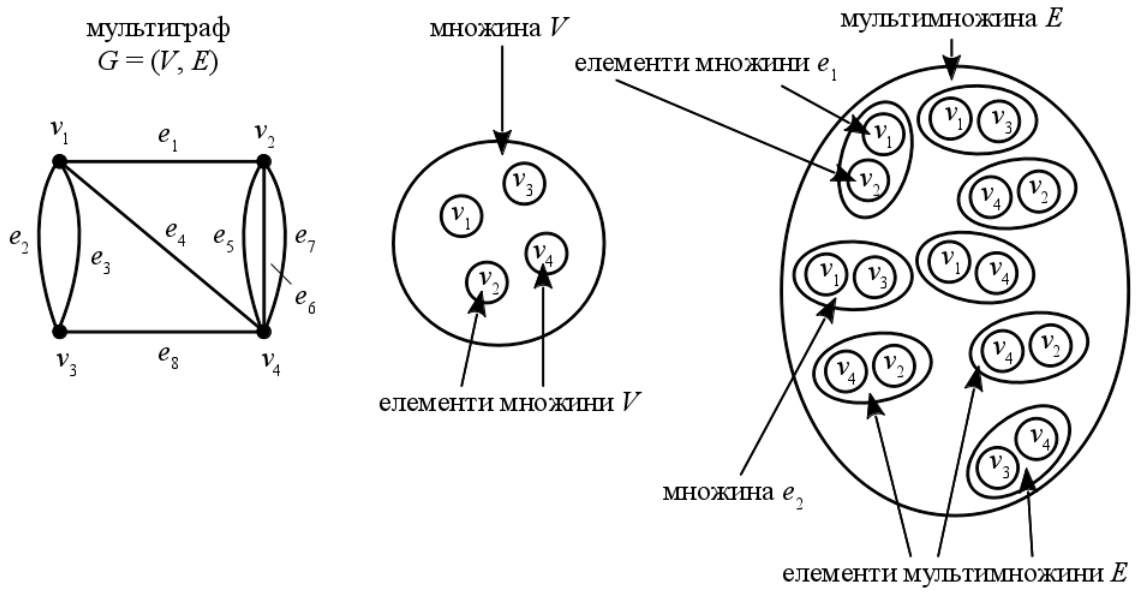


Рис. 1.3: Мультиграф G, його множина V та мультимножина E

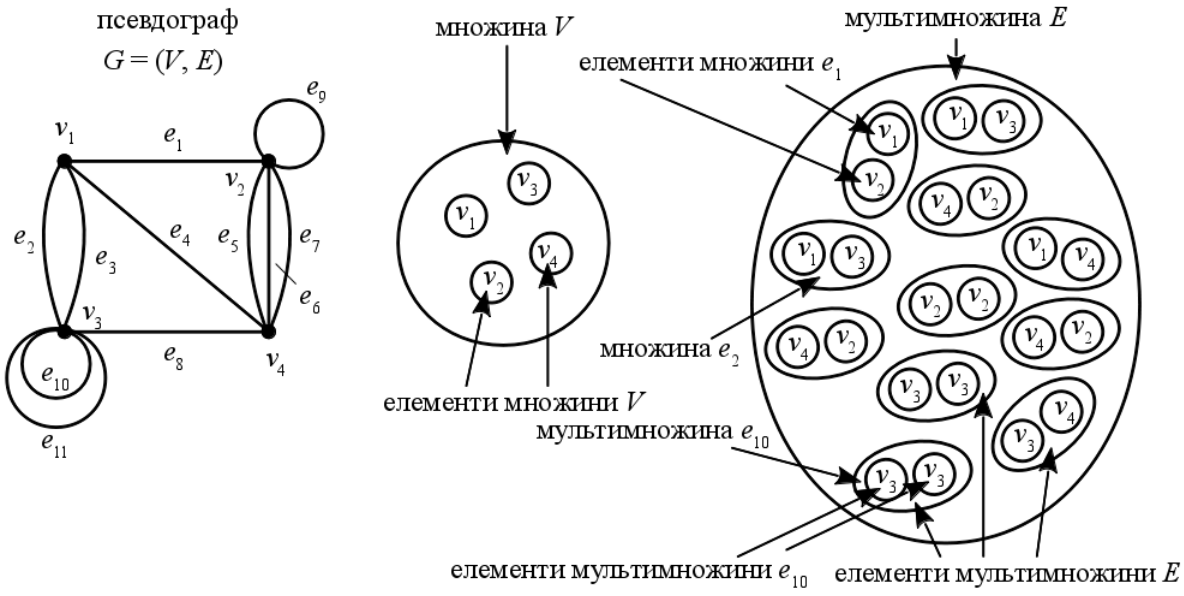


Рис. 1.4: Псевдограф G, його множина V та мультимножина E

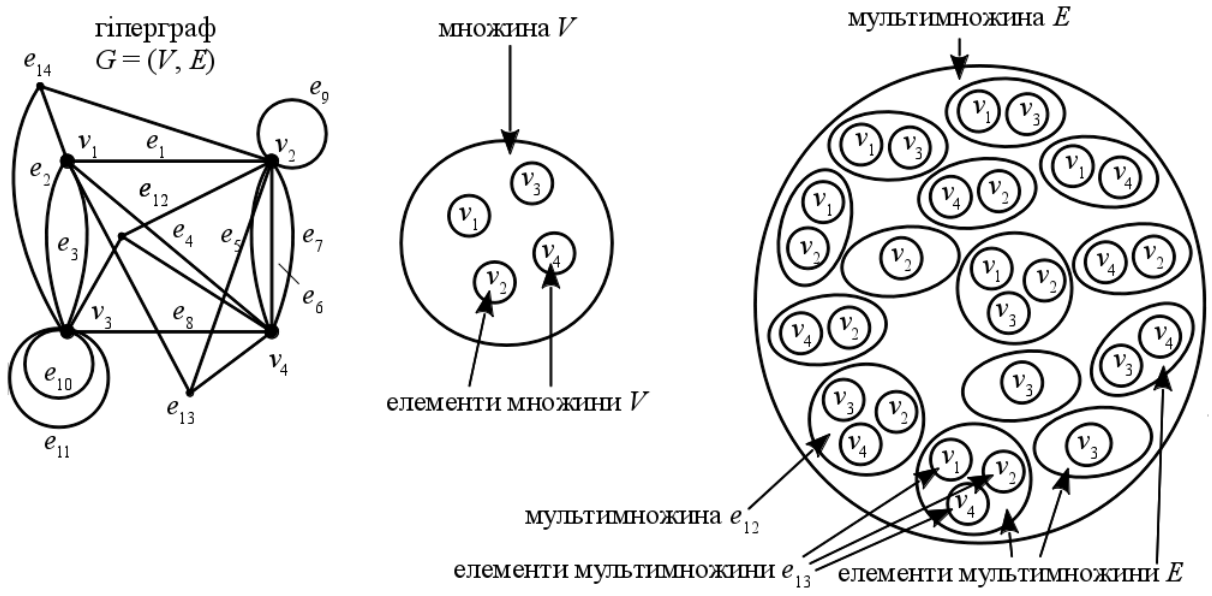


Рис. 1.5: Гіперграф G, його множина V та мультимножина E

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 мал. 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:

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

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, что ми провели *розфарбовку вершин або ребер* графа (vertex or edges coloring).

Розглянемо деякі види графів, що часто зустрічаються в застосуваннях.

Означення 1.14. 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:

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

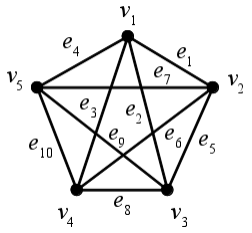
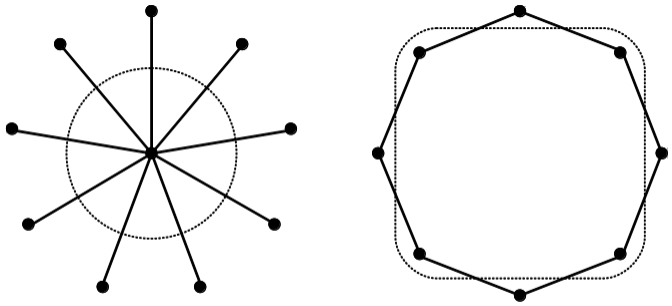


Рис. 1.6: Кліка K_5



а) *б*)

Рис. 1.7: Дводолеві графи

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 2 кінця, звідсіля й отримуємо формулу (1.4). На мал. 1.6 показана кліка K_5.

Означення 1.15. 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. $\diamondsuit$

Дводолеві графи зазвичай позначають як сукупність трьох множин:

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

Прикладами дводолевих графів є зірка з будь-якою кількістю променів та багатокутник із парною кількістю вершин (мал. 1.7). Розбиття множини вершин на дві частини показане тут тонкою штриховою лінією. Але зазвичай вершини дводолевих графів зображають на двох вертикалях або горизонталях, як на мал. 1.8б. Дводолеві графи часто використовуються у застосуваннях. Наприклад, задачі призначення, розподілу, плани перевезень описуються дводолевими графами (працівники та роботи, джерела ресурсів та споживачі, склади та магазини). Але й в теоретичних дослідженнях нам доводиться стикатися з ними. Зокрема, будь-якому гіперграфу можна співставити у взаємно однозначну відповідність (бієкцію) дводолевий граф. Для цього достатньо вершини початкового гіперграфа віднести до першої долі V, а кожному *гіперребру* поставити у відповідність *вершину* з другої долі W. Тепер з'єднуємо кожну вершину v_i у V з тими вершинами у W, які відповідають гіперребрам початкового гіперграфа, інцидентним до v_i.

Означення 1.16. Дводолевий граф $D = (V',\, W',\, E')$ називається *відповідним* (incidence) до гіперграфа $G = (V,\, E)$, якщо $\big|V'\big| = \big|V\big|$, $\big|W'\big| = \big|E\big|$, а ребро $e'_k = \big\{v'_i,\, w'_j\big\} \in E'$ тоді й тільки тоді, коли у гіперграфі G вершина v_i інцидента до ребра e_j. $\diamondsuit$

Приклад. Побудуємо дводолевий граф, відповідний до гіперграфа з мал. 1.4. На мал. 1.8а показаний початковий гіперграф, а на 1.8б — відповідний до нього дводолевий граф. У початкового гіперграфа $n = 4$, $m = 14$, тому в першій долі дводолевого графа буде 4 вершини, а у другій — 14.

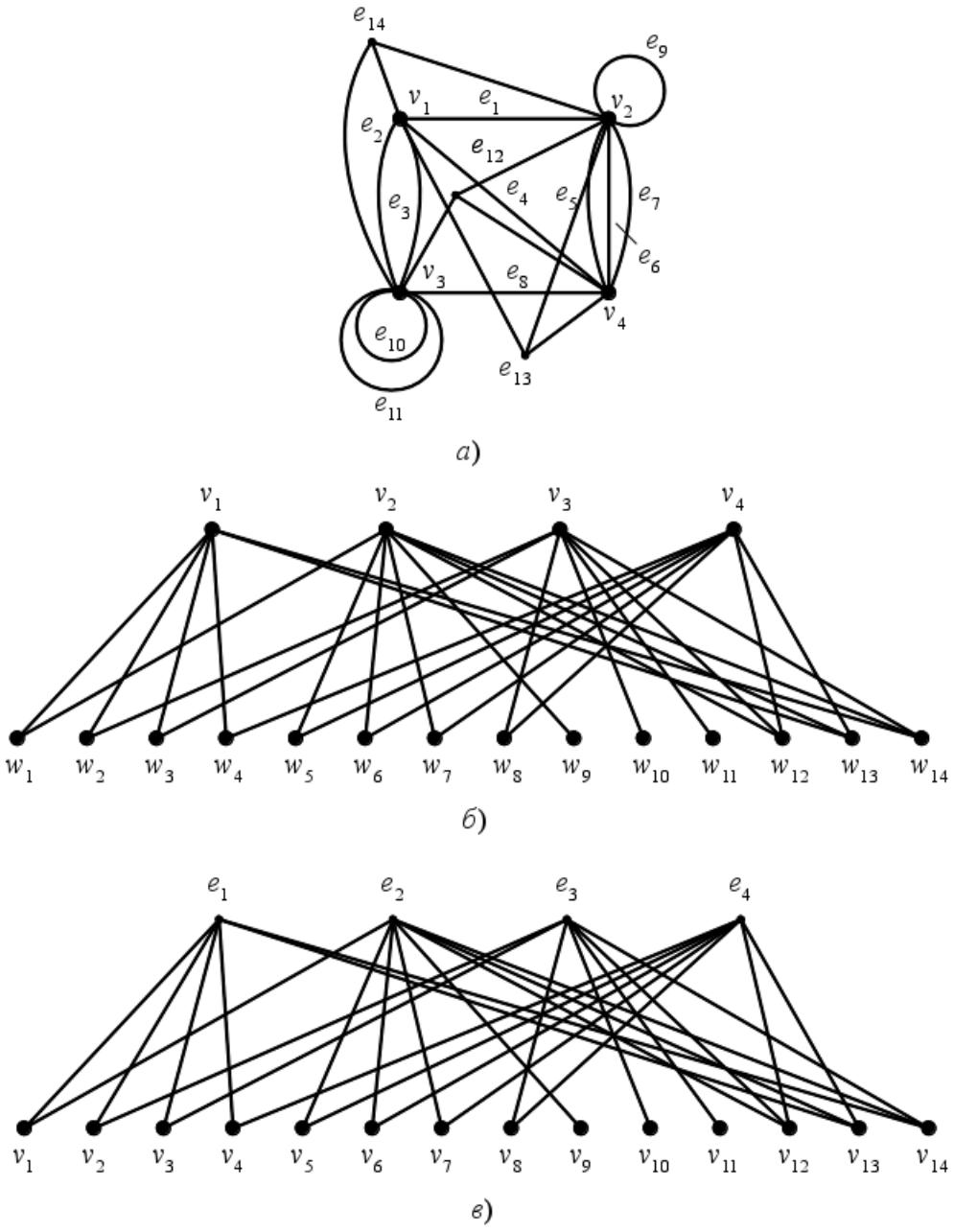


Рис. 1.8: Гіперграф (а), відповідний до нього дводолевий граф (б) та двоїстий гіперграф (в)

У початковому гіперграфі v_1 інцидентна до e_1, e_2, e_3, e_4, e_{13} та e_{14}, тому у 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 дводолевим графом однозначно відновлюється початковий гіперграф. Відновлення можна провести, наприклад, так. У відповідному дводолевому графі $\big|V\big| = 4$, тому малюємо 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, значить, 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 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 — значить, 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 дводолевому графі, а потім спробуємо відновити гіперграф? Ми отримаємо гіперграф, двоїстий до початкового.

Означення 1.17. Гіперграф $G' = \{V', E'\}$ називається *двоїстим* (dual) по відношенню до гіперграфа $G = \{V, E\}$, якщо $\big|V'\big| = \big|E\big|$, $\big|E'\big| = \big|V\big|$, і гіперребро e'_j є інцидентним до вершини v'_i тоді й тільки тоді, коли гіперребро e_i є інцидентним до вершини v_j. ♢

Продовжимо приклад. Побудуємо гіперграф, двоїстий до гіперграфа з мал. 1.5. Ми вже побудували дводолевий граф, відповідний до нього (мал. 1.8б). 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}. Аналогічно будуємо й інші гіперребра. Результат показаний на мал. 1.8в. Вочевидь, гіперграф, двоїстий до двоїстого, співпадає з початковим. Тобто ми можемо казати про пару взаємно двоїстих гіперграфів.

Означення 1.18. 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 мал. 1.1б. 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, як ми побачимо далі). Цього цілком достатньо для розв'язання задач теорії графів. Але для малювання 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 малювати граф у просторі (як якусь аксонометричну проекцію), то у третьому стовпчику задаємо аплікати. Якщо координати не задавати, можна малювати граф з лінійним розташуванням вершин, як на мал. 1.8в. 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 мал. 1.1а та орграфа з мал. 1.1б 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 одинакових числа. Але гіперграфи задавати таким масивом не вдається. 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 мал. 1.8в 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, т. я. ми нумеруємо вершини числами від 1 до n, а вершини с номером 0 немає.

За списком ребер (чи гіперребер) легко знаходяться m і n. Потужність графа m — це кількість рядків матриці, а розмір n — максимальне число в ній. Якщо у графі немає висячих вершин (вершин без ребер), то n можна окремо не задавати. Розмір графа n потрібно буде задати окремо тільки тоді, коли у графі є висячі вершини з номерами, більшими за максимальний елемент матриці списку ребер. Наприклад, якщо б у графа на мал. 1.1а була б ще й висяча вершина v_5, треба було б задати $n = 5$ у явному вигляді.

За необхідності також задаються ваги вершин та ребер (дуг). Це одновимірні масиви дійсних чисел довжиною n та m відповідно. Зауважимо, що наведений вище набір даних однозначно характеризує також і гіперграф з мал. 1.8а, який є двоїстим до гіперграфа з мал. 1.8в. Для кожної вершини (рядка) тут вказані номери інцидентних до неї ребер. І навпаки: матриця списку гіперребер для гіперграфа з мал. 1.8а 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
```

І цей набір даних також однозначно характеризує гіперграф з мал. 1.8в, двоїстий до гіперграфа з мал. 1.8а. Тут теж для кожної вершини (рядка) вказані номери інцидентних до неї ребер.

Тому інформацію про гіперграф (точніше, про пару взаємно двоїстих гіперграфів) можно задавати й у вигляді масиву з n рядків, у кожному з яких перелічені номери ребер, інцидентних до відповідної вершини, і коротші рядки дополнені нулями. Доречі, в різних мовах програмування є засоби, що дозволяють не дописувати нулі в кінці рядка (структури, списки тощо).

Але, якщо ми маємо не гіперграф, а граф (мультиграф, псевдограф, орграф), то зручніше все ж таки задавати його у вигляді списку ребер (дуг) розміром $m \times 2$, і доповнювати за необхідності одновимірними векторами ваг та розміром графа n.

1.3 Матричні представлення графів

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

Означення 1.19. *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 v_i інцидента до e_k. $\diamondsuit$

Для простого графа у кожному стовпчику матриці інцидентності буде рівно дві одиниці, а решта нулі. Так, для графа з мал. 1.1а 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 одинакові стовпці. Для псевдографів у стовпцях, що відповідають петлям, буде лише одна одиниця (або двійка, якщо це обумовлено конкретною задачею). У кожному стовпчику матриці інцидентності гіперграфа може бути скільки завгодно одиниць. Ось якою є, наприклад, матриця інцидентності гіперграфа з мал. 1.5а:

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

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

Як бачимо, за заданими множинами V та E ця матриця будується однозначно. Ось як виглядає псевдокод побудови матриці інцидентності графа (мультиграфа, псевдографа) за заданим масивом E розміром $m \times 2$ зі списком ребер:

```
for k=1 step 1 to m do
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 тут буде ще один 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 $m \times e_{\max}$.

3. Переглядаємо кожен стовпчик матриці інцидентності $\boldsymbol{A}$, і заповнюємо відповідний рядок масиву E номерами одиничних елементів.

Якщо побудувати матрицю інцидентності для двоїстого гіперграфа (мал. 1.8в), то неважко помітити, що вона буде транспонованою до матриці інцидентності початкового гіперграфа. Можна довести відповідну теорему — вона майже очевидна з означення 1.1.

Означення 1.20. *Матрицею інцидентності* (incidence matrix) орграфа G називається матриця $\boldsymbol{A}$ розміром $n \times m$, кожен елемент якої $a_{ik} = 1$ 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 $a_{ik} = 0$. $\diamondsuit$

Так виглядає матриця інцидентності для орграфа з мал. 1.1б:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{pmatrix}. \tag{1.9}$$

При перенумерації ребер (дуг) треба переставити стовпці матриці інцидентності, 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 ж правилами, транспонуємо її та множимо $\boldsymbol{T}'_V$ на $\boldsymbol{A}$.

Наступна матриця, яку ми розглянемо, це матриця суміжності вершин.

Означення 1.21. *Матрицею суміжності вершин* (adjacency matrix) простого графа G називається булівська (з елементами true та false) або бінарна (з елементами 1 та 0) 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. $\diamondsuit$

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 мал. 1.1а:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}. \tag{1.10}$$

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:

$$\boldsymbol{B} = \begin{pmatrix} \mathbf{0} & \boldsymbol{B}_{rs} \\ \boldsymbol{B}_{rs}^T & \mathbf{0} \end{pmatrix}, \tag{1.11}$$

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

Означення 1.22. *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. $\diamondsuit$

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

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & -1 & -1 & 0 \end{pmatrix}. \tag{1.12}$$

Ця матриця, як і матриця інцидентності, будується однозначно. Псевдокод побудови матриці суміжності вершин для простого графа за заданим масивом E розміром $m \times 2$ зі списком ребер виглядає так:

```
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; {або -1 для орграфів}
end {for k};
```

Зворотна задача теж розв'язується однозначно з точністю до нумерації ребер. Алгоритм дуже простий: переглядаємо всі елементи матриці $\boldsymbol{B}$ над головною діагоналлю. Якщо побачимо ненульовий елемент, то його індекси (номер рядка та стовпчика) — це буде черговий рядок масива E зі списком ребер.

Перенумерація ребер не змінює матрицю $\boldsymbol{B}$. При перенумерації вершин треба переставити рядки та стовпчики за алгоритмом, описаним вище для матриці інцидентності.

І ще одна матриця, пов'язана з графом — матриця суміжності ребер.

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. $\diamondsuit$

Ця матриця теж є симетричною. Для графа з мал. 1.1а вона має вигляд:

$$\boldsymbol{C} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \tag{1.13}$$

Псевдокод для однозначної побудови матриці суміжності ребер для простого графа за заданим масивом E розміром $m \times 2$ зі списком ребер є таким:

```
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 {then}
      C(k,l)=1;
      C(l,k)=1;
    end {then};
  end (for l);
end (for k);
```

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

Перенумерація вершин на матрицю суміжності ребер не впливає, а при перенумерації ребер її рядки та стовпчики переставляються згідно з описаним вище алгоритмом.

Розділ 2

Ізоморфізм графів

2.1 Постановка задачі

За означенням у графі, а також в усіх його узагальненнях координати вершин не задані. Як його малювати на площині — не має значення. Ми його малюємо, як зручніше. Наприклад, на мал. 2.1а та 2.1б зображений один і той самий граф.

Але якщо не тільки інакше намалювати 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 мал. 2.2а та 2.2б насправді є двома представленнями одного й того ж самого графа, якщо їхні вершини перенумеровані таким чином:

$$\begin{pmatrix} V \\ W \end{pmatrix} = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ w_1 & w_6 & w_8 & w_3 & w_5 & w_2 & w_4 & w_7 \end{pmatrix}. \tag{2.1}$$

Після такої перенумерації (підстановки) всі 12 ребер поєднують ті ж самі вершини:

$$\begin{pmatrix} E \\ F \end{pmatrix} = \begin{pmatrix} e_{15} & e_{16} & e_{17} & e_{25} & e_{26} & e_{28} & e_{35} & e_{37} & e_{38} & e_{46} & e_{47} & e_{48} \\ f_{15} & f_{12} & f_{14} & f_{65} & f_{62} & f_{67} & f_{85} & f_{84} & f_{87} & f_{32} & f_{34} & f_{37} \end{pmatrix}. \tag{2.2}$$

Саме такі графи називаються ізоморфними.

Означення 2.1. *Ізоморфізмом графів* (graph isomorphism) $G = (V, E)$ та $H = (W, F)$ називається бієкція (взаємно однозначна відповідність) між множинами вершин V та W:

$$f : V \leftrightarrow W \tag{2.3}$$

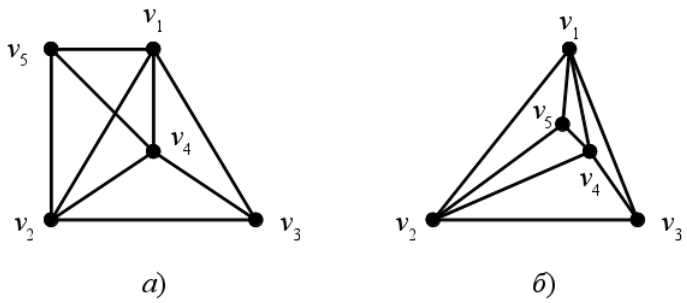


Рис. 2.1: Два різні зображення одного й того ж нрафа

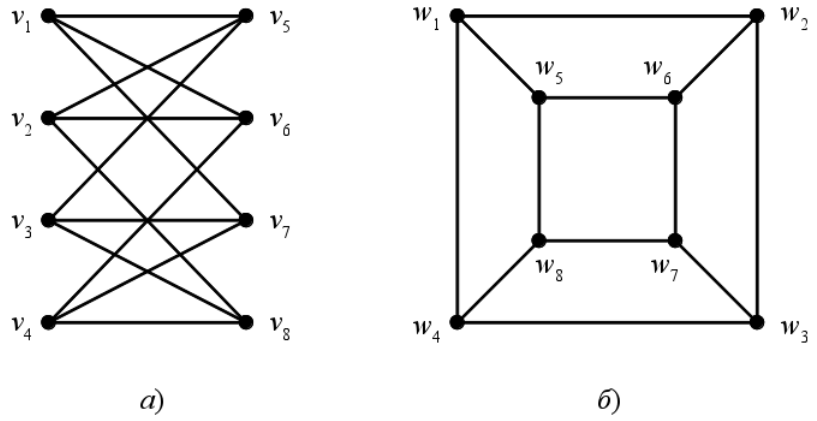


Рис. 2.2: Два різні представлення одного й того ж графа

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. $\diamondsuit$

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 (2.3). Якщо існує, треба її знайти: побудувати (2.1).

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 та 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 роспорядженні 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 $10^{-24}\,\mathrm{m}^3$, 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 швідкістю $299792458\,\mathrm{m/s}$. 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. Але 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 (2.3). Розглянемо їх.

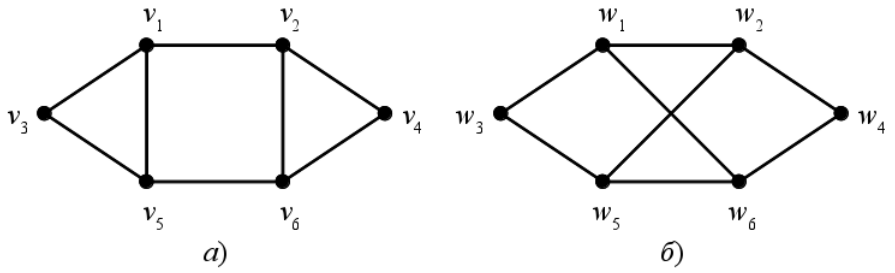


Рис. 2.3: Неізоморфні графи з одинаковими розмірами та потужностями

2.2 Інваріанти графів

Означення 2.2. *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 не ізоморфні 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 (2.3) неможливо.

Інваріант 2.1. $G_1 \simeq G_2 \Longrightarrow n_1 = n_2$. ♢

Інваріант 2.2. $G_1 \simeq G_2 \Longrightarrow m_1 = m_2$. ♢.

Так, у графів з мал. 2.2: $n_1 = n_2 = 8$; $m_1 = m_2 = 12$. Але, якщо $n_1 = n_2$ та (або) $m_1 = m_2$, це не означає, що графи ізоморфні. На мал. 2.3 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 відстань (найменшу кількість ребер, які треба пройти) від кожної вершини до всіх інших, а потім візьмемо максимальну з цих величин. Вона називається *діаметром* графа: $\operatorname{diam}(G)$. Як це зробити, ми розглянемо далі, у главі 8. Зрозуміло, що, як би ми не нумерували вершини, найкоротша відстань між найвіддаленішими вершинами не зміниться.

Інваріант 2.3. $G_1 \simeq G_2 \Longrightarrow \operatorname{diam}(G_1) = \operatorname{diam}(G_2)$. ♢

У графа G з мал. 2.2а найкоротша відстань між найвіддаленішими вершинами, наприклад, між v_1 та v_8, це 3. У графа H з мал. 2.2б дістатися, наприклад, з w_1 до w_7 теж можна лише за 3 кроки; тут $\operatorname{diam}(G) = \operatorname{diam}(H) = 3$. З іншого боку, і у графів на мал. 2.3 теж одинакові діаметри: $\operatorname{diam}(G) = \operatorname{diam}(H) = 3$. Але ці графи неізоморфні.

Можна також обчислити відстань від кожної вершини v_i до кожної іншої v_j: $d(v_i,\, v_j)$, а потім знайти їхню суму, яка називається *індексом Вінера* (Wiener index):

$$W(G) = \sum_{\forall i,\, j} d(v_i,\, v_j)\,. \tag{2.4}$$

Якщо графи ізоморфні, то між множинами відстаней (як і між множинами вершин) існує бієкція. Як наслідок, сума відстаней не змінюється.

Інваріант 2.4. $G_1 \simeq G_2 \Longrightarrow W(G_1) = W(G_2)$. ♢

Матриці відстаней між будь-якою парою вершин графів з мал. 2.2 не співпадають:

$$\boldsymbol{D}_G = \begin{pmatrix} 0 & 2 & 2 & 2 & 1 & 1 & 1 & 3 \\ 2 & 0 & 2 & 2 & 1 & 1 & 3 & 1 \\ 2 & 2 & 0 & 2 & 1 & 3 & 1 & 1 \\ 2 & 2 & 2 & 0 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 3 & 0 & 2 & 2 & 2 \\ 1 & 1 & 3 & 1 & 2 & 0 & 2 & 2 \\ 1 & 3 & 1 & 1 & 2 & 2 & 0 & 2 \\ 3 & 1 & 1 & 1 & 2 & 2 & 2 & 0 \end{pmatrix}; \ \boldsymbol{D}_H = \begin{pmatrix} 0 & 1 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 2 & 2 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 3 & 2 & 1 & 2 \\ 1 & 2 & 1 & 0 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 0 & 1 & 2 & 1 \\ 2 & 1 & 2 & 3 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 2 & 2 & 1 & 0 & 1 \\ 2 & 3 & 2 & 1 & 1 & 2 & 1 & 0 \end{pmatrix}, \tag{2.5}$$

але суми елементів над головною діагоналлю для них одинакові: $W(G) = W(H) = 48$. На жаль, те ж саме маємо і для неізоморфних графів з мал. 2.3. Їхні матриці відстаней такі:

$$\boldsymbol{D}_G = \begin{pmatrix} 0 & 1 & 1 & 2 & 1 & 2 \\ 1 & 0 & 2 & 1 & 2 & 1 \\ 1 & 2 & 0 & 3 & 1 & 2 \\ 2 & 1 & 3 & 0 & 2 & 1 \\ 1 & 2 & 1 & 2 & 0 & 1 \\ 2 & 1 & 2 & 1 & 1 & 0 \end{pmatrix}; \ \boldsymbol{D}_H = \begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 1 \\ 1 & 0 & 2 & 1 & 1 & 2 \\ 1 & 2 & 0 & 3 & 1 & 2 \\ 2 & 1 & 3 & 0 & 2 & 1 \\ 2 & 1 & 1 & 2 & 0 & 1 \\ 1 & 2 & 2 & 1 & 1 & 0 \end{pmatrix}, \tag{2.6}$$

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 рядки матриці 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 (2.3) встановити між вершинами G_1 та G_2 не вдасться: для чотирьох вершин ступеня 5 у G_2 не знайдеться відповідних у G_1. Значить, необхідною умовою ізоморфізму є співпадіння кількості нулів, кількості одиниць, двійок, ... 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.

Інваріант 2.5. $G_1 \simeq G_2 \Longrightarrow \boldsymbol{p}(G_1) = \boldsymbol{p}(G_2)$. $\diamondsuit$

Матриці суміжності вершин для графів з мал. 2.2 мають вигляд:

$$\boldsymbol{B}_G = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}; \ \boldsymbol{B}_H = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}, \tag{2.7}$$

а упорядковані вектори ступенів вершин складаються лише з трійок:

$$\boldsymbol{p}(G) = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}; \ \boldsymbol{p}(H) = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}. \tag{2.8}$$

Для графів з мал. 2.3 їхні матриці суміжності вершин такі:

$$\boldsymbol{B}_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}; \ \boldsymbol{B}_H = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \tag{2.9}$$

а упорядковані вектори ступенів вершин мають вигляд:

$$\boldsymbol{p}(G) = \begin{pmatrix} 2 & 2 & 3 & 3 & 3 & 3 \end{pmatrix}; \ \boldsymbol{p}(H) = \begin{pmatrix} 2 & 2 & 3 & 3 & 3 & 3 \end{pmatrix}, \tag{2.10}$$

тобто теж одинакові, хоча ці графі не є ізоморфними.

Вже на цьому етапі, якщо виявиться, що $\boldsymbol{p}(G_1) = \boldsymbol{p}(G_2)$, можна спробувати відшукати перестановку ізоморфізму (2.1), перебираючи не всі $n!$ перестановок, а лише $n_1! \times n_2! \times n_3! \times \ldots \times n_k!$, де k — максимальний ступінь вершини. Тобто можна окремо переставляти між собою вершини ступеня 1, ступеня 2 і т. д. 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 ступенів вершин $\boldsymbol{p}(G) = \begin{pmatrix} 3 & 3 & 4 & 4 & 4 & 5 & 5 & 7 & 7 & 8 \end{pmatrix}$, 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.

Зі ступенями вершин пов'язаний ще один інваріант: *індекс Рандича* (Randić index), який використовується в математичній хімії та хемоінформатиці. Знаходимо ступені кожної вершини $\deg(v_i)$; потім для кожного ребра $e_{ij} = \{v_i, v_j\}$ обчислюємо $(\deg(v_i)\deg(v_j))^{-1/2}$ та додаємо їх:

$$r(G) = \sum_{\forall e_{ij} \in E} \frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}. \tag{2.11}$$

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 ж самими ступенями вершин на його кінцях.

Інваріант 2.6. $G_1 \simeq G_2 \Longrightarrow r(G_1) = r(G_2)$. $\diamondsuit$

Неважко усно порахувати, що для обох графів з мал. 2.2 індекс Рандича дорівнює 4. Але і для обох неізоморфних графів з мал. 2.3 цей індекс теж одинаковий: $r(G) = r(H) = \frac{2\sqrt{6}+4}{3} \approx 2.966$.

У главі 4 ми будемо розглядати правильні розфарбовки. Зараз тільки відмітимо, що кількість кольорів у мінімальній правильній розфарбовці вершин (хроматичне число) $\chi(G)$ теж буде інваріантом.

Інваріант 2.7. $G_1 \simeq G_2 \Longrightarrow \chi(G_1) = \chi(G_2)$. $\diamondsuit$

Наприклад, граф на мал. 2.2а правильно фарбується двома фарбами: ліва доля однією, а права — іншою. У графі з мал. 2.2б 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 мал. 2.3! Для правильної розфарбовки графа з мал. 2.3а потрібно три фарби, бо в ньому є трикутники K_3. А вершини графа з мал. 2.3б 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 (2.7) або (2.9). При бієкції (2.3) вершини перенумеровуються, що відповідає перестановці рядків та стовпців матриці суміжності вершин. З алгебри матриць відомо, що для перестановки стопців якоїсь матриці треба її помножити 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 (2.1) має вигляд:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.12}$$

Множення $\boldsymbol{B}_G$ на $\boldsymbol{T}$ переставляє стовпці у $\boldsymbol{B}_G$ згідно з (2.1):

$$\boldsymbol{B}_G\boldsymbol{T} = \begin{pmatrix} 0&0&0&0&1&1&1&0\\ 0&0&0&0&1&1&0&1\\ 0&0&0&0&1&0&1&1\\ 0&0&0&0&0&1&1&1\\ 1&1&1&0&0&0&0&0\\ 1&1&0&1&0&0&0&0\\ 1&0&1&1&0&0&0&0\\ 0&1&1&1&0&0&0&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0 \end{pmatrix} = \begin{pmatrix} 0&1&0&1&1&0&0&0\\ 0&1&0&0&1&0&1&0\\ 0&0&0&1&1&0&1&0\\ 0&1&0&1&0&0&1&0\\ 1&0&0&0&0&1&0&1\\ 1&0&1&0&0&1&0&0\\ 1&0&1&0&0&0&0&1\\ 0&0&1&0&0&1&0&1 \end{pmatrix}. \tag{2.13}$$

Як бачимо, 1-й стовпчик залишився 1-м, 2-й став 6-м, 3-й — 8-м і т. д. Для перестановки рядків тепер треба помножити матрицю $\boldsymbol{T}^T$ на результат (2.13). Рядки у $\boldsymbol{B}_G$ будуть переставлені у такому ж порядку, як і раніше стовпці:

$$\boldsymbol{T}^T\boldsymbol{B}_G\boldsymbol{T} = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&0 \end{pmatrix} \begin{pmatrix} 0&1&0&1&1&0&0&0\\ 0&1&0&0&1&0&1&0\\ 0&0&0&1&1&0&1&0\\ 0&1&0&1&0&0&1&0\\ 1&0&0&0&0&1&0&1\\ 1&0&1&0&0&1&0&0\\ 1&0&1&0&0&0&0&1\\ 0&0&1&0&0&1&0&1 \end{pmatrix} = \begin{pmatrix} 0&1&0&1&1&0&0&0\\ 1&0&1&0&0&1&0&0\\ 0&1&0&1&0&0&1&0\\ 1&0&1&0&0&0&0&1\\ 1&0&0&0&0&1&0&1\\ 0&1&0&0&1&0&1&0\\ 0&0&1&0&0&1&0&1\\ 0&0&0&1&1&0&1&0 \end{pmatrix}. \tag{2.14}$$

Після такої перестановки стовпців і рядків ми отримали $\boldsymbol{B}_H$:

$$\boldsymbol{B}_H = \boldsymbol{T}^T\boldsymbol{B}_G\boldsymbol{T}. \tag{2.15}$$

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:

$$\boldsymbol{B}_H = \boldsymbol{T}^{-1}\boldsymbol{B}_G\boldsymbol{T}. \tag{2.16}$$

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 (2.16) 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 (2.3). З теорії лінійних операторів відомо, що при зміні базису не змінюється характеристичний поліном його матриці $D(\lambda)$ і, відповідно, його корені (тобто спектр оператора). Т. ч., характеристичний поліном матриці суміжності вершин (та його корені) є інваріантом.

Інваріант 2.8. $G_1 \simeq G_2 \Longrightarrow D_1(\lambda) = D_2(\lambda)$. $\diamondsuit$

На практиці легше перевіряти коефіцієнти характеристичного полінома (цілі числа).

Для обох графів з мал. 2.2: $D(\lambda) = |\boldsymbol{B}_G - \lambda\boldsymbol{E}| = |\boldsymbol{B}_H - \lambda\boldsymbol{E}| = \lambda^8 - 12\lambda^6 + 30\lambda^4 - 28\lambda^2 + 9$. А для неізоморфних графів з мал. 2.3 вони різні: $D_1(\lambda) = |\boldsymbol{B}_G - \lambda\boldsymbol{E}| = \lambda^6 - 8\lambda^4 - 4\lambda^3 + 12\lambda^2 + 8\lambda$; $D_2(\lambda) = |\boldsymbol{B}_H - \lambda\boldsymbol{E}| = \lambda^6 - 8\lambda^4 + 4\lambda^2$. З іншого боку, є неізоморфні графи, для яких $D(\lambda)$ співпадають. Наприклад, графи на мал. 2.4 явно неізоморфні: у них різні ступені вершин. Але характеристичні поліноми в них однакові: $D(\lambda) = |\boldsymbol{B}_G - \lambda\boldsymbol{E}| = |\boldsymbol{B}_H - \lambda\boldsymbol{E}| = -\lambda^5 + 4\lambda^3$.

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}, …, $b_{n-1,\,n}$, то отримаємо послідовність нулів та одниць, яку можна розглядати як двійкове число. Це число після переведення в десяткову систему координат називається кодом матриці $\boldsymbol{B}$ та позначається $\mu(G)$. При різній нумерації вершин, тобто при різних перестановках рядків та стовпців $\boldsymbol{B}$, її 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.

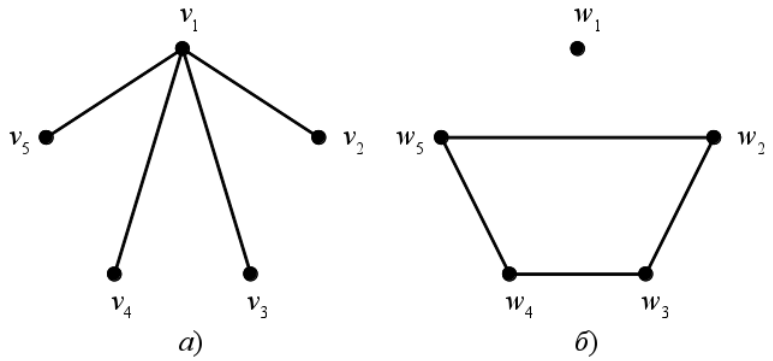


Рис. 2.4: Неізоморфні графи з одинаковими характеристичними поліномами

Інваріант 2.9. $G_1 \simeq G_2 \Longleftrightarrow (\mu_{\min}(G_1) = \mu_{\min}(G_2)) \wedge (\mu_{\max}(G_1) = \mu_{\max}(G_2)) \wedge (n_1 = n_2)$. $\diamondsuit$

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. Цікавих результатів у цьому напрямку досягла школа Хітрова з СПбДУ (Г.М.Хитров, СПбГУ). Його роботи та роботи його учнів можна відшукати на сайті ГрафоMann [8].

Розділ 3

Пакування, покриття, домінуючі множини

У цій главі будуть розглянуті 7 задач: вершинні та реберні пакування, покриття та домінуючі множини (це $2{\times}3 = 6$ задач) та задача про максимальний повний підграф (максимальну кліку). Усі ці задачі формулюються як задачі цілочисельного (навіть бінарного) лінійного програмування (ЦЛП).

3.1 Реберне пакування

Означення 3.1. *Реберним пакуванням* у графі $G = (V, E)$ називається підмножина попарно несуміжних ребер $E_1 \subseteq E$. 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 складається з максимально можливої кількості ребер. ♢

На мал. 3.1а 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 мал. 3.1б. Це паросполучення не є максимальним за включенням: до нього можна долучити ребро e_{13}, як показано на мал. 3.1в. 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 максимальним: замість одного ребра e_{13} до множини $\{e_1, e_4\}$ можна долучити два ребра: e_3 та e_6 (мал. 3.1г). Паросполучення $\{e_1, e_4, e_3, e_6\}$ вже буде максимальним: для 8 вершин більш ніж 4 несуміжних ребра вибрати неможливо.

Означення 3.2. 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 розділ 3.3). Його можна побудувати (якщо взагалі можна) лише для графа з парною кількістю вершин, і кількість ребер у ньому дорівнює $n/2$. Тому можна сказати й так: якщо у графі з парною кількістю вершин знайдене паросполучення з $n/2$ 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.

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.

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

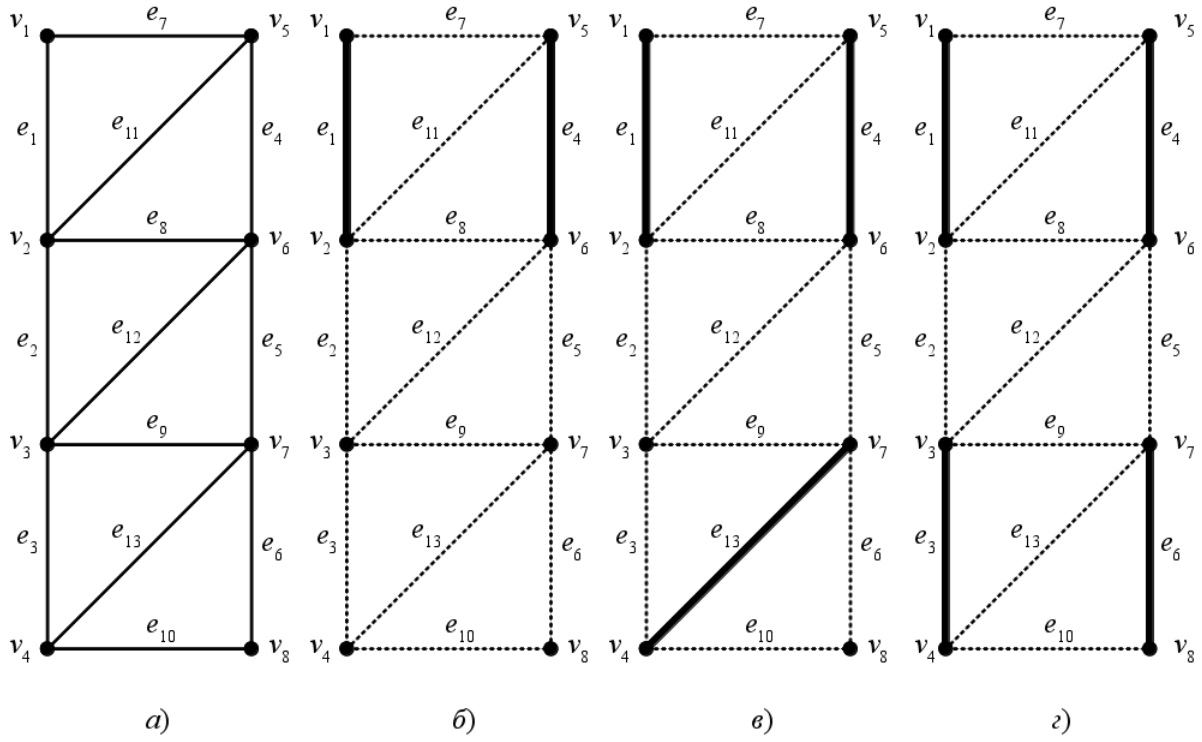
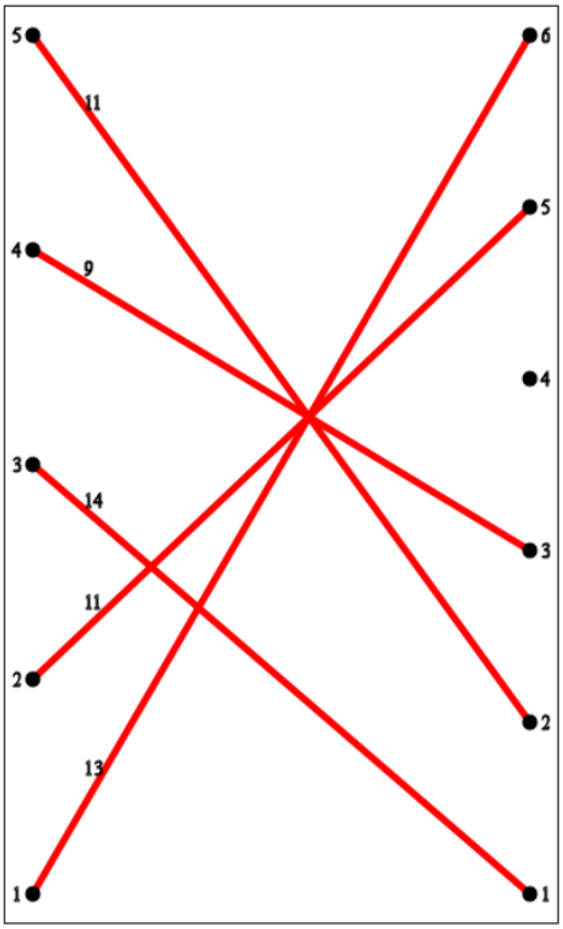


Рис. 3.1: Граф (а), його паросполучення (б), максимальне за включенням паросполучення (в) та максимальне паросполучення (г)

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

а)



б)

Рис. 3.2: Матриця продуктивностей (а) та розв'язок задачі про призначення (б)

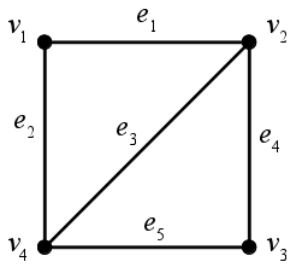


Рис. 3.3: Приклад графа

На мал. 3.2а показана матриця продуктивностей для задачі про призначення п'яти працівників на шість робіт, а на мал. 3.2б — її розв'язок: знайдене максимальне зважене паросполучення на повному дводолевому графі $K_{5,6}$.

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

Введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Назвемо цей вектор *асоційованим* з ребрами графа E, т. я. 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:

$$z = \sum_{k=1}^{m} e_k = (\mathbf{1},\, \boldsymbol{e})\,, \tag{3.1}$$

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:

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1,\, m}; \end{cases} \tag{3.2}$$

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.

Так, для графа з мал. 3.3 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, і т. д.:

$$\begin{array}{ll} v_1: & e_1 + e_2 \le 1; \\ v_2: & e_1 + e_3 + e_4 \le 1; \\ v_3: & e_4 + e_5 \le 1; \\ v_4: & e_2 + e_3 + e_5 \le 1. \end{array} \tag{3.3}$$

Якщо скористатися матрицею інцидентності, яка для графа з мал. 3.3 має вигляд:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \tag{3.4}$$

то умова відсутності суміжних ребер записується так:

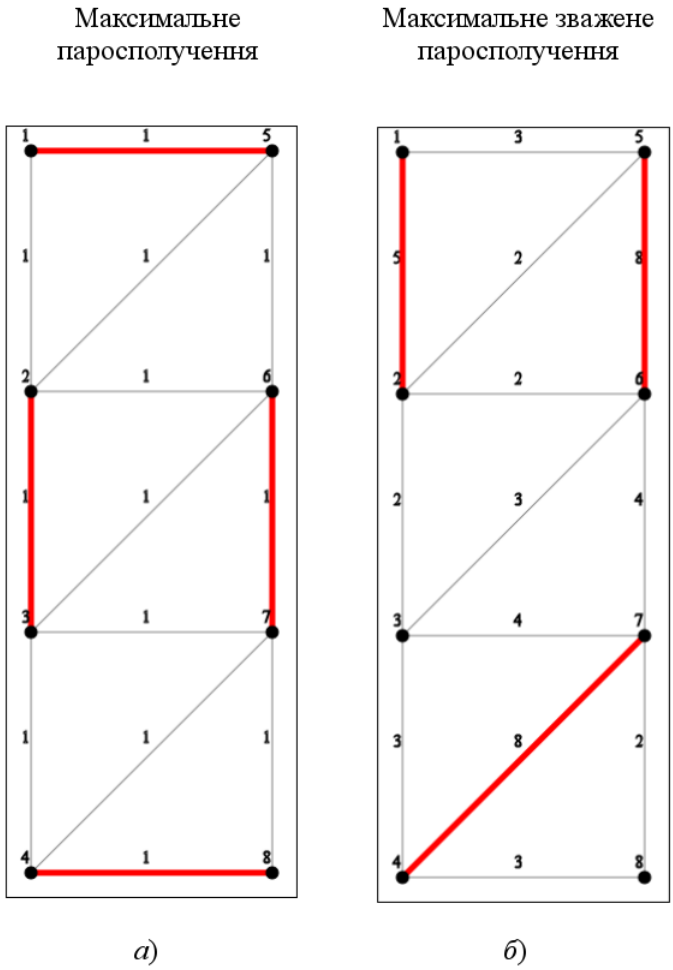


Рис. 3.4: Максимальне паросполучення (а) та максимальне зважене паросполучення (б)

$$\boldsymbol{Ae} \leq \mathbf{1}. \tag{3.5}$$

Тут векторна нерівність означає одночасне виконання нерівностей за всіма координатами. Т. ч., маємо задачу ЦЛП:

$$\begin{cases} z = (\mathbf{1},\, \boldsymbol{e}) \to \max; \\ \boldsymbol{Ae} \leq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.6}$$

В задачі про максимальне зважене паросполучення треба максимізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг ребер як $\boldsymbol{c}$. Тоді замість (3.6) будемо мати задачу ЦЛП, що відрізняється від (3.6) тільки цільовою функцією:

$$\begin{cases} z = (\boldsymbol{c},\, \boldsymbol{e}) \to \max; \\ \boldsymbol{Ae} \leq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.7}$$

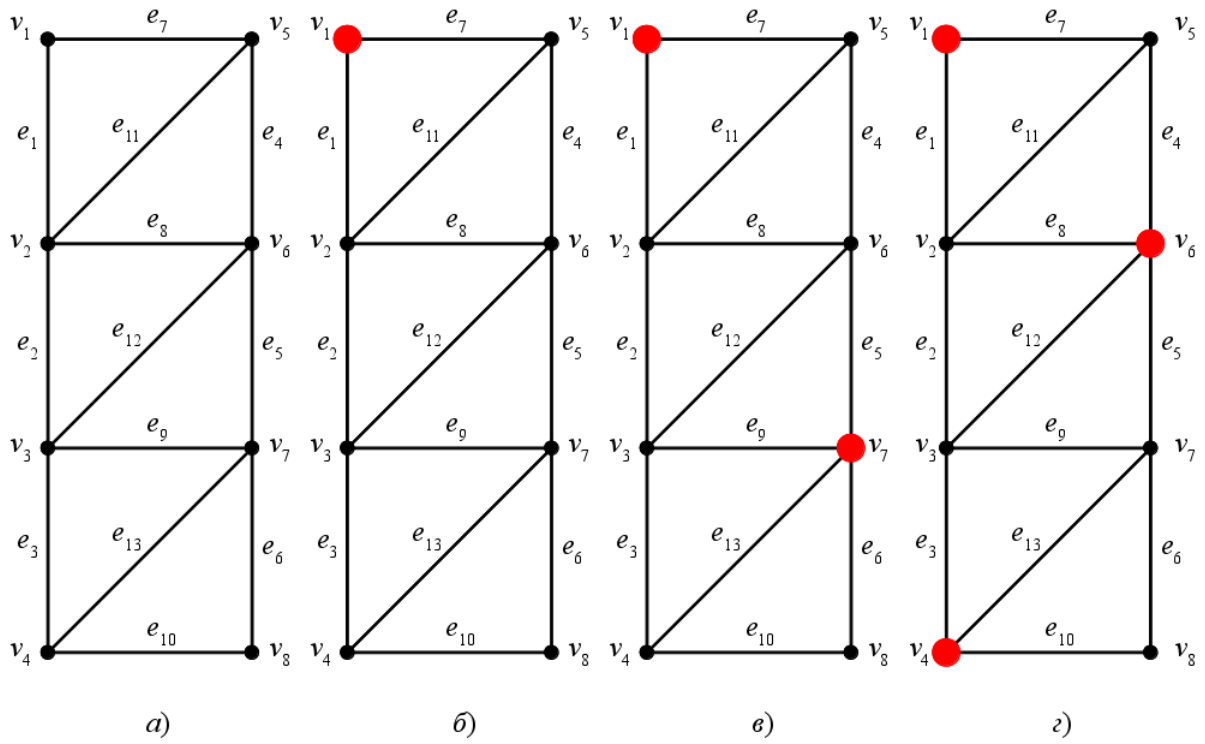


Рис. 3.5: Граф (а), його незалежна множина вершин (б), максимальна за включенням незалежна множина вершин (в) та максимальна незалежна множина вершин (г)

На мал. 3.4 — результати розв'язання задачі про максимальне паросполучення для графа з мал. 3.1а. Зліва, на мал. 3.4а — незважена задача: біля кожного ребра проставлена його одинична вага. Знайдене максимальне паросполучення з 4 ребер. Хоча воно й інше, ніж на мал. 3.1г, це все одно максимальне паросполучення, до того ж довершене: його ребра є інцидентними до всіх вершин. Справа, на мал. 3.4б — результат розв'язання зваженої задачі (вага кожного ребра проставлена біля нього). Як бачимо, у максимальному зваженому паросполученні може бути менше ребер, ніж у незваженому. В цьому прикладі загальна вага знайденого паросполучення з трьох ребер $8 + 5 + 8 = 21$ більша за загальну вагу будь-яких чотирьох несуміжних ребер.

3.2 Вершинне пакування

Означення 3.3. *Вершинним пакуванням* у графі $G = (V, E)$ називається підмножина попарно несуміжних вершин $V_1 \subseteq V$. 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 складається з максимально можливої кількості вершин. ♢

На мал. 3.5а представлений граф з $n = 8$ вершинами та $m = 13$ ребрами, той самий, що й на мал. 3.1а. Одна його вершина v_1 утворює незалежну множину вершин (мал. 3.5б). Ця незалежна множина не є максимальною за включенням: до неї можна долучити вершину v_7, як показано на мал. 3.5в. 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. Але вона не є максимальною: замість однієї вершини v_7 до множини $\{v_1\}$ можна долучити дві вершини: v_6 та v_4 (мал. 3.5г). 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. В 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 вибрати з цього колективу підгрупу незалежних одне від одного експертів із максимальним загальним рівнем компетентності.

Зведемо задачу про максимальну незалежну множину вершин до задачі ЦЛП. Введемо до розгляду вектор-стовпчик $\boldsymbol{v}$ довжиною n. Цей вектор буде асоційованим з вершинами: якщо якась вершина v_i входить до незалежної множини, то відповідна змінна v_i буде приймати значення 1, а якщо ні — то 0. Т. ч., координати вектора $\boldsymbol{v}$ є цілочисельними та можуть приймати лише одне з двох можливих значень — 0 або 1:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1,\, n}; \end{cases} \tag{3.8}$$

Треба максимізувати загальну кількість вершин:

$$z = \sum_{i=1}^{n} v_i = (\mathbf{1},\, \boldsymbol{v}) \tag{3.9}$$

за умови, що серед них немає суміжних. Ця вимога означає, що для кожного ребра існує не більше однієї вершини, 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 мал. 3.3 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, і т. д.:

$$\begin{array}{ll} e_1: & v_1 + v_2 \le 1; \\ e_2: & v_1 + v_4 \le 1; \\ e_3: & v_2 + v_4 \le 1; \\ e_4: & v_2 + v_3 \le 1; \\ e_5: & v_3 + v_4 \le 1. \end{array} \tag{3.10}$$

Якщо скористатися матрицею інцидентності (3.4), то умова відсутності суміжних вершин записується так:

$$\boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}. \tag{3.11}$$

Т. ч., маємо задачу ЦЛП:

$$\begin{cases} z = (\mathbf{1},\, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.12}$$

В задачі про максимальну зважену незалежну множину вершин треба максимізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг вершин $\boldsymbol{d}$. Тоді замість (3.12) будемо мати задачу ЦЛП, що відрізняється від (3.12) тільки цільовою функцією:

$$\begin{cases} z = (\boldsymbol{d},\, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T \boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.13}$$

На мал. 3.6 — результати розв'язання задачі про максимальну незалежну множину вершин для графа з мал. 3.5а. Зліва, на мал. 3.6а — незважена задача: біля кожної вершини проставлена її одинична вага. Знайдена максимальна незалежна множина з 3 вершин. Хоча вона й інша, ніж на мал. 3.5г, це все одно максимальна незалежна множина. Справа, на мал. 3.6б — результат розв'язання зваженої задачі (вага кожної вершини

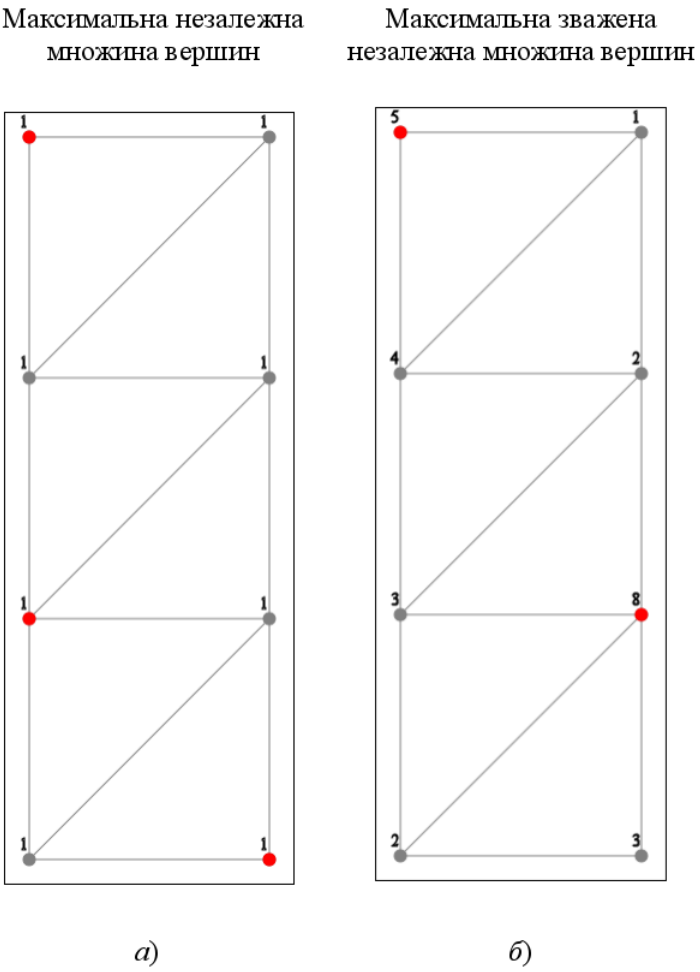


Рис. 3.6: Максимальна незалежна множина вершин (а) та максимальна зважена незалежна множина вершин (б)

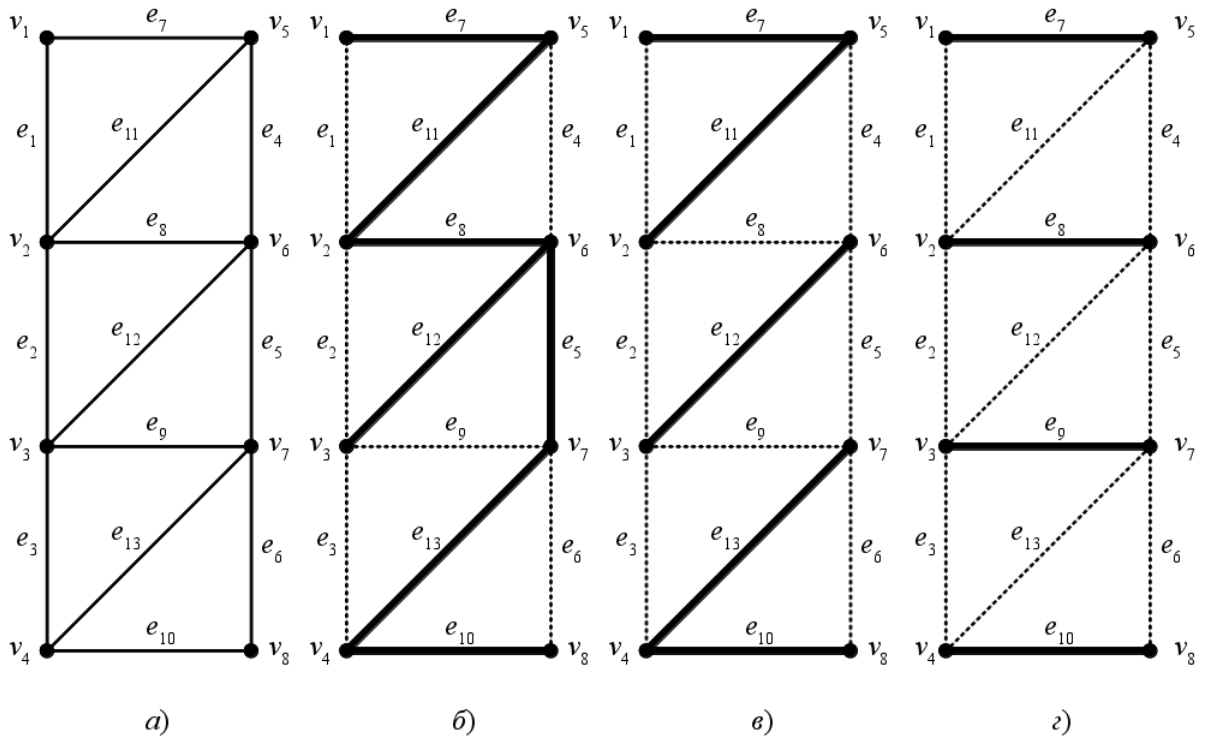


Рис. 3.7: Граф (а), його реберне покриття (б), мінімальне за виключенням реберне покриття (в) та мінімальне реберне покриття (г)

проставлена біля неї). Як і у попередній задачі, у максимальній зваженій незалежній множині виявилося менше вершин, ніж у незваженій. В цьому прикладі загальна вага знайденої незалежної множини з двох вершин $8 + 5 = 13$ більша за загальну вагу будь-яких трьох несуміжних вершин.

3.3 Реберне покриття

Означення 3.4. *Реберним покриттям* графа $G = (V, E)$ (line-covering, edge covering) називається підмножина ребер $E_1 \subseteq E$, що є інцидентними до всіх вершин графа. Реберне покриття називається *мінімальним за виключенням*, якщо будь-яка його підмножина з меншою кількістю ребер не є реберним покриттям. Реберне покриття називається *мінімальним*, якщо воно складається з мінімально можливої кількості ребер. ♢

На мал. 3.7а показаний граф з $n = 8$ вершинами та $m = 13$ ребрами (той самий приклад, що й раніше). Одне з його реберних покриттів — це $\{e_5, e_7, e_8, e_{10}, e_{113}, e_{12}, e_{13}\}$ з мал. 3.7б. Його ребра є інцидентними до всіх вершин. Це покриття не є мінімальним за виключенням: з нього можна вилучити, наприклад, ребра e_5 та e_8 (мал. 3.7в). Ті ребра, що залишилися: $\{e_7, e_{10}, e_{11}, e_{12}, e_{13}\}$, також є інцидентними до всіх вершин. Значить, 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. Значить, покриття $\{e_7, e_{10}, e_{11}, e_{12}, e_{13}\}$ є мінімальним за виключенням. 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 мал. 3.7г. Для цього прикладу знайдене мінімальне реберне покриття є водночас довершеним паросполученням (див. розділ 3.1). 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 ко-

мунікаціями (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), інші важче (більша вага). Тоді маємо зважену задачу: найменшою ціною знищити всю оборону супротивника.

Розглянемо формулювання задачі про мінімальне (зважене) реберне покриття як задачі ЦЛП. Як і в задачі про паросполучення (див. розділ 3.1), введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Його координати є асоційованими с 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:

$$t = \sum_{k=1}^{m} e_k = (\mathbf{1},\, \boldsymbol{e})\,. \tag{3.14}$$

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

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1,\, m}; \end{cases} \tag{3.15}$$

і ребра покривають усі вершини (тобто є інцидентними до всіх вершин). Ця вимога означає, що для кожної вершини існує не менше одного ребра з шуканого покриття, інцидентного 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 мал. 3.3 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 є інцидентними 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, і т. д. Як бачимо, нерівності-обмеження в цій задачі відрізняються від нерівностей-обмежень задачі про паросполучення (3.3) лише знаком:

$$\begin{array}{ll} v_1: & e_1 + e_2 \geq 1; \\ v_2: & e_1 + e_3 + e_4 \geq 1; \\ v_3: & e_4 + e_5 \geq 1; \\ v_4: & e_2 + e_3 + e_5 \geq 1. \end{array} \tag{3.16}$$

З використанням матриці інцидентності (3.4) система (3.16) записується у вигляді:

$$\boldsymbol{A}\boldsymbol{e} \geq \mathbf{1}. \tag{3.17}$$

Т. ч., маємо задачу ЦЛП:

$$\begin{cases} t = (\mathbf{1},\, \boldsymbol{e}) \to \min; \\ \boldsymbol{A}\boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.18}$$

В задачі про максимальне зважене реберне покриття треба мінімізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг ребер $\boldsymbol{c}$. Тоді замість (3.18) будемо мати задачу ЦЛП, що відрізняється від (3.18) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{d},\, \boldsymbol{e}) \to \min; \\ \boldsymbol{A}\boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.19}$$

На мал. 3.8 — результати розв'язання задачі про мінімальне реберне покриття для графа з мал. 3.7а. Зліва, на мал. 3.8а — незважена задача: біля кожного ребра проставлена його одинична вага. Знайдене мінімальне реберне покриття з 4 ребер. Хоча воно й інше, ніж на мал. 3.7г, це все одно мінімальне реберне покриття,

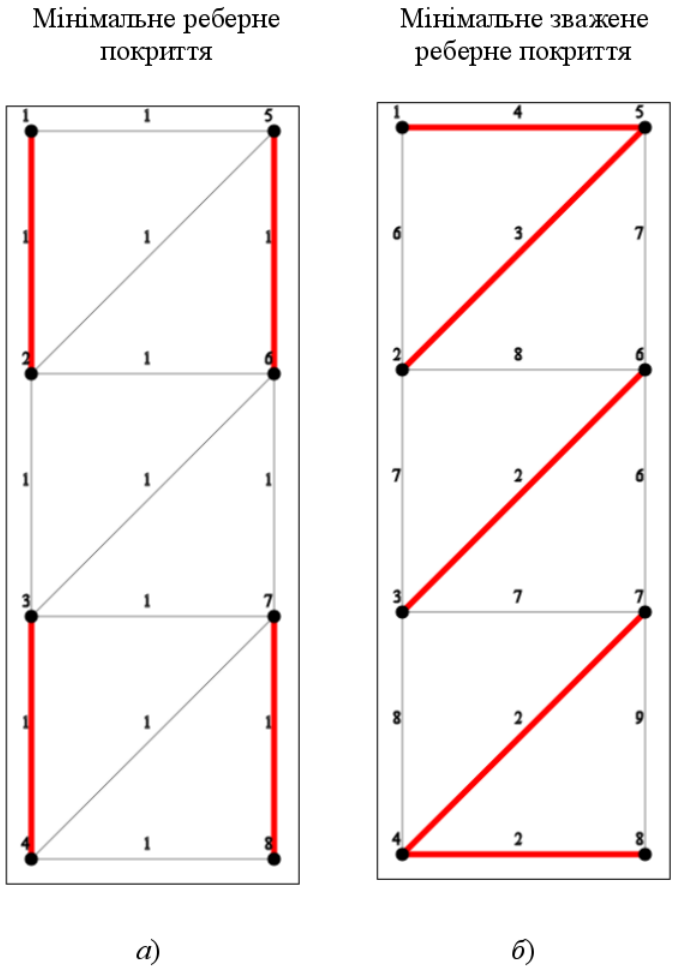


Рис. 3.8: Мінімальне реберне покриття (а) та мінімальне зважене реберне покриття (б)

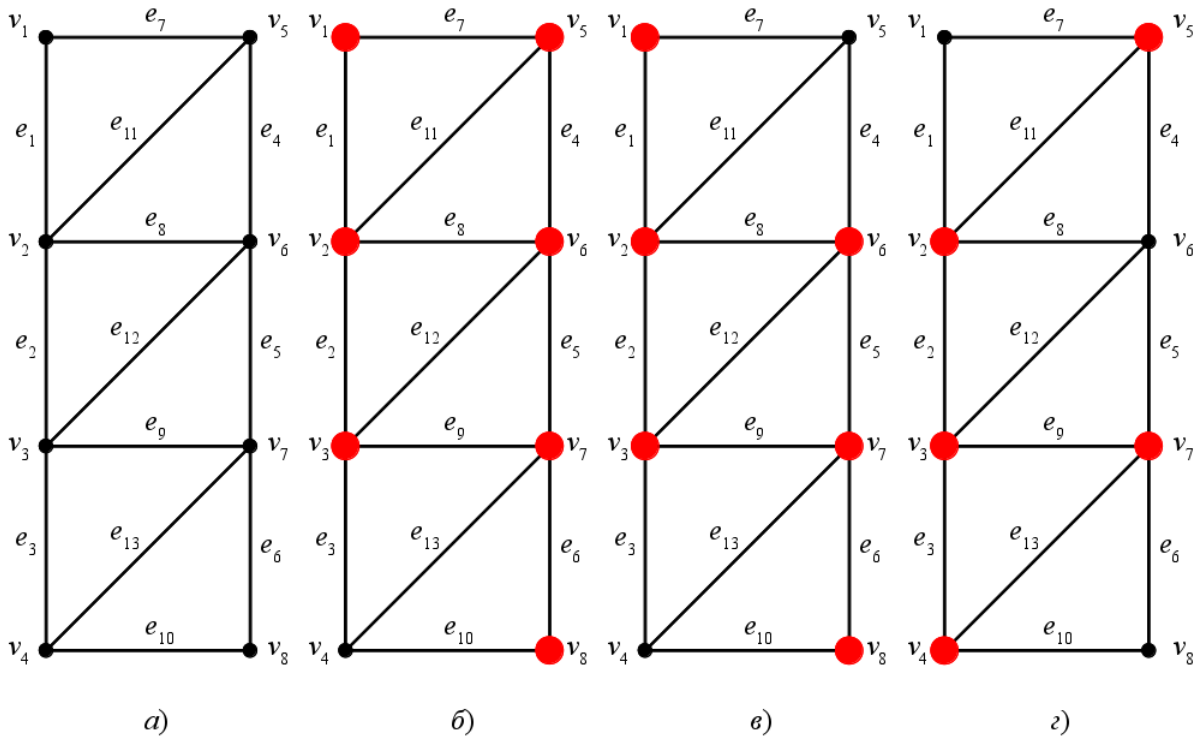


Рис. 3.9: Граф (а), його вершинне покриття (б), мінімальне за виключенням вершинне покриття (в) та мінімальне вершинне покриття (г)

яке є водночас довершеним паросполученням. Справа, на мал. 3.8б — результат розв'язання зваженої задачі (вага кожного ребра проставлена біля нього). Як бачимо, у мінімальному зваженому реберному покритті може бути більше ребер, ніж у незваженому. В цьому прикладі загальна вага знайденого реберного покриття з п'яти ребер $4 + 3 + 2 + 2 + 2 = 13$ менша за загальну вагу будь-яких чотирьох ребер, інцидентних до всіх вершин.

3.4 Вершинне покриття

Означення 3.5. *Вершинним покриттям* графа $G = (V, E)$ (vertex covering, transversal set) називається підмножина вершин $V_1 \subseteq V$, що є інцидентними до всіх ребер. Вершинне покриття називається *мінімальним за виключенням*, якщо будь-яка його підмножина з меншою кількістю вершин не є вершинним покриттям. Вершинне покриття називається *мінімальним*, якщо воно складається з мінімально можливої кількості вершин. ♢

Якщо з множини V вершин графа, показаного на мал. 3.9а, вилучити вершину v_4, то множина вершин, що залишилися $\{v_1, v_2, v_3, v_5, v_6, v_7, v_8\}$, утворює вершинне покриття (мал. 3.9б): вони покривають усі ребра. Це покриття не є мінімальним за виключенням: з нього можна вилучити v_5, і воно залишиться покриттям (мал.3.23.9в), бо 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 виключенням: якщо з нього вилучити ще якусь вершину, то якесь ребро не буде покриватися. Але це покриття не є мінімальним: можна створити покриття не з шести, а з п'яти вершин $\{v_2, v_3, v_4, v_5, v_7\}$, які є інцидентними до всіх ребер (мал.3.23.9г). 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, які є інцидентними до всіх ребер.

Приклад — військова задача. Є опорні пункти супротивника (вершини графа) та комунікації між ними (ребра). Знищення опорного пункту знищує і всі 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 ЦЛП. 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. Т. ч., координати вектора $\boldsymbol{v}$ є цілочисельними та можуть приймати лише одне з двох можливих значень — 0 або 1:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1,\, n}; \end{cases} \tag{3.20}$$

Треба мінімізувати загальну кількість вершин:

$$t = \sum_{i=1}^{n} v_i = (\mathbf{1},\, \boldsymbol{v}) \tag{3.21}$$

за умови, що кожна вершина покриває якесь ребро. Ця вимога означає, що для кожного ребра існує не менше однієї вершини, 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 мал. 3.3 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, і т. д. Нерівності-обмеження відрізняються від відповідних обмежень задачі про вершинне пакування (3.10) лише знаком:

$$\begin{array}{ll} e_1: & v_1 + v_2 \geq 1; \\ e_2: & v_1 + v_4 \geq 1; \\ e_3: & v_2 + v_4 \geq 1; \\ e_4: & v_2 + v_3 \geq 1; \\ e_5: & v_3 + v_4 \geq 1. \end{array} \tag{3.22}$$

Якщо скористатися матрицею інцидентності (3.4), то ці нерівності записуються так:

$$\boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}. \tag{3.23}$$

Т. ч., маємо задачу ЦЛП:

$$\begin{cases} t = (\mathbf{1},\, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.24}$$

В задачі про мінімальне зважене вершинне покриття треба мінімізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг вершин $\boldsymbol{d}$. Тоді замість (3.24) будемо мати задачу ЦЛП, що відрізняється від (3.24) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{d},\, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T \boldsymbol{v} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.25}$$

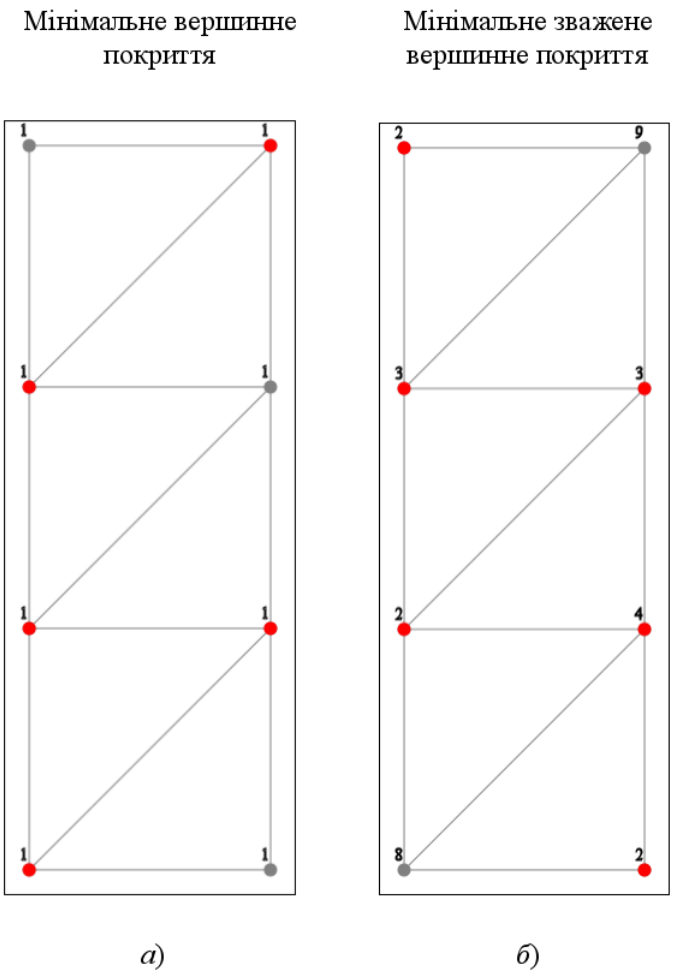


Рис. 3.10: Мінімальне вершинне покриття (а) та мінімальне зважене вершинне покриття (б)

На мал. 3.10 — результати розв'язання задачі про мінімальне вершинне покриття для графа з мал. 3.9а. Зліва, на мал. 3.10а — незважена задача: біля кожної вершини проставлена її одинична вага. Знайдене мінімальне вершинне покриття з 5 вершин, таке ж, як на мал. 3.9г. Справа, на мал. 3.10б — результат розв'язання зваженої задачі (вага кожної вершини проставлена біля неї). Як бачимо, у мінімальному зваженому вершинному покритті може бути більше вершин, ніж у незваженому. В цьому прикладі загальна вага знайденого вершинного покриття з шести вершин $2 + 3 + 3 + 2 + 4 + 2 = 16$ менша за загальну вагу будь-яких п'яти вершин, інцидентних до всіх ребер.

3.5 Двоїстість задач про пакування та покриття

Порівняємо між собою незважені задачі про пакування та покриття (3.6, 3.12, 3.18, 3.24). Зведемо їх до однієї таблиці 2×2 (3.26). Перший рядок — пакування, другий — покриття, перший стовпчик — реберні, другий — вершинні.

$$\begin{array}{cc} \text{Реберні} & \text{Вершинні} \\ \text{Пакування} \left\{\begin{array}{l} z = (\mathbf{1},\, \boldsymbol{e}) \to \max; \\ \boldsymbol{A}\boldsymbol{e} \le \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{array}\right. & \left\{\begin{array}{l} z = (\mathbf{1},\, \boldsymbol{v}) \to \max; \\ \boldsymbol{A}^T\boldsymbol{v} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{array}\right. \\ \nwarrow\ \nearrow \quad \swarrow\ \searrow & \\ \text{Покриття} \left\{\begin{array}{l} t = (\mathbf{1},\, \boldsymbol{e}) \to \min; \\ \boldsymbol{A}\boldsymbol{e} \ge \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{array}\right. & \left\{\begin{array}{l} t = (\mathbf{1},\, \boldsymbol{v}) \to \min; \\ \boldsymbol{A}^T\boldsymbol{v} \ge \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{array}\right. \end{array} \tag{3.26}$$

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 $z_{\max} = t_{\min}$. Опираючись на структуру матриці інцидентності $\boldsymbol{A}$ (вона складається з нулів та одиниць), ми можемо бути впевнені, що розв'язки усіх чотирьох задач за умови невід'ємності змінних напевно існують. Значить, рівності $z_{\max} = t_{\min}$ для обох пар задач виконуються. Позначимо це число:

$$z_{\max} = t_{\min} = \omega_{\text{opt}}. \tag{3.27}$$

У наших задачах (3.6, 3.12, 3.18, 3.24) 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 (3.26) маємо нерівеність:

$$z_{\max} \le t_{\min}. \tag{3.28}$$

Це означає: в одному й тому ж графі кількість ребер у максимальному паросполученні не може перевищувати кількість 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 (3.7, 3.13, 3.19, 3.25). Як би ми не зважували ребра, при побудові паросполучення вимога несуміжності ребер залишається. Тому кількість ребер у зваженому максимальному 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. Значить, і у зважених задачах ми можемо зробити той самий висновок: кількість ребер у максимальному зваженому

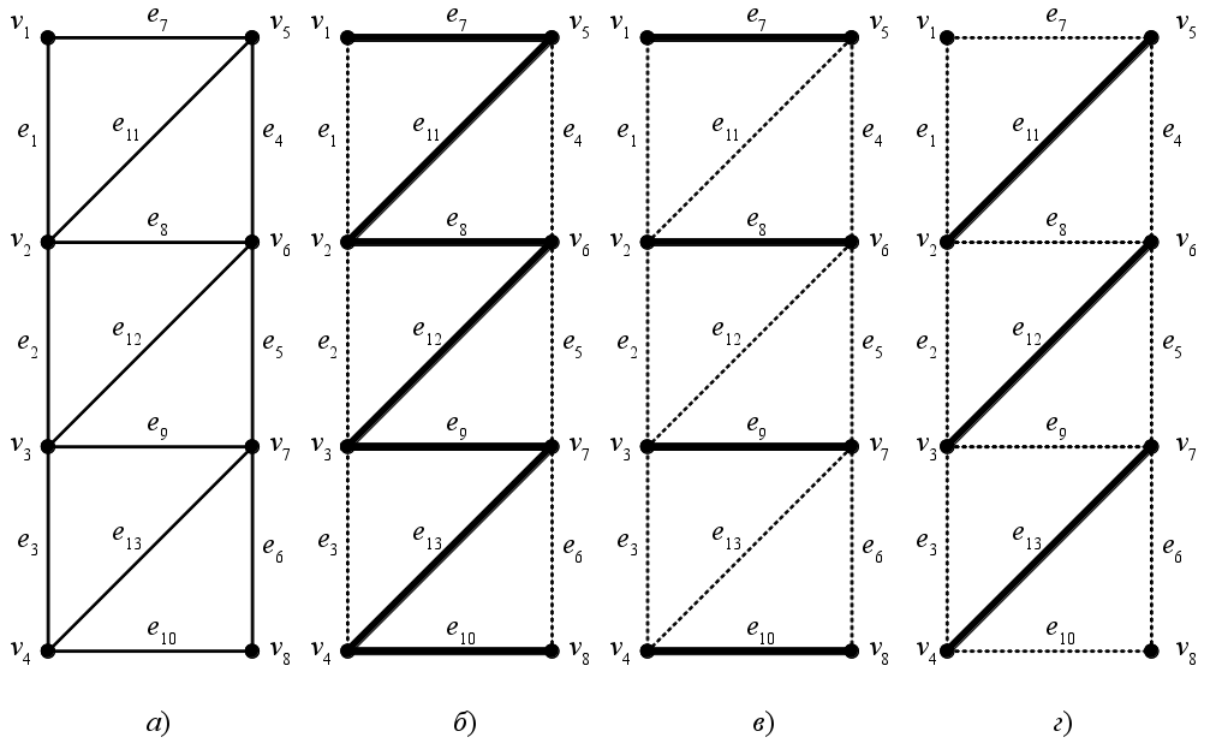


Рис. 3.11: Граф (а), його домінуюча множина ребер (б), мінімальна за виключенням домінуюча множина ребер (в) та мінімальна домінуюча множина ребер (г)

паросполученні ніколи не буде перевищувати кількості вершин у мінімальному зважному вершинному покритті. Але нерівність (3.28) у зважених задачах у загальному випадку вже не має місця, т. я. тепер $z_{\max}$ і $t_{\min}$ обчислюються через ваги ребер і вершин.

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

3.6 Домінуюча множина ребер

Означення 3.6. *Домінуючою множиною ребер* у графі $G = (V, E)$ називається така підмножина ребер $E_1 \subseteq E$, 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* (absorbant set, external stability set). Домінуюча множина ребер називається *мінімальною за виключенням*, якщо будь-яка її підмножина з меншою кількістю ребер не є домінуючою. Домінуюча множина ребер називається *мінімальною*, якщо вона складається з мінімально можливої кількості ребер. ♢

На мал. 3.11а показаний граф з $n = 8$ вершинами та $m = 13$ ребрами (той самий приклад, що й раніше). Одна з його домінуючих множин ребер — це $\{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}\}$ з мал. 3.11б. Її ребра є суміжними з усіма ребрами. Ця домінуюча множина не є мінімальною за виключенням: з неї можна вилучити, наприклад, ребра e_{11}, e_{12} та e_{13} (мал. 3.11в). Ті ребра, що залишилися: $\{e_7, e_8, e_9, e_{10}\}$, також є суміжними з усіма ребрами. Значить, це домінуюча множина ребер. Якщо вилучити з неї ще будь-яке ребро, якесь із ребер не охопиться суміжністю. Значить, домінуюча множина ребер $\{e_7, e_8, e_9, e_{10}\}$ є мінімальною за виключенням. Але це не мінімальна домінуюча множина: можна створити домінуючу множину не з чотирьох, а з трьох

ребер, які будуть суміжними до всіх ребер. Одна з таких мінімальних домінуючих множин $\{e_{11}, e_{12}, e_{13}\}$ представлена на мал. 3.11г.

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 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 усі суміжні комунікації, то маємо задачу: як знищити усі комунікації супротивника, завдавши найменшу кількість ударів по комунікаціях? Інколи знищувати одні комунікації легше (вага відповідного ребра менша), інші важче (більша вага). Тоді маємо зважену задачу: найменшою ціною знищити всі комунікації супротивника.

Розглянемо формулювання задачі про мінімальну (зважену) домінуючу множину ребер як задачі ЦЛП. Як і в попередніх задачах, введемо до розгляду вектор-стовпчик $\boldsymbol{e}$ довжиною m. Його координати є асоційованими с 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:

$$t = \sum_{k=1}^{m} e_k = (\mathbf{1},\, \boldsymbol{e})\,. \tag{3.29}$$

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:

$$\begin{cases} e_k = 0 \vee 1; \\ k = \overline{1,\, m}; \end{cases} \tag{3.30}$$

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 мал. 3.3 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, і т. д. Ось як виглядає ця система обмежень-нерівностей для графа з мал. 3.3:

$$\begin{aligned} e_1:&\quad e_1 + e_2 + e_3 + e_4 \geq 1; \\ e_2:&\quad e_2 + e_1 + e_3 + e_5 \geq 1; \\ e_3:&\quad e_3 + e_1 + e_2 + e_4 + e_5 \geq 1; \\ e_4:&\quad e_4 + e_1 + e_3 + e_5 \geq 1; \\ e_5:&\quad e_5 + e_2 + e_3 + e_4 \geq 1. \end{aligned} \tag{3.31}$$

Якщо скористатися матрицею суміжності ребер, яка для графа з мал. 3.3 має вигляд:

$$\boldsymbol{C} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \tag{3.32}$$

то умова суміжності всіх ребер до ребер домінуючої множини записується так:

$$(\boldsymbol{C} + \boldsymbol{E})\,\boldsymbol{e} \geq \mathbf{1}, \tag{3.33}$$

де $\boldsymbol{E}$ — одинична матриця. Т. ч., маємо задачу ЦЛП:

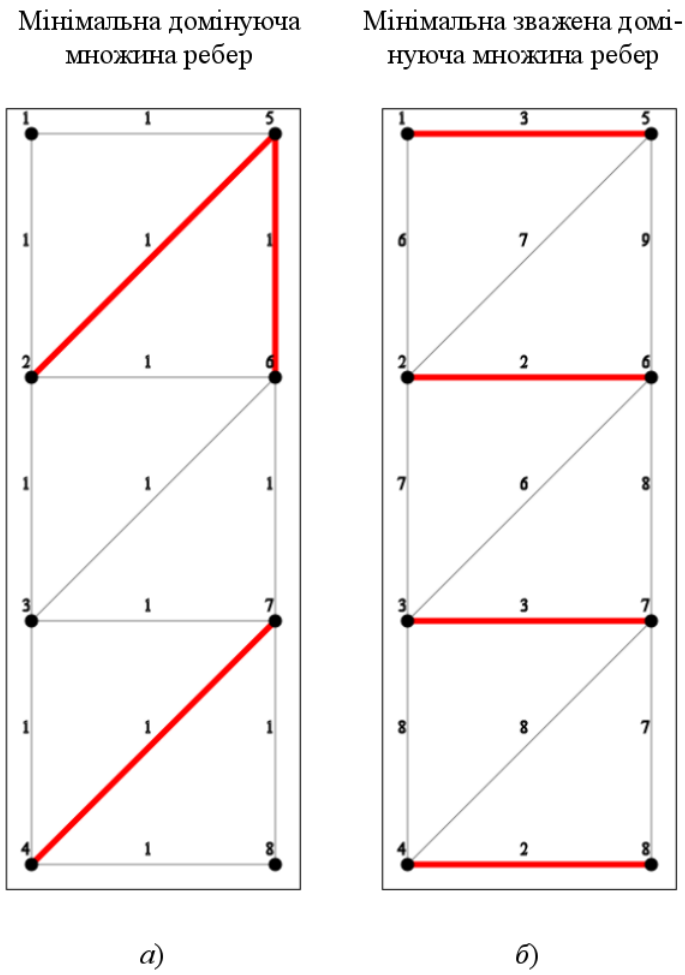


Рис. 3.12: Мінімальна домінуюча множина ребер (а) та мінімальна зважена домінуюча множина ребер (б)

$$\begin{cases} t = (\mathbf{1},\, \boldsymbol{e}) \to \min; \\ \boldsymbol{C}_1\boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}, \end{cases} \tag{3.34}$$

де

$$\boldsymbol{C}_1 = \boldsymbol{C} + \boldsymbol{E} \tag{3.35}$$

— матриця суміжності ребер з одиницями на головній діагоналі.

В задачі про максимальну зважену домінуючу множину ребер треба мінімізувати не загальну кількість ребер, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг ребер $\boldsymbol{c}$. Тоді замість (3.34) будемо мати задачу ЦЛП, що відрізняється від (3.34) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{c},\, \boldsymbol{e}) \to \min; \\ \boldsymbol{C}_1\boldsymbol{e} \geq \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.36}$$

На мал. 3.12 — результати розв'язання задачі про мінімальну домінуючу множину ребер для графа з мал. 3.11а. Зліва, на мал. 3.12а — незважена задача: біля кожного ребра проставлена його одинична вага. Знайдена мінімальна домінуюча множина з 3 ребер. Хоча вона й інша, ніж на мал. 3.11г, це все одно мінімальна домінуюча множина ребер: будь-яке ребро графа або входить до неї, або є суміжним з ребром з неї. Справа, на мал. 3.12б — результат розв'язання зваженої задачі (вага кожного ребра проставлена біля нього). Як бачимо, у мінімальній зваженій домінуючій множині може бути більше ребер, ніж у незваженій. В цьому прикладі загальна вага знайденої домінуючої множини з чотирьох ребер $3+2+3+2=10$ менша за загальну вагу будь-яких трьох ребер, суміжних до всіх ребер.

Зауважимо, що двоїста до (3.34) задача в силу симетрії матриці $\boldsymbol{C}_1$ буде мати вигляд:

$$\begin{cases} z = (\mathbf{1},\, \boldsymbol{e}) \to \max; \\ \boldsymbol{C}_1 \boldsymbol{e} \le \mathbf{1}; \\ e_k = 0 \vee 1;\ k = \overline{1,\, m}. \end{cases} \tag{3.37}$$

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.

3.7 Домінуюча множина вершин

Означення 3.7. *Домінуючою множиною вершин* у графі $G = (V,\, E)$ називається така підмножина вершин $V_1 \subseteq V$, 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* (absorbant set, external stability set). Домінуюча множина вершин називається *мінімальною за виключенням*, якщо будь-яка її підмножина з меншою кількістю вершин не є домінуючою. Домінуюча множина вершин називається *мінімальною*, якщо вона складається з мінімально можливої кількості вершин. ♢

На мал. 3.13а 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 мал. 3.13б. Її вершини є суміжними з усіма вершинами. Ця домінуюча множина не є мінімальною за виключенням: з неї можна вилучити, наприклад, вершину v_6 (мал. 3.13в). Ті вершини, що залишилися: $\{v_1,\, v_3,\, v_8\}$, також є суміжними з усіма вершинами. Значить, це домінуюча множина вершин. Якщо вилучити з неї ще будь-яку вершину, якась із вершин не охопиться суміжністю. Значить, домінуюча множина вершин $\{v_1,\, v_3,\, v_8\}$ є мінімальною за виключенням. Але це не мінімальна домінуюча множина: можна створити домінуючу множину не з трьох, а з двох вершин, які будуть суміжними до всіх вершин: це $\{v_4,\, v_5\}$. Вона представлена на мал. 3.13г.

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 загальною вагою.

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 усі опорні пункти супротивника, завдавши найменшу кількість ударів по опорних пунктах? Інколи знищувати одні опорні пункти легше (вага відповідної вершини менша), 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 задачі ЦЛП. 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 домінуючу множину, можна записати у вигляді:

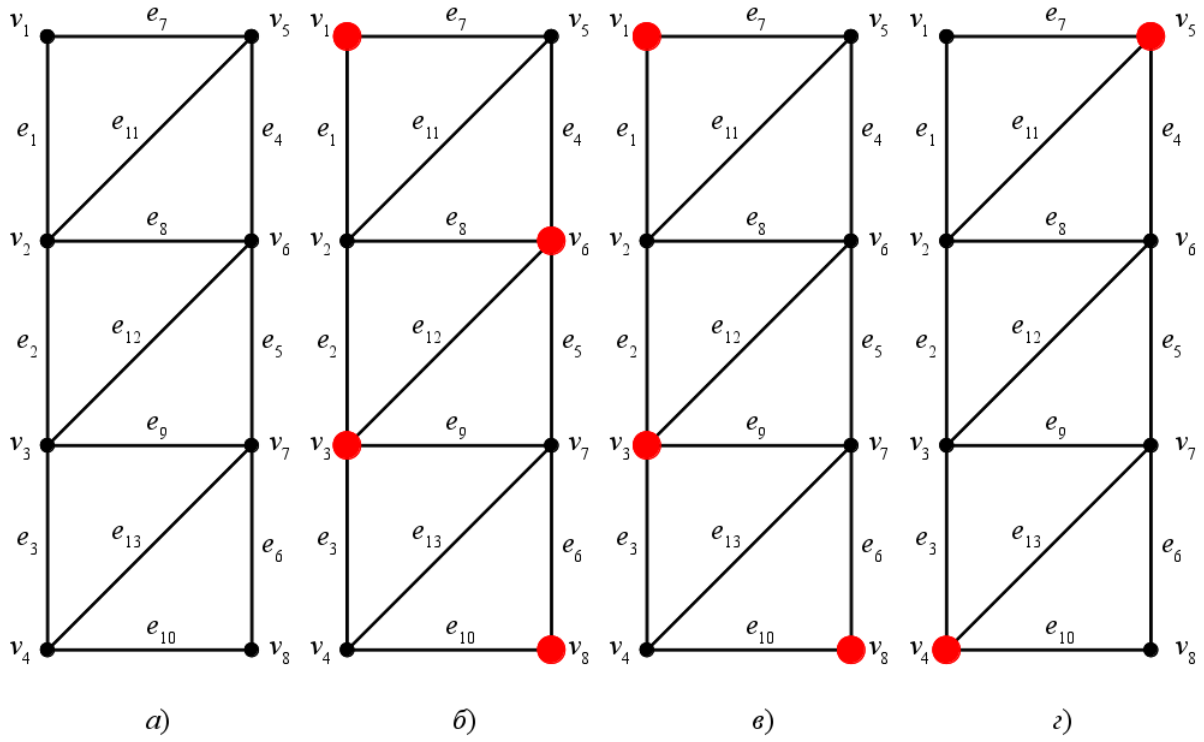


Рис. 3.13: Граф (а), його домінуюча множина вершин (б), мінімальна за виключенням домінуюча множина вершин (в) та мінімальна домінуюча множина вершин (г)

$$t = \sum_{i=1}^{n} v_i = (\mathbf{1},\, \boldsymbol{v})\,. \tag{3.38}$$

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:

$$\begin{cases} v_i = 0 \vee 1; \\ i = \overline{1,\, n}, \end{cases} \tag{3.39}$$

і 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 мал. 3.3 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 є суміжними 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, і т. д. Ось як виглядає ця система обмежень-нерівностей для графа з мал. 3.3:

$$\begin{array}{ll} v_1: & v_1 + v_2 + v_3 \geq 1; \\ v_2: & v_2 + v_1 + v_3 + v_4 \geq 1; \\ v_3: & v_3 + v_1 + v_2 + v_4 \geq 1; \\ v_4: & v_4 + v_2 + v_3 \geq 1. \end{array} \tag{3.40}$$

Якщо скористатися матрицею суміжності вершин, яка для графа з мал. 3.3 має вигляд:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \tag{3.41}$$

то умова суміжнисті всіх вершин до вершин домінуючої множини записується так:

$$(\boldsymbol{B} + \boldsymbol{E})\,\boldsymbol{v} \geq \mathbf{1}, \tag{3.42}$$

де $\boldsymbol{E}$ — одинична матриця. Т. ч., маємо задачу ЦЛП:

$$\begin{cases} t = (\mathbf{1},\, \boldsymbol{v}) \to \min; \\ \boldsymbol{B}_1 \boldsymbol{e} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}, \end{cases} \tag{3.43}$$

де

$$\boldsymbol{B}_1 = \boldsymbol{B} + \boldsymbol{E} \tag{3.44}$$

— матриця суміжності вершин з одиницями на головній діагоналі.

В задачі про максимальну зважену домінуючу множину вершин треба мінімізувати не загальну кількість вершин, а їхню загальну вагу. Позначимо вектор-стовпчик з ваг вершин $\boldsymbol{d}$. Тоді замість (3.43) будемо мати задачу ЦЛП, що відрізняється від (3.43) тільки цільовою функцією:

$$\begin{cases} t = (\boldsymbol{d},\, \boldsymbol{v}) \to \min; \\ \boldsymbol{B}_1 \boldsymbol{e} \geq \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.45}$$

На мал. 3.14 — результати розв'язання задачі про мінімальну домінуючу множину вершин для графа з мал.3.13а. Зліва, на мал. 3.14а — незважена задача: біля кожної вершини проставлена її одинична вага. Знайдена мінімальна домінуюча множина з 2 вершин, та ж сама, що на мал. 3.13г. Справа, на мал. 3.14б —

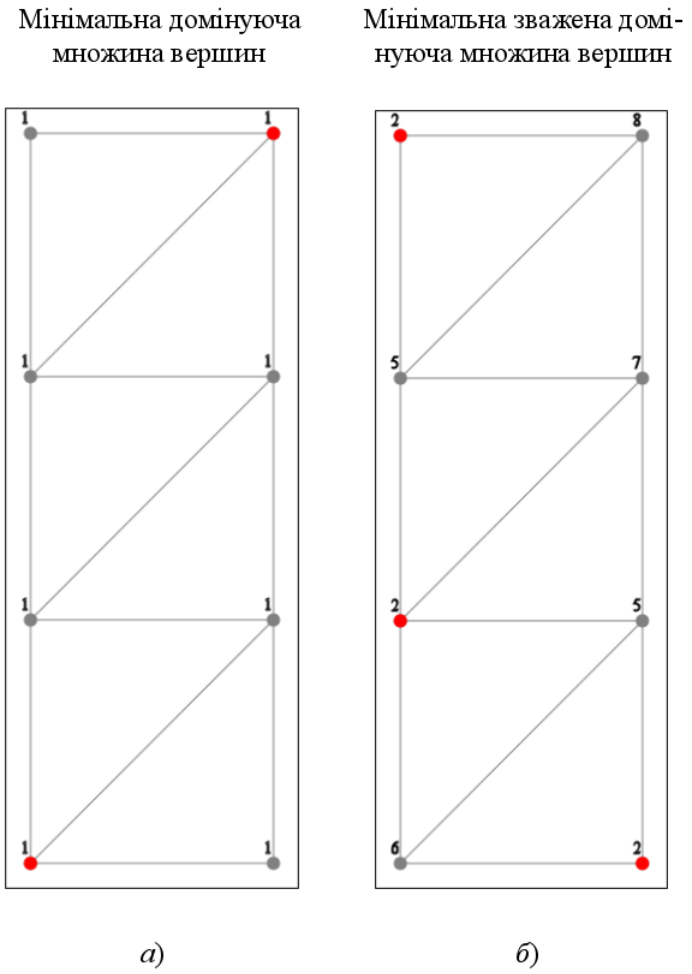


Рис. 3.14: Мінімальна домінуюча множина вершин (а) та мінімальна зважена домінуюча множина вершин (б)

результат розв'язання зваженої задачі (вага кожної вершини проставлена біля неї). Як бачимо, у мінімальній зваженій домінуючій множині може бути більше вершин, ніж у незваженій. В цьому прикладі загальна вага знайденої домінуючої множини з трьох вершин $2+2+2=6$ менша за загальну вагу будь-яких двох вершин, суміжних з усіма вершинами.

Зауважимо, що двоїста до (3.43) задача в силу симетрії матриці $\boldsymbol{B}_1$ буде мати вигляд:

$$\begin{cases} z = (\mathbf{1},\, \boldsymbol{v}) \to \max; \\ \boldsymbol{B}_1 \boldsymbol{e} \le \mathbf{1}; \\ v_i = 0 \vee 1;\ i = \overline{1,\, n}. \end{cases} \tag{3.46}$$

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 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.

3.8 Повний підграф

Означення 3.8. *Повний підграф* (complete subgraph) у графі $G = (V,\, E)$ — це підмножина вершин $V_1 \subseteq V$, 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. $\diamondsuit$

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) клику в графі $G = (V,\, E)$ еквівалентна до задачі про максимальну (зважену) незалежну множину для графа $\overline{G} = \big(V,\, \overline{E}\big)$, де $\overline{E}$ — ребра, яких немає в графі G. Тому, щоб звести задачу про максимальну кліку до задачі ЦЛП, треба:

1. побудувати граф $\overline{G}$, тобто знайти всі ребра, яких немає в G;
2. розв'язати для графа $\overline{G}$ задачу (3.12) про максимальну незалежну множину або (3.13) про максимальну зважену незалежну множину вершин.

Для графа з мал. 3.15 (біля вершин проставлені їхні номери) максимальною є кліка з чотирьох вершин. Наприклад, $\{v_2,\, v_3,\, v_6,\, v_7\}$. Але, якщо вершини зважені (мал. 3.16, біля кожної вершини її вага), то максимальною зваженою буде кліка з трьох вершин $\{v_8,\, v_9,\, v_{11}\}$. Її загальна вага $4+6+6=14$, що більше, ніж у будь-якої кліки з чотирьох вершин.

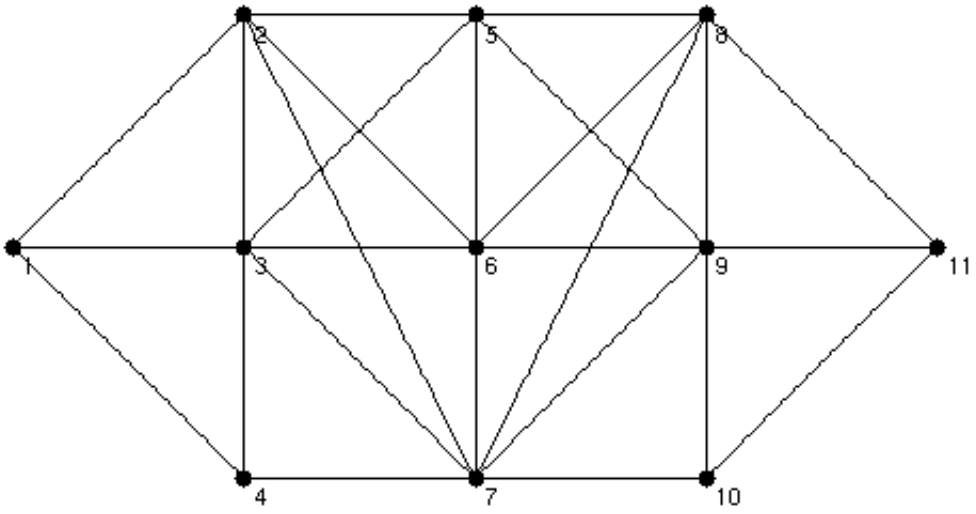


Рис. 3.15: Максимальна кліка

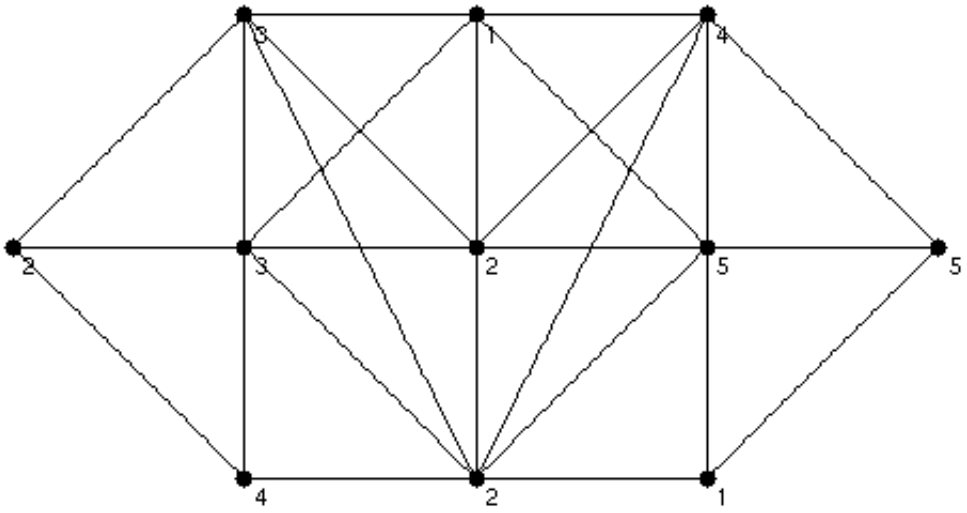


Рис. 3.16: Максимальна зважена кліка

Розділ 4

Правильна розфарбовка графів

У цій лекції будуть розглянуті 2 задачі: про правильну розфарбовку вершин та ребер графа. Обидві задачі формулюються як задачі цілочисельного лінійного програмування (ЦЛП).

4.1 Правильна розфарбовка вершин графа

Означення 4.1. Граф $G = (V, E)$ називається графом з *розфарбованими вершинами* (colored vertices), якщо задане відображення множини вершин V на множину натуральних чисел:

$$\varphi : V \to \mathbb{N}. \diamondsuit \tag{4.1}$$

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

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 наступній главі 5. Для цього знайдемо у графі максимальну незалежну множину вершин. Ці вершини є несуміжними між собою, тому їх можна пофарбувати одним кольором. Пофарбуємо їм 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 буде ця правильна розфарбовка мінімальною? Поглянемо на мал. 4.1.

На мал. 4.1а показаний граф з $n = 4$ вершинами та $m = 3$ ребрами. Якщо взяти максимальну незалежну множину вершин $\{v_1, v_3\}$ та надати її вершинам колір номер 1 (на мал. 4.1б 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 мал. 4.1в це синій). Як бачимо, в цьому

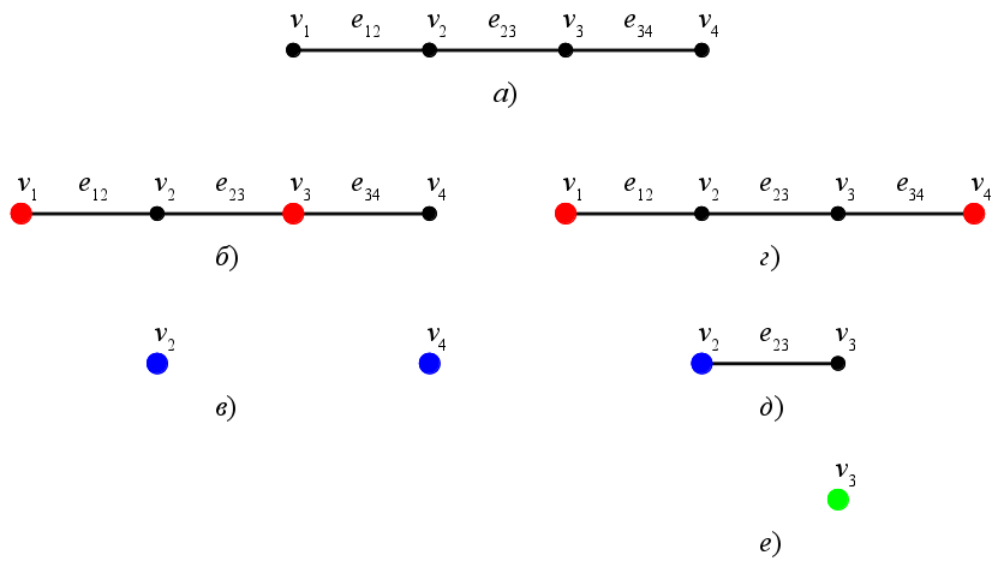


Рис. 4.1: Жадібний алгоритм не завжди дає мінімальну правильну розфарбовку вершин графа

випадку задача розв'язана вірно: маємо два кольори у мінімальній правильній розфарбовці. Менше не буває, тому що в графі є ребра.

Але якщо взяти максимальну незалежну множину вершин $\{v_1, v_4\}$, а це теж правильний результат розв'язання задачі про максимальну незалежну множину вершин (мал. 4.1г), то правильна розфарбовка вже не буде мінімальною. Дійсно, після вилучення вершин v_1, v_4 та інцидентних до них ребер отримаємо граф на мал. 4.1д. В ньому тільки одній вершині, наприклад, v_2, можна надати колір номер 2, а вершину v_3 вже треба фарбувати кольором номер 3 (мал. 4.1е).

Як бачимо, жадібний алгоритм не завжди дає правильний розв'язок задачі про мінімальну правильну розфарбовку вершин. Розглянемо алгоритм, що дає правильне розв'язання. Сформулюємо нашу задачу як задачі 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: це номер фарби вершины v_i. В найгіршому випадку кожна вершина буде пофарбована у свій колір, тому максимальне значення кожної змінної v_i — це n:

$$\begin{cases} v_i \in \{1,\, n\}\,; \\ i = \overline{1,\, n}. \end{cases} \tag{4.2}$$

Нам треба мінімізувати максимальне v_i:

$$t = \max_{i=\overline{1,\, n}} v_i \to \min\,. \tag{4.3}$$

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:

$$\begin{cases} v_i \le v_0; \\ i = \overline{1,\, n}. \end{cases} \tag{4.4}$$

Тоді цільова функція — це:

$$t = v_0 \to \min\,. \tag{4.5}$$

В 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:

$$\begin{cases} |v_i - v_j| \geq 1; \\ \forall e_{ij} \in E. \end{cases} \tag{4.6}$$

Ці обмеження (4.6) — нелінійні: в них є модуль. Перехід від кожної нерівності з модулем (4.6) до двох нерівностей без модулів:

$$\begin{cases} \left[\begin{array}{l} v_i - v_j \geq 1; \\ v_j - v_i \geq 1; \end{array} \right. \\ \forall e_{ij} \in E \end{cases} \tag{4.7}$$

теж нічого не дає, т. я. маємо не систему, а об'єднання нерівностей (треба, щоб виконувалася або одна, або інша нерівність). Але сформулювати (4.6) як систему лінійних нерівностей все ж можливо. Щоб це зробити, зауважимо спочатку, що змінні v_i відрізняються одна від одної не більш ніж на $n-1$, т. я. максимальний номер фарби — це n, а мінімальний — 1. Тому насправді замість (4.7) ми маємо:

$$\begin{cases} \left[\begin{array}{l} 1 \leq v_i - v_j \leq n-1; \\ 1 \leq v_j - v_i \leq n-1; \end{array} \right. \\ \forall e_{ij} \in E; \end{cases} \tag{4.8}$$

при цьому праві нерівності автоматично виконуються в силу (4.2). Тепер введемо до розгляду бінарні змінні e_{ij}, асоційовані з ребрами, яким дозволимо приймати лише одне з двох можливих значень — 0 або 1:

$$\begin{cases} e_{ij} = 0 \vee 1; \\ \forall e_{ij} \in E. \end{cases} \tag{4.9}$$

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

$$\begin{cases} v_i - v_j - n e_{ij} \leq -1; \\ v_j - v_i + n e_{ij} \leq n-1; \\ \forall e_{ij} \in E. \end{cases} \tag{4.10}$$

Якщо змінна e_{ij} приймає значення 0, то система (4.10) дає другу пару нерівностей з (4.8), а якщо 1 — то першу.

Т. ч., маємо таку задачу ЦЛП. Необхідно мінімізувати функцію t (4.5), яка формально залежить від $n+m+1$ змінних v_0, v_1, …, v_n, $\forall e_{ij}$, але фактично до неї входить тільки v_0. На змінні v_i накладені обмеження (4.4) — усього n обмежень. Для кожної змінної e_{ij} та відповідних до неї v_i та v_j повинні виконуватися по 2 обмеження (4.10) — усього $2m$ обмежень. Усі змінні v_i — цілочисельні та можуть приймати значення від 1 до n (4.2). Усі змінні e_{ij} — бінарні: вони можуть приймати значения тільки 0 або 1 (4.9).

На мал. 4.2 показаний результат роботи цього алгоритму: мінімальна правильна розфарбовка чотирьма фарбами одного з графів Петерсена.

4.2 Правильна розфарбовка ребер графа

Означення 4.3. Граф $G = (V, E)$ називається графом з *розфарбованими ребрами* (colored edges), якщо задане відображення множини ребер E на множину натуральних чисел:

$$\psi : E \to \mathbb{N}. \diamondsuit \tag{4.11}$$

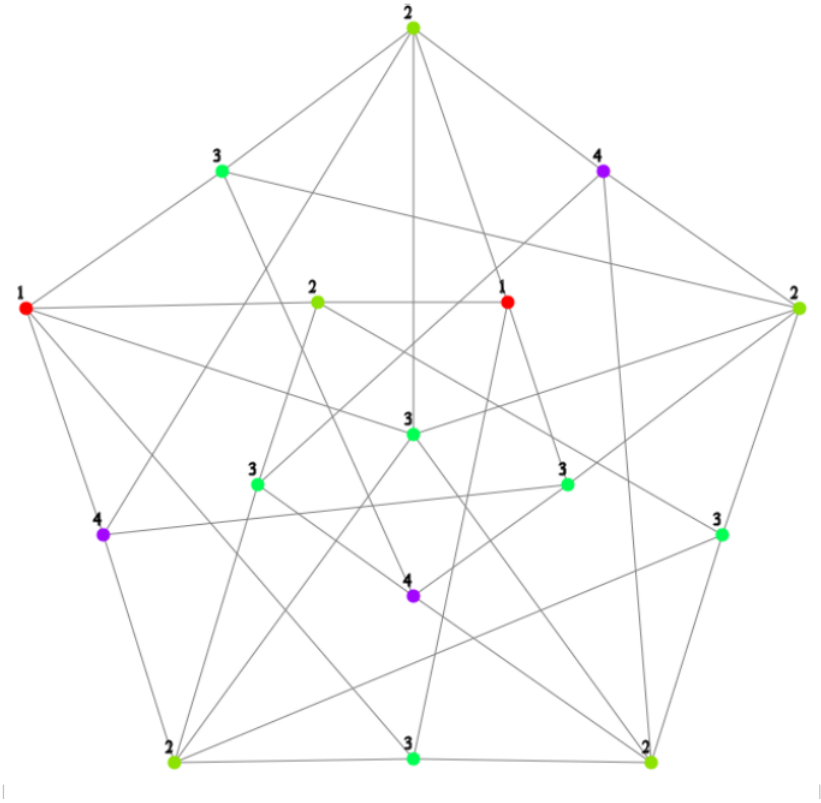


Рис. 4.2: Мінімальна правильна розфарбовка вершин графа Петерсена

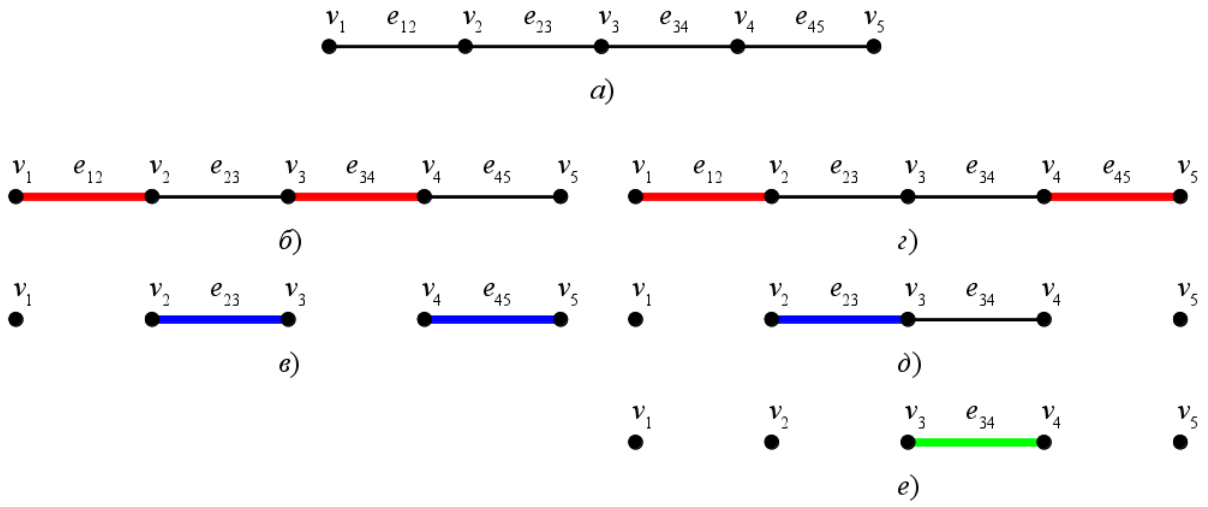


Рис. 4.3: Жадібний алгоритм не завжди дає мінімальну правильну розфарбовку ребер графа

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 4.4. Розфарбовка ребер графа називається *правильною* (regular edge coloring), якщо суміжні ребра фарбуються різними кольорами. *Мінімальною правильною розфарбовкою ребер* графа (minimal regular edge coloring) називається правильна розфарбовка мінімальною кількістю фарб. Кількість фарб у мінімальній правильній розфарбовці ребер графа називається *хроматичним індексом* графа (chromatic index). ♢

Якщо максимальний ступінь вершин графа позначити як δ, то, вочевидь, хроматичний індекс 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.

Теорема 4.1 Візінга (Vizing). Хроматичний індекс графа не перевищує $\delta + 1$. ♢

Т. ч., усі графи можна розбити на два класи за хроматичним індексом: клас "δ" та клас "$\delta + 1$". На жаль, визначення, до якого класу належить конкретний граф, є досить складною задачею.

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 їм 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 буде ця правильна розфарбовка мінімальною? Поглянемо на мал. 4.3.

На мал. 4.3а показаний граф з $n = 5$ вершинами та $m = 4$ ребрами. Якщо взяти максимальне паросполучення $\{e_{12}, e_{34}\}$ та пофарбувати його ребра у колір номер 1 (на мал. 4.3б це червоний), то на наступному кроці у графі залишаться лише ребра e_{23} та e_{45}, які не є суміжними. Цим ребрам можна надати колір номер 2 (на мал. 4.3в це синій). Як бачимо, в цьому випадку задача розв'язана вірно: маємо два кольори у мінімальній правильній розфарбовці. Менше не буває, тому що максимальний ступінь вершин у цьому графі дорівнює 2.

Але якщо взяти максимальне паросполучення $\{e_{12}, e_{45}\}$, а це теж правильний результат розв'язання задачі про максимальне паросполучення (мал. 4.3г), то правильна розфарбовка вже не буде мінімальною. Дійсно, після вилучення ребер e_{12} та e_{45} отримаємо граф на мал. 4.3д. В ньому тільки одному ребру, наприклад, e_{23}, можна надати колір номер 2, а ребро e_{34} вже треба фарбувати кольором номер 3 (мал. 4.3е).

Як бачимо, і в цій задачі жадібний алгоритм не завжди дає правильний результат. Розглянемо алгоритм, який дає правильне розв'язання нашої задачі. Сформулюємо 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: це номер фарби ребра e_k. В найгіршому випадку кожне ребро буде мати свій колір, тому максимальне значення кожної змінної e_k — це m:

$$\begin{cases} e_k \in \{1,\, m\}\,; \\ k = \overline{1,\, m}. \end{cases} \tag{4.12}$$

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

$$t = \max_{k=\overline{1,\, m}} e_k \to \min\,. \tag{4.13}$$

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:

$$\begin{cases} e_k \le e_0; \\ k = \overline{1,\, m}. \end{cases} \tag{4.14}$$

Тоді цільова функція — це:

$$t = e_0 \to \min\,. \tag{4.15}$$

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:

$$\begin{cases} |e_{i1} - e_{i2}| \ge 1; \\ \forall v_i \in V; \end{cases} \tag{4.16}$$

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 p_i, то усього для кожної вершини v_i буде $f_i = \frac{p_i(p_i-1)}{2}$ таких нерівностей. Як і в задачі про правильну розфарбовку вершин, ці нелінійні нерівності (4.16) можна звести до лінійних, якщо ввести до розгляду f_i бінарних змінних для кожної вершини v_i:

$$\begin{cases} v_{ij} = 0 \vee 1; \\ j = \overline{1,\, f_i}; \\ \forall v_i \in V. \end{cases} \tag{4.17}$$

Тоді маємо систему нерівностей:

$$\begin{cases} e_{i1} - e_{i2} - m v_{ij} \le -1; \\ e_{i2} - e_{i1} + m v_{ij} \le m - 1; \\ j = \overline{1,\, f_i}; \\ \forall v_i \in V. \end{cases} \tag{4.18}$$

Якщо змінна v_{ij} приймає значення 0, то система (4.18) дає нерівність (4.16) в один бік, а якщо 1 — то в інший. За теоремою Візінга у цих формулах замість m можна поставити $\delta + 1$.

Т. ч., маємо таку задачу ЦЛП. Необхідно мінімізувати функцію t (4.15), що формально залежить від $1 + m + f_1 + f_2 + \ldots + f_n$ змінних: e_0, e_1, …, e_m, $\forall v_{ij}$, але фактично до неї входить тільки e_0. На змінні e_k накладені обмеження (4.14) — усього m обмежень. Для кожної змінної v_{ij} та відповідних e_{i1} і e_{i2} повинні виконуватися по 2 обмеження (4.18) — усього $2\,(f_1 + f_2 + \ldots + f_n)$ обмежень. Усі змінні e_k — цілочисельні та можуть приймати значення від 1 до m (4.12). Усі змінні v_{ij} — бінарні: вони можуть приймати тільки значення 0 або 1 (4.17).

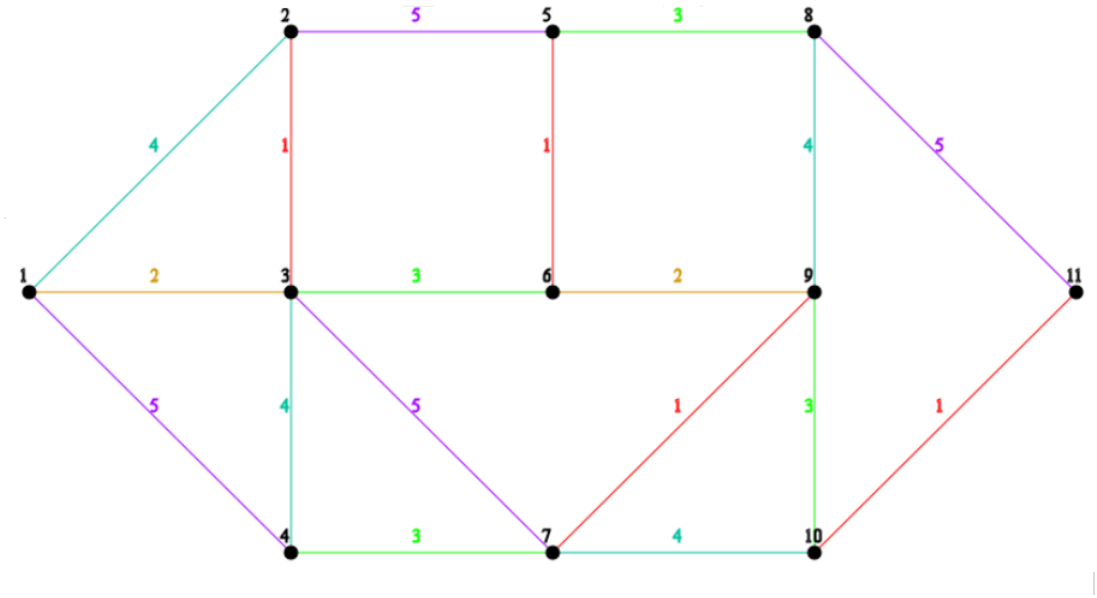


Рис. 4.4: Мінімальна правильна розфарбовка ребер графа

Слід зауважити, що обмежень (4.18) буде дуже багато, і вони є досить складними для сучасних алгоритмів розв'язання задачі ЦЛП. Так, для досить простого графа з мал. 4.4 будемо мати $3+3+10+3+3+3+6+3+6+3+1=44$ пари, тобто 88 обмежень (4.18). Це — найбільший граф, для якого мені вдалося правильно розфарбувати ребра за допомогою пакета Glpk.js на сайті [9].

Розділ 5

Мінімальні остовні дерева

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

5.1 Жадібні алгоритми та матроїди

Означення 5.1. *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 початкого плану перевезень у транспортній задачі. Побудований цим методом план може виявитися не оптимальним, і знадобиться перекидання вантажу за циклом. Інший приклад ми розглянули у попередній главі: 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 мал. 5.1.

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

$$\boldsymbol{C} = \begin{pmatrix} 8 & 7 & 7 & 6 \\ 7 & 7 & 5 & 8 \\ 8 & 6 & 6 & 7 \end{pmatrix}. \tag{5.1}$$

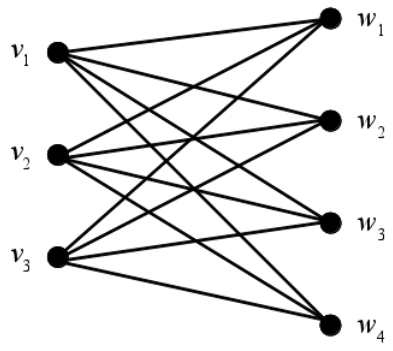


Рис. 5.1: Приклад задачі про призначення

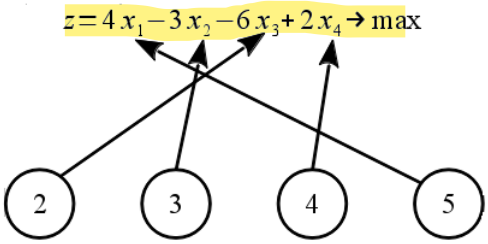


Рис. 5.2: Максимум лінійної функції на перестановці значень її аргументів

Скористаємося жадібним алгоритмом. Ми бачимо, що 1й працівник найкраще впорається з 1ю роботою, тому включимо до паросполучення ребро e_{11}. Другого працівника спрямовуємо на 4у роботу, т. я. саме на ній його продуктивність є максимальною: додаємо ребро e_{24}. Тепер у 3го працівника залишився невеликий вибір: або 2а, або 3я роботи з одинаковою продуктивністю 6. Оберемо, наприклад, e_{23}. Т. ч., побудовано максимальне за включенням зважене паросполучення $\{e_{11}, e_{24}, e_{32}\}$ з загальною вагою 22. Але цей розв'язок — не найкращий. Можна взяти $\{e_{12}, e_{24}, e_{31}\}$ 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 максимум лінійної функції n змінних $z = (\boldsymbol{c}, \boldsymbol{x}) \to \max$ на перестановці P_n значень її аргументів $\boldsymbol{x}$. Скажімо, такої:

$$z = 4x_1 - 3x_2 - 6x_3 + 2x_4 \to \max, \tag{5.2}$$

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 та 5, щоб максимізувати функцію (5.2)? 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 мал. 5.2. Отримаємо $z = 7$. Можна перебрати всі 24 перестановки та переконатися, що дійсно знайдене максимальне значення. Доведення цього факту дуже просте: для кожної пари коефіцієнтів та значень аргументів має місце:

$$\begin{gathered}(c_i > c_k) \cap (x_i > x_k) \Rightarrow c_i x_i + c_k x_k > c_i x_k + c_k x_i : \\ c_i x_i + c_k x_k - c_i x_k - c_k x_i = c_i (x_i - x_k) - c_k (x_i - x_k) = (c_i - c_k)(x_i - x_k) > 0.\end{gathered} \tag{5.3}$$

Чому ж в останньому прикладі жадібний алгоритм привів до успіху, а у 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 1му етапі ми забрали ребро e_{11}, 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_{12} і e_{31}, 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. Ось як виглядає, наприклад, означення матроїда через незалежні множини.

Означення 5.2. *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 умови:

1. серед елементів I є пуста підмножина: $\varnothing \in I$;

2. якщо якась множина A елементів S належить до I, то й будь-яка власна підмножина A 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, знайдеться такий елемент x, що його об'єднання з B також буде належати до I: $(\forall A,\, B \in I) \cap (|A| > |B|) : \; \exists x \in A \setminus B : \; B \cup \{x\} \in I$. $\diamondsuit$

У нашому випадку носієм матроїда буде множина усіх можливих упорядкованих пар $(c_i,\, x_k)$, де c_i — це коефіцієнти функції (5.2), 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:

$$\begin{aligned} S = \{&(c_1,\, x_1)\,,\; (c_1,\, x_2)\,,\; (c_1,\, x_3)\,,\; (c_1,\, x_4)\,,\\ &(c_2,\, x_1)\,,\; (c_2,\, x_2)\,,\; (c_2,\, x_3)\,,\; (c_2,\, x_4)\,,\\ &(c_3,\, x_1)\,,\; (c_3,\, x_2)\,,\; (c_3,\, x_3)\,,\; (c_3,\, x_4)\,,\\ &(c_4,\, x_1)\,,\; (c_4,\, x_2)\,,\; (c_4,\, x_3)\,,\; (c_4,\, x_4)\}\,. \end{aligned} \tag{5.4}$$

Сімейство незалежних множин I створимо з елементів S таким чином: з кожного рядка формули (5.4) оберемо не більше одного елемента, і їхня сукупність і буде одним з елементів множини I. Наприклад, можна 1го рядка (5.4) обрати елемент $(c_1,\, x_3)$, з 2го — $(c_2,\, x_2)$, з 3го — нічого, з 4го — $(c_4,\, x_2)$. 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 5.2 додамо до I пусту множину $\emptyset$, тобто з кожного рядка (5.4) не обираємо жодного елемента. Той матроїд, який ми побудували, називається *матроїдом розбиттів* (partition matroid). В загальному випадку для отримання матроїда розбиттів треба здійсними розбиття множини S на деякі непусті підмножини, та обрати з кожної підмножини не більше заданої кількості елементів. Ми розбили S на 4 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 5.2. Зокрема, умова 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 $B \vee \{x\} \in I$.

Означення 5.3. *Базою матроїда* (base, basis of matroid) називається будь-який максимальний за включенням елемент множини I. $\diamondsuit$

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 B. Оскільки бази — це теж незалежні множини, то $B \subset I$. Для баз матроїда мають місце такі властивості, які ми залишимо без доведення.

1. Множина B є непустою (очевидно).

2. Якщо $B_1,\, B_2 \in B$, і $B_1 \neq B_2$, то $B_1 \nsubseteq B_2$ і $B_2 \nsubseteq B_1$ (теж очевидно: бази є максимальними за включенням і не можуть бути підмножинами одна одної).

3. Якщо $B_1,\, B_2 \in B$, то $\forall x \in B_1 \setminus B_2 : \exists y \in B_2 \setminus B_1$ такий, що $(B_1 \setminus \{x\}) \cup \{y\} \in B$. Пояснення до цієї умови. 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.

4. Усі бази мають одинакову потужність (кількість елементів): $|B_1| = |B_2| = \ldots$.

Означення 5.4. *Рангом матроїда* (rank of matroid) називається потужність будь-якої його бази. $\diamondsuit$

Побудований нами на множині (5.4) матроїд розбиттів має ранг 4, бо кожна з його баз містить 4 елементи — по одному з кожного рядка формули (5.4). Усього матроїд має $|B| = 4^4 = 256$ баз.

Реалізація жадібного (чи будь-якого іншого) алгоритму передбачає введення вагової функції на елементах S. У нашому випадку вага кожного елемента — це добуток c_i на x_k:

$$w\,(c_i,\, x_k) = c_i x_k; \tag{5.5}$$

а вага бази — це сума ваг її елементів. Для розв'язання задачі (5.2) нам треба знайти таку базу, в якій усі x_k різні, та яка має максимальну вагу. Одна з основних лем теорії матроїдів стреджує.

Лема 5.1. Жадібний алгоритм завжди будує базу максимальної ваги. ♢

Різні доведення цієї леми можна знайти в Інтернеті. Найпростіше з них проводиться від протилежного. Знайдіть його самостійно.

За цією лемою жадібний алгоритм побудови бази A найбільшої ваги виглядає так.

1. Сортуємо елементи S у порядку зменшення ваги.
2. $A = S_1$ (додаємо до шуканої множини елемент найбільшої ваги).
3. Поки $|A| < 4$, додаємо до A наступний елемент, якщо його x_k ще немає серед елементів A і отримана нова множина $A \in I$. У нашому випадку додається перший серед наступних елементів S з тими c_i та x_k, яких ще немає в A.

В результаті буде побудована база максимальної ваги з різними x_k. Як бачимо, досить складна теорія матроїдів пояснює, чому формула (5.3) 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.

5.2 Мінімальне остовне дерево (МОД)

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

Означення 5.5. *Шляхом* (маршрут, path) у графі $G = (V, E)$ — це послідовність вершин та ребер виду $v_1\, e_1\, v_2\, e_2\, \ldots\, v_k$, у якій сусідні елементи є інцидентними. ♢

Означення 5.6. Маршрут називається *простим* (simple path), якщо кожна вершина зустрічається в ньому лише один раз. ♢

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

Означення 5.7. Граф $G = (V, E)$ називається *зв'язним* (connected graph), якщо існує шлях із будь-якої його вершини в 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 самому 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 5.1. 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 з собою нульовою кількістю ребер. Дві вершини можна поєднами 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 kго 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 в графі з n вершинами, щоб він залишився зв'язним: $n - 1$.

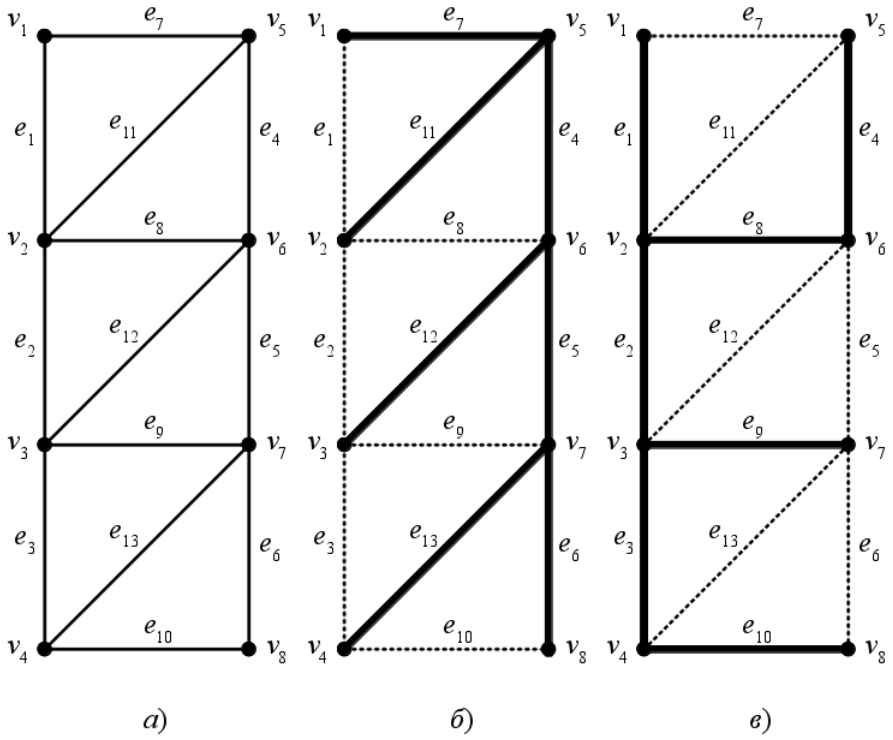


Рис. 5.3: Граф (а) та деякі з його остовних дерев (б, в)

Означення 5.8. Зв'язний граф $G = (V, E)$ з n вершинами та $n-1$ ребрами називається *остовним деревом* (spanning tree). ♢

На мал. 5.3 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 властивоті, які ми зараз розглянемо. Але спочатку ще деякі означення.

Означення 5.9. Шлях $v_1\, e_1\, v_2\, e_2 \ldots v_k$ називається *циклом* (cycle), якщо в ньому початкова та кінцева вершини співпадають. ♢

Означення 5.10. 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, значить, немає і повторюваних ребер. Вочевидь, з будь-якого непростого цикла можна виділити принаймні два різних простих цикли, а з непростого маршруту — принаймні один простий цикл.

Тепер розглянемо властивості остовних дерев.

Властивість 5.1. В остовному деревеі $G_1 = (V, E_1)$ 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 цикла, що залишилася. Значить, у G_1 більше, ніж $n-1$ ребер: видалення з нього одного ребра не порушує зв'язності. Т. ч., G_1 не є остовним деревом. ♢

Властивість 5.2. І 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 таки $(n-1)$ ребер, що утворюють остовне дерево. Тоді кожне ребро з решти додає інший шлях між якоюсь парою вершин, тобто утворює цикл. ♢

Властивість 5.3. В остовному дереві G_1 зв'язного графа існує єдиний 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 $v_i \ldots v_j \ldots v_i$, що протирічить властивості 5.2. ♢

Властивість 5.4. Навпаки, якщо у зв'язному графі $G = (V, E)$ існує єдиний 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 5.2 у ньому є хоча б один цикл, а значить, існує більш ніж один шлях між якоюсь парою вершин. ♢

Т. ч., 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 єдиний шлях з будь-якої вершины в будь-яку іншу.

Якщо подивитися на мал. 5.3, можна побачити, що в остовних деревах дійсно немає циклів, і з будь-якої вершини можна дістатися будь-якої іншої лише єдиним шляхом.

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 алгоритма для знаходження МОД. Спочатку ми розглянемо два найпоширеніших алгоритма побудови МОД. Далі доведемо, что обидва вони призводять до одного й того ж правильного результату. І, нарешті, покажемо, що знаходження МОД — це задача на матроїді, тобто жадібний алгоритм тут дійсно працює.

В алгоритмах, що ми розглянемо, основний принцип побудови МОД такий: будемо не видаляти ребра з заданого графа, а додавати їх у МОД, що будується. В описах алгоритмів будемо позначати ребро, що поєднує вершини v_i та v_j, як e_{ij}, тобто двома індексами поєднуваних вершин.

Алгоритм Прима (Prim's algorithm). Він показаний на мал. 5.4. 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. Візьмемо перше ребро мінімальної ваги, що виходить з вершини v_1. Таким ребром буде e_{12}: його вага 2 менша за вагу 3 ребра e_{14}. Тому починаємо побудову МОД з нього (а). Далі переглядаємо всі ребра, суміжні до вже побудованого фрагменту МОД, тобто до e_{12}. Всього таких ребер 4 (одне у вершині v_1 та три у v_2). Обираємо з них ребро мінімальної ваги. Мінімальна вага серед цих 4 ребер — 1, вона у ребра e_{24}. Його й приєднуємо до МОД, що будується (б). Продовжуємо переглядати ребра, суміжні до вже побудованого фрагменту МОД і такі, що не утворюють у ньому циклів. Беремо всі ребра, інцидентні до v_1, v_2 та v_4 (крім e_{14} — його брати не можна, бо утворюється цикл), та обираємо з них ребро мінімальної ваги. Мінімальна вага 2 — у ребра e_{25}, йо й приєднуємо (в). Продовжуємо процес. Тепер у нас з'явилась можливість приєднати до МОД ребро e_{56} вагою 1 (г), а потім — e_{69} (д). Серед ребер, що залишилися, мінімальна вага 2 — у ребра e_{58}, його також можно долучити до МОД (е). І, нарешті, приєднуємо ребра e_{36} і e_{47} ваги 3 (є, ж). Отримали 8 ребер — МОД побудоване. Його вага 15.

Алгоритм Краскала (Kruskal's algorithm). Він показаний на мал. 5.5. Основна його відмінність від алгоритма Прима — 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_{24} — воно зустрілося першим (а). Наступне ребро ваги 1 — це e_{56}, і воно не утворює циклів — додаємо його (б). Далі перевіряємо e_{69}: циклів немає, долучаємо (в). Всі ребра з вагою 1 вичерпані. Переходимо до ребер з мінімальною вагою 2. Ребро e_{12} можно приєднати (г), e_{25} — також (д), e_{58} теж підходить (е). Далі — ребра з вагою 3. Ребро e_{14} не годиться (утворюється цикл), а ось e_{36} підходить (є), і e_{47} — також (ж). В результаті отримали те же саме МОД ваги 15, что й за алгоритмом Прима.

Як бачимо, ці алгоритми відрізняються порядком приєднання ребер. В алгоритмі Прима ми весь час приєднували суміжні ребра, тобто нарощували МОД, залишаючі його зв'язним. В алгоритмі Краскала будуються окремі частини МОД, які потім поєднуються. Кожна з цих частин називається деревом, а їхня сукупність — лісом.

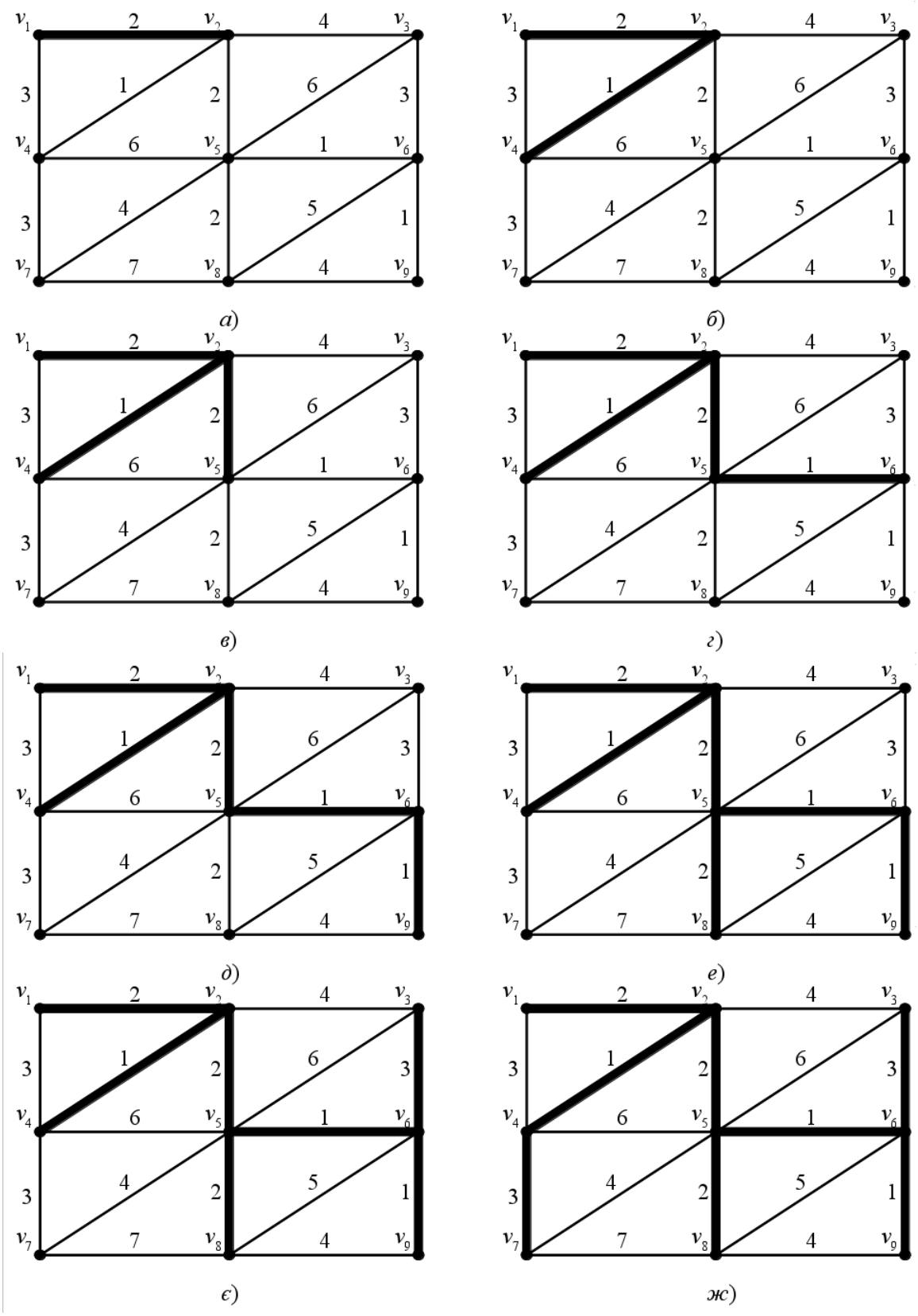


Рис. 5.4: Алгоритм Прима

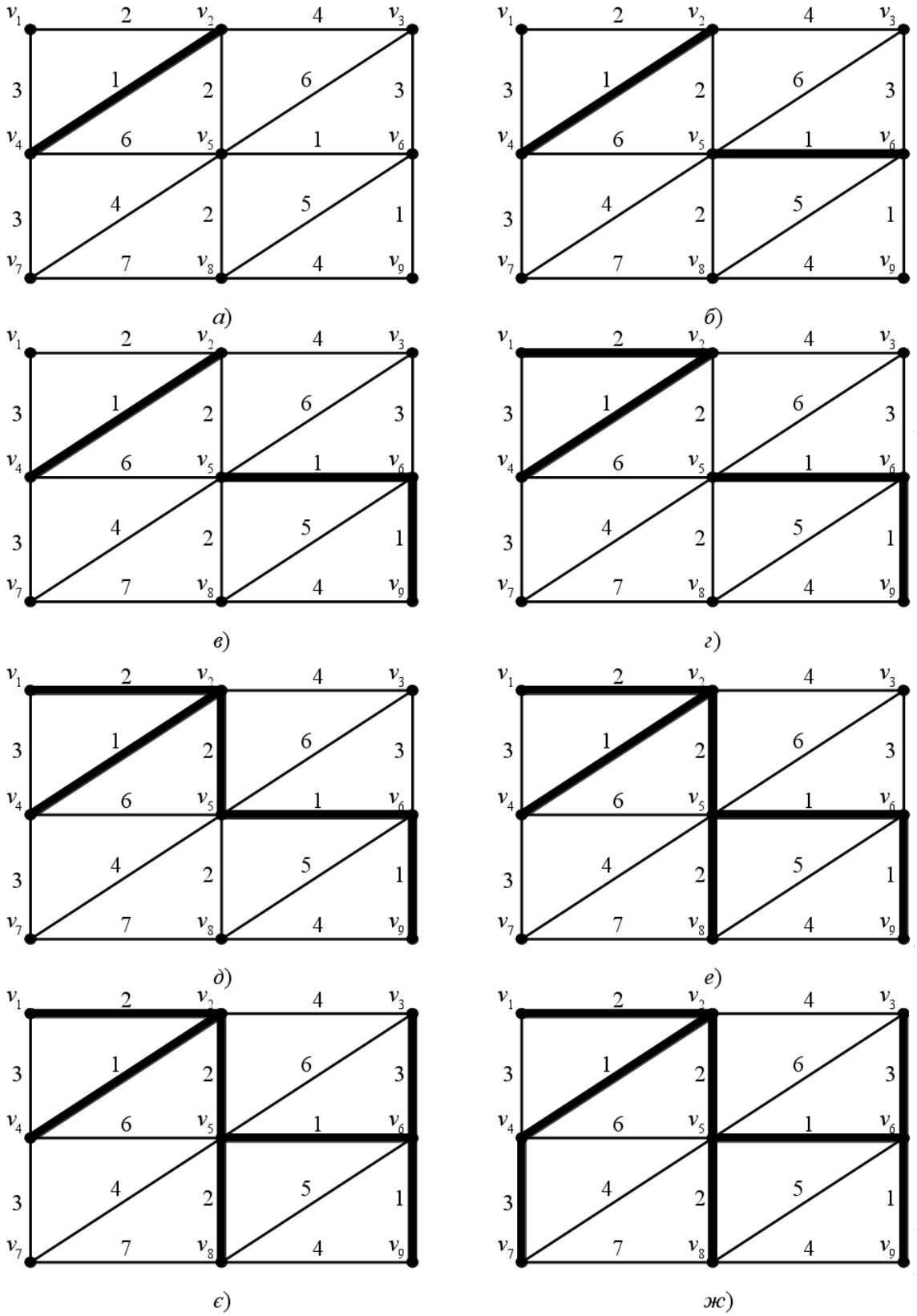


Рис. 5.5: Алгоритм Краскала

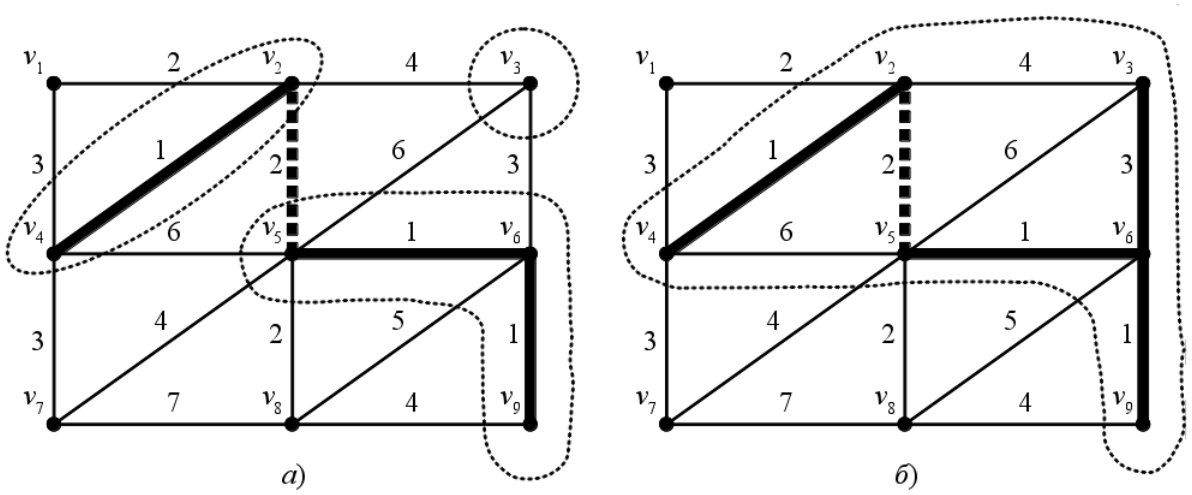


Рис. 5.6: До доведення теореми 5.2

Означення 5.11. *Деревом* (tree) називається остовне дерево, побудоване на деякій підмножині вершин $V_1 \subseteq V$ та інцидентних до всіх них ребер. ♢

Означення 5.12. *Лісом* (forest) називається множина дерев, що не перетинаються. ♢

Наприклад, ліс на мал. 5.5г складається з дерев, побудованих на підмножинах вершин $\{v_1, v_2, v_4\}$, $\{v_3\}$, $\{v_5, v_6, v_9\}$, $\{v_7\}$ та $\{v_8\}$. У підмножинах з однієї вершини ніяких ребер немає, и відповідне дерево складається тільки з вершини.

Зараз ми сформулюємо та доведемо теорему, що встановлює можливість застосування жадібного алгоритму до побудови МОД. Зауважимо спочатку, що МОД може бути й не єдиним. Якщо у графі багато ребер одинакової малої ваги, то, цілком можливо, існує кілька МОД. Зокрема, якщо всі ребра мають одинакову вагу, то будь-яке остовне дерево буде мінімальним. Тому ми будемо казати не про МОД, а про одне з МОД або будь-яке МОД.

Теорема 5.2. Нехай на підмножинах вершин $V_1, V_2, \ldots, V_k$, що не перетинаються між собою, побудовані дерева мінімальної ваги $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, $\ldots$, $G_k = (V_k, E_k)$. Нехай також e_j — ребро мінімальної ваги, один кінець якого входить до G_1 (тобто є інцидентнин до якоїсь вершини з V_1), а інший — у якесь інше дерево: $G_2, G_3, \ldots, G_k$. Тоді серед усіх дерев мінімальної ваги, побудованих на об'єднанні ребер $\cup E_i$, існує дерево мінімальної ваги, що вміщує ребро e_j. ♢

Пояснимо формулювання теореми на прикладі. Візьмемо один з проміжних етапів побудови МОД, показаний на мал. 5.5в. Намалюємо його окремо (мал. 5.6а). Тут у нас побудовано кілька мінімальних дерев. Розглянемо випадок $k = 3$. Нехай, наприклад, $V_1 = \{v_2, v_4\}$, $V_2 = \{v_5, v_6, v_9\}$, $V_3 = \{v_3\}$, і на кожній підмножині вершин побудоване мінімальне дерево. Ці дерева обведені штриховими контурами. У нас $E_1 = \{e_{24}\}$, $E_2 = \{e_{56}, e_{69}\}$, $E_3 = \emptyset$. Знайдемо ребро мінімальної ваги, що виходить із G_1 у бік G_2 або G_3. У бік G_2 з G_1 виходять ребра e_{25} і e_{45}, а вбік G_3 — e_{23}. Із цих трьох ребер мінімальна вага 2 — у ребра e_{25}. У формулюванні теореми — це e_j, воно позначене на малюнку жирною штриховою лінією. Теорема стверджує наступне. Якщо ми хочемо побудувати мінімальне дерево на множині вершин $\{v_2, v_4, v_5, v_6, v_9, v_3\}$, то серед усіх мінімальних дерев, у яких є ребра e_{24}, e_{56}, e_{69}, напевно знайдеться хоча б одне мінімальне дерево, що містить e_{25} (мал. 5.6б).

Доведення. Від протилежного: припустимо, що існує дерево $G_0 = (V_0, E_0)$, побудоване на об'єднанні вершин: $V_0 = \cup V_i$, яке включає в себе об'єднання ребер: $E_0 = \cup E_i$, але не включає ребро e_j, і яке має вагу меншу, ніж вага будь-якого мінімального дерева, що включає e_j. В ілюстрації на мал. 5.6 таке було б, якби замість e_{25} ми спробували б поєднати G_1 з G_2 ребром e_{45}. Доведемо, що таке неможливо. Для цього додамо до G_0 ребро e_j. Після цього G_0 вже не буде деревом: в ньому утвориться один цикл. Цей цикл буде заходити в G_1, але не буде знаходитися в ньому цілком: він буде заходити також і в якусь іншу множину вершин з V_2,

V_3, …, V_k. Дійсно: один з кінців e_j є інцидентним до вершини з G_1, а інший — до якоїсь іншої вершини з G_2, G_3, …, G_k. Так, у графі на мал. 5.6 утворився б цикл $v_2\, e_{25}\, v_5\, e_{45}\, v_4\, e_{24}\, v_2$. Цей цикл заходить у G_1, але охоплює ще й G_2. Видалимо з цього цикла якесь інше ребро, що одним кінцем входить до G_1, а іншим — в іншу G_i. Таке ребро напевно існує, інакше не було б цикла, що охоплює G_1 та якусь іншу G_i. І вага ребра, що видаляється, не менша (а може й більша), ніж у приєднуваного e_j. У прикладі з графом на мал. 5.6 ми видаляємо e_{45}. Зв'язність усіх вершин у V_0 при цьому зберігається: ми можемо перейти від G_1 до іншої G_i через e_j замість видаленого ребра. Значить, після такої заміни ребер у нас залишиться дерево, і вага його — не менша, ніж у початкового дерева. А це протирічить припущенню про те, що G_0 має меншу вагу, ніж у дерева з e_j. ♢

За цією теоремою ми можемо стверджувати, що обидва алгоритми, розглянуті вище, дають МОД (хоча, може, й різні). В алгоритмі Прима спочатку кожне G_i складається з однієї вершини без ребер. На першому етапі ми обираємо ребро мінімальної ваги. Воно поєднує дві G_i, одну з яких можна позначити як G_1. Значить, воно напевно увійде до одного з МОД. Далі позначаємо як G_1 мінімальне дерево, утворене двома вершинами та ребром, що їх поєднує, і знаходимо ребро мінімальної ваги, суміжне до G_1 одним зі своїх кінців. За теоремою 5.2 це e_j. Згідно цієї теореми ми можемо приєднувати його до дерева, що будується, при цьому не втрачається можливість побудувати одне з МОД. Ну а алгоритм Краскала — це просто ілюстрація до доведення теореми!

Поглянемо на проблему побудови МОД як на задачу на матроїді. За носій матроїда S (див. означення 5.2) візьмемо множину всіх ребер графа. Розглянемо будь-яку підмножину ребер, в якій немає циклів, тобто будь-яке дерево або ліс, включно з пустою множиною. Перевіримо систему таких підмножин на незалежність згідно означення 5.2. Пусту множину ми включили до системи, тому умова 1 виконується. Якщо з дерева або лісу вилучити одне ребро або кілька ребер, то циклів не утвориться. Залишиться дерево або ліс, який теж є у нашій системі підмножин. Значить, умова 2 теж виконується. Залишилося довести виконання умови 3. Сформулюємо її у вигляді теореми.

Теорема 5.3. 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 після долучення його до B залишає B деревом або лісом, тобто не створює у ньому циклів. ♢

Доведення. Зауважимо спочатку, що, оскільки $|A| > |B|$, то у меншому дереві (лісі) B буде менше, ніж $n-1$ ребер. Значить, B не може бути остовним деревом і, т. ч., має більше однієї компоненти зв'язності (ізольована 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 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 висновку, що у дереві (лісі) A не більше ребер, ніж у B, що протирічить умові. Значить, 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 5.3: множина усіх дерев та лісів графа є незалежною множиною матроїда, побудованого на носії — множині його ребер. Цей матроїд так і називається:

Означення 5.13. *Графовий матроїд* (graphic matroid) — це матроїд, побудований на носії S — множині всіх 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. Алгоритм Краскала — 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 Прима є додаткова умова для долучення ребер: їхня суміжність з тим деревом, що вже побудоване. На мою думку, алгоритм Прима більш придатний для розв'язання задачі вручну, а алгоритм Краскала — для програмування.

Перевірка множини ребер на незалежність (відсутність циклів) може здійснюватися, наприклад, за таким алгоритмом.

1. Обчислюємо у побудованому дереві (лісі) ступені всіх вершин. Для цього можна, наприклад, додати всі

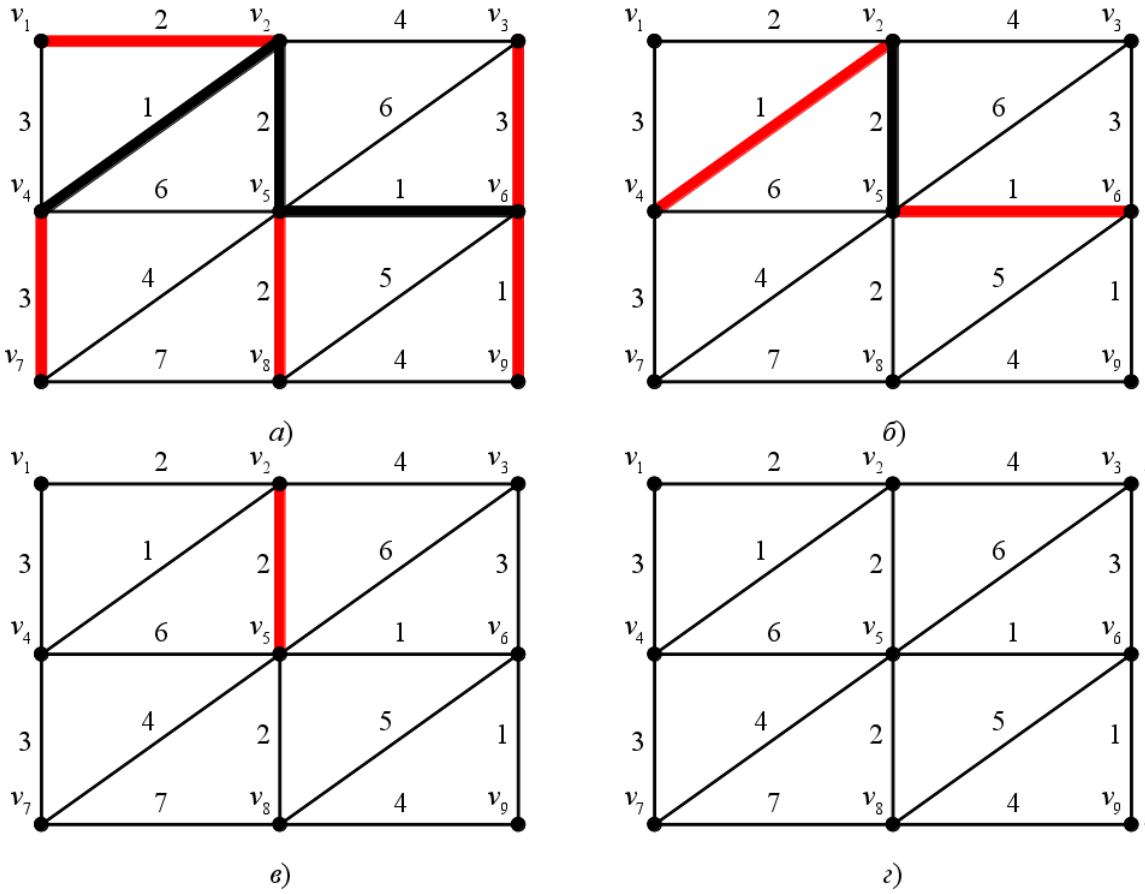


Рис. 5.7: Перевірка на відсутність циклів (немає циклів)

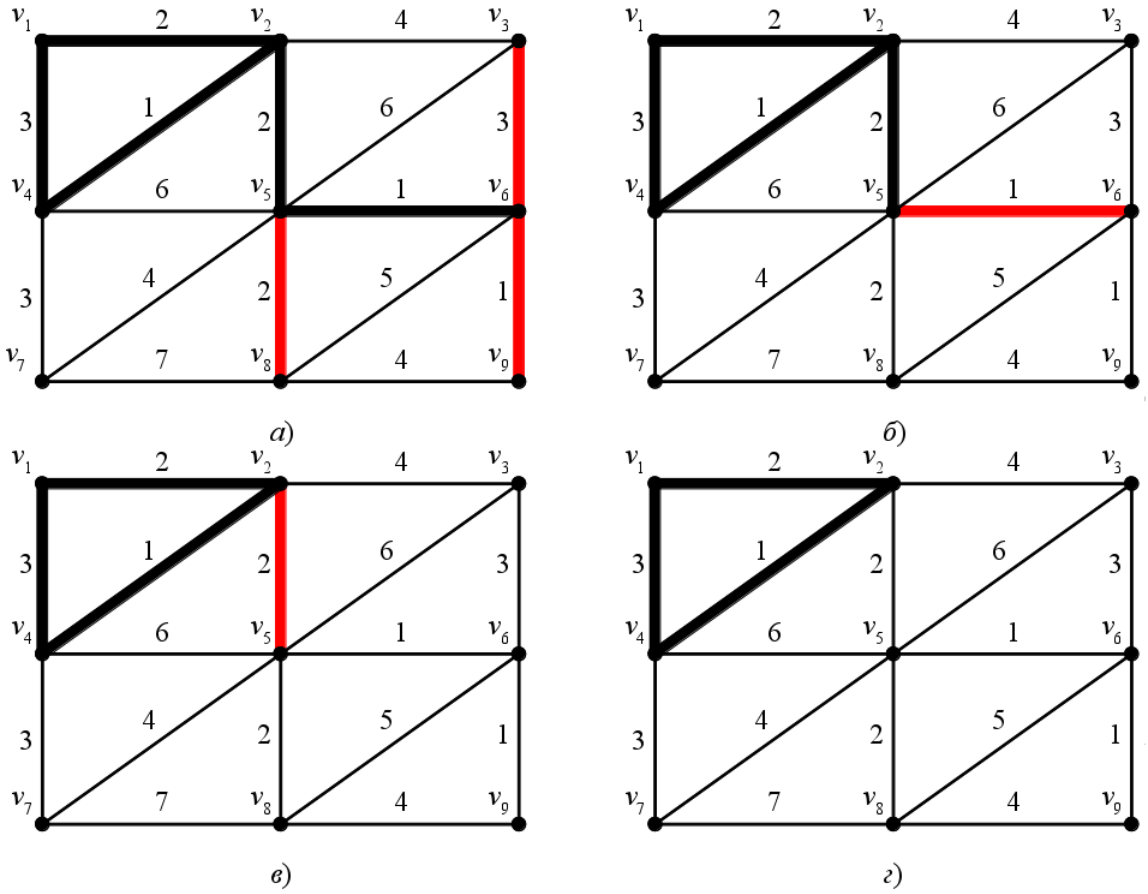


Рис. 5.8: Перевірка на відсутність циклів (є цикли)

стовпці або рядки матриці суміжності вершин.

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, і таке ребро не можна включати до того МОД, що будується.

Ось як виглядає алгоритм перевірки на наявність циклів для множини ребер з мал. 5.5ж. На мал. 5.7а показані ці ребра. Висячі вершини: v_1, v_3, v_7, v_8, v_9. Інцидентні до них ребра відмічені на мал. 5.7а червоним кольором. Вилучаємо їх (мал. 5.7б). Перевіряємо знову: тут висячі вершини: v_4, v_6, інцидентні до них ребра теж позначені червоним кольором. Після їхнього вилучення маємо мал. 5.7в, а ще після одного кроку — пустий граф на мал. 5.7г. Ребер немає — дійсно мали остовне дерево, циклів не було.

Якщо б на останньому етапі алгоритма Краскала (мал. 5.5є) ми б додали замість ребра e_{47} ребро e_{14}, то перевірка на наявність цикла показана на мал. 5.8.

Кроки вилучення ребер, інцидентних до висячих вершин, такі ж самі. На останньому етапі вершин ступеня 1 немає, але є три вершини ступеня 2, що свідчить про наявність цикла. Значить, ребро e_{14} не можна включати до МОД, що будується.

Розділ 6

Цикли та коцикли

У цій лекції буде 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.

6.1 Ейлерові та гамільтонові цикли

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 (мал.6.1) 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 6.1. Цикл у зв'язному графі чи мільтиграфі, 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 цикла. Але кратні ребра суттєво впливають на ейлеровість. Додавання чи вилучення паралельного ребра може надати мультиграфу ейлеровості або, навпаки, позбавити її. Тому будемо розв'язувати задачу про ейлерів цикл для мультиграфів, але не для псевдографів.

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

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

Означення 6.2. *Шляхом* у зв'язному графі чи мільтиграфі, що проходить один і тільки один раз через кожне ребро, називається *ейлеровим* (Eulerian path). Граф (мультиграф), у якому існує ейлерів шлях, 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, є.

Теорема 6.1. Необхідна ознака ейлеровості. Якщо зв'язний 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, у т. ч. й через v_i. Підемо з v_i уздовд

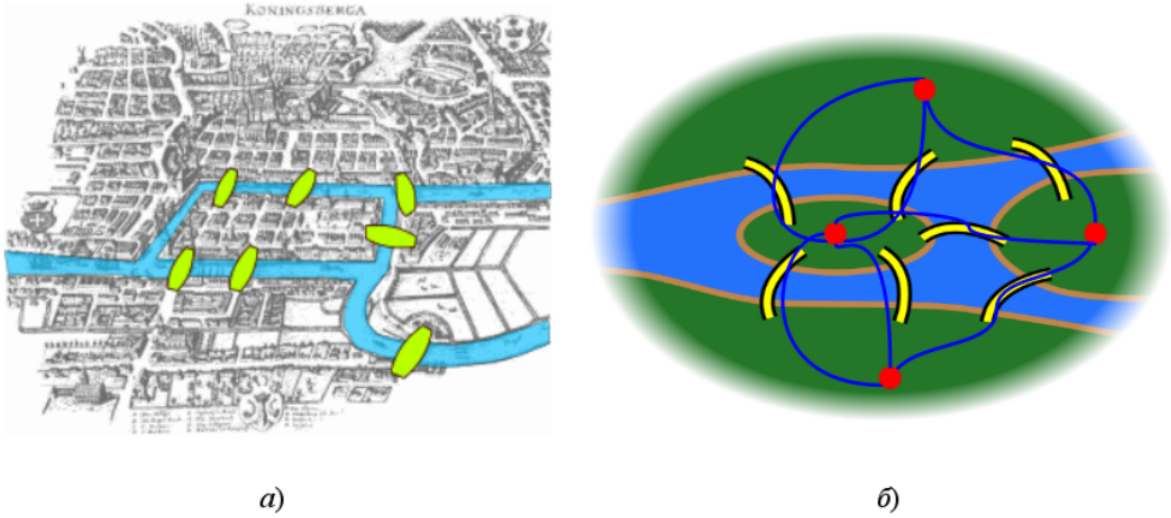


Рис. 6.1: Мости Кьоніґсберга

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 мал. 6.1б ступені вершин є 3, 3, 3, 5 — усі непарні. Значить, побудувати ейлерів цикл черз сім мостів Кьоніґсберга неможливо. А якщо б були парні?

Теорема 6.2. 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. Т. я. 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 $e_1, \ldots, e_k$, що проходить через вершини $v_0, \ldots, v_k$. 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, по яких ми входили та виходили, різні. Т. ч., у побудованому шляху рівно $2l-1$ ребер, інцидентних до v_k. Оскільки ступінь вершини v_k парна, існує інцидентне до v_k ребро, що не входить у побудований шлях, що суперечить нашій побудові. Отже, $v_k = v_0$.

Позначимо цикл, утворений ребрами $e_1, \ldots, e_k$, через C_1 (мал. 6.2). Якщо він ейлерів, побудову закінчено. Нехай C_1 не ейлерів, тобто в мультиграфі G_1, отриманому з мультиграфа G вилученням ребер $\{e_1, \ldots, e_k\}$, 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 $e_1, \ldots, e_k$ ступінь кожної з вершин $v_0, \ldots, v_k$ 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 $f_1, \ldots, f_m$ (мал. 6.3), у

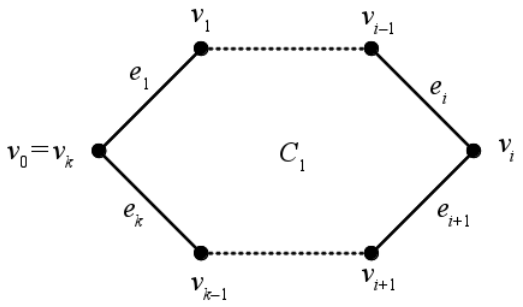


Рис. 6.2: До доведення теореми 6.2

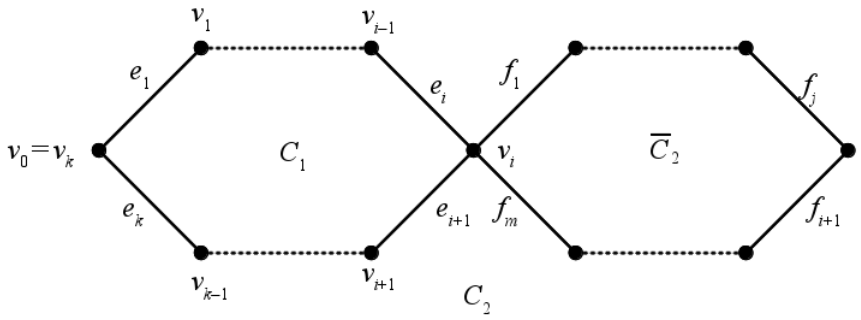


Рис. 6.3: До доведення теореми 6.2

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

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

1. Будуємо матрицю суміжності вершин $\boldsymbol{B}$ розміром $n \times n$.
2. Додаємо її рядки (або стовпці) — отримуємо вектор ступенів вершин.
3. Перевіряємо: якщо всі координати цього вектора парні, маємо ейлерів мультиграф; якщо дві координати непарні, а решта парні — маємо напівейлерів мультиграф; у 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.

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.

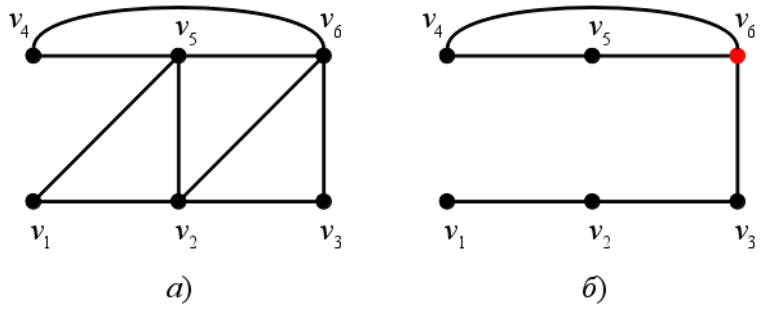


Рис. 6.4: Ейлерів граф (а) та побудова ейлерового циклу (б)

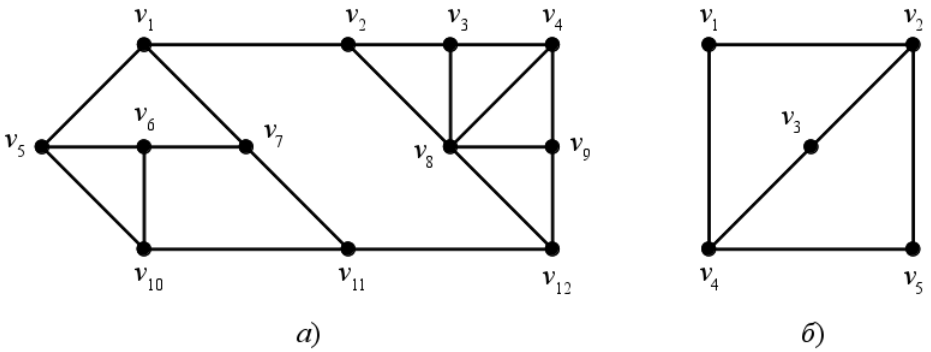


Рис. 6.5: Гамільтонів (а) та напівгамільтонів (б) граф

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 у ньому.

На прикладі мал. 6.4 розглянемо, як працює алгоритм Фльорі. Граф на мал. 6.4а ейлерів: усі вершини мають парний ступінь. Припустимо, на першому кроці ми обрали вершину v_1. Ступінь цієї вершини $2 > 1$. Вилучення як ребра e_{15}, так і ребра e_{12} не збільшує кількість компонент зв'язності, тому на кроці 2 обмеження ніяк не позначається; нехай обрано, наприклад, ребро e_{15}. На двох наступних ітераціях обмежень на вибір теж не виникає; нехай обрані ребра e_{52} та e_{26}. Тоді поточним графом стає граф, зображений на мал. 6.4б (поточна вершина — v_6). На наступній ітерації не можна обрати ребро e_{63} через обмеження: ступінь вершини v_6 дорівнює 3, а вилучення ребра e_{63} збільшує кількість компонент зв'язності. Тому обираємо якесь інше ребро, наприклад, e_{65}. Подальший вибір ребер тепер однозначний (поточна вершина завжди буде мати ступінь 1), так що в підсумку буде побудований такий ейлерів цикл: $e_{15} \to e_{52} \to e_{26} \to e_{65} \to e_{54} \to e_{46} \to e_{63} \to e_{32} \to e_{21}$.

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

Означення 6.3. Цикл, що проходить через кожну вершину графа один і тільки один раз, називається *гамільтоновим* (Hamiltonian cycle). Граф називається *гамільтоновим* (Hamiltonian graph), якщо в ньому існує гамільтонів цикл. ♢

Означення 6.4. Шлях, що проходить через кожну вершину графа один і тільки один раз, називається *гамільтоновим* (Hamiltonian path). Граф називається *напівгамільтоновим* (semi-Hamiltonian graph), якщо в ньому існує гамільтонів шлях. ♢

Наприклад, граф на мал. 6.5 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 мал. 6.5б лише напівгамільтонів: його гамільтонів шлях $v_1 \to v_2 \to v_3 \to v_4 \to v_5$.

На відміну від ейлерових графів, задача визначення гамільтоновості графа та знаходження гамільтонова цикла 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 6.3 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 6.4 Дірака (Dirac). Нехай $G = (V, E)$ — простий зв'язний граф з $n > 2$. Якщо ступінь будь-якої вершини v є не меншою за $n/2$: $\deg v \geq n/2$, 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 несуміжних вершин $\deg v + \deg w = 4 < 6$. Але, вочевидь, цей граф гамільтонів: є гамільтонів цикл уздовж шести його вершин.

6.2 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 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 цикла. У попередній лекції 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 відновити можна. 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 мал. 6.6а — це підмножини множини 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 мал. 6.6бвг.

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 $C_1, C_2, \ldots$. У нашому прикладі у графа з мал. 6.6а всього 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 6.1. Множина ребер, що утворює простий цикл, не є пустою. ♢

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

Властивість 6.2. 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. ♢

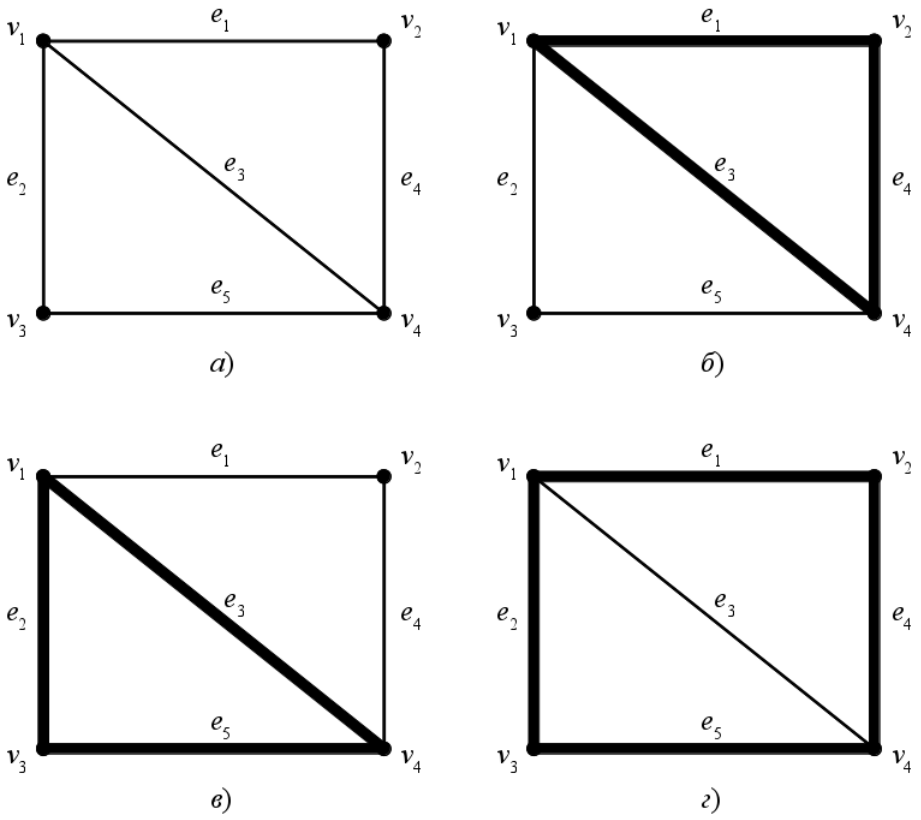


Рис. 6.6: Граф (а) та його прості цикли (б, в, г)

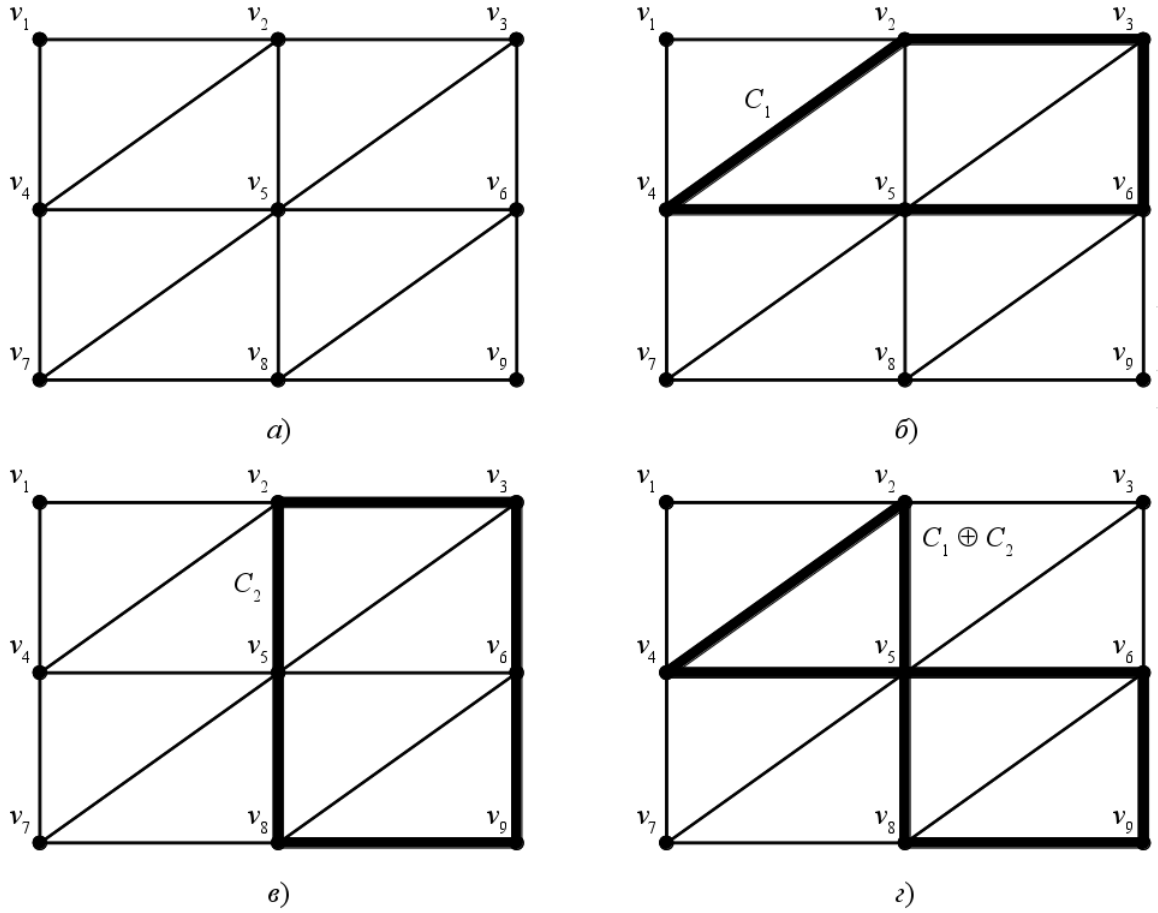


Рис. 6.7: Граф (а), його прості цикли (б, в) та їхня пряма сума (г)

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

Означення 6.5. *Прямою сумою* двох простих циклів C_i і C_j називається підмножина ребер $C_i \oplus C_j$, які входять 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 мал. 6.6. 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 мал. 6.7.

На мал. 6.7а показаний граф, а на мал. 6.7бв — два його прості цикли: $C_1 = \{e_{23}, e_{24}, e_{36}, e_{45}, e_{56}\}$ і $C_2 = \{e_{23}, e_{25}, e_{36}, e_{58}, e_{69}, e_{89}\}$. В результаті їхнього прямого додавання отримаємо підмножину ребер $\{e_{24}, e_{25}, e_{45}, e_{56}, e_{58}, e_{69}, e_{89}\}$, яка не є простим циклом. Це видно й на мал. 6.7г, і з перевірки властивості 6.2. У цієї множини є власні підмножини $\{e_{24}, e_{25}, e_{45}\}$ та $\{e_{56}, e_{58}, e_{69}, e_{89}\}$, які є простими циклами. Оскільки властивість 6.2 не виконується, то отримана множина $C_1 \oplus C_2$ не є простим циклом.

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

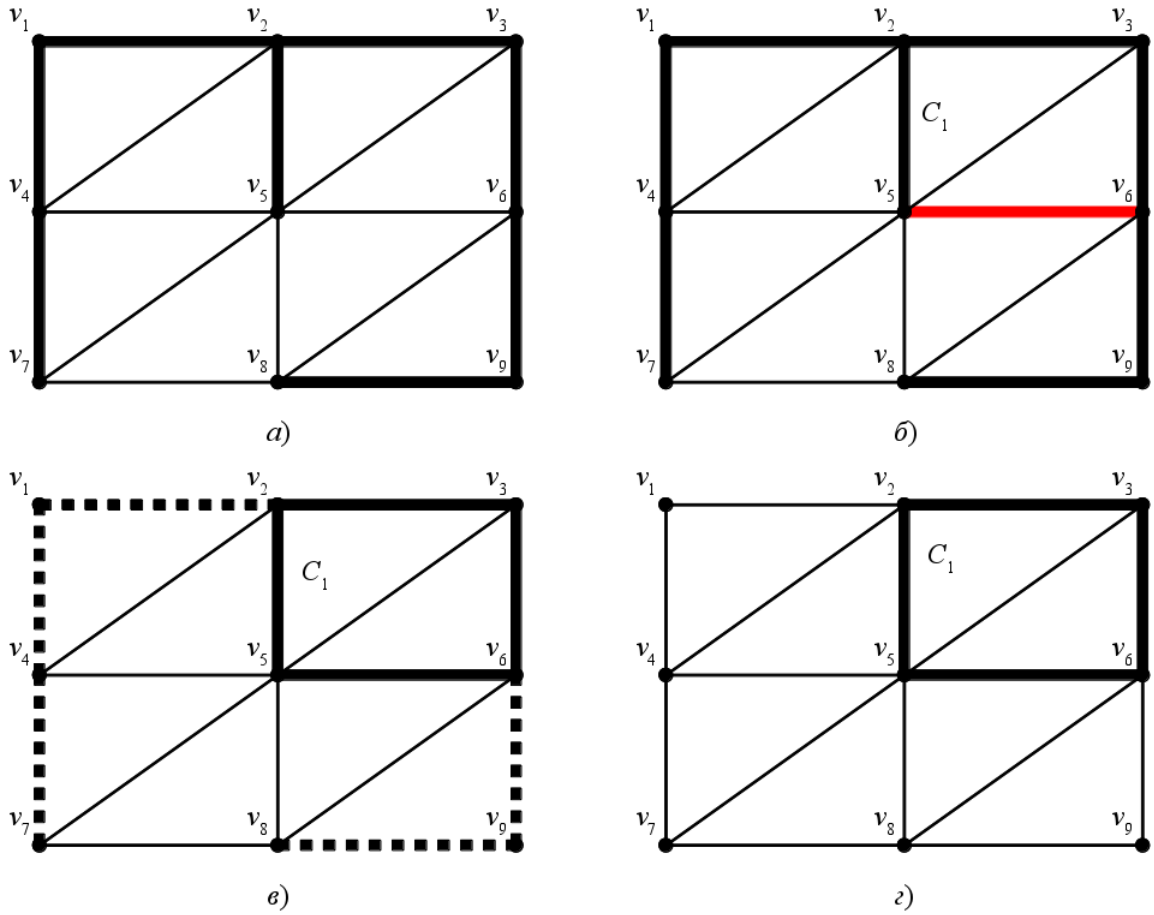


Рис. 6.8: Додавання до остовного дерева (а) ребра (б) утворює множину ребер (в), що містить простий цикл (г)

Властивість 6.3. Якщо C_i і 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 і 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 і C_j можуть повторюватися (не більш ніж двічі). При цьому ступені вершин додаються, і будуть або $0+0=0$, або $0+2=2$, 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 вибрати 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 I такого матроїда є всі можливі дерева та ліси, а базами — остовні дерева. З теорії остовних дерев відомо, що додавання до остовного дерева (мал. 6.8а) ще одного ребра (мал. 6.8б) утворює рівно один простий цикл. Крім цього цикла, в отриманій підмножині ребер, можливо, будуть і ребра, що не входять у цикл — такі собі "хвости" (мал. 6.8в). Після видалення цих "хвостів" отримаємо простий цикл (мал. 6.8г).

Цим діям відповідають такі означення теорії матроїдів.

Означення 6.6. *Залежна множина* матроїда (dependent set) — підмножина елементів носія S, що не є

незалежною. ♢

Означення 6.7. *Цикл* матроїда (circuit, cycle) — мінімальна за виключенням залежна множина матроїда. ♢

Залежна множина будується з незалежної додаванням нових елементів носія (ребер графа). 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 на попередній лекції, див. мал. 5.7 та 5.8.

Для циклів матроїда має місце теорема, яку ми вже довели: це властивості 6.1, 6.2 та 6.3.

Теорема 6.5. Множина C є множиною всіх простих циклів матроїда, якщо:

1. Жоден з елементів C не є пустою множиною.
2. Жоден з елементів C не є власною підмножиною іншого елемента C.
3. Якщо $C_1, C_2 \in C$ 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 попередній лекції, а й через залежні множини, мінімальні за виключенням, тобто через прості цикли.

Означення 6.8. Матроїд $M = (S, C)$ — це пара двох множин: S — носія матроїда, та C — сімейства непустих підмножин елементів S, для яких має місце теорема 6.5. ♢

Чи можна серед усіх простих циклів обрати якусь мінімальну їхню множину так, щоб усі інші прості цикли можна було б утворити за властивістю 6.3? Множину 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 6.9. *Базис у просторі циклів* (цикловий базис, фундаментальна система циклів, cycle basis, fundsmental cycles) — це мінімальна за кількістю елементів підмножина простих циклів, з яких можна будувати будь-який інший простий цикл за властивістю 6.3. Кількість циклів у цьому базисі називається *циклічним рангом* (cycle rank) графа. ♢

У звичайних лінійних просторах базисні елементи є лінійно-незалежними. Для перевірки цього факту треба побудувати їхню лінійну комбінацію та спробувати знайти 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 6.10. Прості цикли називаються *незалежними* (independent cycles), якщо вони відрізняються один від одного хоча б одним ребром. ♢

Сформулюємо кілька теорем про незалежні прості цикли.

Теорема 6.6. 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 6.3). ♢

Доведення. Візьмемо будь-який цикл C_0 з системи незалежних простих циклів. В 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 6.3, треба принаймні мати таку систему незалежних простих циклів, в якій є всі ребра з C_0.

Теорема 6.7 (зворотна). Якщо деяка система простих циклів не є незалежною, то принаймні один з них можна побудувати з інших за властивістю 6.3. ♢

Доведення. Згідно означення 6.10 у залежній системі циклів є хоча б один, кожне ребро якого є й якомусь іншому циклі. Позначимо його C_0. Об'єднуючи ребра інших циклів, отримаємо множину ребер, що містить C_0. ♢

Ця теорема стверджує таке. Якщо у нас є якийсь простой цикл C_0, всі ребра якого входять в інші прості цикли, то C_0 завжди можна побудувати з цих інших за властивістю 6.3. Дійсно, поглянемо на мал. 6.7. Тут є 2 простих цикли: $C_1 = \{e_{23}, e_{24}, e_{36}, e_{45}, e_{56}\}$ та $C_2 = \{e_{23}, e_{25}, e_{36}, e_{58}, e_{69}, e_{89}\}$. Розглянемо простой

цикл $C_0 = \{e_{23}, e_{25}, e_{36}, e_{56}\}$. Він є залежним з C_1 та C_2, т. я. всі його ребра є або в C_1, або в C_2. Чи можна побудувати його з C_1 та C_2 за властивістю 6.3? Теорема 6.7 стверджує, що так. Це можна зробити, наприклад, так. Спочатку будуємо $C_3 = \{e_{24}, e_{25}, e_{45}\} \subset C_1 \oplus C_2$, а потім знаходимо $C_0 = \{e_{23}, e_{25}, e_{36}, e_{56}\} = C_1 \oplus C_3$. На другому кроці нам навіть не потрібно було вибирати підмножину: пряма сумма C_1 та C_3 відразу дає потрібний нам простий цикл C_0.

Теорема 6.8. 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. У нього увійдуть $n-1$ ребер, і не війдуть 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 одно ребро, якого немає в інших: це те ребро, що ми додали до остовного дерева для утворення простого цикла. 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 простой цикл. Потім повторимо цю процедуру з іншим відкинутим ребром — отримаємо інший простий цикл, і т. д. — усього будемо мати $m-n+1$ незлежних простих циклів. Але чи достатньо їх? Чи можна сказати, що будь-який інший простий цикл у графі можна буде побудувати з них за властивістю 6.3? Відповідь на це питання дає наступна теорема.

Теорема 6.9. 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 6.8. ♢

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. Тоді у C_0 є щонайменше дві хорди (а може, й більше) та якась кількість ребер з остовного дерева (а може, й жодного нема). 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 $m - n + 1$ 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 6.3. Як кажуть, циклічний ранг зв'язного графа дорівнює кількості хорд будь-якого остовного дерева.

Розглянемо як приклад граф з мал. 6.7а. У нього $n = 9$ вершин та $m = 16$ ребер. Згідно з теоремами 6.8, 6.9 він має $m - n + 1 = 8$ незалежних простих циклів. Для такого простого графа ми й так їх бачимо на мал. 6.7а: це 8 малих трикутників: $C_1 = \{e_{12}, e_{14}, e_{24}\}$; $C_2 = \{e_{24}, e_{25}, e_{45}\}$; $C_2 = \{e_{23}, e_{25}, e_{35}\}$ і т. д. 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 мал. 6.8а, то отримаємо незалежні цикли, представлені на мал. 6.9. Червоним кольором показано те ребро, що додається до остовного дерева (хорда), а штриховими лініями — "хвости", що видаляються для отримання простого цикла. Отримана система простих циклів дійсно є незалежною: у кожному з них є ребро, яких немає в інших циклах, воно намальоване червоним кольором. Оскільки знайдених циклів $m - n + 1 = 8$, то вони утворюють базис: більше незалежних простих циклів побудувати не можна.

Це як і у звичайному лінійному просторі L_n: базис може бути не тільки натуральним $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$, 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.

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 (J. D. Horton) у 1987 р.

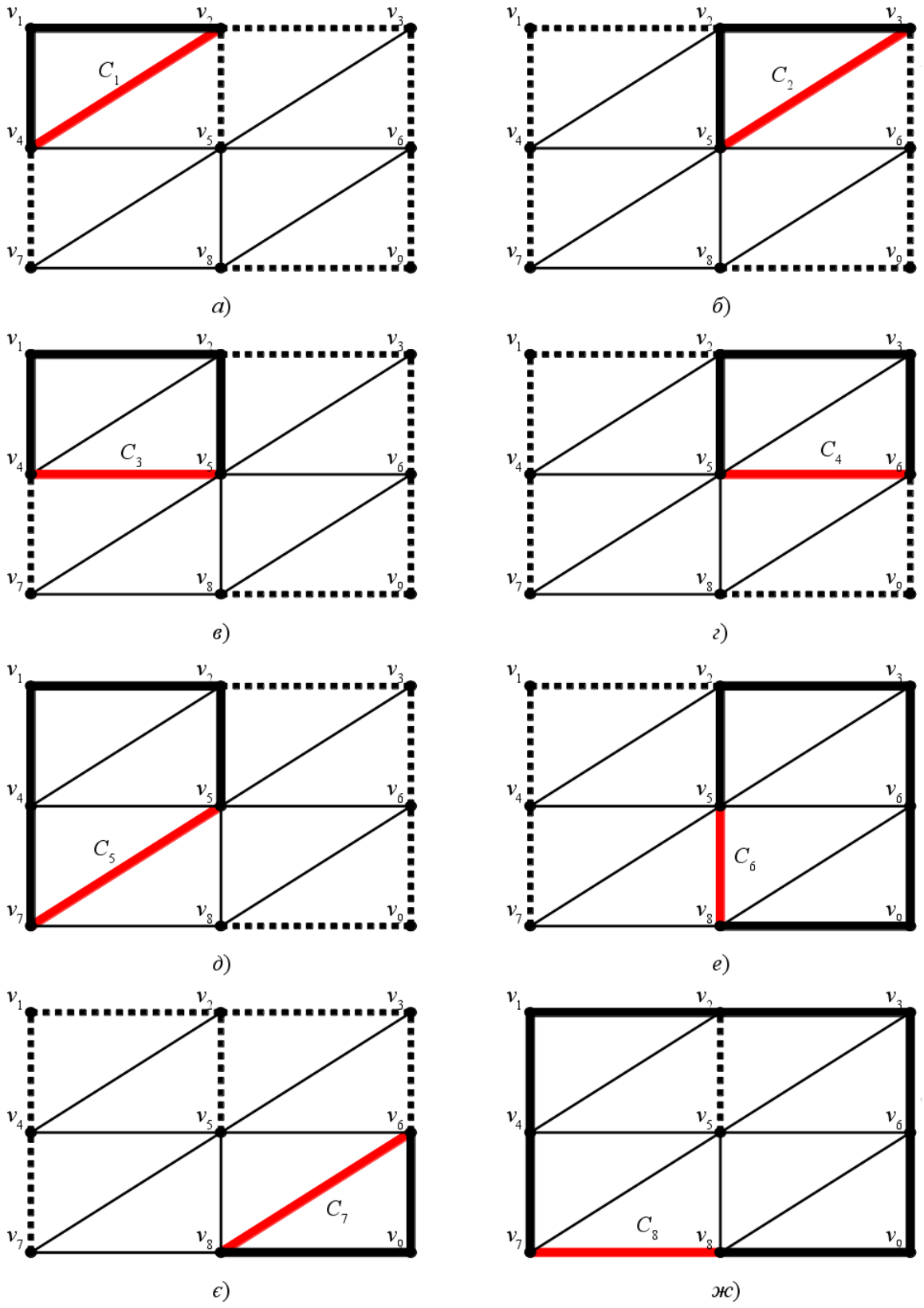


Рис. 6.9: Система незалежних простих циклів

6.3 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, видаливши єдине ребро, що поєднує її з графом. 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 цих вершин ребрами стала однією компонентою зв'язності, а решта вершин V_2 разом з інцидентними лише до цих вершин ребрами іншою? Ця задача теж досить просто розв'язується. Нехай задане розбиття множини вершин на дві підмножини: $V = V_1 \oplus V_2$. 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 6.11. *Розріз* (cut) — це множина ребер, видалення якої зі зв'язного графа робить його незв'язним. *Коцикл* (cocycle) — це розріз з мінімальною за виключенням 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 6.4. Множина ребер, що утворює коцикл, не є пустою (як і в циклі). ♢

Доведення є очевиднім: не видалили ребер — граф не роз'єднався. ♢

Властивість 6.5. 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 виключенням 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 6.1, 6.2. 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 і для циклів).

Означення 6.12. Прямою сумою двох коциклів K_i і K_j називається підмножина ребер $K_1 \oplus K_2$, які входять 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 графі з мал. 6.10а 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 на мал. 6.10бвг. Штриховою лінією показаний відповідний розріз на 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 мал. 6.11аб показані два коцикли: $K_1 = \{e_{12}, e_{24}, e_{25}, e_{35}, e_{56}, e_{68}, e_{89}\}$ та $K_2 = \{e_{23}, e_{35}, e_{47}, e_{56}, e_{57}, e_{58}\}$. Відповідні розрізи намальовані штриховими лініями. Але їхня пряма сума $K_1 \oplus K_2 = \{e_{12}, e_{23}, e_{24}, e_{25}, e_{47}, e_{57}, e_{58}\, e_{68}, e_{89}\}$, зображена на мал. 6.11в, не є коциклом: вона розділяє граф не на дві, а на три частини. Це краще видно на мал. 6.11г, 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 $\{e_{12}, e_{23}, e_{24}, e_{25}\}$, то отримаємо коцикл $\{e_{47}, e_{57}, e_{58}\, e_{68}, e_{89}\}$. Звідсіля — наступна властивість.

Властивість 6.6. Якщо K_i і K_j — коцикли, то існує коцикл $K_k \subseteq K_i \oplus K_j$. ♢

Доведення. Некай коцикл K_1 розрізає вершини графа на підмножини V_1 і V_2, а K_2 — на W_1 і W_2, як показано на мал. 6.12. 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) вилучаються, а залишаються лише ті ребра, що входять тільки в один з коциклів (червоні вертикальні та горизонтальні лінії). Т. ч., ребра, що входять до

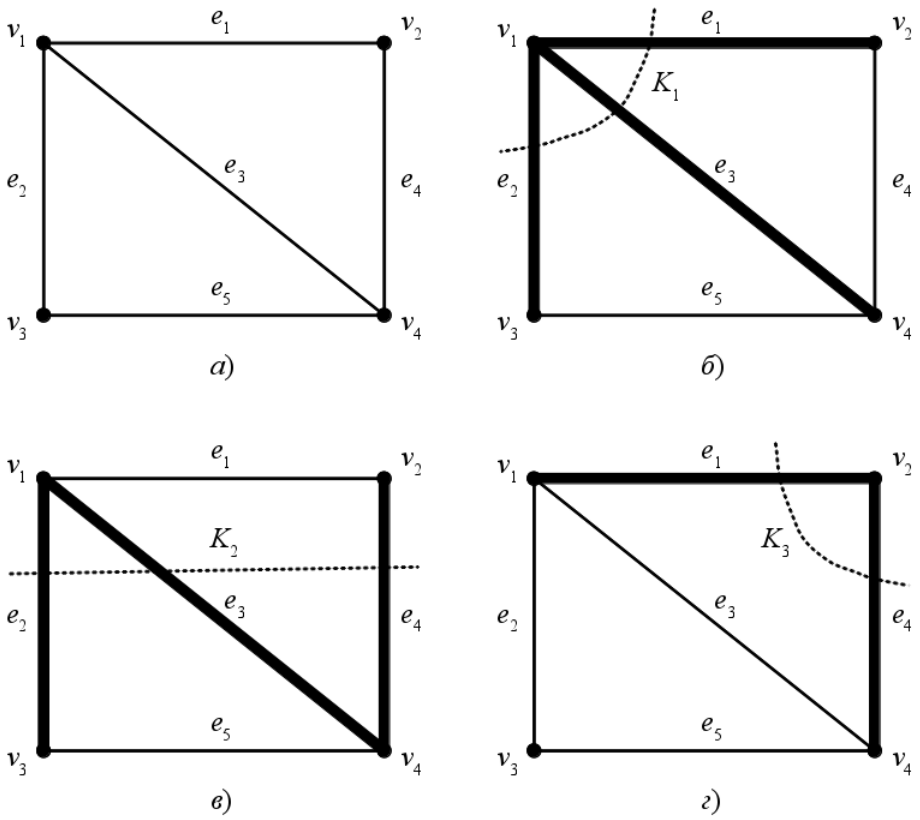


Рис. 6.10: Граф (а) та його коцикли (б, в, г)

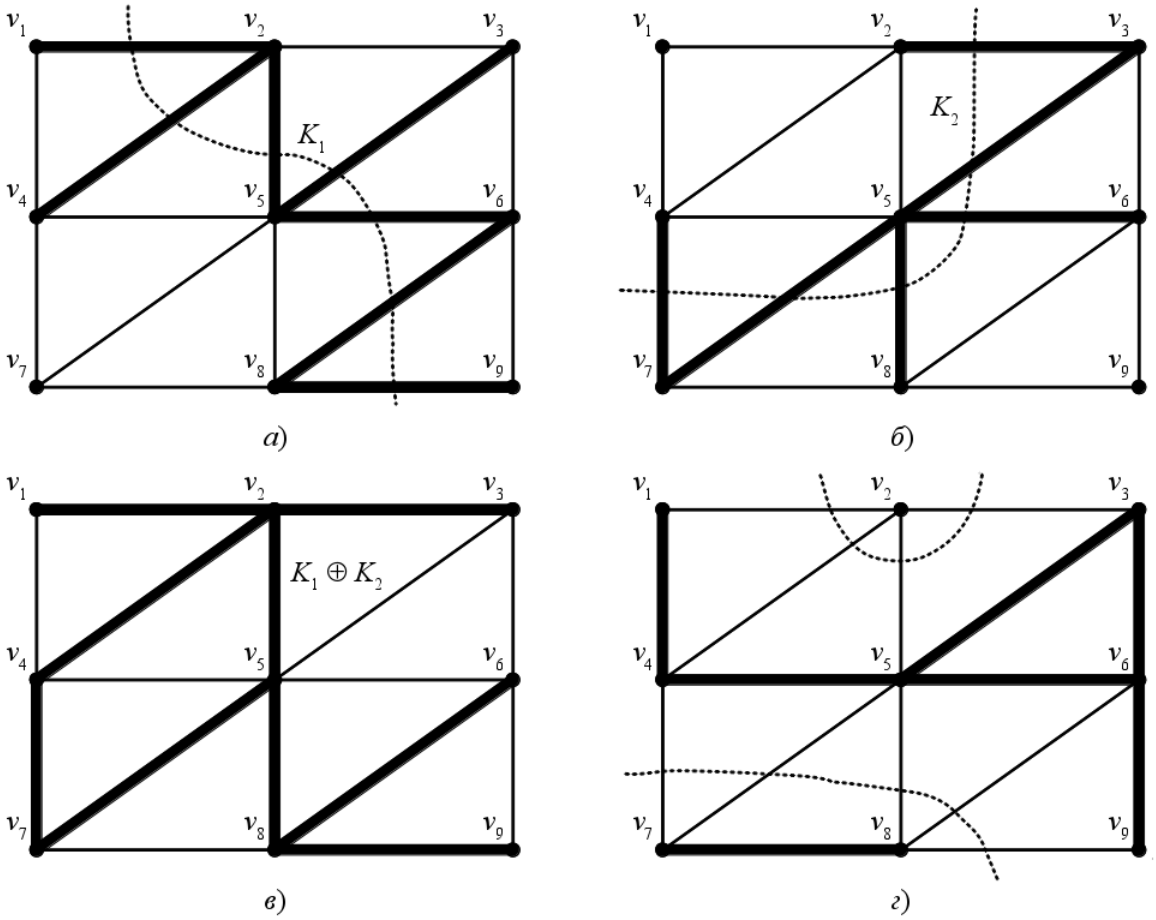


Рис. 6.11: Два коцикли (а, б), їхня пряма сума (в), що містить два коцикли (г)

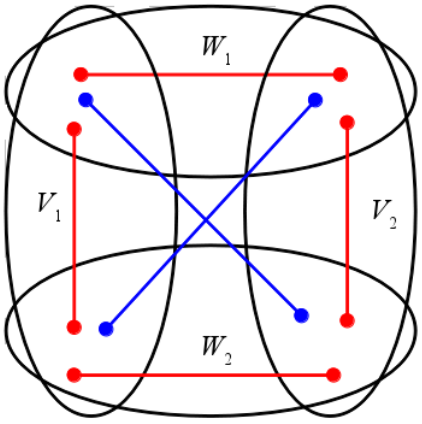


Рис. 6.12: До доведення властивості 6.6

$K_1 \oplus K_2$, є розрізами між підмножинами вершин:

$$\begin{array}{l} \{V_1 \cap W_1\} \leftrightarrow \{V_1 \cap W_2\}; \\ \{V_1 \cap W_1\} \leftrightarrow \{V_2 \cap W_1\}; \\ \{V_1 \cap W_2\} \leftrightarrow \{V_2 \cap W_2\}; \\ \{V_2 \cap W_1\} \leftrightarrow \{V_2 \cap W_2\}. \end{array} \tag{6.1}$$

При цьому ті підмножини вершин, що розташовані на мал. 6.12 вздовж діагоналі, можуть залишитися нерозрізаними, деякі можуть виявитися пустими, але у будь-якому випадку ребрами $K_1 \oplus K_2$ граф розрізається на 2 або 3 або 4 частини. А з цих ребер завжди можна вибрати такі, що розрізають граф лише на 2 частини. ♢

Наприклад, на мал. 6.11 маємо: $V_1 = \{v_1, v_4, v_5, v_7, v_8\}$; $V_2 = \{v_2, v_3, v_6, v_9\}$; $W_1 = \{v_1, v_2, v_4, v_5\}$; $W_2 = \{v_3, v_6, v_7, v_8, v_9\}$. Підмножини вершин, що розрізаються ребрами $K_1 \oplus K_2$: $\{V_1, W_1\} = \{v_1, v_4, v_5\}$; $\{V_1, W_2\} = \{v_7, v_8\}$; $\{V_2, W_1\} = \{v_2\}$; $\{V_2, W_2\} = \{v_3, v_6, v_9\}$. Підмножини $\{V_1, W_1\}$ та $\{V_2, W_2\}$ об'єдналися (сині лінії на мал. 6.12 вилучені), а між $\{V_1, W_2\}$ та $\{V_2, W_1\}$ взагалі не було ребер: вони залишилися роз'єднаними.

Інший приклад — мал. 6.10. 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 $K_1 \oplus K_2$: $\{V_1, W_1\} = \{v_1\}$; $\{V_1, W_2\} = \{\emptyset\}$; $\{V_2, W_1\} = \{v_2\}$; $\{V_2, W_2\} = \{v_3, v_4\}$. Підмножини $\{V_1, W_1\}$ та $\{V_2, W_2\}$ об'єдналися, у $\{V_1, W_2\}$ немає вершин — залишилися дві частини.

Коцикли, як і цикли, є залежними множинами матроїда, який так і називається: матроїд коциклів. Для коциклів має місце теорема 6.5 (тобто властивості 6.4, 6.5, 6.6). А, значить, і означення 6.8.

Переходимо тепер до визначення базису у комбінаторному просторі коциклів, тобто такої мінімальної кількості коциклів, з яких можна будувати всі інші за властивістю 6.6. Матроїдна структура нам у цьому допоможе.

Означення 6.13. *Базис у просторі коциклів* (кocикловий базис, фундаментальна система коциклів, cocycle basis, fundsmental cocycles) — це мінімальна за кількістю елементів підмножина коциклів, з яких можна будувати будь-який інший коцикл за властивістю 6.6. Кількість коциклів у цьому базисі називається *коциклічним рангом* (cocycle rank) графа. ♢

Означення 6.14. Коцикли називаються *незалежними* (independent cocycles), якщо вони відрізняються один від одного хоча б одним ребром. ♢

Наприклад, на мал. 6.10 коцикли K_1, K_2 та K_2 є залежними: кожен з них є прямою сумою інших двох. 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 6.10. У системі незалежних коциклів жоден з них не може бути побудований з інших шляхом додавання за логікою XOR та вибірки підмножини (тобто за властивістю 6.6). ♢

Доведення таке ж, як і для теореми 6.6. ♢

Теорема 6.11 (зворотна). Якщо деяка система коциклів не є незалежною, то принаймні один з них можна побудувати з інших за властивістю 6.6. ♢

Доведення таке ж, як і для теореми 6.7. ♢

Теорема 6.12. У зв'язному графі $G = (V, E)$ з $|V| = n$, $|E| = m$ існує принаймні $n - 1$ незалежних коциклів. ♢

Доведення. Як і для циклів, доведення є конструктивним, тобто дає алгоритм побудови системи з $n - 1$ незалежних коциклів. Побудуємо будь-яке остовне дерево графа G. У 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 одно ребро, якого немає в інших: це те ребро, що ми вилучили з остовного дерева для 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 6.6? Як і для циклів, можна стверджевати, що так. Про це каже остання теорема в цій главі.

Теорема 6.13. 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 6.12. $\diamondsuit$

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. Тоді у K_0 є щонайменше два ребра остовного дерева (а може, й більше) та якась кількість хорд (а може, й жодної нема). 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. $\diamondsuit$

За цією теоремою ми можемо стверджувати, цо $n-1$ незалежних коциклів дійсно утворюють базис у просторі коциклів: будь-який інший коцикл будується з них шляхом прямого додавання з наступною вибіркою згідно властивості 6.6. Як кажуть, коциклічний ранг зв'язного графа дорівнює кількості ребер будь-якого остовного дерева.

Розглянемо як приклад граф з мал. 6.7а. У нього $n = 9$ вершин та $m = 16$ ребер. Згідно з теоремами 6.12, 6.13 він має $n - 1 = 8$ незалежних коциклів. Якщо взяти остовне дерево з мал. 6.8а, то отримаємо незалежні коцикли, представлені на мал. 6.13. Червоним кольором показано те ребро, що видаляється з остовного дерева та розділяє граф на дві компоненти зв'язності. У синій колір пофарбовані ребра, що разом з видаленим ребром остовного дерева (червоним) 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, яких немає в інших коциклах, воно намальоване червоним кольором. Оскільки знайдених коциклів $n-1 = 8$, то вони утворюють базис: більше незалежних коциклів побудувати не можна.

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

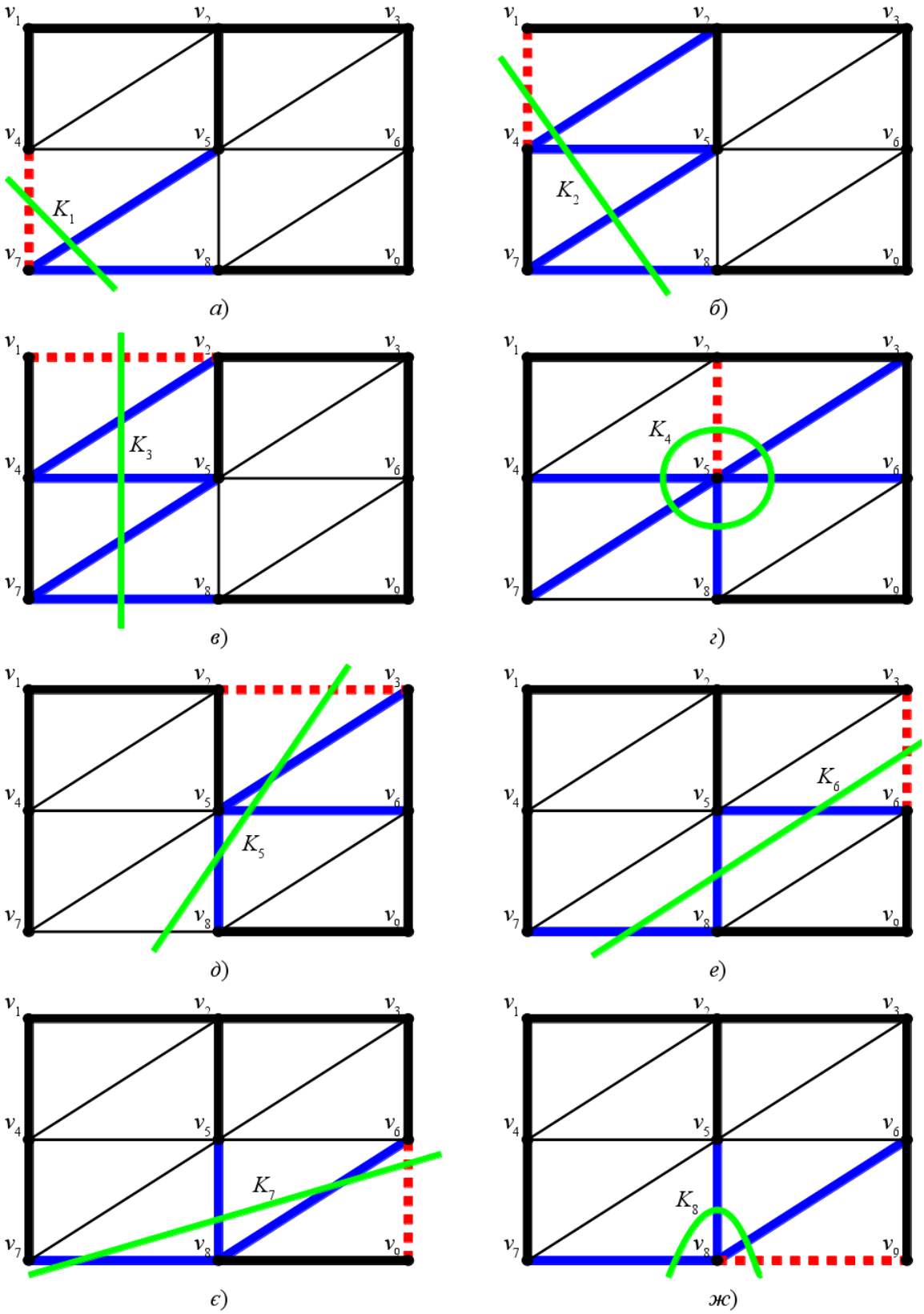


Рис. 6.13: Система незалежних коциклів

Розділ 7

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.

7.1 Бінарні відношення

Означення 7.1. *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 $b \in B$:

$$A \times B = \{(a, b) : \ (a \in A) \cap (b \in B)\} \, . \diamondsuit \tag{7.1}$$

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 цілих чисел є добре відома зі школи таблиця множення.

Означення 7.2. *Бінарним відношенням* (binary relation) називається підмножина декартового добутку якоїсь множини A саму на себе. $\diamondsuit$

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 можно 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 символ $\heartsuit$, щоб не прив'язуватися до жодного конкретного бінарного віношення. Елементи множини A позначені a або a_1, a_2, a_3, ...

Означення 7.3. Бінарне відношення $\heartsuit \subseteq A \times A$ на A називається:

- *рефлексивным* (reflexive), якщо $a \heartsuit a$ має місце $\forall a \in A$;
- *транзитивним* (transitive), якщо $\forall a_1, a_2, a_3 \in A$ з виконання $a_1 \heartsuit a_2$, $a_2 \heartsuit a_3$ випливає, що $a_1 \heartsuit a_3$;
- *повним* (total), якщо $(a_1, a_2) \in A \times A$ обов'язково виконується або $a_1 \heartsuit a_2$, або $a_2 \heartsuit a_1$;
- *антисиметричним* (antisymmetric), якщо з одночасного виконання $a_1 \heartsuit a_2$, $a_2 \heartsuit a_1$ слідує, що $a_1 = a_2$ (елементи еквівалентні в сенсі деякого означення, яке ще треба ввести до розгляду);

- *асиметричним* (asymmetric), якщо з виконання $a_1 \heartsuit a_2$ слідує, що $a_2 \heartsuit a_1$ не виконується ніколи;
- *симетричним* (symmetric), якщо з виконання $a_1 \heartsuit a_2$ слідує, що обов'язково виконується $a_2 \heartsuit a_1$;
- *квазіупорядкованим* (preorder, quasiorder), якщо воно рефлексивне та транзитивне;
- *еквівалентним* (equivalence), якщо воно рефлексивне, транзитивне та симетричне;
- *передупорядкованим*, якщо воно рефлексивне, транзитивне та повне;
- *частково упорядкованим* (partial order), якщо воно рефлексивне, транзитивне та антисиметричне;
- *повністю упорядкованим* (total order), якщо воно рефлексивне, транзитивне, повне та антисиметричне. ♢

7.2 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, значить, 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 і з v_1 (вже не за один, а за два переходи вздовж дуг). Значить, бінарне відношення $\to$ є транзитивним.

Повнота. Не обов'язково! Наприклад, на мал. 7.1а $v_1 \to v_2$, але $(v_2, v_1) \notin \to$.

Антисиметричність. Для перевірки цієї вимоги треба ввести поняття еквівалентних вершин.

Означення 7.4. Вершини орграфа v_i та v_j називаються *сильно зв'язаними* (strongly connected), якщо існують шляхи вздовж дуг із v_i до v_j та із v_j до v_i. ♢

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, т. я. можуть існувати орграфи, у яких $v_i \to v_j$, але $(v_j, v_i) \notin \to$. Приклад такого орграфа — на мал. 7.1а.

Т. ч., множина вершин орграфа з введеним на ній бінарним відношенням $\to$ (досяжність) та поняттям еквівалентності (взаємна досяжність, сильна зв'язаність) є частково упорядкованою, т. я. 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 7.5. *Компонентою сильної зв'язності* орграфа (strongly connected component) називається підмножина вершин, що утворює клас еквівалентності за досяжністю. ♢

На мал. 7.1а показаний як приклад орграф с 5 вершинами. Вони разбиваются на 4 компоненти сильної зв'язності, які обведені замкненими штриховими лініями. На мал. 7.1б 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 малюнку місцями, і при цьому упорядкування збереглося б.

Означення 7.6. *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 7.7. 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

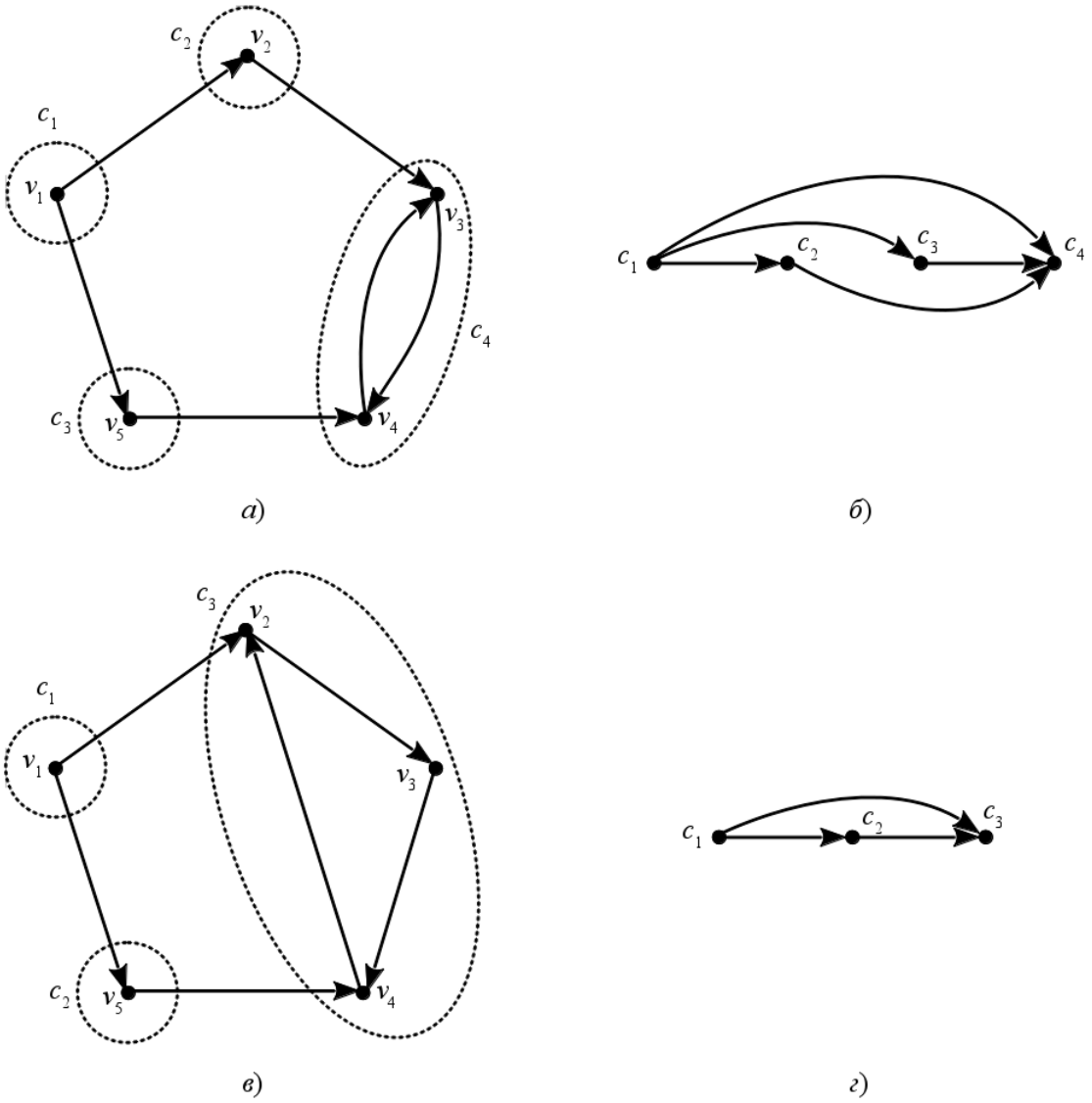


Рис. 7.1: Сильно зв'язані компоненти орграфа

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 мал. 7.1в.

Означення 7.8. *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. $\diamondsuit$

Матриця досяжності відрізняється від матриці суміжності вершин орграфа тільки відсутністю -1. Цю матрицю можна задавати також і для орграфа компонент сильної зв'язності. В цьому випадку вона буде мати розміри $N \times N$, де N — кількість компонент ($N \le n$). 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 мал. 7.1б ця матриця має вигляд:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.2}$$

Тут на головній діагоналі проставлені одиниці, щоб підкреслити рефлексивність бінарного відношення $\to$. Якщо ми тепер замінимо дугу e_{43} на e_{42} (мал. 7.1в), то замість 4 компонент сильної зв'язності отримаємо тільки 3. Їхнє упорядкування з часткового перетворюється вже на повне (мал. 7.1г), а його матриця досяжності буде мати вигляд:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}. \tag{7.3}$$

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, і т. і. Але ми розглянемо лише два перші етапи: розбиття множини вершин на сильно зв'язані компоненти та часткове упорядкування цих компонент.

7.3 Визначення сильно зв'язаних компонент

Нехай ми побудували матрицю досяжності орграфа $\boldsymbol{R}$ за означенням 7.8. 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, дуги з v_i у 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 $(r_{ik} = \text{true}) \cap (r_{kj} = \text{true})$. Щоб це перевірити, треба взяти всі елементи i-го рядка матриці $\boldsymbol{R}$ та перемножити їх (за булівськими правилами) на відповідні елементи j-го стовпця цієї ж матриці, а потім скласти отримані булівські змінні. Якщо хоча б при одному якомусь k було $(r_{ik} = \text{true}) \cap (r_{kj} = \text{true})$, то в результаті отримаємо true. А якщо немає жодної вершини v_k, для якої виконується $(r_{ik} = \text{true}) \cap (r_{kj} = \text{true})$, 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 стовпець з наступним додаванням — це просто множення матриць!

Т. ч., 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 ж вершину, а другий у потрібну (або навпаки).

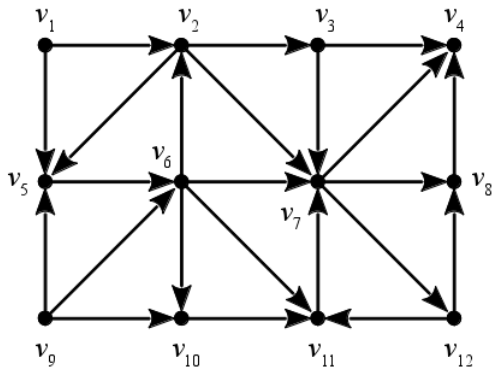


Рис. 7.2: Приклад орграфа

Значить, $\boldsymbol{R}^2$ є матрицею досяжності орграфа за 1 або 2 кроки. За індукцією: $\boldsymbol{R}^3$ є матрицею досяжності за 1, 2 або 3 кроки, $\boldsymbol{R}^4$ — за 1, 2, 3 або 4 кроки, і т. д. 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 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 мал.7.2. У нього $n = 12$; $m = 23$. Його матриця досяжності:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{7.4}$$

Обчислюємо $\boldsymbol{R}^2$:

$$\boldsymbol{R}^2 = \boldsymbol{R} \cdot \boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{7.5}$$

Оскільки $\boldsymbol{R}^2 \neq \boldsymbol{R}$, продовжуємо підводити до квадрату:

$$\boldsymbol{R}^4 = \boldsymbol{R}^2 \cdot \boldsymbol{R}^2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{7.6}$$

Знову маємо $\boldsymbol{R}^4 \neq \boldsymbol{R}^2$. Ще раз підводимо до квадрату:

$$\boldsymbol{R}^8 = \boldsymbol{R}^4 \cdot \boldsymbol{R}^4 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{7.7}$$

Тут вже є співпадіння: $\boldsymbol{R}^8 = \boldsymbol{R}^4$. Далі множити матрицю саму на себе немає сенсу: результат буде таким самим. Т. ч., ми отримали матрицю досяжності за будь-яку кількість кроків, тобто матрицю транзитивної досяжності. Позначимо її буквою $\boldsymbol{D}$:

$$\boldsymbol{D} = \boldsymbol{R}^8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{7.8}$$

У цій матриці $d_{ij} = d_{ji} = \text{true}$ 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, які 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 мал. 7.3а.

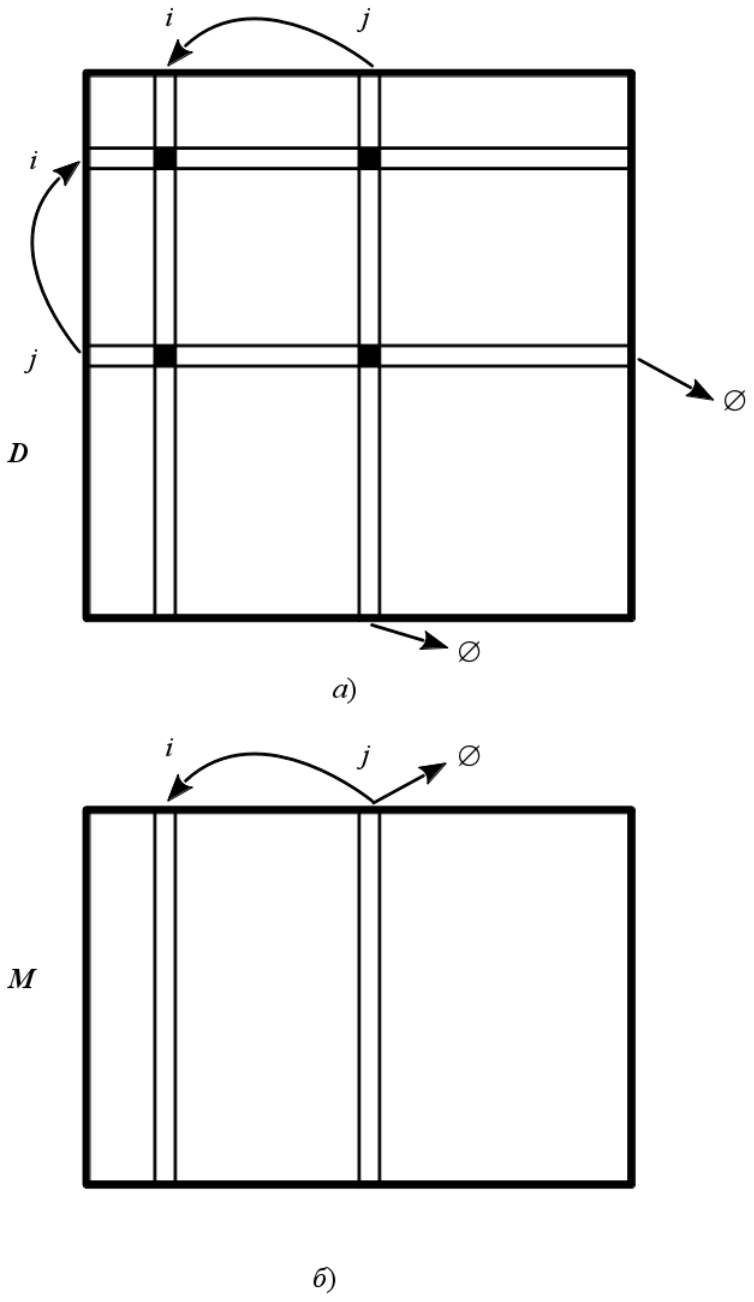


Рис. 7.3: Об'єднання вершин сильно зв'язаної компоненти

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 стовпця матриці $\boldsymbol{D}$ її j-й стовпець, і видалити j-й стовпець;
- додати до i-го рядка матриці $\boldsymbol{D}$ її j-й рядок, і видалити j-й рядок.

Після такої операції матриця досяжності зменшить свої розміри на 1. 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 $\boldsymbol{M}$ — діагональну булеву матрицю розміром $n \times n$. У кожному її 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 $\boldsymbol{M}$ — діагональна. Як тільки знаходиться пара $d_{ij} = d_{ji} = \text{true}$, то крім описаних вище двох дій, виконуємо ще й третю:

- додати до i-го стовпця матриці $\boldsymbol{M}$ її j-й стовпець, і видалити j-й стовпець.

В результаті цієї дії у i-му стовпці будуть збережені номери вершин, які об'єднуються в одну компоненту сильної зв'язності. Ця дія показана на мал. 7.3б.

Додавання та викидання стовпців та рядків можна реалізувати за допомогою множення на відповідну матрицю деформування-додавання-стискання (її інколи називають прокрустовою матрицею). Наприклад, у матриці $\boldsymbol{D}$ з (7.8): $d_{25} = d_{52} = \text{true}$. Прокрустова матриця $\boldsymbol{T}$ будується так. Береться одинична матриця, у ній до 2-го стовпчика додається 5-й, а потім 5-й стовпчик видаляється. Тобто вона буде мати розмір 12×11:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.9}$$

Тепер об'єднання v_2 та v_5 в одну компоненту сильної зв'язності (реалізація дій з мал. 7.3) виглядає так:

$$\boldsymbol{D}_1 = \boldsymbol{T}^T \boldsymbol{D} \boldsymbol{T} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}; \tag{7.10}$$

$$\boldsymbol{M}_1 = \boldsymbol{MT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.11}$$

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 знову $d_{25} = d_{52} = \text{true}$. Прокрустова матриця $\boldsymbol{T}_1$ тепер вже буде мати розмір 11×10. Вона утворюється з одиничної додаванням 2-го та 5-го стовпців та викиданням 5-го стовпця:

$$\boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.12}$$

Подальші дії цієї ітерації:

$$\boldsymbol{D}_2 = \boldsymbol{T}_1^T \boldsymbol{D}_1 \boldsymbol{T}_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}; \tag{7.13}$$

$$\boldsymbol{M}_2 = \boldsymbol{M}_1\boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.14}$$

Як бачимо, об'єдналися v_2, v_5 та v_6. Далі можна об'єднувати за $d_{59} = d_{95} = \text{true}$, і т. д. Після останнього кроку отримаємо такі матриці $\boldsymbol{D}$ та $\boldsymbol{M}$ (нижні індекси в них опускаємо):

$$\boldsymbol{D} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}; \tag{7.15}$$

$$\boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{7.16}$$

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}). Всі інші вершини залишилися на самоті: кожна з них утворює свою компоненту сильної зв'язності.

7.4 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 1-м, 2-м і т. д., поки не знайдемо його місце у послідовності. У найгіршому випадку будемо мати ті ж самі $\frac{n(n-1)}{2}$ операцій порівняння, але "в середньому" їх буде менше. В ідеальному випадку взагалі після 1-го ж порівняння новий елемент відразу стане на 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 із двох середніх, якщо їхня кількість парна). Тим самим ми визначаємо половину, в якій 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 нас ці алгоритми не будуть працювати, т. я. упорядкування у нас тільки часткове. Деякі сильно зв'язані компоненти взагалі не можна порівнювати. Але, як виявляється, нам взагалі не треба порівнювати між собою сильно зв'язані компоненти, т. я. 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 $\boldsymbol{M}$.

В 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 зручно скоритатися матрицею перестановок, як ми це робили при дослідженні ізоморфізму. Наприклад, у матриці $\boldsymbol{D}$ в формулі (7.15) є $d_{72} = \text{true}$. Треба поміняти місцями 2-й та 7-й стовпці та рядки у матриці $\boldsymbol{D}$, а також 2-й та 7-й стовпці у матриці $\boldsymbol{M}$. Створюємо матрицю перестановок $\boldsymbol{T}$. Беремо одиничну матрицю та міняємо в ній 2-й та 7-й стовпці:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.17}$$

Тепер за допомогою відомих формул лінійної алгебри перестявляємо рядки та стовпці, як потрібно:

$$\boldsymbol{D}_1 = \boldsymbol{T}^T \boldsymbol{D} \boldsymbol{T} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}; \tag{7.18}$$

$$\boldsymbol{M}_1 = \boldsymbol{MT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{7.19}$$

Перевіряємо далі. У матриці $\boldsymbol{D}_1$ є істиний елемент нижче головної діагоналі: $d_{73} = \text{true}$. Матриця перестановки 3-го та 7-го стовпців:

$$\boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.20}$$

Відповідна перестановка рядків та стовпців:

$$\boldsymbol{D}_2 = \boldsymbol{T}_1^T \boldsymbol{D}_1 \boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}; \tag{7.21}$$

$$\boldsymbol{M}_2 = \boldsymbol{M}_1 \boldsymbol{T}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{7.22}$$

Як бачимо, одиниці нижче головної діагоналі у матриці $\boldsymbol{D}$ поступово переповзають вгору-праворуч. Через кілька кроків ми отримаємо (нижні індекси опущені):

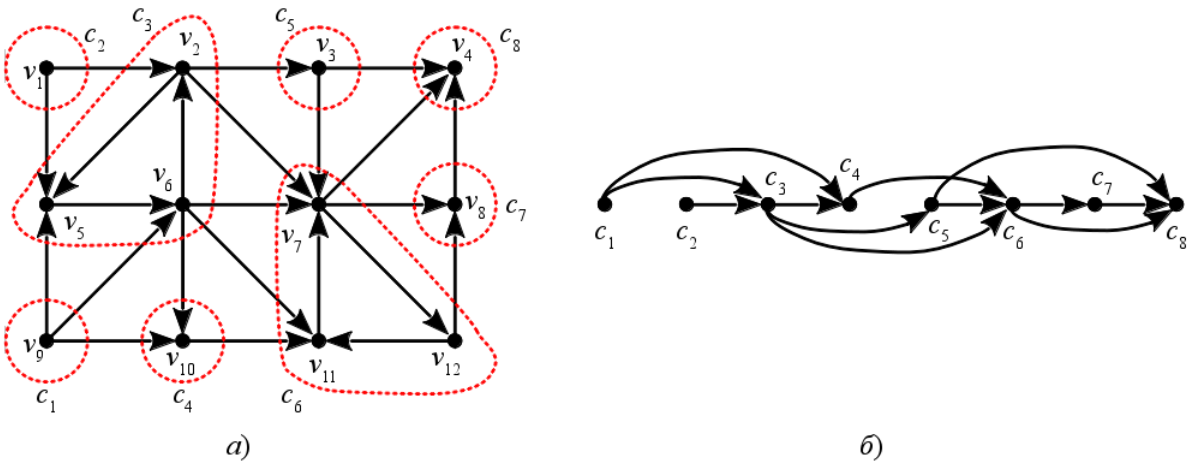


Рис. 7.4: Сильно зв'язані компоненти орграа (а) та їхнє часткове упорядкування (б)

$$\boldsymbol{D} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \tag{7.23}$$

$$\boldsymbol{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{7.24}$$

У матриці $\boldsymbol{M}$ (7.24) — 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 3-й — 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 $\boldsymbol{D}$ (7.23). Одиниця означає досяжність з якоїсь компоненти до якоїсь, а нуль — відсутність досяжності. Зокрема, $d_{12} = 0$ означає, що з 1-ї компоненти (v_9) 2-а компонента (v_1) не досягається. Так само з 4-ї компоненти (v_{10}) не досягається 5-а (v_3). В усіх інших випадках компонента з більшим номером завжди досягається з компоненти з меншим номером. На мал. 7.4а показаний наш орграф з мал. 7.2, на якому позначені компоненти сильної зв'язності, а на мал. 7.4б — їхнє часткове упорядкування.

Щоб не затіняти, на мал. 7.4б показані лише реальні дуги між компонентами, а не транзитивне упорядкування, яке записане у формулі (7.23). Тобто, наприклад, у (7.23) $d_{15} = 1$, що означає досяжність компоненти

c_5 із c_1. На мал. 7.4б 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.

Зауважимо, що цей 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 увійдусь до однієї компоненти сильної зв'язності. Т. ч., знайдені компоненти сильної зв'язності доповненого зворотними дугами орграфа будуть співпадати з компонентами зв'язності графа.