3. Теорія графів

3.1. Основні положення теорії графів

3.1.1. Історія виникнення теорії графів

Історія теорії графів почалася із задачі про кенігсберзькі мости, придуманої і розв’язаної Ейлером у 1736 році. Ейлер поставив собі задачу, якобійти всі чотири частини суші, пройшовши по кожному з мостів рівно одинраз, і повернутися у початкове місце. Мости, по яких ходив Ейлер, розташовувалися, як показано на рис. 3.1.

Рис. 3.1. Розташування кенігсберзьких мостів

Для розв’язування цієї задачі Ейлер позначив ділянки суші точками, а мости, що їх з’єднують, лініями, одержавши таким чином перше представлення графа.

Один з варіантів графа задачі про кенігсберзькі мости має такий вигляд

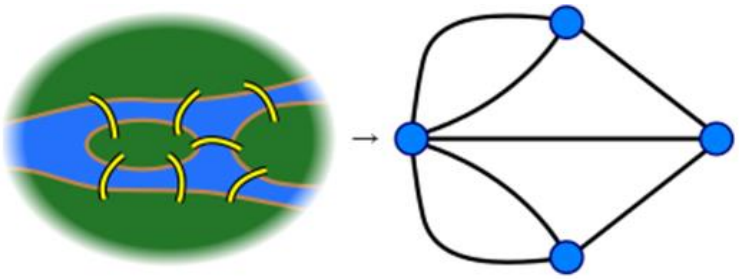


Рис. 3.2. Граф розташування кенігсберзьких мостів

Як видно з рис. 3.2, граф складається з точок та ліній. Точки називаютьвершинами. Лінії, що з’єднують вершини, називають ребрами.

Теорія графів у математиці займається вивченням особливого видуматематичних структур - графів, що використовуються для моделювання парнихвідношень між б'єктами. Графи у цьому контексті складаються з вершин (точок), які з'єднані ребрами (лініями).

Враховуючи, що набір вершин можна використовувати для абстрагування будь-якого типу комп'ютерних даних, теорія графів глибоко вивчає взаємозв'язок між ними та дає відповіді на ряд питань розташування, налаштування мережі, оптимізації, узгодження та ін.

В такому ключі структури даних і їх використання вперше розглянув відомий

математик Леонард Ейлер, який сформулював основні поняття теорії графів якрозділу математики.

Теорія графів відіграє критичну роль у багатьох проблемах інформатики. Зокрема, теорія графів використовується для моделювання парних взаємозв'язківміж об'єктами певної множини. Ряд комп'ютерних підходів було розроблено дляполегшення використання графів, наприклад як SPANTREE або GTP для передачіданих.

Тобто, графи - це метод візуальної ілюстрації даних та відношень між ними.

Мета графів - представити занадто численні, або складні дані, для їх адекватногоопису в тексті, або алгоритмі. Забезпечення чіткості та коректність опису даних уграфах забезпечують ефективність використання графів.

3.1.2. Основні визначення графів

Зображення графа містить точки та ребра. Однак, точки та ребра не є графом, це лише елементи зображення графа, як наприклад показано на рис.3.1.

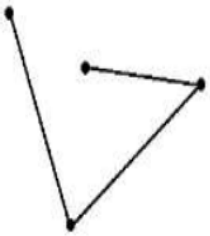


Рис. 3.1. Загальний вигляд графа

Щоб перейти до визначення графа розглянемо спочатку кілька прикладів.

Приклад 3.1. Розглянемо множину V студентів в аудиторії. Потужністьмножини |V| = 21. Якщо студенти сидять по двоє за партою і студентів позначимоточками, то студентів за однією партою можна позначити ребром. Маємозображення, що показано на рис. 3.2. Із зображення зрозуміло, що парти стоять в дваряди, чотирнадцять студентів сидять по двоє за партою, п’ять студентів сидять поодному.

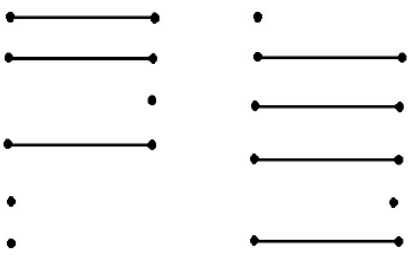


Рис.3.2. Розміщення студентів за партами

На рисунку 3.2. показано розміщення студентів за партами, відповідно до прикладу 3.1. Із рис.3.3 видно, що в аудиторії залишилося 7 студентів, один

студентсидить сам за партою, а ще шість студентів сидять попарно, проте партиіперекомбіновані, але це не значить що вони розміщені одна на одній.

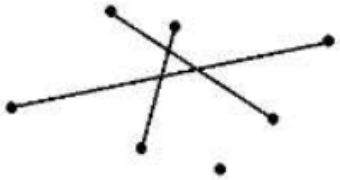


Рис. 3.3. Розміщення 7 студентів за партами

В обох випадках, описаних в прикладі 3.1 і показаних на рис. 3. 2-3.3 маємо справу
з графом. Тобто, використання графа дозволяє підвищити інформативністьпередачі матеріалу і його засвоєння. Можна сказати, що граф це не просто зображення на площині.

Граф – це абстрактний комбінаторний об'єкт, що складається з двох множин
V (vertex) – множина вершин і E (edge) – множина ребер. Саме тому в прикладі 3.1 кожне ребро має дві вершини – двох студентів, щосидять за однією партою.

Неорієнтований граф

Визначення. Неорієнтованим графом *G* (*V*, *E*) називають сукупність двох
множин:
– непустої множини *V*, яку називають множиною **вершин**;
– множини *E* неупорядкованих пар елементів множини *V*, яку називають множиною **ребер.** Множина ребер у графі може бути порожньою.

Умови існування неорієнтованого графа *G* (*V*, *E*) задають наступними правилами:
1. $V \neq \varnothing$. Множина вершин графа непорожня.
2. $E \subset V \times V$. Елементами множини ребер є відношення, елементамиякого є двійки з вершин графа. Кількість двійок є підмножиноюдекартового добутку $V \times V$.
3.$E = E^{-1}$. Множина ребер неорієнтованого графа – це симетричневідношення.
Приклад неорієнтованого графа показаний на рис. 3.4.

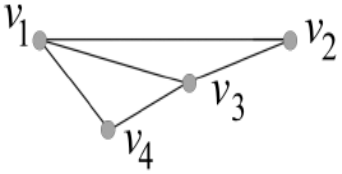


Рис. 3.4. Неорієнтований граф

Кількість вершин графа *G* позначимо через *p*, а кількість ребер графа *G* позначимо через *q*: *p* = *p* (*G*) = *V*, *q* = *q* (*G*) = *E*,
$p = p(G) = |V|$, $q = q(G) = |E|$, $V = \{v_1, v_2, v_3, v_4\}$, $E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4)\}$,

$E^{-1} = \{(v_2, v_1), (v_3, v_1), (v_4, v_1), (v_3, v_2), (v_4, v_3)\}$.

Приклад 3.2. Розглянемо ту ж множину студентів в аудиторії V, що являють
собою вершини графа. А в якості відношення між вершинами візьмемо спів падінняднів народження – це буде множина ребер Е. Тобто, V – множина студентів ваудиторії, Е – множина людей зі співпадаючим днем народження. Розглянемо приклад 3.3, що включає складнішу конструкцію – зорієнтованими ребрами.

Приклад 3.3. Повернемося до множини студентів в аудиторії V, а в якостівідношення, що характеризує множину ребер, розглядатимемо симпатію між ними.

В цьому разі симпатія є направленою. І може бути як односторонньою, так івзаємонаправленою, як показано на рис.3.5.

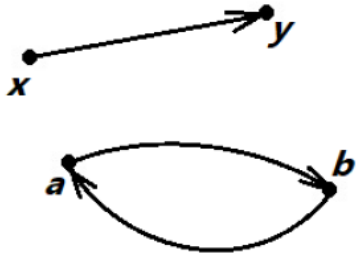


Рис.3.5. Орієнтовані графи

Як видно з верхнього графа на рис. 3.5, $E = (x, y)$, тобто x симпатизує y. Проте,
зворотне не вірно $(x, y) \neq (y, x)$. В цьому разі ребра зображають зі стрілкою іназивають дугами, а сам граф називають орієнтований граф, або скорочено орграф. Із нижнього графа на рис.3.5. можна зробити висновок, що $(a, b) = (b, a)$, тобто між a і b існує взаємна симпатія.

Орієнтований граф

Визначення. Орієнтованим графом $G(V, E)$ називають сукупність двох множин:

– непорожньої множини V, яку називають множиною **вузлів**;

– множини E упорядкованих пар елементів множини V, яку називають множиною **дуг**.

Умови існування орієнтованого графа $G(V, E)$ задають наступнимиправилами:

1. $V \neq \varnothing$. Множина вузлів графа непорожня.

2. $E \subset V \times V$. Елементами множини дуг є відношення, елементи якого –це двійки з вузлів графа. Кількість двійок є підмножиною декартовогодобутку $V \times V$.

3. $E = E^{-1}$. Множина дуг орієнтованого графа – це антисиметричневідношення.

Визначення.Якщо елементами множини є впорядковані пари, то граф називають

орієнтованим (або **орграфом**). Дуги зображують лініями зі стрілками, що вказують напрямок. Прикладорієнтованого графа показаний на рис. 3.6.

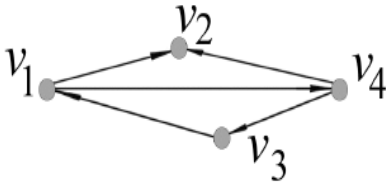


Рис.3.6. Орієнтований граф

$V = \{v_1, v_2, v_3, v_4\},\ E = \{(v_1, v_2), (v_1, v_4), (v_4, v_2), (v_4, v_3), (v_3, v_1)\}.$

Приклад 3.4. Дано орграф *G* (*V*, *E*), зображений на рис. 3.7:

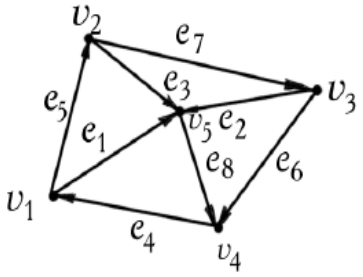


Рис.3.7. Орграф *G* (*V*, *E*)

Визначити впорядковані пари множин $E = \{e_i | i = 1,2,3,\ldots,7\}$, що задає дуги даного графа.

Розв'язок. Запишемо відношення, яке відповідає множині впорядкованих пар вузлів *V*/

$E = \{(v_1, v_5), (v_3, v_5), (v_2, v_5), (v_4, v_1), (v_1, v_2), (v_3, v_4), (v_2, v_3)\}$, $e_1 = (v_1, v_5)$, $e_2 = (v_3, v_5)$,

$e_3 = (v_2, v_5)$.

Помічені графи

Визначення. Якщо задана функція F:$V \to M$або F: $E \to M$, то множину M називають множиною **міток**, а граф називають **поміченим графом**. Такі графи показані на рис. 3.8.

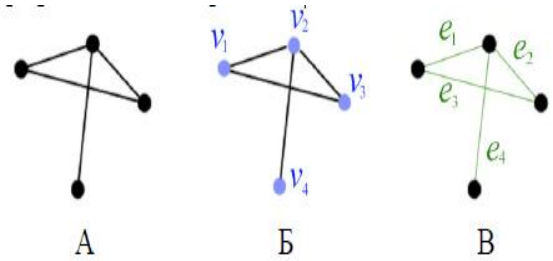


Рис. 3.8. Помічені (Б, В) і непомічені графи (А),

Де А – непомічений граф, Б – граф з поміченими вершинами $V = \{v_1, v_2, v_3, v_4\}$ (вершинно-помічений граф), В – граф з поміченими ребрами $E = \{e_1, e_2, e_3, e_4\}$ (реберно-помічений граф).

Граф з петлями

Якщо серед елементів множини *E* зустрічаються пари, які містятьоднакові вершини, то такий граф називають **графом з петлями.**

Приклад 3.5. На рис. 3.7 показаний граф з петлями і орграф з петлями.

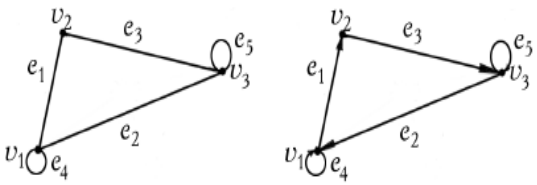


Рис. 3.9. Граф $G_1(V_1, E_1)$,та орграф $G_2(V_2, E_2)$з петлями

Задати ці графи аналітично.

Розв'язок. $G_1(V_1, E_1)$; $V_1 = \{v_1, v_2, v_3\}$; $E_1 = \{(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_2), (v_1, v_3), (v_3, v_1), (v_1, v_1), (v_3, v_3)\}$;

$G_2(V_2, E_2)$; $V_2 = \{v_1, v_2, v_3\}$; $E_2 = \{(v_1, v_2), (v_2, v_3), (v_3, v_1), (v_1, v_1), (v_3, v_3)\}$.

Приклад 3.6. Повернемось до задачі про Кенігсбергські мости. Ця задача сформульована вXVIII ст Леонардо Ейлером поклала початок теорії графів. Суть задачі в

наступному. Ейлер, живучи в Кенігсберзі любив прогулюватися понад річкою ічерез мости. І от в нього виникало запитання чи існує такий шлях, щоб вийти здому, обійти всі мости і повернутися додому але обійти мости так, щоб кожен містпройти рівно один раз. Ейлер сам цю задачу сформулював і сам її розв'язав тимсамим поклавши початок використанню графів у математиці. Більш того у цьомуразі суша – це вершини графа, мости – ребра. Тоді схема мостів у задачі Ейлераприводить нас до графа, що показаний на рис. 3.10. Як видно з рис.3.10, з вершини довершини є кілька ребер. Такі ребра називаються кратними, а сам граф носить назву мультиграфа.

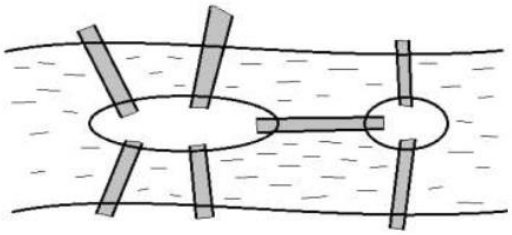


Рис.3.10. Схема мостів у задачі Ейлера

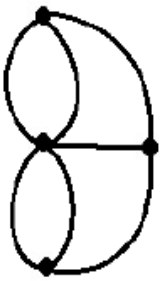


Рис. 3.11. Мультиграф до задачі Ейлера

Мультиграф

Визначення. Якщо множина E містить повторювані елементи, товідповідний граф G (V, E) включає кратні ребра. Тоді його називають**мультиграфом.**

Приклад 3.7. Мультиграф з петлями G (V, E) показаний на рис. 3.12.

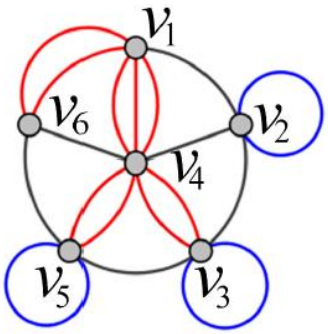


Рис. 3.12. Графічне представлення мультиграфа

Описати цей граф аналітично.

Розв'язок.

Множина вершин: $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$,

Множина ребер: $E = \begin{Bmatrix} (v_1,v_2),(v_1,v_6),(v_1,v_6),(v_1,v_4),(v_1,v_4)(v_1,v_4),(v_2,v_2),(v_2,v_3),(v_2,v_4), \\ (v_3,v_3),(v_3,v_5),(v_3,v_4), \\ (v_3,v_4),(v_4,v_5),(v_4,v_5),(v_4,v_6),(v_5,v_6),(v_5,v_5) \end{Bmatrix}$

В цілому ми розглянули чотири види графів: неорієнтований, орієнтований, граф з петлями імультиграф.

Приклад 3.8. Розглянемо структуру певного Web-сайту де вершинами будуть

сайти, а ребрами – гіперлінки. У цьому разі зображення графа може мати вигляд, наведений на рис. 3.13. Зліва на рис. показано гіперлінк сторінки на саму себе. Ребро

такого типу називається петля. Як бачимо з рис. 3.13, петлі можуть бути кратні. Граф,

що містить лише петлі (x, x) називається **псевдографом**.

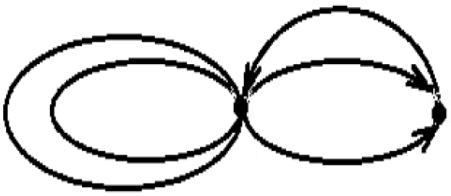


Рис. 3.13.Граф Web-сайту

В цілому, граф на рис. 3.13 є ор-, мульти-, псевдографом.

Таким чином, підходимо до визначення. Граф G = <V, E> – це абстрактний комбінаторний об'єкт, що складається з двох множин. V – довільна

множинавершин, або об’єктів. Множина може бути нескінченною. *E*- множина ребер, абопар об’єктів з *V*.

При цьому

1. Якщо немає кратних ребер, то кожна пара об’єктів зустрічається в множині *V* не більше одного разу.
2. Якщо це не орграф, то пари в *E* – не впорядковані.
3. Якщо немає петель, то (*x*, *x*) ∉E.

Якщо пункт 1 не виконується – маємо мультиграф. Якщо пункт 2 невиконується – маємо орграф. Якщо пункт 3 не виконується маємо псевдограф. Можлива також комбінація цих пунктів, наприклад псевдо- мультиграф і т.п.Тобто, графом можна оперувати не використовуючи його зображення наплощині.

Повний граф

Визначення. Якщо кожна пара вершин графа *G* (*V*, *E*) з’єднана ребром, то такий граф називають **повним**. Кількість ребер в повному графі дорівнює $m = \frac{n(n-1)}{2}$. Повний граф з *n* вершин позначають K_n.

Насиченість графа *D* визначається: $D = \frac{2m}{n(n-1)}$. Для повного графа *D*=1.

Насичений граф – це граф, в якому кількість ребер наближається до максимально можливої: $|E| = O(|V^2|)$.

Розріджений граф – це граф, в якому кількість ребер наближається до кількості вершин: $|E| = O(|V|)$.

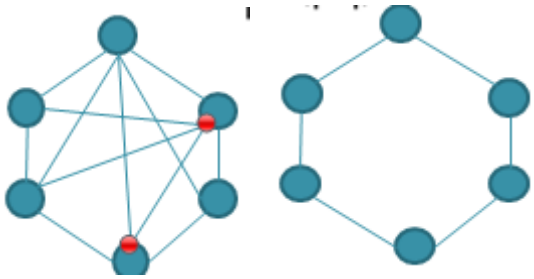


Рис.3.14. Насичений і розріджений графи

Для насиченого графа $D = \frac{2 \cdot 12}{6 \cdot 5} = 0{,}8 > 0{,}5.$

Для розрідженого графа: $D = \frac{2 \cdot 6}{6 \cdot 5} = 0{,}4 < 0{,}5$

Приклад 3.9. На рис. 3.15представлені повні графи:

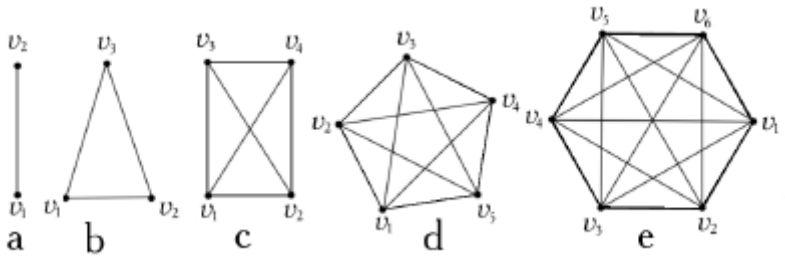


Рис. 3.15. Повні графи a) K_2, b) K_3, c) K_4, d) K_5, e) K_6

Дводольний граф

Визначення. Граф G (V, E) називають **дводольним**, якщо множину його вершин V можна представити об'єднанням множин, які не перетинаються.

Нехай $V = A \cup B$, $A \neq \emptyset$, $B \neq \emptyset$, $A \cap B = \varnothing$, де $A = \{a_1, \ldots, a_i, \ldots, a_n\}$ і $B = \{b_1, \ldots, b_i, \ldots, b_m\}$.

Тоді в дводольному графі існують тільки ребра(a_i, b_j) або (b_j, a_i), $1 \leq i \leq n$, $1 \leq j \leq m$.

Таким чином, кожне ребро зв'язує вершину, яка належить множині A, звершиною, яка належить множині B, але жодні дві вершини з A або двівершини з B не мають спільних ребер. Приклади графічного представленнядводольних графів показані на рис. 3.15.

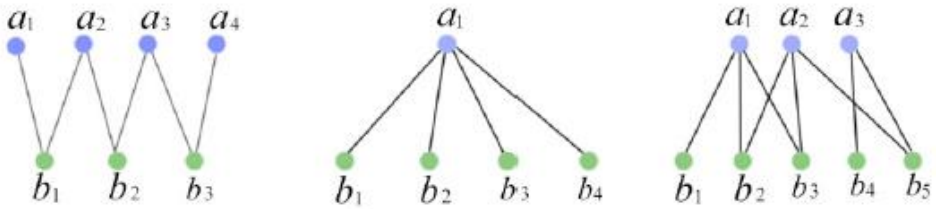


Рис. 3.16. Дводольні графи

Повний дводольний граф

Визначення. Дводольний граф називають **повним дводольним** графом $K_{m,n}$,

якщо A містить m вершин, B містить n вершин і кожна вершина з множиниA з'єднана ребром з кожною вершиною з множини B.

$K_{m,n} \to V = A \cup B$, $A \cap B = \emptyset$, для будь-яких $a \in A$, $b \in B$, якщо $(a, b) \in E$.

На рис. 3.17 представлені повні дводольні графи $K_{1,2}$, $K_{2,3}$, $K_{2,2}$, $K_{3,3}$.

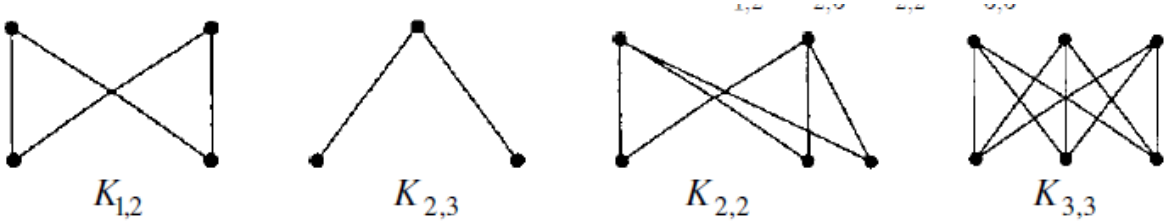


Рис. 3.17. Повні дводольні графи

Граф, який може бути зображено на площині (без перетину ребер), називається **планарним.**

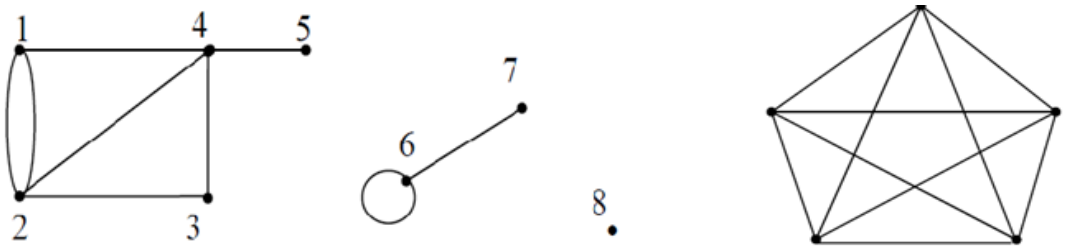


Рис.3.18. Планарний і непланарний графи

Граф, вершини якого можна розбити на n непересічних підмножини так, що ніякі дві вершини, що належать одній підмножині, не суміжні, називається **n-дольним**.

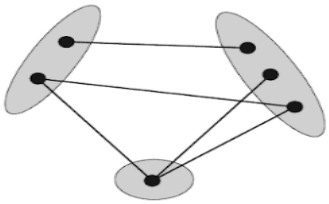


Рис.3.19. Тридольний граф

3.1.3. Суміжність

Нехай $v_1 \in V$і $v_2 \in V$– вершини, $e = (v_1, v_2)$– ребро, що з'єднує вершини v_1та v_2, $e \in E$. Тоді вершина v_1 та ребро e інцидентні. Також вершина v_2та ребро e інцидентні.

Два ребра, які інцидентні одній вершині, називають **суміжними ребрами** (рис. 3.20 а).

Дві вершини, які інцидентні одному ребру, називають **суміжними вершинами**
(рис. 3.20 б).

Рис. 3.20. Суміжні ребра (а) та суміжні вершини (б)

Визначення. Множину вершин, суміжних з вершиною v, називають**множиною суміжності вершини**або відображенням вершини v, і позначають $G(v)$. На рис. 3.21 показано множину вершин $\{u_1, u_2, u_3, u_4, u_5\}$, суміжних з вершиною v.

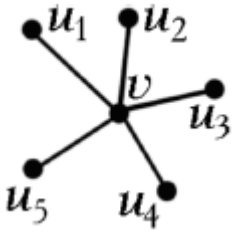


Рис. 3.21. Вершини $\{u_1, u_2, u_3, u_4, u_5\}$, суміжні з v

Вершини утворюють множину відображень $G(v)$ вершини v. Звідси $G(v) = \{u_i \in V | (u_i, v) \in E, 0 \le i \le p-1\}$, де $p = |V|$.

Приклад 3.10. Нехай дано граф G (V, E), показаний на рис. 3.22

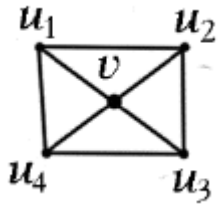


Рис. 3.22. Неорієнтований граф G (V, E)

Описати аналітично множини вершин та ребер даного графа та задатимножину G(v) предикатом.

Розв'язок. Множина вершин: $V = \{u_1, u_2, u_3, u_4, u_5\}$, $p = |V| = 5$.

Множина ребер:

$E = \{(u_1, v), (u_2, v), (u_3, v), (u_4, v), (u_1, u_2), (u_1, u_4), (u_3, u_4), (u_3, u_2)\}$, $q = |E| = 8$..

Множина відображення вершини v: $G(v) = \{u_i \in V | (u_i, v) \in E, i = 1, \ldots, 4\}$.

3.1.4. Степінь вершини

Визначення. Степенемвершини v називаютькількість ребер, інцидентних цій вершині. **Степінь** вершини позначають deg (v) або d (v), для будь-якого $v \in V$ $0 \le deg(v) \le p-1$, де $p = |V|$.

Зауваження. Наявність петлі не збільшує степінь вершини. **Степінь** вершини дорівнює потужності множини суміжності: $deg(v) = |G(v)|$.

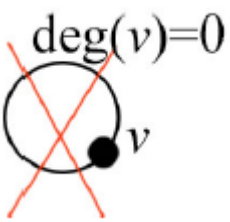


Рис.3.23. Степінь вершини з петлею

Позначимо мінімальний степінь вершини графа G через $\delta(G)$, а максимальний – через $\Delta(G)$. Тоді $\delta(G) = \min_{v \in V} \deg(v)$, $\Delta(G) = \max_{v \in V} \deg(v)$.

Розглянемо характеристики неорієнтованого графа $G(V, E)$, показаного на рис. 3.24.

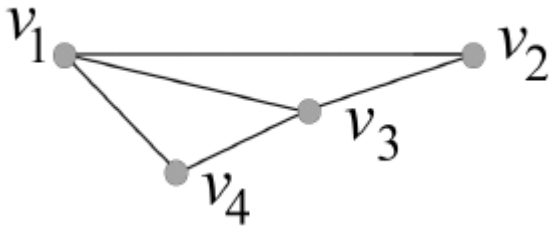


Рис. 3.24. Неорієнтований граф *G* (*V*, *E*)

$\boldsymbol{V = \{v_1, v_2, v_3, v_4\},\ p = |V| = 4.}$

$\boldsymbol{E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4)\}.}$

Вершина$\boldsymbol{v_1}$характеризується відображенням $G(v_1)$ і потужністю відображення $|G(v)|$, яка дорівнює степеню даної вершини $deg(v_1)$: $\boldsymbol{G(v_1) = \{v_2, v_3, v_4\},\ |G(v_1)| = 3,}$ $\boldsymbol{deg(v_1) = 3.}$

Вершина v_2 : $G(v_2) = \{v_1, v_3\}$, $\boldsymbol{|G(v_2)| = 2,}$ $\boldsymbol{deg(v_2) = 2.}$

Вершина v_3: $G(v_3) = \{v_1, v_2, v_4\}$, $\boldsymbol{|G(v_3)| = 3,}$ $\boldsymbol{deg(v_3) = 3.}$

Вершина v_4: $G(v_4) = \{v_1, v_3\}$, $\boldsymbol{|G(v_4)| = 2,}$ $\boldsymbol{deg(v_4) = 2.}$

Отже, $\delta(G) = deg(v_2) = deg(v_4) = 2$, $\Delta(G) = deg(v_1) = deg(v_3) = 3$.

Регулярний граф

Визначення. Якщо степені всіх вершин дорівнюють *k*, то граф називають **регулярним графом** зі степенем *k*.

Приклад регулярного графа *G* (*V*, *E*), для якого $\delta(G) = \Delta(G)$ = 4, показанийна рис. 3.25.

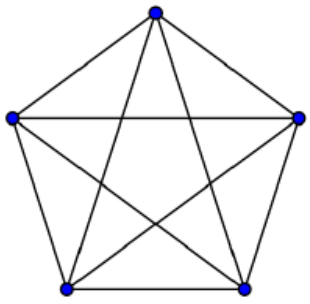


Рис. 3.25. Регулярний граф

Для регулярного *k* -графа справедливе співвідношення: $\delta(G) = \Delta(G)$ = *k*.

Приклад 3.11. Визначити, чи є повним графом граф K_5.

*Розв'язок.*Повний граф K_5 є регулярним графом, оскільки $\delta(K_5) = 4$, $\Delta(K_5) = 4$, як показано на рис. 3.26.

Ізольована вершина

Визначення. Вершину *v*, для якої deg (*v*) = 0, називають **ізольованою.** На рис. 3.26 наведено приклад графа з ізольованою вершиною deg (4) = 0.

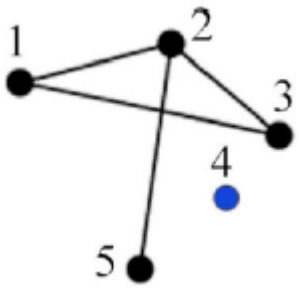


Рис. 3.26. Граф, який містить ізольовану вершину 4

Висяча вершина

Визначення. Вершину *v*, для якої deg (*v*) = 1, називають **кінцевою** або **висячою.** На рис. 3.27 показано граф, який містить **висячу вершину** deg (4) = 1.

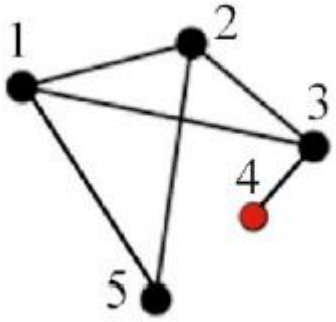


Рис. 3.27. Граф, який містить висячу вершину 4

Напівстепені у орграфі

Визначення. Напівстепенем виходу вузла або прямим відображенням вузла
у орграфі називають кількість дуг, які виходять з даного вузла. Напівстепінь виходу для вузла *v* позначають як $deg^+(v) = |G^+(v)|$. 

Визначення. Напівстепенем входу вузла або зворотним відображенням вузла у орграфі називають кількість дуг, які входять у даний вузол. Напівстепінь входу для вузла *v* позначають як $deg^-(v) = |G^-(v)|$. 

Розглянемо характеристики орієнтованого графа *G* (*V*, *E*), показаного на рис. 3.28.

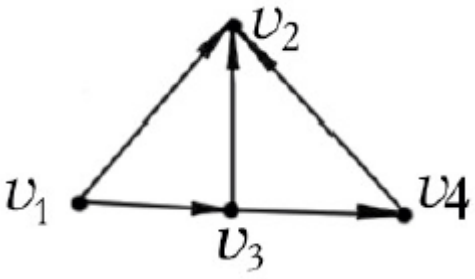


Рис. 3.28. Орієнтований граф *G* (*V*, *E)*

$V = \{v_1, v_2, v_3, v_4\}$, $p = |V| = 4$,

$E = \{(v_1, v_2), (v_1, v_3), (v_3, v_2), (v_3, v_4), (v_4, v_2)\},\ q = |E| = 5,$
$G^+(v_1) = \{v_2, v_3\},\ deg^+(v_1) = |G^+(v_1)| = 2,$
$G^-(v_2) = \{v_1, v_3, v_4\},\ deg^-(v_2) = |G^-(v_2)| = 3,$
$G^+(v_3) = \{v_2, v_4\},\ deg^+(v_3) = |G^+(v_3)| = 2,$
$G^-(v_3) = \{v_1\},\ deg^-(v_3) = |G^-(v_3)| = 1,$
$G^+(v_4) = \{v_2\}, deg^+(v_4) = |G^+(v_4)| = 1,$
$G^-(v_4) = \{v_3\}, deg^-(v_4) = |G^+(v_4)| = 1.$

3.1.5. Теореми про степені вершин графа

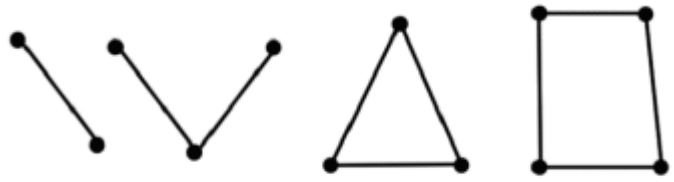
Теорема 3.1. Сума степенів вершин графа завжди парна.

Доведення. Кожне ребро графа має два кінці. Тому кожне ребро збільшуєстепінь кожної з 2-х інцидентних вершин на одиницю. Таким чином, кожнеребро збільшує суму степенів усіх вершин на 2. Отже, сума степенів усіхвершин завжди кратна 2, тобто, парна.

Приклад 3.12. Зобразити 4 довільних графи та визначити суму їх степенів.

*Розв'язок.*Розглянемо 4 графи, показані на рис. 3.28, і визначимо суму їх степенів S_i, де

i = 1, ..., 4.



$S_1 = 2,\ S_2 = 4,\ S_3 = 6,\ S_4 = 8.$

Рис. 3.29. Неорієнтовані графи для визначення суми степенів вершин

Теорема 3.2. У будь-якому графі кількість вершин непарного степеня парна.

Доведення. Доведення виконуватимемо методом від протилежного. Припустимо, що теорема не вірна.

1. Якщо теорема не вірна, то існує непарна кількість вершин, степені якихнепарні.

2. Якщо в графі немає вершин з парними степенями як показано на рис. 3.30, то відразу виникає протиріччя з першою теоремою.

Рис. 3.30. Графи з вершинами, які мають тільки непарні степені

Протиріччя полягає в тому, що кількість вершин у цьому випадку повинна бути парною, оскільки сума степенів вершин графа завжди парна.

3. Якщо в графі є вершини з парними і непарними степенями як показано на
рис. 3.31, то очевидно, що сума степенів вершин з парними степенями парна.

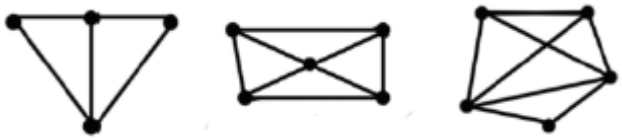


Рис. 3.31. Графи з вершинами, які мають непарні і парні степені

4. Однак, оскільки сума всіх степенів графа парна, то знову виникаєпротиріччя з початковим припущенням. Протиріччя полягає в тому, що, оскільки сума непарної кількості і парної кількості є число непарне, то в цьому випадку сума степенів усіх вершин повинна бути непарною. Але це суперечить теоремі, тому ми прийшли до протиріччя.

Отже, робимо висновок, що теорема доведена.

Теорема 3.3. Теорема Ейлера. Сума степенів вершин графа дорівнює подвоєній кількості ребер:

$\sum_{v\in V}\deg(v)=2$ - для неорієнтованого графа,

$\sum_{v\in V}\deg^{-}(v)+\sum_{v\in V}\deg^{+}(v)=2q$ - для орграфа, де $q=|E|$- потужність множини ребер.

Доведення. При підрахунку суми степенів вершин кожне ребро враховуєтьсядвічі: для одного кінця ребра і для іншого. Звідси випливає, що кількість реберу графі має дорівнювати половині суми степенів вершин. Теорема доведена.

Приклад 3.13. Перевірити задане теоремою Ейлера співвідношення вершин на
графах, показаних на рис. 3.32.

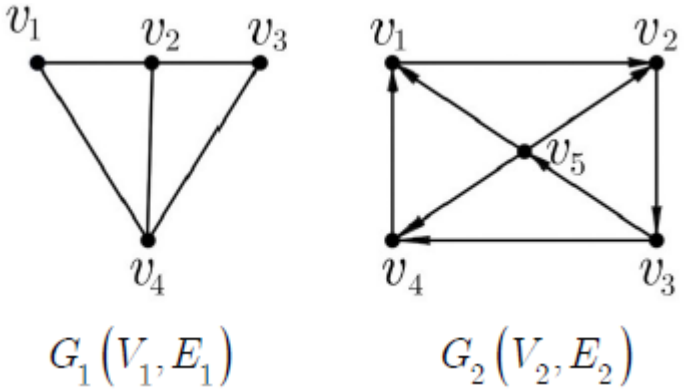


Рис. 3.32. Графи для визначення співвідношення вершин і ребер

Неорієнтований граф $G_1(V_1,E_1)$:

$\sum_{i=1}^{3}\deg(v_i)=10,\ q=|E|=5,$

Орієнтований граф $G_2(V_2, E_2)$

$\sum_{i=1}^{5} \deg^-(v_i) = 8$, $\sum_{i=1}^{5} \deg^+(v_i) = 8$, $q = |E| = 8$.

3.1.6. Підграф графа

Визначення. Граф $G' = (V', E')$називають **під графом** графа $G(V, E)$ $G'(V', E') < G(V, E)$, якщо $V' \subseteq V$і $E' \subseteq E$. Отже,

- кожна вершина в G'є одночасно вершиною в G,
- кожне ребро в G'є одночасно ребром в G.

Визначення. Граф $G'(V', E')$називають **остовним підграфом** G або **суграфом** графа G, якщо $V' = V$і $E' \subseteq E$. Отже,

- множини вершин графа G'та графа G співпадають,
- кожне ребро в G'є одночасно ребром в G.

Визначення. Граф $G'(V', E')$називають **правильним підграфом** графа G, якщо $V' \subset V$і G'містить усі можливі ребра G: для будь-яких $(u, v) \in G'(u, v)$ таких, що $(u, v) \in E \Rightarrow (u, v) \in E'$.

Правильний підграф утворюють шляхом виключення з графа певної кількості
вершин та інцидентних до них ребер.

Приклад 3.14. На рисунку (a) показаний граф G (V, E). Визначити, якими є його підграфи (b), (c) і (d).

Розв'язок.

(**b**). На рисунку представлений **підграф $G_1(V_1, E_1)$** графа G (V, E), оскільки$V_1 \subset V$ і $E_1 \subset E$.

(**c**). Граф $G_2(V_2, E_2)$є **остовним графом** або суграфом графаG (V, E), тому що $V_2 = V$ й $E_2 \subset E$.

(**d**). Граф $G_3(V_3, E_3)$є **правильним підграфом** графа G (V, E), оскількимістить усі його можливі ребра.

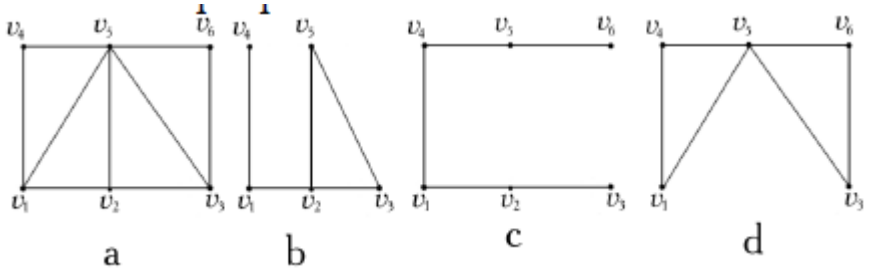


Рис. 3.33. Граф G (V, E) (a) та його підграфи (b), (c) і (d)

3.1.7. Структурні характеристики графів

Неорієнтованіі графи

Нехай G= (V, E) – скінченний неорієнтований граф. Скінчена послідовність вершин та ребер графа $v_0e_1v_1e_2v_2\dots v_{k-1}e_kv_k$, в якій кожне ребро e_kє ребро, яке з'єднує вершини v_{k-1}та v_k, називається **маршрутом** на графі G.

Маршрут з'єднує вершини v_0та v_k.

Число k називають **довжиною маршруту**, тобто це кількість ребер, які входять до маршрута.

Маршрут називають **замкненим**, якщо $v_0 = v_k$.

Маршрут, в якому всі ребра є різні, називають **ланцюгом**.

Замкнений ланцюг називають **циклом.**

Цикл повністю визначають множиною його ребер.Тому часто під циклом ми будемо розуміти відповідну йому множину ребер.

Петля дає цикл довжини 1.

Пара кратних ребер утворює цикл довжини 2.

Цикли довжини 3 зазвичай називають *трикутниками.*

Ланцюг називають **простим**, якщо всі його вершини є різні.

Простий цикл– це цикл, у якому всі вершини, окрім першої та останньої, є різні.

Граф без циклів називається **ациклічним**, в іншому разі граф називається **циклічним**.

Кожна вершина з'єднується сама з собою маршрутом довжини **0** і цей маршрут є простим циклом. Такий цикл називають **нульовим** (якщо сказано просто цикл, то мається на увазі, що він **не є нульовий**).

Лема. Якщо для деяких двох вершин u і v в графі існує (u, v)-маршрут, то існує і простий (u, v)-ланцюг.

*Доведення.*Розглянемо в графі (рис. 3.33) (u, v)-маршрут найменшої довжини.

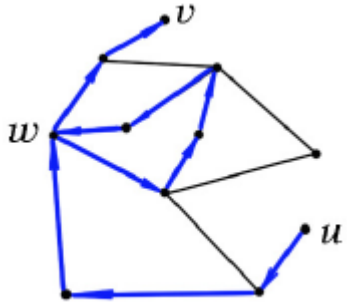


Рис. 3.34. Граф з маршрутом (u, v)

Покажемо, що цей маршрут є простим ланцюгом. Якщо в ньомує повторювана вершина w, то, заміняючи частину маршруту від першоговходження вершини w до її другого входження на одну вершину w, миодержимо більш короткий (u, v)-маршрут.

Приклад 3.15. Визначити структурні характеристики неорієнтованого графа.

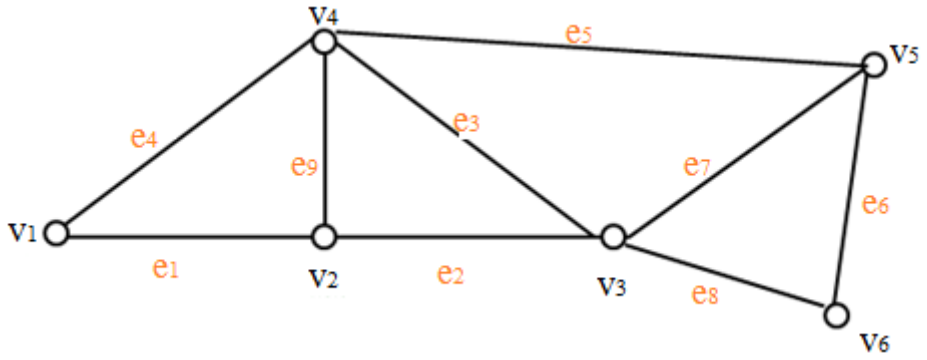


Рис.3.35 Структурні характеристики неорієнтованого графа

Маршрут – $v_1e_1v_2e_1v_1e_4v_4e_3v_3e_3v_4e_5v_5$.

Ланцюг (всі ребра різні) – $v_1e_4v_4e_9v_2e_2v_3e_3v_4e_5v_5$.

Простий ланцюг (всі ребра і вершини різні) – $v_1e_4v_4e_9v_2e_2v_3e_7v_5$.

Цикл – $v_1e_1v_2e_9v_4e_3v_3e_7v_5e_5v_4e_4v_1$.

Простий цикл – $v_1e_4v_4e_9v_2e_1v_1$.

Орієнтовані графи

Орієнтовані маршрути: в орграфі рух за маршрутом допускається лише в напрямках, зазначених стрілками.

Маршрут, який не містить повторних дуг, називається **шляхом**, а той, що не містить повторних вершин, – **простим шляхом**. Замкнений шлях називається **контуром**, а простий замкнений шлях – **простим контуром**.

Граф без циклів називається **безконтурним**, в іншому разі орграф називається **контурним**.

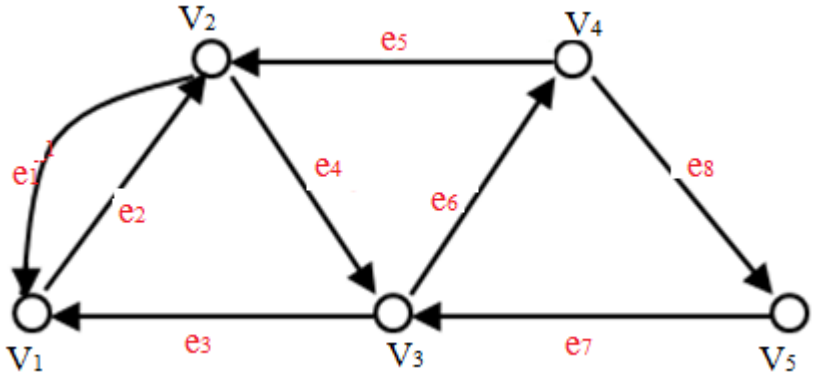


Рис.3.36 Структурні характеристики орієнтованого графа

Маршрут - $v_1e_2v_2e_4v_3e_6v_4e_5v_2e_4v_3$.

Шлях (всі ребра різні) - $v_3e_6v_4e_8v_5e_7v_3e_3v_1$.

Простий шлях (всі ребра і вершини різні) - $v_5e_7v_3e_6v_4e_5v_2e_1v_1$.

Контур - $v_1e_2v_2e_4v_3e_6v_4e_5v_2e_1v_1$.

Простий контур - $v_1e_2v_2e_4v_3e_3v_1$.

3.2. Способи задавання й властивості графів

3.2.1. Способи представлення графів

Представлення графа в пам'яті (формат збереження) визначає обчислювальну складність операцій над графом і об'єм необхідної пам'яті.

У дискретній математиці прийнято розглядати три способи задавання графів: 1. *Аналітичний спосіб*

Аналітичний спосіб припускає представлення графа G (V, E) у вигляді множин V і E. Для задавання цих множин можуть використовуватися всі три способи задавання множин. На рис. 3.37 показано граф G для математичного представлення.

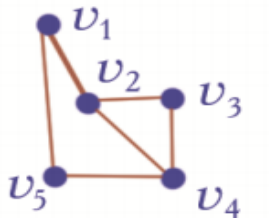


Рис.3.37. граф G

Явний спосіб у вигляді кортежу $G(V,E)$, де $V=\{v_1,v_2,v_3,v_4,v_5\}$ - множина вершин, $E=\{(v_1,v_2),(v_2,v_3),(v_2,v_4),(v_3,v_4),(v_4,v_5),(v_5,v_1)\}$- множина ребер.

Задавання предикатом: $V=\{v_i|i=1,\ldots,n\}$, $E=\{(v_i,v_j)|i=2k+1,j=2k,k=1<\ldots,2n-1\}$.

Рекурсивною процедурою: $V=\{v_i|i=i+1,i<m\}$,
$E=\{(v_i,v_j)|j=j+1,i=i+2,i,j<n\}$.

2. *Графічний спосіб*

Вершини представлені точками, а ребра – лініями, які з'єднують ці точки. На рис. 3.38 показані способи графічного представлення різних графів.

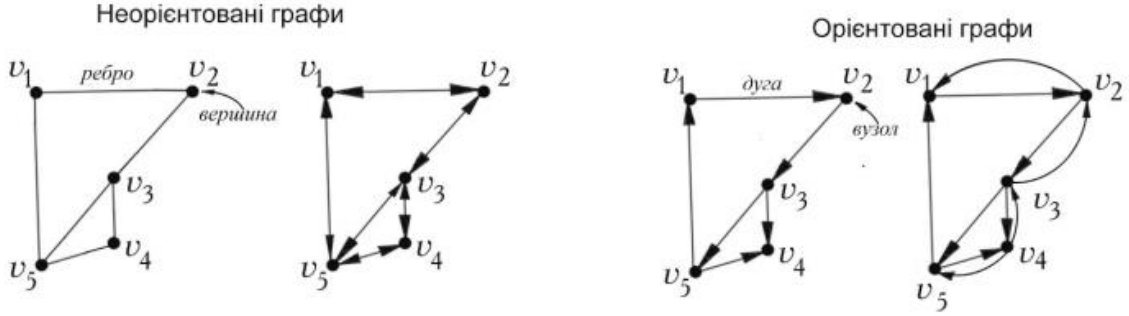


Рис. 3.38. Способи графічного представлення орієнтованих і неорієнтованих графів

3. *Матричний спосіб*

Граф задають у вигляді матриці інцидентності або матриці суміжності. Задавання неорієнтованого графа за допомогою матриці інцидентності НехайG – неорієнтований граф. Нехай В – матриця, кожний рядок якої відповідає вершині графа, а кожний стовпець відповідає ребру графа.

$$B = \begin{pmatrix} & e_1 & e_2 & \dots & e_j & \dots & e_m \\ v_1 & 1 & 0 & 0 & 0 & 0 & 1 \\ v_2 & 0 & 1 & 0 & 1 & 0 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ v_i & 0 & 0 & 0 & b_{ij} & 0 & 0 \\ \dots & 0 & 1 & 0 & 0 & 0 & 0 \\ v_n & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$G\,(V, E), V = \{v_1, v_2, \dots, v_i, \dots, v_n\}, E = \{e_1, e_2, \dots, e_j, \dots, e_m\}.$

Елемент i-го рядка та j-го стовпця матриці В позначають b_{ij}. b_{ij}=1, якщоi-та вершина інцидентнаj-му ребру, і дорівнює 0 у протилежному випадку.

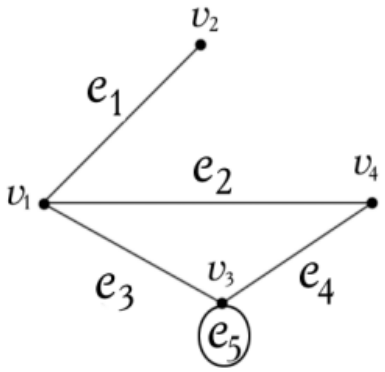


Рис. 3.39. Графічне представлення графа, заданого матрицею В

Матрицю В називають **матрицею інцидентності** неорієнтованого графа G. Отже, елементи матриці інцидентності $B = (b_{ij})$задаються формулою:

$$b_{ij} = \begin{cases} 1, \textit{якщо ребро } e_i \quad \textit{інцидентне вершині } v_j \\ 0, \textit{в протилежному випадку}. \end{cases}$$

Приклад 3.16. Граф G=(V, E) задано аналітично множнами $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5\} =$
$\{(v_1, v_2), (v_1, v_4), (v_1, v_3), (v_3, v_4), (v_3, v_3)\}$.
Сформувати матрицю інцидентності та зобразити даний граф графічно.
Розв'язок.
Матриця інцидентності графа G= (V, E) має вигляд:

	e_1	e_2	e_3	e_4	e_5
v_1	1	1	1	0	0
v_2	1	0	0	0	0
v_3	0	0	1	1	1
v_4	0	1	0	1	0

або у більш формалізованому вигляді:

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Графічне представлення графа, заданого матрицею B, відповідає зображенню графа на рис. 3.39.

Властивості матриці інцидентності неорієнтованого графа

1. Для вершин без петель степінь вершини дорівнює сумі одиничних елементів відповідного рядка матриці, оскільки кожна одиниця в цьому рядку представляє інцидентність цієї вершини ребру.
2. У кожному стовпці, який не представляє ребро петлі, будуть дві одиниці, тому що кожне таке ребро інцидентне двом вершинам.
3. У рядку матриці інцидентності, який відповідає вершині з петлею, сума одиниць на одну більше степеня даної вершини.
4. Стовпець, що відповідає ребру петлі, містить тільки одну одиницю.

Задавання орієнтованого графа за допомогою матриці інцидентності

НехайG – орієнтований граф. Тоді матриця інцидентності $B = (b_{ij})$ включає елементи, які дорівнюють 1, якщо вершина інцидентна з початком ребра, дорівнюють -1, якщо вершина інцидентна з кінцем ребра, дорівнюють 0, якщо вершина і ребро не інцидентні, дорівнюють 2 або будь-якому числу, якщо вершина містить петлю.

$$b_{ij} = \begin{cases} 1, \text{ якщо вершина } v_j \text{ є початком ребра } e_i, \\ -1, \text{ якщо вершина } v_j \text{ є кінцем ребра } e_i, \\ 2, \text{ якщо вершина } v_j \text{ є початком і кінцем ребра } e_i, \\ 0, \text{ в інших випадках.} \end{cases}$$

Приклад 3.17. Нехай задано орієнтований граф $G = (V, E)$, де $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. Описати даний граф матрицею та зобразити графічно.

Розв'язок.

Графічне представлення даного орграфа показано на рис. 3.39:

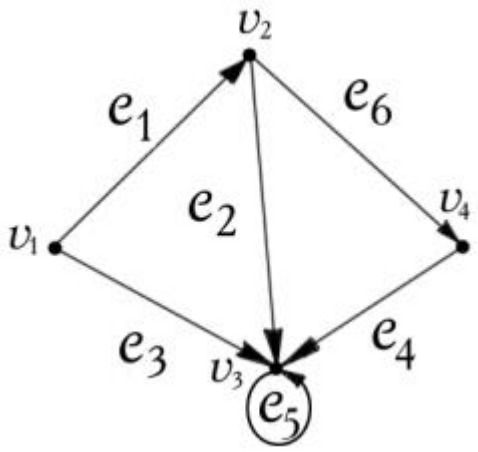


Рис. 3.40. Орграф $G = (V, E)$

Матриця інцидентності орграфа має вигляд:

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	1	0	1	0	0	0
v_2	−1	1	0	0	0	1
v_3	0	−1	−1	−1	2	0
v_4	0	0	0	1	0	−1

або

$$B = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

Властивості матриці інцидентності орграфа

1. Для вершин без петель напівстепінь виходу дорівнює сумі додатних одиничних елементів відповідного рядка
2. Для вершин без петель напівстепінь входу дорівнює сумі від'ємних одиничних елементів відповідного рядка.
3. У кожному стовпці, який не представляє ребро петлі, сума елементів дорівнює нулю.
4. Якщо дуга – це петля, то в стовпці один елемент, який дорівнює 2.

Задавання неорієнтованого графа за допомогою матриці суміжності

НехайG – неорієнтований граф. Нехай C – матриця, рядки якої позначені вершинами графа і стовпці позначені тими ж вершинами в тому ж самому порядку.

Елементi-го рядка й j-го стовпця матриці C позначається c_{ij}, і:

- дорівнює 1, якщо існує одне ребро з i-ї вершини в j-у вершину,
- дорівнює числу ребер з i-ї вершини в j-у вершину за наявності декількох ребер,
- дорівнює 0, якщо ребер між вершинами не існує.

$$C = \begin{pmatrix} & v_1 & v_2 & v_i & v_j & v_n \\ v_1 & 0 & 0 & 1 & 0 & 1 \\ v_2 & 0 & 0 & 1 & 1 & 1 \\ v_i & 1 & 1 & 0 & c_{ij} & 1 \\ v_j & 0 & 1 & c_{ji} & 0 & 0 \\ v_n & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Матрицю C називають **матрицею суміжності** графаG.

Формула для визначення елементів матриці суміжності графа:

$$c_{ij} = \begin{cases} 1, \textit{ якщо існує ребро } (v_i, v_j), \\ k, \textit{ якщо існують ребра } \{\overbrace{(v_i, v_j), (v_i, v_j), \ldots, (v_i, v_j)}^{k}\} \\ 0, \textit{ в інших випадках.} \end{cases}$$

Приклад 3.18. Розглянемо неорієнтований граф G (V, E), показаний на рис. 3.58.

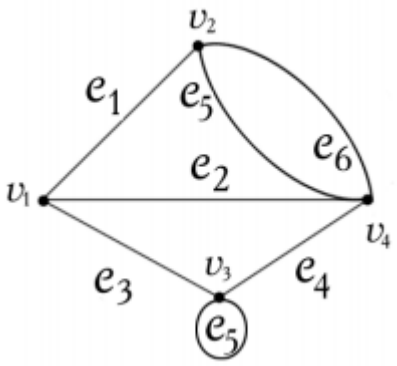


Рис. 3.41. Неорієнтований граф G (V, E)

Сформувати матрицю суміжності для даного графа.

Розв'язок.

Матриця суміжності даного графа має вигляд:

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 1 & 1 & 1 \\ v_2 & 1 & 0 & 0 & 2 \\ v_3 & 1 & 0 & 1 & 1 \\ v_4 & 1 & 2 & 1 & 0 \end{array} \quad \text{або } C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

Властивості матриці суміжності

1. Матриця суміжності неорієнтованого графа симетрична відносно головної діагоналі.

2. Якщо вершина має петлі, то їх число розміщається на головній діагоналі матриці суміжності.

3. Якщо між двома вершинами графа існує кілька ребер, то на перетині рядків і стовпців проставляють їх кількість.

Матриця суміжності орієнтованого графа

Нехай G – орієнтований граф. Нехай С – матриця, рядки якої позначені вершинами графа і стовпці позначені тими ж вершинами в тому ж самому порядку.

Елемент i-ого рядка й j-го стовпця матриці С, позначається c_{ij}, і:

- дорівнює 1, якщо ребро виходить із вершиниv_i, представленої i-м рядком і входить у вершину v_j, представлену j-м стовпцем матриці.

- дорівнює числу ребер з i-ї вершини в j-у вершину за наявності декількох ребер,

- дорівнює 0, якщо ребер між вершинами не існує.

Матрицю С називають матрицею суміжності орграфаG.

Приклад 3.19. Розглянемо орієнтований граф G (V, E), що показаний на рис. 3.42:

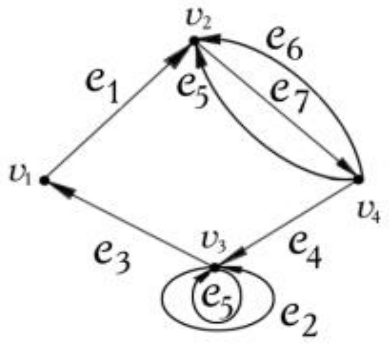


Рис. 3.42. Орієнтований граф G (V, E)

Сформувати матрицю суміжності для даного графа.

Розв'язок.

Його матриця суміжності має вигляд:

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 1 & 0 & 0 \\ v_2 & 0 & 0 & 0 & 1 \\ v_3 & 1 & 0 & 2 & 0 \\ v_4 & 0 & 2 & 1 & 0 \end{array} \text{ або } C = \begin{Vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 \end{Vmatrix}.$$

Властивості матриці суміжності орієнтованого графа

1. Матриця суміжності несиметрична відносно головної діагоналі.
2. Сума чисел у рядку матриці суміжності без урахування чисел на головній діагоналі дозволяє визначити потужність напівстепеня виходу для кожної вершини орграфа: $deg^+(v_i)$, де $1 \le i \le n$.
3. Сума чисел у стовпці матриці суміжності без урахування чисел на головній діагоналі дозволяє визначити потужність напівстепеня входу для кожної вершини орграфа: $deg^-(v_i)$, де $1 \le i \le n$.

Задавання графа за допомогою списку ребер

Список ребер графа (орграфа) представлено двома стовпцями. Перший стовпець містить ребра, а другий – інцидентні з ними вершини. Для неорієнтованого графа порядок проходження вершин довільний. Для орграфа першим стоїть номер вершини, з якої ребро виходить. Список ребер графа (орграфа) може також бути представлений послідовністю елементів, кожний з яких містить ребро та інцидентну йому пару вершин.

Приклад 3.20. Розглянемо граф G (V, E), показний на рис. 3.43.

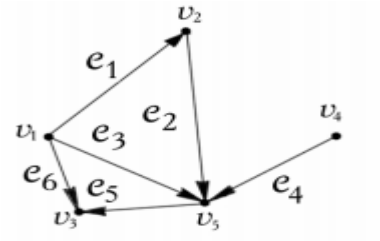


Рис. 3.43. Граф *G* (*V*, *E*)

Описати даний граф списком його ребер.

Розв'язок.

Орграф і список його ребер

$e_1 \to (v_1, v_2),\ e_2 \to (v_2, v_3),\ e_3 \to (v_1, v_5),\ e_4 \to (v_4, v_5),\ e_5 \to (v_5, v_3),\ e_6 \to (v_1, v_3),$

3.2.2. Ізоморфізм графів

З попереднього ми дізналися про різні способи задавання графів. Отже, довільний граф можна задати графічно (рисунком), матрицею інцидентності, матрицею суміжності і списком ребер.

При графічному задаванні графа вигляд рисунка залежить від форми ліній і взаємного розташування вершин. Тому не завжди легко зрозуміти, чи однакові графи, зображені на різних рисунках. Наприклад, на рисунках 3.44 а та 3.44 б зображено один і той же граф.

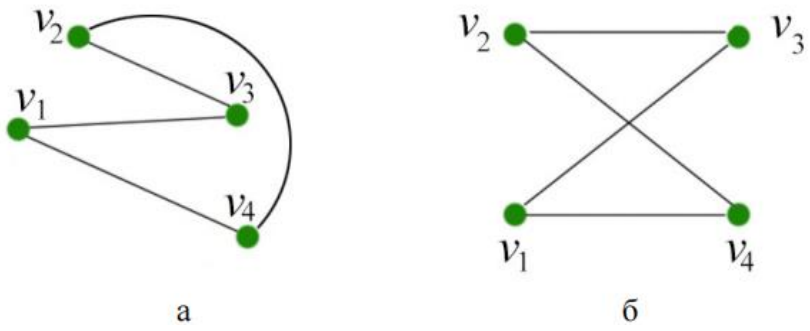


Рис. 3.44. Різні графічні представлення одного і того ж графа

При задаванні графа матрицями або списком ребер їх вигляд залежить від нумерації вершин і ребер графа. Будемо вважати, що граф повністю заданий, якщо нумерація його вершин зафіксована.

Визначення. Графи, що відрізняються тільки нумерацією вершин, називають ізоморфними графами.

Ізоморфізм графів

Нехай $G = (V_1, E_1)$ і $H = (V_2, E_2)$– графи. Між вершинами даних графів існує взаємно однозначна відповідність (бієкція) $R: V_1 \to V_2$. Потужності множин V_1та V_2співпадають: $|V_1| = |V_2|$.

Визначення. Відображення R називають ізоморфізмом графів G і H, якщо для будь-яких суміжних вершин $v_i, v_j \in G$ їх образи $(u_k, u_m) \in H$ також суміжні.

Визначення. Якщо відображення R існує на графах G і H, які відповідають попереднім умовам, то такі графи називають ізоморфними графами.

Приклад 3.21. На рис. 3.45 показані графи G (V, E) (а) та H (U, F) (б).

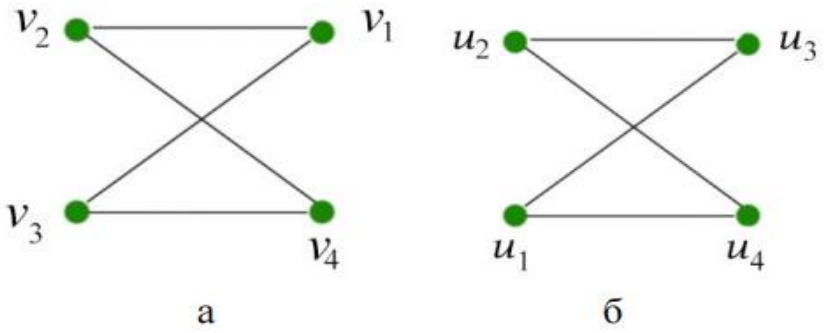


Рис. 3.45. ГрафиG (V, E) (а) та H (U, F) (б).

Визначити, чи дані графи є ізоморфними.

Розв'язок.

Порівняємо множини відображень для вершини v_1у графі G (V, E) та u_1у графі H (U, F).

$$\begin{cases} \Gamma(v_1) = \{(v_1, v_2), (v_1, v_3)\} \\ \Gamma(u_3) = \{(u_3, u_2), (u_3, u_1)\} \end{cases}$$

З даних відображень можна зробити висновок, що $R: v_1 \to u_3$.

Отже, відношення, яке задане перерахуванням, має вигляд:

$R = \{(v_1, u_3), (v_2, u_2), (v_3, u_1), (v_4, u_4)\}$.

Приклад 3.22. На рис. 3.46 показані графи $G(V_1, E_1)$ та $H(V_2, E_2)$.

Визначити, чи є дані графи ізоморфними.

Розв'язок.

1. Визначаємо потужність вершин графів: $|V_1| = 8$, $|V_2| = 8$, $|V_1| = |V_2|$

2. Знаходимо однозначне відображення вершин.

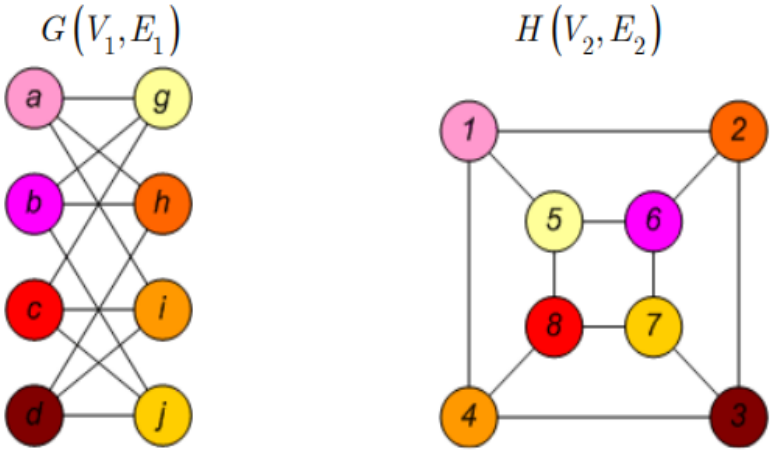


Рис.3.46 Графи $G(V_1, E_1)$ та $H(V_2, E_2)$

$f(a) = 1$
$f(b) = 6$
$f(c) = 8$
$f(d) = 3$
$f(g) = 5$
$f(h) = 2$
$f(i) = 4$
$f(j) = 7$

$(c,g) \to (8,5)$
$(c,i) \to (8,4)$
$(c,j) \to (8,7)$
$(d,h) \to (3,2)$
$(d,i) \to (3,4)$
$(d,j) \to (3,7)$

$(a,g) \to (1,5)$
$(a,h) \to (1,2)$
$(a,i) \to (1,4)$
$(b,g) \to (6,5)$
$(b,h) \to (6,2)$
$(b,j) \to (6,7)$

3. Одержуємо відношення:

$R = \{(a,1),(b,6),(c,8),(d,3),(h,2),(g,5),(i,4),(j,7)\}$.

Приклад 3.23. На рис. 3.47 показані ізоморфні графи G й H.

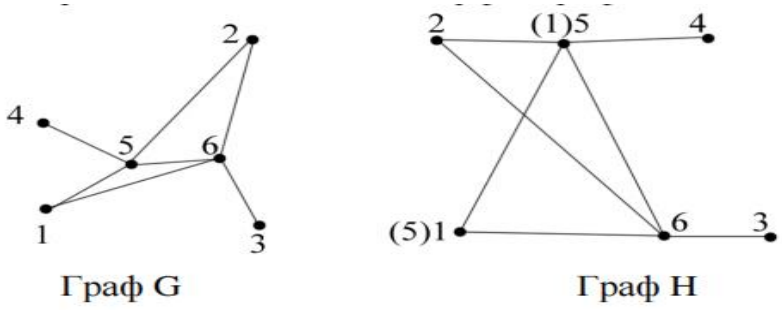


Рис. 3.47. Ізоморфні графи G і H

Побудувати матриці суміжності даних графів.

Розв'язок.

$G_{сум}$– матриця суміжності графа G й $H_{сум}$– матриця суміжності графа H.

$$G_{\text{сум}} = \begin{Vmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{Vmatrix} \quad H_{\text{сум}} = \begin{Vmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{Vmatrix}$$

Властивість матриць суміжності ізоморфних графів

Якщо графиG й H ізоморфні, то з матриці суміжності $G_{сум}$ графа можна одержати матрицю суміжності $H_{сум}$ шляхом послідовної перестановки рядків і стовпців. Для цього потрібно виконати максимально $n!$ перестановок, де n – число вершин графа.

Ізоморфізм орграфів

Визначення. Орграфи, що відрізняються нумерацією вершин та мають з точністю до нумерації однаково направлені ребра, називають **ізоморфними орграфами**.

3.2.3. Алгоритм розпізнавання ізоморфізму графів

1. Перевіряємо умову $|V| = |W| = n$. Якщо кількість вершин графа $|V|$не дорівнює кількості вершин графа $|W|$, то графи однозначно неізоморфні.
2. Сортуємо елементи множин $V = \{v_1, v_2, v_3, \dots, v_n\}$ і $W = \{w_1, w_2, w_3, \dots, w_n\}$за критерієм величини степенядля кожної вершини.
3. В отриманих упорядкованих послідовностях вершин шукаємо вершини з однаковими значеннями критерія упорядкування, тобто шукані вершини повинні мати однакові степені.
4. Якщо такі вершини знайдені, то з'єднуємо їх ребром з метою побудови графа взаємно однозначної відповідності. Якщо такої відповідності немає, тобто одна зі знайдених вершин уже включена в граф відповідності, то початкові графи G й H неізоморфні.
5. Якщо граф взаємно однозначної відповідності побудований, то розглянуті графи ізоморфні, а його ребра вказують на перестановки, які потрібно зробити для ізоморфного перетворення графа G в граф H.

На рис. 3.48 відповідні вершини ізоморфних графів з'єднані лініями.

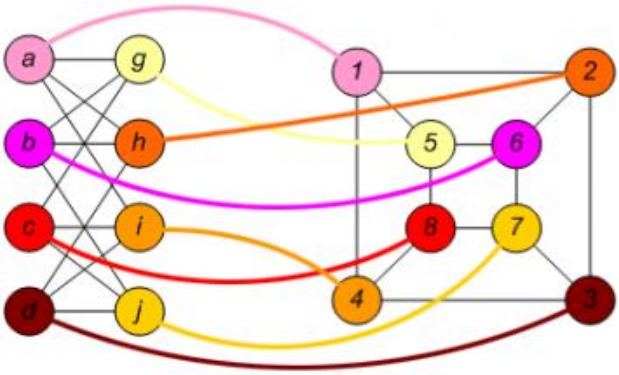
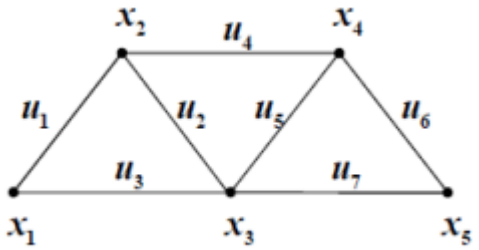


Рис. 3.48. Ізоморфні графи

Обов'язкові завдання

1.У графі, заданому на рисунку визначити маршрут, ланцюг і простий ланцюг.



2. Вважаючи даний граф неорієнтованим, позначити його вершини і ребра різними символами і визначити.

1.1 Локальні степені кожної вершини.

1.2 Побудувати матриці сміжності і інцидентності;

1.3 Показати підграф, що складається з трьох вершин;

1.4 Навести приклади циклічного маршруту, ланцюга, простого ланцюга.

Вважаючи граф орієнтованим, визначити:

2.1 Степені вершин;

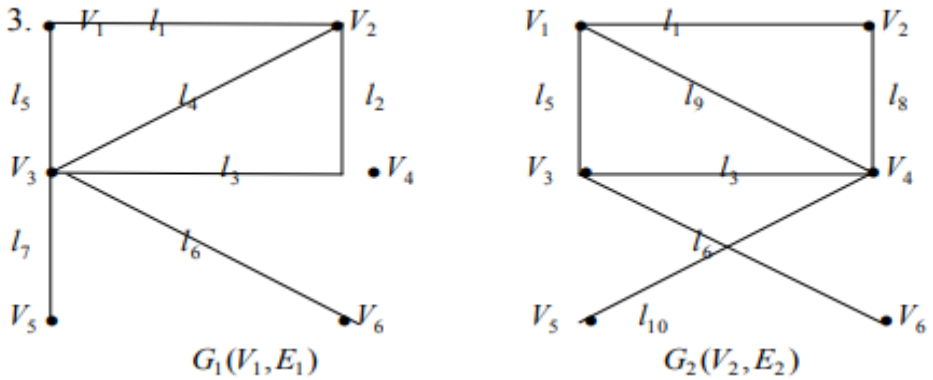
2.2 Матриці інцидентнсті і суміжності;

2.3 Навести приклади шляху, орієнтованого ланцюга, простого ланцюга, контура, цикла і простого цикла

3. Побудуйте граф відношення "$x + y \leq 7$" на множині $M = \{1,2,3,4,5,6\}$. Визначте його властивості.

4. Нехай задано граф $G = (V, E)$; $V = \{1,2,3,4\}$,

$E = \{(1,3), (2,3), (3,4), (4,1), (4,2)\}$. Побудувати діаграму, матриці суміжності і інцидентності.



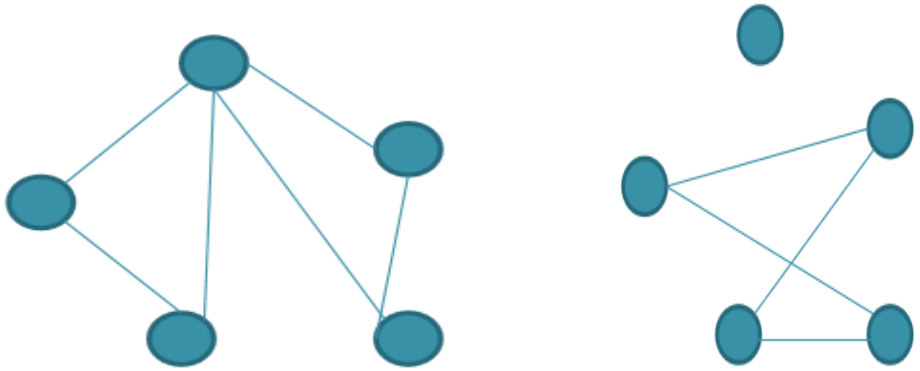
Визначити степінь кожної вершини графів G_1, G_2. Побудувати по два підграфи для кожного графа.

3.3. Операції над графами

3.3.1. Унарні операції над графами

Доповнення.

Доповненням графа $G = (V, E)$ називають граф $\overline{G} = (V, E')$, якщо його ребро (v_i, v_j) входить в V' тоді і тільки тоді, коли воно не входить в V. Іншими словами, дві вершини v_i і v_j суміжні в $\overline{G}$, якщо вони не суміжні в G.

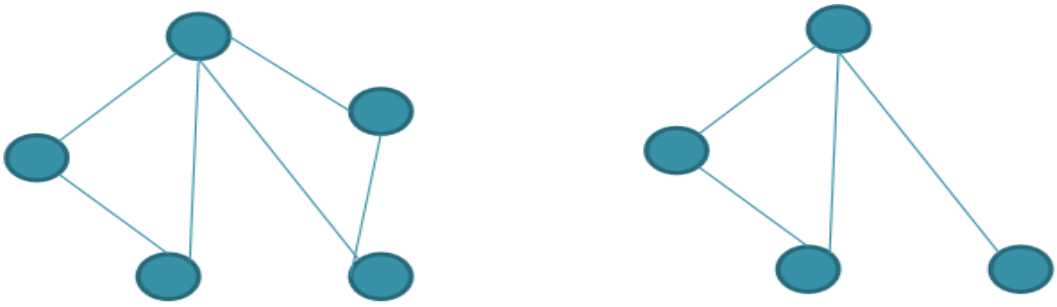


$G = (V, E)$ $\overline{G} = (V, E')$

Рис. 3.49. Операція доповнення

Видалення вершини.

Нехай v_i – вершина графа $G = (V, E)$. $G\text{-}v_i$ – граф, що отриманий видаленням з графа G вершини v_i та ребер інцидентних цій вершині.

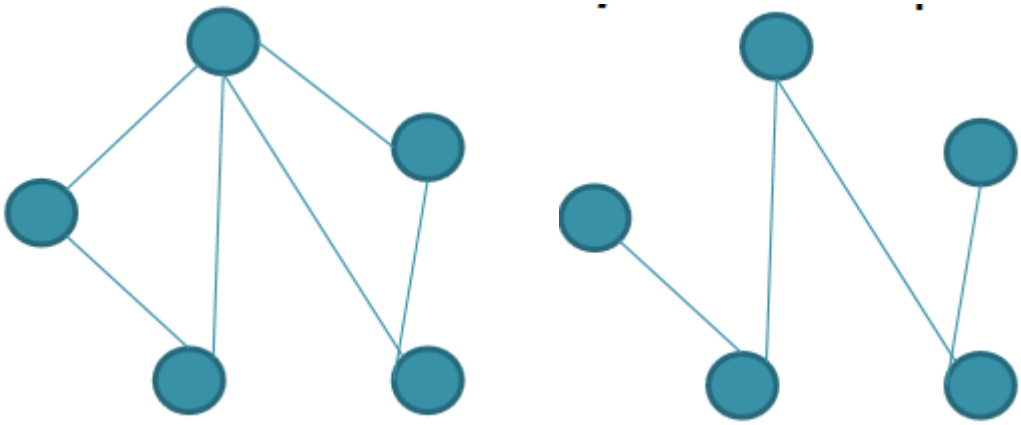


$G = (V,E)$ $G\text{-}v_i$

Рис.3.50. Операція видалення вершини

Видалення ребер.

Нехай e_i – ребро графа G = (V,E). Тоді G-e_i – підграф графа G, який отримано після викидання ребра e_i.

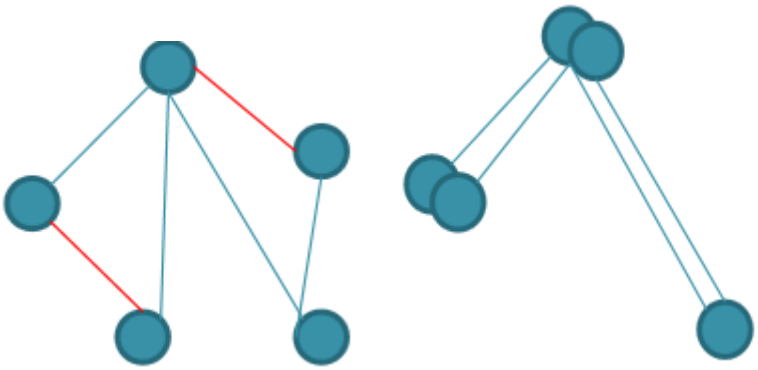


$G = (V, E)$ $G\text{-}e_i$

Рис.3.51. Операція видалення ребра

Стягування.

Стягування – операція видалення ребра e_i і ототожнювання його кінцевих вершин. Граф G називають стягненим до графу H, якщо H можна отримати з G послідовністю стягувань .



G H

Рис.3.52. Операція стягування

Замкнення або ототожнювання.

Кажуть, що пара вершин графу G v_i та v_j **замикаються (ототожнюються)**, якщо вони замінюються новою вершиною, всі ребра графу G інциндентні v_i та v_j, стають інциндентними новій вершині.

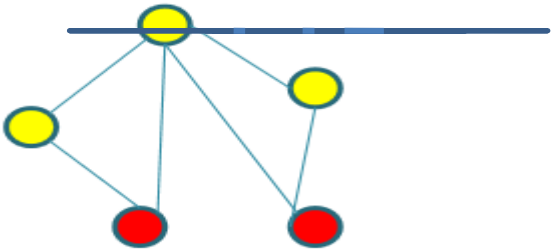


Рис.3.53. Операція замкнення

3.3.2. Бінарні операції над графами

Об'єднання графів.

Об'єднанням графів G_1 та G_2, позначається $G_1 \cup G_2$ є граф $G_3 = (V_1 \cup V_2, E_1 \cup E_2)$ множина його вершин є об'єднанням V_1 та V_2, а множина його ребер є об'єднанням E_1 та E_2 .

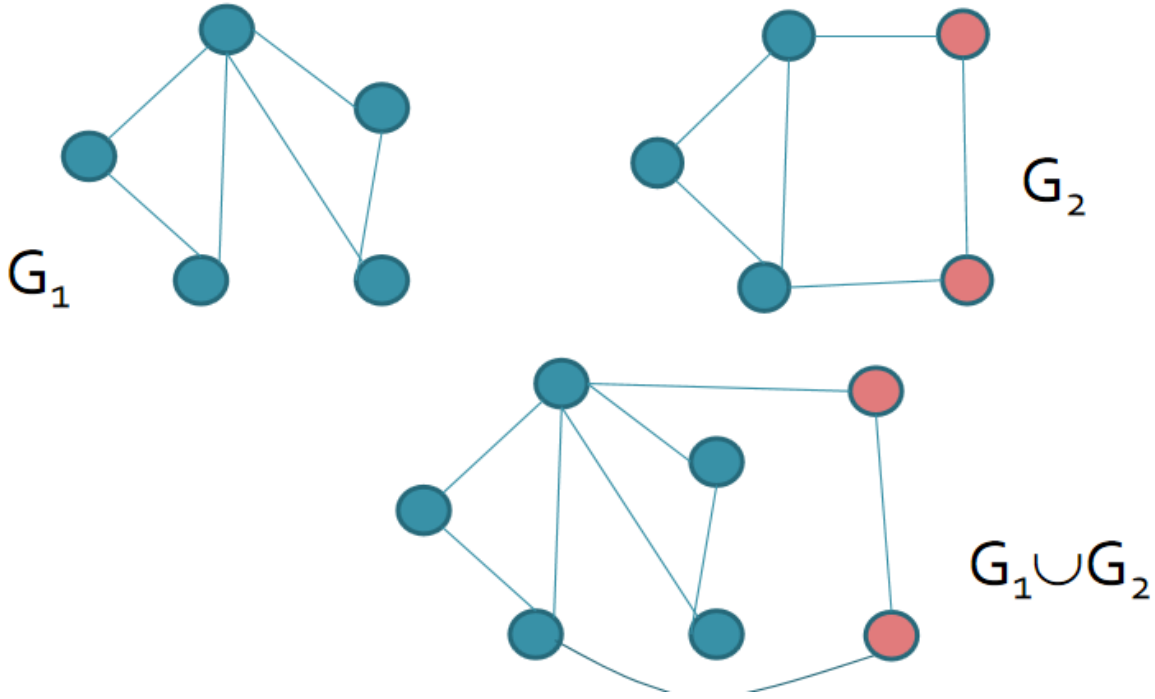


Рис.3.54. Операція об'єднання

Приклад. **3.24.**Об`єднання графів $G_1$і $G_2$,що позначається як$G_1 \cup G_2$, представляє такий граф$G_3 = (X_1 \cup X_2, A_1 \cup A_2)$, що множина його вершин є об`єднанням X_1і X_2, а множина ребер – об`єднанням $A_1$і $A_2$. Граф $G_3$, що отриманий за допомогою операції об`єднання графів G_1і G_2, показаний на рис. 3.55, д, а його матриця суміжності – на рис. 3.55, е.

Матрицю суміжності результуючого графу отримуємо за допомогою операції поелементного логічного ддавання матриць суміжності вхідних графів G_1і G_2.

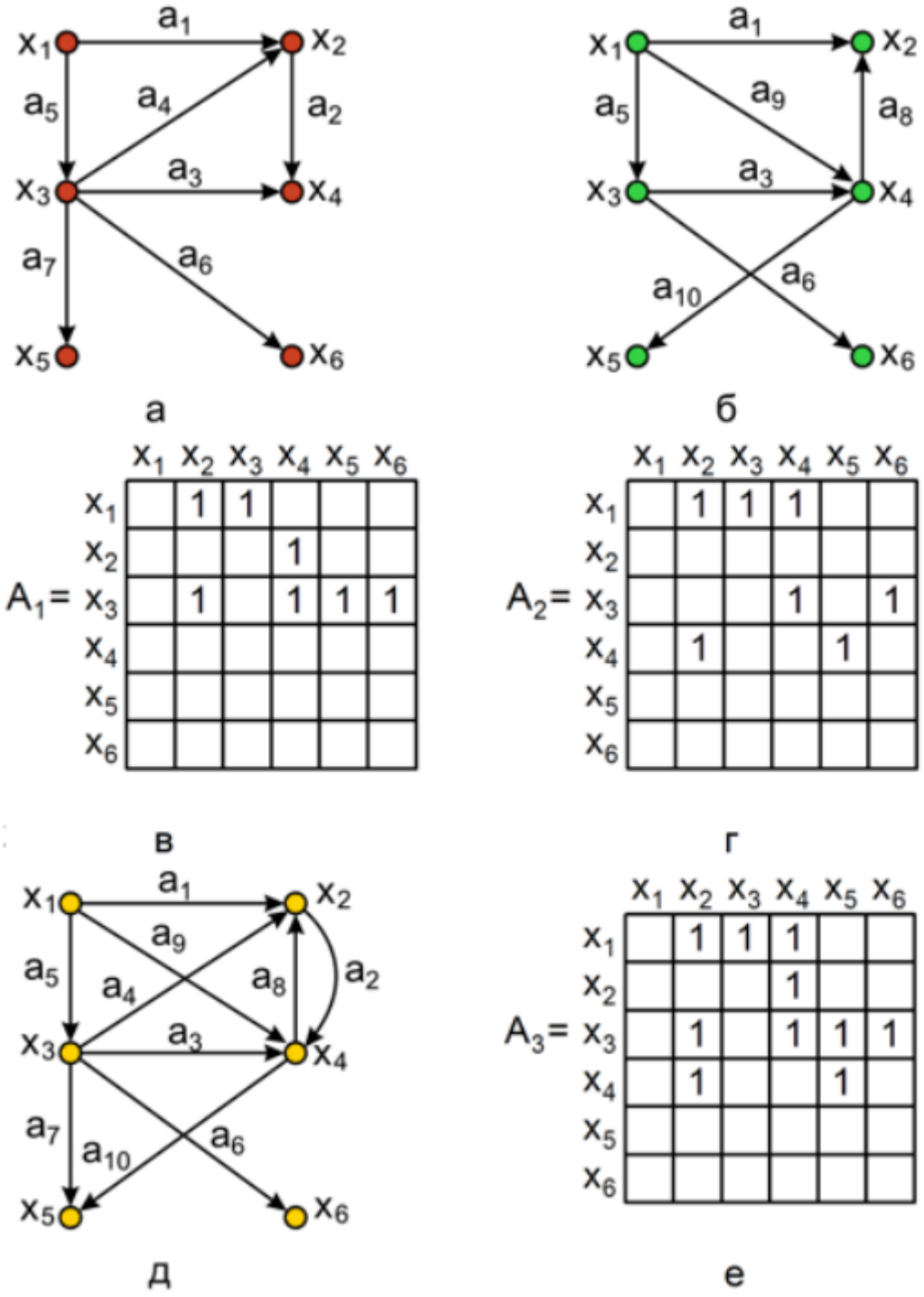


Рис. 3.55. Приклад обєєднання графів G_1 і G_2

Перетин

Перетином графів G_1 та G_2, позначається $G_1 \cap G_2$ є граф $G_3 = (V_1 \cap V_2, E_1 \cap E_2)$, тобто множина його вершин складається лише з тих вершин, які є одночасно в G_1 та G_2, а множина ребер G_3 складається з ребер, які одночасно присутні в G_1 та G_2 .

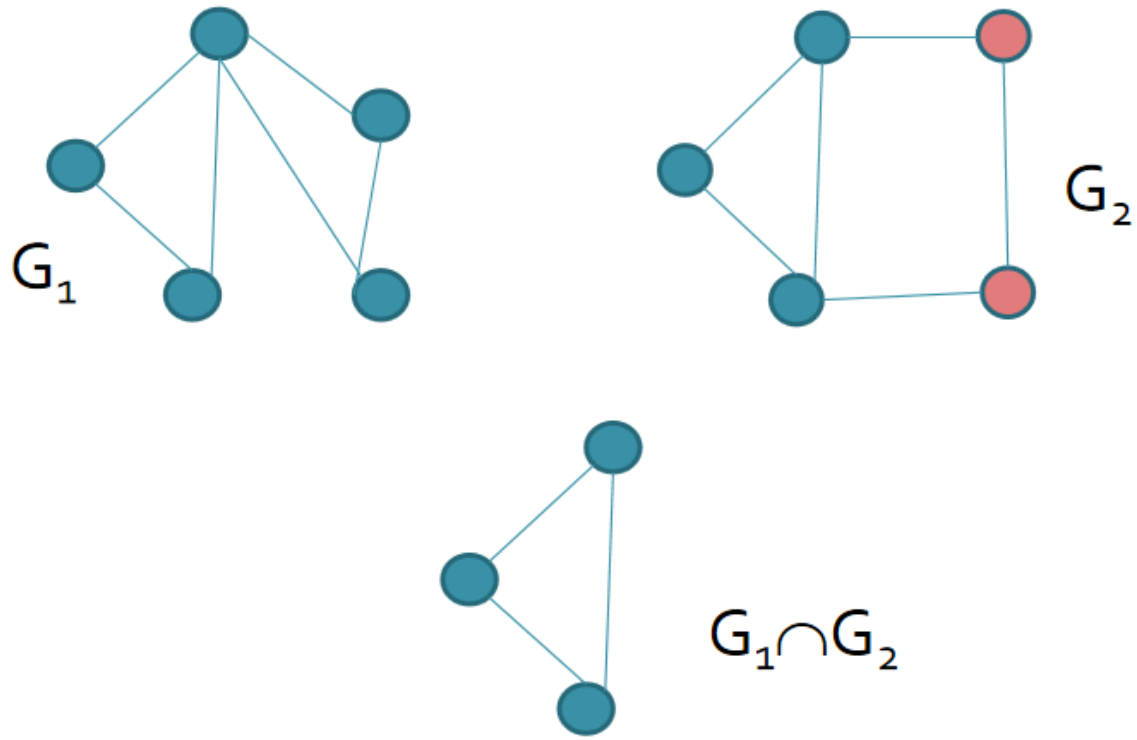


Рис.3.56. Операція перетину

Приклад 3.25.Перетин графів G_1і G_2, що позначається як $G_1 \cap G_2$, представляє собою граф$G_4 = (X_1 \cap X_2, A_1 \cap A_2)$. Таким чином, множина вершин графу G_4складається з вершин, що присутні одночасно в G_1і G_2. Операція перетину графів показана на рис. 3.57, в, а результуючу матрицю суміжності отримуємо за допомогою операціїпоелементного логічного добутку матриць суміжності вхідних графів G_1і G_2,що оказано на рис. 3.57.г.

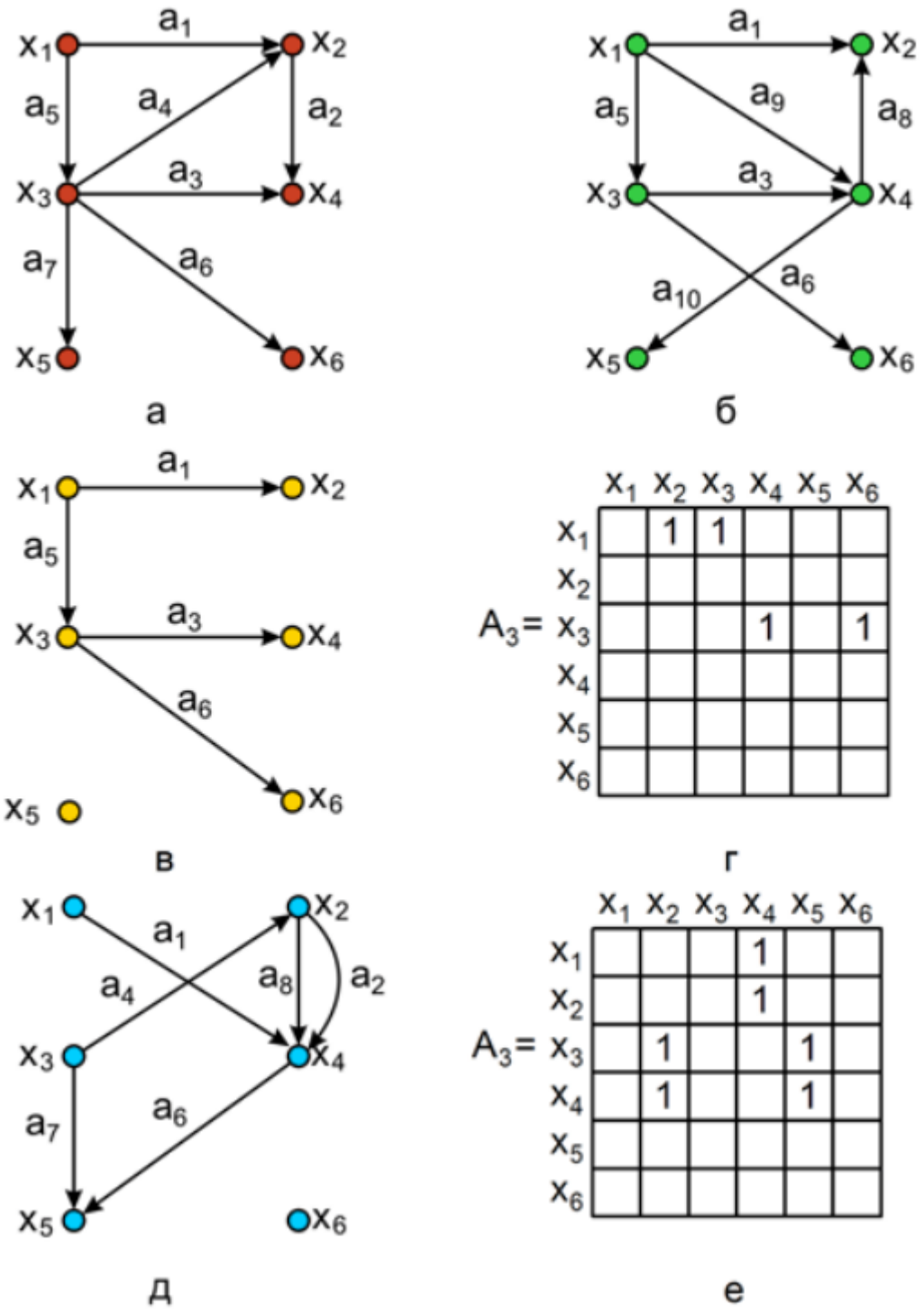


Рис. 3.57. Операція перетину і кільцевої суми: а – граф G_1; б – граф G_2; в – граф $G_1 \cap G_2$; г– матриця суміжності графа $G_1 \cap G_2$; д – граф $G_1 \oplus G_2$; е – матриця суміжності графа $G_1 \oplus G_2$

Кільцева сума

Кільцева сума двох графів G_1 та G_2 позначається $G_1 \oplus G_2$, являє собою граф G_3, який складається з ребер, що присутні або в G_1, або в G_2, але не в обох одночасно.

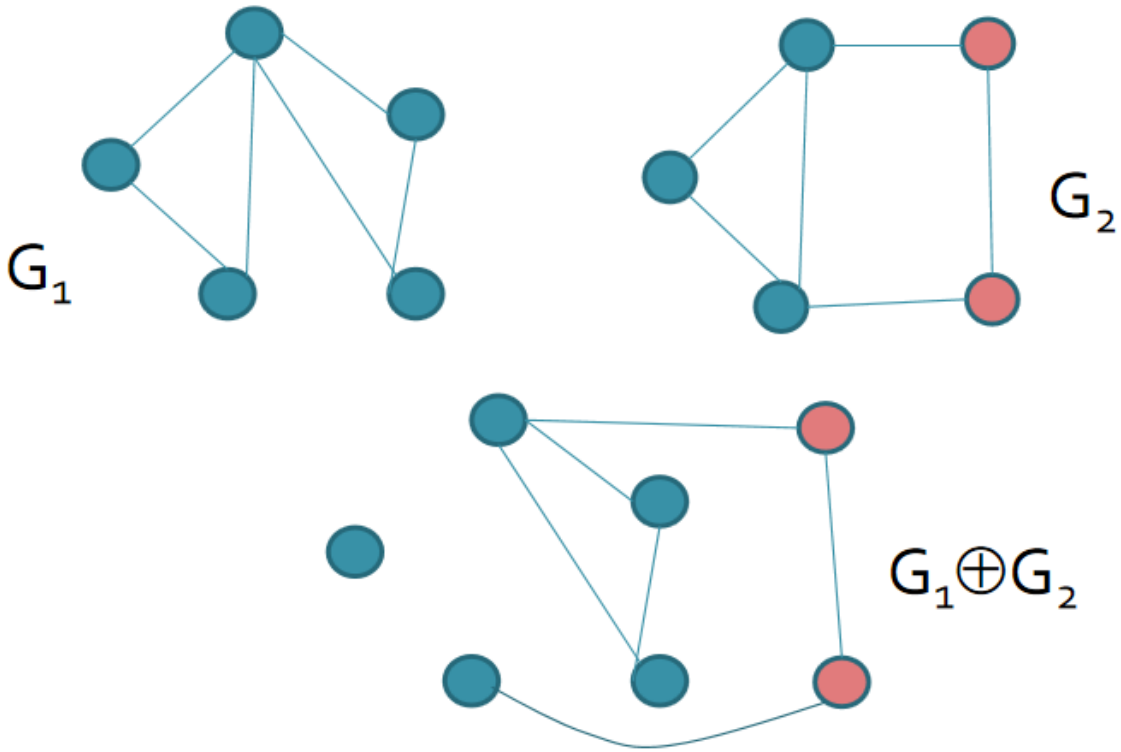
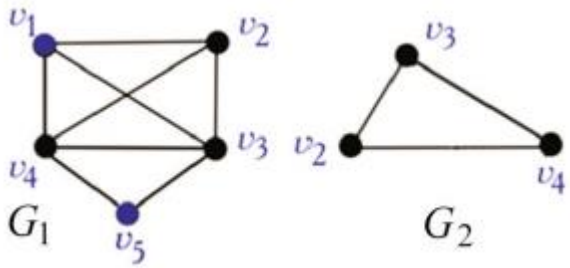


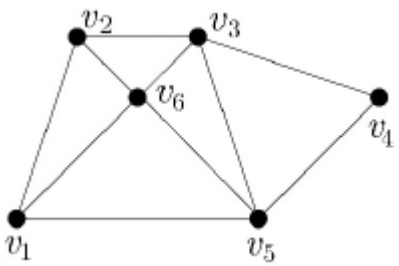
Рис.3.58. Операція кільцевої суми

Обов'язкові завдання.

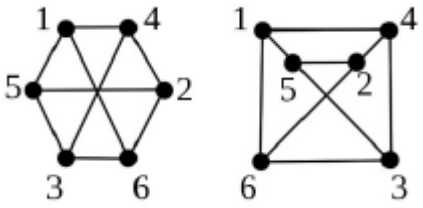
1.Визначте мінімальну кількість та послідовність операцій над графом G_1 для перетворення його у граф G_2.



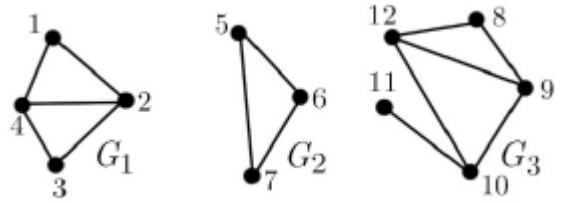
2.Для графа, зображеного на рис., побудувати матриці суміжності та інцидентності. Спираючись на властивості даних матриць, визначити основні характеристики графа.



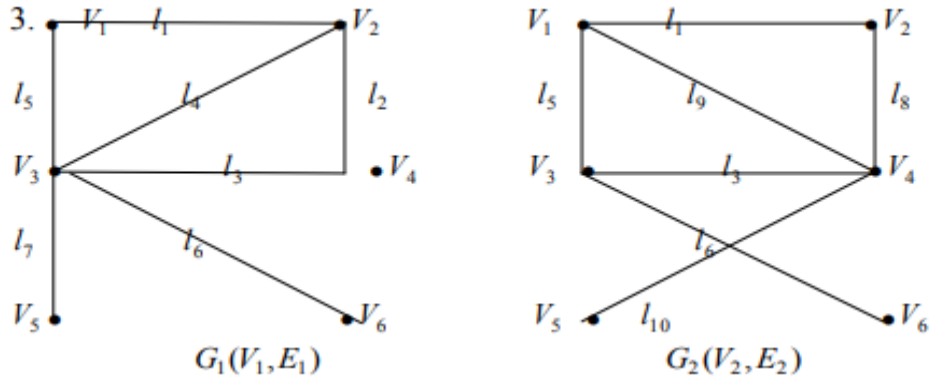
4. Довести ізоморфізм графів, показаних на рис.



5.Побудувати доповнення незв'язного графа, показаного на рис.



6.



Визначити об'єднання, перетин і кільцеву суму двох графів G_1і G_2.

3.4. Графи й бінарні відношення

Відношенню R, заданому на множині V, взаємно однозначно відповідає орієнтований граф $G(R)$ без кратних ребер з множиною вершин V, у якому ребро (v_i, v_j) існує тільки тоді, коли виконано $v_i R v_j$. Представимо на графах деякі бінарні відношення.

Рефлексивність.

Відношення Rна множині V рефлексивне, якщо для кожного елемента $v \in V$справедливе $(v, v) \in R$. На графі це зображується петлею, а матриця суміжності графа з рефлексивними відношеннями містить одиниці на головній діагоналі. Іншими словами, якщо відношення R рефлексивне, то граф $G(R)$ без кратних ребер має петлі у всіх вершинах.

Приклад 3.26. Зобразити графічно та задати матрицею суміжності граф $G(R)$, заданий рефлексивним відношенням R.

Розв'язок.

На рис. 3.59 показаний приклад графа рефлексивного відношення.

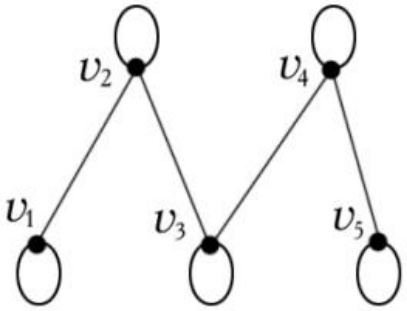


Рис. 3.59. Граф рефлексивного відношення

Головна діагональ матриці суміжності $G(R)$ складається з одиниць.

$$\mathbf{C} = \begin{Vmatrix} \mathbf{1} & 1 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 1 & 0 \\ 0 & 0 & 1 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 1 & \mathbf{1} \end{Vmatrix}$$

Антирефлексивність

*Я*кщо відношення *R*на множині *V*антирефлексивне, то для всіх елементів *v*множини *V*справедливе $(v, v) \notin R$. *Я*кщо *R*антирефлексивне, то граф $G(R)$ без кратних ребер не має петель.

Приклад 3.27. Зобразити графічно та задати матрицею суміжності граф $G(R)$, заданий антирефлексивним відношенням R.

Розв'язок.

На рис. 3.60 показаний приклад графа антирефлексивного відношення.

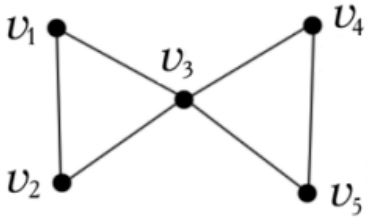


Рис. 3.60. Граф антирефлексивного відношення

Головна діагональ матриці суміжності $G(R)$ складається з нулів

$$\mathbf{C} = \begin{Vmatrix} \mathbf{0} & 1 & 1 & 0 & 0 \\ 1 & \mathbf{0} & 1 & 0 & 0 \\ 1 & 1 & \mathbf{0} & 1 & 1 \\ 0 & 0 & 1 & \mathbf{0} & 1 \\ 0 & 0 & 1 & 1 & \mathbf{0} \end{Vmatrix}$$

Обов'язкові завдання

1. Побудувати граф, який характеризується рефлексивністю, симетричністю і транзитивністю.

2. Зобразити графічно та задати матрицею суміжності граф G (R), заданий антитранзитивним та антисиметричним відношенням R.

3. Нехай дано множину вершин: $V = \{v_1, v_2, v_3, v_4, v_5\}$та множину, на якій задано два відношення: $R_1 = \{(v_1, v_2).(v_2, v_3), (v_4, v_1), (v_4, v_3), (v_4, v_2), (v_3, v_5), (v_1, v_5)\}$,
$R_2 = \{(v_1, v_2).(v_2, v_3), (v_4, v_1), (v_4, v_3), (v_1, v_3), (v_2, v_5), (v_4, v_5)\}$. Побудувати множину ребер графа, задану на об'єднанні відношень R_1 та R_2. Представити графічно графи $G(R_1 \cup R_2)$, $G(R_1)$ та $G(R_2)$.

Розглянемо неорієнтований граф G (V, E), де
$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$.
Побудувати граф G (V, E).

3.5. Зв'язність графа

Зв'язність неорієнтованих графів

Нехай граф ***G* = (*V*, E)** – неорієнтований. Вершина ***a*** називається зв'язаною з вершиною ***b***, якщо існує маршрут, який з'єднує ці вершини. Стверджують, що вершина ***b*** **досяжна** з вершини ***a***.

Граф, будь-яка пара вершин якого є зв'язана, називається **зв'язним**.

Відношення зв`язності рефлексивне (вершина завжди зв`язана сама з собою), симетричне (із зв`язності вершини з вершиною випливає зв`язність вершини з вершиною) і транзитивне (якщо в довільному графі ***G*** вершина ***a*** зв'язана з ***b***, а вершина ***b*** зв'язана з ***c***, то, вочевидь, що ***a*** є зв'язана з ***c***). Таким чином, відношення зв`язності для вершин є відношенням еквівалентності. Тому існує таке розбиття множини вершин графа на попарно неперетинаємі підмножини (класи еквівалентності), що всі вершини в кожній підмножині зв`язані, а вершини з різних підмножин не зв`язані. Кожна така підмножина вершин графа разом з ребрами, інцидентними цим вершинам, створює зв`язний підграф. Отже, неорієнтований граф можна представити єдиним чином у вигляді об`єднання неперетинаємих зв`язних підграфів. Ці підграфи називають **зв`язними компонентами** графа, що розглядається. Зв`язнии граф є своєюй єдиною компонентою зв`язності.

Відношення зв'язності для вершин є **відношенням еквівалентності.**

Отже, існує таке розкладання множини вершин ***V***:

1) $V = \boldsymbol{V_1} \cup \boldsymbol{V_2} \cup \ldots \cup V_p$, $V_i \cap V_j = \emptyset$ за $i \neq j$, на попарно неперерізувані підмножини, що всі вершини ***b***однієї множини V_i є зв'язані між собою, а вершини з різних множин V_i не є зв'язані.

2) $E = \boldsymbol{E_1} \cup E_2 \cup \ldots \cup E_p$, $E_i \cap E_j = \emptyset$ за $i \neq j$.

Тоді, відповідно до 1) та 2) матиме пряме розкладання.

3) $G = G_1 \cup G_2 \cup \ldots \cup G_p$,де $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, … , $G_n = (V_p, E_p)$ – неперерізувані зв'язні підграфи.

Ці підграфи називаються **зв'язними компонентами графа *G***, або **компонентами зв'язності графа*G***.

Число ***p*** – ще одна числова характеристика графа. Для зв'язного графа $p = 1$; якщо граф є незв'язний, то $p \geq 2$.

Теорема. Кожен неорієнтований граф розкладається в єдиний спосіб на пряму суму власних зв'язних компонент.

Зауваження. Якщо певний граф не є зв'язним і розкладається на декілька компонентів, то вивчення цього незв'язного графа можна звести до досліджування окремих його компонентів, які є зв'язні.

Тому у переважній більшості випадків має сенс припускати, що заданий граф є зв'язний.

Граф, зображений на рисунку 3.61, має три компоненти зв'язності.

Через те, що кількість компонентів зв'язності дорівнює кількості зв'язних підграфів графа, наведений граф – тризв'язний (число зв'язності $p=3$).

Рис.3.61. Тризв'язний граф

Зв'язність орієнтованих графів

Зв'язність для орієнтованих графів (орграфів) визначається так само, як і для неорієнтованих, тобто без урахування напрямків дуг.

Якщо в орграфі існує маршрут, що пов`язує вершини і, то кажуть, що вершина **досяжна** з вершини. Будь-яка вершина вважається досяжною з себе самої. Вершина орграфа називається **джерелом**, якщо з неї досяжна будь-яка вершина орграфа.

Зв`язність орієнтованих графів визначається в принципі так само, як і неорієнтованих, тобто без урахування напрямку дуг. Специфічним для орграфа є поняття сильної зв`язності.

Орграф називається сильним (або **сильнозв`язним**), якщо будь-які дві його вершини досяжні одна з одної. Орграф називається **одностороннім** (або **одностороннєзв`язним**), якщо для будь-якої пари його вершин, меншою мірою, одна з них досяжна з іншої. Орграф називається сильно зв'язним, якщо для кожної пари його вершин ***a*** та ***b*** існує шлях з вершини ***a*** до вершини ***b***.

Рис.3. 62. Орграфи (зліва направо): сильний, односторонній і слабкий.

Орграф називається **слабким** (або *слабозв`язним*), якщо зв`язним графом є його неорієнтований дублікат. Оскільки вершина графа досяжна з себе самої, то одновершинний орграф буде одночасно і сильним, і одностороннім, і слабким. Кожний сильный орграф є одностороннім, а кожний односторонній - слабким. Вочевидь, дві будь-які неспівпадаючі вершини сильного орграфа належать деякому контуру.

В деяких задачах істотною є вимога сильної зв`язності графа. Наприклад, граф, що представляє план міста з одностороннім рухом по деяким вулицям, повинен бути сильно зв`язаним, так як, в протилежному випадку, знайдуться вершини (площі і перехрестя), між якими не можна було б проїхати по місту без порушення правил руху.

Маршрут, що містить всі вершини орграфа, називається **остовним**.

Теорема. Орграф є сильним тоді і тільки тоді, коли в ньому є остовний контур, є одностороннім тоді і тільки тоді, коли в ньому є остовний шлях.

Відношення взаємної досяжності вершин орграфа рефлексивно, симетрично і транзитивно. Якщо це відношення еквівалентності, то воно розбиває множину вершин орграфа на класи еквівалентності, об`єднуючи в один клас всі вершини, що досяжні одна з одної. Вершини, що входять в такі класи, разом з дугами, їм інцидентними, обидві кінцеві вершини яких належать цьому ж класу, створюють підграфи, що називаються сильними (або сильнозв`язними) компонентами орграфа.

Орграф називається незв`язним, коли його неорієнтований дублікат не є зв`язним графом.

Орграф, зображений на рис. 3.38, має чотири сильні компоненти з множинами вершин $\{v_1, v_2, v_3, v_4\}$, $\{v_5, v_6, v_8\}$, $\{v_7\}$, $\{v_9\}$. В орграфі можуть бути дуги, що не входять в жодну з його сильних компонент, наприклад, дуги (v_3, v_5), (v_4, v_6), (v_9, v_2) і (v_9, v_8)у орграфа на рис. 3. 63.

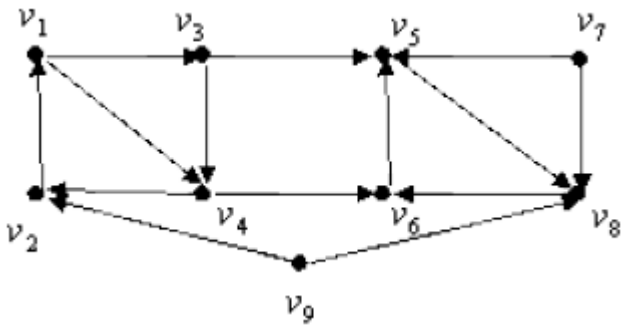


Рис.3.63. Чотиризв’язний орграф

3.5.1. Вершинна зв`язність

Розглянемо граф, вершини якого відповідають деяким технологічним об`єктам, а ребра показують, які об`єкти можуть взаємодіяти або безпосередньо один з одним, або опосередковано через інші об`єкти. Технологічна система, що представлена цим графом, вважається функціонуючою, якщо кожна пара її об`єктів зв`язана між собою. В цьому випадку система повинна мати зв`язний граф. Важливою характеристикою системи є її надійність, під якою зазвичай розуміють здібність системи функціонувати при виході зі строю одного або декількох об`єктів і (або) порушенні зв`язку між деякими з них. Вочевидь, що менш надійною слід вважати ту систему, яка перестає функціонувати при виході зі строю меншої кількості її елементів. Оцінити степінь надійності такої системи можуть допомогти ті поняття, про які згадувалося вище і які зараз будуть визначені.

Визначення. Числом вершиною зв`язності** (або просто **числом зв`язності) $\chi(G)$ графа G називається число, що дорівнює найменшому числу вершин, видалення яких приведе до незв`язного або одновершинного графу.

Граф G, представлений на рис. 3.64, зв`язний, але він перестане бути зв`язним, якщо видалити вершину 4. Тому $\chi(G) = 1$.

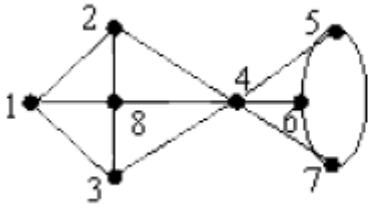


Рис.3.64

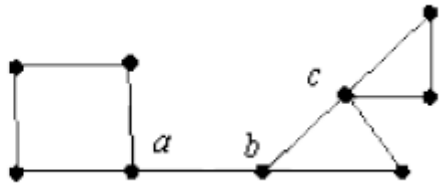


Рис.3.65

Можна порушувати зв`язність графа, видаляючи деякі його ребра (дуги). У графа (рис. 3.64) для цього треба видалити не менше трьох ребер. Наприклад, граф розпадеться на дві компоненти після видалення ребер 4&5, 4&6, 4&7.

3.5.2. Реберна зв’язність

**Визначення. Числом реберної зв`язності** $\boldsymbol{\lambda(G)}$ графа $G$ називається число, яке дорівнює найменшому числу ребер, видалення яких приведе до незв`язного графу. Число реберної зв`язності одновершинного графу дорівнює нулю.

Вище показано, що для графа G (рис. 3.61) $\boldsymbol{\lambda(G) = 3}$.

Вершина v графа G називається **точкою зчленування**, якщо граф G-v, отриманий після операції видалення у графа G вершини v, має більше компонент

зв`язності, ніж сам граф $G$. Якщо $G$ зв`язний і v точка зчленування, то G-v не зв`язний.

Ребро e графа G називається **мостом**, якщо його видалення збільшує число компонент зв`язності графа.

Таким чином, точки зчленування і мости - це свого роду «вузькі місця» графа. Граф на рис. 3.65 має три точки зчленування це вершини a, b, c і один міст - ребро $a\&b$.

Повертаючись до технологічної системи, річ, про яку йшлася спочатку, зазначимо, що число вершинної зв`язності і число реберної зв`язності її графа відображають чуттєвість системи до пошкоджень, а точки зчленування і мости графа системи вказують на найбільш вузькі місця системи.

Граф називається **нероздільним**, якщо він зв`язний і не має точок зчленування. Граф, що має хоча б одну точку зчленування, є роздільним і називається **сепарабельним.** Він розбивається на блоки, кожний з яких представляє собою максимальний нероздільний
підграф.

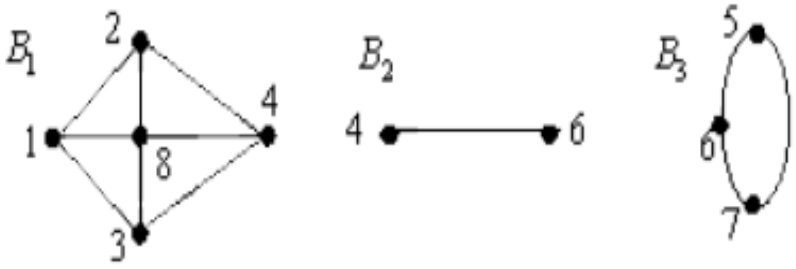


Рис.3.66. Блоки B_1, B_2, B_3 графа з рис.3.64.

Якщо $\delta(G)$є мінімальна степінь вершин графа G, то вочевидь, що $\lambda(G) \leq \delta(G)$, оскільки видалення всіх ребер, інцидентних даній вершині, приведе до збільшення числа компонент зв`язності графа.

Теорема. *Для будь-якого графа G справедливі нерівності*: $\chi(G) \leq \lambda(G) \leq \delta(G)$.

Граф G називається *k-зв`язним*, якщо $\chi(G) \geq k$, *реберно- k-зв`язним, якщо* $\lambda(G) \geq k$.

Граф, зображений на рис. 3.64, 1-зв`язний і реберно-3-зв`язний.

Отже, підсумовуючи: Означення. Дві вершини v і w називаються зв’язаними, якщо існує маршрут з кінцями v та w. Граф називається зв'язним, якщо будь-яка пара його вершин є зв’язаною. Якщо граф не є зв'язним, то він називається незв'язним.

Визначення. Зв’язністю графа називається мінімальна кількість вершин, вилучення яких приводить до утворення незв'язного графа.

Приклад 3. 26. Граф, наведений на рис. 3.67, є зв’язним, тому що будь-яка пара його вершин зв’язана. Але яка його зв’язність? Тобто, якою є мінімальна

кількість вершин його графа, вилучення яких приводить до утворення незв'язного графа?

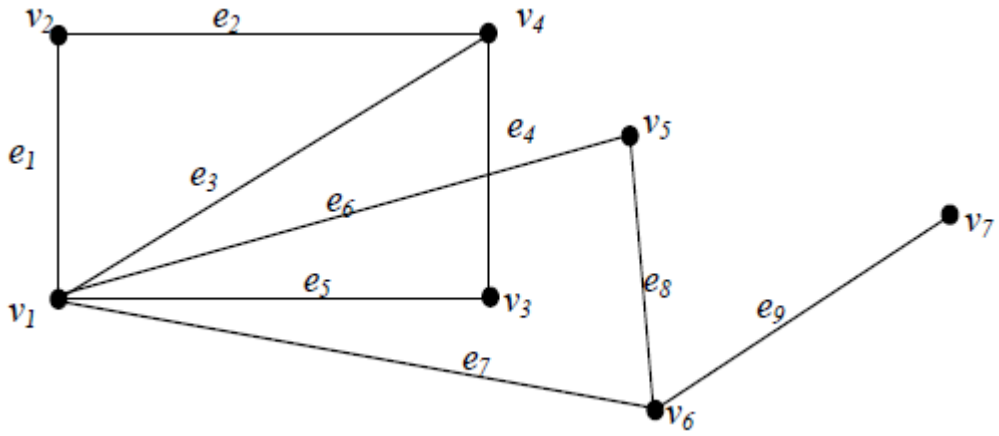


Рис. 3.67. Зв'язний граф

Очевидно, що зв`язність дорівнює одиниці, тому що вилучення однієї вершини v_1
приводить до утворення незв`язного графа.

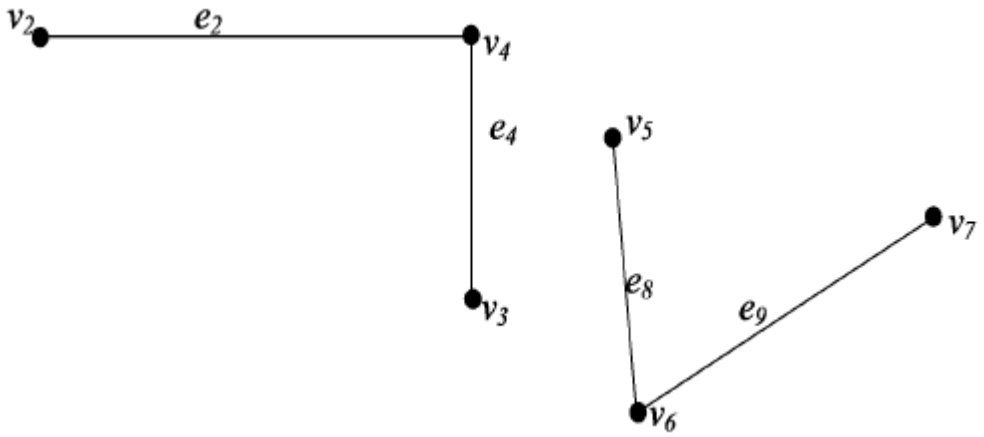
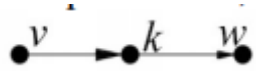


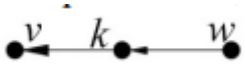
Рис.3.68. Граф, після вилучення вершини v_1

3.6. Досяжність і контрдосяжність вершини в графах

Визначення. Вершину *w* графа *D* (або орграфа) називають досяжною з вершини *v*, якщо *w=v*, або існує шлях з *v* у *w* (маршрут від *v* у *w*).



Визначення. Вершину *w* графа *D* (або орграфа) називають контрдосяжною з вершини *v*, якщо існує шлях з *w* у *v* (маршрут від *w* у *v*).



Матриця відстаней графа

Нехай

задано зв'язний граф *G* =(*V*, *E*). Відстанню поміж двома вершинами *v* та *w* графа *G* називається довжина найкоротшого ланцюга, який зв'язує ці вершини.

Відстань між вершинами *v* та *w* графа *G* позначається через *G d* (*v*, *w*).
Аксіоми метрики:
1) *d* (*v*, *w*)> 0; *d* (*v*, *w*) = 0 <=> *v* = *w*;
2) *d* (*v*, *w*) = *d* (*v*, *w*);
3) *d* (*v*, *z*) = *d* (*v*, *w*) + *d* (*w*, *z*).

Діаметром графа називається величина *d*(*G*), де максимум береться за всі ма можливими парами вершин графа: $d(G) = \underset{v,w}{max} d(G)$

Визначимо для кожної вершини x графа *G* величину *r*(*x*), тобто відстань в ід вершини *x* до найвіддаленішої від *x* вершини графа:

$$r(x) = \underset{y}{max} d(x, y).$$

Мінімум цієї величини за всіма вершинами
графа називається **радіусом** графа *G*:

$$r(G) = \underset{x}{min} r(x) = \underset{x}{min} \underset{y}{max} d(x, y)$$

Вершина x_0, якій досягається цей мінімум, називається **центральною**.

Приклад 3.27. Знайти діаметр і радіус для графа, зображеного на рис.3.69.

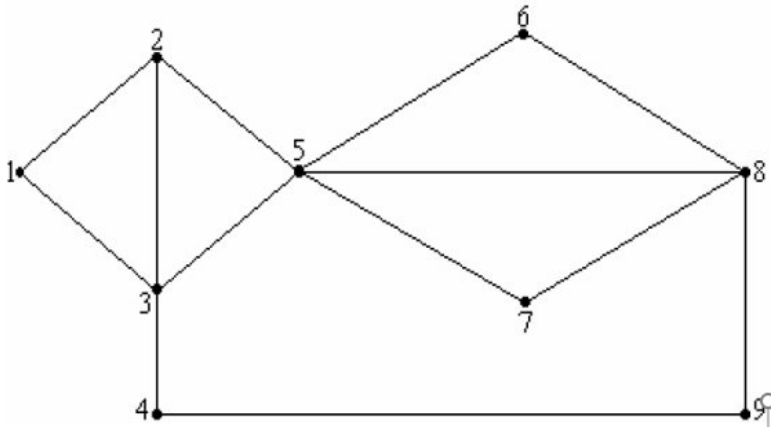


Рис.3.69.

Для розв’язання цієї задачі зручно попередньо обчислити так звану мат рицю відстаней поміж вершинами графа.
У даному разі це буде матриця розміром 9×9, елемент якої, що стоїть на місці *(i*, *j*), дорівнює відстані від вершини і до вершини *j*.

$$\begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 3 & 3 & 3 & 3 \\ 1 & 0 & 1 & 2 & 1 & 2 & 2 & 2 & 3 \\ 1 & 1 & 0 & 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 2 & 1 & 0 & 2 & 3 & 3 & 2 & 1 \\ 2 & 1 & 1 & 2 & 0 & 1 & 1 & 1 & 2 \\ 3 & 2 & 2 & 3 & 1 & 0 & 2 & 1 & 2 \\ 3 & 2 & 2 & 3 & 1 & 2 & 0 & 1 & 2 \\ 3 & 2 & 2 & 2 & 1 & 1 & 1 & 0 & 1 \\ 3 & 3 & 2 & 1 & 2 & 2 & 2 & 1 & 0 \end{pmatrix} \begin{matrix} 3 \\ 3 \\ 2 \\ 3 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \end{matrix}$$

За визначенням, діаметр графа дорівнює найбільшому елементові матриці відстаней. Отже, $d = 3$.

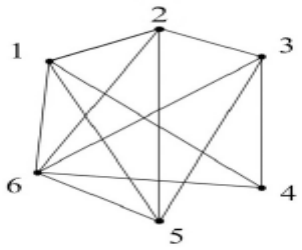
Для знаходження радіуса відшукаємо в кожному рядку найбільше число; ці числа виписано праворуч від матриці відстаней. Найменші з них дають значення радіуса $r = 2$. Вершини 3-тя та 5-та є центральними.

Обов'язкові завдання

1.Нехай дано зв'язний граф $G(V, E)$. Визначити величину загального степеня вершин графа за умови, що $|V| = 5$ і $|E| = 8$. Побудувати варіант графа з максимально можливим степенем вершини.

2. Позначимо через n – кількість будинків і m – кількість доріг. Будинки пронумеровані від 1 до n. Кожна дорога визначається двійкою чисел – двома номерами будинків – кінців дороги. У кожному будинку живе по одній людині. Знайти точку – місце зустрічі всіх людей, від якої сумарна відстань до всіх будинків буде мінімальною за умови, що довжина усіх доріг однакова.

3. Побудувати підграф, правильний підграф та остовний підграф графа, показаного на рис.



4. Побудувати незв'язний граф, який містить 12 вершин, 3 компоненти і мінімально можливу кількість вершин.

3.7. Цикломатика графів

Цикломатика – це вивчення циклів у графі.

З усієї сукупності циклів даного графа можна відокремити цілковито пев ну кількість незалежних (базисних) циклів, решту здобути з базисних циклів за допомогою спеціальної операції додавання.

Приклад такого додавання:

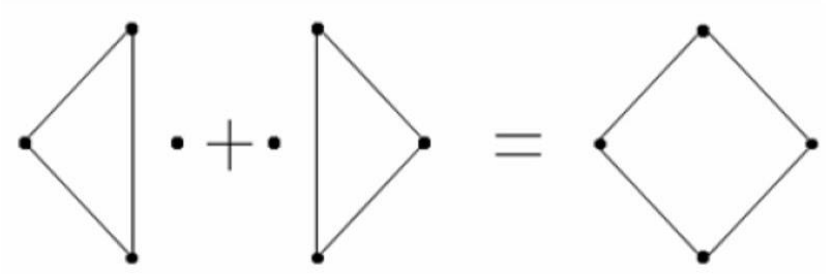


Рис. 3.70. Приклад додавання базисних циклів

3.7.1. Циклові ребра і перешийки

Нехай задано граф G = (V, E). Ребро графа, через яке проходить хоча б один цикл, назвемо цикловим ребром. Ребро, яке не входить до жодного циклу, називається **перешийком.**

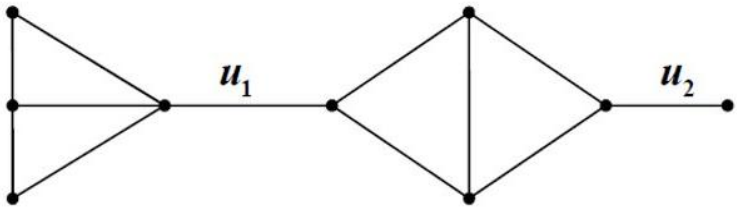


Рис. 3.71. Граф з перешийками і цикловими ребрами

Ребра u_1 та u_2 – перешийки, решта ребер − циклові.

Лема. При вилученні зі зв'язного графа циклового ребра він залишається зв'язним. При вилученні зі зв'язного графа перешийка граф розпадається на дві компоненти зв'язності.

Д о в е д е н н я:

Якщо ребро $\{a, b\}$, яке вилучається, є цикловим, то після його вилучення Вершини a та b, як і раніше, можна з'єднати в ланцюг – залишком циклу, до якого входило ребро $\{a, b\}$. Звідси випливає, що і кожні дві вершини графа після вилучення ребра {a, b} залишаються зв'язаними ланцюгом. Й навпаки, якщо $\{a, b\}$ – перешийок, то після його вилучення вершини a та b не можна зв'язати ланцюгом, інакше цей ланцюг разом з ребром $\{a, b\}$ утворить цикл у вихідному графі. З іншого боку, кожна вершина залишається зв'язаною чи то з вершиною a, чи то з вершиною b.

Н а с л і д о к. При вилученні з графа циклового ребра кількість зв'язних компонентів графа не змінюється. При вилученні перешийка кількість зв'язних компонентів графа збільшується на одиницю.

3.7.2. Цикломатичне число

Нехай задано граф $G = (V, E)$, $|V|=n$, $|E|=m$; p – кількість компонент зв'язності графа. Величина $\lambda = m - n + p$ називається **цикломатичним числом графа.**

Теорема. Для кожного графа цикломатичне число є невід'ємне, тобто $\lambda \geq 0$.

Доведення:

Вилучаємо з графа по одному ребру, і відстежуємо змінювання величини λ.

Параметри вихідного графа позначимо m, n і p. Ті самі величини після вилученн я ребра позначимо через m', n' та p'.

У процесі вилучення ребер можливі два випадки:

а) ребро, яке вилучається – циклове. $m' = m - 1$, $n' = n$, $p' = p$;

$\lambda' = m' - n' + p' = m - 1 - n + p = \lambda - 1$.

б) ребро, яке вилучається − перешийок. $m' = m - 1$, $n' = n$, $p' = p + 1$.

$\lambda' = m' - n' + p' = m - 1 - n + p + 1 = m - n + p = \lambda$.

Отже, при вилученні ребра величина λ або не змінюється, або зменшується на одиницю.

Після вилучення всіх ребер дістанемо порожній граф, у якому $m_0 = 0$, $n_0 = n$, $p_0 = n$, тобто $\lambda_0 = m_0 - n_0 + p_0 = n - n = 0$.

Отже, у вхідного графа $\lambda \geq 0$.

Зауваження. Доведення теореми свідчить про зв'язок цикломатичного числа з наявністю циклів у графі.

Якщо $\lambda > 0$, то у графі є принаймні один цикл.

При вилученні циклового ребра деякі цикли розриваються − і це призводить до зменшення λ. Якщо продовжувати вилучення ребер, то, зрештою, розриваються всі цикли − і λ стає дорівнюваним 0.

Після цього λ вже не змінюється, тому що всі ребра стали перешийками.

Цикломатичне число графа вказує кількість ребер, які необхідно видалити, щоб отримати дерево (для зв'язного грфа) або ліс (для незв'язного графа), тобто досягнти відсутності циклів.

Цикломатичне число графа дорівнює максимальній кількості незалежних циклів.

Знання цикломатичного числа корисне при аналізі топології електросхем, а також для задач конструкторського проектування на ЕОМ.

Приклад 3.28. Визначити цикломатичне число графа G (V, E), показаного на рис. 3.72.

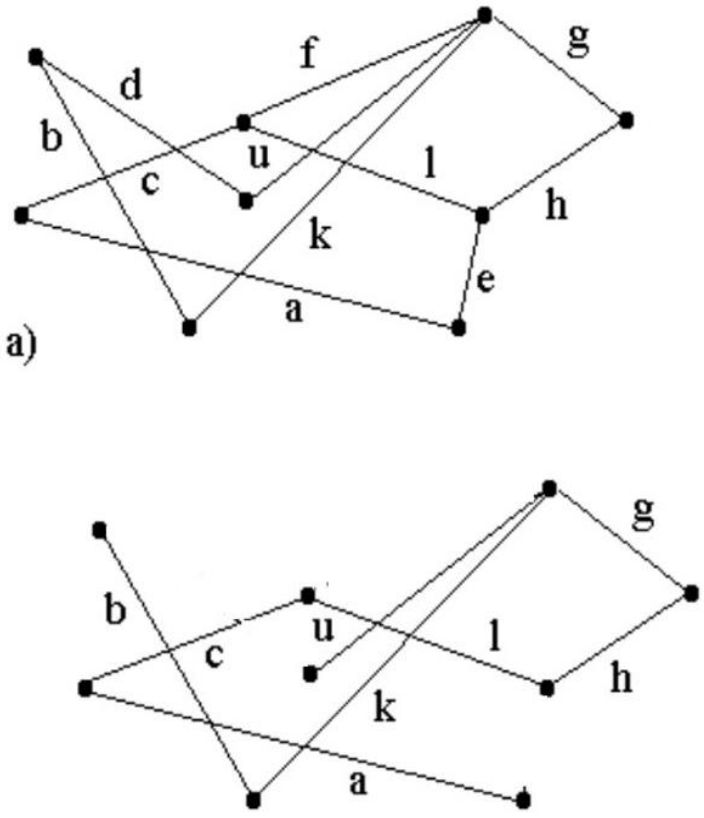


Рис.3.72. а) Граф $G(V,E)$, b) Граф $G(V,E)$ після вилучення циклових ребер
Розв'язок: m=11; n=9; p=1; λ=11-9+1=3.

Приклад 3.29. Визначити цикломатичне число графа G (V, E), показаного на рис. 3.73.

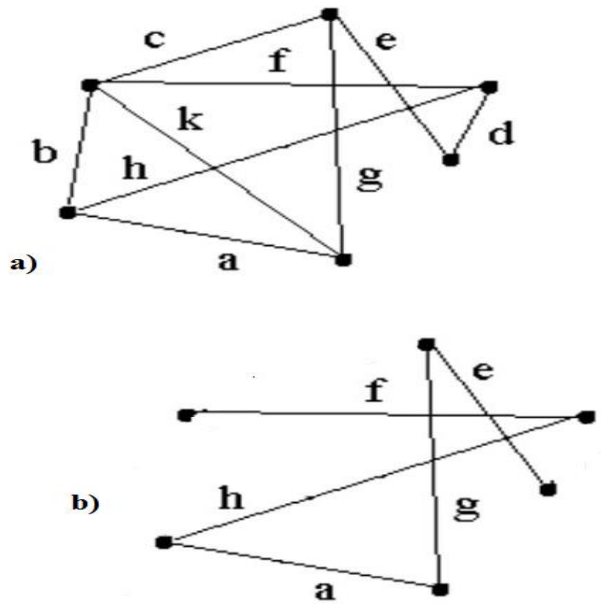


Рис.3.73. а) Граф $G(V,E)$, b) Граф $G(V,E)$ після вилучення циклових ребер
Розв'язок: m=9; n=6; p=1; λ=9-6+1=4.

Обов'язкові завдання

1. Визначити цикломатичне число зв'язного графа G (V, E), якщо граф характеризується множиною вершин $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, а потужність множини ребер $|E| = 12$.

2. Нехай дано граф G (V, E), де $V = \{v_1, v_2, v_3, v_4, v_5\}$.
$E = \{(v_1, v_2), (v_5, v_2), (v_3, v_4), (v_1, v_4), (v_1, v_5), (v_3, v_2), (v_4, v_5), (v_4, v_2)\}$.
Скільки ребер потрібно видалити, щоб граф став деревом? Визначити підмножини ребер, які можуть бути видалені без втрати зв'язності графа.

3.8. Дерева та їхні властивості, ліс, цикли

3.8.1. Визначення дерева. Властивості дерев

Дерево – це сукупність елементів, що називаються вузлами (один з яких корінь), та відношень („батьківських"), що утворюють ієрархічну структуру вузлів. Вузли можуть бути елементами будь-якого типу (літерами, рядками, числами).

Піддерево, корінь якого знаходиться в лівому (правому) нащадку вершини, називається лівим (правим) піддеревом цієї вершини.

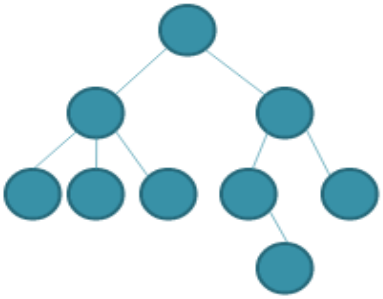


Рис. 3.74. Приклад дерева

Висота вузла дерева - це довжина самого довгого шляху з цього вузла до будь-якого листа.

Глибина вузла – це довжина шляху від кореня до цього вузла.

Степінь вузла – це кількість дуг, що з нього виходить.

Степінь дерева дорівнює максимальному степеню вузла, що входить у дерево.

Листя в дереві - це вузли, що мають степінь нуль.

Бінарне дерево – це дерево степінь якого дорівнює два . Дерева, степінь яких більше двох, називаються **розгалуженими**.

Повне бінарне дерево - це дерево для якого на всіх рівнях менше чим n вузли мають степінь 2, а на рівні n – степінь 0.

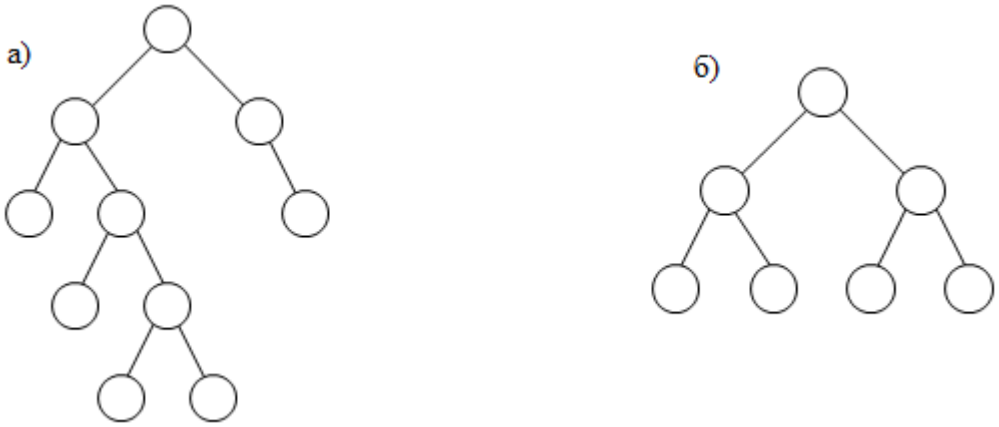


Рис.3.75. Бінарні дерева. а) неповне бінарне дерево; б) повне бінарне дерево

Строго бінарне дерево складається тільки з вузлів, що мають степінь 2 або 0.

Нестрого бінарне дерево містить вузли зі степенем 1.

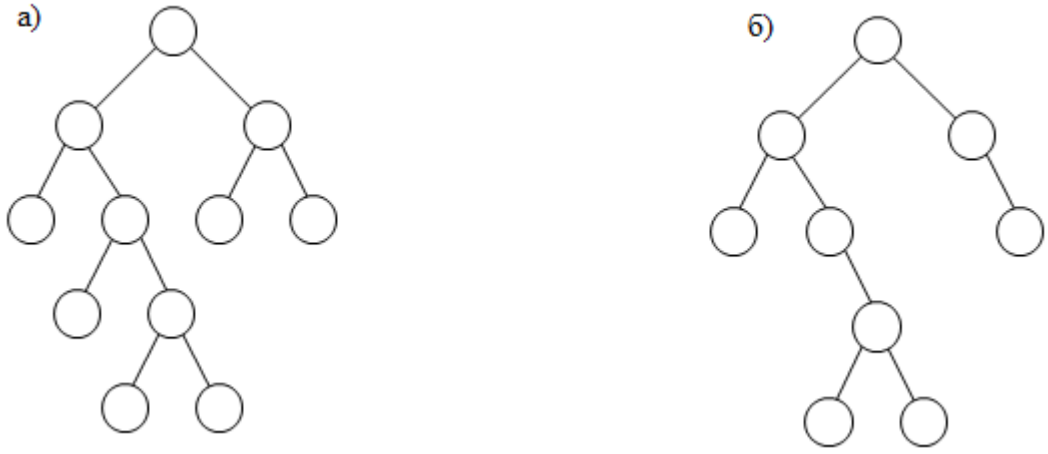


Рис.3.76. Бінарні дерева. а) строго бінарне дерево; б) нестрого бінарне дерево

Бінарне дерево пошуку (binary search tree – BST) – це бінарне дерево в якому:

- Кожен вузол має не більше двох нащадків;
- Кожен вузол має ключ (key) і значення (value);
- Ключі всіх вузлів лівого піддерева менші значення ключа батьківського вузла;
- Ключі всіх вузлів правого піддерева більші значення ключа батьківського вузла.

Використовуються для реалізації словників та множин.

Алгоритм вставки елемента

1. Починаємо з кореня.
2. Якщо елемент < об’єкта в вершині, переходимо до лівого сина.
3. Якщо елемент > об’єкта в вершині, переходимо до правого сина.
4. Повторюємо кроки 2 і 3, доки не досягнемо вершини, яка не визначена.
5. Якщо досягнута вершина не визначена, то визначаємо вершину і вставляємо елемент.

Приклад 3.30. Побудувати дерево пошуку: 54, 37, 63, 21, 46, 73, 59, 12, 40, 60.

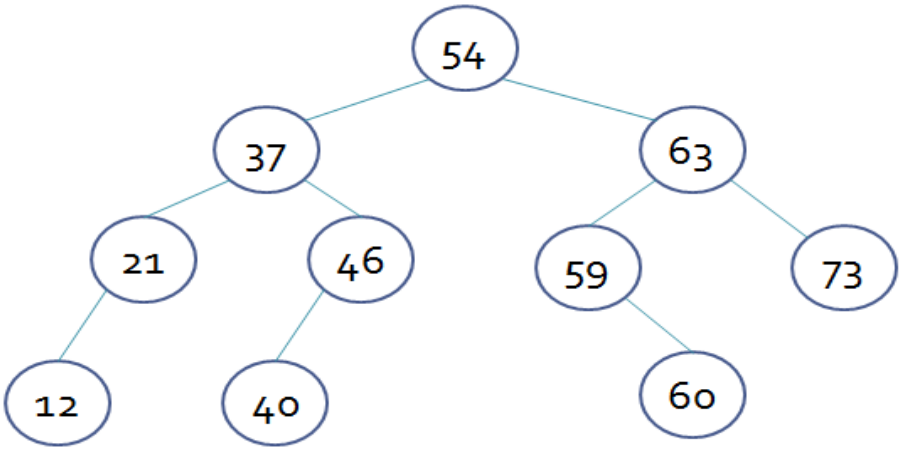


Рис.3.77. Бінарне дерево пошуку

Алгоритм пошуку елемента

1. Починаємо з кореня.
2. Якщо елемент < об’єкта в вершині, переходимо до лівого сина.
3. Якщо елемент > об’єкта в вершині, переходимо до правого сина.
4. Якщо елемент = об’єкту в вершині, то елемент знайдено; виконуємо відповідні дії і виходимо.
5. Повторюємо кроки 2, 3 і 4 доки не досягнемо вершини, яка не визначена.
6. Якщо досягнута вершина не визначена і в дереві немає шуканого елемента, то виконуємо відповідні дії і виходимо.

Алгоритм видалення елемента

1. Якщо вершина v_0 не має синів, просто видаляємо її.
2. Якщо вершина v_0 має одного сина, видаляємо v_0 і заміняємо її сином.
3. Якщо v_0 має двох синів, знаходимо правого сина v_1 вершини v_0, а потім знаходимо лівого сина вершини v_1 (якщо він існує). Продовжуємо вибирати лівих синів кожної знайденої вершини, доки не знайдеться така вершина v, у якої не буде лівого сина. Замінимо v_0 на v і зробимо правого сина вершини v лівим сином батька вершини v.

Приклад 3.31.

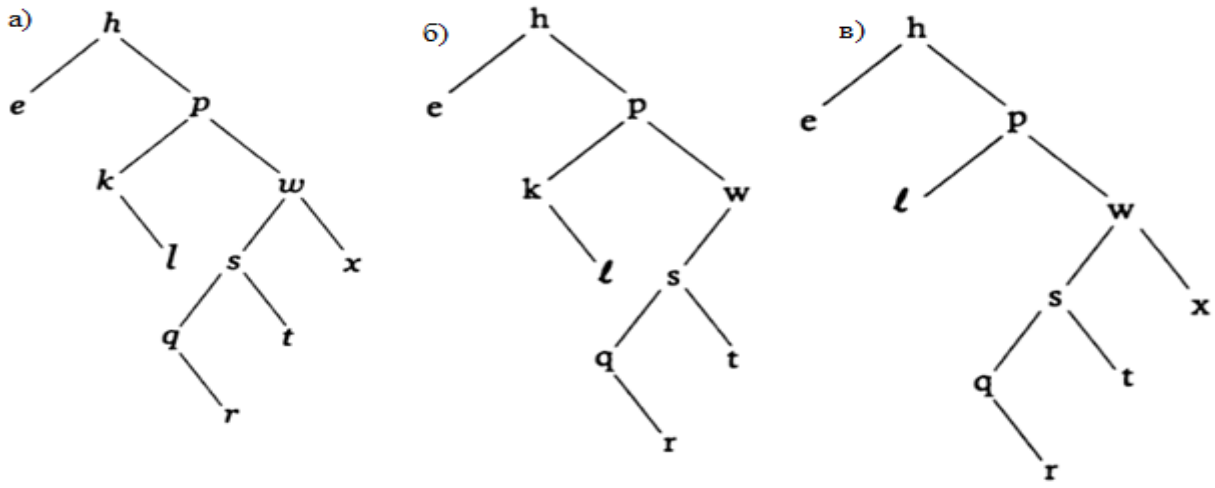


Рис.3.78. а) задане дерево; б) дерево після видалення вершини *x*; в) дерево, після видалення вершини *k*

Якщо v_0 має двох синів, знаходимо правого сина v_1 вершини v_0, а потім знаходимо лівого сина вершини v_1 (якщо він існує). Продовжуємо вибирати лівих синів кожної знайденої вершини, доки не знайдеться така вершина *v*, у якої не буде лівого сина. Замінимо v_0 на *v* і зробимо правого сина вершини *v* лівим сином батька вершини *v*.

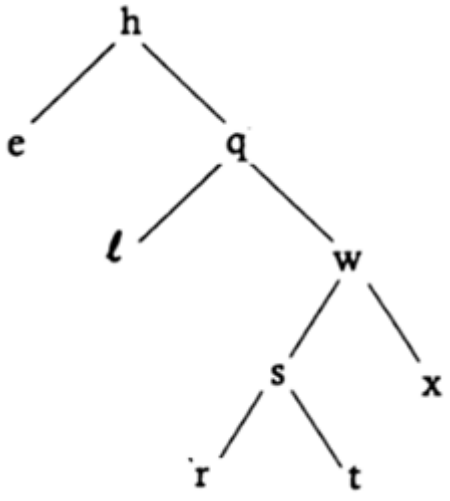


Рис.3.79. Дерево після видалення вершини *q*

Збалансоване дерево пошуку – це дерево пошуку, в якому число вершин в його лівих і правих піддеревах відрізняються не більше ніж на 1.

Збалансовані дерева пошуку:

- Червоно-чорні дерева
- АВЛ – дерева
- 2-3-дерева

- B-дерева
- …

Обхід дерев

Під **обходом** бінарного дерева розуміють визначений порядок проходження всіх вершин дерева. Розрізняють: прямий, зворотній та симетричний порядки обходу.

Прямий порядок обходу*:*

1. почати з кореня R
2. пройти в прямому порядку ліве піддерево A
3. пройти в прямому порядку праве піддерево B .

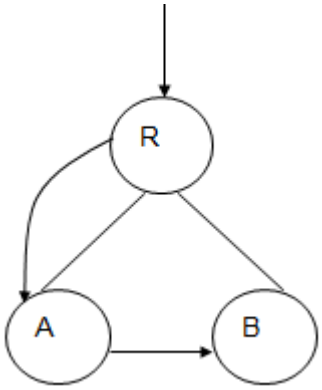


Рис.3.80. Ілюстрація прямого порядок обходу дерева

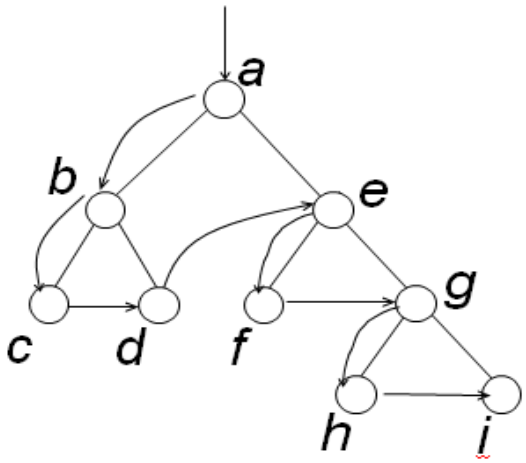


Рис.3.81.Прямий порядок обходу бінарного дерева: *a b c d e f g h i*

Обхід дерева в зворотньому порядку:

- пройти в зворотному порядку ліве піддерево A
- пройти в зворотному порядку праве піддерево B
- потрапити в корінь R

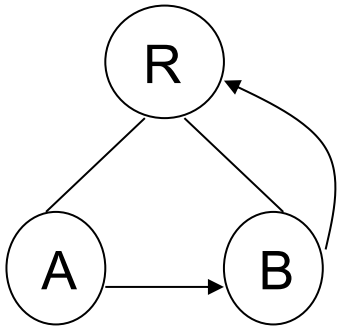


Рис.3.82. Ілюстрація зворотнього порядку обходу дерева

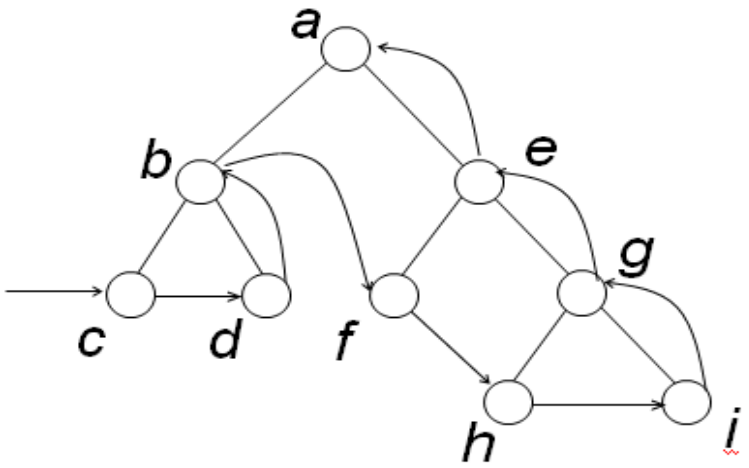


Рис.3.83. Зворотній порядок обходу бінарного дерева: *c d b f h i g e a*

Симетричний порядок обходу бінарного дерева:

- пройти в симетричному порядку ліве піддерево A
- потрапити в корінь R
- пройти в симетричному порядку праве піддерево B

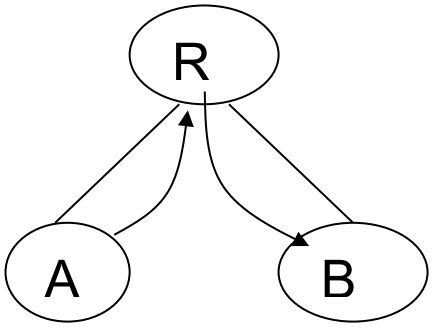


Рис.3.84. Ілюстрація симетричного порядку обходу дерева

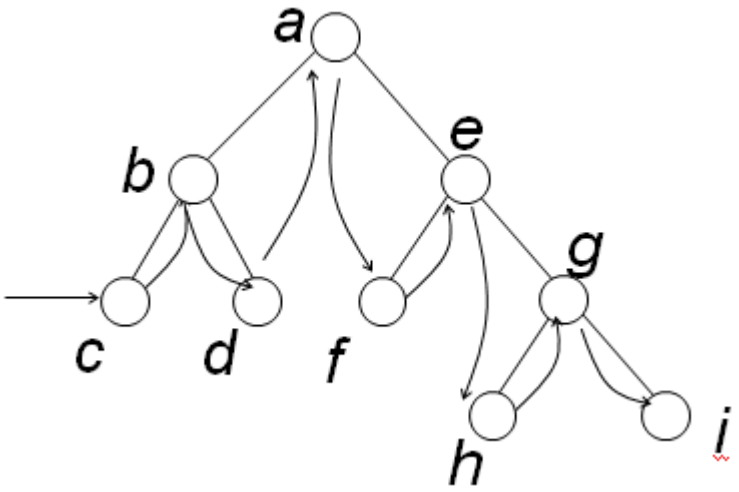


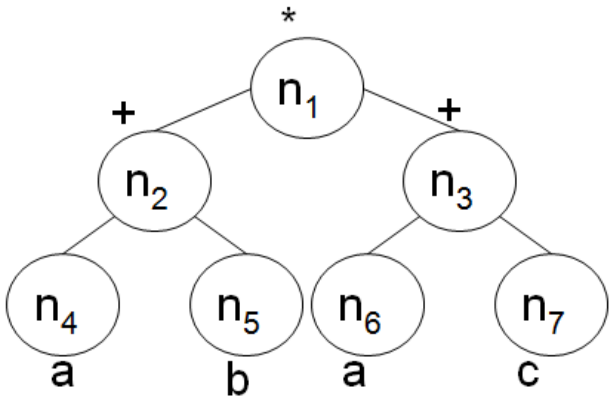
Рис.3.85.Симетричний порядок обходу бінарного дерева: *c b d a f e h g i*

Дерева виразів

Часто при обході дерев складається список не імен вузлів, а їх міток (label) – значень, які зберігаються у вузлі. Такі дерева називаються **дерева з мітками**. Проводять наступну аналогію: дерево – список, вузол – позиція, мітка – елемент.

За допомогою дерев можна представляти довільні арифметичні вирази. Кожному листові в такому дереві відповідає операнд, а кожному батьківському вузлу – операція. У загальному випадку дерево при цьому може виявитись не бінарним.

Приклад3.32. На рисунку наведено дерево з мітками, що представляє арифметичний вираз $(a+b)*(a+c)$, де n_1, n_2 …n_7 – імена вузлів, а мітки представлені поруч з відповідними вузлами.



- При прямому впорядкуванні міток отримуємо **префіксну** форму виразів, де оператор передує і левому і правому операндам. Префіксний вираз: *+*ab*+*ac*.
- Зворотнє впорядкування міток дерева виразів дає **постфіксне** (або польське) представлення виразів, при якому оператор іде після лівого і правого операндів. Постфіксна форма: *ab*+*ac*+*.
- При симетричному обході дерев виразів отримаємо так звану **інфіксну** форму виразу, яка співпадає зі звичайною стандартною формою запису виразу, але не використовує дужок. Інфіксний вираз: *a*+*b**a*+*c*.

Орієнтованим деревом називають зв'язний орієнтований граф без циклів, у якому напівстепінь входу кожної вершини, за винятком кореневої, дорівнює одиниці, а напівстепінь входу кореневої вершини дорівнює 0 (рис. 3.77).

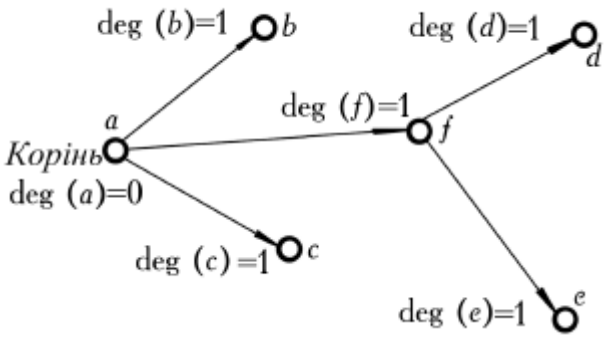


Рис.3.86.Орієнтоване дерево

Остовним підграфом називають такий підграф, у якому множина його вершин збігається з множиною вершин самого графа.

Остовним деревом графа *G* називають **остовний підграф** графа *G*, який є деревом.

Початковий граф, остовний підграф та остовне дерево, які утворені шляхом
послідовного видалення ребер, показані на рис. 3.78.

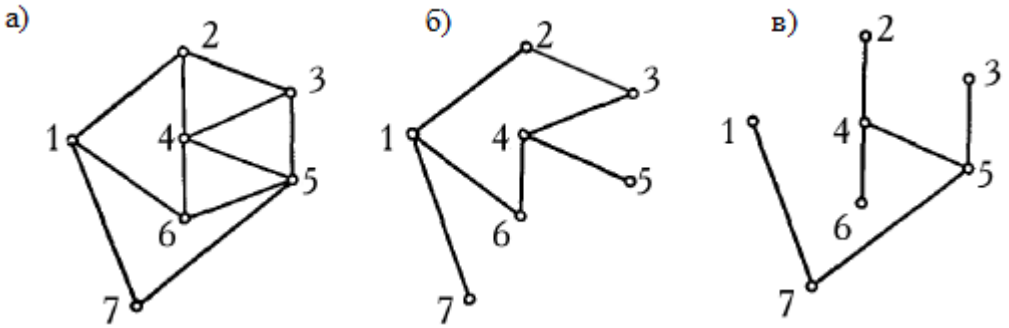


Рис.3.87. Утворення остовного підграфа та остовного дерева шляхом видалення ребер; а) початковий граф; б) остовний підграф; в) остовне дерево

Теорема 3.1. Граф є деревом тоді і тільки тоді, коли будь-які дві його вершини зв'язані єдиним ланцюгом.

Доведення. Нехай граф є деревом. Якщо припустити існування більш ніж одного ланцюга, що зв'язує будь-які дві його вершини, то в такому графі існує цикл, тобто граф не може бути деревом (рис. 3.88).

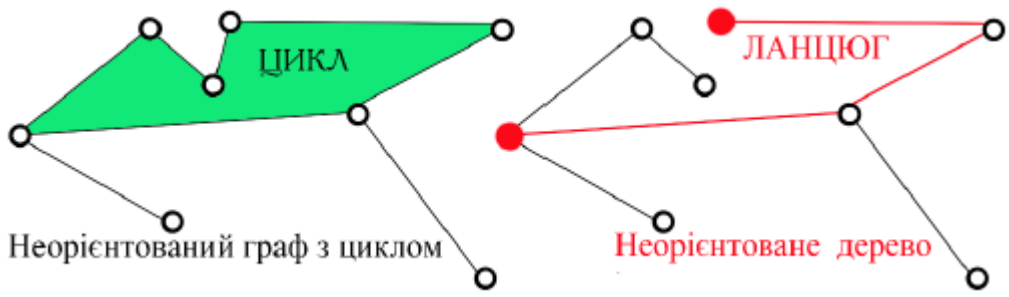


Рис. 3.88. Граф з циклом та неорієнтоване дерево

Навпаки, оскільки будь-які дві вершини графа з'єднані ланцюгом, то граф зв'язний, а в силу того, що цей ланцюг єдиний, він не має циклів. Отже, граф є деревом. Теорема доведена.

Наслідок 1. Якщо T – дерево і u – його кінцева вершина (листок), то граф $T-u$ *(T мінус u)* – дерево. Дійсно, граф $T-u$ – **правильний підграф** дерева T, для якого виконуються всі умови теореми (рис. 3.89).

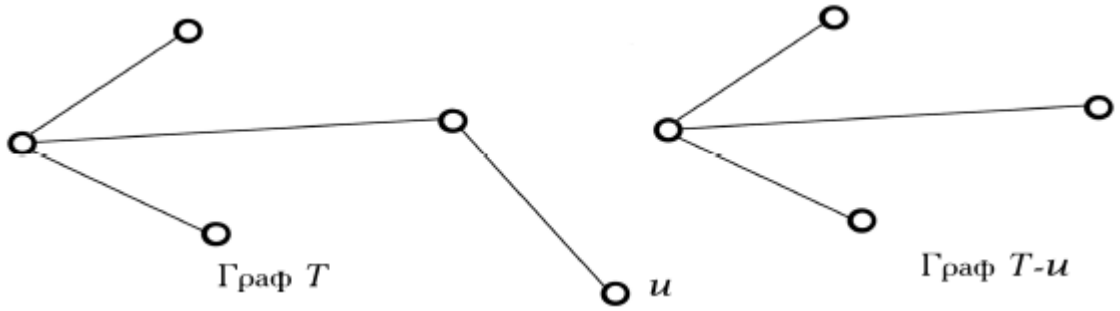


Рис. 3.89. Неорієнтовані дерева T і $T-u$

Наслідок 2. Будь-яке непусте дерево має принаймні дві висячі вершини і одне висяче ребро.

1. **Висяча вершина в неорієнтованому графі** – це вершина степеня 1.
2. **Висяча вершина в орграфі** – вершина з напівстепенем входу, що дорівнює 1, і напівстепенем виходу, що дорівнює 0.
3. **Висяче ребро** – це ребро, інцидентне вершині зі степенем 1.

Визначення. Ребро зв'язного графа називають **істотним**, якщо його видалення веде до порушення зв'язності цього графа.

Визначення. У неорієнтованому графі істотним ребром є міст.

Теорема 3.2. У дереві кожне ребро істотне.

Доведення. Доведення випливає з того, що видалення ребра $e = (u, v)$ у дереві T через наявність єдиного ланцюга, який з'єднує вершини u і v, веде до появи двох компонентів зв'язності: один компонент містить вершину u , а інший – вершину v . Отже, граф $T-e$ не є зв'язним, як показано на рис. 3.90.

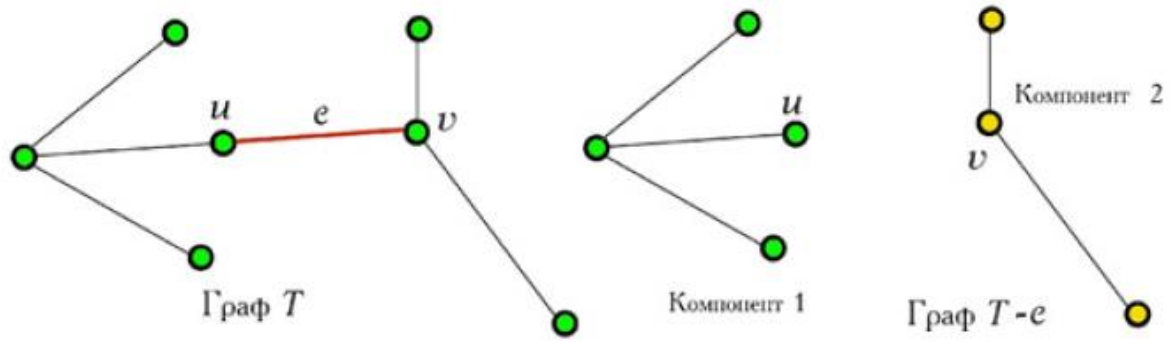


Рис. 3.90. Зв'язний граф T і незв'язний граф $T-e$

Теорема 3.3. Якщо $T = (V, E)$ – дерево і $v \notin V$ – вершина, u – довільна вершина $u \in V$, то граф $T' = (V \cup \{v\}, E \cup \{(u, v)\})$, теж є деревом.

Доведення. Оскільки T – дерево, то існує єдиний ланцюг, що з'єднує будь-яку вершину u' з вершиною u . Оскільки вершина $v \notin V$, то додавання одного кінцевого ребра (u, v) приводить до того, що з кожної вершини u' маємо лише єдиний ланцюг, який з'єднує вершини u' і v. Виходячи з теореми, граф T' є деревом.

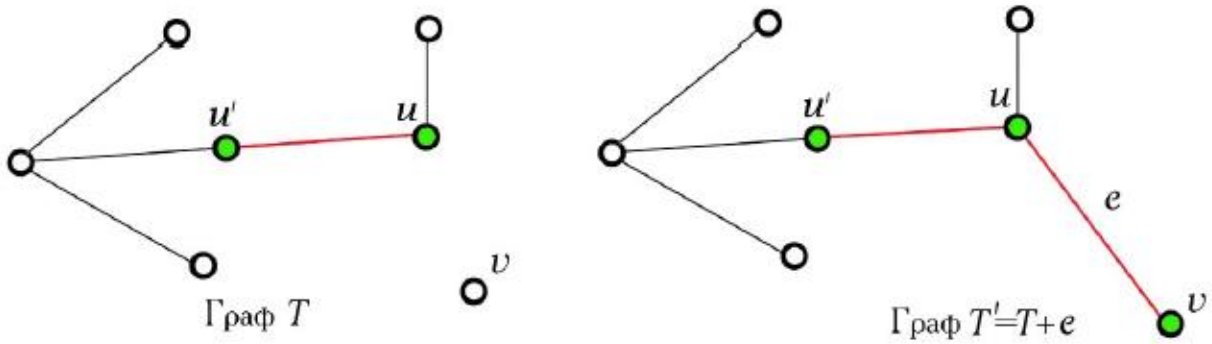


Рис. 3.91. Зв'язні графи-дерева T і $T+e$

Теорема 3.4. Нехай дерево T має n вершин. Тоді еквівалентними є такі твердження:

1. T не має циклів і має $n-1$ ребро.
2. T – зв'язний граф і має $n-1$ ребро.
3. T – зв'язний граф і кожне його ребро є мостом.
4. Будь-які дві вершини графа T з'єднані тільки одним простим ланцюгом.
5. T не має циклів, але додавання будь-якого нового ребра до T сприяє виникненню тільки одного циклу.

Визначення. Ліс – незв'язний n -граф без циклів.

1. Зв'язні компоненти лісу є деревами.
2. Будь-яка частина лісу або дерева також не має циклів.
3. Будь-яка частина лісу є лісом або деревом.

Приклад лісу показаний на рис. 3.92.

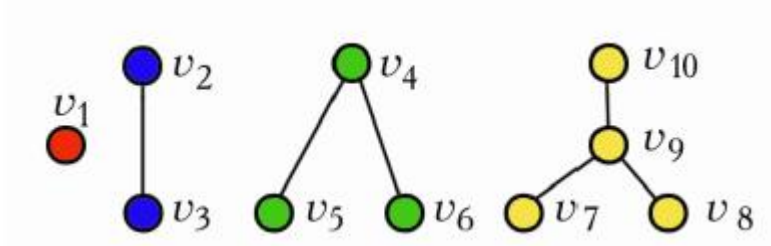


Рис. 3.92. Ліс з чотирьох дерев

Теорема 3.5. Нехай ліс G містить n вершин і k компонентів. Тоді ліс G має $n-k$ ребер.

Доведення. З умови 2 теореми 3.4 випливає, що кожний компонент G_i має $(n_i - 1)$ – ребро. Але тоді число ребер у G дорівнює $(n_1 - 1) + (n_2 - 1) + \ldots + (n_k - 1) = n_1 + n_2 + \ldots + n_k - k = n - k$,що і потрібно довести.

Теорема 3.6. Теорема Келі. Число різних дерев, які можна побудувати на n вершинах, дорівнює n^{n-2}.

3.8.2. Процедури побудови остовного дерева та лісу

Процедура побудови остовного дерева

1. Видалення із зв'язного графа G одного ребра, що належить деякому циклу, не порушує зв'язності графа G.
2. Застосуємо процедуру видалення ребра до одного з циклів у графі G.
3. Будемо повторювати видалення доти, поки в G не залишиться жодного циклу.
4. У результаті одержимо дерево, що містить усі вершини графа G. Це дерево називають **остовним деревом графа** G.

Приклад побудови остовного дерева шляхом почергового видалення ребер показано на рис. 3.93.

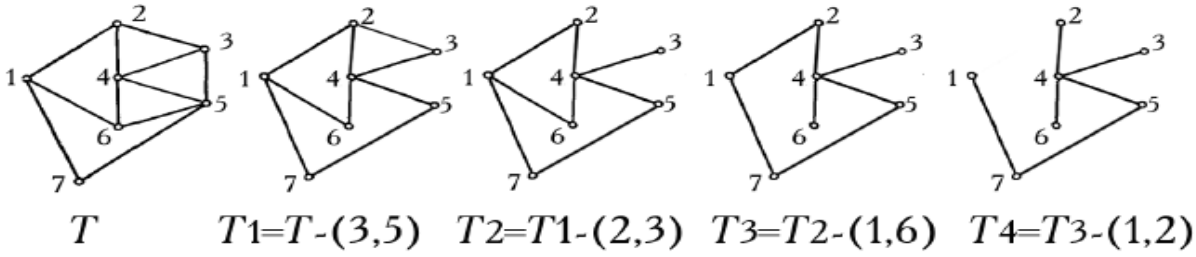


Рис. 3.93. Приклад побудови остовного дерева

Процедура побудови лісу

Нехай G є графом з n вершинами, m ребрами і k компонентами.

1. Застосуємо процедуру видалення ребра до одного з циклів у кожному компоненті зв'язності графа G.
2. Будемо повторювати видалення доти, поки в кожному компоненті G не залишиться жодного циклу.
3. У результаті одержимо граф, який називають **остовним лісом**.
4. Число ребер, які при цьому видаляються, називають **цикломатичним числом** або **циклічним рангом графа** G і позначають $C(G)$. Таким чином, цикломатичне число є мірою зв'язності графа.

Приклад побудови остовного лісу шляхом почергового видалення ребер показано на рис. 3.94.

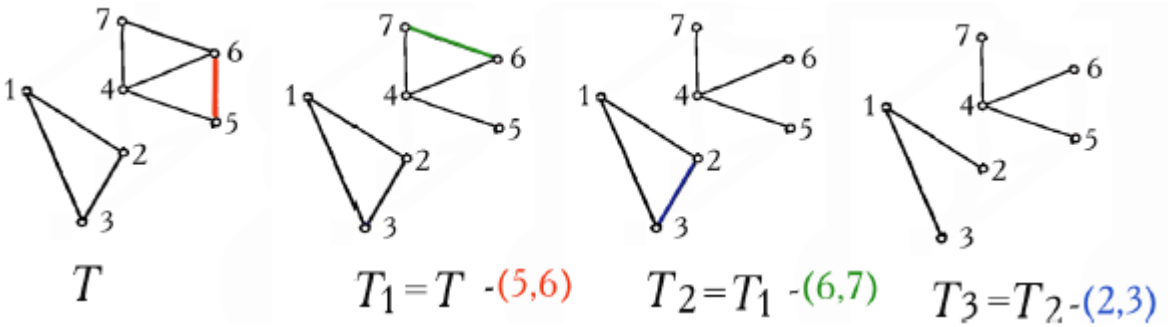


Рис. 3.94. Приклад побудови остовного лісу: $C(G) = 3$

3.8.3. Властивості циклічного рангу

1. **Циклічний ранг дерева** дорівнює нулю.
2. **Циклічний ранг циклічного графа** дорівнює одиниці.

Визначення. Циклічний граф – зв'язний регулярний граф степеня 2, єдиний компонент зв'язності циклічного графа є простим циклом (рис. 3.95).

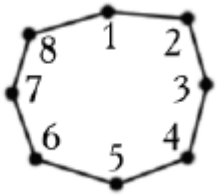


Рис. 3.95. Циклічний граф

Визначення. Остовне дерево графа – це дерево, що містить усі вершини графа.

Визначення. Остовним лісом називають незв'язний граф, що складається з компонентів, які є остовними деревами.

Теорема 3.7. Нехай T – остовний ліс графа G. Граф G', отриманий із графа G шляхом видалення всіх ребер графа T, називають **доповненням** остовного лісу T графа G (рис. 3.96).

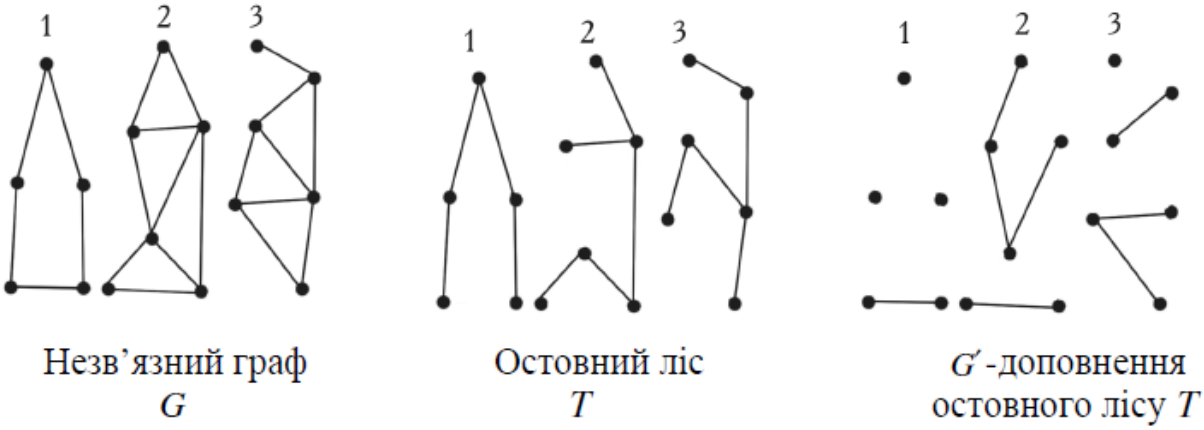


Рис. 3.96. Побудова доповнення остовного лісу G'

3.8.4. Фундаментальна система циклів графа

Нехай T – остовний ліс графа G. Якщо додати до T будь-яке ребро графа G, що не входить до нього, то за п. 5 теореми 3.4 одержимо єдиний цикл.

Визначення. Множину всіх циклів, які одержують шляхом додавання окремо
кожного ребра з G, що не входить до T, називають фундаментальною системою циклів, асоційованою з T.

Цикли даної фундаментальної системи будуть різними, але їх кількість дорівнює циклічному рангу графа G.

На рис. 3.97 графи показують фундаментальну систему циклів.

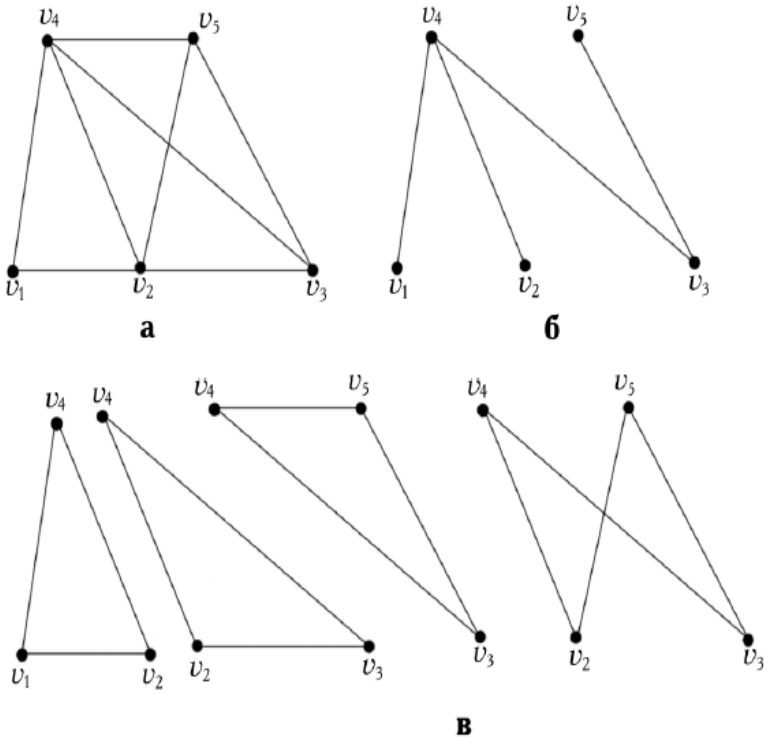


Рис. 3.97: Граф та його фундаментальні цикли: **а** – граф G; **б** – остовне дерево графа G; **в** – фундаментальна система циклів, асоційована з остовним деревом графа G

Теорема 3.8. Нехай $G = (V, E)$ – довільний скінченний граф. Число ребер графа G, які необхідно вилучити для одержання остовного лісу T, не залежить від порядку їх видалення і дорівнює $C(G) = E - V + k$, де k – число компонентів зв'язності графа G.

Наслідок 3.8.1. Граф G є остовним лісом тоді і тільки тоді, коли $C(G) = 0$.

Наслідок 3.8.2. Граф G має єдиний цикл тоді і тільки тоді, коли $C(G) = 1$.

Наслідок 3.8.3. Граф G, у якого число ребер перевищує число вершин, має цикл.

Наслідок 3.8.4. Будь-яке дерево порядку $n \geq 2$ має принаймні дві кінцеві вершини (рис. 3.98).

Рис. 3.98. Мінімальний граф з двома кінцевими вершинами

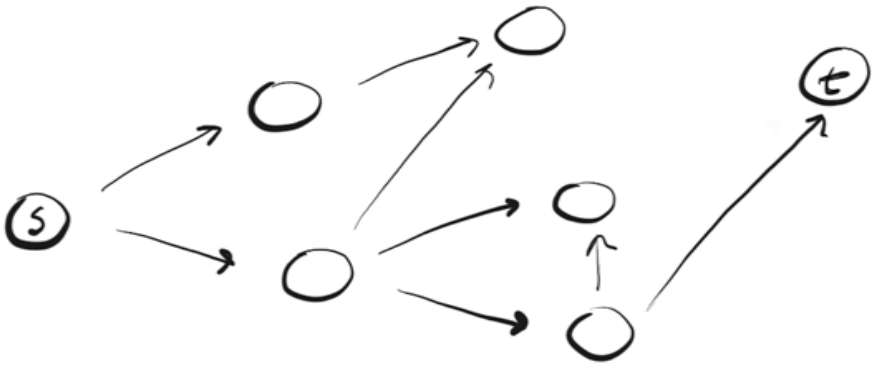
3.8.5. Остовні дерева. Пошук в глибину

Методами пошуку остовного дерева називають алгоритми обходу вершин графа, при якому кожна вершина отримує унікальний порядковий номер.

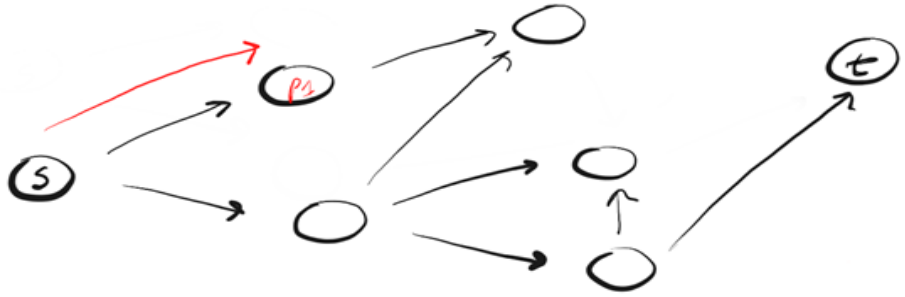
Цей метод називають **пошуком в глибину** або DFS-методом від англійського Depth First Search.

Нехай $G=(V,E)$ – простий зв'язний граф, усі вершини якого позначені попарно різними символами. У процесі пошуку вершинам надають номери (DFS-номери), які для вершини x позначають DFS(x). Пошук в глибину використовує для збереження множин структуру даних стек.

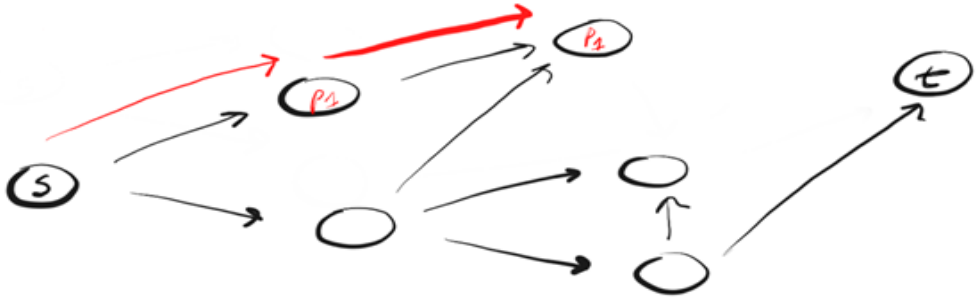
DFS слідує концепції «поринай глибже, головою вперед» («godeep, headfirst»). Ідея полягає в тому, що ми рухаємося від початкової вершини (точки, місця) в заданому напрямку (за заданим шляхом) до тих пір, поки не досягнемо кінця шляху або пункту призначення (шуканої вершини). Якщо ми досягли кінця шляху, але він не є пунктом призначення, то ми повертаємося назад (до точки розгалуження шляхів) і йдемо по іншому маршруту. Давайте розглянемо приклад. Нехай, в нас є орієнтований граф, який має наступний вигляд:



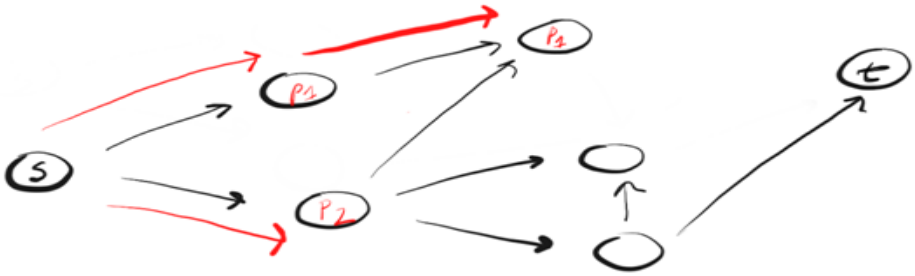
Ми знаходимося в точці «s» і нам потрібно знайти вершину «t». Застосовуючи DFS, ми досліджуємо один з можливих шляхів, рухаємося по ньому до кінця і, якщо не знайшли t, повертаємося і досліджуємо інший шлях. Ось як виглядає процес:



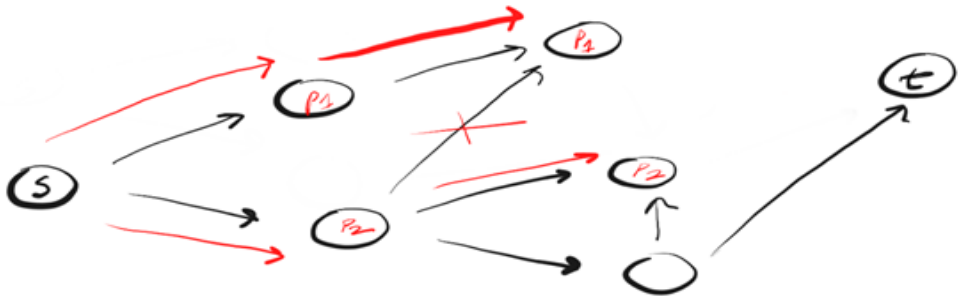
Тут ми рухаємося по маршруту (p1) до найближчої вершини і бачимо, що це не кінець шляху. Тому ми переходимо до настуної вершини.



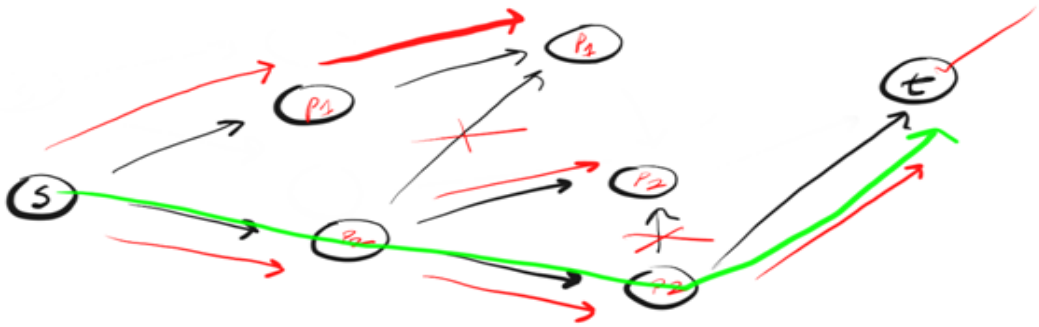
Ми досягли кінця p1, але не знайшли t, тому повертаємося в s і рухаємося по іншому маршруту.



Досягнувши найближчої доточки «s» вершини шляху «p2» ми бачимо три можливих напрямки для подальшого руху. Оскільки вершину, де закінчувався перший напрямок, ми вже відвідували, то рухаємося по другому.



Ми знову досягли кінця шляху, але не знайшли t, тому повертаємося назад. Слідуємо за третім шляхом і, нарешті, досягаємо шуканої вершини «t».



Так працює DFS. Рухаємося за заданим шяхом до кінця. Якщо кінець шляху — це шукана вершина, ми закінчили. Якщо ні, повертаємося назад і рухаємося іншим шляхом, доти, доки не дослідимо всі варіанти.
Ми слідуємо за цим алгоритмом, застосовуючи його до кожної відвідуваної вершини.
Необхідність багаторазового повторення процедури вказує на необхідність застосування рекурсії для реалізації алгоритму.

Примітка. Цей спеціальний DFS-алгоритм дозволяє перевірити, чи можливо дістатися з одного місця в інше. DFS може застосовуватися з різною метою. Від цієї мети залежить те, як буде виглядати сам алгоритм. Тим не менше, загальна концепція виглядає саме так.

Ось JavaScript-код:

```
// за умови, що ми маємо справу з суміжним списком
// наприклад, таким: adj = {A: [B, C], B: [D, F], ...}
Function dfs (adj, v, t) {
    // adj - суміжний список
    // v – відвіданий вузол (вершина)
    // t - пункт призначення
    // це загальні випадки
    // або досягли пункту призначення, або вже відвідували вузол
    If (v === t) return true
    if (v. visited) return false
    // помічаємо вузол як відвіданий
    v.visited = true
    // досліджуємо всіх сусідів (найближчі сусідні вершини) v
    For (let neighbor of adj[v]) {
        // якщосусід не відвідувався
        If (! neighbor. visited) {i
            // рухаємося шляхом і перевіряємо, чи не досягли ми пункту призначення
            let reached = dfs (adj, neighbor, t)
            // повертаємо true, якщодосягли
```

```
                    if(reached) return true
                }
            }
            // якщо від v до t дістатися неможливо
            return false
        }
```

Алгоритм пошуку остовного дерева в глибину

Крок 1. Почати з довільної вершини v_i. Покласти $\mathrm{DFS}(v_i) = 1$. Включити вершину v_iв стек.

Крок 2. Розглянути вершину, яка знаходиться у вершині стеку: нехай це буде вершині *x.* Якщо всі ребра, що інцедентні вершині *x*, позначені, то перейти до кроку 4, інакше – до кроку 3.

Крок 3. Нехай $\{x, y\}$- непозначене ребро. Якщо DFS(*y*) вже визначений, то ребро $\{x, y\}$ позначити штриховою лінією і перейти до кроку 2. Якщо DFS(*y*) не визначений, то ребро позначити потовщеною лінією, визначити DFS(*y*) як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.

Крок 4. Виключити вершину зі стеку. Якщо стек порожній, то зупинитись, інакше – перейти до кроку 2.

- Для однозначності вибору номерів доцільно домовитись, що аналіз вершин, суміжних з вершиною, яка вже отримала DFS-номер, здійснюють за зростанням їх порядкового номера (або в алфавітному порядку).
- Динаміку роботи алгоритму зручно відобразити за допомогою таблиці з трьома стовпцями: вершина, DFS-номер, вміст стеку. Цю таблицю називають **протоколом обходу графа** пошуком в глибину.

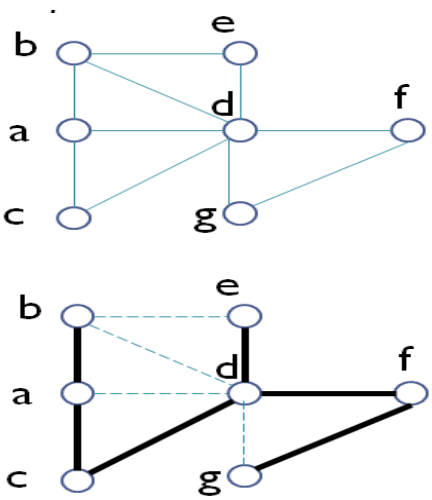
Аналіз DFS -алгоритму

Давайте проаналізуємо цей алгоритм. Оскільки ми обходимо кожного «сусіда» кожного вузла, ігноруючи тих, яких відвідували раніше ми маємо час виконання, що дорівнює O(*V* + *E*). Коротке пояснення того, що означає *V*+*E*: *V* — загальна кількість вершин. *E* — загальна кількість граней (ребер). Може здатися, що правильніше застосувати *V***E*, однак давайте подумаємо, що означає *V***E*. *V***E* означає, що стосовно кожної вершини, ми повинні дослідити всі грані графа безвідносно приналежності цих граней конкретній вершині.

З іншого боку, *V*+*E* означає, що для кожної вершини ми оцінюємо лише грані, що примикають до неї. Повертаючись до прикладу, кожна вершина має визначену кількість граней і, в гіршому випадку, ми обійдемо всі вершини (O(*V*)) і дослідимо всі грані (O(*E*)). Ми маємо *V* вершин і *E* граней, тому отримаємо *V*+*E*.

Далі, оскільки ми застосували рекурсію для обходу кожної вершини, це означає, що застосовується стек (безкінечна рекурсія приводить до помилки переповнення).

Приклад 3.33.



Протокол

Вершина	DFS-номер	Вміст стеку
b	1	b
a	2	ba
c	3	bac
d	4	bacd
e	5	bacde
-	-	bacd
f	6	bacdf
g	7	bacdfg
-	-	bacdf
-	-	bacd
-	-	bac
-	-	ba
-	-	b

3.8.6. Остовні дерева. Пошук в ширину

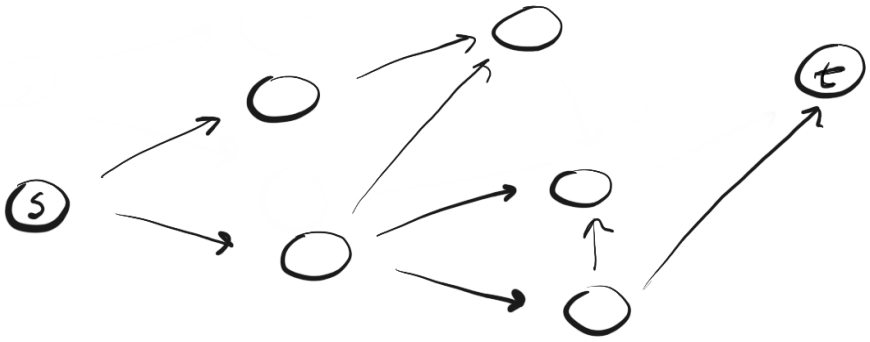
Цей метод називають **пошуком в ширину** або BFS-методом від англійського Breadth First Search.

У процесі пошуку вершинам надають номери (BFS -номери), які для вершини *x* позначають BFS(*x*). Пошук в ширину використовує для збереження множин структуру даних - черга.

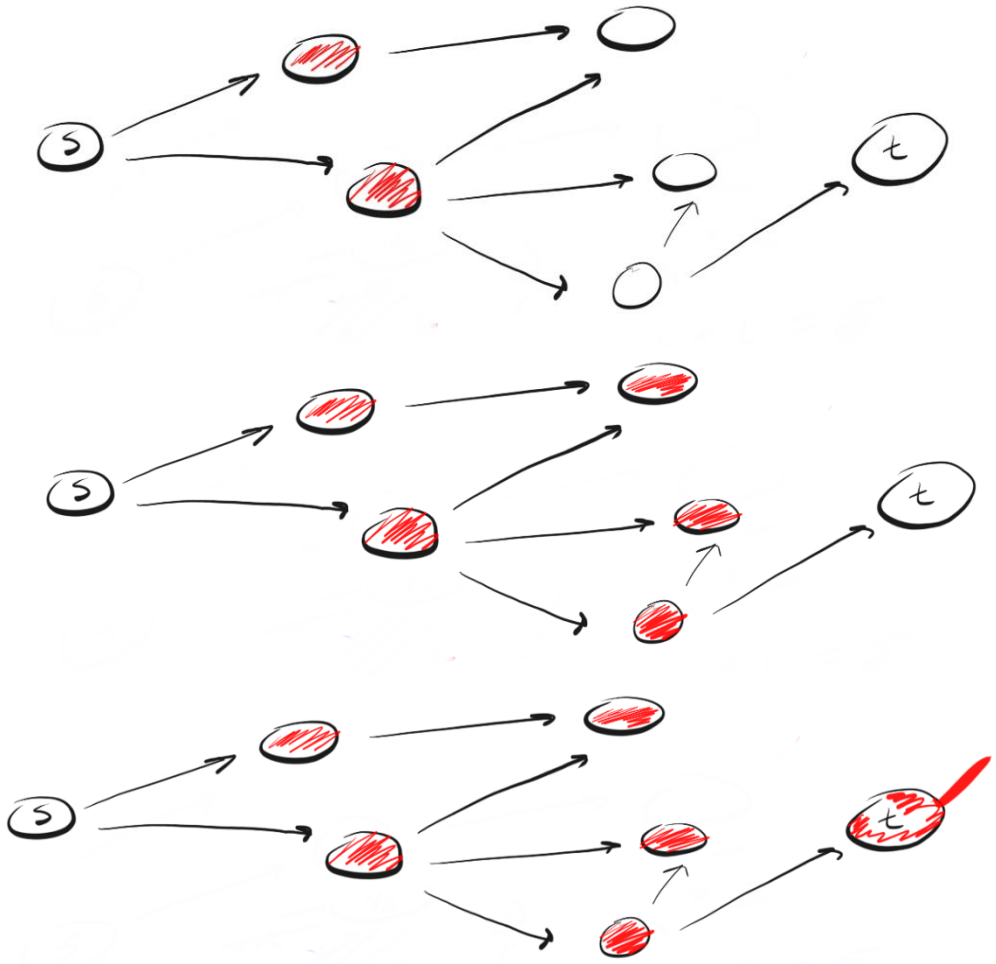
Для того, щоб результат виконання алгоритму був однозначним, вершини суміжні з вершиною *x*, аналізуємо за зростанням їхніх порядкових номерів (або в алфавітному порядку).

Динаміку роботи алгоритму відображаємо **протоколом обходу графа пошуком в ширину.**

BFS слідує концепції «розширюйся, піднімаючись на висоту пташиного польоту» («gowide, bird'seye-view»). Замість того, щоб рухатися заданим шляхом до кінцяа, BFS передбачає рух вперед по одному сусіду за раз. Це означає наступне:



Замість слідування шляхом, BFS передбачає відвідування найближчих до s сусідів за один крок, потім відвідування сусідів і так далі до тих пір, поки не буде досягнута t.



Чим DFS відрізняється від BFS? Мені здається що, DFS йде напролом, а BFS не поспішає, а досліджує все в межах одного кроку.

Далі виникає питання: як дізнатися, яких сусідів слід відвідати першими? Для цього ми можемо використати концепцію «першим зайшов, першим вийшов» (first-in-first-out, FIFO) з черги (queue). Ми заносимо в чергу спочатку найближчу до нас вершину, потім її невівіданих сусідів, і продовжуємо цей процес, доки черги не залишиться або доки ми не знайдемо шукану вершину.

Ось JavaScript-код:

```
// за умовою, що мимаємо справу з суміжним списком
// наприклад, таким: adj = {A:[B,C,D], B:[E,F], ... }
functionbfs (adj, s, t) {
    // adj - суміжний список
    // s - початкова вершина
    // t - пункт призначення
    // ініціалізуємо чергу
    letqueue = []
    // додаємо s в чергу
    queue. push(s)
    // помічаємо s як відвідану вершину, щоб не повторити додавання в
чергу
```

```
        s. visited = true
        while (queue. length > 0) {
            // видаляємо перший (верхній) елемент з черги
            let v = queue. shift ()
            // abj[v] - сусіди v
            for (let neighbor of adj[v]) {
                // якщо сусід не відвідувався
                if (! neighbor. visited) {
                    // додаємойого вчергу
                    queue. push (neighbor)
                    // помічаємо вершину як відвідану
                    neighbor. visited = true
                    // якщо сусід є пунктом призначення, ми
перемогли
                    if (neighbor === t) return true
                }
            }
        }
        // якщо t не знайдена, значить пункта призначення досягти
неможливо
        returnfalse
    }
```

Алгоритм пошуку остовного дерева в ширину

- Крок 1. Почати з довільної вершини v_i. Покласти BFS(v_i)=1. Включити вершину v_i в чергу.
- Крок 2. Розглянути вершину, яка знаходиться на початку черги: нехай це буде вершина *x*. Якщо для всіх вершин, суміжних з вершиною *x,* вже визначені BFS-номери, то перейти до кроку 4, інакше – до кроку 3.
- Крок 3. Нехай $\{x, y\}$ - ребро, в якому номер BFS(*y*) не визначений. Позначити це ребро потовщеною суцільною лінією, визначити BFS(*y*) як черговий BFS-номер, включити вершину до черги й перейти до кроку 2.
- Крок 4. Виключити вершину *x* з черги. Якщо черга порожня, то зупинитись, інакше – перейти до кроку 2.

Аналіз BFS

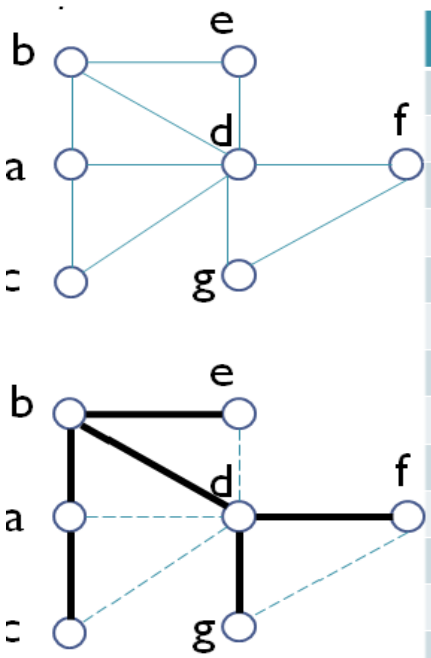
Може здатися, що BFSпрацює повільніше. Однак, якщо уважно придивитися до візуалізацій, можно побачити, що вонимають однаковий час виконання.

Черга передбачає обробку кожної вершини перед досягненням пункту призначення. Це означає, що, в гіршому випадку, BFS досліджує всі вершини і грані.

Не дивлячись на те, що BFS може здаватися повільнішим, але він скоріший, оскільки при роботі з великими графами виявляється, що DFS витрачає багато часу на слідування шляхами, які в кінцевому рахунку виявляються хибними. BFS часто застосовують для знаходження найкоротшого шляху між двома вершинами.

Таким чином, виконання BFS також складає O(V + E), а оскільки ми застосовуємо чергу, що вміщує всі вершини, його просторова складність складає O(V).

Приклад 3.34.



Протокол

Вершина	BFS-номер	Вміст черги
b	1	b
a	2	ba
d	3	bad
e	4	bade
-	-	ade
c	5	adec
-	-	dec
f	6	decf
g	7	decfg
-	-	ecfg
-	-	cfg
-	-	fg
-	-	g
-	-	∅

Обчислювальна складність обох алгоритмів обходу однакова й у випадку зображення графа списками суміжності складає O(|*V*|+|*E*|).

Аналоги з реального життя

Якщо наводити приклади з реального життя, то ось як я представляю собі роботу DFS і BFS.

Коли я думаю про DFS, то представляю собі мишу в лабіринті в пошуку їжі. Для того, щоб потрапити до їжі миша змушена багато разів потрапляти в тупик, повертатися і рухатися в іншому напрямку, і так доки вона не знайде вихід з лабіринту або їжу.

Рис.3.99. Ілюстрація DFS-алгоритму

Спрощена версія виглядає так:

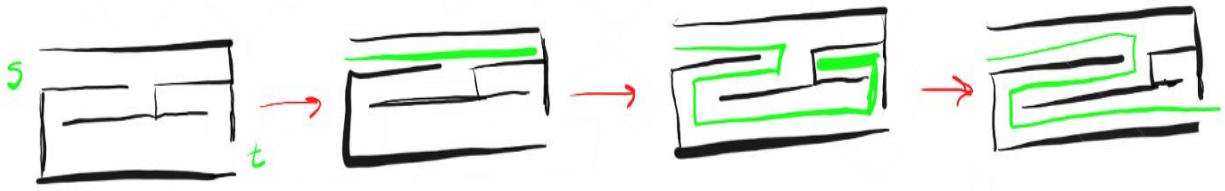


Рис.3.100. Спрощена версія DFS-алгоритму

В свою чергу, коли я думаю про BFS, то представляю собі кола на воді. Падіння каменю в воду приводить до утворення кіл по всіх напрямках від центру.

Рис.3.101. Ілюстрація BFS-алгоритму

Спрощена версія виглядає так:

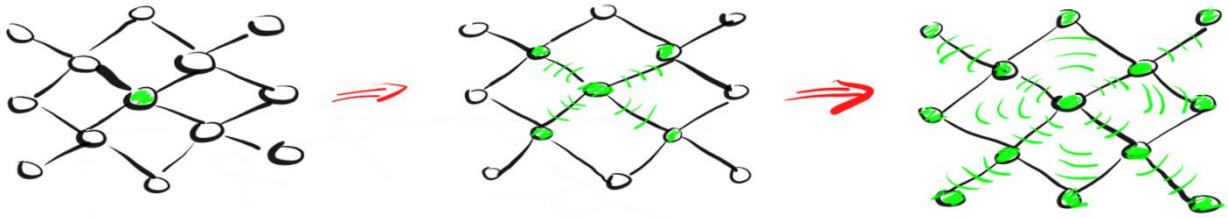


Рис.3.102. Спрощена версія BFS-алгоритму

Висновки

- Пошук в глибину і пошук в ширину використовують для обхода графа.
- DFS рухається по гранях (ребрах) туди і назад, а BFS розповсюджується по сусідах в пошуку призначення.
- DFS використовує стек, а BFS — чергу.
- Час виконання обох складає O($V + E$), а просторова складність — O(V).
- Дані алгоритми мають різну філософію, але однаково важливі для роботи з графами.

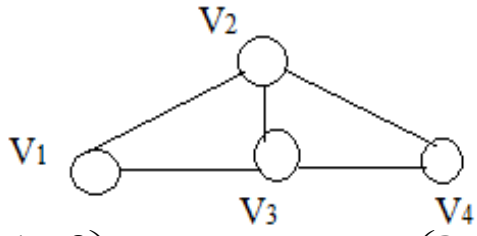
Алгоритм пошуку максимальної кількості остовних дерев

Цей алгоритм застосовується для будь-якого зв'язного графа $G(V,E)$ з поміченими вершинами, в якому необхідно знайти всі можливі остовні дерева.

Кроки алгоритму пошуку максимальної кількості остовних дерев

- Знайти степені вершин цього зв'язного графа;
- Побудувати матрицю степенів вершин зв'язного графа;
- Побудувати матрицю суміжності зв'язного графа;
- Знайти різницю між матрицями степенів і суміжності зв'язного графа $G(V,E)$;
- Знайти будь-яке з алгебраїчних доповнень отриманої матриці різниці на кроці 4 алгоритму, значення якого є кількісним значенням остовних дерев для графа $G(V,E)$.

Приклад 3.35. Використовуючи розглянутий алгоритм, знайти максимальну кількість остовних дерев для зв'язного графа $G(V,E)$, наведеного на рисунку:



$$A=\begin{pmatrix}0&1&1&0\\1&0&1&1\\1&1&0&1\\0&1&1&0\end{pmatrix} \qquad C=\begin{pmatrix}2&0&0&0\\0&3&0&0\\0&0&3&0\\0&0&0&2\end{pmatrix}$$

Розв'язок.

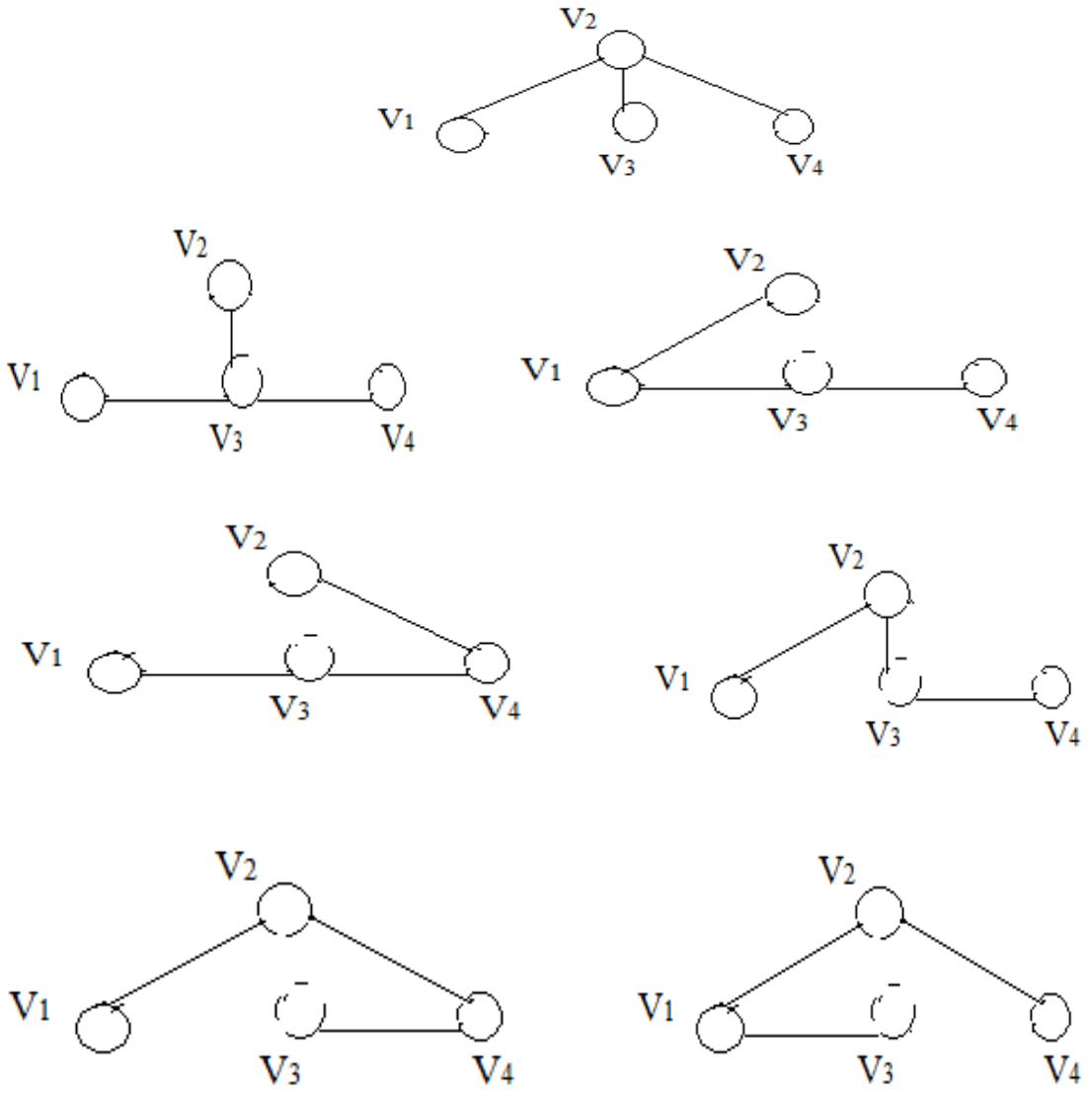
Відповідно до кроку 1 алгоритму, знайдемо ступені вершин графа $G(V,E)$, які дорівнюють: $\deg(V_1) = \deg(V_4) = 2$; $\deg(V_2) = \deg(V_3) = 3$. Згідно з кроком 2 алгоритму будуємо степенну вершинну матрицю C і кроком 3 – матрицю суміжності A графа $G(V,E)$. Користуючись кроком 4 алгоритму, знаходимо різницю між двома матрицями C і A, внаслідок чого отримуємо третю матрицю K:

$$K=C-A=\begin{pmatrix}2&0&0&0\\0&3&0&0\\0&0&3&0\end{pmatrix}-\begin{pmatrix}0&1&1&0\\1&0&1&1\\1&1&0&1\end{pmatrix}=\begin{pmatrix}2&-1&-1&0\\-1&3&-1&-1\\-1&-1&3&-1\end{pmatrix}$$

Використовуючи крок 5 алгоритму, знаходимо алгебраїчне доповнення до матриці K, наприклад K11, яке дорівнює

$$K_{11} = \det\begin{pmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{pmatrix} = 3\cdot\det\begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix} - (-1)\begin{pmatrix} -1 & -1 \\ -1 & 2 \end{pmatrix} + (-1)\begin{pmatrix} -1 & 3 \\ -1 & -1 \end{pmatrix} = 3\cdot 5 - 3 - 4 = 8$$

Виходячи із значення цього доповнення, можна сказати, що граф G(V,E), який наведений на рисунку, має вісім остовних дерев:



Обов'язкові завдання.

1. Сформулюйте задачу обходу графа та базовий алгоритм обходу. Яку вершину називають закритою вершиною?
2. Опишіть принципи, на яких ґрунтується алгоритм обходу графа в глибину. Яка умова зупинки алгоритму обходу графа в глибину?
3. Написати програму та вивести на друк проміжні результати обходу графу (рис. 3.103) в глибину.

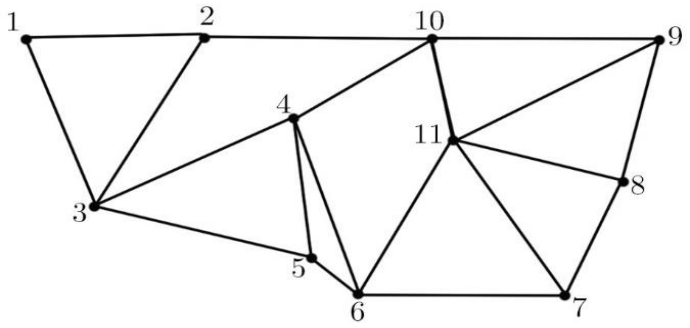


Рис. 3.103. Граф до задач 3 та 4

4.Створити програму обходу графа в ширину. Застосувати дану програму до графа, показаного на рис. 3.103.

3.8.7. Остовне дерево найменшої ваги

Нехай кожному ребру графа $G = (V, E)$ поставлена у відповідність деяка вага $d_i = d(e_i), e_i \in E, i = 1,2,\ldots,|E|$.

Необхідно в графі G знайти остовне дерево, сума ваг d_i у якому найменша. $S = \min \sum_{e \in E_i} (d(e_i))$. Число d_i називають **вагою ребра** e_i , а сам граф G – таким, що має **вагу (зваженим)**. Отже, завдання полягає в тому, щоб знайти остовне дерево з найменшою вагою.

Практичні застосування задачі пошуку остовного дерева з мінімальною вагою.

Ця задача виникає при проєктуванні доріг, ліній електропередач, трубопроводів та ін., коли необхідно з'єднати задані точки деякою системою зв'язку так, щоб загальна довжина ліній зв'язку була мінімальною. На рис. 3.91 показані об'єкти, при проєктуванні яких застосовують розв'язування задачі пошуку остовного дерева з мінімальною вагою.

Рис. 3.104. Приклади об’єктів, для яких необхідна мінімізація параметрів
Розглянемо кілька методів розв’язування цієї задачі.

Розв’язування, що випливає з теореми Келі

1. Розглянемо всі можливі остовні дерева повного графа G з n вершинами. Згідно з теоремою Келі число різних дерев, які можна побудувати на n вершинах, дорівнює n^{n-2} .
2. При виборі кожного дерева визначимо суму його ваг.
3. Застосувавши сортування за величиною суми ваг, на початку списку одержимо ті остовні дерева, які мають мінімальну суму ваг. Але оскільки число n^{n-2} навіть при невеликому n буде дуже великим, то такий шлях розв’язування даної задачі досить трудомісткий. Тому актуальним є використання більш ефективних алгоритмів.

Попередні зауваження

1. **Порядок графа** – число, яке дорівнює кількості вершин графа.
2. **Порожній граф** – граф, що не містить ребер або регулярний граф степеня 0.

Постановка задачі. Задано зв’язний неорієнтований граф $G = (V, E)$, де V –множина вершин, а E – множина ребер і для кожного ребра $(u,v) \in E$ задано вагу $w(u,v)$. Потрібно знайти ациклічну підмножину $T \in E$, яка з’єднує всі вершини і загальна вага якої

$w(T) = \sum_{(u,v)\in T} w(u,v)$ мінімальна.

Задача пошуку дерева T називається задачею пошуку мінімального остовного дерева.

Ми розглянемо два алгоритми розв’язку даної задачі – алгоритм Прима та алгоритм Крускала.

3.8.8.Алгоритм Прима

Ізольованою називається вершина, яка на деякому етапі побудови не зв'язана з іншими вершинами. *Фрагмент* – це підмножина вершин зв'язаних ребрами.

Ізольованим фрагментом називається фрагмент, який на даному етапі побудови не зв'язаний з іншими вершинами або фрагментами.

Принципи побудови дерева мінімальної довжини:

- Довільна ізольована вершина з'єднується з найближчим сусідом – вершиною, яка знаходиться на найменшій відстані від даної вершини.
- Довільний ізольований фрагмент з'єднується з найближчим сусідом найкоротшим ребром.

Алгоритм Прима.

1. Побудова матриці суміжності ваг.

Якщо вершини u та v не з'єднані, то в матриці на перетині рядка u та стовпчика v ставиться нескінченість (∞). Діагональні елементи умовно приймаються рівними нескінченості (∞). Всі інші елементи матриці дорівнюють $w(u,v)$.

2. Визначення першого фрагменту.

За початкову обирається довільна вершина. Згідно принципу 1 для цієї вершини знаходимо найближчого сусіда. Для цього в матриці обирається рядок відстаней від обраної вершини до всіх інших і визначається вершина до якої відстань найменша.

3. Розширення фрагменту.

Для розширення фрагменту порівнюються відстані від отриманого фрагменту до кожної ізольованої вершини. З усіх можливих з'єднань обирається така ізольована вершина відстань до якої найменша.

4. Закінчення.

Якщо всі ізольовані вершини приєднані, то мінімальне остовне дерево побудоване (роботу алгоритму завершено), якщо – ні, то перейти на крок 3.

Псевдокод алгоритму Прима:

// Вхідні дані: Зважений зв'язний граф $G=(V, E)$

//Вихідні дані: E_T, множина ребер, які утворюють мінімальне остовне дерево T

$V_T \leftarrow \{ v_0 \}$ //Множина вершин остовного дерева T

$E_T \leftarrow \varnothing$ //Множина ребер остовного дерева T

for $i \leftarrow 1$ **to** $|V|-1$ **do**

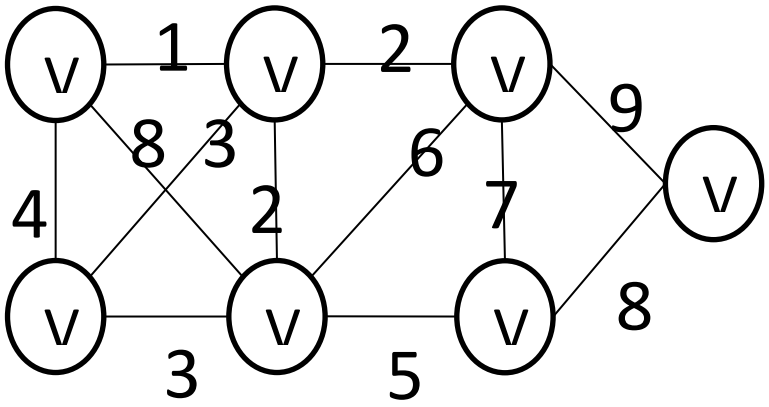
// Пошук ребра з мінімальною вагою $e^* = (u^*,v^*)$ серед всіх ребер (u,v) таких, що $v \in V_T$ та $u \in V-V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

return E_T

Приклад 3.35. Для заданого графа G побудувати остовне дерево мінімальної ваги, використовуючи алгоритм Прима.

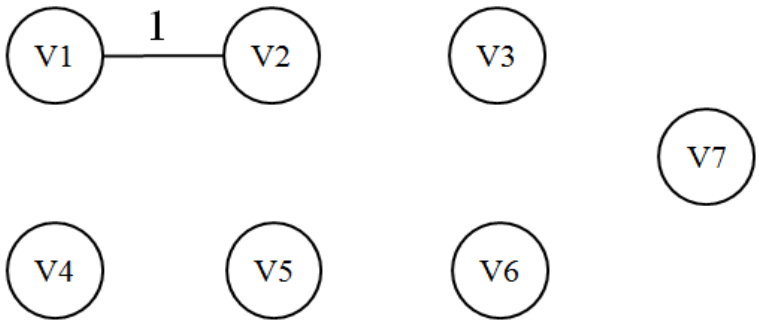


Побудуємо матрицю суміжності ваг W:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	∞	1	∞	4	8	∞	∞
v_2	1	∞	2	3	2	∞	∞
v_3	∞	2	∞	∞	6	7	9
v_4	4	3	∞	∞	3	∞	∞
v_5	8	2	6	3	∞	5	∞
v_6	∞	∞	7	∞	5	∞	8
v_7	∞	∞	9	∞	∞	8	∞

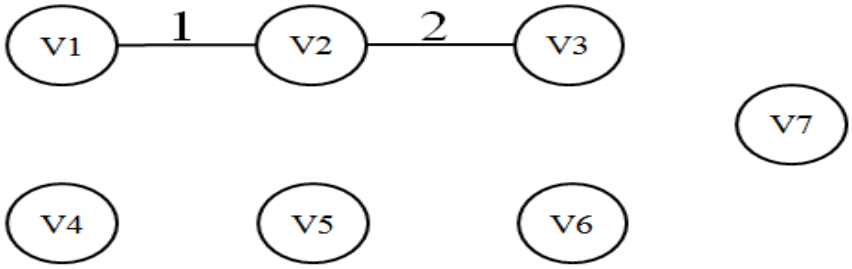
За початкову обираємо довільну вершину, нехай це буде вершина V_1, і для неї шукаємо найближчого сусіда, тобто вершину відстань до якої найменша:

	v_2	v_3	v_4	v_5	v_6	v_7
v_1	1	∞	4	8	∞	∞



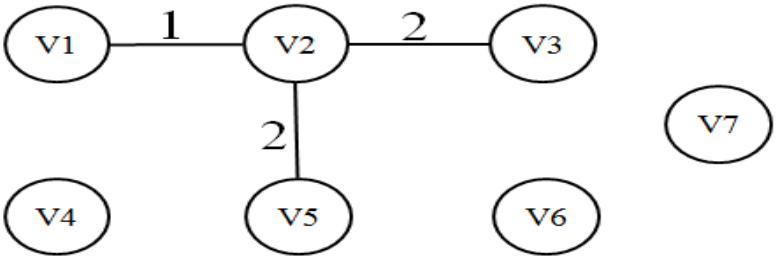
Розширюємо фрагмент

	v_3	v_4	v_5	v_6	v_7
v_1	∞	4	8	∞	∞
v_2	2	3	2	∞	∞



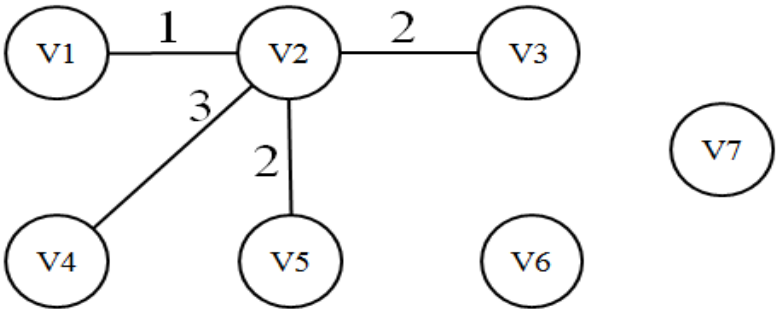
Розширюємо фрагмент

	v_4	v_5	v_6	v_7
v_1	4	8	∞	∞
v_2	3	2	∞	∞
v_3	∞	6	7	9

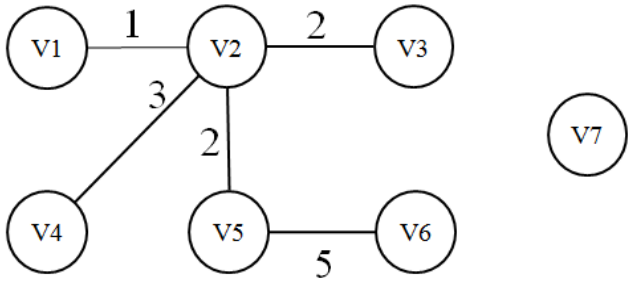


Розширюємо фрагмент

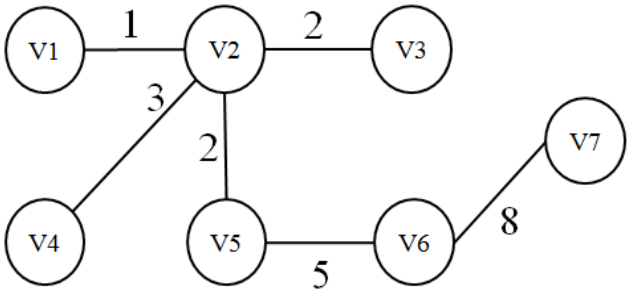
	v_4	v_6	v_7
v_1	4	∞	∞
v_2	3	∞	∞
v_3	∞	7	9
v_5	3	5	∞



	v_6	v_7
v_1	∞	∞
v_2	∞	∞
v_3	7	9
v_4	∞	∞
v_5	5	∞



	v_7
v_1	∞
v_2	∞
v_3	9
v_4	∞
v_5	∞
v_6	8



$$w(T) = \sum_{(u,v)\in T} w(u,v) = 1+2+2+3+5+8 = 21$$

Властивості алгоритму Прима

1. Алгоритм використовується для зважених **неорієнтованих зв'язних** графів.
2. Обчислювальна складність алгоритму Прима $O(n^2)$, де n – кількість вершин графа. Якщо значення n достатньо **велике**, то використовувати цей алгоритм **не раціонально**.
3. Якщо кількість ребер e **значно меншою за n^2**, то алгоритм Краскала кращий, але якщо e ***на*близжена до n^2**, то рекомендують застосовувати алгоритм Прима.

3.8.9.Алгоритм Крускала

Алгоритм Крускала будує мінімальне остовне дерево як послідовність підграфів, котрі завжди ациклічні, але на проміжних стадіях не завжди зв'язні.

Алгоритм Крускала.

1. Відсортувати ребра графу G в зростаючому порядку.
2. Вибрати ребро e_1, яке має в графі G найменшу вагу.
3. На кожному кроці обирати ребро (відмінне від попередніх) з найменшою вагою і таке, що не утворює простих циклів з попередніми ребрами. Отримане дерево T з множиною ребер

$E_T = \{e_1, e_2, e_3, \ldots, e_{n-1}\}$ є мінімальним остовним підграфом графу G.

Псевдокод алгоритму Крускала:

// Вхідні дані: Зважений зв'язний граф $G=(V,E)$

//Вихідні дані: E_T, множина ребер, які утворюють мінімальне остовне дерево T

Сортування множини E за зростанням ваг ребер $w(e_1) \le \ldots \le w(e_n)$, $n = |E|$

$E_T \leftarrow \varnothing$ //Множина ребер остовного дерева T

ecounter $\leftarrow 0$//розмір дерева T

$k \leftarrow 0$ //кількість оброблених ребер

while *ecounter* < $|V|-1$ **do**

$k \leftarrow k + 1$

if $E_T \cup \{e_k\}$ – ациклічний граф **then**

$E_T \leftarrow E_T \cup \{ e_k \}$; *ecounter* $\leftarrow$ *ecounter* + 1

return E_T

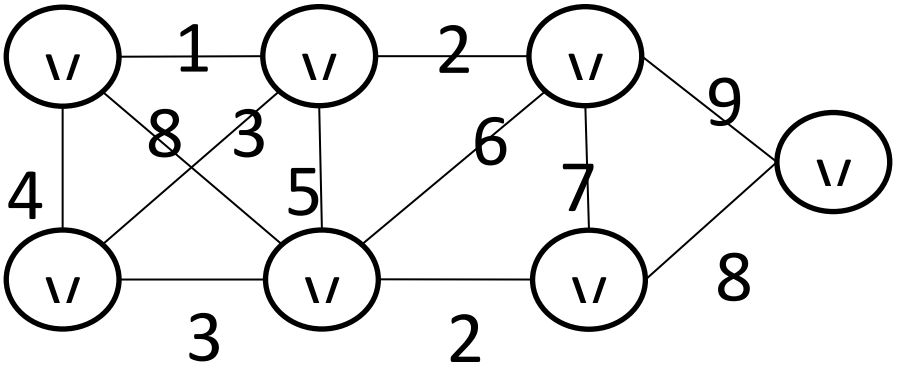
Одна з можливих реалізацій алгоритму Крускала.

Початок. Упорядкувати множину ребер за зростанням ваг: e_1, e_2, e_3, …, e_n. Виконати розбиття множини вершин .

Ітерація. Вибрати таке чергове ребро з упорядкованої послідовності ребер, кінці якого містяться в різних множинах розбиття. Нехай обрано ребро $e_i(u,v)$, тоді множини, що містять вершини u та v об'єднуються в одну множину.

Закінчення. Роботу закінчити, коли буде вибране (n–1) ребро, при цьому всі підмножини розбиття об'єднуються в одну.

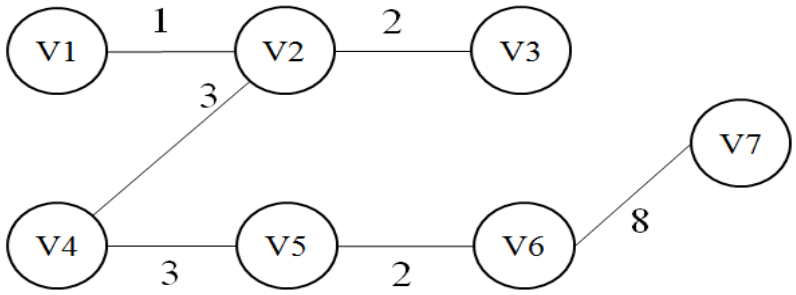
Приклад 3.36. Для заданого графа G побудувати остовне дерево мінімальної ваги, використовуючи алгоритм Крускала.



Ребра впорядковані за зростанням	Розбиття множини вершин	Вибір ребра у мінімальний остов T
	$\rho_0=\{\{v_1\},\{v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\}\}$	
$e_1(v_1, v_2) = 1$	$\rho_1=\{\{v_1,v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\}\}$	$E_T = \{ e_1 \}$
$e_2(v_2, v_3) = 2$	$\rho_2=\{\{v_1,v_2,v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\}\}$	$E_T = \{ e_1, e_2 \}$
$e_3(v_5, v_6) = 2$	$\rho_3=\{\{v_1,v_2,v_3\},\{v_4\},\{v_5,v_6\},\{v_7\}\}$	$E_T = \{ e_1, e_2, e_3 \}$
$e_4(v_2, v_4) = 3$	$\rho_4=\{\{v_1,v_2,v_3,v_4\},\{v_5,v_6\},\{v_7\}\}$	$E_T = \{ e_1, e_2, e_3, e_4 \}$
$e_5(v_4, v_5) = 3$	$\rho_5=\{\{v_1,v_2,v_3,v_4,v_5,v_6\},\{v_7\}\}$	$E_T = \{ e_1, e_2, e_3, e_4, e_5 \}$
$e_6(v_1, v_4) = 4$	ρ_5	$e_6 \notin E_T$
$e_7(v_2, v_5) = 5$	ρ_5	$e_7 \notin E_T$
$e_8(v_3, v_5) = 6$	ρ_5	$e_8 \notin E_T$
$e_9(v_3, v_6) = 7$	ρ_5	$e_9 \notin E_T$
$e_{10}(v_1, v_5) = 8$	ρ_5	$e_{10} \notin E_T$
$e_{11}(v_6, v_7) = 8$	$\rho_6=\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$	$E_T = \{ e_1, e_2, e_3, e_4, e_5, e_{11} \}$
$e_{12}(v_3, v_7) = 9$	-	-

При приєднанні ребра e_{11} робота алгоритму закінчується, так як вже приєднано

$n-1= 7-1=6$ ребер і всі підмножини розбиття об'єдналися в одну $\rho_6=\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$. Остовне дерево мінімальної ваги T утворюють ребра e_1, e_2, e_3, e_4, e_5, e_{11} .



Властивості алгоритму Краскала

1. Алгоритм використовується для зважених неорієнтованих графів.
2. Алгоритм може бути використаний для побудови остовного лісу в багатокомпонентних незв'язних графах.

У цьому випадку **структури даних**, що описують кожну з компонент зв'язності, **повинні бути окремими**.

Обчислювальна складність алгоритму Краскала O ($e \times \log e$), де e – кількість ребер у даному графі.

Обов'язкові завдання.

1. Розглянемо ліс, який складається з трьох компонентів. Множина вершин лісу $V = \{v_1.v_2.v_3.v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$. Знайти потужність множини ребер лісу.

2. Дано множину вершин графа $V = \{v_1.v_2.v_3.v_4\}$. Обчислити кількість дерев, які можна побудувати на множині вершин *V*. Побудувати всі можливі дерева.

3. Побудувати остовний ліс та доповнення остовного лісу для незв'язного графа, показаного на рис. 3.105.

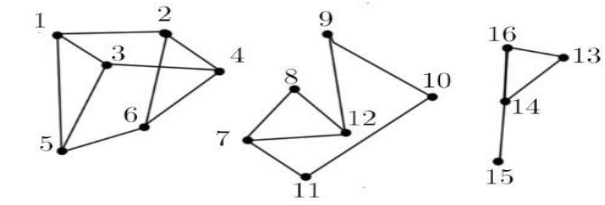


Рис. 3.105. Граф до задачі 3

4. Визначити кількість фундаментальних циклів у графі, показаному на рис. 3.106 та побудувати підграфи, кожен з яких містить тільки один фундаментальний цикл даного графа.

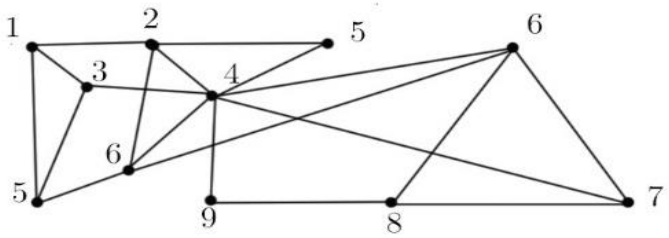


Рис. 3.106. Граф до задачі 4

5. Дано зв'язний неорієнтований зважений граф *G*, показаний на рис. 3.107. Побудувати послідовність створення остовного дерева мінімальної ваги за алгоритмом Крускала.

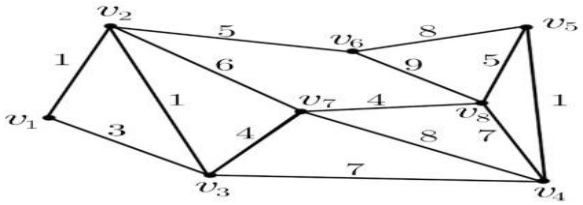


Рис. 3.107. Граф до задач 5 та 6

6. Побудувати остовне дерево мінімальної ваги за алгоритмом Прима, використовуючи граф, показаний на рис. 3.107.

3.9. Алгоритми пошуку найкоротших шляхів у графі

Задача про найкоротший шлях полягає у знаходженні найкоротшого шляху від заданої початкової вершини *a* до заданої вершини z.

Наступні дві задачі є безпосередніми узагальненнями сформульованої задачі про найкоротший шлях.

1. Для заданої початкової вершини знайти найкоротші шляхи від *a* до всіх інших вершин.
2. Знайти найкоротші шляхи між всіма парами вершин.

Розглянемо два алгоритми. Перший алгоритм розв'язує задачу номер один, другий - спеціально призначений для розв'язування задачі два.

3.9.1. Алгоритм Дейкстри

Найефективнішим алгоритмом знаходження довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої є алгоритм, запропонований 1959 р. нідерландським математиком Е. Дейкстрою (E. Dijkstra).

Цей алгоритм застосовується лише у випадку, коли вага кожної дуги додатня.

Нехай G=(V, E) – орієнтований граф, $w(v_i,v_j)$ – вага дуги (v_i,v_j) .

Пошук мінімального шляху здійснюється за допомогою присвоювання вершинам міток. Мітки є двох типів - тимчасові й постійні. Вершини з постійними мітками групують у множину M, яку називають **множиною позначених вершин**. Решта вершин має тимчасові мітки, і множину таких вершин позначають через *T* ($T=V\backslash M$).

Величина постійної мітки вершини $l(v)$ дорівнює довжині найкоротшого шляху від вершини *a* до вершини *v*. Якщо ж мітка тимчасова, то вона дорівнює довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.

Формальний опис алгоритму Дейкстри:

Крок 1. **Присвоювання початкових значень**. Виконати $l(a)=0$ і вважати цю мітку постійною. Виконати $l(v)=\infty$ для всіх $v\neq a$ і вважати ці мітки тимчасовими. Виконати $x=a$, M={*a*}.

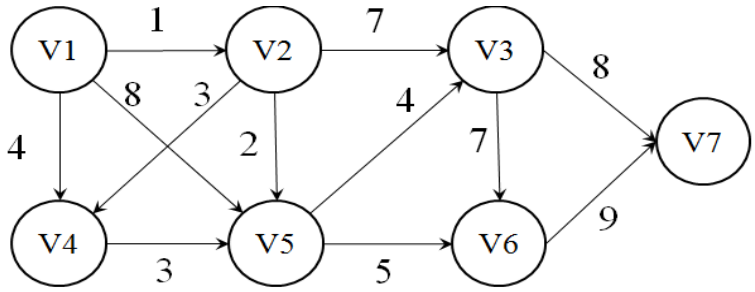
Крок 2. **Оновлення міток.** Для кожної вершини $v\in V\backslash M$ замінити мітки:
$l(v)=\min\{l(v), l(x)+w(x, v)\}$, тобто оновлювати тимчасові мітки вершин, у які з вершини *x* іде дуга.

Крок 3. **Перетворення мітки у постійну.** Серед усіх вершин з тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v^* з умови $l(v^*)=\min l(v)$, $v\in T$, $T=V\backslash M$.

Крок 4. Вважати мітку вершини v^* постійною і покласти $M=M\cup v^*$, $x=v^*$.

Крок 5. (*а*) (Якщо потрібно знайти шлях від *a* до *z*.) Якщо , $x=z$, то $l(z)$ – довжина найкоротшого шляху від *a* до *z*; зупинитись. Якщо $x\neq z$, то перейти до кроку 2.
(*б*) (Якщо потрібно знайти шлях від *a* до всіх інших вершин.) Якщо всі вершини отримали постійні мітки (включені у множину M), то ці мітки дають довжини найкоротших шляхів; зупинитись. Якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

Приклад 3.37. Знайти найкоротший шлях від вершини v_1 до вершини v_7.



Початкові значення:

$M=\{V_1\}$, $T=\varnothing$, $l(V_1)=0$, $l(v_2)= l(V_3)=\dots\ l(V_7)=\infty$

Крок 1. $T_1 = \{v_2(1, V_1), v_4(4, V_1), v_5(8, V_1)\}$ – min V_2

$M_1= \{V_1,V_2\ (1, V_1)\}$

Крок 2. $T_2= \{v_4(4, v_1), v_5(8, v_1), v_4(4, v_2), v_5(3, v_2), v_3(8, v_2)\}= \{v_4(4, v_1), V_5\ (3, V_2), V_3\ (8, V_2)\}$– min V_5

$M_2= \{V_1, V_2\ (1, V_1), V_5\ (3, V_2)\}$

Крок 3. $T_3= \{V_4\ (4, V_1), V_3\ (8, V_2), V_3\ (7, V_5), V_6\ (8, V_5)\}= \{V_4\ (4, V_1), V_3\ (7, V_5), V_6\ (8, V_5)\}$ – min V_4

$M_3= \{V_1, V_2\ (1, V_1), V_5\ (3, V_2), V_2\ (4, V_1)\}$

Крок 4. $T_4= \{V_3\ (7, V_5), V_6\ (8, V_5)\}$ – min v_3

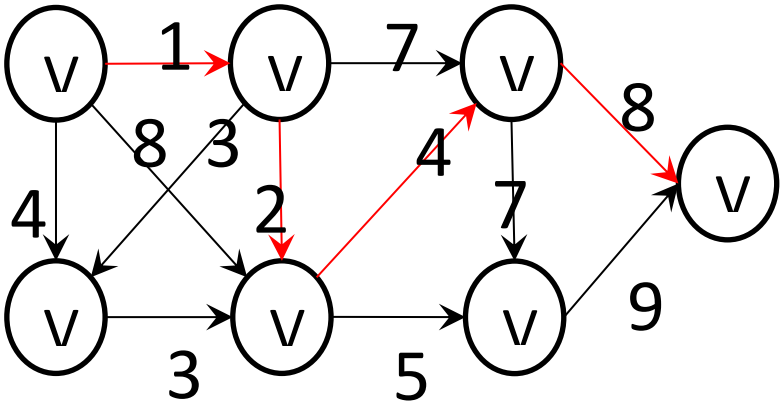
$M_4= \{V_1, V_2\ (1, V_1), V_5\ (3, V_2), V_4\ (4, V_1), V_3\ (7, V_5)\}$

Крок 5. $T_5= \{V_6\ (8, V_5), V_6\ (14, V_3), V_7\ (15, V_3)\}= \{V_6\ (8, V_5), V_7\ (15, V_3)\}$ – min V_6

$M_5= \{V_1, V_2\ (1, V_1), V_5\ (3, V_2), V_4\ (4, V_1), V_3\ (7, V_5), V_6\ (8, V_5)\}$

Крок 6. $T_6= \{V_7\ (15, V_3), V_7\ (17, V_6)\}= \{V_7\ (15, V_3)\}$

$M_6= \{V_1, V_2\ (1, V_1), V_5\ (3, V_2), V_4\ (4, V_1), V_3\ (7, V_5), V_6\ (8, V_5), V_7\ (15, V_3)\}$



Машинний алгоритм Дейкстри

Структура даних

1. Логічний масив *visited*[*n*]: *False* – вершина не розглянута; *True* –вершина розглянута.
2. Масив *nlen*[*n*] містить поточні найкоротші відстані від початкової до відповідної вершини.
3. Масив *path*[*n*] містить номери вершин. *path*[*k*] містить номер передостанньої вершини на поточному найкоротшому шляху з початкової вершини у вершину *k*.
4. *matrix* [*i, j*] матриця відстаней; $1 \le i \le n$; $1 \le j \le n$.

Кроки алгоритму Дейкстри:

1. **Ініціалізація.**

Очистимо масив *visited* для всіх вершин графа: **for** i **in range** (1, n+1): visited[i]=False.
Виберемо стартову вершину s.
Заповнимо масив шляху стартовою вершиною s.
for i **in range** (1, n+1): path[i]=s.
Перепишемо рядок стартової вершини з матриці відстаней у масив nlen
for i **in range** (1, n+1): nlen[i]=matrix [s, i].
Виконаємо початкову установку для стартової вершини s.
visited[d]=True; path[s]=0.

2. **Загальний крок**

Виконуємо перевірку на те, чи залишилися невідмічені вершини.

```
def possible ():
   r=True
   for i in range (1, n+1):
        if not visited[i]: exit
        r=False
   return=r
```

Знайдемо серед невідмічених ту вершину, що має мінімальну відстань від поточної вершини.

```
def min ():
    minvalue=infinity
    for i in range (1, n+1):
         if not visited[i]:
           if nlen[i]<minvalue:
currentmin=i
minvalue=nlen[i]
min=currentmin
```

Знайдену вершину з мінімальною відстанню від поточної позначаємо як позначену:

visited[k]=True (Позначка стає постійною)
Потім модифікуємо списки nlen й path з метою визначення суміжних вершин і відстані до них від стартової вершини.

```
for i in range (1, n+1):
    if nlen[i]>nlen[k]+mattr [i, k]:
      nlen[i]=nlen[k]+mattr [i, k]
path[i]=k
```

3. **Завершальні дії**

Якщо всі елементи списку visited дорівнюють True, тобто всі вершини графа позначені, то довжина шляху від *i v* до *k v* дорівнює nlen[k].
Вивід вершин, що входять у шлях.

```
input ('Кінцева вершина: ', finish)
print(finish)
k=finish
finish=path[pinish]
while finish! = start:
  print (' <- ', finish)
  finish=path[finish]
print (' <- ', start)
print ('Довжина шляху = ', nlen[k])
```

3.9.2. Алгоритм Форда – Беллмана знаходження мінімального шляху

Передбачається, що орієнтований граф не містить контурів від'ємної довжини.

Основними величинами, які потрібно обчислювати в цьому алгоритмі, є величини $\lambda_i(k)$, де i =1,2, …, n (n – число вершин графа); k=1,2, …, n-1. Для фіксованих i і k величина $\lambda_i(k)$ дорівнює довжині мінімального шляху, що веде із заданої початкової вершини v_1 у вершину v_i і складається з не більше ніж k дуг.

Крок 1. Установка початкових умов. Ввести кількість вершин графа n і матрицю ваг $C = |c_{ij}|$.

Крок 2. Встановити k=0. Встановити $\lambda_i(0) = \infty$ для всіх вершин, крім v_1; встановити $\lambda_1(0) = 0$.

Крок 3. У циклі по k, k=1,2,…,n-1, кожній вершині v_i на k-му кроці приписати індекс $\lambda_i(k)$ за наступним правилом: $\lambda_i(k) = \min\limits_{1 \le j \le n} \{\lambda_j(k-1) + c_{ji}\}$ для всіх вершин, крім v_1, встановити $\lambda_1(k) = 0$.

В результаті роботи алгоритму формується таблиця індексів $\lambda_i(k)$, i=1,2, …, n; k=0,1,2, …, n-1.

При цьому $\lambda_i(k)$ визначає довжину мінімального шляху з першої вершини у вершину *i,* що містить не більше, ніж *k* дуг.

Крок 4. Відновлення мініма ного шляху. Для будь-якої вершини v_i попередня до неї вершина v_r визначається зі співвідношення: $\lambda_r(n-2)+c_{rs}=\lambda_s(n-1)$. $v_r \in G^{-1}(v_s)$, де $G^{-1}(v_s)$- прообраз вершини v_s.

Для знайденої вершини v_r попередня до неї вершина v_qвизначається із співвідношення $\lambda_q(n-3)+c_{qr}=\lambda_r(n-2)$, $v_q \in G^{-1}(v_r)$, де $G^{-1}(v_r)$- прообраз вершини v_rі т.д.

Послідовно застосовуючи це співвідношення, починаючи від останньої вершини v_i, знайдемо мінімальний шлях.

Макроструктура алгоритму

Алгоритм послідовно уточнює значення функції *d* (*cur*) .

1. Спочатку задаємо значення *d* (*s*) = 0; *d* (*cur*) = ∞ для будь-якого $cur \neq s$.

2. Виконуємо *n* −1 ітерацій, під час яких виконуємо релаксацію всіх ребер графа.

Вхідні дані:

Граф: вершини *V*, ребра (*i, j*)∈ *E* з вагами *W* (*i, j*). Початкова вершина *s* .

Вихідні дані: відстані *d* (*cur*) до кожної вершини $cur \in V$ від вершини *s* .

Код алгоритму Форда – Беллмана

```
start=1
ML = 10 ** 9
d = [ML] * N
d[start] = 0
for k in range (1, N):
    for i in range(N):
        for j in range(N):
            if d[j] + W[j][i] < d[i]:
                d[i] = d[j] + W[j][i]
```

3.9.3.Алгоритм Флойда-Воршелла

Метод Флойда-Воршелла безпосередньо ґрунтується на тому факті, що в графі

з додатними вагами ребер будь-який неелементарний (довжиною більше 1 ребра) найкоротший шлях складається з інших найкоротших шляхів.

Цей алгоритм більш загальний на відміну від алгоритма Дейкстри, так як він знаходить найкоротші шляхи між будь-якии двома вузлами мережі. В цьому алгоритмі мережа представлена у вигляді квадратної матриці з *n* рядками і *n* стовпчиками. Елемент (*i, j*) дорівнює відстані d_{ij} від вузла *i* до вузла *j*, яке має

кінцеве значення, якщо існує дуга (i, j), і дорівнює нескінченості в протилежному випадку.

Покажемо спочатку ідею методу Флойда. Нехай задані три вузли i, j і k і задані відстані між ними (рис. 3.108). Якщо виконується нерівність $d_{ij} + d_{jk} < d_{ik}$, то доцільно замінити шлях $i \to k$ на шлях $i \to j \to k$. Така заміна (далі її будемо умовно називати **трикутним оператором**) виконується систематично в процесі виконання алгоритму Флойда.

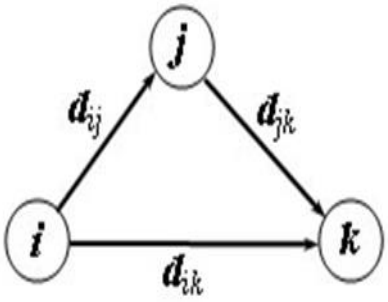


Рис.3.108. Граф G

Ідея алгоритму Флойда:

Припустимо, що нам відомі:

1. Найкоротший шлях з вершини i у вершину k, в якому як внутрішні допускають використання лише перших (k-1) вершин;
2. Найкоротший шлях з вершини k у вершину j, у якому як внутрішні допускають використання лише перших (k-1) вершин;
3. Найкоротший шлях з вершини i у вершину j, у якому як внутрішні допускають використання лише перших (k-1) вершин.

Оскільки за припущенням граф G не містить циклів від'ємної довжини, то один з двох шляхів – шлях 3) або об'єднання шляхів 1) та 2) – є найкоротшим шляхом з вершини i у вершину j, у якому як внутрішні допускають використання лише перших k вершин.

$$w_{ij}^{(k)} = \min\left\{w_{ik}^{(k-1)} + w_{kj}^{(k-1)}, w_{ij}^{(k-1)}\right\}$$

Основний алгоритм методу Флойда

Крок 0. Визначаємо вхідну матрицю відстаней D_0 і матрицю послідовності вузлів S_0. Діагональні елементи обох матриць помічаються знаком "-", який показує, що ці елементи не приймають участі в обчисленнях. Покладаємо $k = 1$:

$D_0 =$

	1	2	…	j	…	n
1	—	d_{12}	…	d_{1j}	…	d_{1n}
2	d_{21}	—	…	d_{2j}	…	d_{2n}
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
i	d_{i1}	d_{i2}	…	d_{ij}	…	d_{in}
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
n	d_{n1}	d_{n2}	…	d_{nj}	…	—

$S_0 =$

	1	2	…	j	…	n
1	—	2	…	j	…	n
2	1	—	…	j	…	n
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
i	1	2	…	j	…	n
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
n	1	2	…	j	…	—

Основний крок *k*. Задаємо рядок *k* і стовпчик *k* як **головний рядок** і **головний стовпчик**. Розглядаємо можливість застосування трикутного оператора до всіх елементів d_{ij} матриці D_{k-1}. Якщо виконується нерівність $d_{ik} + d_{kj} < d_{ij}$**,** *(i <> k, j <> k, i <> j)*, тоді виконуємо наступні дії:

- створюємо матрицю D_k шляхом заміни в матриці D_{k-1} елемента d_{ij} на суму $d_{ik} + d_{kj}$,
- створюємо матрицю S_k шляхом заміни в матриці S_{k-1} елемента s_{ij} на *k*. Покладаємо

k = *k* + 1 і повторюємо крок *k*.

- Пояснимо дії, що виконуються на *k*-му кроці алгоритму, представивши матрицю D_{k-1} так, як вона показана на рисунку 3.109. На цьому рисунку рядок *k* і стовпчи *k* є головними. Рядок *i* – будь-який рядок з номером від 1 до *k* - 1, а рядок *p* – довільний рядок з номером від *k* + 1 до *n*. Аналогічно стовпчик **j** представляє будь-який стовпчик з номером від 1 до *k* - 1, стовпчик *q* - довільний стовпчик з номером від *k* + 1 до *n*. Трикутний оператор виконується наступним чином. Якщо сума елементів головних рядка і стовпчика (що показані в квадратах) менша за суму елементів, що знаходяться на перетині стовпчика і рядка (показані в колах), що відповідають головним елементам, що розглядаюься, то відстань (елемент в колі) замінюється на суму відстаней, що представлені головними елементами:

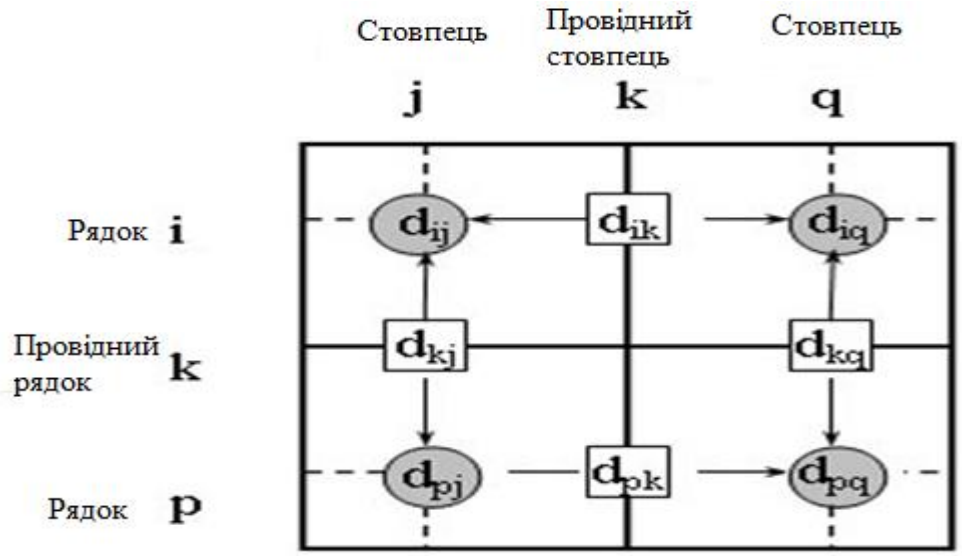


Рис.3. 109. Ілюстрація алгоритму Флойда

- Після реалізації n кроків алгоритму визначення за матрицями D_n і S_n найкоротшого шляху між вузлами i і j виконується за наступними правилами.

1. Відстань між вузлами i і j дорівнює елементу d_{ij} в матриці D_n.
2. Проміжні вузли шляху від вузла i до вузла j визначаємо за матрицею S_n. Нехай

   $s_{ij} = k$, тоді маємо шлях $i \to k \to j$. Якщо далі $s_{ik} = k$ і $s_{kj} = j$, тоді вважвємо, що весь шлях визначений, так як знайдені всі проміжні вузли. В іншому випадку повторюємо описану процедуру для шляхів від вузла i до вузла k і від вузла k до вузла j.

Приклад 3.38. Знайдемо для мережі, що показана на рисунку 3.110, найкоротші шляхи між будь-якими двома вузлами. Відстань між вузлами цієї мережі проставлені на рисунку біля відповідних ребер. Ребро (3, 5) орієнтоване, тому не допускається рух від вузла 5 до вузла 3. Всі інші ребра допускають рух в обох напрямках:

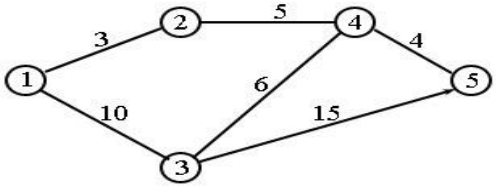


Рис.3.110. Граф G

Крок 0**.** Початкові матриці D_0 і S_0 будуються безпосередньо за заданою схемою мережі. Матриця D_0 симетрична, за виключенням пари елементів d_{35} і d_{53}, де d_{53} дорівнює нескінечнності, оскільки неможливий перехід від вузла 5 до вузла 3:

D_0

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	∞	5	∞
3	10	∞	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

S_0

	1	2	3	4	5
1	—	2	3	4	5
2	1	—	3	4	5
3	1	2	—	4	5
4	1	2	3	—	5
5	1	2	3	4	—

Крок 1. В матриці D_0 виділені головні рядок і стовпчик (**k = 1**). Подвійною рамкою представлені елементи d_{23} і d_{32}, єдині серед елементів матриці D_0, значення яких можна покращати за допомогою трикутного оператора. Таким чином, щоб на основі матриць D_0 і S_0 отримати матриці D_1 і S_1, виконуємо наступні дії.

1. Замінюємо d_{23} на $d_{21} + d_{13} = 3 + 10 = 13$ і встановлюємо $s_{23} = 1$.
2. Замінюємо d_{32} на $d_{31} + d_{12} = 10 + 3 = 13$ і встановлюємо $s_{32} = 1$.

- Матриці D_1 і S_1 мають наступний вигляд:

D_1

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	**13**	5	∞
3	10	**13**	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

S_1

	1	2	3	4	5
1	—	2	3	4	5
2	1	—	**1**	4	5
3	1	**1**	—	4	5
4	1	2	3	—	5
5	1	2	3	4	—

Крок 2. Покладаємо k = 2; в матриці D_1 виділені наступні рядок і стовпчик. Трикутний оператор застосовується до елементів матриць D_1 і S_1, що виділені подвійною рамкою. В результаті отримуємо матриці D_2 і S_2:

D_2

	1	2	3	4	5
1	—	3	10	**8**	∞
2	3	—	13	5	∞
3	10	13	—	6	15
4	**8**	5	6	—	4
5	∞	∞	∞	4	—

S_2

	1	2	3	4	5
1	—	2	3	**2**	5
2	1	—	1	4	5
3	1	1	—	4	5
4	**2**	**2**	3	—	5
5	1	2	3	4	—

Крок 3. Покладаємо k = 3; в матриці D_2 виділені головні рядок і стовпчик. Трикутний оператор застосовується до елементів матриць D_2 і S_2, що виділені подвійною рамкою. В результаті отримуємо матриці D_3 і S_3:

D_3

	1	2	3	4	5
1	—	3	10	8	**25**
2	3	—	13	5	**28**
3	10	13	—	6	15
4	8	5	6	—	4
5	∞	∞	∞	4	—

S_3

	1	2	3	4	5
1	—	2	3	2	**3**
2	1	—	1	4	**3**
3	1	1	—	4	5
4	2	2	3	—	5
5	1	2	3	4	—

Крок 4. Покладаємо $k = 4$, головні рядок і стовпчик в матриці D_3 виділені. Отримуємо нові матриці D_4 і S_4:

D_4

	1	2	3	4	5
1	—	3	10	8	**12**
2	3	—	**11**	5	**9**
3	10	**11**	—	6	**10**
4	8	5	6	—	4
5	**12**	**9**	**10**	4	—

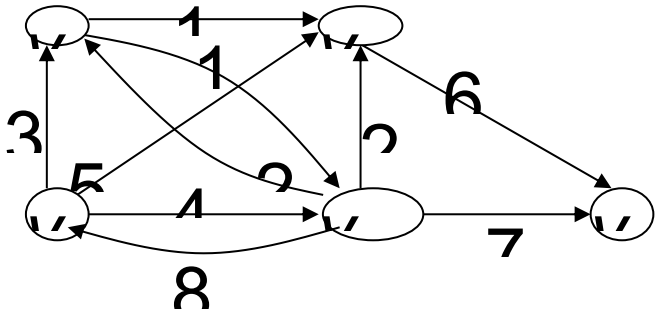
S_4

	1	2	3	4	5
1	—	2	3	2	**4**
2	1	—	**4**	4	**4**
3	1	**4**	—	4	**4**
4	**2**	2	3	—	5
5	**4**	**4**	**4**	4	—

Крок 5. Покладаємо $k = 5$, головні рядок і стовпчик в матриці D_4 виділені. Ніяких дій на цьому кроці не виконуємо; обчислення закінчені.

- Кінцеві матриці D_4 і S_4 мають всю інформацію, необхідну для визначення найкоротших шляхів між будь-якими двома вузлами мережі. Наприклад, найкоротша відстань між вузлами 1 і 5 дорівнює $d_{15} = 12$.
- Для знаходження відповідних маршрутів нагадаємо, що сегмент маршруту (i, j) складається з ребра (i, j) тільки в тому випадку, коли $s_{ij} = j$. В протилежному випадку вузли i і j зв`язані, меншою мірою, через один проміжний вузол. Наприклад, оскільки $s_{15} = 4$ і $s_{45} = 5$, спочатку найкоротший маршрут між вузлами 1 і 5 буде мати вигляд $1 \to 4 \to 5$. Але так як $s_{14}$ не дорівнює 4, вузли 1 і 4 у визначеному шляху не зв`язані одним ребром (але у вхідній мережі вони можуть бути зв`язані безпосередньо). Далі слід визначити проміжний вузол (вузли) між першим і четвертим вузлами. Маємо $s_{14} = 2$ і $s_{24} = 4$, тому маршрут $1 \to 4$ замінюємо $1 \to 2 \to 4$. Оскільки $s_{12} = 2$ і $s_{24} = 4$, інших проміжних вузлів немає. Комбінуючи визначені сегменти маршруту, нарешті отримаємо наступний найкоротший шлях від вузла 1 до вузла 5: $1 \to 2 \to 4 \to 5$. Довжина цього шляху дорівнює 12 кілометрам.

Приклад 3.39. Знайти найкоротший шлях між всіма парами вершин.



атриця суміжності ваг

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{pmatrix} 0 & 3 & 4 & 5 & \infty \\ \infty & 0 & 1 & 1 & \infty \\ 8 & 2 & 0 & 2 & 7 \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \end{array} \quad k=1$$

$$\begin{pmatrix} 0 & 3 & 4 & 5 & \infty \\ \infty & 0 & 1 & 1 & \infty \\ 8 & 2 & 0 & 2 & 7 \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad k=2$$

$$\begin{pmatrix} 0 & 3 & 4 & (4) & \infty \\ \infty & 0 & 1 & 1 & \infty \\ 8 & 2 & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad k=3$$

$$\begin{pmatrix} 0 & 3 & 4 & 4 & \infty \\ (9) & 0 & 1 & 1 & \infty \\ 8 & 2 & 0 & 2 & \infty \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad k=4$$

Матриця маршруів

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & (2) & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 1 \\ (3) & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 3 & 4 & 4 & (10) \\ 9 & 0 & 1 & 1 & (7) \\ 8 & 2 & 0 & 2 & (8) \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad k=5 \quad \begin{pmatrix} 1 & 1 & 1 & 2 & (4) \\ 3 & 2 & 2 & 2 & (4) \\ 3 & 3 & 3 & 3 & (4) \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Відповідь:

$$\begin{pmatrix} 0 & 3 & 4 & 4 & 10 \\ 9 & 0 & 1 & 1 & 7 \\ 8 & 2 & 0 & 2 & 8 \\ \infty & \infty & \infty & 0 & 6 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 2 & 4 \\ 3 & 2 & 2 & 2 & 4 \\ 3 & 3 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

3.9.4. Алгоритм Джонсона

Алгоритм Джонсона знаходить найкоротший шлях між всіма парами вершин у зваженому графі з від`ємними вагами без негативних контурів. Давайте розглянемо умови задачі частинами.

Граф, в якому кожне ребро має напрямок, називається орієнтованим (або коротко – орграфом), а його ребра називаються дугами. Наприклад, візьмемо орграф з 4 вершин і 8 дуг:

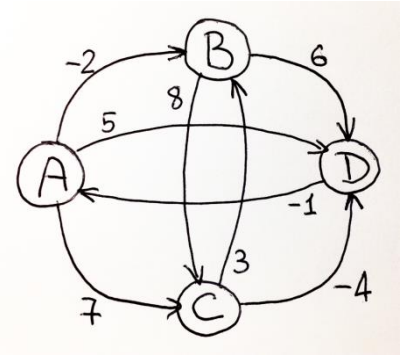


Рис.3.111. Орграф G

Ми можемо переміщуватися з однієї вершини в іншу по дугам у вказаному напрямку. Переміщення по одній або декількам дугам називається «шлях» або «маршрут».

Для кожної дуги може бути вказане число, яке називається «вага» або «довжина» дуги. Ми будемо дивитись на кожне число, як на довжину. Мета алгоритму – знайти найкоротший шлях між будь-якими двома вершинами.

Ви, напевно, вже помітили, що деякі дуги мають від`ємну вагу. Подорож по таким дугам приносить дохід. Мрією мандрівника було б знайти замкнений шлях з від`ємною довжиною і ходити по ньому вічно. Замкнений шлях називається «циклом» або «контуром», і якщо в орграфі є такий негативний

контур, то рішення задачі відсутнє, так як мінімальна довжина шляху може досягти мінус бескінечності.

Отже, ми розібрали всі терміни умови, сформулюємо мету алгоритму ще раз, більш конкоретно: Алгоритм Джонсона знаходить найкоротший шях між всіма парами вершин у зваженому орієнтованому графі з від`ємними вагами без негативних контурів.

Для вирішення цієї задачі можна застосувати алгоритм Флойда-Воршелла, який вирішує задачу «в лоб» повним перебором:

for k = 1 to n // n – кількість вершин в графі

for x = 1 to n

for y = 1 to n

$W[x][y] = \min(W[x][y], W[x][k] + W[k][y])$

Значення $W[x][y]$ елементумістить довжину найкоротшого шляху з вершини x в вершину y.

Початкові значення для W – це матриця суміжності вихідного графа. Замість нулів треба записати плюс бескінечність, так як шлях між такими вершинами ще не знайдений.

Суть алгоритму в постійному «релаксуванні» довжини маршруту. На кожній ітерації минамагаємось дістатися з X в Y через проміжну вершину K, іякщо цей шлях коротший – запам`ятовуємо зменшену довжину.

Алгоритм Флойда-Воршелла перебирає всі можливі варіанти побудови шляхів, складність в нього – кубічна, і якщо в графі вершин багато – він буде працювати дуже довго.

Ще є ефективний алгоритм Дейкстри для пошуку найкоротшого шляху від однієї вершини до всіх інших. Якщо застосувати алгоритм Дейкстри для кожної вершини, то ми отримаємо найкоротший шлях для всіх пар вершин, причомузначно швидше, ніж в алгоритміФлойда-Воршелла.

Однак, алгоритм Дейкстри некорректно працює з графом з від`ємними вагами.

Наприклад, при пошуку від вершини A, на першому етапі буде знайдений найкоротший шлях в вершину B (-2), а на другому етапі «найкоротшим» буде шлях з B в D (-2+6=4). На цьому алгоритм відрапортує про результати і закінчить роботу. Від`ємна дуга CD даже не буде розглянута і правильна відповідь не буде знайдена. Ось так. Один алгоритм повільний, другий некоректний. Що ж робити?

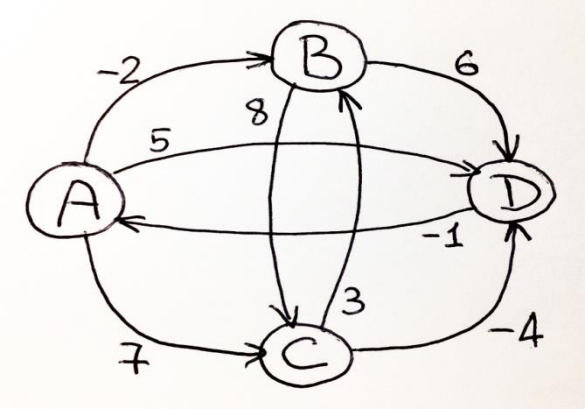
Відповідь очевидна: застосувати алгоритм Джонсона! Ідея геніальна: на основі даного орграфа сформувати новий орграф таким чином, щоб в ньому не було від`ємних дуг, а всі найкоротші шляхи були такими ж. Пропустити отриманий орграф через алгоритм Дейкстри і на виході отримати вірну відповідь! Як же нам сформувати такий орграф?

Просте невірне рішення – збільшити довжину кожної дуги на константне значення, щоб від`ємних значень не залишилось. Чому цей варіант некоректний? Тому що він дає фору більш коротким маршрутам, безпідставно збільшуючи сумарну довжину на ту саму константу з кожною новою дугой.

Наприклад, якщо в запропонованому графі довжину кожної дуги збільшити на 4, то найкоротшим шляхом з A в D стане прямий шлях: 5 + 1 * 4 = 9. Тоді як правильна відповідь з 3 дуг (A-B-C-D) отримає зайвих 12 грамів навантаження і випаде з конкуренції: -2 + 8 – 4 + 3 * 4 = 14.

Отже, наша задача – змінити довжину кожної дуги таким чином, щоб позбутися від`ємних дуг і щоб всі найкоротші відстані залишилися такими ж самими. Як це організувати? Давайте до довжини кожної дуги XY додамо $h(X)$ і віднімемо $h(Y)$,где $h(v)$ – «чиста» функція, яка визначає де яке константне значення для кожної вершини. Отримаємо такі довжини дуг:

$$A \to D$$
$$|AD| = 5$$
$$|ABD| = -2 + 6 = 4$$
$$|ACD| = 7 - 4 = 3$$
$$|ABCD| = -2 + 8 - 4 = 2$$
$$|ACBD| = 7 + 3 + 6 = 16$$



$$|AC| = 7 + h(A) - h(C)$$
$$|AB| = -2 + h(A) - h(B)$$
$$|AD| = 5 + h(A) - h(D)$$
$$|BC| = 8 + h(B) - h(C)$$
$$|BD| = 6 + h(B) - h(D)$$
$$|CB| = 3 + h(C) - h(B)$$
$$|CD| = -4 + h(C) - h(D)$$
$$|DA| = -1 + h(D) - h(A)$$

Подивимось, як в цьому випадкузміниться довжина кожного маршруту з вершини A в D:

$$|AD| = 5 + h(A) - h(D)$$
$$|ABD| = -2 + h(A) - h(B) + 6 + h(B) - h(D) = 4 + h(A) - h(D)$$
$$|ACD| = 7 + h(A) - h(C) - 4 + h(C) - h(D) = 3 + h(A) - h(D)$$
$$|ABCD| = -2 + h(A) - h(B) + 8 + h(B) - h(C) - 4 + h(C) - h(D)$$
$$= 2 + h(A) - h(D)$$
$$|ACBD| = 7 + h(A) - h(C) + 3 + h(C) - h(B) + 6 + h(B) - h(D)$$
$$= 16 + h(A) - h(D)$$

Як бачимо, будь-який шлях з вершини A в D змінюється на однакову величину, $h(A) - h(D)$, а, отже, всі найкоротші шляхи залишаться такими ж самими! Як раз те, що потрібно. Залишилось тепер знайти значення функції h, що підходить, щоб модифіковані довжини дуг стали невід`ємними.

Розглянемо, як можна розрахувати такі значення. Для цього додамо в орграф нову вершину-джерело S, з якої проведемо нульові дуги увсі вершини графа.Звернемо увагу, що додавання вершиниS і дуг S* не змінює найкоротші шляхи в вихідному графі, тому як всі додані дуги йдуть тільки з S, а «нульового» маршруту в зворотньому напрямку немає.

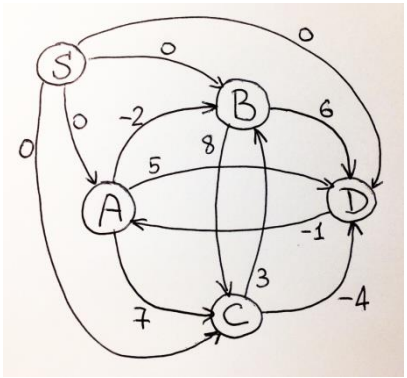
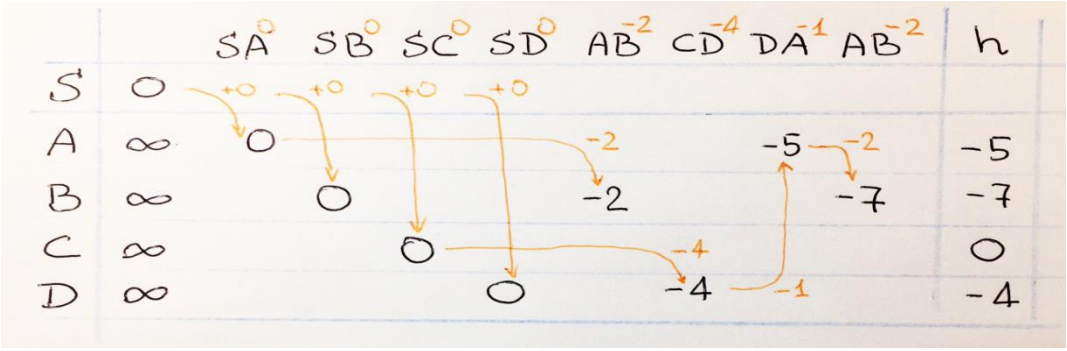


Рис.3.112. Граф G з вершиною-джерелом S

Тепер запустимо алгоритм Беллмана-Форда, який знайде найкоротші шляхи від S до всіх інших вершин. Цей алгоритм N разів поспіль перебирає всі дуги на предмет «релаксації» найкоротших маршрутів з S у всі інші вершини. В таблиці перераховані «значимі» ітерації роботи цього алгоритму, в кожному стовпці вказана чергова дуга, яка скорочує маршрут від однієї з вершин:



Результат роботи цього алгоритму як раз і дасть нам шукане значення функції h для кожної вершини. Суть цихзначень – найкоротший шлях з вершини S. Оскільки всі дуги з S нульові, то і значення функції h – недодатнє. Але нехай це нас не спантеличує, головне, w що отримані значення зафіксовані, не змінюються, а, значить, ми можемо перерахувати довжинивсіх дуг вихідного орграфа:

$$\hat{w}(x,y) = w(x,y) + h(x) - h(y)$$

$$|AC| = 7 + h(A) - h(C) = 7 + (-5) - (0) = 2$$

$$|AB| = -2 + h(A) - h(B) = -2 + (-5) - (-7) = 0$$

$$|AD| = 5 + h(A) - h(D) = 5 + (-5) - (-4) = 4$$

$$|BC| = 8 + h(B) - h(C) = 8 + (-7) - (0) = 1$$

$$|BD| = 6 + h(B) - h(D) = 6 + (-7) - (-4) = 3$$

$$|CB| = 3 + h(C) - h(B) = 3 + (0) - (-7) = 10$$

$$|CD| = -4 + h(C) - h(D) = -4 + (0) - (-4) = \otimes$$

$$|DA| = -1 + h(D) - h(A) = -1 + (-4) - (-5) = 0$$

В результатіми отримаємо орграф без від`ємних дуг:

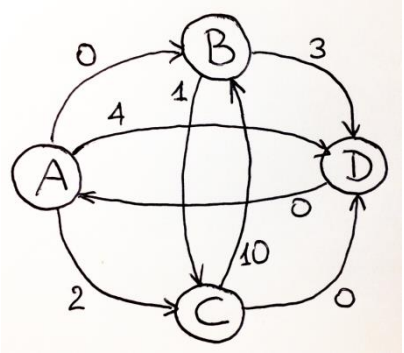
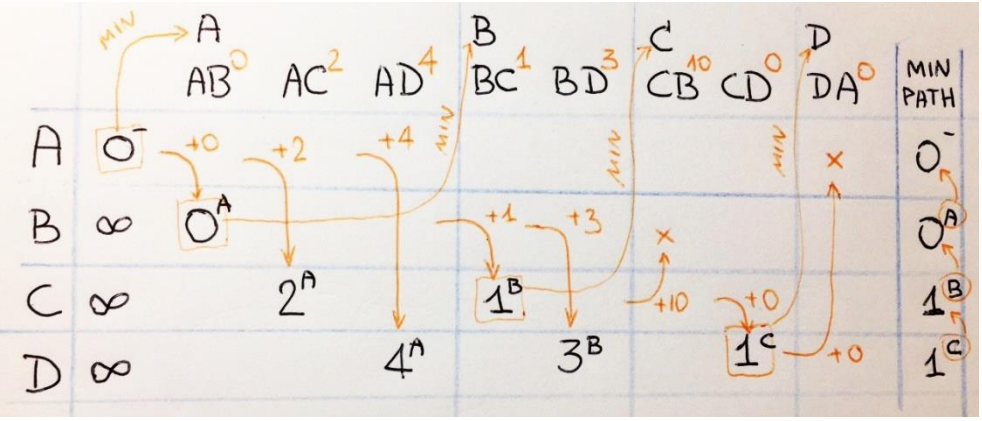


Рис. 3.113. Граф G без від’ємних дуг

Тут немає від`ємних дуг, а всі найкоротші маршрути такі ж, як і в вихідному орграфі, здорово! Тепер ми можемо зі спокійною душею запустити

алгоритм Дейкстри на цьому орграфі і для кожної вершиниіотримати правильні відповіді.

Подивимось, як будуть знайдені найкоротші маршрути з вершини *A* до всіх інших:



В кожному стовпці, поруч з новим значенням найкоротшого маршруту, записана вершина, звідки ми в неї прийшли. По цим літерам ми можемо встановити найкоротший шлях. Наприклад, найкоротший маршрут з *A* в *D* встановлюється за останнім стовпчиком і записується справа наліво: $A \Leftarrow B \Leftarrow C \Leftarrow D$, що і треба було знайти.

Складність

Алгоритм Джонсона працює за $O(VE+VD)$, де $O(D)$ — час роботи алгоритму Дейкстри. Якщо в алгоритміДейкстри неспадаюча черга з пріорітетами реалізована у вигляді фібоначчієвої купи, то час роботи алгоритму Джонсона - $O(V2\log V+VE)$. У випадку реалізації черги з пріоритетами у вигляді двоїчної купи - час роботи дорівнює $O(VE\log V)$.

Обов'язкові завдання.

1. Знайти найкоротший шлях з вершини v_1 до вершини v_5 з використанням алгоритму

Дейкстри для орграфа *G*, який задано ваговою матрицею.

	v_1	v_2	v_3	v_4	v_5
v_1	0	5	0	0	0
v_2	0	0	0	4	2
v_3	0	0	0	0	1
v_4	0	0	2	0	0
v_5	3	0	0	0	0

2. Визначити мінімальний шлях з вершини v_1 до вершини v_6 за допомогою алгоритму Форда – Беллмана у орграфі *G*, який показаний на рис. 3.114.

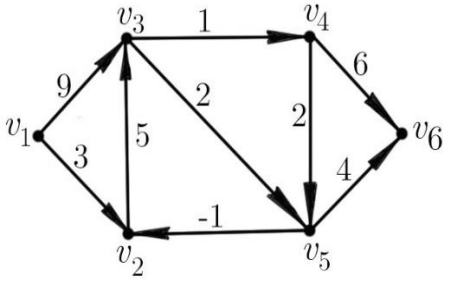


Рис.3.114. Граф *G* до задачі 2

3. Визначити найкоротший шлях між усіма вершинами за допомогою алгоритму Флойда – Воршелла у графі, заданому на рис. 3.115.

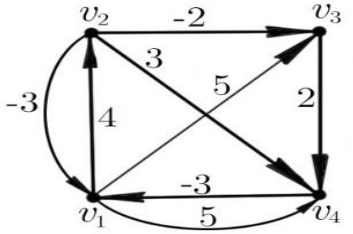


Рис. 3 115. Граф *G* до задачі 3

3.10. Розфарбування графа

3.10.1. Задачі розфарбування

Задачі розфарбування вершин або ребер графа займають важливе місце в теорії графів.

До задачі розфарбування графа зводиться цілий ряд практичних задач. Одна з областей – складання розкладів:

• розкладу для освітніх закладів;
• розкладу в спорті;
• планування зустрічей, зборів, інтерв'ю;
• розкладу транспорту, у тому числі – авіатранспорту;
• розкладу для комунальних служб;
• інші.

3.10.2. Основні визначення

Нехай $G = (V, E)$ – скінченний граф, а k – деяке натуральне число.

Вершинне розфарбування

Довільну функцію виду $f : V \to N_k$ де $N_k = \{1,2,...k\}$, $|V| = 4$, N_2 називають вершинним k -розфарбуванням, або просто k -розфарбуванням графа G .

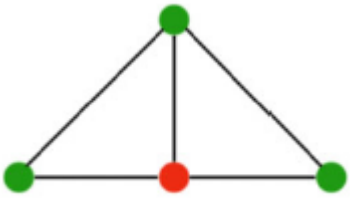


Рис. 3.116. Вершинне 2-розфарбування графа G

Правильне розфарбування

Розфарбування називають правильним, якщо кольори суміжних вершин не співпадають, тобто для будь-яких $(u, v) \in E \Rightarrow f(u) \neq f(v)$. $|V| = 4$, N_3

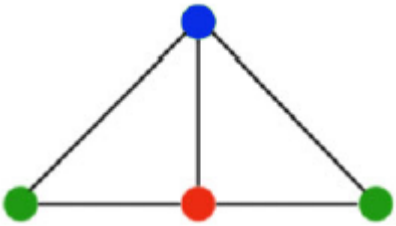


Рис. 3.117. Вершинне правильне 3-розфарбування графа G

Розфарбований граф. Граф, для якого існує правильне k -розфарбування, називають **розфарбованим графом.**

Базовий принцип оптимізації розфарбування

• Якщо функція f не взаємно однозначна, то при $V = k$ фактично може бути використано менше, ніж k кольорів.

• Правильне розфарбування – це розбиття множини вершин.

Правильне k -розфарбування можна розглядати як розбиття множини вершин V графа G на класи

o $V_1 \cup V_2 \cup \ldots \cup V_l = V$, де $l \leq k$,

o $V_i \neq \emptyset$, $i = 1, 2, \ldots, l$.

Кожний клас i V_i– це незалежна множина. Такі класи називають **кольоровими класами.**

3.10.3. Хроматичне число

Визначення. Мінімальне число k, при якому існує правильне k - розфарбування графа G, називають **хроматичним числом** цього графа і позначають $X_p(G)$.

Визначення. Якщо $X_p(G) = k$, то граф G називають **k -хроматичним**. Тобто

його вершини можна розфарбувати k різними кольорами так, що у будь-якого ребра інцидентні вершини матимуть різний колір.

Визначення. Правильне k -розфарбування графа G при $k = X_p(G)$ називають **мінімальним.**

Визначення. Хроматичне число незв'язного графа дорівнює максимальному з хроматичних чисел його компонент зв'язності.

Приклад 3.40. Розглянемо граф G, зображений на рис. 3.118, на якому показано одне із правильних k -розфарбувань. Натуральними числами 1,2,3,4 позначені кольори відповідних вершин.

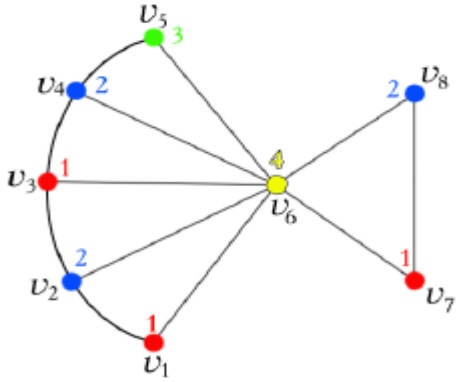


Рис. 3.118. Розфарбований граф G

Хроматичні числа деяких графів

Для деяких простих графів неважко знайти хроматичні числа.

1. Повний граф K_n, що складається з n вершин, має хроматичне число $X_p(K_n) = n$

Приклад 3.41. Знайти хроматичні числа правильних графів K_2, K_3, K_4, K_5.

Розв'язок. Для знаходження хроматичних чисел необхідно виконати мінімальне правильне розфарбування відповідних графів, як показано на рис. 3.119.

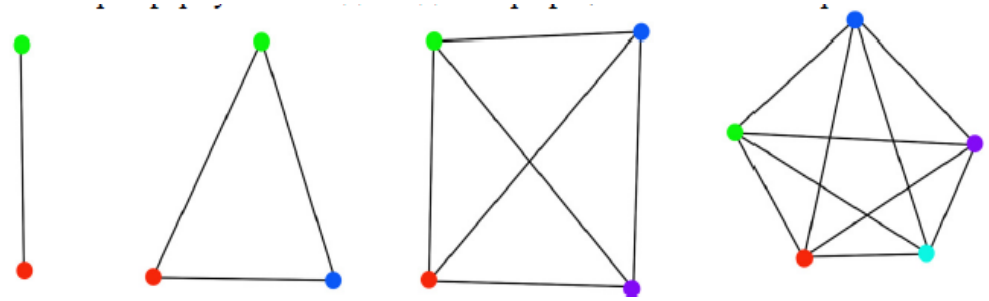


Рис. 3.119. Мінімальне правильне розфарбування повних графів

Повний граф $K_n - e$, який складається з n вершин з одним відсутнім ребром, має хроматичне число $X_p(K_n - e) = n - 1$.

Приклад 3.42. Знайти хроматичні числа графів $K_3 - 1, K_4 - 1, K_5 - 1$.

Розв'язок. Для знаходження хроматичних чисел необхідно виконати мінімальне правильне розфарбування графів, як показано на рис. 3.120.

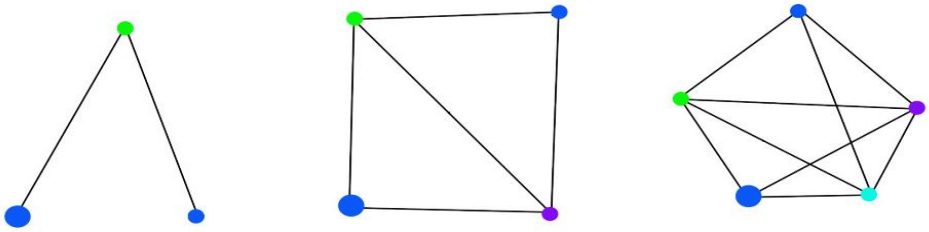


Рис. 3.120. Мінімальне правильне розфарбування графів 3.

Повні дводольні графи $K_{m,n}$, що складаються з долей $|A| = m$і $|B| = n$, мають хроматичне число $X_p(K_{m,n}) = 2$.

Приклад 3.43. Знайти хроматичні числа графів $K_{1,2}, K_{2,2}, K_{2,3}, K_{3,3}$.

Розв'язок. Для знаходження хроматичних чисел необхідно виконати мінімальне правильне розфарбування біхроматичних графів, як показано на рис. 3.121.

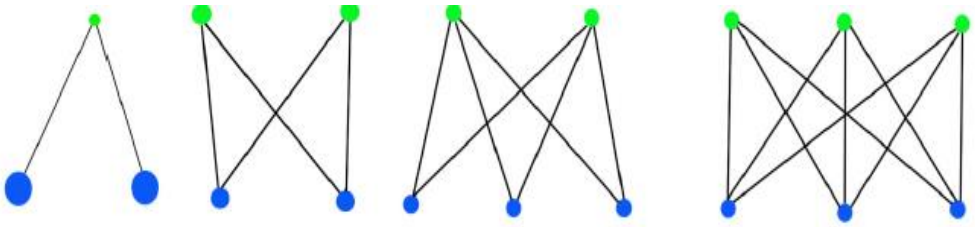


Рис. 3.121. Мінімальне правильне розфарбування біхроматичних графів

Теорема 3.17. Непорожній граф є **біхроматичним** тоді й тільки тоді, коли він
не має циклів непарної довжини.

Приклад 3.44. Навести приклади 1-хроматичного, 2-хроматичного і 3-хроматичного графів.

*Розв'язок.*1-*хроматичний граф* – порожній граф, показаний на рис. 3.122.

Рис. 3.122. 1-хроматичний граф

2-*хроматичний граф* – дводольний непорожній граф, показаний на рис. 3.123. 2-хроматичні графи, як правило, називають **біхроматичними**.

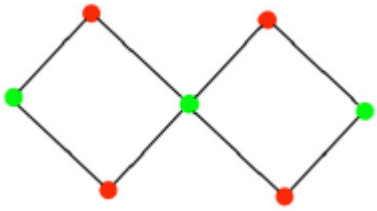


Рис. 3.123. 2-хроматичний граф

3-*хроматичний граф* – циклічний граф з непарним числом вершин у кожному з циклів, показаний на рис. 3.124.

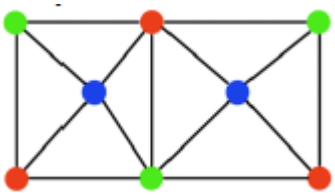


Рис. 3.124. 3-хроматичний граф

Визначення. Якщо граф має n вершин, то його хроматичне число не перевищує n.

Визначення. Якщо граф має підграф K_m, то його хроматичне число не менше, ніж m.

3.10.4. Хроматичне число й стандартні характеристики

У загальному випадку хроматичне число графа не можна обчислити, знаючи тільки його стандартні числові характеристики: число вершин, ребер, компонент зв'язності, розподіл степенів вершин.

Розглянемо графи G_1 й G_2, показані на рис. 3.125.

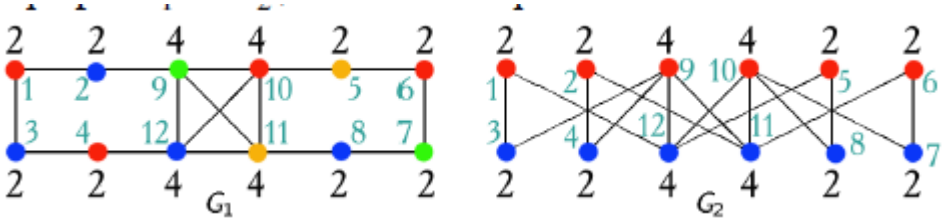


Рис. 3.125. Графи G_1 і G_2

Кожний з них має 12 вершин, у тому числі 4 вершини зі степенем 4 і 8 вершин зі степенем 2, 16 ребер, 1 компонент зв'язності. Але, як видно з рисунка, $X_p(G_1)=4$, а $X_p(G_2)=2$.

Оскільки G_1 містить K_4 підграф, то $X_p(G_1)=4$. Оскільки граф G_2 – дводольний, маємо $X_p(G_2)=2$.

Тому надалі вестимемо мову про оцінки, а не про точні значення хроматичного числа.

3.10.5. Хроматичне число й щільність графа. Три нижні оцінки хроматичного числа

Під **нижніми оцінками** хроматичного числа будемо розуміти нерівності виду $X(G)\geq c$, де c – деяка константа, що обчислюється на графі G.

Верхня оцінка хроматичного числа – це нерівності виду $X(G)\leq c$, де c – деяка константа, що обчислюється на графі G.

Визначення. Максимальне число вершин, що утворюють повний підграф у графі G, називають щільністю G і позначають через $\omega(G)$.

Повний підграф деякого графа G – це підграф, що складається з попарно суміжних вершин.

Перша нижня оцінка може застосовуватися у випадку, якщо підграфом деякого графа є повний підграф.

Перша нижня оцінка

Для довільного графа G справедлива нерівність $X(G)\geq\omega(G)$.

Визначення. Будь-яку множину попарно несуміжних вершин графа G називають **незалежною множиною.**

Визначення. Максимальне число вершин у незалежній множині називають **числом незалежності** графа G й позначають через $\beta(G)$.

Число незалежності графа – це поняття, протилежне за змістом поняттю щільності графа. Якщо G – звичайний граф, а $\overline{G}$ – його доповнення, то $\beta(G)=\omega(\overline{G})$.

Друга нижня оцінка

Для довільного графа G справедлива нерівність $X(G)\geq\frac{n(G)}{\beta(G)}$**,** де $n = n\ (G)$ − кількість вершин графаG,

$\beta\ (G)$ − число незалежності, яке дорівнює числу внутрішньої стійкості графа G.

Третя нижня оцінка хроматичного числа

Існують **нижні оцінки** хроматичного числа, які використовують тільки ті характеристики графа, що легко обчислюються. Наведемо без доведення одну з них.

Якщо G – звичайний граф і $n = n\ (G)$ − кількість вершин графа G, $m = m(G)$ – кількість ребер графа G, то хроматичне число $X(G)\geq\frac{n^2}{n^2-2m}$.

Легко зрозуміти, що в повному графі (як і в будь-якому звичайному графі) подвоєне число ребер менше квадрата числа вершин, і тому число, що стоїть в знаменнику в правій частині нерівності, завжди додатне.

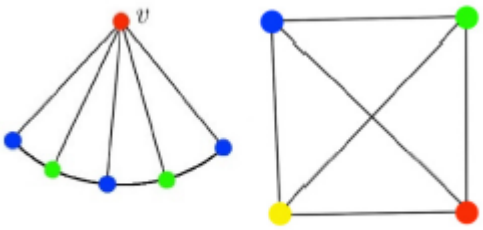
Як видно з описаних вище результатів, **задачі визначення хроматичного числа** графа й побудови мінімального розфарбування довільного графа **досить складні**, а ефективні алгоритми їх розв’язування невідомі. Розглянемо простий алгоритм побудови правильного розфарбування, який у деяких випадках дає розфарбування, близькі до мінімальних.

3.10.6. Верхня оцінка хроматичного числа

Теорема. Для будь-якого графа G наявна нерівність $X_p(G)\leq r+1$, де $r=\max_{v\in V}(\deg(v))$.

Приклад 3.45. Перевірити правильність верхньої оцінки хроматичного числа

для графів, зображених на рис. 3.126.



а б

Рис. 3.126. Графи для визначення верхньої оцінки хроматичного числа

Розв'язок. Розглянемо граф, що показаний на рис. 3.126 а.

1. Знаходимо вершину графа, яка має максимальний степінь. $r = \max\limits_{v \in V}(\deg(v)) = 5$

2. Визначаємо верхню оцінку хроматичного числа $X_p(G) \le r + 1$.

Оскільки $r = 5$, то $X_p(G) \le 6$. З рис. 3.126 а видно, що $X_p(G) = 3$. Оскільки $3 < 6$, то оцінка правильна. Граф, показаний на рис. 3.126 б, є регулярним графом. Тому $r = 3$. Звідси $X_p(G) \le 4$. З рис. 3.126 б видно, що $X_p(G) = 4$.

Отже, оцінка правильна і у цьому випадку.

Наслідок. Будь-який кубічний граф розфарбовується за допомогою чотирьох фарб. На рис. 3.127 наведено приклад розфарбування кубічного графа.

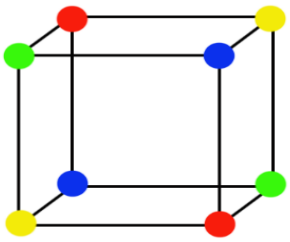


Рис. 3.127. Мінімальне правильне розфарбування куба

Теорема Брукса дає можливість більш точної верхньої оцінки хроматичного числа для спеціальних графів.

3.10.7. Теорема Брукса

Якщо G – зв'язний неповний граф і $r \ge 3$, де $r = \max\limits_{v \in V}(\deg(v))$, $X_p(G) \le r$

Приклад 3.46. Перевірити правильність верхньої оцінки хроматичного числа

для графа, зображеного на рис. 3.128.

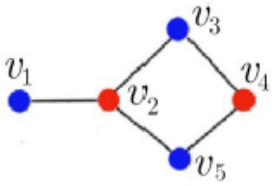


Рис. 3.128. Мінімальне правильне розфарбування графа

Розв'язок.

Розглянемо умови застосування теореми Брукса.

1. Вершина v_2 має максимальну степінь deg (v_2) = 3.

2. Граф є зв'язним та неповним. Отже, $X_p(G) \le 3$ за теоремою Брукса. Звідси можна зробити висновок, що оцінка правильна. Хоча обидві теореми й дають певну інформацію про хроматичне число графа, але їх оцінки досить неточні. Дійсно, **зірковий граф** K_{1n}, який згідно з теоремою Брукса **розфарбовується** *n* **фарбами**, насправді є **біхроматичним** (рис. 3.129).

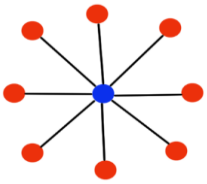


Рис. 3.129. Біхроматичний граф

Ця ситуація значно спрощується, якщо обмежитися ***планарними графами***. У цьому випадку легко довести такий досить загальний і важливий факт.

3.10.8. Теореми про шість, п'ять та чотири фарби

Теорема про шість фарб. Для будь-якого планарного (ізоморфного плоского (у якому ребра перетинаються лише у вершинах)) графа *G* вірна нерівність $X_p(G) \le 6$.

Більш детальний аналіз шляхів зниження верхньої границі хроматичного числа приводить до так званої **теореми про п'ять фарб.**

Теорема про п'ять фарб. Для будь-якого планарного графа *G* вірна нерівність $X_p(G) \le 5$.

Теорема про чотири фарби. Кожний планарный граф без петель і кратних ребер є не більш ніж 4-хроматичним. Проблема чотирьох фарб залишалася невирішеною протягом багатьох років. Стверджується, що ця теорема була доведена за допомогою певних міркувань і комп'ютерної програми в 1976 році (**Kenneth Appel and Wolfgang Haken. Every Planar Map is Four Colorable. Contemporary Mathematics 98, American Mathematical Society, 1980**).

3.10.9. Задача про розподіл устаткування

На підприємстві планують виконати 8 робіт. Для виконання цих робіт необхідні механізми. Використання механізмів для кожної з робіт визначається наступною таблицею 3.1.

Таблиця 3.1.

Таблиця використання механізмів

Механізм	Робота						
	v_1	v_2	v_3	v_4	v_5	v_6	v_7
a_1	+		+				+
a_2		+		+			
a_3			+			+	+
a_4	+	+		+	+		
a_5			+		+		
a_6					+	+	

Жоден з механізмів не може бути використаний одночасно на двох роботах. Виконання кожної роботи займає 1 годину. Як розподілити механізми, щоб сумарний час виконання всіх робіт був мінімальним і який цей час?

Розв'язок. Розглянемо граф G, показаний на рис. 3.130. Вершинами даного графа є плановані роботи $v_1, v_2, \ldots, v_8$, а ребра з'єднують роботи, у яких бере участь хоча б один загальний механізм (і які, з цієї причини, не можна проводити одночасно).

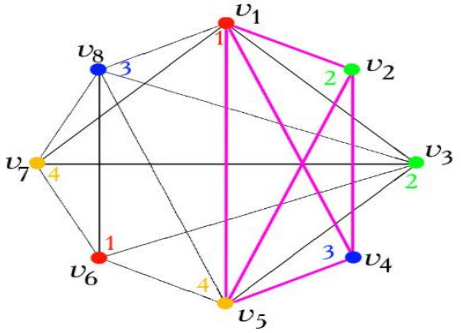


Рис. 3.130. Граф планування робіт

Вершини v_1, v_2, v_4, v_5 породжують підграф графа G, ізоморфний K_4. Отже, $X\{G\} \geq 4$. Отже, $X(G)$. Таким чином, усі роботи можна виконати за 4 години.

Для цього, відповідно до знайденого розфарбування графа G, потрібно за першу годину виконувати роботи v_1 і v_6, за другу годину — роботи v_2 і v_3, за третю годину — роботи v_4 і v_8, за четверту годину — роботи v_5 й v_7.

У таблиці 3.2 кольорами показана послідовність виконання робіт, яка забезпечує максимальне завантаження механізмів.

Таблиця 3.2.

Таблиця визначення послідовності робіт

Механізм	Робота							
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
a_1	+		+				+	+
a_2		+		+				
a_3			+			+	+	
a_4	+	+		+	+			
a_5			+		+			+
a_6					+	+		+

3.10.10. Задача складання розкладу

Умова задачі

Потрібно прочитати лекції з чотирьох предметів двом групам студентів. Деякі з лекцій не можуть бути прочитані одночасно з ряду причин, а саме:

1. Лекцію з даного предмету в різних групах читає той самий лектор.
2. Дві лекції одній і тій же групі одночасно читатися не можуть.
3. Різні лекції повинні проходити в тому самому приміщенні.

Потрібно скласти розклад так, щоб читання всіх лекцій зайняло мінімально можливий час (за «одиницю часу» у даній задачі природно розглядати одну пару).

Конкретизуємо постановку задачі

1. Будемо розглядати дві групи студентів: 1 і 2.
2. Предмети:

- обчислювальні методи – читає викладач X,
- дискретна математика – читає викладач X,
- математичний аналіз – читає викладач Y,
- українська мова – читає викладач Z.

Знайти мінімальне число пар, у які можна «укласти» усі заняття, і скласти відповідний розклад. **Тобто, створити максимально щільний розклад**

Розв'язок. Створимо граф, показаний на рис. 3.131. Вершини даного графа позначені такими символами: О1, О2, Д1, Д2, М1, М2, У1 і У2 (літера відповідає предмету, а цифра – номеру групи). З'єднаємо ребрами вершини, відповідні до пар, які не можна проводити одночасно.

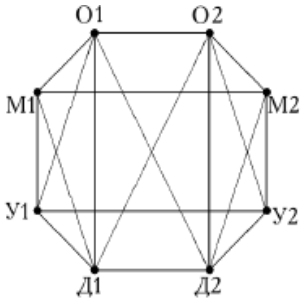


Рис. 3.131. Граф для складання розкладу

Вершини О1, О2, Д1 і Д2 цього графа породжують у ньому підграф, ізоморфний графу K_4. Отже, хроматичне число нашого графа не менше 4.

На рис. 3.132 зазначене правильне розфарбування нашого графа в 4 фарби.

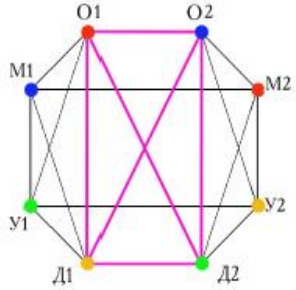


Рис. 3.132. Мінімальне правильне розфарбування графа розкладу

Отже, хроматичне число графа дорівнює 4, тобто всі заняття можна провести за 4 пари. Відповідний розклад зазначений у таблиці 3.3.

Таблиця 3.3.

Оптимальний розклад

	Група 1	Група 2
1 пара	Обч. методи	Матем. аналіз
2 пара	Матем. аналіз	Обч. методи
3 пара	Українська мова	Дискр. матем
4 пара	Дискр. матем.	Українська мова

Обов'язкові завдання

1. Для яких графів відомі точні значення хроматичних чисел?
2. Чому дорівнює хроматичне число графа *G*, показаного на рис. 3.133?

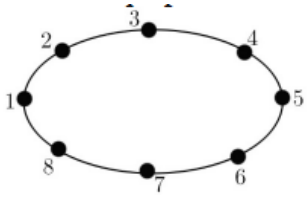


Рис. 3.133. Граф до задачі 2

3. Визначити першу нижню оцінку хроматичного числа для графа, показаного на рис. 3.134.

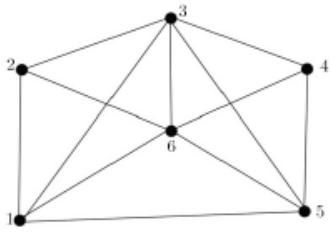


Рис. 3.134. Граф до задачі 3

4. Визначити другу та третю нижні оцінки хроматичного числа для графа, показаного на рис. 3.135.

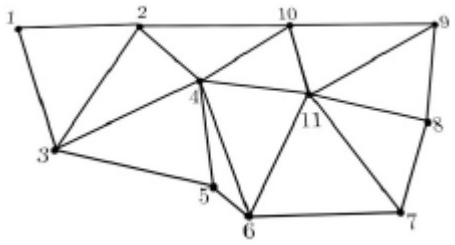


Рис. 3.135. Граф до задач 4 та 5

5. Застосувати теорему Брукса для визначення верхньої оцінки хроматичного числа для графа, зображеного на рис. 3.135.

6. Визначте точне значення хроматичного число повного графа з 15 вершин, після видалення одного ребра.

3.11. Основні алгоритми розфарбування графів

3.11.1. Базові відомості

Будемо розглядати алгоритми розфарбування на неорієнтованих графах без петель.

Граф називають **r -хроматичним**, якщо його вершини **можуть бути розфарбовані** з використанням *r* кольорів (фарб) так, що не знайдеться двох суміжних вершин одного кольору. Найменше число *r*, таке, що граф *G* є **r-хроматичним**, називають **хроматичним числом** графа і позначають *X* (*G*).

Задачу знаходження хроматичного числа графа називають **задачею про розфарбовування (або завданням розфарбовування**) графа. Відповідно до хроматичного числа розфарбування вершин розбиває множину вершин графа на *r* підмножин, кожна з яких містить вершини одного кольору. Ці множини є незалежними, оскільки в межах однієї множини немає двох суміжних вершин.

Завдання знаходження хроматичного числа довільного графа стало предметом багатьох досліджень наприкінці XIX та у XX столітті. З цього питання отримано багато цікавих результатів.

Хроматичне число графа не можна знайти, знаючи тільки кількість вершин і ребер графа. Недостатньо також знати степінь кожної вершини, щоб обчислити хроматичне число графа. При відомих величинах n (кількість вершин), m (кількість ребер) і $\deg(v_1),...,\deg(v_n)$ (степені вершин графа) можна отримати тільки верхню і нижню оцінки для хроматичного числа графа.

3.11.2. Алгоритм неявного перебору

Алгоритм прямого неявного перебору є найпростішим алгоритмом вершинного розфарбування графів. Цей алгоритм дозволяє реалізувати правильне розфарбування графа з вибором мінімальної в рамках даного алгоритму кількості фарб.

Нехай множина вершин графа довільно упорядкована, тобто для довільної i − ї вершини графа існує певний індекс v_i. Тоді всі вершини графа G утворюють множину

$V = \{v_1, v_2, ..., v_n\}$, де n −потужність множини вершин: $|V| = n$.

Тоді перше допустиме розфарбування може бути одержано так:

1. Розфарбуємо першу довільну вершину v_i кольором 1.

2. Кожну наступну вершину будемо розфарбовувати, застосовуючи таке правило:

3. Вибираємо вершину v_{i+1}:

- намагаємося використати колір 1.
- якщо колір 1 використати не вдається, то вводимо колір 2 і розфарбовуємо вершину кольором 2.

4. Вибираємо вершину v_{i+2} - намагаємося використати колір 1.

- якщо колір 1 використати не вдається, то намагаємося використати колір 2.
- якщо колір 2 використати не вдається, то вводимо колір 3 і розфарбовуємо вершину кольором 3.

5. Продовжуємо алгоритм, використовуючи такий порядок розфарбування до того моменту, коли всі вершини графа будуть розфарбовані.

Вибравши початковою іншу вершину або інакше упорядкувавши граф, можемо одержати інше розфарбування.

Кількість обчислень за даним алгоритмом може бути мінімальною, якщо вершини пронумерувати таким чином, щоб першими стояли ті вершини, які входять у **кліку** графа.

Кліка в неорієнтованому графі – це підмножина його вершин, яка утворює повний підграф.

Отже, якщо вершини $v_1, v_2, \ldots, v_l$ розфарбовані l кольорами 1,2, ..., l; $l \le i$, то новій довільно взятій вершині v_{i+1} припишемо мінімальний колір, не використаний при розфарбуванні суміжних з нею вершин.

Розфарбування, до якого приводить описаний алгоритм, називають алгоритмом прямого неявного перебору або послідовним алгоритмом розфарбування.

Розглянемо кільцевий граф, вершини якого пронумеровані у різний спосіб (рис. 3.136). У першому випадку маємо граф G_1 (а), а у другому випадку – граф G_2 (б).

а б

Рис. 3.136. Розфарбування кільцевих графів з різною нумерацією

Отримане розфарбування завжди правильне, але не завжди оптимальне навіть для простих графів. Починаючи з вершини v_1 та використовуючи алгоритм неявного перебору, одержимо розфарбування лише двома фарбами. У другому випадку, при розфарбуванні вершини v_4, виникає ситуація, коли необхідно вводити колір 3. Отже, основним недоліком алгоритму прямого неявного перебору є його залежність від нумерації вершин.

3.11.3. Програмний код алгоритму прямого неявного перебору

Програмний код написаний мовою програмування *Python*.

Він забезпечує генерацію випадкової симетричної матриці суміжності *a*[*r*] та випадкового списку кольорів *colarr*[i].

```
from random import *
n = 5 # максимальна кількість вершин графа
colarr=[0 for i in range(n)]
# Генерація симетричної матриці
a = [ ]
for r in range (n):  # n рядків
    a.append([ ]) # створили рядок
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            # додаємо однонаправлене ребро
            a[r].append(randrange(0,2))
        else:
```

```
            a[r].append(0) # протилежний напрям
# Формування протилежних напрямів
for r in range(n):
    for c in range(n):
        if a[r][c]!=0:
            a[c][r]=a[r][c]
for r in range(n): # 6 рядків
    print(a[r])
def color(i):
#Функція вибору фарби для розфарбування вершини з номером i
    w = {0};
    for j in range(i):
        if a[j][i]>0: w.add(colarr[j])
    curcol=0
    while True:
        curcol+=1
        if curcol not in w: break
    return curcol
for i in range(n):
    colarr[i]=color(i)
print(colarr)
```

Наведемо приклад розфарбування графа за даним алгоритмом.

Приклад 3.47. Розглянемо граф $G(V, E)$, показаний на рис. 3.137.

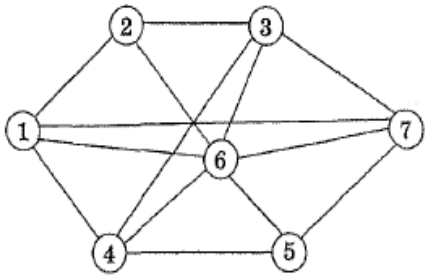


Рис. 3.137. Граф для розфарбування прямим неявним перебором

Множину вершин графа $V=\{1,2,3,4,5,6,7\}$ потрібно розфарбувати з використанням алгоритму послідовного розфарбування.

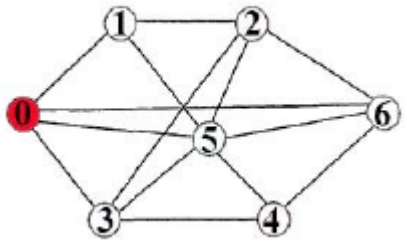
Розв'язок.

Сформуємо матрицю суміжності A:

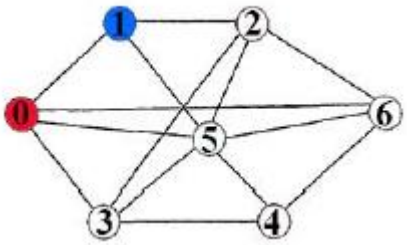
$$A = \begin{pmatrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 3 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 6 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Використовуючи дану матрицю, виконаємо алгоритм покроково.

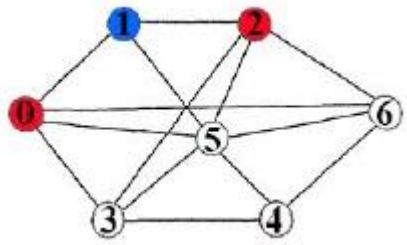
***Крок* 0**. Розглядаємо вершину 0. Множина розфарбованих суміжних вершин *w* містить колір 0. Тому функція color (0) повертає 1. Нехай колір 1- червоний.



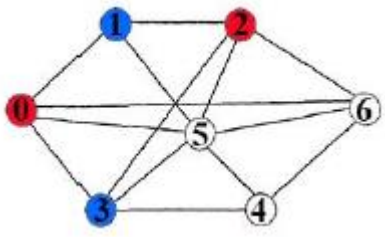
***Крок* 1**. Розглянемо вершину 1. Єдиною меншою за номером суміжною вершиною є вершина 0, яка уже червона. Тому множина w містить елементи {0,1}. Функція color (1) повертає наступну за номером фарбу синього кольору: curcol=1.



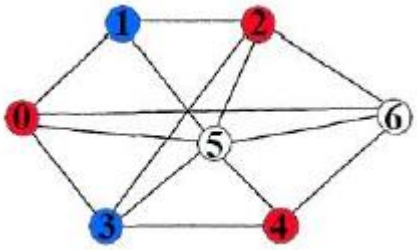
***Крок* 2**. Вершина 2 має єдину суміжну вершину 1 з меншим номером. Множина w містить елементи [0,2]. Функція color (2) повертає фарбу з номером 1 червоного кольору.



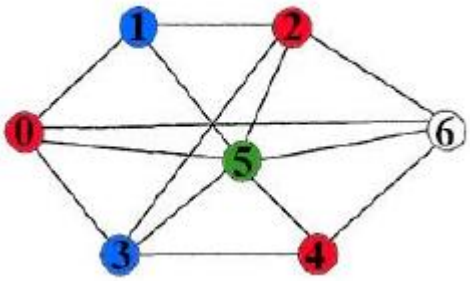
***Крок* 3**. Вершина 3 має дві суміжні вершини з меншими номерами: 0 і 2. Оскільки обидві вершини розфарбовані в колір 1, то множина w містить елементи {0,1}. Тому функція color (3) повертає наступну за номером фарбу 2 синього кольору.



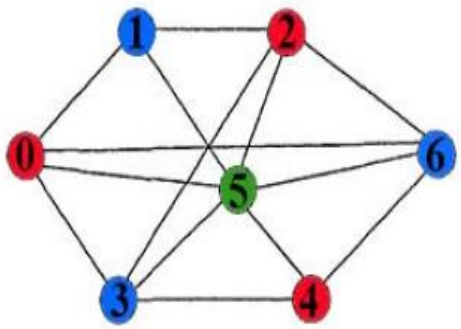
***Крок* 4**. Вершина 4 має єдину суміжну вершину з меншим номером. Це вершина 3. Множина w містить елементи {0,2}. Тому функція color (4) повертає фарбу з номером 1 червоного кольору.



***Крок* 5**. Вершина 5 має такі суміжні вершини з меншими номерами:0,1,2 і 4. Ці вершини розфарбовані в колір 1 та колір 2. Отже, множина містить елементи: {0,1,2}. Тому функція color (5) повертає наступну за номером фарбу 3 зеленого кольору.



***Крок* 6**. Вершина 6 має такі суміжні вершини з меншими номерами: 0, 2, 4 і 5. Ці вершини розфарбовані в колір 1 та колір 3. Отже, множина w містить два елементи: {0,1,2}. Тому функція color(6) повертає фарбу 2 синього кольору. В результаті роботи даного алгоритму одержуємо правильно розфарбований граф, що показаний на рисунку.



3.11.4. Евристичний алгоритм розфарбування

Точні методи розфарбовування графа складні для програмної реалізації. Однак існує багато евристичних процедур розфарбовування, які дозволяють знаходити хороші наближення для визначення хроматичного числа графа. Такі процедури також можуть з успіхом використовуватися при розфарбовуванні графів з великим числом вершин, де застосування точних методів не виправдане з огляду на високу трудомісткість обчислень.

З евристичних процедур розфарбовування слід зазначити послідовні методи, засновані на впорядкуванні множини вершин. В одному з найпростіших методів вершини спочатку розташовуються в порядку зменшення їх степенів. Перша вершина зафарбовується в колір 1, потім список вершин переглядається за зменшенням степенів, і в колір 1 афарбовується

кожна вершина, яка не є суміжною з вершинами, зафарбованими в той же колір. Потім повертаємося до першої в списку незафарбованої вершини, фарбуємо її в колір 2 і знову переглядаємо список вершин зверху вниз, зафарбовуючи в колір 2 будь-яку езафарбовану вершину, яка не з'єднана ребром з іншою, вже пофарбованою в колір 2, вершиною. Аналогічно діємо із кольорами 3, 4 і т. д., допоки не будуть пофарбовані всі вершини. Кількість використаних кольорів буде тоді наближеним значенням хроматичного числа

графа.

Послідовність дій в евристичному алгоритмі має вигляд:

1. Упорядкувати вершини за спаданням степеня.
2. Вибрати колір фарбування 1.
3. Розфарбувати першу вершину в колір 1.
4. Поки не пофарбовані всі вершини, повторювати п. 4.1–4.2:

4.1. Розфарбувати в обраний колір кожну вершину, яка не суміжна з іншою вершиною, вже пофарбованою в цей колір.

4.2. Вибрати наступний колір.

5. Повернутися до першої в списку нерозфарбованої вершини. Число використаних кольорів буде тоді наближеним значенням хроматичного числа графа.

На рис. 3.138 показано приклад вибору кольору вершини v_{i+1} за евристичним алгоритмом.

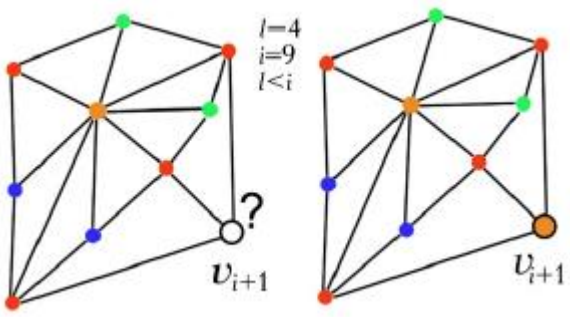


Рис. 3.138. Розфарбування графа за евристичним алгоритмом

Розглянемо кроки евристичного алгоритму детально:

***Крок* 1**. Сортувати вершини графа за степенями зменшення:

$\deg(x_i) \geq \deg(x_j)$, для будь-яких $x_i, x_j \in G$

Встановити поточний колір p =1 та i =1.

***Крок* 2**. Вибрати чергову нерозфарбовану вершину зі списку і призначити її новий колір $col(x_i) = p$, $X = \{x_i\}$.

***Крок* 3**. i =i +1. Вибрати чергову не розфарбовану вершину x_i і перевірити умову суміжності: $x_i \cap \Gamma(X) = \varnothing$, де X – множина вершин, уже розфарбованих у колір p. Якщо вершина x_i не є суміжною з даними вершинами, то також присвоїти їй колір p: $col[x_i] = p$.

***Крок* 4**. Повторювати крок 3 до досягнення кінця списку (i == n).

***Крок* 5**. *Якщо* всі вершини графа розфарбовані, *то* – кінець алгоритму; *інакше*: p = p +1;

i =1. Повторити крок 2.

Для роботи алгоритму можна використовувати довільну структуру даних, яка однозначно задає граф.

Розглянемо приклад програми реалізації евристичного алгоритму зі структурою графа, яка задана матрицею суміжності a.

3.11.5. Програмний код евристичного алгоритму

Як і у попередньому випадку, даний алгоритм передбачає початкову генерацію симетричної матриці суміжності та списку кольорів.

```
# Код евристичного алгоритму
from random import *
n = 5 # максимальна кількість вершин графа
# Початковий колір
curcol=1
# Список кольорів вершин
colarr=[0 for i in range(n)]
# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
# Формування списку за степенями
```

```
def degforming():
    def getkey(item):
        return item[0]
    # Список кортежей (степінь, номер вершини)
    degarr=[[0 for i in range(2)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            degarr[i][0] += a[i][j]
            degarr[i][1] = i
    # Сортуємо кортежі за спаданням степенів
    degarr.sort(key=getkey, reverse=True)
    return degarr
# Розфарбовка вершин
def dyer(curcol,node):
    for k in range(n):
        if a[node][k]==0:
            if colarr[k]==0:colarr[k]= curcol
# Основний код
sortarr=degforming()
for i in range(n):
    if not colarr[sortarr[i][1]]:
        colarr[sortarr[i][1]]=curcol
        dyer(curcol,sortarr[i][1])
        curcol+=1
            for r in range(n): print(a[r])
                print(sortarr)
                print(colarr)
```

3.11.6. Приклад евристичного алгоритму розфарбування

Розфарбуємо граф *G*, зображений на рис. 3.139.

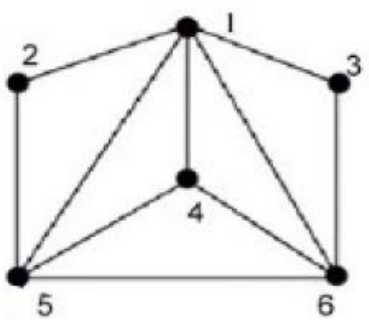


Рис.3.139. Граф для розфарбування евристичним алгоритмом

Проміжні дані для вирішення завдання будемо записувати в таблицю 3.4.

Таблиця 3.4

Таблиця розфарбування

Номери вершин SortArr	1	5	6	4	2	3
Степені вершин DegArr	5	4	4	3	2	2
CurCol = 1	1	-	-	-	-	-
CurCol = 2	1	2	-	-	-	2
CurCol = 3	1	2	3	-	3	2
CurCol = 4	1	2	3	4	3	2

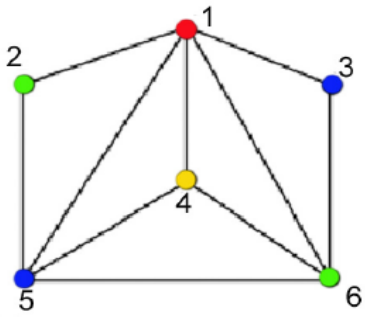
Крок 1. Першою SortArr стоїть вершина 1, яку фарбуємо червоним кольором 1. Несуміжних з 1 немає.

Крок 2. Другою в SortArr стоїть вершина 5, яку фарбуємо синім кольором 2. Несуміжна з 5 вершина 3, яку процедура dyer(curcol,5) фарбує також синім кольором 2.

Крок 3. Третьою в SortArr є вершина 6, яку фарбуємо зеленим кольором 3. Несуміжна з 6 вершина 2, яку процедура dyer(curcol,6) фарбує також зеленим кольором 3.

Крок 4. Четвертою в SortArr є вершина 4, яку фарбуємо жовтим кольором 4. Всі несуміжні вершини з 4 вже розфарбовані.

$$A = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 0 & 1 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 & 1 \\ 6 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$



3.11.7. Рекурсивна процедура послідовного розфарбування

1. Фіксуємо порядок обходу вершин.
2. Ідемо по суміжних вершинах, використовуючи такий найменший колір, який не створить конфліктів.
3. Якщо на черговому кроці колір вибрати не вдалось, то «відкочуємось» до попередньої вершини й вибираємо для неї наступний колір, який не створить конфліктів. У процедурі використовується рекурсивний виклик процедури фарбування наступної вершини у випадку успішного фарбування попередньої вершини.

Код алгоритму має вигляд:

```
#Код рекурсивного алгоритму
from random import *
```

```
# Максимально допустима кількість кольорів
cmax=10
n = 5 # максимальна кількість вершин графа
#Список кольорів вершин
color=[0 for i in range(n)]
# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
def visit (i):
# Функція вибору фарби для розфарбування вершини
з номером i
    def nicecolor():
        w = {0}
        newcol=0
        for j in range(n):
          if a[i][j] > 0: w.add(color[j])
for cm in range(1,cmax):
            if cm not in w:
                newcol=cm
                break
return newcol
# код функції visit
  if i == n:
#Якщо всі вершини розфарбовані, те виводимо результат
        print("FINAL")
    else:
#Якщо поточна вершина не розфарбована
          if color[i]==0:
           curcol=nicecolor()
           if curcol >0:
#Якщо неконфліктний, то розфарб. вершину і фарбою c
                    color [i] = curcol
#Рекурсивно викликаємо для наступної вершини
                    visit (i + 1)
#Основний код програми
```

```
visit (0)
for r in range(n):
    print(a[r])
print()
print(color)
```

3.11.8. Приклад роботи рекурсивної процедури

Розглянемо граф G та застосуємо рекурсивну процедуру для його розфарбування (рис. 3.140).

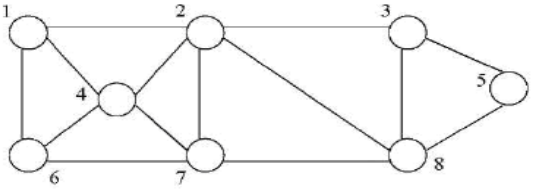


Рис. 3.140. Граф G для рекурсивного розфарбування

Матриця суміжності має такий вигляд

$$A = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 4 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 6 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 7 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 8 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Робота алгоритму зведена в таблицю 3.5.

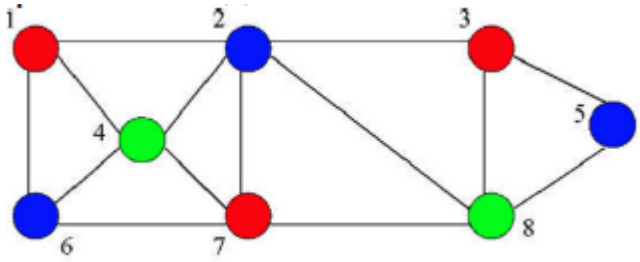
Перший стовпець містить виклики процедури Visit, а решта стовпців показує, яка фарба була прийнята, а яка відхилена.

Таблиця 3.6

Послідовність розфарбувань рекурсивним алгоритмом

	Червоний	Синій	Зелений
Visit(1)	+		
Visit(2)	-	+	
Visit(3)	+		
Visit(4)	-	-	+
Visit(5)	-	+	
Visit(6)	-	+	
Visit(7)	+		
Visit(8)	-	-	+

Розфарбований граф має вигляд:



3.11.9. «Жадібний» алгоритм розфарбування

Нехай дано зв'язний граф $G(V, E)$.

1. Задамо множину *monochrom* $=\varnothing$, куди будемо записувати всі вершини, які можна пофарбувати одним кольором.

2. Переглядаємо всі вершини й виконуємо наступний «жадібний» алгоритм:

Procedure Greedy

For (для кожної незафарбованої вершини $v \in V$) **do**

If *v не суміжна з вершинами з monochrom* **then**

begin

color (*v*) =колір*;*

monochrom =monochrom $\cup \{v\}$

end.

Розглянемо детальніше програмну реалізацію даного алгоритму за умови, що для представлення графа використовують матрицю суміжності.

```
# код жадібного алгоритму
from random import *
n=5
allcolored = False #Ознака того, що всі вершини не розфарбовані
color=0
#Список кольорів вершин
colarr=[0 for i in range(n)]
# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
#Функція вибору фарби для розфарбування вершини з номером i
def avid (i,color):
    def check (i): #Перевірка кольору суміжних вершин
        ch = True
     for j in range(n):
```

```
#Якщо вершина j суміжна з тією, що підлягає перевірці
                    if a [i][j] == 1:
                      if (j in w): ch = False
            return ch
    w = set() # Очищаємо множину одноколірних вершин
#Розфарбовуємо першу вершину новою фарбою
     colarr [i]=color
#Доповнюємо множину одноколірних вершин вершиною
     w.add(i)
     #Перевіряємо інші вершини на можливість розфарбування цією фарбою
     for k in range(n):
          if colarr[k] == 0:
             if check(k):
                 colarr [k] = color
                 w.add(k)
# Головний код
while not allcolored: #Цикл по вершинах графа
      allcolored = True
      for i in range(n):
           if colarr [i] == 0: #Знайшли не розфарбовану вершину
              color += 1 # Встановлюємо новий колір
              allcolored = False
              avid (i,color) #процедура жадібного розфарбування
for r in range(n): # 6 рядків
print(a[r])
print()
print(colarr)
```

Можливі результати роботи «жадібного» алгоритму розфарбування

Розглянемо варіанти розфарбування графа, показаного на рис. 3.141. Нехай зафарбуємо вершину 1 у синій колір, а потім, пропустивши вершину 2, зафарбуємо в синій колір вершини 3 і 4. Тоді можна одержати 2 фарби.

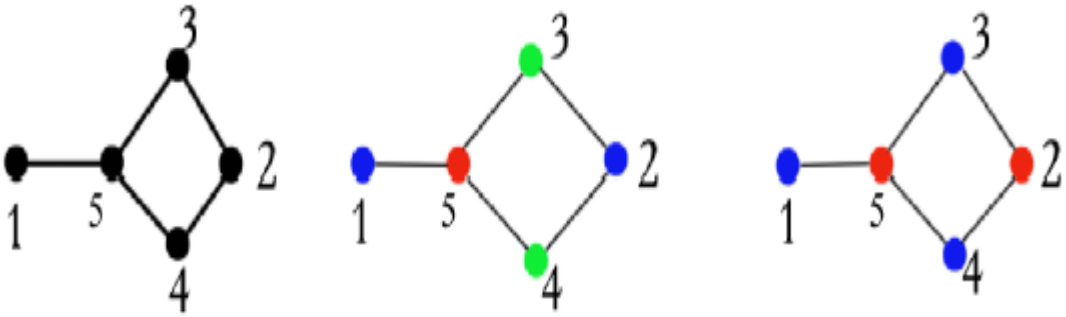


Рис. 3.141. Варіанти розфарбування графа

Але «жадібний» алгоритм, ґрунтуючись на нумерації вершин, зафарбує в синій колір вершини 1 і 2, для розфарбування графа тепер потрібно 3 фарби.

3.11.10. Приклад роботи «жадібного» алгоритму розфарбування

Розглянемо граф G (рис. 3.142) та застосуємо до нього «жадібний» алгоритм розфарбування.

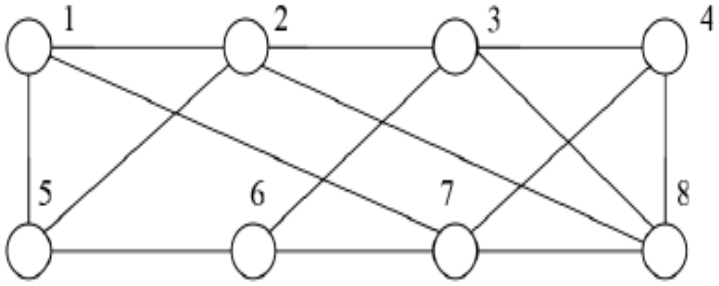


Рис. 3.142. Граф для розфарбування жадібним алгоритмом

Матриця суміжності A має такий вигляд:

$$A = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 5 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 7 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 8 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Спочатку розфарбування вибираємо вершину з номером 1 та розфарбовуємо її в колір 1 (червоний). Далі відбувається пошук несуміжної вершини з вершиною 1. Якщо така вершина знайдена, то вона також розфарбовується в колір 1 (червоний). Наступна знайдена для розфарбування кольором 1 вершина повинна бути не суміжною з двома попередніми. Процес продовжується до того часу, поки всі можливості розфарбувати вершини кольором 1 будуть вичерпані. Після цього вибираємо фарбу кольору 2 (синя) і розфарбовуємо нею вершину з мінімальним номером, яка є не розфарбованою до цього часу. Наступна придатна для розфарбування фарбою 2 вершина повинна бути не суміжною з вершиною, яка була розфарбована кольором 2 (синій) на попередньому кроці. Процес розфарбування фарбою 2 також продовжується до того часу, поки не будуть вичерпані всі можливості розфарбування вершин цією фарбою.

Перед вибором чергової фарби для розфарбування завжди перевіряємо, чи залишилися ще не розфарбовані вершини. Якщо такі вершини знайдено, то

вибираємо чергову фарбу і продовжуємо процес розфарбування. Якщо ж всі вершини графа розфарбовано, то процес розфарбування жадібним алгоритмом закінчується. Результат розфарбування «жадібним» алгоритмом графа *G* показано на рисунку 3.143.

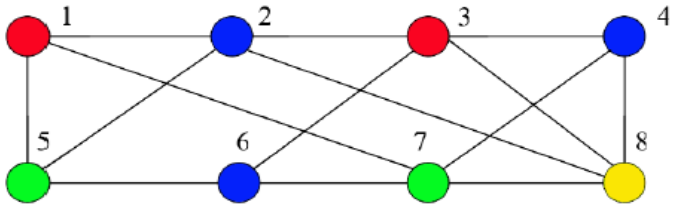


Рис. 3.143.Результат розфарбування графа *G* «жадібним» алгоритмом

Обов'язкові завдання

1. Які переваги і недоліки алгоритму прямого неявного перебору?
2. На яких базових принципах ґрунтується евристичний алгоритм розфарбування?
3. Опишіть у вигляді псевдокоду принцип роботи «жадібного» алгоритму розфарбування графа.
4. Які переваги і недоліки рекурсивного алгоритму розфарбування графів?

3.12. Шляхи та цикли Ейлера. Плоскі та планарні графи

Визначення. Нехай$G = (V, E)$ – граф. Цикл, який включає всі ребра і вершини графа *G*, називають *циклом Ейлера.* Якщо ця умова виконується, говорять, що граф *G* має цикл Ейлера.

Якщо тепер повернутися до задачі про кенігсберзькі мости, легко переконатися, що вона зводиться до спроби визначити, чи містить граф, що ілюструє задачу, цикл Ейлера. Для цього нам буде потрібна наведена нижче теорема. Ця теорема справедлива також для мультиграфів. Більше того, доведення теореми залишається тим же. Для ясності будемо використовувати термін граф, розуміючи, що кожне твердження справедливе для мультиграфів.

Теорема 3.12.1. Граф з більше, ніж однією вершиною, має цикл Ейлера тоді і тільки тоді, коли він зв'язний і кожна його вершина має парний степінь.

Приклад 3.48. Наведений на рис. 3.144 граф має цикл Ейлера, оскільки степінь кожної його вершини парна.

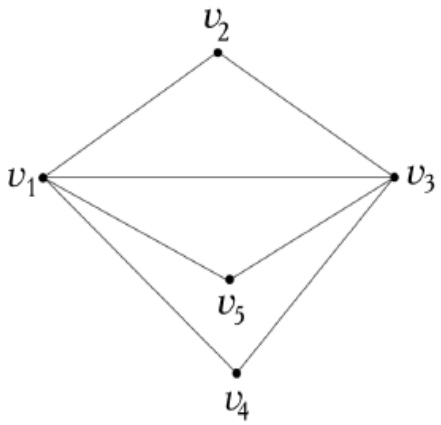


Рис.3.144. Граф з циклом Ейлера

Приклади циклів Ейлера:

$(v_1, v_2, v_3, v_4.v_1, v_5, v_3, v_1)$, $(v_4, v_3, v_2, v_1, v_3, v_5, v_1, v_4)$ і т. д.

Повертаючись до задачі про кенігсберзькі мости, виявляємо, що мультиграф, побудований Ейлером для опису кенігсберзьких мостів, має непарні степені всіх його вершин.

Отже, цей мультиграф не має циклу Ейлера, тому неможливо пройти кожний міст по одному разу і повернутися у початкову точку шляху у графі, показаному на рис. 3.145.

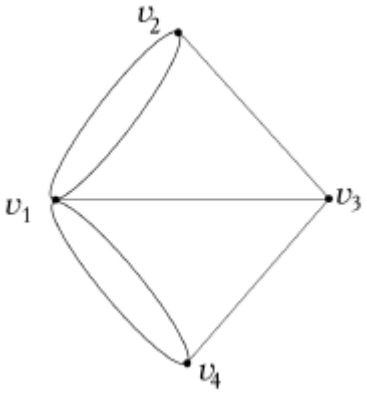


Рис. 3.145. Граф кенігсберзьких мостів

Як видно з рисунка, задача про кенігсберзькі мости була розв’язана з використанням мультиграфа. Однак цю задачу можна розв’язати, використовуючи простий граф. Для цього початковий мультиграф необхідно привести до простого графа шляхом введення додаткових вершин (рис. 3.146):

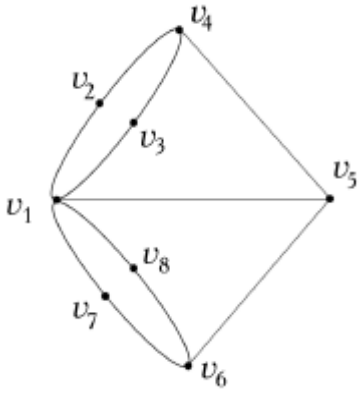


Рис. 3.146. Простий граф кенігсберзьких мостів

Якщо поставити задачу не повертатися у початкову точку, то попередня задача зводиться до пошуку шляху в графі.

Визначення. Нехай G (V, E) – граф. Шлях, який включає кожне ребро графа G тільки один раз, називають *шляхом Ейлера*. У цьому випадку говорять, що граф G має шлях Ейлера.

У випадку, коли шлях Ейлера не є циклом Ейлера, його називають власним шляхом Ейлера.

Теорема 3.12.2. Граф або мультиграф має власний шлях Ейлера тоді і тільки тоді, коли він зв'язний і рівно дві його вершини мають непарний степінь.

Оскільки граф для кенігсберзьких мостів має чотири вершини з непарними степенями, можна зробити висновок про неможливість пройти кожний міст по одному разу, навіть якщо не потрібно повертатися у початкову точку маршруту.

Приклад 3.49. На рис. 3.147 показано граф, який має шлях Ейлера, оскільки дві його вершини мають непарний степінь.

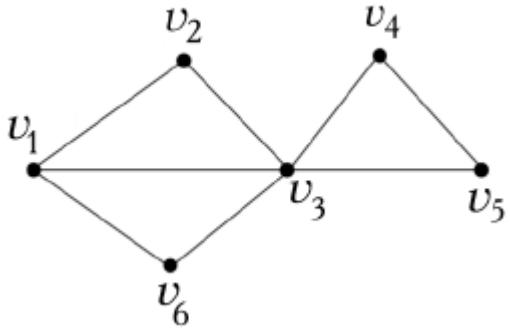


Рис. 3.147. Граф з двома непарними степенями

Приклад шляху Ейлера: $v_1, v_2, v_3, v_1, v_6, v_3, v_4, v_5, v_3$.

Цикл Ейлера в орієнтованому графі

Визначення. Нехай G (V, E) – орієнтований граф. **Орієнтованим циклом** називають орієнтований шлях ненульової довжини з вершини в ту ж вершину без повторення ребер.

Визначення. Нехай G (V, E) – орієнтований граф. Орієнтований цикл, який включає всі ребра і вершини графа G, називають **циклом Ейлера**. У цьому випадку говорять, що орієнтований граф G має цикл Ейлера.

Теорема 3.12.3. Орієнтований граф має цикл Ейлера тоді і тільки тоді, коли він зв'язний і напівстепінь входу кожної вершини дорівнює її напівстепені виходу.

Алгоритм побудови циклу Ейлера

1. Виходимо з довільно обраної вершини v і кожне пройдене ребро закреслюємо.
2. Ніколи не йдемо по тому ребру e, яке в даний момент є мостом (тобто при видаленні якого граф, утворений незакресленими ребрами, розпадається на два компонента зв'язності, що мають хоча б по одному ребру).
3. Не вибираємо ребро, що веде в v, поки є інші можливості.

3.13. Цикли Гамільтона (основні визначення)

Визначення. Гамільтоновим ланцюгом графа називають його простий ланцюг, який проходить через кожну вершину графа точно один раз.

Визначення. Цикл графа, що проходить через кожну його вершину, називають **гамільтоновим циклом**.

Визначення. Граф називають **гамільтоновим**, якщо він має гамільтонів цикл.

Визначення. Граф, який містить простий шлях, що проходить через кожну його вершину, називають **напівгамільтоновим**.

Це визначення можна поширити на орієнтовані графи, якщо шлях вважати орієнтованим. Гамільтонів цикл не обов'язково містить усі ребра графа. Ясно, що гамільтоновим може бути тільки зв'язний граф і що будь-який гамільтонів граф є напівгамільтоновим.

Помітимо, що гамільтонів цикл існує далеко не в кожному графі.

Зауваження: Будь-який граф G можна перетворити в гамільтонів граф, додавши достатню кількість вершин. Для цього, наприклад, достатньо до вершин $v_1,...,v_p$ графа G додати вершини $u_1,...,u_p$ і множини ребер $\{v_i, u_i\}$ та $\{u_i, v_{i+1}\}$.

Степенем вершини v називають число ребер $\deg(v)$, інцидентних їй, при цьому петля не враховується. У випадку орієнтованого графа розрізняють степінь $\deg^+(v)$ по вихідних дугах і $\deg^-(v)$ – по вхідних.

Теорема 3.13.1. Нехай G має $p \geq 3$ вершин. Якщо для будь-якого n, $1 \leq n \leq \frac{p-1}{2}$, кількість вершин зі степенями, що не перевищують n, менше ніж n,

і для непарного p кількість вершин степеня $\frac{p-1}{2}$ не перевищує $\frac{p-1}{2}$, то G – гамільтонів граф.

Наслідок 1. Якщо $p \geq 3$ і $\deg(u) + \deg(v) \geq p$ для будь-якої пари u і v несуміжних вершин графа G, то G – гамільтонів граф.

Наслідок 2. Якщо $p > 3$ і $\deg(v) \geq \frac{p}{2}$ для будь-якої вершини v графа G, то G – гамільтонів граф.

Теорема 3.13.2. У повному графі $G(V,)$ завжди існує гамільтонів шлях.

3.14. Плоскі та планарні графи. Загальні поняття про плоский граф

У визначенні графа як геометричної фігури до цього ми не накладали жодних обмежень на розташування фігури в просторі. Тепер же будемо говорити, що граф зображений на поверхні (площини, сфері, і т. п.), якщо всі його **вершини і ребра належать цій поверхні**.

Визначення. Граф, зображений на площині, називають **плоским**, якщо його ребра не перетинаються в точках, відмінних від вершин графа.

Відзначимо, що властивість графа **бути** *або* **не бути плоским –** це властивість геометричного зображення, а не алгебраїчного об'єкта.

На рис. 3.148 показані графи G_1 і G_2, які є ізоморфними, але G_1 – плоский, а G_2 – неплоский.

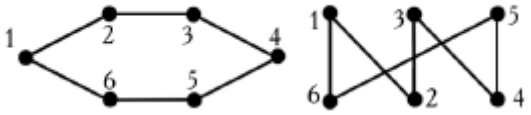


Рис. 3.148. Ізоморфні графи

Загальні поняття про планарний граф

Таким чином, термін «плоский граф» завжди відноситься до **конкретного** (одного з багатьох) геометричного зображення графа.

Той самий граф (як множина вершин + множина ребер) може мати **як плоскі, так і не плоскі зображення**.

У той же час, принципове питання, на яке потрібно відповідати при розв'язуванні задач типу прокладки комунікацій: «Чи має даний граф хоча б одне плоске зображення?». Визначимо клас графів, для яких відповідь на це питання позитивна.

Визначення. Граф називають **планарним**, якщо він ізоморфний плоскому графу. Іншими словами. **Планарним графом** називають граф, який може бути зображений на площині так, що його ребра не перетинаються.

Укладання графа на поверхні

Про планарні графи говорять, що вони мають **плоске укладання**, або що вони **укладаються на площині**.

Можна визначити укладання графів не тільки на площині, але і на **інших поверхнях** у просторі.

Властивість графа укладатися на поверхні безумовно **залежить від виду цієї поверхні**.

Однак багато поверхонь із огляду на укладання графів нічим **не відрізняються від площини**.

Жорданова крива

Жордановою кривою на площині називають неперервну криву лінію без самоперетинань.

Замкненою жордановою кривою називають жорданову криву, початок і кінець якої збігаються.

Теорема Жордана. Якщо S – замкнена жорданова крива на площині, а x та y – дві різні точки цієї кривої, то будь-яка жорданова крива, що з'єднує точки x і y, або повністю лежать всередині S, крім точок x і y, або поза кривою S, окрім точок x і y, або перетинає криву S у деякій точці, відмінній від точок x і y.

На рис. 3.149 показано три варіанти взаємного розташування жорданових кривих.

Рис. 3.149. Жорданові криві

Будемо говорити, що граф G укладається в просторі L, якщо існує взаємно однозначне відображення вершин графа G у точки і ребер у жорданові криві цього простору таке, що криві, які відповідають різним ребрам, перетинаються в інцидентних даним ребрам вершинах. Зображений у такий спосіб граф G у просторі L називають **укладанням графа** G.

Теорема про укладання графа в тривимірному просторі

Теорема 3.14.1. Будь-який граф укладається в тривимірному просторі.

Доведення. Розмістимо всі вершини графа $G = (V, E)$ на осі OX. З пучка площин, що проходять через цю вісь, виберемо $|E|$ різних площин. Далі кожне ребро $(u, v) \in E$ зобразимо у відповідній площині півколом, що проходить через вершини u і v, як показано на рис. 3.150. Ясно, що в результаті одержимо укладання графа G у тривимірний простір, оскільки всі ребра лежать у різних площинах, і тому не перетинаються ні в яких точках, крім вершин.

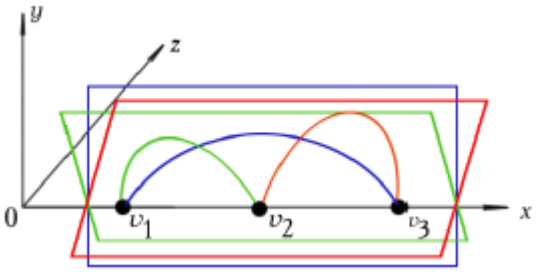


Рис. 3.150. Граф з ребрами у різних площинах

3.15. Непланарні графи

Теорема 3.15.1. Графи K_5 (рис. 3.151) і $K_{3,3}$ не є планарними.

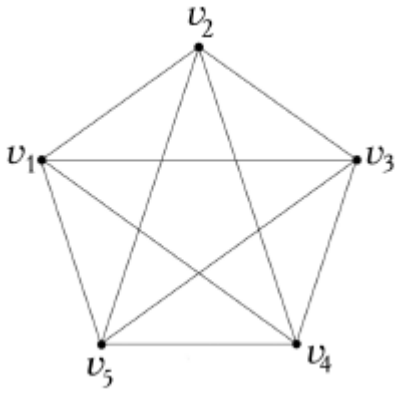


Рис. 3.151. Граф K_5

Доведення. Припустимо протилежне, що граф K_5 планарний. Оскільки він має цикл довжини 5, наприклад, $(v_1, v_2, v_3, v_4, v_5)$, то можна вважати, що при будь-якому укладанні цього графа на площину цей цикл зображується правильним п'ятикутником. За теоремою Жордана ребро (v_1, v_3) має лежати або цілком усередині цього п'ятикутника, або цілком поза ним. Третю можливість (коли ребро має загальну точку з п'ятикутником) ми не розглядаємо, оскільки маємо справу з укладанням на площину.

1. Розглянемо спочатку випадок, коли (v_1, v_3) лежить усередині п'ятикутника. Оскільки ребра (v_2, v_4)і (v_2, v_5) не мають перетинати ребро (v_1, v_3), то обидва ці ребра лежать поза п'ятикутником. Ця ситуація зображена на рис. 3.152.

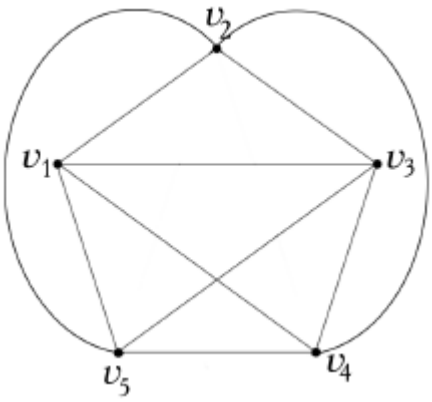


Рис. 3.152. Спроба уникнути перетину з ребром (v_1, v_3)

Ребро (v_1, v_4) не має перетинати ребро (v_2, v_5) і тому воно має лежати усередині п'ятикутника. Аналогічно ребро (v_3, v_5) має лежати усередині п'ятикутника, оскільки не має перетинатися з ребром (v_2, v_4). Однак ребра (v_1, v_4) і (v_3, v_5) обов'язково перетинаються, тому, згідно з теоремою Жордана, одне з них має лежати поза п'ятикутником.

Таким чином, ми приходимо до протиріччя з нашим припущенням. Наявність даного протиріччя доводить помилковість початкового припущення, тобто граф K_5 не є планарним.

Аналогічно доводять, що граф $K_{3,3}$ не є планарним, оскільки неможливо побудувати ребро (v_1, v_4) без перетину з іншим ребром графа $K_{3,3}$ (рис. 3.153).

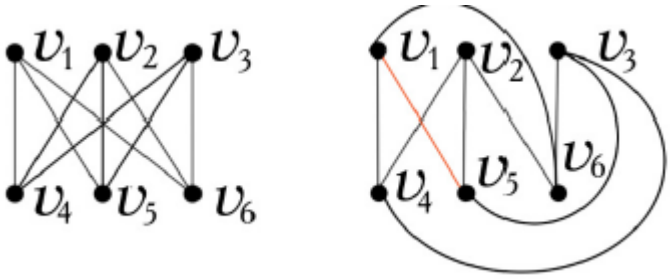


Рис. 3.152. Доведення того, що граф $K_{3,3}$ не є планарним

3.15.1. Грані плоского графа

Точку *x* площини *S* , на якій розміщено граф *G* , називають **диз'юнктною** з *G* , якщо ця точка не відповідає жодній вершині графа *G* і не лежить на жодному ребрі цього графа. Дві точки *x* і *y* площини *S* називають еквівалентними, якщо вони диз'юнктні з *G* і їх можна з'єднати такою жордановою кривою, усі точки якої диз'юнктні з *G* .

Введене відношення еквівалентності точок площини розбиває множину всіх точок площини *S* на класи еквівалентності, які називають гранями графа *G* .

Наприклад, граф G, зображений на рис. 3.153, має 4 грані: f_1, f_2, f_3, f_4. Причому грань f_1 – нескінченна.

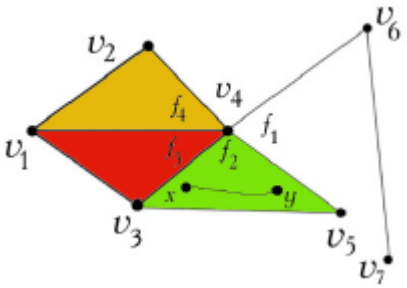


Рис. 3.153. Грані графа G

3.15.2. Теорема Ейлера

Теорема 3.15. 1. Нехай G – зв'язний плоский граф, у якому n – число вершин,
m – число ребер і f – число граней. Тоді справедливе співвідношення: $n + f = m + 2$.

Доведення. План доведення ґрунтується на виконанні індукції за числом ребер у графі G.

1. Нехай $m = 0$, то $n = 1$ (оскільки за умовою теореми G – зв'язний) і $f = 1$ (нескінченна грань). У цьому випадку теорема вірна, оскільки $n + f = 1+1 = 2$ і $m + 2 = 0 + 2 = 2$. Приклад даного графа показаний на рис. 3.154.

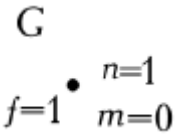


Рис. 3.154. Зв'язний граф G з параметрами $n = 1$, $m = 0$, $f = 1$

2. Нехай граф представлений такими параметрами:

Ребро e_1 є петлею, і в цьому випадку число граней збільшується на одну, а число вершин залишається незмінним. Граф з однією вершиною і двома гранями показаний на рис. 3.155.

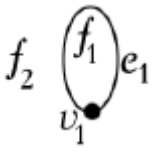


Рис. 3.155. Зв'язний граф G з параметрами $n = 1$, $m = 1$, $f = 2$

Для розглянутого графа $G = (V, E)$ одержуємо $V = \{v_1\}$, $E = \{e_1\}$, $F = \{f_1, f_2\}$.
Тому $n = |V| = 1$, $m = |E| = 1$, $f = |F| = 2$. Звідси $n + f = 1+ 2 = 3$ і $m + 2 = 1+ 2 = 3$.

3. Нехай граф G представлений такими параметрами:
ребро e_1 з'єднує дві різні вершини в G. Граф з двома вершинами і одним ребром показаний на рис. 3.156.

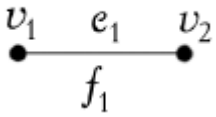


Рис. 3.156. Зв'язний граф G з параметрами $n = 2$, $m = 1$, $f = 1$

У даному графі $G = (V, E)$ одержуємо $V = \{v_1, v_2\}$, $E = \{e_1\}$, $F = \{f_1\}$.

Тому $n = |V| = 1$, $m = |E| = 1$, $f = |F| = 1$. Звідси $n + f = 2 + 1 = 3$ і $m + 2 = 1 + 2 = 3$.

4. Нехай зв'язний граф G містить кілька вершин і ребер. Граф з трьома вершинами і трьома ребрами показаний на рис. 3.157.

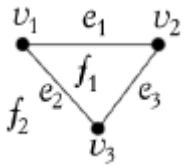


Рис. 3.157. Зв'язний граф G з параметрами $n = 3, m = 3, f = 2$

Граф $G = (V, E)$ характеризується параметрами $V = \{v_1.v_2, v_3\}$, $E = \{e_1, e_2, e_3\}$, $F = \{f_1, f_2\}$.

Тому $n = |V| = 3$, $m = |E| = 3$, $f = |F| = 2$. Звідси $n + f = 3 + 2 = 5$ і $m + 2 = 3 + 2 = 5$.

5. До графа, показаного на рис. 3.157, додамо петлю. У результаті одержимо граф, показаний на рис. 3.158.

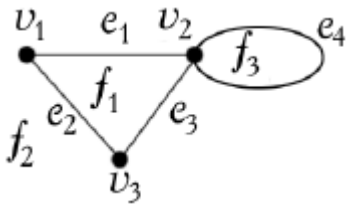


Рис. 3.158. Зв'язний граф G з параметрами $n = 3, m = 3, f = 3$

Граф $G = (V, E)$ характеризується параметрами $V = \{v_1.v_2, v_3\}$, $E = \{e_1, e_2, e_3, e_4\}$, $F = \{f_1, f_2, f_3\}$. Тому $n = |V| = 3$, $m = |E| = 4$, $f = |F| = 3$. Отже, $n + f = 3 + 3 = 6$ і $m + 2 = 4 + 2 = 6$.

6. Нехай зв'язний граф G_1 містить 4 вершини і 4 ребра (рис. 3.159 а).

Даний граф має такі параметри:

$$V = \{v_1.v_2, v_3, v_4\}, E = \{e_1, e_2, e_3, e_4\}, F = \{f_1, f_2\}.$$

Побудуємо граф G_2 (рис. 3.159 б), додавши ребро, що з'єднує дві його вершини.

$$V = \{v_1.v_2, v_3, v_4\}, E = \{e_1, e_2, e_3, e_4, e_5\}, F = \{f_1, f_2, f_3\}.$$

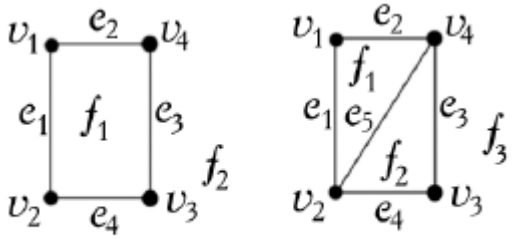


Рис. 3.159. Граф G_1 (а) і G_2 (б)

Параметри нового графа $n = 4$, $m = 5$, $f = 3$.

Отже, $n + f = 4 + 3 = 7$, $m + 2 = 5 + 2 = 7$.

7. Якщо до зв'язного графа G_2 (рис. 3.159 б), який має такі параметри $V = \{v_1.v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3, e_4, e_5\}$, $F = \{f_1, f_2, f_3\}$ додати ребро, інцидентне тільки з однією з його вершин, то одержимо граф з параметрами (рис. 3.160):

$$V = \{v_1.v_2, v_3, v_4, v_5\}, E = \{e_1, e_2, e_3, e_4, e_5, e_6\}, F = \{f_1, f_2, f_3\}$$

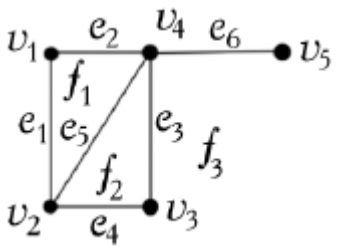


Рис. 3.160. Зв'язний граф G з параметрами $n = 5$, $m = 6$, $f = 3$

Одержуємо $n + f = 5 + 3 = 8$, $m + 2 = 6 + 2 = 8$.

Наслідок. Нехай – плоский граф з n вершинами, m ребрами, f гранями і k компонентами зв'язності; тоді $n + f = m + k + 1$.

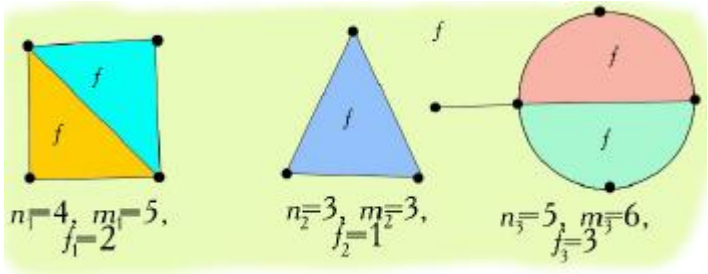


Рис. 3.161. Незв'язний граф G з параметрами $n = 12$, $m = 14$, $f = 6$

$$n = n_1 + n_2 + n_3 = 4 + 3 + 5 = 12$$

$$m = m_1 + m_2 + m_3 = 5 + 3 + 6 = 14$$

$$f = f_1 + f_2 + f_3 = 2 + 1 + 3 = 6$$

$$n + f = 12 + 6 = 18$$

$$m + k + 1 = 14 + 3 + 1 = 18$$

Наслідок. Якщо G – зв'язний планарний граф (рис. 3.162) з m ребрами і $n \geq 3$ вершинами, то $m \leq 3n - 6$.

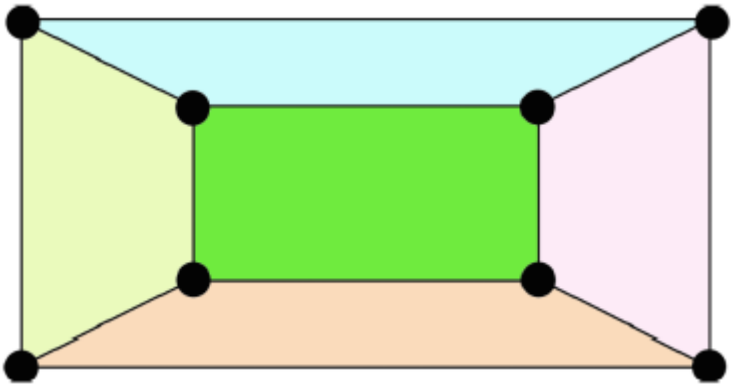


Рис. 3.162. Зв’язний граф з параметрами $n = 8$, $m = 12$

$n = 8 > 3$, $m = 12$

$m < 3n - 6$, $m < 3 \times 8 - 6$, $m < 24 - 6, 12 < 18$

Наслідок. У будь-якому планарному графі існує вершина, степінь якої не більше п’яти (рис. 3.163).

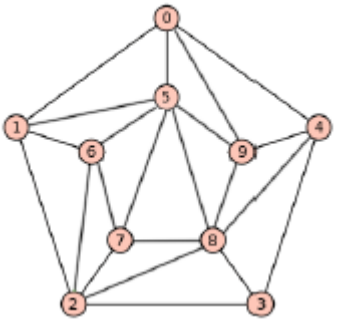


Рис. 3.163. Планарний граф

Твердження. Будь-який підграф планарного графа теж планарний.

Твердження. Якщо граф включає непланарний підграф, то і сам граф непланарний.

Наслідок. Будь-який підграф, що включає граф K_5 або $K_{3,3}$, не є планарним.

Виявляється, що графи K_5 (рис. 3.164) і $K_{3,3}$ (рис. 3.165) – **єдині непланарні** графи в тому розумінні, що будь-який непланарний граф включає один з них як підграф.

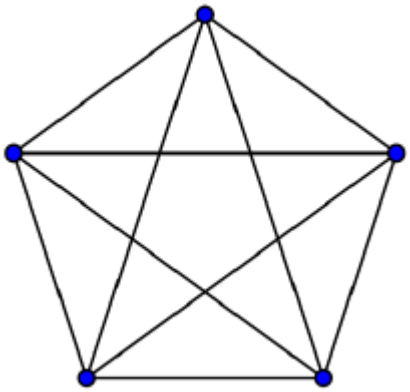


Рис. 3.164. Зв’язний граф K_5

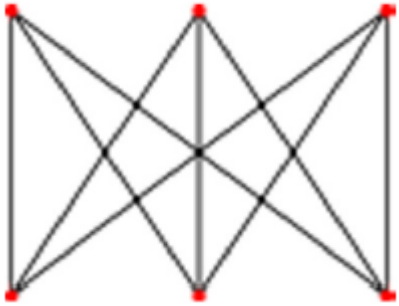


Рис. 3.165. Зв’язний граф $K_{3,3}$

Для формулювання цього результату введемо поняття **гомеоморфності графів.**

3.15. 3. Гомеоморфні графи

Визначення 1. Два графа називають **гомеоморфними** або **тотожними** з точністю до вершин степеня 2, якщо вони можуть бути отримані з того самого графа за допомогою операції введення вершини степеня 2 у ребро. Наприклад, графи, зображені на рис. 3.166, гомеоморфні.

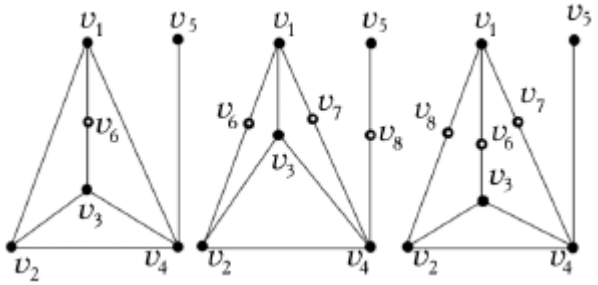


Рис. 3.166. Приклад гомеоморфних графів

Неважко переконатися, що гомеоморфізм є відношенням еквівалентності.

Визначення 2. Два графи називають гомеоморфними, якщо кожен з них може бути одержаний розбиттям певного графа.

Визначення 3. Два графа G_1 і G_2 є гомеоморфними, якщо можливо сформувати таке розбиття графа G_1 і розбиття графа G_2, які ізоморфні між собою.

Визначення 4. Розбиття графа – це граф, який отримуємо послідовним застосуванням до даного графа операції розбиття ребер. При розбитті, ребро e, яке інцидентне вершинам $\{u,v\}$, замінюємо на вершину w і два ребра $\{u,w\}$ і $\{w,v\}$ (рис. 3.167).

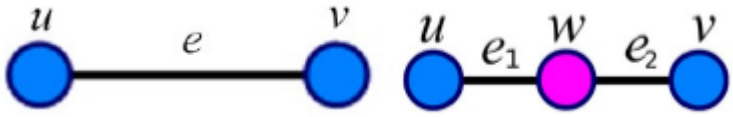


Рис. 3.167. Розбиття ребра e

3.15.4. Теорема Понтрягіна – Куратовського

Граф є планарним тоді і тільки тоді, коли він не має підграфів, гомеоморфних K_5 і $K_{3,3}$.

Доведення. З геометричної точки зору додавання вершини степеня 2 – це додавання точки на ребрі, а стирання такої вершини поєднує два ребра із спільним кінцем в одне. Очевидно, що кожна із цих операцій, застосована до плоского графа, знову дасть плоский граф. Отже, за наслідками з теореми Ейлера, жодний плоский (а, отже, і планарний) граф не гомеоморфний графам K_5 і $K_{3,3}$. З урахуванням зауваження про непланарні підграфи, теорема доведена.

3.15.5. Операція стягування

Нехай (*u*, *v*) – ребро графа *G*. Вилучимо із графа *G* вершини *u i v*. Після цього додамо в граф *G* нову вершину *w* і з'єднаємо її ребрами з усіма вершинами, з якими була суміжна хоча б одна з вершин *u* і *v* (рис. 3.168).

Рис. 3.168. Стягування графа G_1 до графа G_2

Позначимо отриманий граф через G_2. Говорять, що граф G_2 отримано з G_1 **стягуванням ребра** (*u*, *v*).

Якщо скінченним (можливо нульовим) числом операцій стягування ребра із

графа G_1 можна одержати граф G_2, то говорять, що G_1 **стягується** до G_2.

3.15.6. Теорема Вагнера

Теорема. Граф планарний тоді і тільки тоді, коли він не має підграфів, які стягуються графом K_5 і $K_{3,3}$.

Доведення необхідності. Якщо стягти ребро в плоскому графі, він залишиться плоским. Отже, за наслідками з теореми Ейлера, жоден плоский (а отже, і планарний) граф не стягується ні до графа K_5, ні до графа $K_{3,3}$. З урахуванням зауваження про непланарні підграфи, необхідність доведена.

Процес стягування до $K_{3,3}$ і K_5 (рис. 3.169).

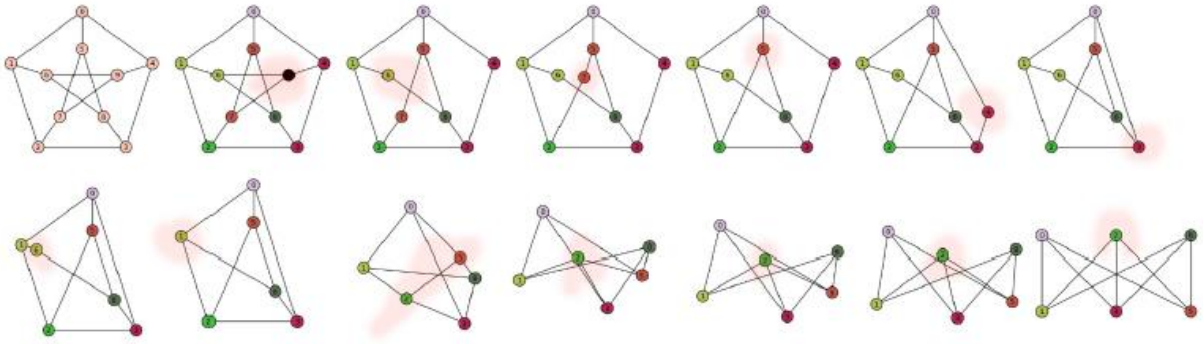


Рис. 3.169. Процес стягування

Обов'язкові завдання

1. Коли існує шлях Ейлера в графі? Поясніть, чому граф для кенігсберзьких мостів не має шляху Ейлера.

2. Дайте визначення гамільтонового графа. Чи існує можливість перетворення негамільтонового графа в гамільтоновий?

3. Чи є повний граф з множиною вершин $V = \{v_1, v_2, v_3, v_4, v_5\}$ планарним графом? Якщо так, то виконайте його укладку на площині.

4. Перевірте, чи є планарним граф, який задано такою матрицею суміжності:

$$S = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Якщо даний граф планарний, то виконайте його укладку на площину.

5. Визначте число граней графа, заданого матрицею суміжності:

$$S = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

6. Нехай дано граф, який містить 10 ребер і 6 граней. Визначте кількість вершин даного графа.

4. Математична логіка

4.1. Алгебра висловлювань

4.1.1. Загальні поняття

Логіка – це наука про закони мислення та його форми. Вона як мистецтво суджень бере свій початок з далекої давнини. В логіку було впроваджено математичну символіку, і сьогодні вона використовує мову й методи математики. Звідси й назва – математична логіка.

Основи математичної логіки було закладено в середині XIX сторіччя ірландським математиком Дж. Булем. Останніми десятиліттями логіка набула широкого астосовування в техніці під час дослідження та розроблення електронних схем, обчислювальних машин, дискретних автоматів. Вона використовується також і в інших науках: економіці, біології, психології тощо.

4.1.2. Мова алгебри висловлювань

Мова алгебри висловлювань—це штучна мова, призначена для аналізу логічної структури складних висловлювань. Вона характеризується алфавітом (списком знакових засобів) і визначенням формули.

Під алфавітом будемо розуміти кожну не порожню множину символів:

пропозиційних змінних – x, y, z, x_1, x_2, …; логічних зв'язок ¬, ∧, ∨, →, ~, …;

технічних символів – (,) тощо.

Словом у певному алфавіті називається довільна скінчена послідовність

символів (можливо, порожня). Слово a називається підсловом b, якщо $b = b_1ab_2$ для певних слів b_1 та b_2. Слово ab називається сполученням (конкатенацією) слів a та b.

Висловлюванням називають осмислений вираз звичайної мови, якому можна приписати значення істинності. Таким виразом може стати твердження або розповідне речення, про яке можна сказати, істинне воно або хибне. Значення істинності – це абстрактний об'єкт, що ставиться у відповідність висловлюванню: істина–коли висловлювання відповідає дійсності, хибність–коли висловлювання не відповідає дійсності. Позначають: "Істина"–I, T (True), або 1; "Хибність"– X, F (False) або0.

Приклад 4.1. Визначити які з даних речень є висловлюваннями: "Дніпро впадає в Чорне море"; "Дніпро впадає в Азовське море"; "Хто ви?"; "Відстань від Землі до Сонця дорівнює 150 млн. км".

Розв'язок. Перші два речення є висловлюваннями, причому перше є істинним, адруге–хибним висловлюванням. Третє речення не є висловлюванням, оскільки воно не розповідне. Четверте речення також не є висловлюванням. Його істинність або хибність залежить від потрібної точності.

Атомом (елементарним висловлюванням) називається таке висловлювання, яке є простим розповідним реченням, тобто немає складових частин. Для позначення атомів використовують, як символи, букви латинського алфавіту з індексами або без них: A, B, p, q, r, s, p1, q2. Складні речення, як правило, складаються з простих речень, поєднаних сполучниками. Тобто прості речення, які представляють атоми та сполучники, є елементами словника, необхідного для формалізації природної мови за допомогою логіки висловлювання. Значення істинності складного висловлення визначається значеннями істинності його складових частин.

A, B, C, D, A1, B1...-позначають масиви висловлювань (схеми формул).

Поміж висловленнями використовуються різні логічні зв'язки: «якщо ..., то ...», «... або ...», «... і ...» тощо. За їхньої допомоги будуються інші нові висловлення.

Логічною операцією називається операція, в якій операндами є висловлення, а ператорами – логічні зв'язки.

За допомогою алгебри логіки можна, наприклад, описувати роботу релейно-контактних схем. Для конкретики обмежимося розгляданням двополюсних схем, у яких поміж полюсами можуть існувати релейні контакти, з'єднані послідовно чи паралельно. При цьому стан контакту – 1 (0) означає, що він замкнений (розімкнений), тобто сигнал 1 (0) переводить електронний елемент у відкритий (закритий) стан.

Розглянемо спочатку схеми з одним контактом (рис. 2.1), на яких сам контакт та його стан позначено через x, а стан двополюсника позначатимемо літерою y.

Рис. 4.1. Схеми з одним контактом

Вочевидь, змінна x є незалежною, а змінна y – залежною бульовою змінною.

У разі першої схеми коло буде замкнене, якщо буде змінено стан контакту (замкнено), тобто змінна y набуде істинного значення ($y = 1$) тоді й лише тоді, коли змінна x також набуде істинного значення ($x = 1$). У разі другої схеми, навпаки, змінна y набуде істинного значення ($y = 1$), коли змінна x збереже хибне значення, тобто стан контакту не зміниться ($x = 0$).

Перейдемо тепер до розглядання схеми „або" і схеми „і" (рис. 4.2).

Рис. 4.2. Схеми з двома контактами

Якщо контакти x_1 та x_2 з'єднані паралельно, то коло буде замкнене, тобто змінна y набуде істинного значення ($y = 1$), коли хоча б один з контактів x_1 та x_2 є замкненим, і розімкненим, тобто y набуватиме хибного значення, коли обидва контакти x_1 та x_2 є розімкненими.

При послідовному з'єднанні контактів x_1 та x_2 коло буде замкнене ($y = 1$), коли обидві змінні x_1 та x_2 набуватимуть істинного значення (тобто $x_1 = 1$, $x_2 = 1$), і озімкнене ($y = 0$), коли хоча б одна зі змінних x_1 та x_2 набуде хибного значення.

Опис більш складних релейно-контактних схем здійснюється за допомогою бульових функцій (див. п. 4.2).

Для логічних зв'язок застосовуються символи: ¬ – для „не" (заперечення», ∧ – для „і" (кон'юнкції), ∨ – для „або" (диз'юнкції), → – для «якщо …, то …» (імплікації), ~ – для «тоді й лише тоді, коли» (еквіваленції). Наприклад, якщо A та B – твердження, то $\overline{A}$, $A \wedge B$, $A \vee B$, $A \to B$, $A \sim B$ будуть, відповідно, запереченням твердження A, кон'юнкцією тверджень A та B тощо.

Задамо значення істинності висловлень A та B таблицею істинності, тоді для розглянутих зв'язок таблиця істинності має наступний вигляд:

A	B	$\overline{A}$	$A \wedge B$	$A \vee B$	$A \to B$	$A \sim B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Таким чином, якщо значення істинності простих висловлень є відомі, то значення істинності складних висловлень може бути визначено за допомогою цих таблиць.

4.1.2 Формули алгебри висловлень. Семантика. Класифікація та рівносильність формул

Алгебра висловлювань – це алгебраїчна структура [{I,X}, ∨,∧,¬,→,~,X,I] з носієм –двійковою множиною {X: "Хибність", I: "Істина"},операціями–логічними зв'язками: кон'юнкція, диз'юнкція, заперечення, імплікація, еквівалентність і константами: X–хибність і I–істина. Операції ∨,∧,→,~ є бінарними логічними зв'язками, а операція ¬ –унарною. Складні висловлювання, які побудовані з простих висловлювань, називають **формулами** або **молекулами.**

Логічні зв’язки можна розглянути як формальні позначення зв'язок, що їм відповідають

Алгебра логіки (алгебра висловлень) являє собою науку про сукупність висловлень, над якими визначені логічні операції.

Назва	Позначення	Аналог природної мови
заперечення	$\neg$	“ ні”, “ неправильно, що”
диз’юнкція	$\vee$	“або”
кон’юнкція	$\wedge$	“і”
імплікація	$\rightarrow$	“ якщо ...,то ”
еквівалентність	$\sim$, $\leftrightarrow$, $\Leftrightarrow$	“ еквівалентно”, “рівносильно”, “... тоді і тільки тоді ”, “ якщо і тільки якщо ”

Формулою алгебри висловлень (ФАВ) називається слово, яке задовольняє такому визначенню:

1) кожна пропозиційна змінна – формула;

2) якщо A та B – формули, то $(\overline{A})$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \sim B)$ – формули;

3) слово є формулою, якщо воно побудоване лише з використанням п. 1 та п. 2.

Наприклад, вирази (слова)

$((A \rightarrow B) \vee A)$, $((A \sim B) \rightarrow (C))$

є формулами, а слова $(A \sim B)$, $(\overline{C})$, A, B, C – підформули останньої формули.

З метою економії дужок операції виконуються в такому порядку (пріоритет операцій): $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\sim$.

Формули логіки висловлювань, які відповідають складним висловлюванням, приймають значення І і Х залежно від значень елементарних висловлювань і логічних зв’язків, з яких вони побудовані.

Формули логіки висловлювань зручно представляти таблицями істинності, які надають значення істинності формули в залежності від послідовного перебору всіх можливих значень істинності простих висловлень, що складають формулу:

A	B	¬A	A∧B	A∨B	A→B	A~B
X	X	I	X	X	I	I
X	I	I	X	I	I	X
I	X	X	X	I	X	X
I	I	X	I	I	I	I

Для позначення істинного висловлення використовують літеру *і* (істина, чи цифру 1, чи Т – true), а для хибного – літеру *х* **(**хибність, чи цифру 0, чи F –false).

Висловлювання А∧В називають **кон'юнкцію** висловлювання А і В, яке істинне тоді і тільки тоді, коли істинні обидва висловлювання А і В. Це логічна операція відповідає у природній мові зв'язці "і", що з'єднує два речення.

Приклад 4.2. Записати у вигляді формули логіки висловлювань і визначити істинне значення таких висловлювань:

1."8 ділиться на 4 і 8більше 6"; 2."8 ділиться на 4 і 7 більше 8".

Розв'язок. У даних висловлюваннях виділимо атоми. Їх три: А–"8 ділиться на 4", В–"8 більше 6" С– "7 більше 8".

1)А∧В=І∧І=І;

2)А∧С=І∧Х=Х.

Висловлювання А∨В називають **диз'юнкцією** висловлювань А або В, яке дійсне тоді і тільки тоді, коли істинне хоча би одне логічне висловлювання А або В. Ця логічна операція відповідає у природній мові зв'язці "або", що з'єдну єдва речення.

Приклад 4.3. Записати у вигляді формули логіки висловлювань і визначити істинні значення таких висловлювань:

1."3+2=6 або 3×2=6";

2."4–2=3 або 4×2=7".

Розв'язок. У даних висловлювань виділимо атоми: А –"3+2=6"; В– "3×2=6"; С– "4–2=3"; D– "4×2=7"

1) А∨В =Х∨І=І;

2) С∨D=Х∨Х=Х.

Слабка/нестрога диз'юнкція — це логічний сполучн ик, який є хибним лише в одному випадку, коли логічні значення його складників хибні. В усіх інших випадках цей логічний сполучник істинний.

Приклад 4.4. Розглянемо таке висловлювання: «Ця людина—викладач або письменник». Це висловлювання складене з двох простих:

A=«Ця людина —викладач»,

B=«Ця людина—письменник». Висловлювання A∨B може бути оцінене як істинне лише в тих випадках, коли:

-людина є і викладачем, і письменником;

-людина є викладачем, але не є письменником;

-людина не є викладачем, але є письменником.

Хибним це складне висловлювання буде лише тоді, якщо ця людина не є ні викладачем, ні письменником.

Приклад 4.5. Нехай задані висловлювання: A–“Микола полюбляє грати у футбол”; B–“Микола полюбляє грати у волейбол”; C–“Микола полюбляє грати у теніс”. Необхідно записати висловлювання “Микола полюбляє грати у футбол і неправильно, що він полюбляє грати у волейбол або теніс” у вигляді формули логіки висловлювань і побудувати відповідно їй таблицю істинності.

*Розв’язок.*Висловлювання “Микола полюбляє грати у волейбол або теніс” можна записати у вигляді формули логіки висловлювань як B∨C. Висловлювання “Неправильно, що він полюбляє грати у волейбол або теніс ”може бути записане у вигляді формули логіки висловлювань ¬(B∨C), оскільки заперечення застосовне до всього висловлювання, яке слідує після зв’язки “що...”. Виходячи з цього, вихідна форма складного логічного висловлювання буде мати вигляд A∧¬(B∨C).

A	B	C	B∨C	¬(B∨C)	A∧¬(B∨C)
X	X	X	X	I	X
X	X	I	I	X	X
X	I	X	I	X	X
X	I	I	I	X	X
I	X	X	X	I	I
I	X	I	I	X	X
I	I	X	I	X	X
I	I	I	I	X	X

Із таблиці істинності складного висловлювання слідує, що тільки в одному випадку дане логічн е висловлювання буде істинне, коли висловлювання A– істинне, а висловлювання B і C–хибні.

На відміну від слабкої (нестрогої) диз'юнкції **сильна (строга) диз'юнкція** передбачає використання сполучника «або» встрого розмежувальному сенсі— «A або B, але не обидва разом». Для неї в логіці висловлювань вводиться новий символ —«V».

Сильна (строга) диз'юнкція —це логічний сполучник, який буде істинним лише в тих випадках, коли логічні значення його складників не співпадають. Цей логічний сполучник буде хибним, коли логічні значення його складників співпадають.

Приклад 4.6. Розглянемо таке висловлювання: «Ця людина народилася у Києві або у Львові». Це висловлювання складене з двох простих:

A= «Ця людина народилася у Києві».

B=«Ця людина народилася у Львові».

Це висловлювання A∨B може бути оцінене як істинне тільки у тому випадку, якщо одне з простих висловлювань буде істинним, адруге —хибним.

Якщо ж обидва простих висловлювання виявляться одночасно істинними, тоді їхня диз'юнкція з необхідністю буде оцінена як хибна, оскільки неможливо одночасно народитися у двох містах.

Хибною строга диз'юнкція буде й у тому випадку, коли всі прості висловлювання, які її складають, виявляться хибними, тобто якщо людина, про яку йдеться в складному висловлюванні, не народилася ні у Києві, ні у Львові. Через нестрогу диз'юнкцію можна записати:(A∨B)\(A∧B)=(A∨B)¬∧(A∧B)

Приклад 4.7. E = «Сонце світить –Петрик грає у футбол, вітер дме–Петрик читає книжку». Атоми:

A= «Сонце світить»,

B = «Петрик грає у футбол»,

C = «Вітер дме»,

D = «Петрик читає книжку».

Форма складного висловлювання: E=(A→B)∨(C→D).

Приклад 4.8. E = “Ваш приїзд не є ні необхідним, ні бажаним”;

A = “Ваш приїзд необхідний”;

B = “Ваш приїзд бажаний”.

Форма складного висловлювання: E=¬A¬∧B.

Приклад 4.9. E= “Пошуки ворога тривали вже 3 години, протее результату не було, притаївшись, ворог нічим себе не видавав”;

A = “Пошуки ворога тривали 3 години”;

B = “Ворога знайшли (був результат)”;

C= “Ворог себе видав”.

Форма складного висловлювання: E=(A¬∧B)¬→C

Приклад 4.10. Побудувати фразу:

A = “Дехто є лікарем”;

B = “Хворий поговорив з лікарем”; C=“Хворому стало краще”.

Вислів Бехтерева: «Якщо хворому після розмови з лікарем не стало краще, то це не лікар: E=(B¬∧C)¬→A.

Раніше зазначалося, що висловлення може бути чи то істинне, чи хибне.

Інтерпретувати формулу – означає приписати їй одне з двох значень істинності. Набір правил інтерпретації формул (семантика числення висловлень) має бути композиційним, тобто значення формули має бути функцією значень її складових. Для інтерпретації можна використовувати таблиці істинності.

Наприклад, покажемо істинність формули $A \to B \sim \overline{A} \vee B$ за будь-яких інтерпретацій:

A	B	$A \to B$	$\overline{A}$	$\overline{A} \vee B$	$A \to B \sim \overline{A} \vee B$
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	1	1

Якщо існує хоча б одна інтерпретація, за якої формула набуває істинного значення, вона називається **здійсненною**, а якщо існує хоча б одна інтерпретація, за якої формула набуває хибного значення, вона називається ***спростовною.***

Серед формул алгебри висловлювань особливе місце займають ті формули $A\,(p_1, p_2, ..., p_n)$, які на всіх наборах $(a_1, a_2, ..., a_n)$ значень своїх змінних набувають значення 1.

Формула називається **тавтологією** (чи тотожно-істинною, чи загальнозначущою), якщо за будь-яких інтерпретацій її складових (змінних) вона набуває істинного значення. Позначення тавтології є $\models A$.

Наведемо приклади деяких важливих тавтологій:

$(p \vee (\neg p))$ (*закон виключення третього*),

$(\neg(p \vee (\neg p)))$ (*закон виключення суперечності*),

$(p \to p)$ (*закон тотожності*).

Формула називається **протиріччям** (тотожно-хибною), якщо за будь-яких інтерпретацій вона набуває хибного значення.

Формулу, яка не є ні тавтологією, ні суперечністю, називають **нейтральною**.

Множина всіх формул алгебри висловлювань розбивається на тавтології, суперечності та нейтральні формули. Формула, яка не є суперечністю, називається **виконуваною**.

Областю істинності (областю хибності) формули називається множина наборів значень змінних, за яких формула набуває істинного (хибного) значення.

Результати цих означень:

формула A є тавтологією тоді й лише тоді, коли A не є спростовною;

формула A є тотожно-хибна тоді й лише тоді, коли A не є здійсненною;
формула A є тавтологією тоді й лише тоді, коли $\overline{A}$ є тотожно-хибною;
формула A є тотожно-хибна тоді й лише тоді, коли $\overline{A}$ тавтологія;
формула $A \sim B$ – тавтологія тоді й лише тоді, коли A та B набувають однакових значень (є рівносильні) за всіма наборами значень змінних.

Вочевидь, що дві формули є рівносильні тоді й лише тоді, коли за будь-яких інтерпретацій їхніх змінних вони набувають однакових значень. Позначення: $A = B$.

Теорема 4.1. Дві формули A та B є рівносильні тоді й лише тоді, коли $\models A \sim B$.

Д о в е д е н н я. Нехай $P_1, P_2, ..., P_m$ є сукупність усіх простих компонентів в A та B. Якщо цим компонентам надано певні істинні значення, то перша частина обчислення значень формули $A \sim B$ полягає в обчисленні значень A та B, після чого обчислення завершується застосуванням таблиці істинності для еквіваленції, за якою значення $A \sim B$ буде істинним тоді й лише тоді, коли значення для A та B будуть однакові, тобто A та B є рівносильні.

Н а с л і д о к. Нехай с − формула, в якій є певні входження формули A, і нехай F_2 − результат заміни цього входження формули A на формулу B.
Тоді:
якщо $\models A \sim B$, то $\models F_1 \sim F2$;
якщо $\models A \sim B$ і $\models F_1$, то $\models F_2$.

Теорема 4.2. Якщо $\models A$ і $\models A \to B$, то $\models B$.

Д о в е д е н н я. Нехай $P_1, P_2, ..., P_m$ є сукупність усіх простих компонентів в A та B. Якщо цим компонентам надано певних значень істинності, то перша частина обчислення значень формули $A \to B$ полягає в обчисленні значень A та B, після чого обчислення завершуються застосуванням таблиці істинності для імплікації.

З припущення $\models A$ та $\models A \to B$ випливає, що значення A та $A \to B$ будуть істинними.

З таблиці для $A \to B$ випливає, що B теж матиме значення „істина”. Через то що це має місце для кожних значень компонентів $P_1, P_2, ..., P_m$, формула B є загальнозначуща.

Подані теореми дозволяють здійснювати еквівалентні перетворювання формул і здобувати нові загальнозначущі формули.

Наприклад, тавтології можна здобути з рівносильності заміною знака = на знак $\sim$. Скажімо, з рівносильності $A \vee AB = A$ здобуваємо тавтологію $\models A \vee AB \sim A$. Доведення тавтології, наприклад, $\models (A \to B)\,(A \to C) \sim (A \to BC)$,
можна виконати за допомогою перетворень:

$(A \to B)\,(A \to C) = (\,\overline{A} \vee B)\,(\,\overline{A} \vee C) = \overline{A} \vee \overline{A}B \vee \overline{A}C \vee BC = \overline{A} \vee BC = A \to BC.$

4.1.3.Числення висловлювань

У найзагальнішому вигляді **формальну теорію** *T* (інший термін-**числення**) будують таким чином:

1. Визначають набір основних символів –**алфавіт** теорії.

2. Конструктивно визначають **множину формул**, або правильно побудованих виразів, яка утворює мову теорії.

3. Виокремлюють підмножину формул, які називають **аксіомами** теорії.

4. Задають **правила виводу (виведення**) теорії.

Правило виводу $R(F_1, F_2, ..., F_m, G)$ – це відношення (або операція) на множині формул.

Якщо формули $F_1, F_2, ..., F_m, G$ знаходяться у відношенні *R*, то формула *G* називається **безпосередньо вивідною** з формул $F_1, F_2, ..., F_m$ за правилом *R*.

Часто правило виводу $R(F_1, F_2, ..., F_m, G)$ записують у вигляді $\frac{F_1, F_2, ..., F_m}{G}$

Формули $F_1, F_2, ..., F_m$ називають **припущеннями, посилками** або **гіпотезами** правила *R*, а формулу *G* – **висновком** або **наслідком**.

Виведенням (виводом, вивідністю) формули *B* з формул $A_1, A_2, ..., A_n$ називають послідовність формул $F_1, F_2, ..., F_m$ таку, що $F_m = B$, а будь-яка формула F_i, $i = 1,2,...,m$ є:

1)або аксіомою;

2)або однією з початкових формул $A_1, A_2, ..., A_n$;

3)або безпосередньо вивідною з формул $F_1, F_2, ..., F_{i-1}$ (або будь-якої їх підмножини) за одним з правил виведення.

Якщо існує виведення формули *B* зформул $A_1, A_2, ..., A_n$, то кажуть, що *B* є **вивідною** з $A_1, A_2, ..., A_n$ і позначають цей факт так: $A_1, A_2, ..., A_n \vdash B$. Формули $A_1, A_2, ..., A_n$ називають **посилками** або **гіпотезами** виведення. Перехід у виведенні від формули F_{i-1} до F_i називають *i*-м кроком виведення.

Доведенням формули *B* у теорії *T* називають виведення *B* з порожньої множини формул, тобто виведення, в якому як початкові формули використовують тільки аксіоми теорії.

Формула *B*, для яко існує доведення, називається формулою **довідною(вивідною)** у теорії *T*, або **теоремою** теорії *T*; факт довідності формули *B* позначають $\vdash B$.

$\vdash$-знак виведення.

Довести, що формули є тавтологіями можна за допомогою відповідних таблиць істинності. Той факт, що формула A алгебри висловлювань є тавтологією позначають так$\vDash A$. Символ $\vDash$, як і A належать метамові.

При вивченні формальних теорій існує два типи тверджень:

1) твердження самої теорії або її теореми;

2) твердження про теорію (про властивості її теорем, властивості доведень тощо).

Перші є елементами (словами, виразами, формулами) внутрішньої мови теорії, адругі-зовнішніми і формулюються у термінах мови, зовнішньої по відношенню до теорії і званої **метамовою** теорії; самі ці твердження називають **метатеоремами**.

Для розпізнавання мови логіки висловлювань і метамови дослідника між посилками і наслідком вводимо такі заміни:

Об'єктні символи мови логіки висловлювань	Суб'єктні символи метамови
Імплікація $\rightarrow$	Метаімплікація $\Rightarrow$
Кон'юнкція $\wedge$	Метакон'юнкція ,
Диз'юнкція $\vee$	Метадиз'юнкції ;

Тоді твердження, яке необхідно довести, в логіці висловлювань можна оформити у вигляді такого причинно-наслідкового відношення:

$$P_1, P_2, ..., P_{n+1}, P_n \Rightarrow \text{C}(1)$$

де P_i –посилка (причина); С–висновок (наслідок).

Читається: "*Якщо посилки* $P_1, P_2, ..., P_{n+1}, P_n$ *істинні, то і висновок С також істинний*", або "*Якщо причини* $P_1, P_2, ..., P_{n+1}, P_n$ *мали місце, то матиме місце і наслідок*".

Щоб не переплутати об'єктивне висловлювання із суб'єктивним висловлюванням, справедливість якого необхідно встановити, домовимося речення (1) називати клаузою(clause).

Клаузою називають **метапропозицію**, в якій виконується відношення порядку, оформлене через символ метаімплікації "⇒".

Твердження, справедливість яких очевидна:

1. Заперечення тавтології є суперечністю і навпаки.

2. Кожна тавтологія є виконуваною формулою (навпаки, взагалі кажучи, ні).

3. Кожна нейтральна формула є виконуваною, але не навпаки.

4. Заперечення виконуваної формули може бути, як виконуваною формулою, так і невиконуваною формулою.

Дві формули A і B алгебри висловлювань називаються **рівносильними**, якщо їм відповідає та сама функція істинності. Рівносильність формул A і B позначають за допомогою знака $=(\equiv \text{або} \leftrightarrow)$: записують $A=B$ ($A \equiv B$ або $A \leftrightarrow B$).

Очевидно, що відношення рівносильності на множині формул є **відношеннямеквівалентності**, тому часто це відношення називають **еквівалентністю**.

Теорема 1. Формули алгебри висловлювань A і B рівносильні тоді і тільки тоді, коли формула $((A \to B) \wedge (B \to A))$ є тавтологією.

З метою скорочення запису формул, подібних до формули з наведеної теореми, до сигнатури алгебри висловлень вводять додаткову операцію, що позначається ~ і позначається так: $(A \sim B)$ є скороченим записом формули $((A \to B) \wedge (B \to A))$. Отже, теорему можна сформулювати так.

Формули A іB рівносильні тоді і тільки тоді, коли формула $(A \sim B)$ є тотожно істинною.

Нехай $A(p_1, p_2, ..., p_n)$ і $B(p_1, p_2, ..., p_n)$ –дві формули алгебри висловлювань. Будемо говорити, що формула $B(p_1, p_2, ..., p_n)$ є ***логічним слідуванням*** формули $A(p_1, p_2, ..., p_n)$, якщо B приймає значення 1 для всіх тих наборів значень пропозиційних змінних, для яких формула A істинна (тобто приймає значення 1); позначатимемо це $A \Rightarrow B$.

Це означає ,що множина наборів значень змінних, для яких істинна формула A, є підмножиною множини наборів значень змінних, для яких істинна формула B, що є логічним слідуванням формули A.

Теорема 2. Формула $B(p_1, p_2, ..., p_n)$ є логічним слідуванням формули $A(p_1, p_2, ..., p_n)$ тоді і тільки тоді, коли тотожно істинною є формула $(A \to B)$.

Приклад 4.11. Формула $B(x,y,z) = (x \vee z)$ є логічним слідуванням формули $A(x,y,z) = ((x \vee y) \wedge z)$, що випливає з таблиці істинності:

$x\ y\ z$	$A\ \ B$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 0
0 1 1	1 1
1 0 0	0 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 1

Очевидно, що дві формули A і B є рівносильними тоді і тільки тоді, коли кожна з них є логічним слідуванням іншої, тобто $A=B$ тоді і тільки тоді, коли$A\Rightarrow B$ і $B\Rightarrow A$.

4.1.4. Аксіомами числення висловлювань

A1: $a\to(b\to a)$.

A2: $(a\to b)\to((a\to(b\to c))\to(a\to c\ ))$.

A3: $(a\wedge b)\to a$.

A4: $(a\wedge b)\to b$.

A5: $(a\to b)\to((a\to c)\to(a\to(b\wedge c)))$.

A6: $a\to(a\vee b)$.

A7: $b\to(a\vee b)$.

A8: $(a\to c)\to((b\to c)\to((a\vee b)\to c))$.

A9: $(a\to\neg b)\to(b\to\neg a)$.

A10: $\neg\neg a\to a$.

4.1.5.Теорема дедукції

Теорема дедукції дає нове правило виведення у логіці висловлювань, і тому вона є важливою при доведенні теорем і різних висловлювань.

Теорема дедукції 1. Якщо Γ–множина формул, A і B–формули і Γ, А $\vdash B$, то Г $\vdash A\to B$. Зокрема, якщоА $\vdash B$, то $\vdash A\to B$.

Теорема 2. $A\to B$, В$\to C\vdash A\to C$.

Теорема 3. А$\to$(В$\to C$),$B\vdash$А$\to C$

Теорема4. Для будь-яких формул A і B такі формули є теоремами формальної теорії L:

а) $\neg\neg B\to B$;

б) $B\to\neg\neg B$;

д) $(A\to B)\to(\neg B\to\neg A)$;

є) $A\to(\neg B\to(A\to B))$;

в) $\neg A \to (A \to B)$; ж) $(A \to B) \to ((\neg A \to B) \to B)$.

г) $(\neg B \to \neg A) \to (A \to B)$;

а) $\vdash \neg\neg B \to B$

4.1.6. Правила виведення

1) **правило підстановки**. Якщо *A*-вивідна формула, яка містить літеру *p* (позначимо цей факт*A*(*p*)), то вивідною є і формула *A*(*B*), що здобувається з *A* заміною всіх входжень літери *p* на довільну формулу *B*; записується

$$\frac{A(p)}{A(B)} \text{ або } S_B^p A = A(B)$$

S -формула, отримана з Ашляхом підстановки замість p змінної B.

2) **правило висновку** (**modus ponens**). Якщо*A* і *A*→*B*-вивідні формули, то вивідною є й формула *B*; записується

$$\frac{A, A \to B}{B} \text{ або } MP(A, A \to B) = B$$

Якщо відомо,що висловлювання А породжує висловлювання В та А істина, то і В істина.

4.1.7. Доведення логіки висловлювань

4.1.7.1 Доведення за допомогою таблиці істинності

Цей простий метод може бути застосований для перевірки рівносильності або нерівносильності будь-яких формул *A* і *B* довільної складності. Відтак, на перший погляд може здатися, що проблема встановлення рівносильності або нерівносильності формул алгебри висловлень є розв'язаною і до того ж найпростішим чином і отже, всі подальші дослідження у цьому напрямку є непотрібними.

Наведемо лише два міркування, які демонструють, що перше враження є оманливим.

Перше міркування пов'язане з тим, що коли кількість пропозиційних змінних у досліджуваних формулах єз начною, то застосування зазначеного простого методу може стати практично нездійсненним. Адже, вже для 30 змінних необхідно випробувати більш ніж 10^9 наборів значень змінних для кожної формули. Це тільки кількість кроків загальної процедури, а крім того, слід врахувати трудомісткість обчислення значень функцій істинності даних формул на кожному з наборів.

По-друге,-в алгебрі висловлювань у більшості випадків цікавляться не рівносильністю двох будь-яких заданих формул, а рівносильністю нескінченної множини пар формул. Потрібні твердження, згідно яких усі формули певного

типу є рівносильними відповідно формулам певного іншого типу. Якщо множини формул обох цих типів є нескінченними, то подібні твердження, очевидно, не можуть бути встановлені жодним методом, що спирається на побудові таблиць інстинності, а потребують загальних міркувань.

Приклад 4.12. Довести за допомогою таблиці істинності істинність виведення:

$$A \to B, \neg A \to B \vdash B.$$

A	B	¬A	A→B	¬A→B	A→B, ¬A→B
x	x	i	i	x	x
x	i	i	i	i	i
i	x	x	x	i	x
i	i	x	i	i	i

Приклад 4.13. Довести істинність логічного висловлювання за допомогою таблиці:

$$A \to B, \neg C \to \neg B, \neg C \Rightarrow \neg A.$$

A	B	C	¬A	¬B	¬C	A→B	¬C→¬B	P
0	0	0	1	1	1	1	1	1
0	0	1	1	1	0	1	1	0
0	1	0	1	0	1	1	0	0
0	1	1	1	0	0	1	1	0
1	0	0	0	1	1	0	1	0
1	0	1	0	1	0	0	1	0
1	1	0	0	0	1	1	0	0
1	1	1	0	0	0	1	1	0

Клауза вважається істиною, якщо одиниці наслідку (¬A) накривають всі одиниці узагальненої причини P.

4.1.7.2 Аксіоматичний метод доведення

Аксіоматичний метод перевірки тотожної істинності логічних висловлювань ґрунтується на тому, щоб серед нескінченного числа істинних клауз знайти незалежну систему аксіом, за допомогою якої можна було установити справедливість будь-яких клауз.

Оскільки доведення в логіці висловлювань будуються на відношенні порядку, то логіка висловлювань є **розширенням логіки Буля**. Тому всі істинні

тотожності логіки Буля автоматично стають справедливими кляузами логіки висловлювань.

Наприклад, **закон склеювання** логіки Буля

$$(A \vee B) \wedge (A \vee \neg B) = A$$

можна подати такими клаузами:

$$(A \vee B),\ (A \vee \neg B) \Rightarrow A$$
$$A \Rightarrow (A \vee B) \wedge (A \vee \neg B),$$
$$1 \Rightarrow ((A \vee B) \wedge (A \vee \neg B)) \sim A,$$
$$A \vee B \Rightarrow (A \vee \neg B) \rightarrow A$$

Таким чином, незалежна система аксіом логіки Буля, яка складається з чотирьох законів – комутативності, асоціативності, дистрибутивності, нуля і одиниці,–автоматично стає системою аксіом і логіки висловлювань.

Приклад 4.14. Довести істинність логічного висловлювання аксіоматичним методом:

$$A \rightarrow B, \neg C \rightarrow \neg B, \neg C \Rightarrow \neg A.$$

F1: MP $(\neg C \rightarrow \neg B, \neg C) \Rightarrow \neg B$

F2: Закон контрапозиції $A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A$

F3: MP $(\neg B \rightarrow \neg A, \neg B) \Rightarrow \neg A$

Приклад 4.15. Доведемо, що формула $a \rightarrow a$ є теоремою алгебри висловлювалгебри висловлювань.

Підставляючи в аксіому $A2$ змінну a замість змінної c і $b \rightarrow a$ замість b, матимемо вивідну формулу.

F1: $S^{b,c}_{b \rightarrow a, a} A2 = (a \rightarrow (b \rightarrow a)) \rightarrow ((a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow (a \rightarrow a))$

Підформула $a \rightarrow (b \rightarrow a)$ є аксіомою $A1$. Тоді з вивідних формул за правилом висновку виводимо формулу

F2: $MP(A1, F1) = ((a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow (a \rightarrow a))$

Замінимо в аксіомі $A1$ b на $(b \rightarrow a)$:

F3: $S^{b}_{b \rightarrow a} A1 = (a \rightarrow ((b \rightarrow a) \rightarrow a))$

Знову, застосовуючи правило висновку до двох останніх формул, матимемо, що формула $a \rightarrow a$ є вивідною.

F4: $MP(F3, F2) = (a \rightarrow a)$

4.1.7.3 Метод резолюцій

У цьому методі для доведення істинності висловлювань використовують аксіоми порядку. Суть методу зводиться до того, що дві посилки диз'юнкти з протилежними термами завжди можливо склеїти в один остаточний диз'юнкт, у якому вже не буде протилежних термів, наприклад $A \vee B,\ C \vee \neg B \Rightarrow A \vee C$, де A і

C – задовільні терми або цілі диз'юнкти з будь-яким набором термів, включаючи хибність; *B* і $\neg B$ –задовільні терми. Якщо послідовно застосувати метод резолюцій до досліджуваної клаузи, то в результаті цього отримаємо зменшення числа букв, у тому числі й до їх повного зникнення.

Приклад 4.16. Довести істиність клаузи методом резолюцій:

$$D\to F,\ A\to(E\to D),\ C\to(B\to A)\Rightarrow E\to(\neg C\to F)\to\neg B.$$

Запишемо клаузу в КНФ:

$$\neg D\vee F,\ \neg A\neg E\vee D,\ \neg C\vee\neg B\vee A\Rightarrow\neg E\vee\neg C\vee F\vee\neg B.$$

Перенесемо всі посилки в ліву частину клаузи

$$\neg D\vee F,\ \neg A\vee\neg E\vee D,\ \neg C\vee\neg B\vee A,\ E,\ C,\ \neg F,\ B\Rightarrow 0.$$

Випишемо посилки клаузи у стовпчик і склеїмо їх по черзі:

1. $\neg D\vee F$
2. $\neg A\vee\neg E\vee D$
3. $\neg C\vee\neg B\vee A$
4. E
5. C
6. $\neg F$
7. B
8. $\neg A\vee\neg E\vee F$ (1,2)
9. $\neg C\vee A$ (3,7)
10. $\neg C\vee\neg E\vee F$ (8,9)
11. $\neg C\vee F$(4,10)
12. F(5,11)
13. 0(6,12)

Із черги склеювання посилок випливає, що ця клауза істинна.

Приклад 4.17. Довести істиність клаузи методом резолюцій:

$$A\to B,\ C\to D,\ A\vee C,\ \neg A\vee\neg D,\ C\to\neg B\Rightarrow(A\vee B)\to(A\wedge B).$$

Запишемо клаузу в КНФ:

$$\neg A\vee B,\ \neg C\vee D,\ A\vee C,\ \neg A\vee\neg D,\ \neg C\vee\neg B\Rightarrow\neg(A\vee B)\vee(A\wedge B).$$

Перенесемо всі посилки в ліву частину клаузи

$$\neg A\vee B,\ \neg C\vee D,\ A\vee C,\ \neg A\vee\neg D,\ \neg C\vee\neg B,\ A\vee B,\ \neg A\vee\neg B\Rightarrow 0.$$

Випишемопосилкиклаузиу стовпчикі склеїмоїхпо черзі:

1. $\neg A\vee B$
2. $\neg C\vee D$
3. $A\vee C$
4. $\neg A\vee\neg D$
5. $\neg C\vee\neg B$
6. $A\vee B$
7. $\neg A\vee\neg B$
8. $A\vee D$ (2,3)
9. $B\vee D$ (1,8)
10. $B\vee\neg D$ (4,6)
11. B (9,10)
12. $\neg C$ (5,11)
13. A (3,12)
14. $\neg B$ (7,13)
15. 0 (11,14)

Із черги склеювання посилок випливає, що ця клауза істинна

4.1.8. Основні закони алгебри висловлень

Тотожно-істинні формули та формули рівносильності називаються законами алгебри висловлень (властивостями, правилами, теоремами). Існує нескінчена множина тавтологій та рівносильностей, а отже, і законів алгебри висловлень. Нижче наведено закони (з використанням пропозиційних змінних), які найчастіш зустрічаються на практиці:

$x \wedge 0 = 0, x \wedge 1 = x, x \vee 0 = x, x \vee 1 = 1$ – закони сталих (констант);

$x = x$ – закон тотожності;

$\bar{\bar{x}} = x$ – закон подвійного заперечення;

$x \wedge \bar{x} = 0$ – закон протиріччя;

$x \vee \bar{x} = 1$ – закон вилученого третього;

$x \vee y = y \vee x, x \wedge y = y \wedge x$ – комутативність $\vee$ та $\wedge$;

$x \vee (y \vee z) = (x \vee y) \vee z, x \wedge (y \wedge z) = (x \wedge y) \wedge z$ – асоціативність $\vee$ та $\wedge$;

$x \wedge (y \vee z) = x \wedge y \vee x \wedge z$ – перший дистрибутивний закон;

$x \vee y \wedge z = (x \vee y) \wedge (x \vee z)$ – другий дистрибутивний закон;

$x \wedge (x \vee y) = x$ – перший закон поглинання;

$x \vee x \wedge y = x$ – другий закон поглинання;

$x \vee x = x, x \wedge x = x$ – ідемпотентність;

$x \wedge y \vee x \wedge \bar{y} = x$ – перший закон склеювання;

$(x \vee y) \wedge (x \vee \bar{y}) = x$ – другий закон склеювання;

$\overline{x \vee y} = \bar{x} \wedge \bar{y}; \ \overline{x \wedge y} = \bar{x} \vee \bar{y}$ – правила де Моргана;

$\models (x \to y) \wedge x \to y$ – правило твердження, modus ponens;

$\models (x \to y) \wedge \bar{y} \to \bar{x}$ – правило спростування, modus tollens.

4.1.9 Логічний наслідок

Формула алгебри висловлень B є логічним наслідком з формули A (позначається $A \models B$), якщо B є істинне на всіх наборах значень змінних, для яких A є істинна. Наприклад, формула $B = x \vee x$ є логічним наслідком формули $A = x \wedge (y \vee \bar{y})$, тобто $x \wedge (y \vee \bar{y}) \models x \vee x$.

Теорема 5. Формула алгебри висловлень B є логічним наслідком з формули A тоді й лише тоді, коли формула $A \to B$ є загальнозначущою, тобто $A \models B \sim \models A \to B$.

Д о в е д е н н я. Відповідно до визначення імплікації, вираз $A \to B$ є хибний лише за істинного A та хибного B а отже, якщо $A \to B$ – тавтологія, то з істинності A завжди випливає істинність B, тобто $A \models B$.

І, навпаки, якщо $A \models B$, то виключається випадок, коли A є істинне та B −хибне, а отже, $A \to B$ є істинне на всіх наборах значень змінних, тобто $\models A \to B$.

Логічний наслідок можна узагальнити на сукупності формул: формула алгебри висловлень B є логічним наслідком формул $A_1, A_2, ..., A_n$ та позначається як $A_1, A_2, ..., A_n \models B$,

якщо для довільного набору значень з істинності всіх A_i, i =1, n, на цьому наборі випливає істинність B. Наприклад, розглядаючи таблицю істинності, здобудемо три ілюстрації до наведеного визначення:

$x, z, x \wedge y \to \overline{z} \models \overline{y}$ − 6-й рядок;

$x, x \to z, z \models x \vee y \to z$ − 6 та 8-й рядки;

$x \wedge y \to \overline{z}, x \to z \models \overline{x \wedge y}$ − 1, 2, 6-й рядки.

№ пп	x	y	z	$x \wedge y \to \overline{z}$	$\overline{y}$	$x \to z$	$x \vee y \to z$	$\overline{x \wedge y}$
1	0	0	0	1	1	1	1	1
2	0	0	1	1	1	1	1	1
3	0	1	0	1	0	1	0	1
4	0	1	1	1	0	1	1	1
5	1	0	0	1	1	0	0	1
6	1	0	1	1	1	1	1	1
7	1	1	0	1	0	0	0	0
8	1	1	1	0	0	1	1	0

Теорема 6. Формула алгебри висловлень B є логічним наслідком формул $A_1, A_2, ..., A_n$ тоді й лише тоді, коли формула $A_1 \wedge A_2 \wedge ... \wedge A_n \to B$ є загальнозначущою, тобто $A_1, A_2, ..., A_n \models B \sim \models A_1 \wedge A_2 \wedge ... \wedge A_n \to B$.

Д о в е д е н н я.

Доведення виконується аналогічно до доведення теореми 4.3.

Формули алгебри висловлень можна застосовувати для перевірки правильності логічних суджень, незважаючи на конкретний зміст висловлень. Що ж стосується „здорового глузду", то він має виявлятися при використовуванні законів логіки ісловлень у її конкретних додатках.

Наприклад, висловлення «A = 100<10» − хибне. Однак воно стає істинним, якщо вважати, що число 100 записане у двійковій системі числення, а 10 – у десятковій.

П р и к л а д 4.18. Перевірити правильність наступного судження. Якщо замінити мікросхему (A), телевізор працюватиме (B) за умови, що напругу увімкнено (C). Мікросхему замінили, а напругу не увімкнули. Отже, телевізор не працюватиме.

Р о з в ' я з ок. Це судження можна записати у вигляді

$$A \to B \wedge \overline{C}, A, \overline{C} \models \overline{B}.$$

Оскільки формули A, B, C не містять підформул, то можна перейти до відповідних пропозиційних змінних x, y, z.

Тоді даний логічний висновок набуде більш зручного вигляду:

$$x \to y \wedge z, x, \overline{z} \models \overline{y}.$$

Судження буде слушним, якщо формула

$$(x \to y \wedge z) \wedge x \wedge \overline{z} \to \overline{y}$$

є загальнозначуща. При викладках скористаємося основними законами алгебри висловлень:

$$(x \to y \wedge z) \wedge x \wedge \overline{z} \to \overline{y} = \overline{(\overline{x} \vee y \wedge z) \wedge x \wedge \overline{z}} \vee \overline{y} = \overline{0} \vee \overline{y} = 1 \vee \overline{y} = 1.$$

Формула є загальнозначущою, отже, судження є правильне.

В і д п о в і д ь: судження є правильне.

П р и к л а д 4.19. Я піду на лекцію (x) або залишуся в барі й вип'ю кави (y). Я

не піду на лекцію. Отже, я залишуся й вип'ю кави.

Запишемо логічне слідування:

$$x \vee y, \overline{x} \models y.$$

Перевіримо загальнозначимість:

$$(x \vee y) \wedge \overline{x} \to y = \overline{(x \vee y) \wedge \overline{x}} \vee y = \overline{x}\ \overline{y} \vee x \vee y = 1 \vee x = 1.$$

Судження є правильне.

4.2 Функції алгебри логіки. Булева алгебра

4.2.1 Способи задання булевих функцій

Для відображення інформації в комп'ютерах використовується двійкова система числення,тобто всі операції,які виконує комп'ютер, проводяться на множині{0;1}.

Джорджем Булем у середині XIXст. Було створено апарат двійкової логіки, алгебри,яку називають **булевою**.

Ця алгебра використовується при проектуванні інтелектуальних систем, при роботі з базами даних та інше.

Джордж Буль (Boole)– англійський математик і логік, один з засновників математичної логіки. Розробив алгебру логіки (булеву алгебру), основу функціонування цифрових компьютерів. 1854р. «Дослідження законів мислення».

Булева алгебра – це алгебраїчна структура $(A,\wedge,\vee,\bar{},0,1)$ з бінарними операціями $\wedge,\vee$: $A^2 \to A$, унарною операцією «$\bar{}$» : $A \to A$ і виділеними елементами 0,1 в носії *A,* які задовольняють властивості комутативності, асоціативності і дистрибутивності.

Якщо носій алгебраїчної структури $B = \{0,1\}$ складається з двох елементів, то така структура $(B,\wedge,\vee,\bar{})$ називається **двохелементною булевою алгеброю.**

Алгеброю логіки називається двохелементна булева алгебра ($B,\wedge,\vee,\bar{}$, $\to$, ~). $B = \{0,1\}$, в якій множину операцій доповнено двома бінарними операціями: імплікацією і еквівалентністю.

Відношення поміж булевими змінними подаються булевими функціями, які, подібно до числових функцій, можуть залежати від однієї, двох чи більшої кількості змінних (аргументів).

В и з н а ч е н н я. **Булевою функцією** *n* незалежних змінних називається функція $y = f(x_1, x_2, \ldots, x_n)$, $n \geq 1$, в якій кожна змінна і сама функція набувають власних значень з множини {0; l}, тобто $x_k \in \{0; 1\}$, $k = 1,\ldots,n$, $y = \{0, 1\}$.

В и з н а ч е н н я. Кортеж $(x_1, x_2, \ldots, x_n)$ конкретних значень булевих змінних називається **набором**, або **булевим вектором.**

Якщо незалежні змінні розміщено в прямому порядку, тобто у вигляді

$x = (x_1, x_2, \ldots, x_n)$, то набір називається **прямим**, а якщо їх розміщено у зворотному порядку, тобто у вигляді $x = (x_n, x_{n-1}, \ldots, x_1)$, то набір називається **зворотним**.

Областю визначення булевої функції n **аргументів** є сукупність 2^n булевих кортежів. Число різних булевих функцій є скінченне і дорівнює 2^{2^n}. За $n = 1$ число булевих функцій дорівнює 4, а за $n = 2 - 16$.

Існують такі способи задання булевих функцій.

1. Табличний. Функція задається у вигляді таблиці істинності. Наприклад така таблиця визначає функцію *y*.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

2. Графічний. Функція задається у вигляді *n*-вимірного одиничного куба, у вершинах якого записано значення функції (у кружечках) та набори значень аргументів. Наприклад, функції, задані на рис. 4.3 та 4.4.

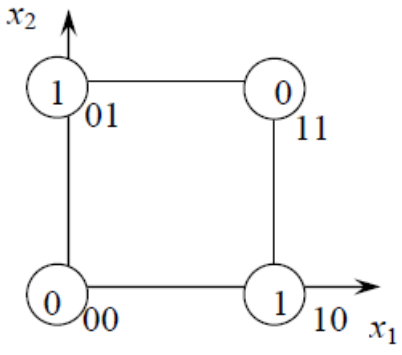


Рис. 4.3.
Двовимірний одиничний квадрат

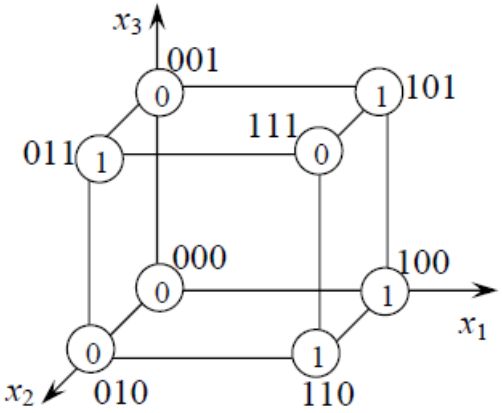
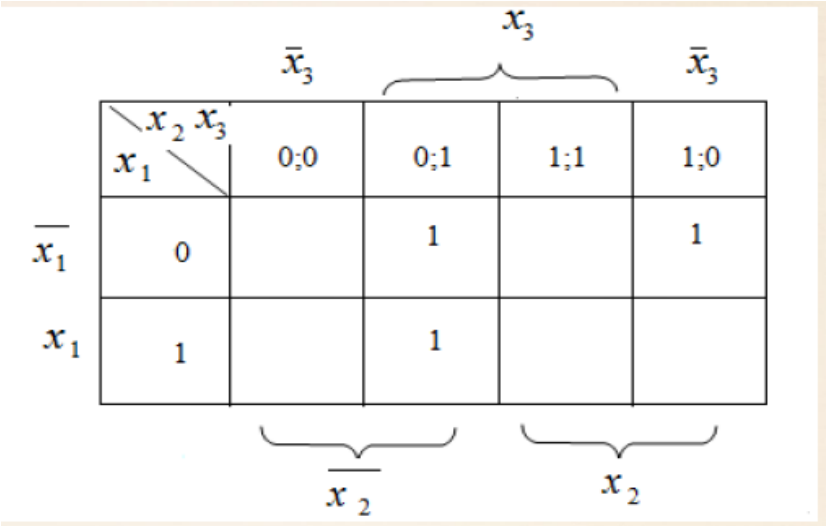


Рис. 4.4.
Тривимірний одиничний куб

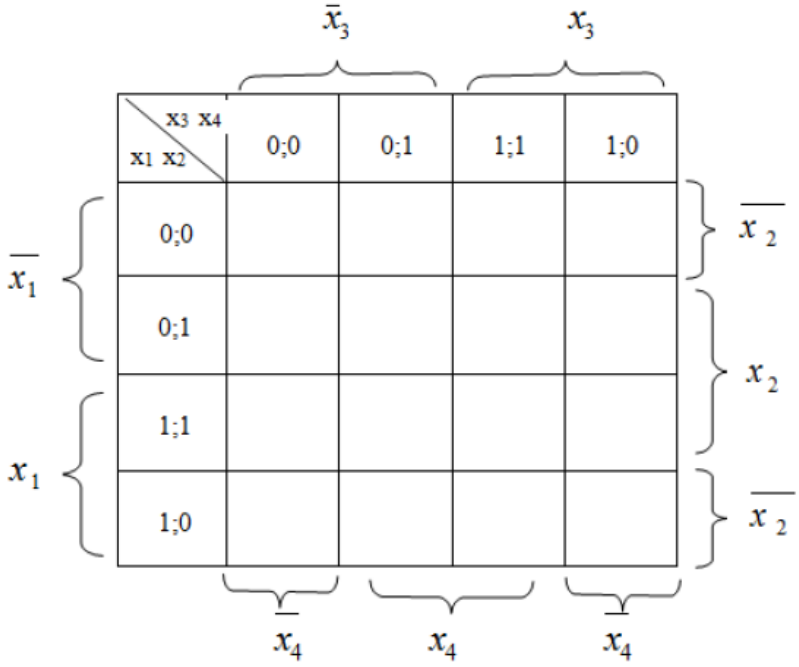
3.Координатний(картою Карно).У клітинках карти записуються значення функції (нулі зазвичай не вписують,їм відповідають порожні клітини).

Карта Карно для трьох змінних



Функція $f(x_1, x_2, x_3)$=1на наборах (0,0,1), (1,0,1), (0,1,0) або $(\overline{x_1}, \overline{x_2}, x_3), (x_1, \overline{x_2}, x_3), (\overline{x_1}, x_2, \overline{x_3})$.

Карта Карно для чотирьох змінних



4. **Числовий.**Функція задається у вигляді цілих десяткових (вісімкових, шістнадцяткових) чисел, які є еквівалентами тих наборів значень аргументів, на яких функція набуває значення 1. Наприклад, *f*= {2;4;5;7} для трьох змінних x_1, x_2, x_3.

Десяткова система числення	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

5. **Аналітичний.** Функція задається у вигляді формули. Наприклад:

$$f = x_1 + x_2 \cdot x_3$$

$$f(x_1, x_2, x_3, x_4) = \left(\left(\left(\overline{x_1 \mid x_2 \mid \overline{x_3} \mid x_4}\right) \rightarrow \left(\overline{\overline{x_1 \vee \overline{x_2} \vee x_3}}\right)\right)\right) \rightarrow$$

4.2.2 Елементарні функції алгебри логіки

Булеві функції однієї та двох незалежних змінних прийнято називати **елементарними** булевими функціями. Вони використовуються як логічні операції над булевими змінними при побудові булевих функцій багатьох незалежних змінних. Алгебра з такими логічними операціями називається **алгеброю логіки**, а булеві функції називаються ще **функціями алгебри логіки**.

Загальне число різних елементарних функцій (логічних операцій) дорівнює загальному числу функцій двох змінних, тобто $2^{2^2} = 16$ (функції однієї змінної є окремим випадком функцій двох змінних). Основними в алгебрі логіки є три логічні операції.

Заперечення (інверсія) − функція $y = f(x)$, яка набуває значення 1, коли $x = 0$, і значення 0 − за $x = 1$. Позначення: $y = \overline{x}$. Читається: «не x».

Диз'юнкція (логічне додавання) – функція $y = f(x_1, x_2)$ яка набуває значення 0 тоді й лише тоді, коли обидва аргументи дорівнюють нулеві.

Позначення: $y = x_1 + x_2$ або $y = x_1 \vee x_2$. Читається: «x_1 плюс x_2» або «x_1 або x_2».

Кон'юнкція (логічне множення) – функція $y = f(x_1, x_2)$, яка набуває значення 1 тоді й лише тоді, коли обидва аргументи дорівнюють одиниці.

Позначається: $y = x_1 \cdot x_2$ або $y = x_1 \wedge x_2$, Читається: «x_1 помножити на x_2» або «x_1 та x_2».

Таблиці істинності наведених логічних операцій мають відповідно вигляд:

x	$y = \overline{x}$
0	1
1	0

x_1	x_2	$y = x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	$y = x_1 \cdot x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Якщо операція містить один операнд, вона називається **одномісною**, або **унарною**, а якщо два, то – **двомісною,** або **бінарною**. Заперечення – це одномісна операція, а диз'юнкція та кон'юнкція − двомісні. При цьому вирази $\overline{x}$, $x_1 + x_2$, $x_1 \cdot x_2$ є прикладами логічних формул. Більш складні формули дістаємо за рахунок суперпозиції логічних формул, які, звичай беруться у круглі дужки. Наприклад: $y = x_1 + x_2 (\overline{x_1} + x_2)(x_1 + x_1 \overline{x_2})$.

Основні логічні операції над висловлюваннями.

Унарні

Константа нуля

Логічні операції можна задавати за допомогою таблиць істинності, що показують відповідність значень істинності висловлювань.

x	*F(x)*
0	0
1	0

Константа одиниці

x	*F(x)*
0	1
1	1

Повторення x

x	*F(x)*

0	0
1	1

Логічним запереченням висловлювання *x* називається висловлювання, яке є істинним тоді і тільки тоді, коли висловлювання *x* є хибним. Заперечення позначається $\bar{x}$ або¬*x* (читається: «не *x*»).

x	$\bar{x}$
1	0
0	1

Бінарні

Константа нуля

x	*y*	*F (x, y)*
1	1	0
1	0	0
0	1	0
0	0	0

Кон`юнкцією** двох висловлювань *x* і *y* називається висловлювання **і**, яке є істинним тоді і тільки тоді, коли істинними є обидва висловлювання *x* і *y*. Кон`юнкція позначається: $x \wedge y$, або *x* &*y* (читається: «*x* **і *y*»). Таблиця істинності для $x \wedge y$ має вигляд:

x	*y*	$x \wedge y$
1	1	1
1	0	0
0	1	0
0	0	0

Заборона по у*:* $f(x, y) = x \Delta y$ або $x \wedge \bar{y}$

x	*y*	*f (x, y)*
1	1	0
1	0	1
0	1	0

0	0	0

Змінна x*: f(x,y)=x*

x	*y*	*f (x, y)*
1	1	1
1	0	1
0	1	0
0	0	0

Заборона по x*:* $f(x,y) = y\Delta x$ або $y \wedge \bar{x}$

x	*y*	*f (x, y)*
1	1	0
1	0	0
0	1	1
0	0	0

Змінна y*: f(x,y)=y*

x	*y*	*f (x, y)*
1	1	1
1	0	0
0	1	1
0	0	0

Диз`юнкцією двох висловлювань** *x* і *y* називається висловлювання, хибне тоді і тільки тоді, коли обидва висловлювання *x* і *y* хибні. Диз`юнкція позначається $x \vee y$ або $x+y$ (читається: «*x* **або *y*»). Таблиця істинності для $x \vee y$ має вигляд:

x	*y*	$x \vee y$
1	1	1
1	0	1
0	1	1
0	0	0

Сума за модулем 2 *(*альтернативною диз`юнкцією, логічною сумою, виключаючим «АБО», строгою диз`юнкцією) двох висловлювань *x* і *y* називається висловлювання, що є істинним тоді і тільки тоді, коли обидва висловлювання *x* і *y приймають різні значення*. Диз`юнкція позначається $x \oplus y$ (читається: «або *x,* або *y*»). Таблиця істинності для $x \oplus y$ має вигляд заперечення еквівалентності, виключаючи «АБО»

$f(x,y) = x \oplus y$ або $(\bar{x} \wedge y) \vee (x \wedge \bar{y})$

x	y	$f(x,y)$
1	1	0
1	0	1
0	1	1
0	0	0

Стрілка Пірса– це заперечення диз`юнкції. Стрілка Пірса позначаєтьсяX ↓ Y. Читається «ані X, ані Y».

Введена до розгляду Чарльзом Пірсом (CharlesPeirce) в 1880—1881 р.р. Таблиця істинності для стрілки Пірса має вигляд: $f(x,y) = x \downarrow y$ або $\overline{x \vee y}$ або $\bar{x} \wedge \bar{y}$

x	y	$f(x,y)$
1	1	0
1	0	0
0	1	0
0	0	1

Заперечення x*:* $f(x,y) = \bar{y}$

x	y	$f(x,y)$
1	1	0
1	0	1
0	1	0
0	0	1

Імплікацією двох висловлювань *x* і *y* називається висловлювання, хибне тоді і тільки тоді, коли висловлювання *x* є істиною, а *y* – хибне. Імплікація позначається: *x*→*y* (читається: «*x* тягне за собою *y*» або «з *x* слідує *y*»). Права імплікація: $f(x,y) = x \vee \bar{y}$ або $y \to x$:

x	y	$f(x,y)$
1	1	1
1	0	1
0	1	0
0	0	1

Ліва імплікація: $f(x,y) = \bar{x} \vee y$ або $x \to y$

x	y	$f(x, y)$
1	1	1
1	0	0
0	1	1
0	0	1

Еквіваленцією (еквівалентністю) двох висловлювань x і y називається висловлювання, що є істинним тоді і тільки тоді, коли істинності висловлювань співпадають. Еквіваленція позначається: $x \leftrightarrow y$, або $x \sim y$,або $(\bar{x} \wedge \bar{y}) \vee (x \wedge y)$, або $\overline{x \oplus y}$ (читається: «x еквівалентний y» або «x тоді і тільки тоді, коли y»). Таблиця істинності для$x \leftrightarrow y$має вигляд:

x	y	$x \leftrightarrow y$
1	1	1
1	0	0
0	1	0
0	0	1

Заперечення у: $f(x, y) = \bar{x}$

x	y	$f(x, y)$
1	1	0
1	0	0
0	1	1
0	0	1

Штрих Шеффера – це заперечення кон`юнкції.

Введений до розгляду Генрі Шеффером в 1913 р. Штрих Шеффера позначається $x|y$ або $\overline{x \wedge y}$ або $\bar{x} \vee \bar{y}$, задається наступною таблицею істинності:

x	y	x|y
1	1	0
1	0	1
0	1	1
0	0	1
0	0	1

Константа одиниці

x	y	$f(x, y)$
1	1	1
1	0	1

0	1	1
0	0	1

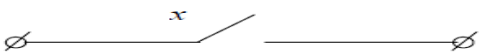
Технічну реалізацію функцій однієї змінної наведено на рис. 4.6.

$F(x)=0$

Константа 0

$F(x)=1$

Константа 1



$F(x)=x$

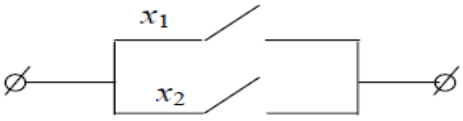
Повторення

$F(x)=\bar{x}$

Заперечення

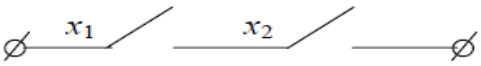
Рис. 4.6. Технічна реалізація функцій однієї змінної

Технічну реалізацію деяких функцій двох змінних показано на рис. 4.7.



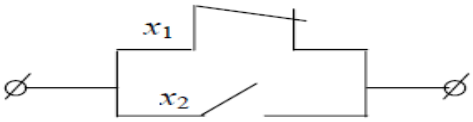
$F(x_1, x_2)= x_1 + x_2$

Диз'юнкція



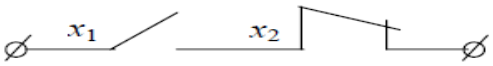
$F(x_1, x_2)= x_1 \cdot x_2$

Кон'юнкція



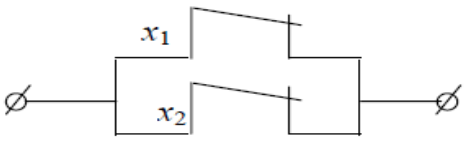
$F(x_1, x_2)= x_1 \to x_2$

Ліва імплікація



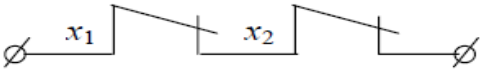
$F(x_1, x_2)= x_1 \leftarrow x_2$

Права імплікація



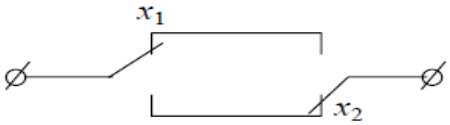
$F(x_1, x_2)= x_1 | x_2$

Штрих Шеффера



$F(x_1, x_2)= x_1 \downarrow x_2$

Стрілка Пірса



$F(x_1, x_2)= x_1 \oplus x_2$

Сума за модулем 2

$F(x_1, x_2)= x_1 \sim x_2$

Еквівалентність

4.2.3 Основні властивості функцій алгебри логіки

Формулою алгебри логіки або **логічним виразом** називається скінчена послідовність булевих змінних, функцій та їх суперпозицій, пов'язаних знаками логічних операцій та круглими дужками.

Суперпозицією називається спосіб одержання нових функцій шляхом підстановки значень одних функцій замість значент аргументів інших функцій, при цьому деякі з функцій можуть тотожно співпадати з однією зі змінних.

Принцип суперпозиції

Функцію, що відповідає формулі *F*, називають суперпозицією функцій з множини функцій, а процес здобуття функції з множини функцій – операцією суперпозиції.

Приклад 4.21. Функція $F_1 = (((x_1 x_2) + x_1) + x_2)$ будується за три кроки (таблиця 1):

- $(x_1 x_2)$
- $((x_1 x_2) + x_1)$
- $(((x_1 x_2) + x_1) + x_2) = F_1$

Таблиця 1

x_1	x_2	x_1x_2	$(x_1x_2)+x_1$	$F_1 = (((x_1x_2)+x_1)+x_2)$
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

Приклад 4.22. Функція $F_2 = \overline{x_1}(x_2 + x_3)$ будується такод за три кроки (таблиця 2):

- $(x_2 + x_3)$
- $(\overline{x_1})$
- $\left((x_2 + x_3)\overline{x_1}\right) = F_2$

Таблиця 2

x_1	x_2	x_3	$x_2 + x_3$	$\overline{x_1}$	$F_2 = \overline{x_1}\,(x_2 + x_3)$
0	0	0	0	1	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

При складанні логічного висловлення із простих використовується принцип суперпозиції, тобто підстановка у функцію замість її аргументу інших функцій. Замість будь-якої змінної використовується як власне незалежна змінна, аргумент, так і змінна, що є функцією інших змінних. Цей принцип є правильним також у звичайній алгебрі. За принципом суперпозиції з двомісних булевих функцій можна побудувати будь-яку булеву функцію.

Принцип суперпозиції дає змогу на основі трьох основних елементарних функцій (заперечення, кон'юнкція, диз'юнкція) здобути складене логічне висловлення, що описує функціонування цифрових систем й автоматів.

Якщо F_1 і F_2 - формули, то $\overline{F_1}, (F_1 \wedge F_2)(F_1 \vee F_2), (F_1 \to F_2), (F_1 \approx F_2)$ - також формули.

На відміну від табличного задання, зображення функції формулою не єдине.

Фомули, що зображають одну й ту саму функцію на зивають **еквівалентними** або **рівносильними.**

Приклад 4.23. Функцію штрих Шеффера можна зобразити за допомогою основних операцій булевої алгебри формулами:

$$F = x_1 | x_2 = \overline{x_1 x_2} = \overline{x_1} \vee \overline{x_2},$$

а функцію стрілка Пірса:

$$F = x_1 \downarrow x_2 = \overline{x_1 \vee x_2} = \overline{x_1} \wedge \overline{x_2}$$

Функція алгебри логіки – це рівність, у лівій частині якої стоїть булева змінна, а у правій − логічний вираз. Отже, функція алгебри логіки визначається формулою. Наприклад, $x_1(x_2 \to x_3)$ − логічний вираз, $y = x_1 + x_2 x_3$ – булева функція.

При обчислюванні логічних виразів дотримуються такого приорітету операцій: насамперед обчислюються функції, потім заперечення, після чого логічне множення і, врешті, логічне додавання. Вирази, які стоять у дужках,

обчислюються в першу чергу. Інші операції мають найменший пріоритет. Порядок їхнього виконання визначається круглими дужками.

Якщо у формулі відсутні дужки, то операції виконуються в такій послідовності:

Заперечення;

Кон'юнкція;

Диз'юнкція;

Імплікація;

Еквівалентність;

Для стрілки Пірса і штриха Шеффера пріоритет не визначений.

Функції, які зводяться до залежності від меншого числа змінних, називаються **виродженими**, а функції, які суттєво залежать від усіх змінних, є **невиродженими**. Наприклад, серед функцій однієї змінної є дві вироджені функції. Це *F(x)=0*, *F(x)=*1, які можна розглядати як функції від нуля змінних.

Функція багатьох змінних $f\ (x_1, x_2, ..., x_n)$ називається функцією, яка **зберігає константу 0**, якщо *f* (0, 0, ..., 0) = 0.

Функція багатьох змінних $f\ (x_1, x_2, ..., x_n)$ називається функцією, яка **зберігає константу 1**, якщо *f* (1, 1, ..., 1) = 1.

Логічна функція $f^*(x_1, x_2, ..., x_n)$ називається **двоїстою** до функції $f\ (x_1, x_2, ..., x_n)$, якщо має місце рівність $f^*(x_1, x_2, ..., x_n) = \overline{f}\left(\overline{x_1}, \overline{x_2}, ..., \overline{x_n}\right)$. Наприклад, функція $f_1 = x_1 \cdot x_2$ має властивість двоїстості до функції $f_2 = x_1 + x_2$,тому що $x_1 \cdot x_2 = \overline{\overline{x_1} + \overline{x_2}}$.

$x1$	$x2$	$f1=x1 \cdot x2$
0	0	0
0	1	0
1	0	0
1	1	1

$\overline{x_1}$	$\overline{x_2}$	$f2=\overline{x_1} + \overline{x_2}$	$\overline{f_2}$
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

Логічна функція $f\ (x_1, x_2, ..., x_n)$ називається **самодвоїстою**, якщо має місце рівність

$$f\ (x_1, x_2, ..., x_n) = \overline{f}\left(\overline{x_1}, \overline{x_2}, ..., \overline{x_n}\right).$$

Наприклад, функція $f\ (x_1, x_2, x_3) = x_1 \cdot x_2 + x_2 \cdot x_3 + x_1 \cdot x_3$ є самодвоїстою, тому що $x_1 \cdot x_2 + x_2 \cdot x_3 + x_1 \cdot x_3 = \overline{\overline{x_1} \cdot \overline{x_2} + \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_3}}$.

Принцип двоїстості:

Для того, щоб одержати двоїсту формулу булевої алгебри, необхідно замінити в ній всі кон'юнкції на диз'юнкції, диз'юнкції на кон'юнкціїї, 0 на1, 1

на 0 і використовувати дужки, де необхідно, щоб порядок операцій залишався попереднім.

Приклад 4.24. Знайти функцію двоїсту до функції $f = x_1 \vee \overline{x_2}\ x_3 \vee 1$.

Розв'язок. Скористаємося правилом отримання двоїстих функцій $f* = x_1 \wedge x_2 \vee x_3 \wedge 0$.

Доведемо за допомогою таблиці істинності.

x_1	x_2	x_3	$\overline{x_2}$	$\overline{x_2}x_3$	$x_1 \vee \overline{x_2}x_3$	$x_1 \vee \overline{x_2}x_3 \vee 1$
0	0	0	1	0	0	1
0	0	1	1	1	1	1
0	1	0	0	0	0	1
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	0	0	1	1
1	1	1	0	0	1	1

x_1	x_2	x_3	$\overline{x_1}$	$\overline{x_3}$	$x_2 \vee \overline{x_3}$	$\overline{x_1} \wedge (x_2 \vee \overline{x_3})$	$f^* = x_1 \wedge (\overline{x_2} \vee x_3) \wedge 0$	$\overline{f^*}$
0	0	0	1	1	1	1	0	1
0	0	1	1	0	0	0	0	1
0	1	0	1	1	1	1	0	1
0	1	1	1	0	1	1	0	1
1	0	0	0	1	1	0	0	1
1	0	1	0	0	0	0	0	1
1	1	0	0	1	1	0	0	1
1	1	1	0	0	1	0	0	1

Булева функція називається **монотонною**, якщо для будь-яких пар наборів значень змінних $(x_1, x_2, ..., x_n)$ та $(y_1, y_2, ..., y_n)$ для яких виконується відношення $(x_1.x_2, ..., x_n) \le (y_1, y_2, ..., y_n)$ правильна і нерівність $f(x_1, x_2, ..., x_n) \le f(y_1, y_2, ..., y_n)$.

Приклад 4.25. Дослідити на монотонність функцію $f(x, y) = x \sim y$.

Розв'язок. Запишемо всі набори значень змінних, для яких виконується відношення порядку, визначемо значення функцій на даних наборах та порівняємо їх.

$(0;0) \le (0;1)$ $f(0;0) = 1$ $f(0;1) = 0$ $f(0;0) \not\le f(0;1)$;

$(0;0) \le (1;0)$ $f(0;0) = 1$ $f(1;0) = 0$ $f(0;0) \not\le f(1;0)$;

$(0;0) \le (1;1)$ $f(0;0) = 1$ $f(1;1) = 1$ $f(0;0) \le f(1;1)$;

$(0;1) \le (1;0)$ $f(0;1) = 0$ $f(1;0) = 0$ $f(1;0) \le f(0;1)$;

$(0;1) \le (1;1)$ $f(0;1) = 0$ $f(1;1) = 1$ $f(0;1) \le f(1;1)$;

$(1;0) \le (1;1)$ $f(1;0) = 0$ $f(1;1) = 1$ $f(1;0) \le f(1;1)$

$$f(x, y) = x \sim y$$

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	1

Функція $f(x, y) = x \sim y$ не є монотонною.

Функція багатьох змінних називається **лінійною**, якщо її можна подати у вигляді многочлена

$f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus ... \oplus a_n x_n$, що не містить кон'юнкцій, де $a_i \in \{0, 1\}$, $i = 0, n$. Наприклад, функція $x \sim y = x \oplus y \oplus 1$ є лінійною.

4.2.4 Повні системи функцій. Базис

Визначення. Система функцій алгебри логіки $\{ f_1, f_2, ..., f_n \}$ називається **повною**, якщо кожна інша функція алгебри логіки може бути виражена за допомогою суперпозицій цих функцій. При цьому стверджують, що повна система функцій утворює базис у логічному просторі.

Визначення. Мінімальним базисом є такий базис, вилучення з якого будь-якої функції порушує його повноту.

Класи булевих функцій

Класами Поста називають такі п'ять множин булевих функцій:

T_0 - клас функцій, що зберігає 0;

$T_0 = \{f \in P \mid f(0,0,...,0) = 0\}$.

T_1 - клас функцій, що зберігає 1;

$T_1 = \{f \in P \mid f(1,1,...,1) = 1\}$.

S – клас самодвоїстих функцій;

$S = \left\{f \in P \middle| \forall (a_1, a_2, ..., a_n) f(a_1, a_2, ..., a_n) = \overline{f}(\overline{a_1}, \overline{a_2}, ..., \overline{a_n})\right\}$.

M- клас монотонних функцій;

$M = \{f \in P \mid \forall \alpha \forall \beta \alpha \le \beta \to f(\alpha) \le f(\beta)\}$.

L – клас лінійних функцій;

$L = \{f \in P | f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus ... \oplus a_n x_n\}$.

Булева функція $f(x_1, x_2, ..., x_n)$ може належати одному або декільком класам Поста, та не належати до жодного.

Теорема Поста-Яблонського Для того щоб система функцій була повною, необхідно і достатньо, щоб вона містила в собі хоча б одну функцію: незберігаючу константу 0, незберігаючу константу 1, несамодвоїсту, немонотонну й нелінійну.

З теореми випливає, що таких функцій має бути п'ять. Але, через те що деякі функції мають одразу кілька потрібних властивостей, базис може складатися з меншого числа функцій.

Властивості функції	**Функції**									
	0	**1**	–	+	•	/	↓	→	⊕	←
Незберігаюча 0		*	*			*	*	*		
Незберігаюча 1	*		*			*	*		*	*
Несамодвоїста	*	*		*	*	*	*	*	*	*
Немонотонна			*			*	*	*	*	*
Нелінійна				*	*	*	*	*		*

З таблиці видно, що повними системами функцій будуть: {¬, +, •},
{ ¬, +},
{¬, •}, {/}, {↓}, {0, →} тощо.

Приклад 4.26. Визначити за допомогою теореми Поста, чи є система $\{x \vee y, \bar{x}\}$ функціонально повною.

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1
$\bar{x}$	$\bar{y}$	$\overline{\bar{x} \vee \bar{y}}$
1	1	0
1	0	0
0	1	0
0	0	1

1) $f(x) = x \vee y = xy \oplus x \oplus y$ - функція нелінійна, бо має кон`юнкцію;
2) Зберігає 0 $f(0,0) = 0$;
3) Зберігає 1 $f(1,1) = 1$
4) $(0,0) \leq (0,1)$; $f(0, 0) = 0$; $f(0,1) = 1$; $f(0,0) \leq f(0,1)$;

$(0,0) \le (1,0)$; $f(0,0)=0$; $f(1,0)=1$; $f(0,0) \le f(1,0)$;
$(0,0) \le (1,1)$; $f(0,0)=0$; $f(1,1)=1$; $f(0,0) \le f(1,1)$;
$(0,1) \le (1,0)$; $f(0,1)=1$; $f(1,0)=1$; $f(0,1) \le f(1,0)$;
$(0,1) \le (1,1)$; $f(0,1)=1$; $f(1,1)=1$; $f(0,1) \le f(1,1)$;
$(1,0) \le (1,1)$; $f(1,0)=1$; $f(1,1)=1$; $f(1,0) \le f(1,1)$;

5) Несамодвоїста $f(x,y) \neq \overline{f}(\overline{x},\overline{y})$.

x	$\overline{x}$
0	1
1	0
$\overline{x}$	$\overline{\overline{x}}$
1	1
0	0

1) $f(x)=\overline{x}=1 \oplus x$ - лінійна;
2) Не зберігає 0 $f(0) \neq 0$;
3) Не зберігає 1 $f(1) \neq 1$;
4) Самодвоїста $f(x)=\overline{f}(\overline{x})$.

Зведемо всі дані в таблицю:

Назва властивості	$\overline{x}$	$x \vee y$
Лінійність	-	+
Монотонність	-	+
Зберігає 0	-	+
Зберігає 1	-	+
Самодвоїстість	+	-

В кожному рядку таблиці присутній знак «мінус». Отже, для кожного класу є хоча б одна функція, що не зберігає 0, що не зберігає 1, нелінійна, немонотонна

і несамодвоїста. За теоремою Поста така система функцій є функціонально повною.

Приклад 4.27. Довести повноту або неповноту приведеної системи булевих функцій $\{f_1 = x_1 \wedge x_2,\ f_2 = 0,\ f_3 = x_1 \approx x_2\}$.

Застосуємо теорему Поста про повноту. Для того, щоб система булевих функцій була повною необхідно і достатньо, щоб вона містила хоча б одну функцію, що не зберігає 0, хоча б одну, що не зберігає 1, хоча б одну нелінійну функцію, хоча б одну несамодвоїсту і хоча б одну немонотонну функції. Складемо таблицю істинності функцій.

x_1	x_2	f_1	f_2	f_3
0	0	0	0	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Функція f_3 не зберігає 0, так як f_3 (0,0)=1. Функція f_2 не зберігає 1, так як f_2 (1,1)=0. Функція f_1 - несамодвоїста, так як $f_1(0,1) \neq \overline{f_1}(1,0)$ =1. Функція f_3 - немонотонна, так як для впорядкованих наборів $(0,0) \leq (0,1)$ вона приймає значення $f_3(0,0)=1 \geq f_3(0,1)$. Функція f_1 - нелінійна. Покажемо це. Нехай

$f_1 = a_0 \oplus a_1 x_1 \oplus a_2 x_2$

$f_1(0,0) = a_0 \oplus 0 \oplus 0 = 0 \Rightarrow a_0 = 0$;

$f_1(0,1) = a_0 \oplus 0 \oplus a_2 = 0 \oplus a_2 = 0 \Rightarrow a_2 = 0$; f=0, що не вірно.

$f_1(1,0) = a_0 \oplus a_1 = 0 \oplus a_1 = 0 \Rightarrow a_1 = 0$.

Таким чином за теоремою Поста система $\{f_1 = x_1 \wedge x_2,\ f_2 = 0, x_3 = x_1 \approx x_2\}$ повна, бо містить функцію, що не зберігає 0, функцію, що не зберігає 1, несамодвоїсту функцію, нелінійну функцію і немонотонну функцію.

Приклад 4.28. Визначити до яких класів Поста відноситься функція $F = \overline{x_1} x_3 \vee x_1 \overline{x_3}$, додати, якщо це необхідно до F елементарні функції, щоб отримана множина була повною.

Складемо таблицю істинності для функції $F = \overline{x_1} x_3 \vee x_1 \overline{x_3}$

x_1	x_2	x_3	$\overline{x_1}$	$\overline{x_3}$	$\overline{x_1} x_3$	$x_1 \overline{x_3}$	F		

0	0	0	1	1	0	0	0		0 1 0 1 1 0 1 0
0	0	1	1	0	1	0	1	$x_3 \Rightarrow$	1 1 1 0 1 1 1
0	1	0	1	1	0	0	0		0 0 1 1 0 0
0	1	1	1	0	1	0	1		0 1 1 1 0
1	0	0	0	1	0	1	1	$x_1 \Rightarrow$	1 0 0 1
1	0	1	0	0	0	0	0		1 0 1
1	1	0	0	1	0	1	1		1 1
1	1	1	0	0	0	0	0		0

Функція F зберігає 0 ($F \in T_0$), так як $F(0,0,0,)=0$;

Функція Fне зберігає 1 ($F \notin T_1$), так як $F(1,1,1)=0$;

Функція Fне є монотонною ($F \notin M$), так як на порівнюваних наборах $(0,0,1) \leq (1,0,1)$, отримуємо $F(0,0,1)=1$ і $F(1,0,1)=0$ $F(0,0,1) \geq F(1,0,1)$.

Функція F не є самодвоїстою ($F \notin S$), так як на протилежних наборах приймає однакові значення $F(0,0,0)=F(1,1,1)=0$

Функція Fє лінійною ($F \in L$), так як її поліном має вигляд $F = x_1 \oplus x_3$.

Щоб доповнити функцію до повної системи необхідно ввести функцію, що не зберігає 0 і є нелінійною, в якості такої функції можна обрати $G = x \to y$ (імплікація). Множина $\{F, G\}$- повна.

4.2.5 Булева алгебра та її основні закони

Визначення. Булевою алгеброю називається множина логічних функцій з операціями диз'юнкція, кон'юнкція та заперечення, − тобто алгебра, базисом якої є система функцій { ¬, +, •}.

Операції булевої алгебри звичайно називають булевими операціями, а функції − булевими функціями.

Розглянемо тепер основні закони булевих операцій:

1) комутативний (для диз'юнкції та кон'юнкції):

$$x_1 \vee x_2 = x_2 \vee x_1; \quad x_1 \wedge x_2 = x_2 \wedge x_1;$$

2) асоціативний:

$$x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3; \quad x_1 \wedge (x_2 \wedge x_3) = (x_1 \wedge x_2) \wedge x_3;$$

3) дистрибутивний:

$x_1 \wedge (x_2 \vee x_3) = x_1 \wedge x_2 \vee x_1 \wedge x_3$ – перший дистрибутивний закон;

$x_1 \vee (x_2 \wedge x_3) = (x_1 \vee x_2) \wedge (x_1 \vee x_3)$ – другий дистрибутивний закон;

4) ідемпотентний:

$$x \vee x = x;\ x \wedge x = x;$$

5) інверсний (формули де Моргана):

$$\overline{x_1 \vee x_2} = \overline{x_1} \wedge \overline{x_2};\quad \overline{x_1 \wedge x_2} = \overline{x_1} \vee x_2;$$

6) закон вилученого третього (для диз'юнкції) і закон суперечності (для кон'юнкції):

$$x \vee \overline{x} = 1;\ x \wedge \overline{x} = 0;$$

7) закони елімінації:

$$x_1 \wedge (x_1 \vee x_2) = x_1;\quad x_1 \vee (x_1 \wedge x_2) = x_1$$

У булевій алгебрі мають місце такі властивості:

$$\overline{0} = 1;\ \overline{1} = 0;\ x \vee 0 = x;\ x \vee 1 = 1;\ x \wedge 0 = 0;\ x \wedge 1 = x;\ \overline{\overline{x}} = x.$$

Решта функцій двох змінних логіки виражаються через базис булевої алгебри в такий спосіб:

$$x_1 \to x_2 = \overline{x_1} \vee x_2;\ x_1 \leftarrow x_2 = x_1 \wedge \overline{x_2};$$

$$x_1 \mid x_2 = \overline{x_1} \vee \overline{x_2};\ x_1 \downarrow x_2 = \overline{x_1} \wedge \overline{x_2};$$

$$x_1 \oplus x_2 = x_1 \wedge \overline{x_2} \vee \overline{x_1} \wedge x_2;\quad x_1 \sim x_2 = x_1 \wedge x_2 \vee \overline{x_1} \wedge \overline{x_2}.$$

У справедливості цих формул легко переконатися за допомогою таблиці істинності.

Закони булевої алгебри та її властивості надають можливість виконувати перетворювання логічних виразів з метою побудови найбільш простих (компактних) формул.

Приклад 4.29. Довести справедливість формул поглинання:

$$x_1 \vee x_1 \wedge x_2 = x_1;\qquad x_1 \wedge (x_1 + x_2) = x_1.$$

Доведення

$$x_1 \vee x_1 \wedge x_2 = x_1 \wedge 1 \vee x_1 \wedge x_2 = x_1 \wedge (1 \vee x_2) = x_2 \wedge 1 = x_1;$$

$$x_1 \wedge (x_1 \vee x_2) = x_1 \wedge x_1 \vee x_1 \wedge x_2 = x_1 \vee x_1 \wedge x_2 = x_1.$$

Приклад 4.30. Спростити: $((x_1 \downarrow x_2) \mid x_3) \to (x_1 \to x_3)$.

Розв'язок

$$\left(\left(\overline{x_1} \wedge \overline{x_2}\right) \mid x_3\right) \to \left(\overline{x_1} \vee x_3\right) = \left(\overline{\overline{x_1} \wedge \overline{x_2}} \vee \overline{x_3}\right) \to \left(\overline{x_1} \vee x_3\right) = \overline{\overline{\overline{x_1} \wedge \overline{x_2}} \vee \overline{x_3}} \vee \overline{x_1} \vee x_3 =$$

$$= \overline{x_1} \vee x_3 \wedge \left(1 \vee \overline{x_1} \wedge \overline{x_2}\right) = \overline{x_1} \vee x_3$$

4.2.6 Нормальні форми булевих функцій

Визначення. **Елементарною диз'юнкцією (кон'юнкцією)** називається диз'юнкція (кон'юнкція) скінченного числа булевих змінних, у якій кожна змінна зустрічається не більше одного разу в прямому чи інверсному вигляді.

Наприклад:

$x_1 \vee \overline{x_2}$, $\overline{x_1} \vee x_2 \vee \overline{x_4}$ – елементарні диз'юнкції;

$x_1 \wedge x_2 \wedge \overline{x_3}$, $\overline{x_5} \wedge x_7 \wedge \overline{x_9} \wedge x_{10}$ – елементарні кон'юнкції.

Визначення. Диз'юнктивною нормальною формою (кон'юнктивною нормальною формою) називається формула, яка містить диз'юнкцію (кон'юнкцію) скінченного числа різних елементарних кон'юнкцій (диз'юнкцій).

Позначення: ДНФ, КНФ.

Наприклад: $x_1 \wedge \overline{x_2} \vee x_1 \wedge x_3$, $x_1 \wedge x_5 \vee \overline{x_6}$ – ДНФ;

$(x_1 \vee x_2) \wedge \overline{x_3}$, $(\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_4}) \wedge (x_2 \vee x_5)$ – КНФ.

Визначення. ДНФ (КНФ) називається **досконалою** і позначається ДДНФ. (ДКНФ), якщо в кожній її елементарній кон'юнкції (диз'юнкції) подано всі змінні.

Наприклад: $x_1 \wedge \overline{x_2} \wedge x_2 \vee x_1 \wedge x_2 \wedge \overline{x_3}$ – ДДНФ;

$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_3 \vee x_4)$ – ДКНФ.

Для того щоб привести формулу до ДДНФ, потрібно:

– за допомогою законів та властивостей булевої алгебри привести її до ДНФ;

– якщо в елементарній кон'юнкції не міститься змінної x_i із загальної кількості змінних, які входять до даної формули, додати до цієї кон'юнкції співмножник $x_i \vee \overline{x_i}$ і розкрити дужки;

– з однакових елементарних кон'юнкцій вилучити всі, окрім однієї.

Приклад 4.31. $x_1 \wedge x_2 \vee \overline{x_3} = x_1 \wedge x_2 (x_3 \vee \overline{x_3}) \vee (x_1 \vee \overline{x_1}) \wedge (x_2 \vee \overline{x_2}) \wedge \wedge \overline{x_3} =$ $x_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge \overline{x_3} \vee x_1 \wedge x_2 \wedge \overline{x_3} \vee x_1 \wedge \overline{x_2} \wedge \overline{x_3} \vee \overline{x_1} \wedge x_2 \wedge \overline{x_3} \vee \overline{x_1} \wedge \overline{x_2}$ $\wedge \overline{x_3} = x_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge x_2 \wedge \overline{x_3} \vee x_1 \wedge \overline{x_2} \wedge \overline{x_3} \vee \overline{x_1} \wedge x_2 \wedge \overline{x_3} \vee \vee \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}$ – ДДНФ.

Для того щоб привести формулу до ДКНФ, доцільно спочатку привести її до ДНФ, а потім від ДНФ перейти до КНФ в такий спосіб.

Нехай ДНФ має вигляд $f = c_1 \vee c_2 \vee \ldots \vee c_m$, де c_i – елементарні кон'юнкції, $i = \overline{1, m}$

Формулу $\overline{c_1 \vee c_2 \vee \ldots \vee c_m}$ приведемо до ДНФ $k_1 \vee k_2 \vee \ldots \vee k_l$, де k_i – елементарні кон'юнкції. Тоді $f = \overline{\overline{c_1 \vee c_2 \vee \ldots \vee c_m}} = \overline{k_1 \vee k_2 \vee \ldots \vee k_l} = \overline{k_1} \wedge \overline{k_2} \wedge \ldots \wedge \overline{k_l}$.

Застосовуючи правило де Моргана, перетворимо елементарні кон'юнкції k_i на елементарні диз'юнкції D_i, де $i = \overline{1,k}$. Отже, дістанемо КНФ.

$$f = D_1 \wedge D_2 \wedge \ldots \wedge D_l .$$

І, врешті, використовуючи закон суперечності та другий дистрибутивний закон, зробимо перехід від КНФ до ДКНФ.

Приклад 4.32.

1) $\overline{x_1 x_2 x_3} \wedge (x_1 x_2 \to x_1 \overline{x_2} x_3) = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1 x_2} \vee x_1 \overline{x_2} x_3) ==$
$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_1) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) =$
$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) =$
$=(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3 \overline{x_3}) = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge\wedge$
$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$ - ДКНФ.

2) $x_1 \overline{x_2} \vee \overline{x_1 x_2} = \overline{x_1 \overline{x_2} \vee \overline{x_1} x_2} = \overline{\overline{x_1 x_2} \wedge \overline{\overline{x_1} x_2}} = \overline{(\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2})} =$
$\overline{\overline{x_1 x_2} \vee x_1 x_2} = \overline{\overline{x_1 x_2}} \wedge \overline{x_1 x_2}$ - ДКНФ

Визначення. Елементарна диз'юнкція (кон'юнкція), яка містить усі змінні, називається **конституентою нуля** (**одиниці**). Наприклад, якщо загальна кількість змінних $n = 3$, то $\overline{x_1} \vee x_2 \vee x_3$– конституента нуля, а $x_1 \wedge \overline{x_2} \wedge x_3$– конституента одиниці.

Вочевидь, що конституента нуля перетворюється на нуль лише за одного набору значень змінних. У нашому прикладі конституенті нуля відповідає набір (1, 0, 0). Аналогічно, конституента одиниці перетворюється на одиницю також лише за одного набору. Наприклад, конституенті одиниці $x_1 \wedge \overline{x_2} \wedge x_3$ відповідає набір (1, 0, 1).

Оскільки для заданої булевої функції її ДДНФ являє собою диз'юнкцію конституент одиниці, а її ДКНФ − це кон'юнкція конституент нуля, то дана функція перетворюється на одиницю чи нуль лише за відповідних цим конституентам наборів значень змінних. Справедливе є і зворотне твердження.

Це дозволяє за заданою таблицею істинності булевої функції одразу записати її досконалі нормальні форми і, навпаки, за заданою ДНФ – скласти таблицю істинності.

Досконалі форми для функції $f(x_1, x_2, \ldots, x_n)$ позначають:

для ДДНФ – $f(x_1, x_2, \ldots, x_n) = \bigvee_1 (x_1, x_2, \ldots, x_n)$;

для ДКНФ – $f\ (x_1, x_2, \ldots, x_n) = \underset{0}{\wedge}(x_1, x_2, \ldots, x_n)$, де символ $\underset{1}{\vee}$або $\underset{0}{\wedge}$ позначає, що диз'юнкція або кон'юнкція виконуються за відповідними конституентами.

Приклад 4.33.Побудувати таблицю істинності для вислолювання: $(x \mid \bar{y}) \to (y \oplus z)$**,** побудувати ДДНФ, ДКНФ.

Розв'язок.

Будуємо таблицю істинності – таблицю, за допомогою якої встановлюється істинне значення складного висловлювання при даних значеннях простих висловлювань, що в нього входять.

x	y	z	$\bar{y}$	$x \mid \bar{y}$	$y \oplus z$	
1	1	1	0	1	0	0
1	0	1	1	0	1	1
1	1	0	0	1	1	1
1	0	0	1	0	0	1
0	1	1	0	1	0	0
0	0	1	1	1	1	1
0	1	0	0	1	1	1
0	0	0	1	1	0	0

За таблицею складаємо диз`юнктивну нормальну форму (ДНФ). ДНФ в булевій логіці — нормальна форма, в якій булева формула має вигляд диз`юнкції декількох кон`юнктів.

Алгоритм отримання ДДНФ за таблицею істинності:

1) Відзначаємо ті рядки, в останньому стовпчику яких стоять 1:

2) Виписати для кожного відзначеного рядка кон`юнкцію всіх змінних наступним чином: якщо значення деякої змінної в даному рядку =1, то в кон`юнкцію включають саму цю змінну, якщо =0, то її заперечення:

3) Всі отримані кон`юнкціїзв`язати в диз`юнкцію:

Вибираємо в таблиці рядки, в яких булева функція приймає значення 1. В даному випадку – це 2-ий, 3-ій, 4-ий, 6-ий і 7-ий рядки.

Для кожного рядка складаємо кон`юнкцію: якщо значення змінної дорівнює 0, то беремо її заперечення, а якщо 1, то беремо саму змінну. Потім складаємо диз`юнкцію отриманих кон`юнкцій: $f(x, y, z) = (x \wedge \bar{y} \wedge z) \vee (x \wedge y \wedge \bar{z}) \vee (x \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge \bar{y} \wedge z) \vee (x \wedge \bar{y} \wedge z)$

Вибираємо в таблиці рядки, в яких булева функція набуває значення 0. В даному випадку – це 1-ий, 5-ийі 8-ий рядки:

$f(x, y, z) = (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee z)$.

Приклад 4.34. Перетворити функцію $\{0,3,5\}_{x_1 x_2 x_3}$ на ДДНФ.

Розв'язок. $y = \{(0,0,0), (011), (101)\} x_1 x_2 x_3 = \overline{x_1 x_2 x_3} \vee \overline{x_1} x_2 x_3 \vee x_1 \overline{x_2} x_3$ – ДДНФ.

Приклад. 4.35. Для функції, заданої власною ДКНФ

$y = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$, скласти таблицю істинності.

Розв'язок

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Обов'язкові завдання

Для заданої логічної функції $F = (\overline{\overline{A} \vee B \cdot \overline{C}}) \cdot (\overline{(\overline{B \downarrow C}) \cdot D})$

1.Знайти ДНФ;

2.Скласти таблицю істинності і діаграму Карно (для самостійного вивчення, будуть питання, пишіть);

3.Отримати мінімальну ДНФ;

4.Від мінімальної ДНФ перейти до КНФ.

4.3 Алгебра Жегалкіна та її основні закони

Визначення. Алгеброю Жегалкіна називається множина логічних функцій з операціями кон'юнкція, додавання за модулем 2 і константа 1, тобто алгебра, базисом якої є система функцій $\{1, \wedge, \oplus\}$.

Подамо основні закони цієї алгебри:

1) комутативний:

$$x_1 \wedge x_2 = x_2 \wedge x_1; \qquad x_1 \oplus x_2 = x_2 \oplus x_1;$$

2) асоціативний:

$$x_1 \wedge (x_2 \wedge x_3) = (x_1 \wedge x_2) \wedge x_3; \quad x_1 \oplus (x_2 \oplus x_3) = (x_1 \oplus x_2) \oplus x_3;$$

3) дистрибутивний:

$$x_1 (x_2 \oplus x_3) = x_1 x_2 \oplus x_1 x_3;$$

4) ідемпотентний:

$$x \wedge x = x;$$

5) закон приведення подібних членів:

$$x \oplus x = 0.$$

В алгебрі Жегалкіна мають місце такі властивості:

$$x \oplus 0 = x; \quad x \wedge 0 = 0; \quad x \wedge 1 = x.$$

Решта операцій алгебри логіки виражаються через базис цієї алгебри в такий спосіб:

$$\overline{x} = x \oplus 1; \quad x_1 \vee x_2 = x_1 \oplus x_2 \oplus x_1 \wedge x_2; \quad x_1 \sim x_2 = 1 \oplus x_1 \oplus x_2;$$

$$x_1 \to x_2 = 1 \oplus x_1 \oplus x_1 \wedge x_2; \quad x_1 \leftarrow x_2 = x_1 \oplus x_1 \wedge x_2;$$

$$x_1 \downarrow x_2 = 1 \oplus x_1 \oplus x_2 \oplus x_1 \wedge x_2; \quad x_1 \mid x_2 = 1 \oplus x_1 \wedge x_2.$$

Визначення. Функція алгебри Жегалкіна, подана у вигляді суми за модулем 2 добутків незалежних змінних, називається канонічним многочленом, або **поліномом Жегалкіна**.

Поліномом Жегалкіна називається скінченна сума за модулем 2 попарно різних елементарних кон'юнкцій над множиною змінних $\{x_1, x_2, ..., x_n\}$. В зарубіжній літературі представлення полінома Жегалкіна зазвичай називається алгебраїчною нормальною формою (АНФ).

Теорема Жегалкіна — стверджує існування і унікальність будь-якої булевої функції у вигляді поліному Жегалкіна. Формально поліном Жегалкіна можна представити у вигляді:

$$P(x_1, ...x_n) = a \oplus a_1 \wedge x_1 \oplus a_2 \wedge x_2 \oplus ... \oplus a_n x_n \oplus a_{12} \wedge x_1 \wedge x_2 \oplus a_{13} \wedge x_1 \wedge x_3 \oplus .... \oplus a_{1..n} \wedge x_1 ... \wedge x_n$$

Кількість змінних, що входять до елементарної кон'юнкції називається **рангом елементарної кон'юнкції**. Кількість попарно різних елементарних кон'юнкцій у поліномі називається **довжиною полінома**.

Функція $f(x_1, x_2, x_3)$ від трьох змінних буде мати наступний поліном Жегалкіна:

$$f(x_1, x_2, x_3) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus a_{23} x_2 x_3 \oplus a_{123} x_1 x_2 x_3.$$

4.3.1. Способи побудови полінома Жегалкіна

4.3.1.1. Метод еквівалентних перетворень

Для функції $f(x_1, x_2, x_3)$ записують ДДНФ, потім виражають диз'юнкцію та заперечення через операції кон'юнкції та суми за модулем 2. Використовуючи тотожності алгебри Жегалкіна, отримують поліном.

Приклад 4.36. Зобразити поліном Жегалкіна для функції $x \sim y$.

Розв'язок. $x \sim y = xy \vee \bar{x}\bar{y} = xy\bar{x}\bar{y} \oplus xy \oplus \bar{x}\bar{y} = xy(\bar{x}\bar{y} \oplus 1) \oplus \bar{x}\bar{y} = xy \oplus (x \oplus 1)(y \oplus 1) =$

$= xy \oplus x \oplus x \oplus y \oplus 1 = x \oplus y \oplus 1$

Приклад 4.37. Зобразити поліном Жегалкіна для функції

$f(x_1, x_2, x_3) = (x_1 x_2 \vee x_3) \to \bar{x_2}$

Застосовуючи основні закони алгебри логіки, приведемо спочатку дану функцію до ДНФ: $f(x_1, x_2, x_3) = (x_1 x_2 \vee x_3) \to \bar{x_2} =$ (застосуємо рівнозначність $x \to y = \bar{x} \vee y) = \overline{(x_1 x_2 \vee x_3)} \vee \bar{x_2} =$ (застосуємо закон де Моргана)= $\overline{x_1 x_2} \wedge \bar{x_3} \vee \bar{x_2} =$(застосуємо ще раз закон де Моргана)=$(\bar{x_1} \vee \bar{x_2}) \wedge \bar{x_3} \vee \bar{x_2} =$ (застосуємо закон дистрибутивності)= $\bar{x_1} \wedge \bar{x_3} \vee \bar{x_2} \wedge \bar{x_3} \vee \bar{x_2} =$(застосуємо закон поглинання $\bar{x_2} \wedge \bar{x_3} \vee \bar{x_2} = \bar{x_2}$)= $\bar{x_1}\bar{x_3} \vee \bar{x_2}$ - ДНФ.

Далі в отриманій ДНФ необхідно позбутися диз`юнкцій, використовуючи закони де Моргана:

$\bar{x_1}\bar{x_3} \vee \bar{x_2} = \bar{x_1} \wedge \bar{x_3} \vee \bar{x_2} = \overline{x_1 \vee x_3} \vee \bar{x_2} = \overline{\overline{x_1 \vee x_3} \vee x_2} = \overline{\overline{x_1} \wedge \overline{x_3} \vee x_2} = \overline{\overline{x_1 x_3 x_2}}$.

Замінюємо кожне заперечення $\bar{x} = 1 \oplus x$ і застосовуємо згадані вище закони, отримаємо:

$\overline{\overline{x_1 x_3 x_2}} = 1 \oplus \overline{\overline{x_1 x_3} x_2} = 1 \oplus (1 \oplus (1 \oplus x_1)(1 \oplus x_3) x_2 = 1 \oplus (1 \oplus 1 \oplus x_3 \oplus x_1 \oplus x_{13}) x_2 = 1 \oplus$

$\oplus (x_3 \oplus x_1 \oplus x_{13}) x_2 = 1 \oplus x_2 x_3 \oplus x_1 x_2 \oplus x_1 x_2 x_3$

Перетворення ДДНФ.

x_1	x_2	x_3	$x_1 x_2$	$x_1 x_2 \vee x_3$	$\bar{x_2}$	f
0	0	0	0	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	0	1
0	1	1	0	1	0	0
1	0	0	0	0	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	1	1	0	0

Для побудови ДДНФ за таблицею істинності обираємо набори, на яких функція f приймає значення, що дорівнює 1. Якщо значення змінної в цьому наборі дорівнює 0, то вона береться із запереченням, якщо значення змінної дорівнює 1, то змінна береться без заперечення. З`єднавши знаком кон`юнкції всі змінні відповідного набору, отримаємо елементарну кон`юнкцію. Тоді диз`юнкція всіх таких елементарних кон`юнкцій і є ДДНФ.

$f(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}\,\overline{x_3}\vee\overline{x_1}\,\overline{x_2}x_3\vee\overline{x_1}x_2\overline{x_3}\vee x_1\overline{x_2}\,\overline{x_3}\vee x_1\overline{x_2}x_3$. Щоб побудувати поліном Жегалкіна через ДДНФ, необхідно виключити операції диз`юнкції і заперечення, потім розкрити дужки.

$$f(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}\,\overline{x_3}\vee\overline{x_1}\,\overline{x_2}x_3\vee\overline{x_1}x_2\overline{x_3}\vee x_1\overline{x_2}\,\overline{x_3}\vee x_1\overline{x_2}x_3=(1\oplus x_1)(1\oplus x_2)(1\oplus x_3)\oplus(1\oplus x_1)$$
$$(1\oplus x_2)x_3\oplus(1\oplus x_1)(1\oplus x_3)x_2\oplus x_1x_3(1\oplus x_2)=(1\oplus x_2\oplus x_1\oplus x_1x_2)(1\oplus x_3)\oplus(1\oplus x_2\oplus x_1\oplus$$
$$\oplus x_1x_2)x_3\oplus(1\oplus x_3\oplus x_1\oplus x_1x_3)x_2\oplus(1\oplus x_3\oplus x_2\oplus x_2x_3)x_1\oplus x_1x_3\oplus x_1x_2x_3=(1\oplus x_3\oplus x_2\oplus$$
$$\oplus x_2x_3\oplus x_1\oplus x_1x_3\oplus x_1x_2\oplus x_1x_2x_3)\oplus(x_3\oplus x_2x_3\oplus x_1x_2x_3)\oplus(x_2\oplus x_2x_3\oplus x_1x_2\oplus x_1x_2x_3)\oplus$$
$$\oplus(x_1\oplus x_1x_3\oplus x_1x_2\oplus x_1x_2x_3)\oplus x_1x_3\oplus x_1x_2x_3=1\oplus x_1x_2\oplus x_2x_3\oplus x_1x_2x_3$$

- Поліном Жегалкіна.

4.3.1.2. Метод трикутника Паскаля.

Розглянемо метод на прикладі

Приклад 4.38. Побудувати поліном Жегалкіна для функції *f*. Візьмемо функцію голосування $f(x_1,x_2,x_3)=x_1x_2\vee x_2x_3\vee x_1x_3=(00010111)$.

Крок 1. Будуємо таблицю значень фнкції (рядки таблиці йдуть в порядку збільшення двоїстих кодів). Таблицю краще розташовувати в лівій частині сторінки.

x_1	x_2	x_3	f		
0	0	0	0	1	**0** 0 0 1 0 1 1 1
0	0	1	0	x_3	**0** 0 1 1 1 0 0
0	1	0	0	x_2	**0** 1 0 0 1 0
0	1	1	1	$x_2x_3\Rightarrow$	**1** 1 0 1 1
1	0	0	0	x_1	**0** 1 1 0
1	0	1	1	$x_1x_3\Rightarrow$	**1** 0 1
1	1	0	1	$x_1x_2\Rightarrow$	**1** 1
1	1	1	1	$x_1x_2x_3\Rightarrow$	**0**

Крок 2.Побудова трикутника. Для цього беремо вектор значень функції і виписуємо його навпроти першого рядка таблиці. Далі заповнюємо трикутник, складаючи попарно сусідні значення за модулем 2. Результат додавання записуємо нижче.

Крок 3. Побудова полінома Жегалкіна. Нас цікавить ліва сторона трикутника (значення виділені жирним). Числа на лівій стороні (виділені жирним шрифтом) трикутника є коефіцієнтами поліному при монотонних кон`юнкціях, які відповідають наборам значень змінних. Тепер випишемо для наочності ці кон`юнкції. Кон`.нкції виписуємо за двоїстими наборами в лівій частині таблиці за наступним принципом: якщо навпроти змінної $x_i$ стоїть 1, то змінна входить в кон`юнкцію, в протилежному випадку – змінна відсутня в кон`юнкції. Набору (0,0,0) відповідає 1.

Якщо принцип отримання кон`юнкцій зрозумілий, то стовпчик з ним (краще) не виписувати, а доразу переходити о побудови поліному. Для побудуви поліному необхідні кон`юнкції тільки з рядків з одиницямина лівій стороні трикутника. Це і є кон`юнкції, що входять до складу полінома Жегалкіна. Залишилося виписати сам поліном:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_1x_3 .$$

Якщо змінних в функції не 3, а 4 і більше, то метод працює без змін, тільки змінюється розмір таблиці. Тим не менш, на відміну від методу невизначених коефіцієнтів, розрахунки можна виконувати без зайвих зусиль.

Приклад 4.39. Знайдемо поліном Жегалкіна функції $f(x_1, x_2, x_3) = (x_1x_2 \vee x_3) \to \overline{x_2}$, застосовуючи трикутник Паскаля.

x_1	x_2	x_3	x_1x_2	$x_1x_2 \vee x_3$	$\overline{x_2}$	f		
0	0	0	0	0	1	1	1	1 1 1 0 1 1 0 0
0	0	1	0	1	1	1	x_3	0 0 1 1 0 1 0
0	1	0	0	0	0	1	x_2	0 1 0 1 1 1
0	1	1	0	1	0	0	$x_2x_3 \Rightarrow$	1 1 1 0 0
1	0	0	0	0	1	1	x_1	0 0 1 0
1	0	1	0	1	1	1	x_1x_3	0 1 1
1	1	0	1	1	0	0	$x_1x_2 \Rightarrow$	1 0
1	1	1	1	1	0	0	$x_1x_2x_3 \Rightarrow$	1

Поясню ще один раз як заповнюється трикутник Паскаля. Верхній рядок трикутника задає вектор значень мулевої функції $f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3 \vee x_1x_3 = (00010111)$. В кожному рядку, починаючи з другого, будь-який елемент такого трикутника обчислюється як сума за модулем 2 двох сусідніх елементів попереднього рядка.

Лівій стороні трикутника Паскаля відповідають набори значень змінних вхідної функції $f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3 \vee x_1x_3 = (00010111)$. З`єднаючи знаком

кон`юнкції змінні, значення яких в наборі дорівнюють 1, ми отримаємо доданок в поліномі Жегалкіна. Набору (0,0,0) відповідає x_3 і так далі. Оскільки одиницям лівої сторони трикутника відповідають доданки 1, $x_1x_2; x_2x_3; x_1x_2x_3$, то поліном Жегалкіна: $f(x_1, x_2, x_3) = 1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_2x_3$.

4.3.1.3. Метод невизначених коефіцієнтів

Нехай *P(x)* поліном Жегалкіна, який реалізує задану функцію.

Для двох змінних:

$$P(x_1, x_2) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_{12}x_1x_2;$$

Для трьох змінни

$$P(x_1, x_2, x_3) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus a_{12}x_1x_2 \oplus a_{13}x_1x_3 \oplus a_{23}x_2x_3 \oplus a_{123}x_1x_2x_3.$$

Коефіцієнти $a_0, a_1, a_2, a_3 \ldots$ - це невизначені коефіцієнти, які необхідно знайти.

Кожну змінну $x_1, x_2, x_3 \ldots$ розглянемо двійковим набором $\sigma_1, \sigma_2, \sigma_3 \ldots$ (наприклад, $x_1 = 001,\ x_2 = 101$).

Для кожного двійкового набору значень змінних записують 2^n рівнянь $f(\sigma_1,\ldots,\sigma_n) = P(\sigma_1,\ldots,\sigma_n)$. Розв'язавши їх, отримують коефіцієнти полінома.

Приклад 4.40. Знайдемо поліном Жегалкіна для функції $f(x_1, x_2, x_3) = (x_1x_2 \vee x_3) \to \overline{x_2}$, використовуючи метод невизначених коефіцієнтів. Для цього спочатку необхідно побудувати таблицю істинності для булевої функції $f(x_1, x_2, x_3) = (x_1x_2 \vee x_3) \to \overline{x_2}$.

x_1	x_2	x_3	x_1x_2	$x_1x_2 \vee x_3$	$\overline{x_2}$	f
0	0	0	0	0	1	1
0	0	1	0	1	1	1
0	1	0	0	0	0	1
0	1	1	0	1	0	0
1	0	0	0	0	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	1	1	0	0

Загальний вигляд поліному Жегалкіна для функції трьох змінних:

$$f(x_1, x_2, x_3) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus a_{12}x_1x_2 \oplus a_{13}x_1x_3 \oplus a_{23}x_2x_3 \oplus a_{123}x_1x_2x_3.$$

Послідовно представимо набори значень змінних і знаходимо коефіцієнти $a_0, a_1, \ldots a_{123}$.

$f(0,0,0) = a_0 = 1;$

$f(0,0,1) = a_0 \oplus a_3 = 1 \oplus a_3 = 1 \Rightarrow a_3 = 0;$

$f(0,1,0) = a_0 \oplus a_2 = 1 \oplus a_2 = 1 \Rightarrow a_2 = 0;$

$f(0,1,1) = a_0 \oplus a_2 \oplus a_3 \oplus a_{23} = 1 \oplus 0 \oplus 0 \oplus a_{23} = 1 \oplus a_{23} = 0 \Rightarrow a_{23} = 1;$

$f(1,0,0) = a_0 \oplus a_1 = 1 \oplus a_1 = 1 \Rightarrow a_1 = 0;$

$f(1,0,1) = a_0 \oplus a_1 \oplus a_3 \oplus a_{13} = 1 \oplus 0 \oplus 0 \oplus a_{13} = 1 \oplus a_{13} = 1 \Rightarrow a_{13} = 0;$

$f(1,1,0) = a_0 \oplus a_1 \oplus a_2 \oplus a_{12} = 1 \oplus 0 \oplus 0 \oplus a_{12} = 1 \oplus a_{12} = 0 \Rightarrow a_{12} = 1;$

$f(1,1,1) = a_0 + a_1 \oplus a_2 \oplus a_3 \oplus a_{12} \oplus a_{13} \oplus a_{23} \oplus a_{123} = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus a_{123} = 1 \oplus a_{123} = 0$
$\Rightarrow a_{123} = 1$

Підставляючи отримані коефіцієнти, отримаємо поліном Жегалкіна:

$$f(x_1, x_2, x_3) = 1 \oplus x_1 x_2 \oplus x_2 x_3 \oplus x_1 x_2 x_3.$$

Обов'язкові завдання

1. Перевірити на повноту систему булевих функцій $\{x_1 \vee x_2, x_1 \to x_2\}$.
2. Перевірити, чи є повною система $J = \{x \to \overline{y}, \overline{x} \wedge y\}$.
3. Скласти поліном Жегалкіна методом невизначених коефіцієнтів і за допомогою трикутника Паскаля для $F = \overline{A}(A \wedge B) \to (A \wedge B \to B)$.

4.4 Мінімізація булевих функцій

Булеві функції, як відомо, можуть бути реалізовані різними формулами, проте для практики найбільше значення мають, так звані мінімальні нормальні форми, у яких число входжень символів змінних найменше.

Для довільних функцій методів знаходження таких форм не існує, мінімізацію проводять лише для диз'юнктивних нормальних форм.

Задачі знаходження (побудови) мінімальних ДНФ називають задачами мінімізації.

Змінні $x_i\left(i = \overline{1,n}\right)$ та $\overline{x_i}\left(i = \overline{1,n}\right)$ досить часто називають **термами**. Повний набір із термів утворює **конституенту**. У процесі мінімізації деякі терми із конституант зникають. Ту частину, яка залишилась, називають **імплікантою**.

Булева функція $g\left(x_1, x_2, ..., x_n\right)$ називається **імплікантою** функції $f\left(x_1, x_2, ..., x_n\right)$, якщо вона перетворюється в одиницю при наборі змінних, на якому сама функція також дорівнює одиниці, тобто, якщо *g*=1, то й *f*=1.

Кожна конституанта одиниці, яка входить до складу ДДНФ, або їхня дих'юнкція є імплікантою певної булевої функції.

Елементарна кон'юнкція $k = x\overline{y}\overline{z}$ - імпліканта функції $f = x\overline{y}\overline{z} \vee \overline{x}\overline{y}\overline{z}$, бо в разі *k*=1значення функції f дорівнює 1.

Імпліканта g називається простою, якщо жодна її частина не може бути імплікантою функції *f*.

Приклад 4.41. Для функції $f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1x_2\overline{x_3} \vee \overline{x_1}x_2x_3$ знайти всі імпліканти.

$$g_1 = x_1x_2x_3;$$

$$g_2 = x_1x_2\overline{x_3};$$

$$g_3 = \overline{x_1}x_2x_3;$$

$$g_4 = x_1x_2\overline{x_3} \vee x_1x_2x_3 = x_1x_2;$$

$$g_5 = \overline{x_1}x_2x_3 \vee x_1x_2x_3 = x_2x_3;$$

$$g_6 = x_1x_2\overline{x_3} \vee \overline{x_1}x_2x_3;$$

$$g_7 = x_1x_2x_3 \vee x_1x_2\overline{x_3} \vee \overline{x_1}x_2x_3 = f$$

x1	x2	x3	f	g1	g2	g3	g4	g5	g6	g7
0	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	0	0	0	0	0	0	0
0	1	1	1	0	0	1	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	1	0	1	1
1	1	1	1	1	0	0	1	1	0	1

ДНФ, що складається з усіх простих імплікант булевої функції, називають її **скороченою диз'юнктивною нормальною формою** (СДНФ).

Скорочена ДНФ (англ. *reduced disjunctive normal form*) — форма запису функції, що має наступні властивості:

1) будь-які два доданки розрізняються мінімум як в двох позиціях,

2) жоден кон`юнкт не міститься в іншому.

За основу розв`язання задач мінімізації довільних логічних функцій приймаємо схему, що містить три етапи:

1) На першому етапі складається таблиця істинності функції $f(x_1, x_2, \ldots, x_n)$;

2) На другому етапі виконується пошук простих імплікант *f*, тобто будується **скорочена** досконала нормальна форма.

Наприклад, (*x*∧*y*) міститься в (*x*∧*y*∧*z*). Функцію можна записати за допомогою скороченої ДНФ не єдиним способом. Запишемо функцію (x, y, z) у вигляді доскональної ДНФ: (*x*∧*y*∧*z*) ∨ (*x*∧*y*∧¬*z*) ∨ (*x*∧¬*y*∧*z*) ∨ (¬*x*∧*y*∧*z*). Відомо, що цей вираз еквівалентний наступному: ((*x*∧*y*∧*z*) ∨ (*x*∧*y*∧¬*z*)) ∨

((x∧¬y∧z) ∨ (x∧y∧z)) ∨ ((¬x∧y∧z) ∨ (x∧y∧z)). Винесемо в кожній дужці спільний кон`юнкт (наприклад, в першій (x∧y∧z) ∨ (x∧y∧¬z) = (x∧y) ∨ (z∧¬z). Так як z∧¬z=0, то такий кон`юнкт не впливає на значення виразу, і його можна опустити. Отримаємо в підсумку формулу (x∧y) ∨ (y∧z) ∨ (x∧z);

3) на третьому етапі проводиться побудова **тупикових** (без зайвих імплікант) досконалих нормальних форм, з числа яких вибирають **мінімальні** досконалі нормальні форми.

ДНФ називають **тупиковою** ДНФ функції, якщо:

а) кожна елементарна кон'юнкція з ТДНФ – проста імпліканта *f*;

б) вилучення з ТДНФ довільного диз'юнктивного члена призводить до ДНФ, яка не задає функцію *f;*

Мінімальна ДНФ булевої функції являє собою її тупикову ДНФ

Існують тупикові, але не мінімальні ДНФ, одна й та сама булева функція f може мати декілька різних мінімальних ДНФ.

Приклад 4.42. Нехай скорочена ДНФ функції має вигляд: $f(x,y,z) = xy \vee \overline{y}z \vee xz$

Тоді її одинична множина може бути представлена у вигляді: $M_f = M_{xy} \vee M_{\overline{y}z} \vee M_{xz} = \{110,111\} \vee \{001,101\} \vee \{101,111\}$. Зазначимо, що набори, що входять до останньої підмножини, знаходяться також і в першому і в другому. Так $101 \in M_{\overline{y}z}$

(говорять, ***набір 101 покривається множиною*** $M_{\overline{y}z}$), а $111 \in M_{xy}$. Значить, якщо прибрати складову M_{xz}, функція від цього не зміниться. Говорять, що ***множина покривається об`єднанням*** $M_{\overline{y}z}$ ***і*** M_{xy}. Отже, ***імпліканта xz – зайва.***

Для розв`язання задач мінімізації на другому етапі застосовують методи законів та тотожностей алгебри логіки, метод Куайна, метод Мак-Класкі, метод Карно-Вейча, метод Куайна-Мак-Класкі і інші. Основним методом для проведення третього етапу є метод імплікантної таблиці логічних функцій (метод Куайна).

Розглянемо дані методи.

4.4.1. Метод послідовного застосування законів і тотожностей алгебри логіки

В основі даного методу лежить пошук виразів, які можна записати у більш простому вигляді. При цьому, як правило, найбільше використовуються такі операції та закони логіки:

$x_1\overline{x_2} \vee x_1x_2 = x_1(\overline{x_2} \vee x_2) = x_1$- операція склеювання;

$x_1 \vee x_xx_2 = x_1(1 \vee x_2) = x_1$- операція поглинання;

$x_1 \vee \overline{x_1}x_2 = (x_1 \vee \overline{x_1})(x_1 \vee x_2) = x_1 \vee x_2$- дистрибутивний закон;

$x \vee x = x$ - ідемпотентність, з цього закону випливає, що кожний доданок в ДНФ можна групувати з іншими не одноразово.

Методом користуються лише в досить простих випадках, він носить елементи довільних розв`язків і є дуже громіздким. Розглянемо приклад.

Приклад 4.43. Мінімізувати булеву функцію

$$\boldsymbol{f(x_1, x_2, x_3, x_4) = \overline{x_1}\overline{(x_2x_3 \vee x_4)(x_1x_2x_3 \vee x_4)}(\overline{x_1}x_2x_3x_4 \vee \overline{x_1}x_2x_3 \vee \overline{x_1}).}$$

Розв'язок. Скористаємося законом де Моргана та дистрибутивним законом.

$$f = \overline{x_1}\overline{((x_2x_1 \vee x_4)} \vee \overline{(x_1x_2x_3 \vee x_4))}(\overline{x_1}x_2x_3x_4 \vee \overline{x_1}x_2x_3 \vee \overline{x_1}) =$$
$$= \overline{x_1}\big((x_2x_1 \vee x_4) \vee (x_1x_2x_3 \vee x_4)\big) \cdot$$
$$\cdot (x_2x_3x_4 \vee x_2x_3 \vee 1) = \overline{x_1}(x_2x_1 \vee x_4 \vee x_1x_2x_3 \vee x_4)(x_2x_3x_4 \vee x_2x_3 \vee 1)$$

Застосуємо дистрибутивний закон та закон поглинання.

$$f = \overline{x_1}(x_2x_3(1 \vee x_1) \vee x_4) = \overline{x_1}(x_2x_3 \vee x_4).$$

4.4.2. Метод Карно-Вейча.

Якщо число змінних логічних функцій мале $(n \le 4)$, знаходження мінімальних форм можна проводити за допомогою спеціальних таблиць, які називаються діаграмами Вейча або картами Карно. Нехай *n*=4, тобто $f = f(x_1, x_2, x_3, x_4)$. Карта Карно для чотирьох змінних являє собою квадрат, що розбитий на 16 малих квадратів (4x4). Складемо карту Карно. Для змінних x_1 і x_2 відведемо вертикальну сторону карти, а для x_3 і x_4, - горизонтальну.

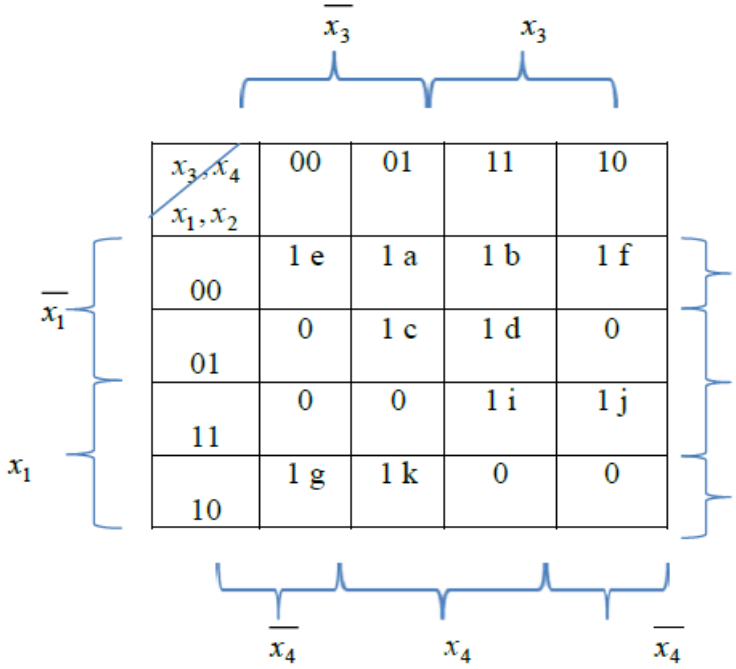
Сенс карти в тому, що функція задається таблицею, але не в стовпчиках, як завжди, а на площині у вигляді 16 квадратів. Набори змінних використовуються в порядку, так званого коду Грея (00), (01), (11), (10). На карті Карно сусідні набори відмінні лише однією координатою від сусіднього по розміщенню. Значення функції записують в малих квадратах. По таблиці Карно з`ясовують, які квадрати карти можна закріпити тією чи іншою імплікантою, якщо два сусідні рядки, чи два сусідні стовпчики заповнені одиницями, то їх можна покрити однотермовою імплікантою, тобто якби в перших двох рядках всюди були одиниці, то: $f = \overline{x_1} \vee \phi(x_1, x_2, x_3, x_4)$.

Зауважимо, що карту Карно треба уявляти так, що вона відтворена не на площині, а на поверхні, що має форму Тора, в якому сусідніми будуть перший та останній рядки, перший та останній стовпчики. Тобто можна стверджувати, що однобуквеним імплікантам відповідають або два сусідні рядки або два сусідні стовпчики.

Двотермові імпліканти розглядають так: $x_1x_2 = 1$відповідає третій рядок. Аналогічно знаходяться три термові і чотири термові імпліканти. Тотбо за допомогою карти Карно графічно виконуються операції склеювання,

поглинання, об`єднання одиниць та груп одиниць між собою. Продемонструємо метод Карно-Вейча на прикладі

Приклад 4.44. Нехай функція $f(x_1, x_2, x_3, x_4)$задана таблицею



Розв`язок. Проаналізуємо дану таблицю істинності і позначимо квадрати, що вміщують одиниці літерами.

З таблиці видно, що однотермові імпліканти відсутні, бо не має двох сусідніх стовпчиків або рядків, що вміщують лише одиниці.

Будемо шукати двотермові імпліканти, тобто чотири квадрати з одниницями, що витягнуті в одну лінію або складені у великий квадрат. Бачимо, що перший рядок (квадрати *e, a, b, f*) – утворює лінію, що покривається $\overline{x_1}$ та $\overline{x_2}$, отже отримано імпліканту $\overline{x_1 x_2}$.

Квадрат *abcd*, що складається з одиниць повністю покривається імплікантою $\overline{x_1} x_4$.

Квадрат *eagk* покривається імплікантою $\overline{x_2 x_3}$ (квадрат утворено замкненням карти до утворення тору). Більше квадратів утворити не можна, тобто

$$f(x_1, x_2, x_3, x_4) = \overline{x_1 x_2} \vee \overline{x_2 x_3} \vee \overline{x_1} x_4 \vee \phi(x_1, x_2, x_3, x_4).$$

В таблиці непокритими залишилися дві одиниці (квадрати *i* та *j*). Покрити їх неможливо, тобто необхідно витратити на них тритермову імпліканту $x_1 x_2 \ \ x_3$.

Всі одиниці покриті імплікантами, за методом Карно-Вейча МДНФ має вигляд:

$$f(x_1, x_2, x_3, x_4) = \overline{x_1 x_2} \vee \overline{x_2 x_3} \vee \overline{x_1} x_4 \vee x_1 x_2 x_3.$$

Приклад 4.45. Знайти мінімальну ДНФ для функції:

$$\boldsymbol{f(x_1, x_2, x_3) = x_1 x_2 \overline{x_3} \vee x_1 \overline{x_2 x_3} \vee \overline{x_1} x_2 x_3 \vee \overline{x_1 x_2 x_3}.}$$

Розв`язок.

x2x3 x1	00	01	11	10
0	1		1	
1	1			1

Мінімальна ДНФ: $f_{\min}(x_1, x_2, x_3) = x_1\overline{x_3} \vee \overline{x_2}\,\overline{x_3} \vee \overline{x_1}x_2x_3$.

Приклад 4.46. Знайти мінімальну ДНФ для функції:

$$f(x_1, x_2, x_3) = x_1\overline{x_2x_3} \vee x_1\overline{x_2}x_3 \vee \overline{x_1}x_2x_3 \vee \overline{x_1x_2}x_3 \vee \overline{x_1x_2x_3}$$

x2x3 x1	00	01	11	10
0	1	1	1	
1	1	1		

Мінімальна ДНФ: $f(x_1, x_2, x_3) = \overline{x_2} \vee \overline{x_1}x_3$.

Приклад 4.47. Знайти мінімальну ДНФ для функції: $f(x_1, x_2, x_3) == x_1x_2x_3 \vee x_1x_2\overline{x_3} \vee x_1\overline{x_2}x_3 \vee x_1\overline{x_2x_3} \vee \overline{x_1}x_2x_3 \vee \overline{x_1x_2}x_3 \vee \overline{x_1x_2x_3}$

x2x3 x1	00	01	11	10
0	1	1	1	
1	1	1	1	1

Мінімальна ДНФ: $f(x_1, x_2, x_3) = \overline{x_2} \vee x_3 \vee x_1$.

Приклад 4.48. Знайти мінімальну ДНФ для функції: $f(x_1, x_2, x_3, x_4) = x_1x_2x_3\overline{x_4} \vee x_1x_2\overline{x_3x_4} \vee x_1\overline{x_2}x_3x_4 \vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1\overline{x_2x_3x_4} \vee\vee \overline{x_1}x_2x_3x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3x_4} \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1x_2}x_3\overline{x_4} \vee \overline{x_1x_2x_3x_4}$

x3x4 x1x2	00	01	11	10
00	1			1
01	1	1	1	1
11	1			1
10	1		1	1

Мінімальна ДНФ: $f(x_1, x_2, x_3) = \overline{x_4} \vee \overline{x_1}x_2 \vee x_1\overline{x_2}x_3$.

4.4.3.Метод Куайна.

За методом Куайна прості імпліканти знаходяться по доскональній диз`юнктивній нормальній формі (ДДНФ) булевої функції в результаті застосування до неї закону неповного склеювання та операції поглинання. Продемонструємо дію методу на прикладі.

Приклад 4. 49. Мінімізувати булеву функцію, що задана таблицею.

a	b	c	d	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Запишемо ДДНФ: Застосуємо операцію склеювання конституент:

~~$\bar{a}\bar{b}\bar{c}\bar{d}$~~ ~~$\bar{a}\bar{b}\bar{d}$~~ $\bar{a}\bar{d}$

~~$\bar{a}\bar{b}c\bar{d}$~~ ~~$\bar{a}c\bar{d}$~~ ~~$\bar{a}\bar{d}$~~

~~$\bar{a}b\bar{c}\bar{d}$~~ ~~$ac\bar{d}$~~

~~$\bar{a}b\bar{c}\bar{d}$~~ $Ax \cdot A\bar{x} = A$ ~~$\bar{a}b\bar{d}$~~

~~$\bar{a}bc\bar{d}$~~ $b\bar{c}\bar{d}$

$a\bar{b}\bar{c}\bar{d}$ $\bar{a}bc$

~~$\bar{a}b\bar{c}\bar{d}$~~ bcd

~~$abc\bar{d}$~~

$\bar{a}\bar{d} \vee b\bar{c}\bar{d} \vee \bar{a}bc \vee bcd \vee a\bar{b}\bar{c}\bar{d}$ - СДНФ

Склеюємо попарно ті конституенти, що відрізняються одним термом. В даному випадку: 1 і 2 конституенти, потім 1 і 3, потім 2 і 4 і т.д. Після першого етапу склеювання повинні залишитися імпліканти, що містять вже 3 терми, замість 4. На другому етапі знову проводимо попарно операцію склеювання так, щоб в імплікантах залишилося по 2 терми.

На четвертому етапі застосуємо операцію поглинання: $BC \vee C = C$. Тобто, викреслюємо всі імпліканти, що містять $\bar{a}\bar{d}$. Записавши імпліканти, що залишилися не викресленими, отримаємо скорочену диз`юнктивну нормальну форму. СДНФ не є останньою степінню спрощення.

Складаємо таблицю Куайна, в якій помістимо отримані спрощенням імпліканти та вихідні конституенти. В стовпчиках записуємо конституенти, в рядках – отримані імпліканти.

Одиницю ставимо там, де імпліканта «покриває» конституенту, це тому, що конституента може бути замінена імплікантою за законом поглинання.

	$\bar{a}\bar{b}\bar{c}\bar{d}$	$\bar{a}\bar{b}c\bar{d}$	$\bar{a}b\bar{c}\bar{d}$	$\bar{a}bc\bar{d}$	$\bar{a}bcd$	$a\bar{b}\bar{c}d$	$ab\bar{c}\bar{d}$	$abcd$
$a\bar{b}\bar{c}d$						1		
$b\bar{c}\bar{d}$			1				1	
~~$\bar{a}bc$~~				~~1~~	~~1~~			
bcd					1			1
$\bar{a}\bar{d}$	1	1	1	1				

Шукаємо ті стовпчики, де є тільки одна одиниця. Це стовпчики 1,2, 6,7 і 8. Ці стовпчики відповідають імплікантам $a\bar{b}\bar{c}d$, $b\bar{c}\bar{d}$, bcd і $\bar{a}\bar{d}$. Ці імліканти складають **ядро** і їх вилучати в жодному разі не можна. Тепер подивимось, чи можемо ми виключити імпліканту. Одиниця, що стоїть в четвертому стовпчику навпроти імпліканти $\bar{a}bc$ покривається одиницею, що стоїть в тому ж стовпчику навпроти імпліканти $\bar{a}\bar{d}$, а одиницю, що стоїть в п`ятому стовпчику навпроти $\bar{a}bc$ покриває одиниця, що стоіть в тому ж стовпчику навпроти bcd. Отже обидві ці одиниці покриваються іншими одиницями. Робимо висновок, імпліканту $\bar{a}bc$ можна виключити. Отже, мінімальна ДНФ має наступний вигляд: МДНФ $=a\bar{b}\bar{c}d \vee b\bar{c}\bar{d} \vee bcd \vee \bar{a}\bar{d}$. Ця МДНФ складається лише з імплікант ядра, але це не завжди так. Бувають випадки, коли спочатку з таблиці Куайна отримуємо **тупикові** ДНФ, з яких потім обираємо мінімальну ДНФ. Тупикова ДНФ складається з ядра і деяких імплікант.

Приклад 4.50. Мінімізувати булеву функцію

$$f(x_1,x_2,x_3,x_4)= x_1x_2x_3\bar{x}_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2x_3x_4 \vee$$
$$\vee \bar{x}_1x_2x_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_4\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2x_3x_4 .$$

Застосуємо закон склеювання конституент

1) $x_1x_2x_3\bar{x}_4$ - 1110 — 1) та 2) 1110 $\vee$ 1100 = 11_0

2) $x_1x_2\bar{x}_3\bar{x}_4$ - 1100 — 1) та 4) 1110 $\vee$ 1010 = 1_101

1) та 7) 1110 $\vee$ 0110 = _110

3) $x_1\bar{x}_2x_3x_4$ - 1011 — 2) та 8) 1100 $\vee$ 0100 = _100

4) $x_1\bar{x}_2x_3\bar{x}_4$ - 1010 — 2) та 5) 1100 $\vee$ 1000 =1_00

3) та 4) 1011 $\vee$ 1010 = 101_

5) $x_1\bar{x}_2\bar{x}_3\bar{x}_4$ - 1000 — 4) та 5) 1010 $\vee$ 1000 =10_0

6) $\overline{x_1}x_2x_3x_4$ - 0111

7) $\overline{x_1}x_2x_3\overline{x_4}$ - 0110

8) $\overline{x_1}x_2\overline{x_3}\overline{x_4}$ - 0100

9) $\overline{x_1}x_2\overline{x_3}x_4$ - 0101

10) $\overline{x_1}x_2\overline{x_3}\overline{x_4}$ - 0100

11) $\overline{x_1}x_2\overline{x_3}\overline{x_4}$ - 0100

4) та 10) 1010 $\vee$0010 = _010

5) та 11) 1000$\vee$0000 =_000

6) та 7) 0111 $\vee$0110 = 011_

6) та 9) 0111 $\vee$ 0101 =01_1

7) та 8) 0110 $\vee$ 0100 = 01_0

7) та 10) 0110 $\vee$0010 =0_10

8) та 9) 0100 $\vee$ 0101 = 010_

8) та 11) 0100 $\vee$0000 =0_00

10) та 11)0010 $\vee$0000 = 00_0

12) 11_0
13) 1_10
14) _110
15) 1_00
16) _100
17) 101_
18) 10_0
19) _010
20) _000
21) 011_
22) 01_1
23) 01_0
24) 0_10
25) 010_
26) 0_00
27) 00_0

Застосуємо закон склеювання імплікант, враховуючи позицію прочерка

12) та 18) 11_0 $\vee$ 10_0 = 1__0

13) та 16) 1_10 $\vee$ 1_00 = 1__0

12) та 23) 11_0 $\vee$ 01_0 = _1_0

13) та 24) 1_10 $\vee$ 0_10 = __10

14) та 15) _110 $\vee$ _100 = _1_0

14) та 19) _110 $\vee$ _010 = __10

15) та 20) _100 $\vee$_000 = __00

16) та 26) 1_00 $\vee$ 0_00 = __00

18) та 27) 10_0 $\vee$ 00_0 = _0_0

19) та 20) _010 $\vee$ _000 = _0_0

21) та 25) 011_ $\vee$ 010_ = 01__

22) та 23) 01_1 $\vee$ 01_0 = 01__

23) та 27) 01_0 $\vee$ 00_0 = 0__0

24) та 26) 0_10 $\vee$ 0_00 = 0__0

1__0
_1_0
1__0
__10
_1_0
__10
__00
__00
_0_0
_0_0
01__

Видаляємо імпліканти, що повторюються

1__0
_1_0
__10
__00
_0_0

Застосуємо закон склеювання імплікант, враховуючи позицію прочерка

1__0 $\vee$0__0 = ___0

_1_0 $\vee$ _0_0 = ___0

__10 $\vee$ __00 = ___0

Видаляємо імпліканти, що

01__ 01__ повторюються

0__0 0__0 ___0

0__0

Мінімальна ДНФ формується з імплікант, які жодного разу не склеїлися (виділені червоним кольором):

101_ 01__ ___0

$$f_{\min} = x_1\overline{x_2}x_3 \vee \overline{x_1}x_2 \vee \overline{x_4}$$

4.4.4. Метод Мак-Класкі

Метод застосовують тоді, коли булева функція задана нормальною формулою.

Алгоритм методу використовує наступні етапи:

1) Кожній конституанті присвоюється індекс – число одиниць термів та номер – відповідне число в десятковій системі числення.
2) Отримані результати заносяться в таблицю, в першому рядку якої записують індекси, а в другому – номери конституент.

Індекс (число одиниць термів)	0	1	2	3	4
Номер (відповідне число в двійковій системі)					

3) Виконується склеювання за правилом: нехай i – індекс, j – індекс, $j>i$.

Склеюються ті конституанти, різниця між m_i та n_j є степінь двійки:

$n_j - m_i = 2^n$; n=0; 1; 2; 4; …

При склеюванні справа вказується величина різниці. Склеювання відбувається доти, доки воно можливе (різниці степені двійки). Склеювати можна лише сусідні індекси.

X1	X2	X3	X4	Десяткове число
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Приклад 4.51. Мінімізувати булеву функцію

$$f(x_1,x_2,x_3,x_4)=x_1x_2x_3\overline{x_4}\vee x_1x_2\overline{x_3}\,\overline{x_4}\vee x_1\overline{x_2}x_3x_4\vee x_1\overline{x_2}x_3\overline{x_4}\vee x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}\vee \overline{x_1}x_2x_3x_4\vee$$
$$\vee\overline{x_1}x_2x_3\overline{x_4}\vee\overline{x_1}x_2\overline{x_4}\,\overline{x_4}\vee\overline{x_1}x_2\overline{x_3}x_4\vee\overline{x_1}\,\overline{x_2}x_3\overline{x_4}\vee\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4.$$

1) $x_1x_2x_3\overline{x_4}-(1110)_3-14$
2) $x_1x_2\overline{x_3}\,\overline{x_4}-(1100)_2-12$
3) $x_1\overline{x_2}x_3x_4-(1011)_3-11$
4) $x_1\overline{x_2}x_3\overline{x_4}-(1010)_2-10$
5) $x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}-(1000)_1-8$
6) $\overline{x_1}x_2x_3x_4-(0111)_3-7$
7) $\overline{x_1}x_2x_3\overline{x_4}-(0110)_2-6$
8) $\overline{x_1}x_2\overline{x_4}\,\overline{x_4}-(0100)_1-4$
9) $\overline{x_1}x_2\overline{x_3}x_4-(0101)_2-5$
10) $\overline{x_1}\,\overline{x_2}x_3\overline{x_4}-(0010)_1-2$
11) $\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}-(0000)_0-0$

Індекс (число одиниць термів)	0	1	2	3	4
Номер (відповідне число в двійковій системі)	0	2 4 8	5 6 10 12	7 11 14	

I

(індекси 1 та 0)

(0,2) (2)

(0,4) (4)

(0,8) (8)

II

(індекси 2 та 1)

(2,6) (4)

(4,6) (2)

(2,10) (8)

(4,12) (8)

(8,12) (4)

(4,5) (1)

(8,10) (2)

III

(індекси 3 та 2)

(5,7) (2)

(10,11) (1)

(10,14) (4)

(6,14) (8)

(12,14) (2)

(6,7) (1)

IV

(етапи I та II)

(0,2,4,6) (2,4)

(0,2,8,10) (2,8)

~~(0,4,2,6) (2,4)~~

(0,4,8,12) (4,8)

~~(0,4,8,12) (4,8)~~

~~(0,2,8,10) (2,8)~~

V

(етапи III та II)

(2,6,10,14) (4,8)

(8,10,12,14) (2,4)

(4,5,6,7) (1,2)

(4,6,12,14) (2,8)

~~(2,6,10,14) (4,8)~~

~~(4,6,12,14) (2,8)~~

~~(4,5,6,7) (1,2)~~

VI

(етапи IV та V)

(0,2,4,6,7,10,12,14) (2,4,8)

~~(0,4,8,12,2,6,10,14) (2,4,8)~~

~~(0,4,8,12,2,6,10,14) (2,4,8)~~

(10,11) (1)

$x_1\overline{x_2}x_3x_4$ - $(1011)_3$ - 11

$x_1\overline{x_2}x_3\overline{x_4}$ - $(1010)_2$ - 10

$x_1\overline{x_2}x_3(101\_)$

(4,5,6,7) (1,2)

$\overline{x_1}x_2x_3x_4$ - $(0111)_3$ - 7

$\overline{x_1}x_2x_3\overline{x_4}$ - $(0110)_2$ - 6

$\overline{x_1}x_2\overline{x_3x_4}$- $(0100)_1$ - 4

$\overline{x_1}x_2\overline{x_3}x_4$- $(0101)_2$ - 5

$\overline{x_1}x_2(01\_\_)$

(0,2,4,6,8,10,12,14) (2,4,8)

$x_1x_2x_3\overline{x_4}$ - $(1110)_3$- 14 $\overline{x_4}$ (_ _ _0)

$x_1x_2\overline{x_3x_4}$ - $(1100)_2$- 12

$x_1\overline{x_2}x_3\overline{x_4}$ - $(1010)_2$- 10

$x_1\overline{x_2x_3x_4}$ - $(1000)_1$ - 8

$\overline{x_1}x_2x_3\overline{x_4}$ - $(0110)_2$ - 6

$\overline{x_1}x_2\overline{x_3x_4}$ - $(0100)_1$ - 4

$\overline{x_1x_2}x_3\overline{x_4}$ - $(0010)_1$ - 2

$\overline{x_1x_2x_3x_4}$- $(0000)_0$ - 0

Після склеювання отримуємо скорочену диз'юнктивну нормальну форму (СДНФ)

$$f_{скор} = x_1\overline{x_2}x_3 \vee \overline{x_1}x_2 \vee \overline{x_4}$$

Для отримання мінімальної ДНФ складемо таблицю Куайна, в якій помістимо отримані спрощенням імпліканти та вихідні конституанти. Одиницю ставимо там, де одиниця «покриває» конституанту, це тому, що конституанта може бути замінена імплікантою за законом поглинання.

x1	x2	x3	x4	1110	1100	1011	1010	1000	0111	0110	0100	0101	0010	0000
_	_	_	0	1	1		1	1		1	1		1	1
1	0	1	_			1	1							
0	1	_	_						1	1	1	1		

Отже, мінімальна ДНФ:

$$f1\overline{x_2}_3\overline{x_1}_2\overline{x_4}_{min}$$

Приклад 4.52. Методом Мак-Класкі мінімізувати булеву функцію, нормальна форма якої : $f = x_4x_3x_2\overline{x_1} \vee x_4x_3\overline{x_2}x_1 \vee x_4\overline{x_3}x_2x_1 \vee \overline{x_4}x_3x_2x_1 \vee x_4x_3\overline{x_2x_1} \vee x_4\overline{x_3x_2}x_1 \vee \overline{x_4x_3}x_2x_1 \vee x_4x_3x_2x_1 \vee \overline{x_1x_2x_3x_4}$

Розв'язок. Запишемо всі конституенти через індекси та присвоєні номери (десяткове числення).

$x_4x_3x_2\overline{x_1}$- $(1110)_3$- 14

$x_4x_3\overline{x_2}x_1$ - $(1101)_3$- 13

$x_4\overline{x_3}x_2x_1$ - $(1011)_3$- 11

$\overline{x_4}x_3x_2x_1$ - $(0111)_3$- 7

$x_4x_3\overline{x_2x_1}$- $(1100)_2$- 12

$x_4\overline{x_3x_2}x_1$ - $(1001)_2$- 9

$\overline{x_4x_3}x_2x_1$ - $(0011)_2$- 3

$x_4x_3x_2x_1$ - $(1111)_4$- 15

$\overline{x_4x_3x_2x_1}$- $(0000)_0$- 0

Отримані результати зенесемо в таблицю:

Індекс	0	1	2	3	4
Номер	0^*	-	3;9;12	7;11;13;14	15

Враховуючи, що склеювати можна лише сусідні індекси, бачимо:

1)конституенту з індексом 0 склеїти ні з якою не можна, бо немає конституент з індексом 1;

2)процес склеювання ($j > i$; n_j - $m_i = 2^n$; n =0;1;2;4;... n_{ij})1 група: (3;7) -(4); (3;11) – (8); (9;11) – (2); (9;13) – (4); (12;13) – (1); (12;14) – (2).(Склеювання конституент з індексом 2 і 3),

2 група: (7;15) – (8); (11;15) – (4); (13;15) – (2); (14;15) – (1). (Склеювання конституент з індексом 3 і 4). Знову проведемо склеювання між 1 і 2 групами, у яких різниці однакові;

$j > i$: (12;13;14;15) – (1;2); (9;11;13;15) – (2;4); (12;13;14;15) – (1;2); (3;7;11;15) – (4;8); (9;11;13;15) – (2;4); (3;7;11;15) – (4;8); (12;13;14;15) – (1;2);

Процес склеювання закінчено. Отримані результати запишемо у вигляді конституент (ліва дужка), права дужка вказує на терми, які треба виключити з конституент, що залишилися.

(12;13;14;15) – (1;2)

$$\left.\begin{aligned} 12 = 1100 - x_4x_3\overline{x_2}\,\overline{x_1} = x_4x_3 \\ 13 = 1101 - x_4x_3\overline{x_2}x_1 = x_4x_3 \\ 14 = 1110 - x_4x_3x_2\overline{x_1} = x_4x_3 \\ 15 = 1111 - x_4x_3x_2x_1 = x_4x_3 \end{aligned}\right\} x_4x_3$$

(9;11;13;15) – (2;4)

$$\left.\begin{aligned} 9 = 1001 - x_4\overline{x_3x_2}x_1 = x_4x_1 \\ 11 = 1011 - x_4\overline{x_3}x_2x_1 = x_4x_1 \\ 13 = 1101 - x_4x_3\overline{x_2}x_1 = x_4x_1 \\ 15 = 1111 - x_4x_3x_2x_1 = x_4x_1 \end{aligned}\right\} x_4x_1$$

(3;7;11;15) – (4;8)

$$\left.\begin{aligned} 3 = 0011 - \overline{x_4x_3}x_2x_1 = x_2x_1 \\ 7 = 1011 - \overline{x_4}x_3x_2x_1 = x_2x_1 \\ 11 = 1011 - x_4\overline{x_3}x_2x_1 = x_2x_1 \\ 15 = 1111 - x_4x_3x_2x_1 = x_2x_1 \end{aligned}\right\} x_2x_1$$

Таким чином, мінімальна нормальна форма, отримана за методом Мак-Класкі для функції має вигляд: $f_{\min} = \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee x_4x_3 \vee x_4x_1 \vee x_2x_1$.

Обов’язкові завдання

1. По приведеній таблиці істинності знайти логічну функцію та спростити її.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

2. Мінімізувати булеву функцію :

$$f(x_1, x_2, x_3, x_4) = \overline{x_4}\overline{x_3}x_2\overline{x_1} \vee \overline{x_4}x_3\overline{x_2}x_1 \vee \overline{x_4}x_3x_2\overline{x_1} \vee \overline{x_4}x_3x_2x_1 \vee$$
$$\vee x_4\overline{x_3}x_2\overline{x_1} \vee x_4x_3\overline{x_2}\overline{x_1} \vee x_4x_3\overline{x_2}x_1 \vee$$
$$\vee x_4x_3x_2\overline{x_1} = f(0010; 0101; 0110; 0111; 1010; 1100; 1101; 1110)$$

3. Мінімізувати булеву функцію : $f = x\bar{y}z\overline{w} \vee x\overline{yzw} \vee xy\overline{zw} \vee \bar{x}y\overline{zw} \vee \overline{xy}zw \vee \overline{x}yzw \vee xyz\overline{w}$ методом Карно-Вейча.

4. Мінімізувати функцію, що задана таблицею, методом Мак-Класкі:

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1