Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

І. В. Кулаковська

ДИСКРЕТНА МАТЕМАТИКА. Частина 2. ТЕОРІЯ ГРАФІВ, ДЕРЕВА. АЛГОРИТМИ НА ГРАФАХ

Методичні рекомендації для виконання лабораторних робіт
з дисципліни «Дискретна математика» студентами спеціальностей
121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки»,
124 «Системний аналіз»

Методичні рекомендації

Випуск 440

ЧНУ
ВИДАВНИЦТВО
Миколаїв – 2024

УДК 004.82
К 90

Рекомендовано до друку Вченою радою факультету комп'ютерних наук ЧНУ ім. Петра Могили (протокол № 6 від 15 січня 2024 року).

Рецензенти:

Н. А. Махровська, д-р тех. наук, доцент кафедри теорії й методики природничо-математичної освіти та інформа-ційних технологій Миколаївського обласного інституту післядипломної педагогічної освіти.

А. В. Швед, д-р тех. наук, професор, кафедра інженерії програмного забезпечення Чорноморського національного університету імені Петра Могили.

К 90 **Кулаковська В. І.** Дискретна математика. Частина 2. Теорія графів, дерева. Алгоритми на графах : метод. рек. для виконання лабораторних робіт з дисципліни «Дискретна математика» студентами спец. 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз» : метод. рек. / В. І. Кулаковська. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2024. – 108 с. – (Методична серія ; вип. 440).

Методичні рекомендації містять теорію та завдання для самостійного виконання студентами лабораторних робіт при вивченні основ теорії графів, дерев, класичних алгоритмів на графах. Поряд із теоретичними відомостями наводяться приклади виконання завдань, програмно-алгоритмічні рішення, вимоги до оформлення звітів та контрольні запитання.

Методичні рекомендації призначені для бакалаврів спеціальностей 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз», які вивчають дисципліни «Дискретна математика» або «Дискретні структури та дискретна математика».

УДК 004.82

ISSN 1811-492X

Зміст

Лабораторна робота №1
Тема: Задачі, які приводять до джерел теорії графів.

Мета: опрацювати методи аналізу графів за степеню вершин, ізоморфізму графів та розв'язати задачу чотирьох фарб.

Теоретичні відомості

У цьому розділі курсу ми розглядаємо поняття графа. Останнім часом теорія графів стала простим, доступним і потужним засобом рішення проблем, що відносяться до широкого круга питань математичного моделювання. Це питання розробки інтегральних схем і схем управління, дослідження математичних моделей та автоматів, логічних ієрархій, блок-схем у програмуванні, проблем економіки і статистики, хімії і біології, теорії розкладів і дискретної оптимізації.

Перші задачі теорії графів були пов'язані з рішенням математичних розважальних задач і головоломок, наприклад:

- ***задача про кенігсберзькі мости*** (задача Ейлера), розвиток якої привів) до циклу задач про обходи графів;
- ***задачі про перевезення***, рішення яких привело до створення ефективних методів рішення ***транспортних задач*** тощо;
- ***задача чотирьох фарб*** привела до досліджень графів та парополучень, які мають теоретичне і прикладне значення.

Багато результатів середини 19-го століття, що відносяться до теорії графів, були отримані при рішенні практичних проблем.

- Г. Кірхгоф при складанні ***повної системи рівнянь для струмів і напруг у електричній схемі,*** власне кажучи, запропонував зображати таку схему графом і знаходити в ньому дерева, за допомогою яких незалежні системи контурів виділяються лінійно.
- А. Келі, виходячи із задач підрахунку числа ізомерів граничних вуглеводнів, прийшов до задач ***перерахування і підрахунку дерев, які мають задані властивості.***

У 20 столітті задачі, пов'язані з графами, почали з'являтись не тільки у фізиці, електротехніці, хімії, біології, економіці, соціології тощо, але й усередині математики, у таких її розділах як алгебра, топологія, теорія ймовірностей, теорія чисел тощо. Методи цих розділів дискретного аналізу стали успішно використовуватися для рішення задач теорії графів.

Поряд із терміном *граф* на початку 20 століття вживалися як синоніми й інші терміни, наприклад, *карта, комплекс, діаграма, мережа*, *лабіринт*.

Означення. Нехай V – непорожня скінченна множина, а V*V – множина всіх двоелементних кортежів множини V, які називатимемо ребрами. ***Графом*** (***неорієнтованим графом або звичайним***) G називається пара множин (V, E), де E– множина ребер або довільна підмножина множини $V^{(2)}$ ($E \subseteq V^{(2)}$). Позначається $G = (V, E)$. При цьому елементи множини $\boldsymbol{V}$ називаються ***вершинами*** графа G елементи множини E – ***ребрами*** графа G. Відповідно, V– ***множиною вершин,*** E– ***множиною ребер*** графа G.

Дві вершини може бути з'єднані двома або більше ребрами, такі ребра називають ***кратними***. Кожен граф можна представити в евклідовому просторі множиною точок, які відповідають вершинам, які з'єднані лініями, які відповідають ребрам – таке представлення називається ***укладкою*** графа.

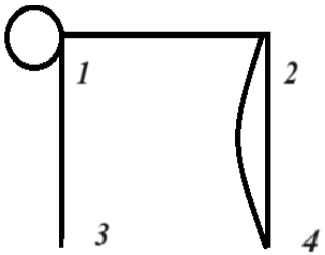
Ступінь вершини графа – це число ребер, які інцидентні даній вершині, причому петлі враховуються двічі. Позначається степінь вершини $\boldsymbol{v}$ через $deg(v)$ або $\delta(v)$.

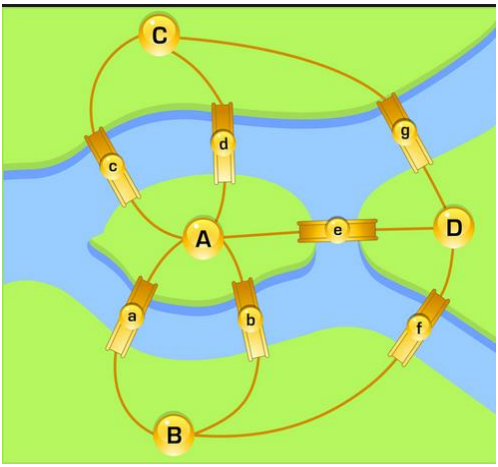
Приклад. $G = (V, E)$; $V = \{1,2,3,4\}$;

$E = \{(1,1),(1,2),(1,3),(2,4),(2,4)\}$

$deg(1)=3$; $deg(2)=3$;

$deg(3)=1$; $deg(4)=2$.

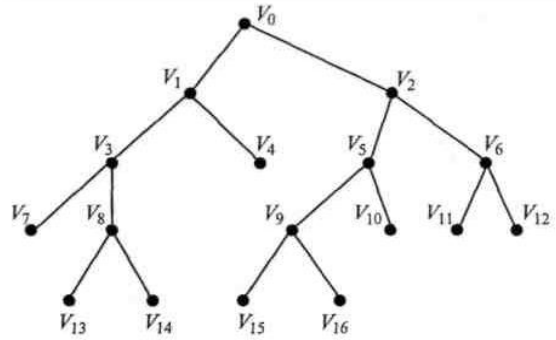




Циклом Ейлера називають цикл, який проходить уздовж кожного ребра графіка рівно один раз. Граф із циклом Ейлера також називають графом Ейлера.

Критерій Ейлера для графа: зв'язний граф є ейлеревим тоді і тільки тоді, коли ступені всіх його вершин парні.

Зв'язний граф без циклів називається деревом.



На практиці дерева особливо часто зустрічаються при описі різних ієрархій. Дерева є дуже зручними інструментами для подання найрізноманітнішої інформації. Дерева відрізняються від простих графів тим, що під час обходу дерева неможливі цикли. Це робить графіки дуже зручною формою організації даних для різних алгоритмів. Тому поняття дерева активно використовується в інформатиці та програмуванні.

Граф без циклів називається лісом. Вершини 1 ступеню у дереві називаються листками.

Проблема чотирьох кольорів

Проблема чотирьох кольорів – це математична проблема, поставлена Френсісом Гатрі (Великобританія) у 1852 році.

Запитання: з'ясуйте, чи можна будь-яку карту розфарбувати чотирма кольорами, щоб будь-які дві області, які мають спільний кордон, були забарвлені різними кольорами.

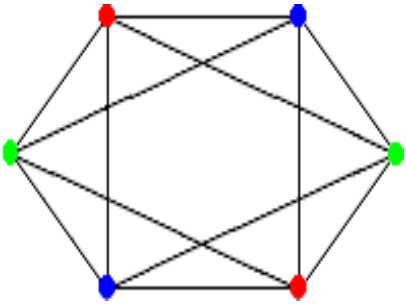
Іншими словами, кількість кольорів зафарбованої поверхні карти не перевищує 4.

Найвідоміші спроби доказів:

У 1879 році Альфред Кемп запропонував доказ, який був спростований у 1880 році на основі його ідеї про те, що будь-яка карта може мати 5 кольорів.

Пітер Тет запропонував інший доказ у 1880 році, але він був спростований у 1891 році.

К. Аппель і В. Хакен продемонстрували в 1976 році, що будь-яку карту можна розфарбувати таким чином, щоб сусідні країни були зафарбовані різними кольорами. Це була перша велика математична теорема, доведена за допомогою комп'ютера. Незважаючи на наступні спрощення, перевірити цей доказ без використання комп'ютера майже неможливо. Тому деякі математики з недовірою ставляться до цього.



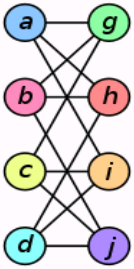
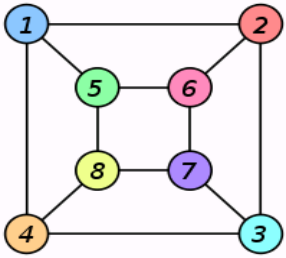
Розфарбовуванням графа G = (V, E) називається відображення $\varphi: V \to N$. Розфарбовування називається правильним, якщо фарби будь-яких двох суміжних вершин різні. Хроматичним числом графа називається мінімальна кількість фарб, необхідна для правильного розфарбовування графа.

Для плоских графів існує знаменита проблема чотирьох фарб. Вона полягає в тому, щоб довести (або спростувати) твердження, що хроматичне число будь-якого

плоского графа не перевищує 4. Ця проблема була позитивно розв'язана всього кілька років тому з використанням комп'ютерного аналізу різних варіантів.

Ізоморфізм графів

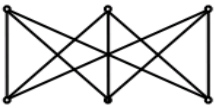
Графи ***G1=(V1,E1) і G2=(V2, E2)*** називаються ізоморфними, якщо існує таке взаємно однозначне відображення φ множин вершин V_1 на множину вершин V_2, що ребро $(i,j) \in E1$ тоді і тільки тоді, коли ребро $(\varphi(i),\varphi(j)) \in E2$.

Граф G	Граф H	Ізоморфізм між G і H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

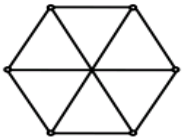
Ізоморфізм графів G і H – це взаємно однозначна бієкція між множинами вершин G і H.

Відображення F: V(G) → V(H) таке, що будь-які дві вершини u і v графа G суміжні тоді і тільки тоді, коли f(u) і f(v) суміжні в H.

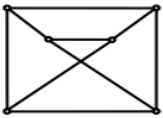
Відображення φ називається ізоморфним відображенням або ізоморфізмом графа G1 на граф G2. Таким чином, ізоморфні графи відрізняються фактично лише ідентифікаторами (іменами) своїх вершин. З точки зору теорії графів ця відмінність не є суттєвою, тому звичайно ізоморфні графи ототожнюють і, зображаючи графи у вигляді діаграм, або зовсім не ідентифікують їхні вершини, або нумерують вершини натуральними числами.



G_1

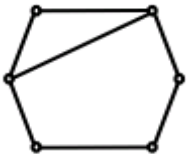


G_2

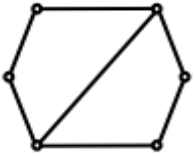


G_3

Пропонуємо переконатись, що графи G1, G2, G3 ізоморфні між собою, а графи H1, H2 – не є ізоморфними.

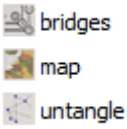


H_1



H_2

Завдання до лабораторної роботи (5 задач).
В папці з лабораторною роботою знаходяться 3 файли:



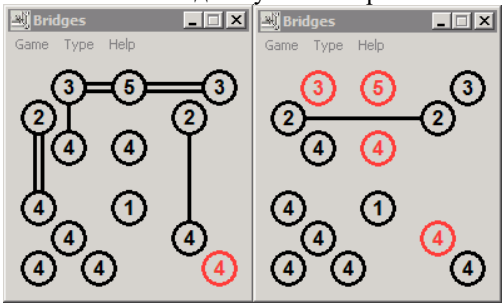
1. Для ознайомлення із суміжністю вершин, ребер, інцедентністю в графі та степенями вершин запустіть перший файл: bridges.

1.1. Переведіть із 7 вершин в режим 10 вершин. Вкладка Type.

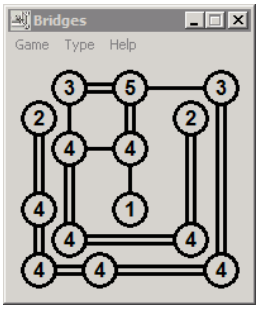
1.2. Потрібно зв'язати вказані вершини кількістю ребер, вказаних цифрою всередині.

1.3.

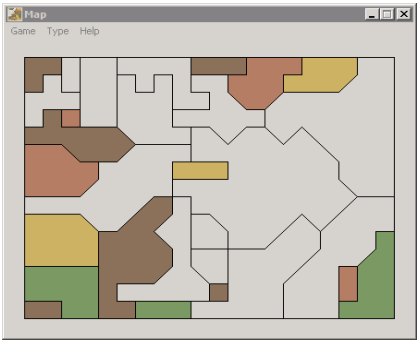
1.4. Для проведення ребра вказати послідовно вершини.
У випадку помилок вони підсвічуються червоним кольором.



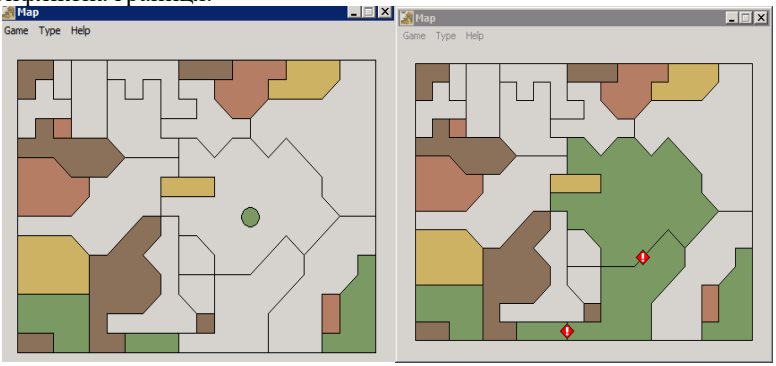
1.5. Результат виконання



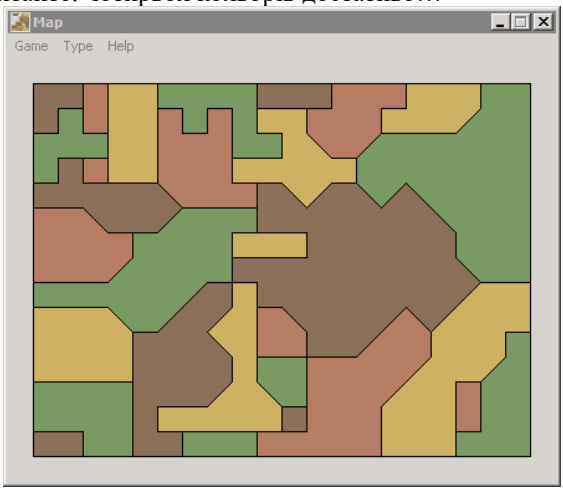
1.6. До звіту вставити скріншоти початку і результату.
2. Для імітації задачі чотирьох фарб запустіть файл: map.
2.1. Вибирайте складність нормал (середню). Області, зафарбовані на початку, колір не змінюють.



2.2. Для перетягування кольору наведіть на область цього кольору і перетягніть в потрібну область. У випадку помилки буде позначена конфліктна границя.



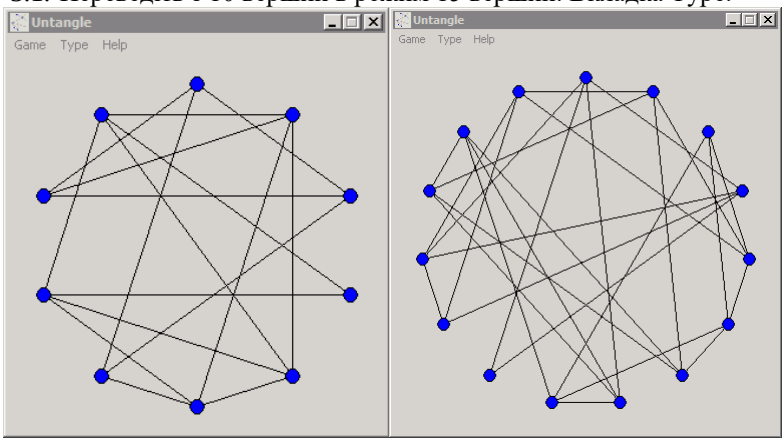
2.3. Пам’ятайте: чотирьох кольорів достатньо!!!



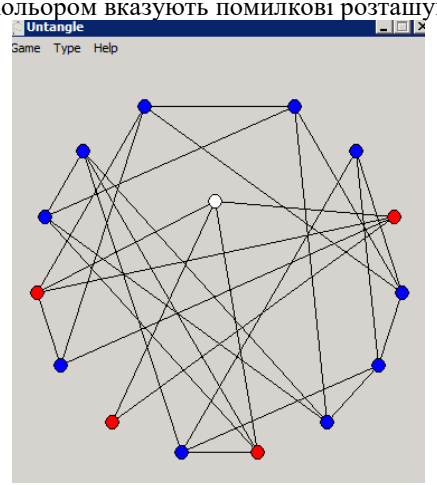
2.4. До звіту вставити скріншоти початку і результату.

3. Для ізоморфності графів використовують представлення графа у вигляді, коли його ребра не перетинаються. Запустіть файл: untangle.

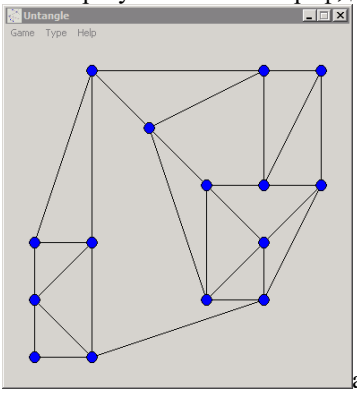
3.1. Переведіть з 10 вершин в режим 15 вершин. Вкладка Type.



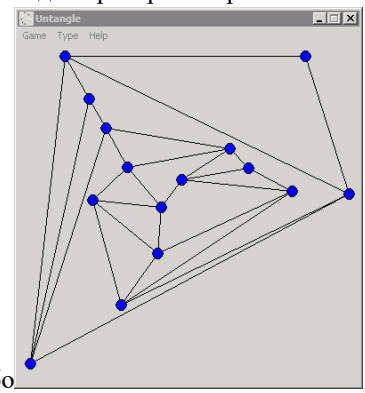
3.2. Переміщуючи вершини, зверніть увагу, що їх суміжність не змінюється, змінюється її положення (координати), довжини ребер, але не суміжність! Кольором вказують помилкові розташування.



3.3. В результаті маємо граф, де жодні з ребер не перетинаються.



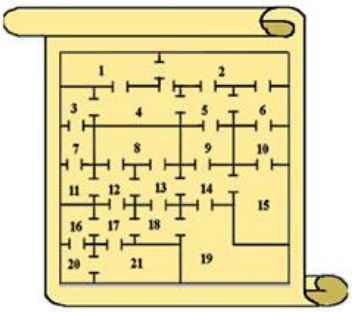
або

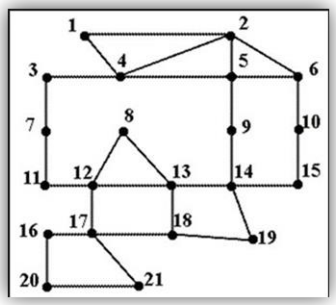


3.4. До звіту вставити скріншоти початку і результату.

4. Ейлерові графи.

4.1 На малюнку дано план підземелля, в одній із кімнат якого прихований потрібний вам ключ. Для відшукання ключа досить увійти в одну з крайніх кімнат підземелля, пройти через всі двері, причому в точності по одному разу через кожну. Ключ схований за тими дверима, які будуть пройдені останніми.





Вкажіть номер кімнати, в якій захований ключ.

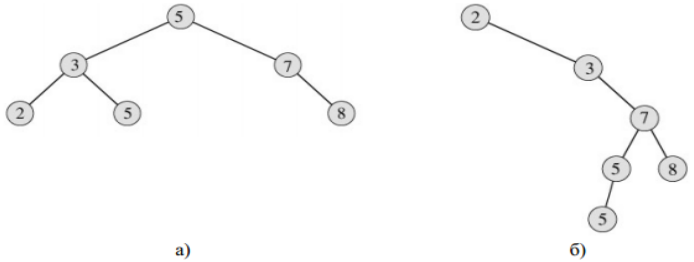
5. Бінарні дерева

5.1. Бінарне дерево в першу чергу може бути представлене за допомогою зв’язаної структури даних, в якій кожний вузол є об’єктом. Ключі у бінарному дереві пошуку зберігаються таким чином, щоб в

будь-який момент задовольняти наступній умові бінарного дерева пошуку: якщо x – вузол бінарного дерева пошуку, а вузол y знаходиться у лівому піддереві x, то $key[y] \leq key[x]$; якщо y знаходиться у правому піддереві x, то $key[y] > key[x]$.

Бінарне дерево [5, 3, 5, 2, 7, 8], представлене на рис. а), ключ кореня дорівнює 5, ключі 2, 3 та 5, які не більші значення ключа в корені, знаходяться в його лівому піддереві, а ключі 7 та 8, які не менші кореня, – в правому піддереві. Та сама властивість виконується й для кожного внутрішнього вузла дерева.

На рис. б) зображено дерево з тими самим вузлами [2, 3, 5, 5, 7, 8], які більш впорядковані, та яке менш ефективне в роботі, адже його висота дорівнює 4, на відміну від дерева на рис. а), висота якого дорівнює 2.



а) б)

Побудуйте дерево[n+3, n, n+6, n+9, n+5, n+8, n+2, n+1, n], де n – акаунт.

Лабораторна робота №2
Тема: Способи задання графів, матриці графів.

Мета: опрацювати алгоритми формування матриць.

Теоретичні відомості

Поняття графа. Нехай V – деяка непорожня скінчена множина, а $V^{(2)}$ – множина всіх двоелементних підмножин (невпорядкованих пар різних елементів) множини V.

Граф (***неорієнтований граф***) G називається пара множин (V, E), де E довільна підмножина множини $V^{(2)}$ $(E \subseteq V^{(2)})$. Позначається $G = (V, E)$. При цьому елементи множини V називаються ***вершинами*** графа G, а елементи множини E ***ребрами*** графа G. Відповідно V називається ***множиною вершин*** і E ***множиною ребер*** графа G. Ребра записуються у круглих дужках(v, w). Число вершин графа позначається $|V|$, число ребер – $|E|$.

Неупорядкована пара вершин називається ***ребром***, упорядкована пара – ***дугою***. Граф, який містить тільки ребра, називається ***неорієнтованим***; граф, який містить тільки дуги – ***орієнтованим*** (або ***орграфом***). Пара вершин може бути з'єднана двома, або більше ребрами (або, відповідно, дугами одного напрямку), такі ребра називають ***кратними***. Дуга може починатися й закінчуватися в одній і тій самій вершині, в цьому випадку вона називається ***петлею***.

Нехай задано граф $G = (V, E)$. Якщо $(v, w) \in E$, то кажуть, що ***вершини*** v і w є ***суміжними,*** у противному разі вершини v і w є **несуміжними**. Якщо $e = (v, w)$ ребро графа, то вершини v і w називаються ***кінцями*** ребра e. У цьому випадку кажуть також, що ребро e ***з'єднує*** вершини v і w. Вершина v і ребро e називаються ***інцидентними***, якщо v є кінцем e.

Два ***ребра*** називаються ***суміжними***, якщо вони мають спільну вершину.

Кожен граф можна представити в евклідовому просторі множиною точок, які відповідають вершинам, які з'єднані лініями, які відповідають ребрам (або дугам – в останньому випадку напрямок вказується стрілками) – таке представлення називається ***укладкою*** графа.

Відомо, що у *3-вимірному просторі* будь-який граф можна представити у вигляді укладки таким чином, що лінії, які відповідають ребрам (дугам) не будуть *перетинатися*. Для *2-вимірного простору* це невірно.

Допускають представлення у вигляді укладки в 2-вимірному просторі графи, які називають ***пласкими***.

Способи задання графів

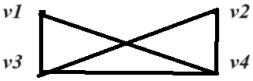
1. Завдання множин V і E за допомогою ***переліку їх елементів***.

Приклад.

Граф $G_1 = (V_1, E_1)$, де $V_1 = \{v_1, v_2, v_3, v_4\}$,

$E_1 = \{(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$

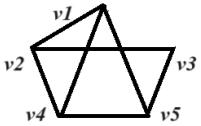
це граф із чотирма вершинами і п'ятьма ребрами.



Граф $G_2 = (V_2, E_2)$, де $V_1 = \{v_1, v_2, v_3, v_4, v_5\}$,

$E_1 = \{(v_1, v_2), (v_2, v_4), (v_1, v_5), (v_3, v_2), (v_3, v_5), (v_4, v_1), (v_5, v_4)\}$

це граф із п'ятьма вершинами і сімома ребрами.



2. Графічний. Граф $G = (V, E)$представляють за допомогою малюнків на площині, який називають ***діаграмою*** графа G. Вершинам графа G ставляться у відповідність точки площини; точки v і w, які з'єднуються лінією (відрізком або кривою) тоді і тільки тоді, коли v і w суміжні вершини.

3. Завдання графа за допомогою матриці суміжності. Занумеруємо всі вершини графа G натуральними числами від 1 до n. ***Матрицею суміжності*** A графа G називається квадратна $n \times n$ -матриця, в якій елемент a_{ij} i-го рядка і j-го дорівнює 1, якщо вершини v_i та v_j з номерами i та j суміжні, і 0 у іншому випадку.

Приклад. Для графів G_1 і G_2 маємо відповідно

$$A_1 = \begin{pmatrix} 0\,0\,1\,1 \\ 0\,0\,1\,1 \\ 1\,1\,0\,1 \\ 1\,1\,1\,0 \end{pmatrix} \text{ і } A_2 = \begin{pmatrix} 0\,1\,0\,1\,1 \\ 1\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1 \\ 1\,1\,0\,0\,1 \\ 1\,0\,1\,1\,0 \end{pmatrix}.$$ Матриці суміжності графів симетричні.

4. Задання графа за допомогою матриці інцидентності. Занумеруємо всі вершини графа G числами від 1 до n і всі його ребра числами від 1 до m. ***Матрицею інцидентності*** B графа G називається $n \times m$ -матриця, в якій елемент $b_{ij}$$i$-го рядка і j-го стовпчика дорівнює 1, якщо вершина v_i з номером iінцидентна ребру e_j з номером j, і дорівнює 0 у противному разі.

Приклад. Для графів G_1 і G_2 маємо.

$$B_1 = \begin{pmatrix} 1\,1\,0\,0\,0 \\ 0\,0\,1\,1\,0 \\ 1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,1 \end{pmatrix} \text{ і } B_2 = \begin{pmatrix} 1\,0\,1\,0\,0\,1\,0 \\ 1\,1\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,1\,0\,0 \\ 0\,1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,0\,1\,0\,1 \end{pmatrix}$$

5. Задання графа за допомогою списку суміжності. При цьому кожній вершині графа відповідає свій список. У список, що відповідає вершині *v*, послідовно записуються всі суміжні їй вершини.

Приклад. Для графів G_1 і G_2 маємо списки

G_1:	v_1: v_3,v_4	v_2: v_3,v_4	v_3: v_1,v_2,v_4	v_4: v_1,v_2,v_3	
G_2:	v_1: v_2,v_4,v_5	v_2: v_1,v_3,v_4	v_3: v_2,v_5	v_4: v_1,v_2,v_5	v_5: v_1,v_3,v_4

Вибір та зручність того чи іншого зі способів завдання графів залежать від особливостей задачі, яка розв'язується.

Якщо порядок елементів, які входять в E, має значення, то граф називається ***орієнтованим*** (*directedgraph*), скорочено – орграф (*digraph*), навпаки – ***неорієнтованим*** (*undirectedgraph*). Ребра орграфа називаються **дугами** (*arcs*). В подальшому будемо вважати, що термін «граф», який застосовують без уточнень «орієнтований» чи «неорієнтований», означає неорієнтований граф.

Ступінь вершини графа – це число ребер, які інцидентні даній вершині, причому петлі враховуються двічі. Позначається ступінь вершини v через $deg(v)$ або $\delta(v)$. Оскільки кожне ребро інцидентне двом вершинам, сума ступенів усіх вершин графа дорівнює подвоєній кількості ребер: $Sum(deg(v_i),\ i = 1,..|V|) = 2 \cdot |E|$.

Властивість 1. Ступені вершин повного графа рівні, і кожен з них на 1 менше числа вершин цього графа.

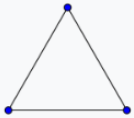
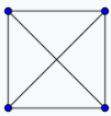
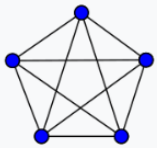
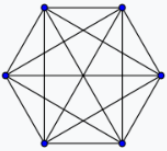
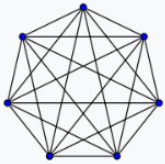
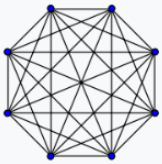
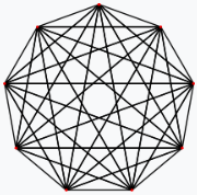
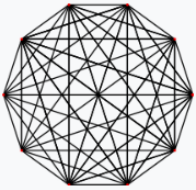
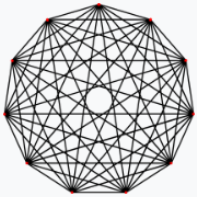
Властивість 2. Сума ступенів вершин графа – число парне, яке дорівнює подвоєному числу ребер графа.

Властивість 3. Число непарних вершин будь-якого графа парне.

Граф, який не містить петель і кратних ребер, називається ***звичайним***, або ***простим графом*** (*simplegraph*). У багатьох публікаціях використовується інша термінологія: під графом розуміють простий граф, граф із кратними ребрами називають ***мультиграфом***, з петлями – ***псевдографом***.

Деякі класи графів отримали особливі найменування. Граф з будь-якою кількістю вершин, який не містить ребер, називається ***порожнім***. Звичайний граф з n вершинами, будь-яка пара вершин якого з'єднана

ребром, називається ***повним*** і позначається K_n (в повному графі $\frac{n(n-1)}{2}$ ребер).

$K_1 : 0$	$K_2 : 1$	$K_3 : 3$	$K_4 : 6$
			
$K_5 : 10$	$K_6 : 15$	$K_7 : 21$	$K_8 : 28$
			
$K_9 : 36$	$K_{10} : 45$	$K_{11} : 55$	
			

Доповненням графа G називається граф $\overline{G}$ з тими самими вершинами, що й граф G, і з тими й тільки тими ребрами, які необхідно додати до графа G, щоб отримати повний граф.

Граф, вершини якого можна розбити на підмножини, що не перетинаються, V_1 и V_2 так, що ніякі дві вершини, які належать одній і тій самій підмножині, не суміжні, називається ***дводольним*** (або ***біхроматичним***, або ***графом Кеніга***) і позначається B_{mn}. ***Повний дводольний*** граф – такий дводольний граф, що кожна вершина множини V_1 зв'язана з всіма вершинами множини V_2, і навпаки; позначається – K_{mn}.

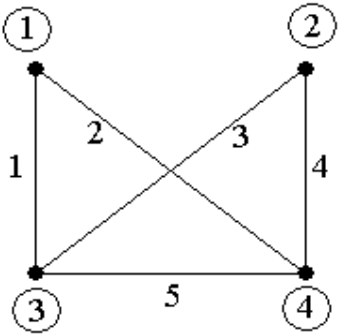
Методичні вказівки до розв'язування задач

1. 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 та інцидентності графа.
Розв'язок.

Побудуємо діаграму графа



Матриця суміжності A і матриця інцидентності B графа мають вигляд:

$$A=\begin{pmatrix}0011\\0011\\1101\\1110\end{pmatrix}$$

$$B=\begin{pmatrix}11000\\00110\\10101\\01011\end{pmatrix}.$$

2. Нехай $V=\{a,b,c,d,e\}$. Граф $G=(V,E)$ задано за допомогою матриці суміжності A.

$$A=\begin{pmatrix}0\,1\,1\,0\,1\\1\,0\,0\,1\,1\\1\,0\,0\,0\,1\\0\,1\,0\,0\,0\\1\,1\,1\,0\,0\end{pmatrix}$$

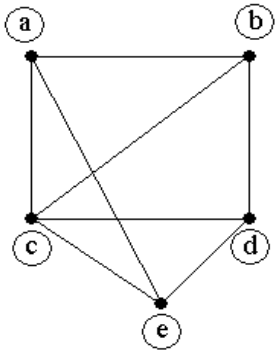
Визначити множину ребер E графа G. Побудувати діаграму та матрицю інцидентності графа G.

Розв'язок. Тобто, маємо

$E=\{(b,c),(a,b),(a,c),(a,e),(c,e),(e,d).(c,d)$

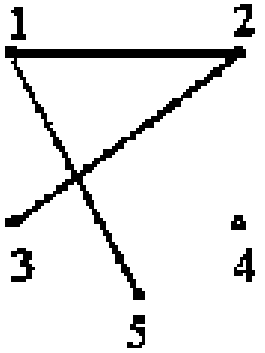
Побудуємо діаграму графа

Матриця інцидентності B графа має вигляд:



$$A = \begin{pmatrix} 01001 \\ 10001 \\ 01000 \\ 10000 \end{pmatrix}$$

3. Граф G задано його діаграмою. Визначити множину вершин V і множину ребер E, матриці суміжності та інцидентності графа G.



Розв'язок. Тобто, маємо $E = \{(1,2), (1,5), (2,3)\}$, $V = \{1,2,3,4,5\}$

Матриця суміжності A і матриця інцидентності B графа мають вигляд:

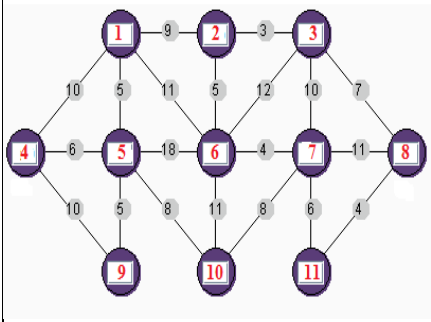
$$A = \begin{pmatrix} 01001 \\ 10100 \\ 01000 \\ 00000 \\ 10000 \end{pmatrix} \text{ і } B = \begin{pmatrix} 11000 \\ 01100 \\ 00100 \\ 00000 \\ 10000 \end{pmatrix}$$

4. Чому дорівнює кількість ребер у графі G, якщо граф G має n вершин і k ребер?

Розв'язок. Нехай G має m ребер. За визначенням граф G має ті самі вершини що й граф G,і тільки ті ребра, які необхідно додати до графа G, щоб отримати повний граф. За означенням, в повному графі $\frac{n(n-1)}{2}$ ребер. Отже, кількість ребер у графі G дорівнює $m = \frac{n(n-1)}{2} - k.$

Приклад завдання

Скласти списки зв'язності для графа, матрицю суміжності, матрицю навантаженості, матрицю інцидентності.



1) Список зв'язності

вершина	список	ступінь
1	2.4.5.6	4
2	1.3.6	3
3	2.6.7.8	4
4	1.5.9	3
5	1.4.6.9.10	5
6	1.2.3.5.7.10	6
7	3.6.8.10.11	5
8	3.7.11	3
9	4.5	2
10	5.6.7	3
11	7.8	2

2) Матриця суміжності, ребер =20

вершина	1	2	3	4	5	6	7	8	9	10	11	ступінь
1		1		1	1	1						4
2	1		1			1						3
3		1				1	1	1				4
4	1				1				1			3
5	1			1		1			1	1		5
6	1	1	1		1		1			1		6
7			1			1		1		1	1	5
8			1				1				1	3
9				1	1							2
10					1	1	1					3
11							1	1				2
ступінь	4	3	4	3	5	6	5	3	2	3	2	TRUE

3) Матриця навантаженості

20	1	2	3	4	5	6	7	8	9	10	11
1		9		10	5	11					
2	9		3			5					
3		3				12	10	7			
4	10				6				10		
5	5			6		18			5	8	
6	11	5	12		18		4			11	
7			10			4		11		8	6
8			7				11				4
9				10	5						
10					8	11	8				
11							6	4			

4) Позначення ребер

1		e1		e2	e3	e4					
2	1		e5			e6					
3		1				e7	e8	e9			
4	1				e10				e11		
5	1			1		e12			e13	e14	
6	1	1	1		1		e15			e16	
7			1			1		e17		e18	e19
8			1				1				e20
9				1	1						
10					1	1	1				
11							1	1			

5) Матриця інцидентності

Реб верш	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1																
2	1				1	1														
3					1		1	1	1											
4		1								1	1									
5			1							1		1	1	1						
6				1		1	1					1			1	1				
7								1							1		1	1	1	
8									1								1			1
9											1		1			1				
10														1				1		
11																			1	1

Лабораторна робота №3
Тема: Основні властивості графів

Теоретичні відомості

Маршрутом (або ***шляхом***) у графі $G = (V, E)$ називається послідовність

$v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1}$, вершин v_i і ребер e_i така, що кожні два сусідні ребра в цій послідовності мають спільну вершину, отже, $e_i(v_i, v_{i+1}), i = 1,2, \ldots k$. Вершина v_1 називається ***початком*** шляху, а вершина v_{k+1}– ***кінцем*** шляху. Всі інші вершини цього шляху називаються ***проміжними***, або ***внутрішніми***, вершинами.

Кількість k ребер у маршруті називається довжиною маршруту. Кажуть, що цей маршрут з'єднує вершини v_1 і v_{k+1} або веде з вершини v_1 у вершину v_{k+1}.

Маршрутом довжини 0 вважається послідовність, що складається з єдиної вершини. Маршрут, в якому всі ребра попарно різні, називається ***ланцюгом***. Маршрут, в якому всі проміжні вершини попарно різні, називається ***простим ланцюгом***.

Маршрут (3.2) називається ***замкненим*** (або ***циклічним***), якщо $v_1 = v_{k+1}$. Замкнений ланцюг називається ***циклом***, а замкнений простий ланцюг –***простим циклом***.

Теорема 1. Будь-який маршрут, що веде з вершини v у вершину w, містить у собі простий ланцюг, що веде з v у w.

Граф, усі ребра якого утворюють простий цикл довжини n, позначається C_n. Простий цикл довжини 3 називається ***трикутником***.

Граф називається ***зв'язним***, якщо будь-яка пара його вершин може бути з'єднана деяким маршрутом.

Компонентою зв'язності (або ***зв'язною компонентою***) графа G називається його зв'язний підграф такий, що він не є власним підграфом жодного іншого зв'язного підграфа графа G.

Відстанню між вершинами v і w зв'язного графа (позначається $d(v, w)$) називається довжина найкоротшого простого ланцюга, що з'єднує вершини v і w. Оскільки кожна вершина графа $G = (V, E)$ з'єднана сама із собою маршрутом довжини 0, то для всіх $v \in V$ виконується

$d(v,w) = 0$. Означена функція відстані задовольняє три аксіоми метрики, тобто для будь-яких вершин $v, w, u \in V$ виконується:

1) $d(v,w) \geq 0$; $d(v,w) = 0$тоді і тільки тоді, коли $v = w$;

2) $d(v,w) = d(w,v)$;

3) $d(v,w) \leq d(v,u) + d(u,w)$.

Ексцентриситетом $e(v)$ довільної вершини v зв'язного графа $G = (V,E)$ називається найбільша з відстаней між вершиною v і всіма іншими вершинами графа G, тобто $e(v) = \max\limits_{w \in V} \{d(v,w)\}$.

Діаметром зв'язного графа G (позначається $D(G)$) називається максимальний з усіх ексцентриситетів вершин графа G. Мінімальний з усіх ексцентриситетів вершин зв'язного графа G називається його ***радіусом*** і позначається $R(G)$.

Вершина v називається ***центральною***, якщо $e(v) = R(G)$. ***Центром*** графа G називається множина всіх його центральних вершин.

Вершини v і w графа G називаються ***зв'язаними***, якщо в G існує маршрут, що з'єднує v і w.

Теорема 2. Будь-який граф однозначно зображується у вигляді прямої суми своїх компонент зв'язності.

Якщо граф G зв'язний, то всі його вершини попарно зв'язані, тобто $V/Z = \{V\}$ і G має єдину зв'язну компоненту, яка збігається із самим графом G.

Теорема 3. Для будь-якого графа $G = (V,E)$ або він сам, або його доповнення є зв'язним графом.

Теорема 4. Якщо G незв'язний граф, то граф $\overline{G}$ зв'язний і $D(\overline{G}) \leq 2$.

Справді, якщо G незв'язний граф, то з доведення теореми випливає, що $\overline{G}$ зв'язний і для будь-яких двох вершин v та w графа $\overline{G}$ виконується або $d(v,w) = 1$, або $d(v,w) = 2$.

Теорема 5. Нехай $G = (V, E)$ зв'язний граф і e деяке його ребро. Розглянемо граф G, який отримано з G вилученням ребра e:

а) якщо ребро e належить деякому циклу графа G, то граф G є зв'язним графом;

б) якщо ребро e не належить жодному циклу графа G, то граф G не є зв'язним графом і має рівно дві компоненти зв'язності.

Теорема 6. Нехай $G = (V, E)$ граф з n вершинами і k компонентами зв'язності. Тоді число його ребер m задовольняє такі нерівності: $n - k \leq m \leq \frac{(n-k)(n-k+1)}{2}$.

Теорема 7. Довільний зв'язний граф з n вершинами містить не менше, ніж $n-1$ ребро.

Теорема 8. Якщо в графі G з n вершинами кількість ребер більша ніж $(n-1)(n-2)/2$, то граф G зв'язний.

Цикломатичним числом графа називається число зв'язних компонент графа плюс число ребер мінус число вершин.

$$\Upsilon = k + m - n$$

Ейлеревим називається цикл, який проходить по кожному ребру графа рівно один раз. Граф, який має ейлерів цикл, також будемо називати ***ейлеровим***.

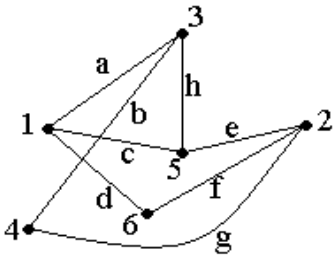
Теорема 9. (Критерій ейлеровості графа). Зв'язний граф є ейлеревим тоді й тільки тоді, коли степені всіх його вершин – парні числа.

Ланцюг, який проходить по кожному ребру рівно один називають **ейлеровими ланцюгом.**

Гамільтоновим називається цикл, який проходить по кожній вершині графа рівно один раз.

Методичні вказівки до розв'язування задач

1. Побудувати 2 маршрути з початком у вершині 2 і кінцем у вершині 2 для даного графа.



Розв'язок. Побудуємо маршрути:

а) $1-a-3-h-5-e-2$,

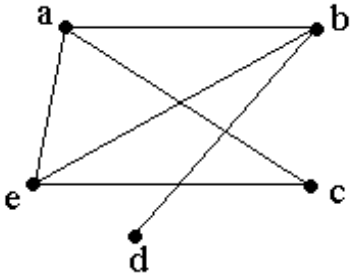
б) $1-c-5-h-3-a-1-d-6-f-2$.

2. Побудувати всі ланцюги для прикладу 1.

Розв'язок. Побудуємо ланцюги:

$1-3-5-1$, $1-5-2-6-1$, $1-3-5-2-6-1$
$1-3-4-2-6-1$, $1-3-4-2-5-1$, $3-4-2-5-3$,
$1-5-3-4-2-6-1$.

3. Знайти для даного графа:
а) радіус,
б) діаметр,
в) цикломатичне число.

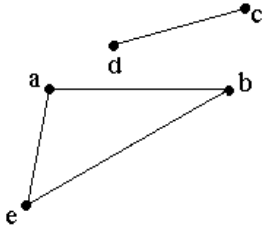


Розв'язок. За означенням, радіус $r = 2$, діаметр $d = 3$.
Цикломатичне число дорівнює $1 + 6 - 5 = 2$.

4. Граф задано матрицею суміжності. Знайти цикломатичне число.

$$A = \begin{pmatrix} 01001 \\ 10001 \\ 00010 \\ 00100 \\ 11000 \end{pmatrix}$$

Розв'язок. Побудуємо діаграму графа.



За означенням, цикломатичне число дорівнює
$2 + 4 - 5 = 1$.

Приклад завдання

Граф G = (X, U) задано матрицею суміжності

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

a) Побудувати геометричне зображення графа. Визначити тип графа, типи ребер та вершин.

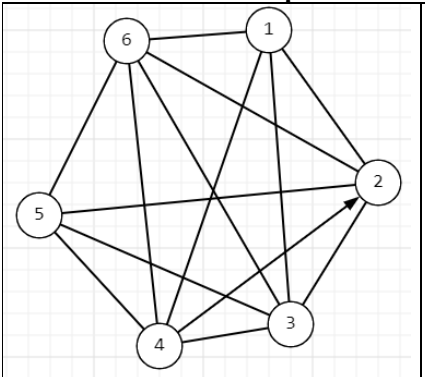
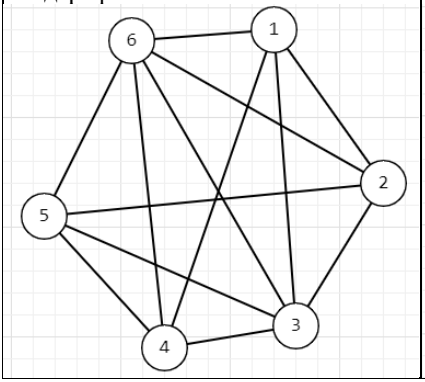
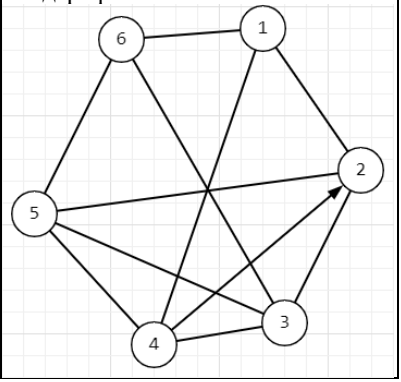
b) Побудувати декілька підграфів, дерево та ліс.

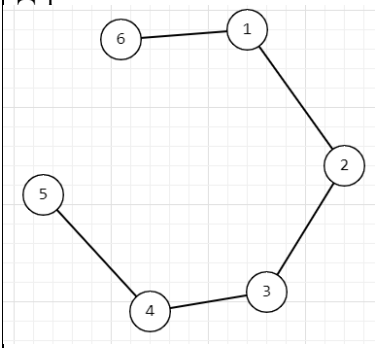
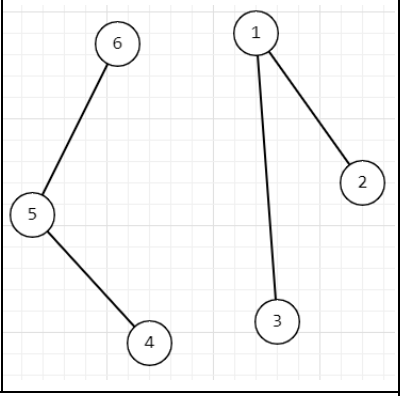
c) Побудувати можливі види маршрутів на графі: маршрут та замкнений маршрут, ланцюг та простий ланцюг, цикл та простий цикл, ланцюг та цикл Ейлера, ланцюг та цикл Гамільтона.

d) Обчислити метричні характеристики графа: ексцентриситети вершин, радіус, діаметр та центр графа, його цикломатичне число.

Таблиця 1

Покроковий опис 2 завдання

<table>
<tr><td> 
</td><td>a) Побудувати геометричне зображення графа та визначити:
тип графа: орієнтовний
типи ребер: неорієнтовні та одне орієнтовне (змішаний граф)
типи вершин: не ізольовані, без петель</td></tr>
<tr><td colspan="2">b) Побудувати декілька підграфів, дерево та ліс.</td></tr>
<tr><td>Підграф 1
 
</td><td>Підграф 2
 
</td></tr>
</table>

<table>
<tr><td>Дерево
 
</td><td>Ліс
 
</td></tr>
<tr><td colspan="2">c) Побудувати можливі види маршрутів на графі: маршрут та замкнений маршрут, ланцюг та простий ланцюг, цикл та простий цикл, ланцюг та цикл Ейлера, ланцюг та цикл Гамільтона.</td></tr>
<tr><td>Маршрут:
1-3-6-4-5-2
4-5-2-6-5-4-2-1
замкнений маршрут:
3-6-1-4-2-5-3
ланцюг: 4-2-3-5-2-1
простий ланцюг: 1-2-3-4-5-6
цикл: 5-2-6-4-6-5
простий цикл: 3-4-2-3</td><td>ланцюг та цикл Ейлера (всі ребра 1 раз):
deg(4,4,5,5,4,5) – більше двох непарних, тому ланцюг та цикл Ейлера – не існують.
ланцюг та цикл Гамільтона (всі вершини 1 раз):
ланцюг 123456
цикл 1234561</td></tr>
<tr><td colspan="2">d) Обчислити метричні характеристики графа: ексцентриситети вершин, радіус, діаметр та центр графа, його цикломатичне число.</td></tr>
<tr><td>ексцентриситети вершин:
e(1)=max{min 1,1,1,2,1}=2;
e(2)=max{min 1,1,2,1,1}=2;
e(3)=max{min 1,1,1,1,1}=1;
e(4)=max{min 1,1,1,1,1}=1;
e(5)=max{min 2,1,1,1,1}=2;
e(6)=max{min 1,1,1,1,1}=1;</td><td>радіус:
R(G)=min{2,2,1,1,2,1}=1
діаметр:
D(G)= max{2,2,1,1,2,1}=2
центр графа:
C(G)={3,4,6}---- e(v)=R
цикломатичне число:
ϒ=e-v+k=14-6+1=9
Тобто, можна видалити 9 ребра це не вплине на зв’язність графа.</td></tr>
</table>

Лабораторна робота №4
Тема: Ізоморфні графи

Теоретичні відомості.

Теорема 1. Графи G_1 і G_1 ізоморфні тоді і тільки тоді, коли матрицю суміжності (матрицю інцидентності) одного з цих графів можна одержати з матриці суміжності (матриці інцидентності) іншого графа за допомогою відповідних перестановок рядків та стовпчиків.

Коли G_1 і G_1 ізоморфні, тоді для їхніх множин вершин існує бієкція φ, а для множин ребер інша бієкція ψ. Загальна ж кількість необхідних кроків для перевірки ізоморфізму графів G_1 і G_1 у цьому випадку не перевищує $n!\cdot m!$.

Доповненням графа $G = (V, E)$ називається граф $\overline{G} = \left(V, V^{(2)} \backslash E\right)$. Отже, граф $\overline{G}$ має ту саму множину вершин V, що і граф G, а вершини графа $\overline{G}$ суміжні тоді і лише тоді, коли вони несуміжні в G. Для графа G з n вершинами виконується $\overline{G} = K_n \backslash G$.

Теорема 2. Графи G_1 і G_2 ізоморфні тоді і тільки тоді, коли ізоморфні їхні доповнення $\overline{G_1}$ і $\overline{G_2}$.

Граф називається ***пласким***, якщо його діаграму можна зобразити на площині так, що лінії, які відповідають ребрам графа, не перетинаються (тобто мають спільні точки тільки у вершинах графа). Таке зображення називається ***пласкою картою*** графа.

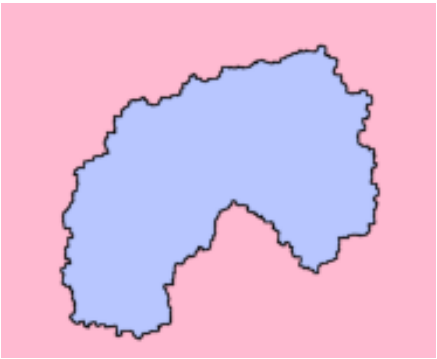
Граф називають ***планарним***, якщо він ізоморфний деякому пласкому графу. Простий цикл, дерево і ліс також планарні графи.

Теорема 3.

1) Будь-який підграф планарного графа є планарним.

2) Граф є планарним тоді і тільки тоді, коли кожна його зв'язна компонента – планарний граф.

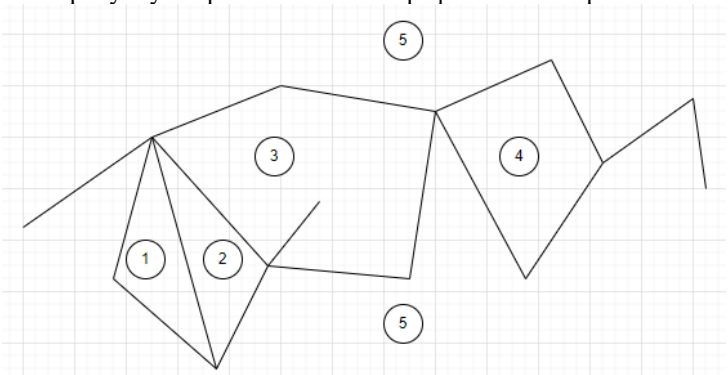
Жордановою кривою будемо називати неперервну лінію, яка не перетинає сама себе. ***Гранню*** плоского графа назвемо множину точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребер графа.



У топології *Жорданова крива* – це довільна замкнена без самоперетинів крива в площині, інакше відома як проста замкнена крива.

Теорема Жордана стверджує, що кожна *Жорданова крива* ділить площину на дві області – внутрішню область, обмежену кривою, і зовнішню, що містить всі ближні і дальні зовнішні точки, причому будь-який шлях, який зв'язує точки з двох регіонів, перетне цю криву в якійсь точці.

Межею грані будемо вважати замкнений маршрут, що обмежує цю грань. На рисунку зображено плаский граф із п'ятьма гранями.



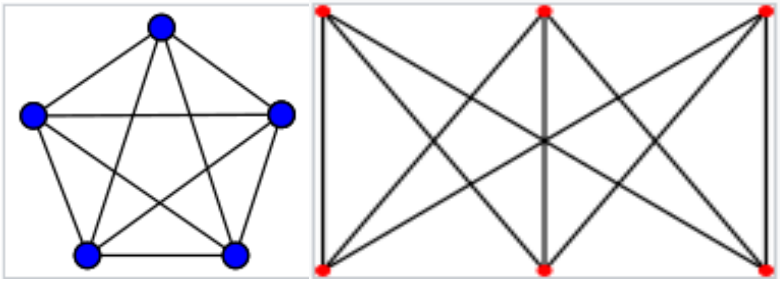
Отже, плаский граф розбиває всю множину точок площини на грані так, що кожна точка належить деякій грані. Відзначимо, що плаский граф має одну, причому єдину, необмежену грань (на рисунку це грань 5). Таку грань будемо називати *зовнішньою*, а всі інші – *внутрішніми* гранями. Множину граней плоского графа позначатимемо через P.

Теорема 4 **(*теорема Ейлера*).** Для будь-якого зв'язного плаского графа $G = (V, E)$ виконується рівність $|V| - |E| + |P| = 2$.

Теорема 5. Для довільного планарного графа $G = (V, E)$ з k компонентами зв'язності виконується $|V| - |E| + |P| = k + 1$

Теорема 6. Графи G і $\overline{G}$ не можуть бути одночасно планарними, якщо кількість вершин у них не менше 11.

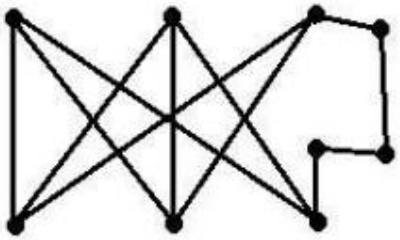
При дослідженні плоских графів особливе місце займають графи K_5 і $K_{3,3}$, зображені на рисунку.



Теорема 8. Графи K_5 і $K_{3,3}$ не є планарними.

Значення графів K_5 і $K_{3,3}$ полягає в тому, що вони є «єдиними» суттєво непланарними графами. Всі інші непланарні графи містять у собі підграфи «подібні» до K_5 або $K_{3,3}$. Характер цієї подібності розкривається за допомогою таких понять.

Елементарним стягуванням графа $G = (V, E)$ називається вилучення в графі G деякого ребра $(v_i, v_j) \in E$ і злиття вершин v_i і v_j в одну вершину v, причому v інцидентна всім тим відмінним від (v_i, v_j) ребрам графа G, які були інцидентні або v_i , або v_j . Кажуть, що граф G ***стягується*** до графа G', якщо G' можна отримати з G за допомогою послідовності елементарних стягувань.



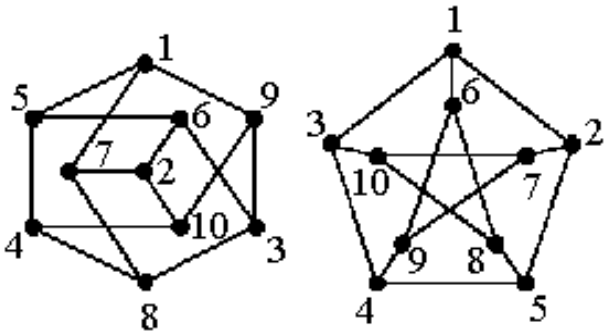
Теорема 9 (***теорема Куратовського***). Граф G є планарним тоді і тільки тоді, коли він не містить підграфів, що стягуються до K_5 або $K_{3,3}$.

Для того, щоб з'ясувати чи є ізоморфними два графа, необхідно впевнитися в тому, що в них: однакова кількість вершин; якщо вершини одного графа з'єднані ребром, то й відповідні їм вершини іншого графа також з'єднані ребром.

Методичні вказівки до розв'язування задач

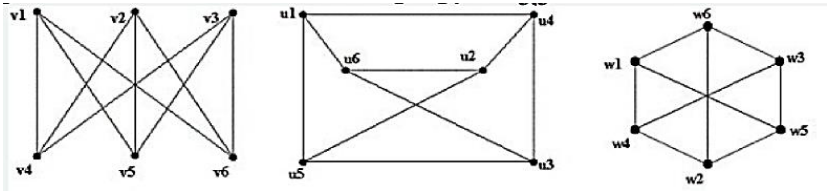
1. Побудувати ізоморфні графи.

Розв'язок. Одним з ізоморфних відображень є: (0,0), (1,3), (2,5), (3,6), (4,7), (5,2), (6,1), (7,4), (8,9), (9,8).



2. Чи є ізоморфними між собою графи G_1, G_2, G_3, G_4?

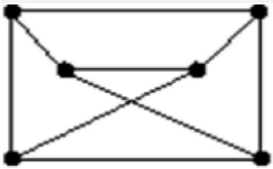
Розв'язок. За означенням графи G_1, G_2, G_3 є ізоморфними між собою, а з графом G_4 вони не є ізоморфними.



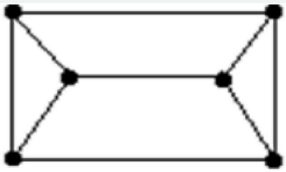
G1 *G2* *G3*

Графи розглядаються з точністю до ізоморфізму, тобто розглядаються класи еквівалентності по відношенню ізоморфізму. Три зовні розрізнюванні діаграми, наведені на рисунку згори, є діаграмами одного і того ж графу К3,3.

Кількість вершин, ребер і кількість суміжних вершин для кожної вершини не визначають граф. На рисунку внизу представлені діаграми графів, у яких інваріанти збігаються, але графи при цьому не ізоморфні.



G2



G4

Приклад завдання

1) Чи є вказаний граф самодоповнювальним? Для цього було виконано ряд операцій, зображених на рисунку 1.

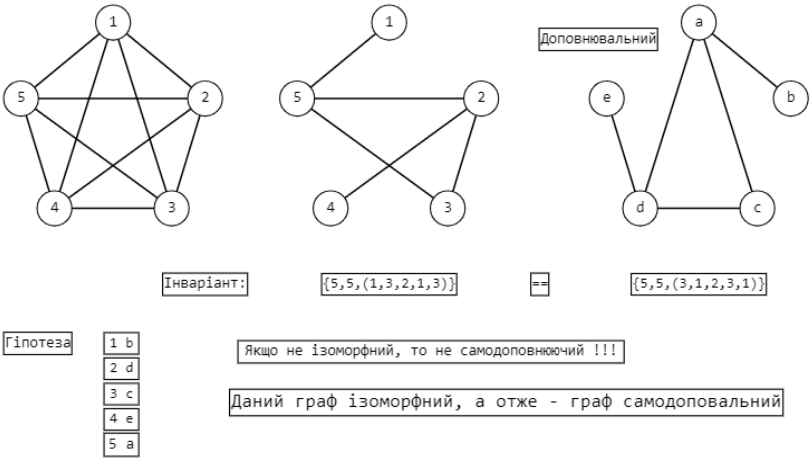


Рисунок 1 – Самодоповненість графа

2) Чи є два вказаних графи ізоморфними? Для цього було виконано ряд операцій, зображених на рис. 2.

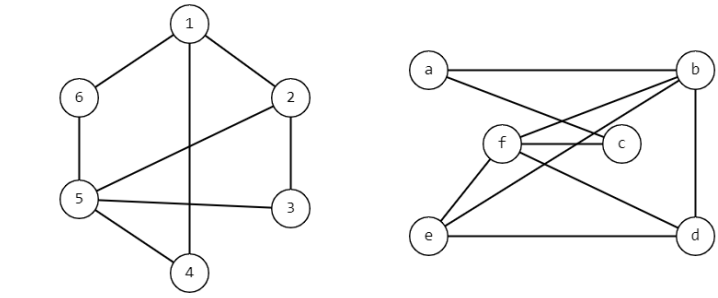


Рисунок 2 – Доведення неізоморфності

3) Чи є два вказаних графи ізоморфними? Для цього було виконано ряд операцій, зображених на рис. 3.

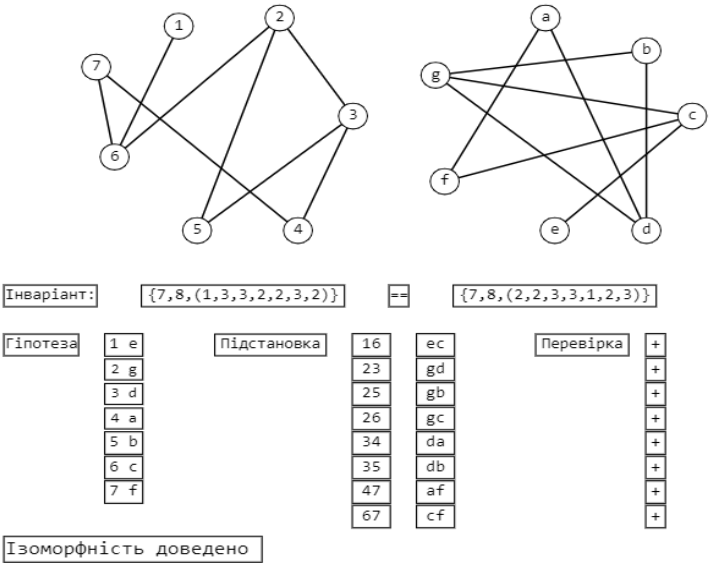
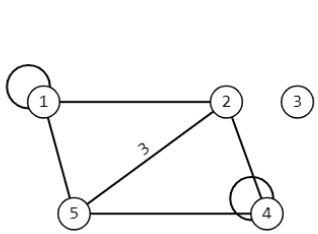


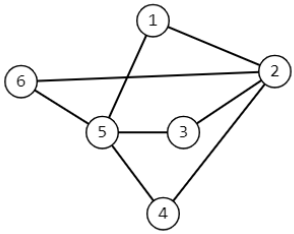
Рисунок 3 – Доведення ізоморфності

4) Вказати для обох даних графів їх матрицю суміжності, побудувати кістякові дерева та відповідні фундаментальні множини циклів і розрізів (рис. 4).



Матриця суміжності

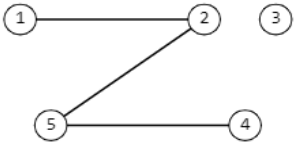
1	1	0	0	1
1	0	0	1	1
0	0	0	0	0
0	1	0	1	1
1	1	0	1	0



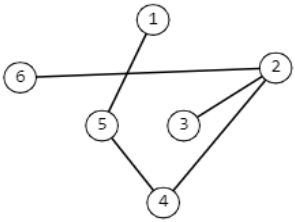
Матриця суміжності

0	1	0	0	1	0
1	0	1	1	0	1
0	1	0	0	1	0
0	1	0	0	1	0
1	0	1	1	0	1
0	1	0	0	1	0

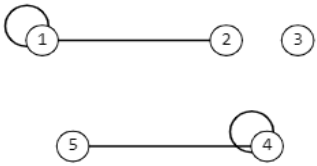
Кістякові дерева - залишається кіль-сть ребер на одне менше за вершини



В даному випадку 3 ребра при 5 вершинах,
так як одна з вершин є ізольованою

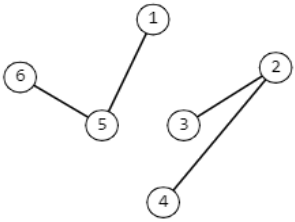


Фундаментальний розріз - коли граф розпадається на частини



Фунд. розр.: (15), (25) та (24)

Цикли: 2452, 1, 42154, ...



Фунд. розр.: (12), (26), (35) та (45)

Цикли: 12351, 65426, ...

Рисунок 4 – Завдання на побудову

Знайти двоїстий граф до першого (плаского) графа та довести, чи є другий граф двоїстим до першого (рис. 5).

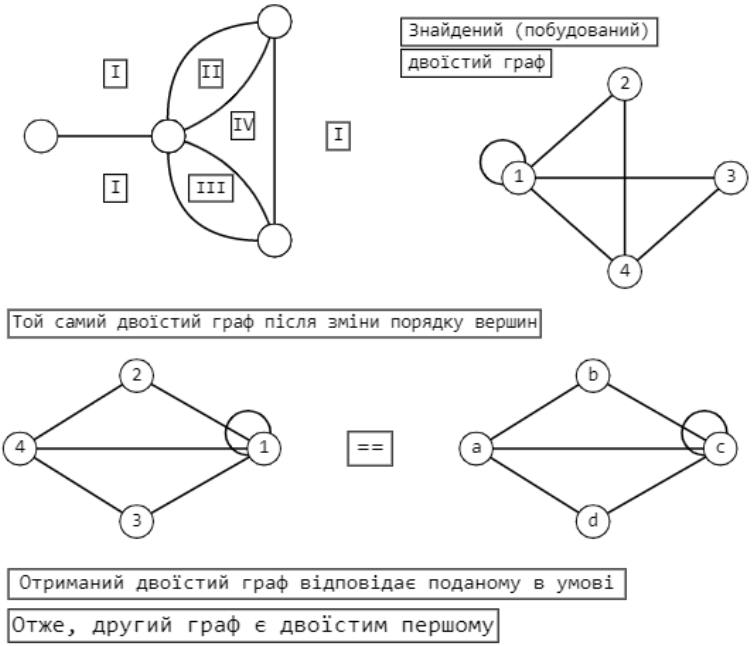


Рисунок 5 – Завдання на двоїстість

Лабораторна робота №5
Тема: Дерева. Обхід графів. BFS, DFS

Мета: опрацювати алгоритми обходу графів.

Теоретичні відомості

Поняття дерева широко використовується в багатьох галузях математики та інформатики. Наприклад, дерева використовуються як обчислювальні інструменти, зручний спосіб зберігання даних, їх сортування або пошуку.

Зв'язний граф без простих циклів називається деревом. Граф, який не містить простих циклів і складається з k зв'язних компонент, називається лісом з k дерев. Дерева та ліси представляють із себе прості діаграми без циклів та орієнтацій. На малюнку 1 показано приклади дерев.

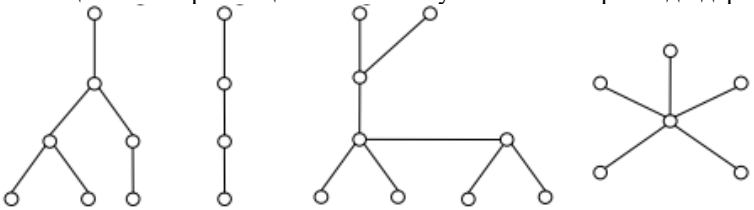


Рисунок 1 – Дерева

Теорема дерева 1. Нехай граф G має n вершин. Тоді наступні твердження еквівалентні:

1) Граф G є деревом;

2) Граф G не містить простих циклів і має (n-1) ребер;

3) Граф G зв'язний і має (n–1) ребер;

4) Граф G зв'язний, але видалення будь-якого ребра роз'єднає його;

5) Будь-які дві вершини графа G з'єднані простим шляхом;

6) Граф G не містить простих циклів, але, додаючи до нього нові ребра, ми отримаємо саме цикл.

У разі, якщо довжина найдовшого ланцюга дерева (діаметр дерева) d(G) = 2n, де $n \in N$, то *центральна вершина v* дерева G має ексцентриситет (найкоротший маршрут до найвіддаленішої вершини), що не перевищує n, $e(v, x) \leq n$, *де* $x \in V$.

Припустимо, що довжина найдовшого ланцюга дерева показана на рис. 2.

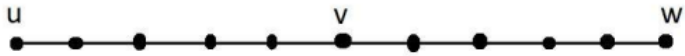


Рисунок 2 – Центральна вершина дерева

Обхід дерев

Багато проблем можна змоделювати за допомогою кореневих дерев. Поширеною є така загальна постановка задачі: виконати задану операцію D над кожною вершиною дерева. Тут D є параметром для більш загального завдання відвідування всіх вершин, або так званого обходу дерева. Враховуючи, що рішенням проблеми є єдиний послідовний процес, який відвідує вершини дерева в певному порядку, їх можна вважати розміщеними одна за одною. Опис багатьох алгоритмів значно спрощується, якщо ми можемо говорити про наступну вершину дерева з урахуванням деякого сортування. Три принципи впорядкування вершин, які природно слідують у структурах дерева. Як і самі деревоподібні структури, їх зручно виражати за допомогою рекурсії. Звертаючись до бінарного дерева, де R – корінь, A та B – ліве та праве піддерева (рис. 3), можна означити такі впорядкування.

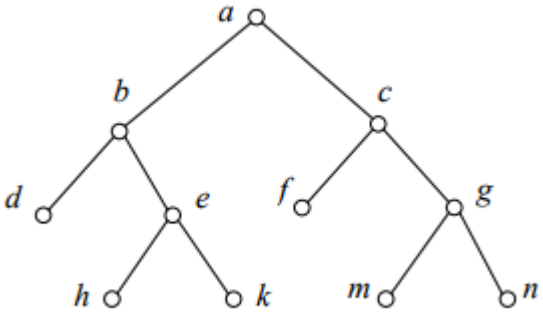


Рисунок 3 – Бінарне дерево

- обхід у ***прямому порядку (preorder)***, або згори донизу: R, A, B (корінь відвідують до обходу піддерев). Для дерева з рис. 2 послідовність вершин буде така: a, b, d, e, h, k, c, f, g, m, n. 2.
- обхід у ***внутрішньому порядку (inorder)***, або зліва направо: A, R, B. Для дерева з рис. 2 послідовність: d, b, h, e, k, a, f, c, m, g, n. 3.
- обхід у ***зворотному порядку (postorder)***, або знизу догори: A, B, R (корінь відвідують після обходу піддерев). Для дерева з рис. 2

послідовність: d, h, k, e, b, f, m, n, g, c, a.

Надзвичайно поширеним застосуванням обходу дерев в інформатиці є подання дерев у вирази (арифметичні, логічні тощо) і побудова різних форм письмових виразів на цій основі. Для ілюстрації суті проблеми зручно використовувати приклади. Розглянемо арифметичний вираз:

$$\left(a + \frac{b}{c}\right)(d - ef)$$

Представимо його у вигляді дерева (рисунок 4). Внутрішні вершини дерева відповідають символам операцій, а листи – операндам.

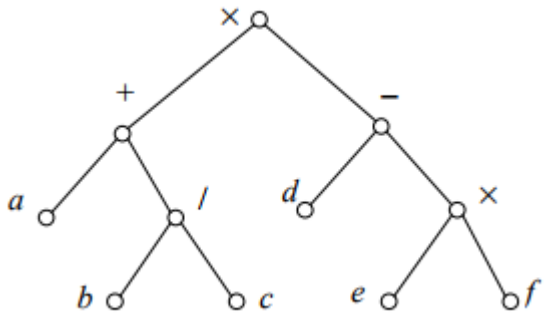


Рисунок 4 – Подання формули у вигляді дерева

Обійдемо це дерево, записуючи символи у вершинах у тому порядку, у якому вони зустрічаються в разі заданого способу обходу. Отримаємо такі три послідовності:

- обхід в прямому порядку – ***префіксний*** (польський) запис:

$$\times + a\ /\ b\ c - d \times e\ f\ ;$$

- обхід у внутрішньому порядку – ***інфіксний*** запис (без дужок, які потрібні для визначення порядку операцій):

$$a + b\ /\ c \times d - e \times f\ ;$$

- обхід у зворотному порядку – ***постфіксний*** (зворотний польський) запис:

$$a\ b\ c\ /\ +\ d\ e\ f\ \times\ -\ \times .$$

Звернемося спочатку до написання інфіксальної форми виразів. Без дужок виникає неоднозначність: один запис може відповідати різним деревам. Щоб уникнути неоднозначності в інфіксних формах, використовуйте дужки щоразу, коли зустрічається операція. Повністю «реконструйований» вираз, отриманий обходом дерева у внутрішньому порядку, називається інфіксною формою запису.

Отже, для дерева на малюнку *інфіксна форма* така: ((a + b c))/(×(d−(e×f))). Наведені вище міркування показують, що писати вирази в інфіксній формі незручно. На практиці використовуються префіксальна і суфіксальна форми, оскільки вони чітко відповідають виразам і не потребують дужок. Ці форми символів називаються польськими символами (на честь польського математика Яна Лукасевича).

Щоб оцінити вираз у польській нотації, відскануйте його справа наліво та знайдіть два операнди і символ оператора перед ними. Операнди та оператори видаляються із запису, виконується операція, а результат записується в тому місці, де були видалені символи. Для прикладу, обчислимо значення виразу в польському записі (^ – означає піднесення до ступеня): «+ × − 4 2 5 ^ 2 / 9 3». За сформульованим правилом виділимо
«/ 9 3», ці символи вилучимо й обчислимо 9 / 3 = 3; результат напишемо на місце вилучення символів «+ × − 4 2 5 ^ 2 3». Продовжимо обчислення. Динаміку процесу відображено в табл. 1.

Таблиця 1.

Крок	Вираз	Виділені символи	Виконання операції
1	+ × – 4 2 5 ^ 2 / 9 3	/ 9 3	9 / 3 = 3
2	+ × – 4 2 5 ^ 2 3	^ 2 3	2^3 = 8
3	+ × – 4 2 5 8	– 4 2	4 – 2 = 2
4	+ × 2 5 8	× 2 5	2 × 5 = 10
5	+ 10 8	+ 10 8	10 + 8 = 18
6	18		

Пошук вшир

На графі існує багато алгоритмів, заснованих на систематичному обході його вершин, або обході вершин, під час якого кожна вершина отримує унікальний порядковий номер. Існує два основних алгоритми обходу графа або пошуку графа – пошук у ширину та пошук у глибину.

Почнемо з методу *Breadth First Search* або *BFS*. Нехай $G = (V, E)$ – простий зв'язний граф, усі його вершини позначені попарно різними символами. Під час процесу пошуку в глибину вершини графа G отримують номери (номери BFS). Під час роботи алгоритму для зберігання колекції використовується структура даних, яка називається чергою. Лише перший елемент, доданий до черги, може бути видалений з черги: черги працюють за принципом «першим прийшов, першим вийшов» (***FIFO***). Елемент включається в кінці черги і виключається з початку черги.

Процедура пошуку вшир BFS наводиться нижче. Вона приймає на

вхід два параметри: граф G (який може бути представлений будь-яким зручним способом: матрицями суміжності та інцидентності, списком суміжності) та початкова вершина s. Процедура починає обхід графу з вершини s, додаю у чергу Q нові вершини. Робота закінчується, коли черга Q стає порожньою.

```
BFS(граф G, початкова вершина s)
1.  позначити вершину s як відвідану та приписати їй номер BFS[s] ← 1
2.  внести в чергу Q вершину s: Q ← [s]
3.  k ← 1
4.  while Q ≠ ∅
5.      v ← перша вершина в Q
6.      для кожного ребра (v, u) в G:
7.          if вершина u ще не відвідана (тобто номер BFS[u] не визначений)
8.              k ← k + 1
9.              позначити u як відвідану та приписати їй номер BFS[u] ← k
10.             додати u в кінець черги Q
```

Лістинг 1 – Процедура BFS пошуку вшир в графі

Щоб зробити результати алгоритму однозначними, вершини, суміжні з вершиною v, аналізуються в порядку зростання їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму зручно відобразити за допомогою таблиці з трьома колонками: вершина, номер BFS, вміст черги. Ця таблиця називається протоколом обходу графа в ширину.

Для прикладу виконаємо обхід графу, який зображено на рис. 5, починаючи з вершини b. Протокол обходу наведено в таблиці 2.

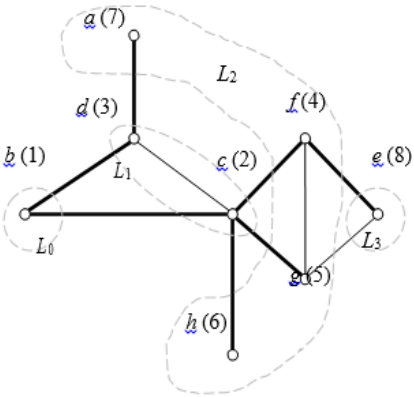


Рисунок 5 – Обхід графу

Таблиця 2.

Вершина	BFS-номер	Черга
b	1	*b*
c	2	*bc*
d	3	*bcd*
-	-	*cd*
f	4	*cdf*
g	5	*cdfg*
h	6	*cdfgh*
-	-	*dfgh*
a	7	*dfgha*
-	-	*fgha*
8	*e*	*fghae*
-	-	*ghae*
-	-	*hae*
-	-	*ae*
-	-	*e*
-	-	∅

У процесі роботи алгоритму пошуку вшир будується дерево *T* пошуку. Це дерево є підграфом графу *G* і складається з тих ребер (*v*, *u*), які були розглянуті на кроці 6 процедури BFS. Якщо граф *G* зв'язаний, то дерево *T* буде включати в себе всі вершини графу. Таке дерево називається **покриваючим деревом графу**. На рис. 3 ребра дерева *T* позначені потовщеними лініями.

Досліджуючи алгоритм пошуку вшир, можна прийти до наступного висновку.

Лема 1. Якщо вершина *v* графу *G* зустрілась під час роботи процедури BFS (*G*, *s*), то в графі *G* існує шлях від *s* до *v*.

Доведення леми тривіальне та ґрунтується на аналізі коду процедури BFS (див. лістинг 1).

Найкоротші відстані на основі пошуку вшир

Пошук вшир може застосовуватись не тільки для обходу графу, але й для різноманітних практичних задач. По-перше, метод пошуку вшир в графі *G* дозволяє знайти найкоротші шляхи від початкової вершини *s* до будь-якої іншої досяжної з неї вершини *v* у графі.

Щоб реалізувати можливість обчислення найкоротших відстаней від початкової вершини *s* до всіх інших досяжних вершин, процедура

BFS потребує мінімальних змін. Необхідно ввести додатковий параметр *dist* [*v*], для будь-якої вершини, ініціалізувавши його 0 для початкової вершини *s* та +1 для всіх інших вершин графу. А також після рядка 8 процедури BFS необхідно додати інструкцію: *dist*[*v*]<*dist*[*v*]+1.

Пошук углиб

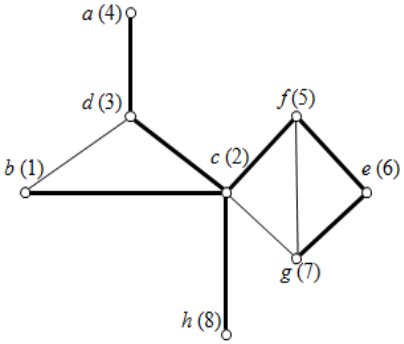
Ще один метод обходу графу – алгоритм **пошуку углиб** або *DFS*-метод (*depth first search*). У процесі роботи алгоритму всім вершинам графу надаються номери (DFS-номери). На відміну від методу пошуку вшир, пошук углиб використовує структуру даних стек для збереження вершин-кандидатів. Зі стеку можна вилучити тільки той елемент, який було додано до нього останнім: стек працює за принципом «останнім прийшов – першим вийшов» (*lastin, firstout* – скорочено LIFO). Інакше кажучи, додавання й вилучення елементів у стеку відбувається з одного кінця, який називається верхівкою стеку.

Процедура DFS повторює процедуру BFS, але відрізняється від неї тим, що замість черги тут використовується стек.

```
DFS(граф G, початкова вершина s)
1.  позначити вершину s як відвідану та приписати їй номер DFS[s] ← 1
2.  внести в стек S вершину s: S ← [s]
3.  k ← 1
4.  while S ≠ ∅
5.      v ← вершина в голові стеку S
6.      if існує ребро (v, u) в G таке, що вершина u ще не відвідана
7.          k ← k + 1
8.          позначити u як відвідану та приписати їй номер DFS[u] ← k
9.          додати u в голову стеку S
10.     else
11.         видалити вершину v з голови стеку S
```

Лістинг2 – Процедура DFS пошуку вглиб в графі

Аналогічно до пошуку DFS, для відстеження роботи процедури DFS використовують протокол обходу. Виконаємо обхід графу з рис. 4, починаючи так само з вершини *b*. Результат обходу зображено на рис. 6, протокол обходу – в таблиці 3.



Таблиця 3.

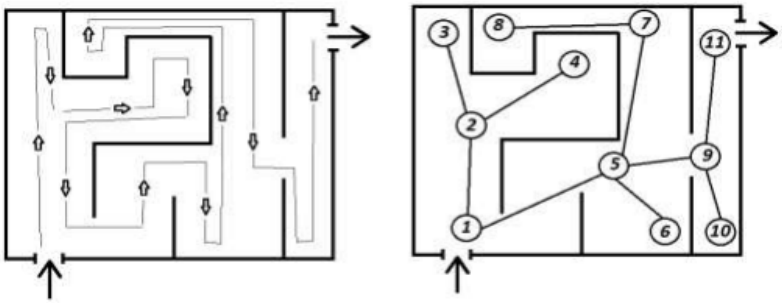
Вершина	DFS-номер	Стек
b	1	*b*
c	2	*bc*
d	3	*bcd*
a	4	*bcda*
-	-	*bcd*
-	-	*bc*
f	5	*bcf*
e	6	*bcfe*
g	7	*bcfeg*
-	-	*bcfe*
-	-	*bcf*
-	-	*bc*
h	8	*bch*
-	-	*bc*
-	-	*b*
-	-	∅

Рисунок 6 – результат обходу

У процесі роботи алгоритму пошуку вглиб, так само, як і у випадку пошуку вшир, будується дерево T пошуку. Це дерево є підграфом графу G і складається з тих ребер (v, u), які були розглянуті на кроці 6 процедури DFS. Якщо граф G зв'язаний, то дерево T буде включати в себе всі вершини графу. На рис. 4 ребра дерева T позначені потовщеними лініями.

Процедура пошуку вглиб у такому вигляді застосовується у багатьох практичних задачах. Найпростішим прикладом роботи алгоритму є пошук виходу з лабіринту – якщо ми можемо рухатися вперед, ми рухаємося вперед, шукаючи вихід. В іншому разі повертаємося на перехрестя і рухаємося за іншим маршрутом. Однак, на відміну від попереднього алгоритму, пошук йде не за рівнями, а за маршрутом.

Приклад 1. Розглянемо лабіринт та його подання у вигляді дерева, як показано на рис. 7. Для знаходження виходу з лабіринту в найгіршому випадку доведеться пройти за всіма шляхами, які вказано стрілками в лабіринті (рис. 7. а).



а) б)

Рисунок 7 – Лабіринт та його граф

Для алгоритмізації цієї задачі, лабіринт зобразимо у вигляді графа, в даному дерева (рис. 7. б). При цьому вершини дерева ставляться у точках зміни шляху, а ребрами є шлях в лабіринті – даний підхід використовується для будь-якого лабіринту.

Алгоритми пошуку в глибину працює наступним чином: починаючи з входу, рухаємося, наприклад , завжди вліво. Тоді маршрут пошуку виходу в лабіринті на рис. 7. б) має наступний вигляд: 1-2-3-2-4-2-1-5-7-8-7-5-9-11. До речі, для графа лабіринту на рис. 7. б) у випадку вибору руху завжди вправо, маршрут виходу буде меншим і має наступний вигляд: 1-5-6-9-10-9-11.

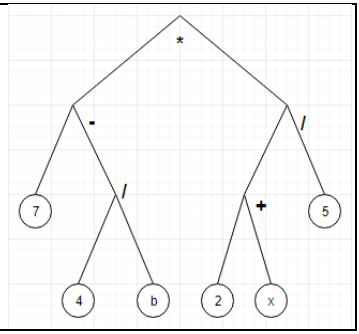
У вищенаведеному прикладі маршрут виходу з лабіринту є частиною маршруту обходу графа в глибину, який включає всі вершини графа.

Приклад завдання

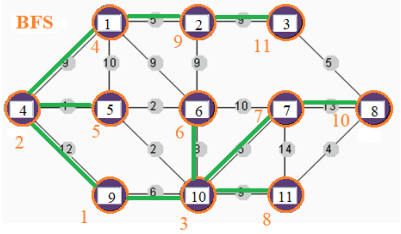
1. Побудувати три види обходів дерев.

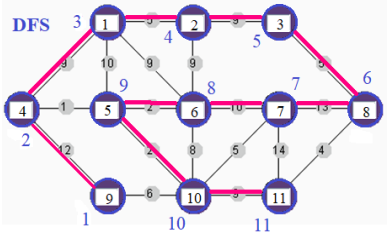
$$(7 - 4/b)\left(\frac{2 + x}{5}\right);$$

- обхід у *прямому порядку (preorder)*
  **** - 7 / 4 b / + 2 x 5***
- обхід у *внутрішньому порядку*
  7 - 4 / b * 2 + x / 5
- обхід у *зворотному порядку*
  7 4 b / - 2 x + / 5*



2. Для зваженого графа у варіанті побудувати дерева за алгоритмами BFS, DFS за початкову вершину взято 9.





Вер-шина	BFS-номер	Черга
9	1	9
4	2	9,4
10	3	9,4,10
–	–	4,10
1	4	4,10,1
5	5	4,10,1,5
–	–	10,1,5
6	6	10,1,5,6
7	7	10,1,5,6,7
11	8	10,1,5,6,7,11
–	–	1,5,6,7,11
2	9	1,5,6,7,11,2
–	–	5,6,7,11,2
–	–	6,7,11,2
–	–	7,11,2
8	10	7,11,2,8
–	–	11,2,8
–	–	2,8
3	11	2,8,3
–	–	8,3
–	–	3
–	–	∅

Вер-шина	DFS-номер	Стек
9	1	9
4	2	4,9
1	3	1,4,9
2	4	2,1,4,9
3	5	3,2,1,4,9
8	6	8,3,2,1,4,9
7	7	7,8,3,2,1,4,9
6	8	6,7,8,3,2,1,4,9
5	9	5,6,7,8,3,2,1,4,9
10	10	10,5,6,7,8,3,2,1,4,9
11	11	11,10,5,6,7,8,3,2,1,4,9
–	–	10,5,6,7,8,3,2,1,4,9
–	–	5,6,7,8,3,2,1,4,9
–	–	6,7,8,3,2,1,4,9
–	–	7,8,3,2,1,4,9
–	–	8,3,2,1,4,9
–	–	3,2,1,4,9
–	–	2,1,4,9
–	–	1,4,9
–	–	4,9
–	–	9
–	–	∅

Лабораторна робота №6
Тема: Алгоритм Крускала-Прима. Алгоритм Дейкстри

Мета: опрацювати алгоритми знаходження кістякового дерева найбільшої на найменшої ваги; знаходження шляху найменшої ваги між вершинами, відновлення маршруту за індексами з алгоритму Дейкстри.

Теоретичні відомості

В теперішній час із розвитком різноманітних мереж: нейронних, мережі Інтернет, транспортних, стільникових, електричних і т.п., виникає завдання створення таких мереж із **мінімальними** витратами, а також їхнє ефективне використання. Різні компанії витрачають величезні гроші на розвиток логістики та вирішення подібних завдань. Але всі ці завдання зводяться до одного завдання теорії графів – **побудови мінімального кістякового дерева**. Для прикладу можна взяти карту України, де вершами будуть населені пункти, а ребра – це дороги між цими населеними пунктами, вийшов зв'язний граф. Метрополітен або залізниця теж є графом (деревом).

Для вирішення подібних завдань існують алгоритми Крускала та Прима, але існують й інші, наприклад, алгоритм Борувки. Алгоритм був вперше опублікований у 1926 році Отакаром Борувкою як метод знаходження оптимальної електричної мережі.

Алгоритм Крускала

Алгоритм Крускала – алгоритм побудови мінімального остовного дерева зваженого неорієнтованого графа. Алгоритм вперше був описаний Джозефом Крускалом у 1956 році.

Розглянемо зважений зв'язний граф $G = (V, E)$, де V набір вершин, а E – набір ребер, кожне ребро має задану вагу (довжину). Тоді множина ациклічних ребер, яка з'єднує всі вершини графа і має найменшу сумарну вагу, називається мінімальним скелетним (або скелетним) деревом.

Алгоритм Крускала спочатку створює вироджений ліс, що містить V дерев, кожне дерево містить вершину. Далі виконується операція по з'єднанню двох дерев, використовуючи для цього найкоротші ребра, поки не буде сформовано єдине дерево. Це дерево буде мінімальним деревом-скелетом.

Рівень складності. Алгоритм Крускала (і алгоритм Прима) – класичний алгоритм розв'язання задачі знаходження мінімального остовного

дерева. Якщо використовується найшвидша реалізація, її час роботи становить Θ(E*logE).

Основна частина часу витрачається на сортування ребер за вагою. Так само, як і в простій версії алгоритму Крускала, відсортуємо всі ребра за вагою у неспадному порядку. Потім помістимо кожну вершину в своє дерево (тобто свою множину), за допомогою виклику функції перебираємо всі ребра (у порядку сортування) і для кожного ребра за $\Theta(1)$ визначаємо, чи належать його кінці різних деревам (за допомогою двох викликів за $\Theta(1)$). Нарешті, об'єднання двох дерев буде здійснюватися викликом функції. Таким чином ми отримуємо складність алгоритму оцінену $\Theta(MlogN+N+M)= \Theta(MlogN)$.

Алгоритм Дейкстри

Алгоритм Дейкстри вирішує завдання про знаходження найкоротших шляхів з однієї вершини до всіх інших для зваженого орієнтованого графа $G = (V, E)$ з вихідною вершиною с, в якому ваги всіх ребер невід'ємні ($(U, V) \geq 0$ для всіх $(i,v) \in E$).

Формальне пояснення. Алгоритм Дейкстри підтримує набір S V, що складається з вершин v, для яких знайдено $\delta(S, V)$. Алгоритм вибирає вершину $u \epsilon V \backslash S$ з мінімальним $d[u]$, додає u до набору S і перераховує всі ребра з U, після чого цикл повторюється. Вершини, які не входять до S, зберігаються в черзі Q, їхній пріоритет визначається значенням функції d. Припустимо, що граф заданий за допомогою списку суміжних вершин. Тобто, ми призначаємо кожній вершині мітку з v – мінімальною відомою відстанню від цієї вершини. Алгоритм працює крок за кроком – на кожному кроці він «відвідує» вершину і намагається зменшити мітку. Алгоритм завершується, коли відвідані всі вершини.

Ініціалізація. Припустимо, що мітка самої вершини дорівнює 0, а мітки інших вершин встановлені на нескінченність. Це відображає той факт, що відстані до інших вершин залишаються невідомими. Усі вершини графа позначені як невідвідані.

Крок алгоритму. Якщо всі вершини мають мітку, алгоритм завершується. В іншому випадку з ще не відвіданих вершин вибирається вершина v, що має найменше значення функції d. Ми розглядаємо маршрути, в яких v є передостаннім пунктом. Вершини, з'єднані з вершиною v ребрами, суміжними. Для всіх суміжних ребер оновимо довжину шляху, що дорівнює сумі поточної мітки u і довжини ребра, що з'єднує з цим сусідом. Якщо отримана довжина менше мітки сусіда, замінимо мітку цієї вершини. Розглянувши всіх сусідів, відмітимо вершину u як відвідану і повторимо крок.

Оскільки алгоритм Дейкстри всякий раз вибирає для обробки вершини з найменшою оцінкою найкоротшого шляху, можна сказати, що він відноситься до жадібних алгоритмів.

Час роботи алгоритму Дейкстри

Складність алгоритму Дейкстри залежить від методу, який використовується для знаходження вершини v, а також від методу, який використовується для зберігання набору невідвіданих вершин і методу, який використовується для оновлення міток. Позначимо через n кількість вершин у графі G і через t кількість ребер.

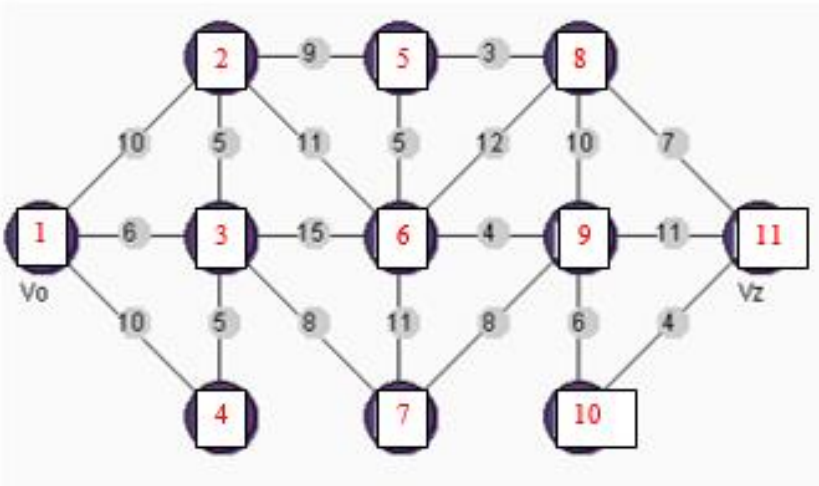
• У найпростішому випадку, коли сканується весь набір вершин для пошуку вершини з мінімальним *d[v]* і використовується масив для збереження значення d, час роботи алгоритму становить $\Theta(n^2 + m)$. Основний цикл виконується приблизно *n* разів, щоразу потрібно приблизно *n* операцій, щоб знайти мінімум, плюс кількість зависання (зміни в мітках), яка не перевищує кількість ребер у вихідному графі.

• Для розріджених графів (тобто таких, для яких *m* набагато менше n^2) невідвідані вершини можна зберігати в списку, а в якості ключа використовувати значення *d [i],* тоді час вилучення вершини з u стане увійти *logn*, при тому, що час модифікації *d[i]* зросте до *logn*. Оскільки цикл виконується порядку *n* разів, а кількість релаксації не більше *m*, швидкість роботи такої реалізації $\boldsymbol{\Theta(n*\log n + m*\log n)}$.

• Якщо ряд Фібоначчі використовується для зберігання невідвіданих вершин, видалення займає в середньому $\Theta(\log n)$, а дельта часу займає в середньому $\Theta(1)$, тоді час роботи алгоритму становитиме $\boldsymbol{\Theta(n*\log n+m)}$.

Приклад завдання

1. Побудувати мінімальне остовне дерево, використовуючи алгоритми Крускала-Прима.



Складемо список ребер з їх довжинами

ступені вершин: 3+4+5+2+3+6+3+4+5+2+3=40; 40/2=20 ребер.

(1,2)	(1,3)	(1,4)	(2,5)	(2,6)	(2,3)	(3,4)	(3,7)	(3,6)	(5,8)
10	6	10	9	11	5	5	8	15	3

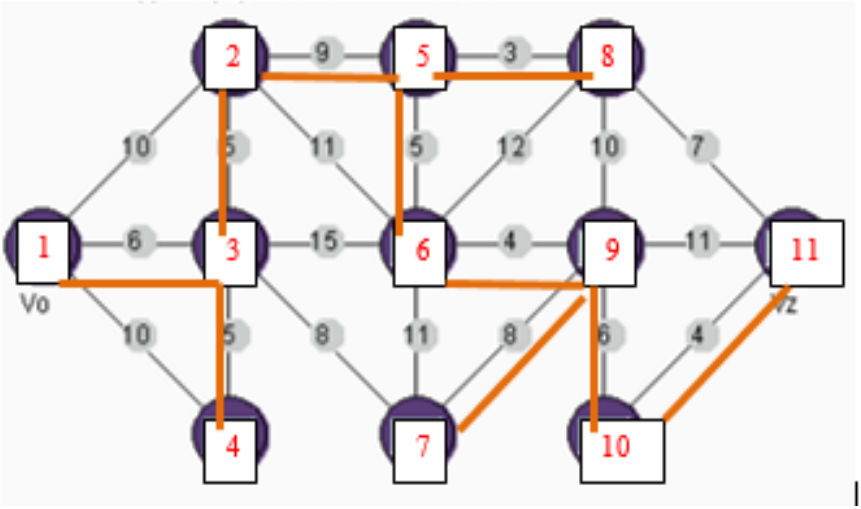
(5,6)	(6,8)	(6,9)	(6,7)	(7,9)	(8,11)	(8,9)	(9,10)	(9,11)	(10,11)
5	12	4	11	8	7	10	6	11	4

Впорядкуємо ребра по зростанню їх ваги (довжин).

3	4	4	5	5	5	6	6	7	8
(5,8)	(6,9)	(10,11)	(2,3)	(3,4)	(5,6)	(1,3)	(9,10)	(8,11)	(3,7)

8	9	10	10	10	11	11	11	12	15
(7,9)	(2,5)	(1,2)	(1,4)	(8,9)	(2,6)	(6,7)	(9,11)	(6,8)	(3,6)

Вибираємо ребра, починаючи з найменшого, зважаючи на умови непояви циклу!!!



Ребро (8, 11) утворює цикл, тому пропускається. Всі вершини з'єднані.

Отримане кістякове дерево найменшої ваги.

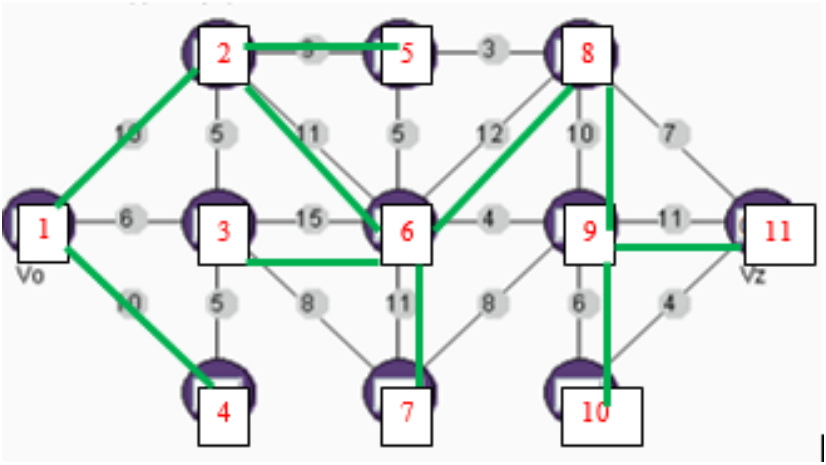
$$W(G)=6+5+5+9+6+4+8+6+3+4=54.$$

2. Побудувати максимальне остовне дерево використовуючи алгоритми Крускала-Прима.

Аналогічно побудувати кістякове дерево найбільшої ваги. Впорядкуємо ребра за спаданням їх ваги (довжин).

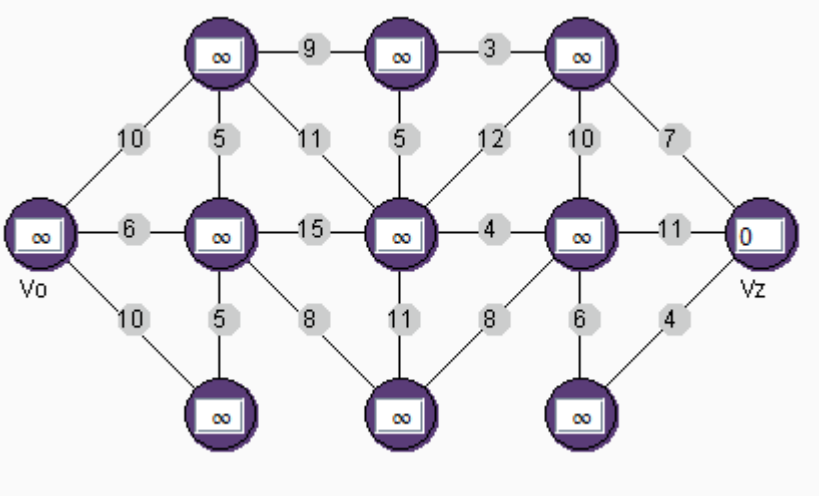
15	12	11	11	11	10	10	10	9	8
(3,6)	(6,8)	(9,11)	(6,7)	(2,6)	(8,9)	(1,4)	(1,2)	(2,5)	(7,9)

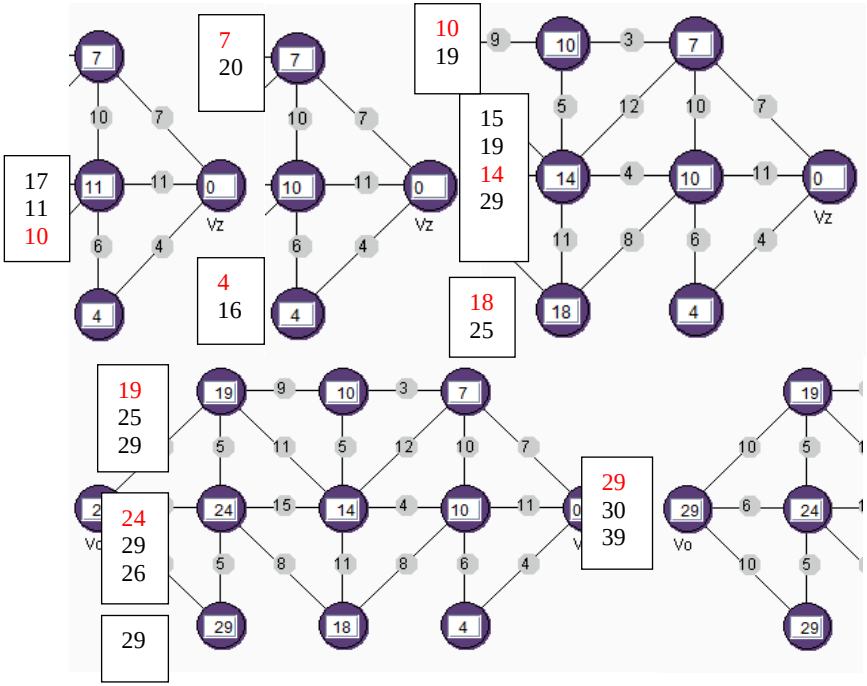
8	7	6	6	5	5	5	4	4	3
(3,7)	(8,11)	(9,10)	(1,3)	(5,6)	(3,4)	(2,3)	(10,11)	(6,9)	(5,8)

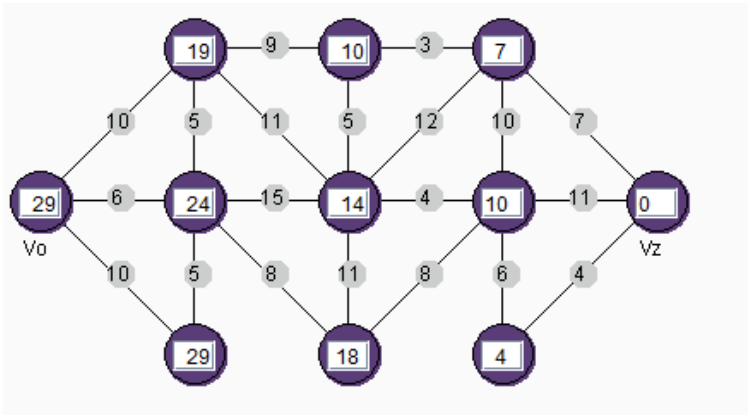


W(G)=10+10+9+11+15+11+12+10+6+11=104,

3. Розставити індекси по вершинах та знайти найкоротший шлях з вершини V_z до вершини V_o, використовуючи алгоритм Дейкстри.



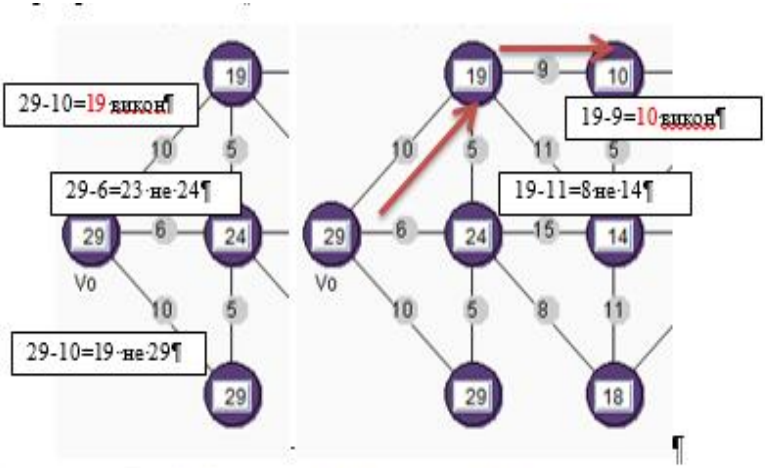


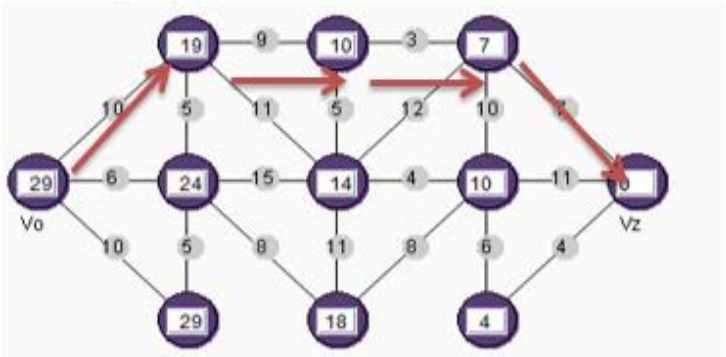


Найкоротша відстань з вершини V_z до вершини V_o збігається з індексом V_o дорівнює $d(v_z,v_0) = 29$.

4. Знаходження найкоротшого шляху

Для знаходження найкоротшого шляху необхідно починати з вершини v_0 і вибирати з доступних вершин таку, індекс якої співпадав з різницею індексу v_0 і довжиною ребра між ними.





Найкоротша відстань з вершини V_z до вершини V_o задається маршрутом, зображеним стрілками.

Лабораторна робота №7
Тема: Представлення дерева кодом Прюфера

Теоретичні відомості

Усі відомі способи представлення графів є дійсними й у випадку дерев. Проте дерево є особливим графом, для якого можливі інші способи представлення.

Вибір кодування дерева залежить від розв'язуваної теоретичної або технічної задачі. Серед усіх можливих кодувань природно відшукати оптимальні за якоюсь якістю розв'язки. Кодування Прюфера застосовується до вільних дерев (неорієнтованих дерев, тобто дерев, в яких немає виділеного кореня).

Наведемо алгоритм кодування поміченого дерева за Прюфером:

1) Знайти висячу вершину з мінімальним номером і.

2) Записати в код Прюфера вершину, суміжну з і.

3) Видалити вершину і з дерева. Якщо дерево не порожнє, то перейти до п. 1.

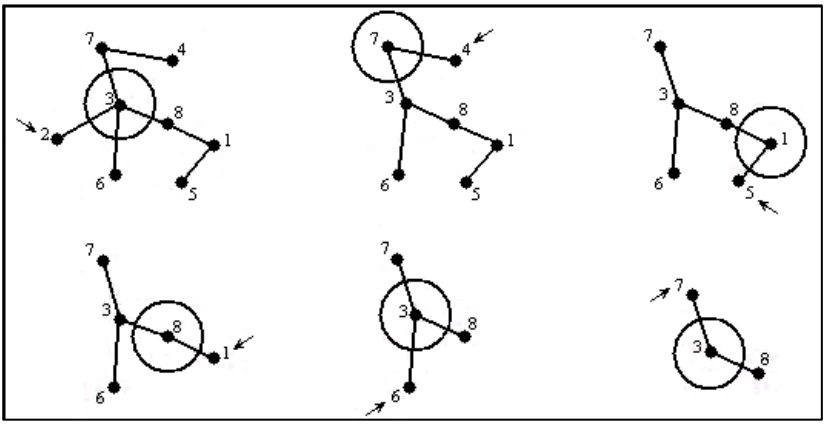
У 1918 році Прюфер запропонував метод представлення дерева з $n > 2$ вершинами, який тепер називають **кодом Прюфера**. Алгоритм укладання коду Прюфера є таким:

```
   PRUF{G,Code)
 Перенумерувати вершини дерева G довільним чином.
 while (у дереві не залишилось одне ребро)
  Знайти висячу вершину з найменшим номером.
  Видалити знайдену вершину разом з інцидентним ребром.
  Занести номер вершини, суміжної видаленій, до коду Code.
 end {while}
end {PRUF}
```

Після укладання коду в дереві завжди залишаються дві вершини, тому довжина коду Прюфера для будь-якого дерева буде на два менше числа вершин дерева: $K = n - 2$. Отже, маючи код, можна визначити кількість вершин у графі: їх завжди буде на дві більше кількості елементів коду.

Номери вершин у коді можуть повторюватися, а у випадку зіркового графа вони взагалі будуть однаковими.

На наступному рисунку показаний процес укладання коду Прюфера для дерева.



Результатом роботи алгоритму є код **К = 3 7 1 8 3 3**.

Відновлення дерева з коду виконується за допомогою антикоду. Антикод – це впорядкований кортеж, складений із номерів вершин, які не увійшли до коду. Для розглянутого прикладу антикод **А = 2 4 5 6**.

Розпакування коду Прюфера Р проводиться за таким алгоритмом.

Введемо список (вектор) деяких елементів $N = [\, a_1, ..., a_n \,]$ і операцію укорочення списку на один перший елемент, позначивши її зірочкою N^*.

$N^* = [a_2, a_3, ..., a_n]$. Тоді

1) $A = P$, $N = [1, ..., n]$;
2) $x = \min N$, $x \notin A$, $y = a_1$;
3) вершини y та x з'єднати ребром;
4) $N = N \setminus x$, $A = A^*$;
5) якщо $|N| = 2$, то з'єднати дві останні вершини, n_1 і n_2, і завершити процедуру; в іншому випадку повернутися до п. 2.

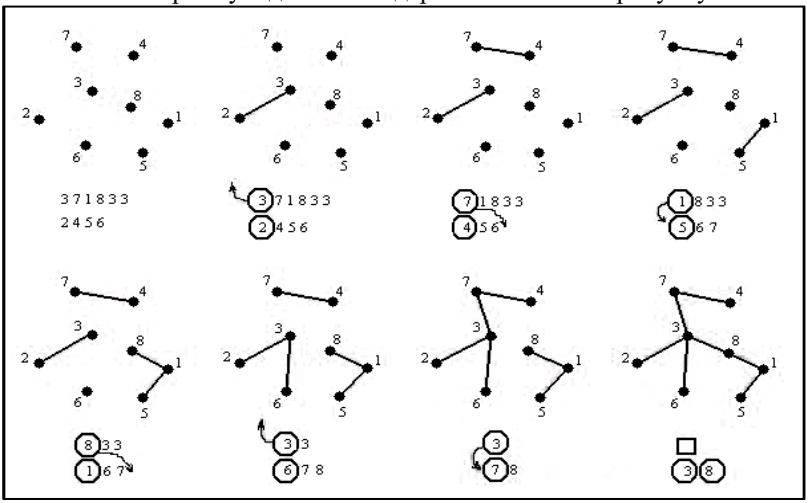
Основна вимога до алгоритмів кодування – однозначність відновлення інформації.

Алгоритм відновлення дерева за кодом Прюфера зображений нижче:

TREE(Code, Tree)
Встановити кількість вершин дерева: **n = Length(Code) + 2**.
Отримати антикод**AntiCode**.
Створити **n** вершин і дати їм номери.
while (Code> 0)
Витягнути перші елементи **u** і **v** з коду й антикоду.

```
  Відновити ребро (u, v) в дереві.
  Видалити елемент v за антикоду.
  if (елемент u ще зустрічається далі в коді)
    then
      Витягнутий елемент коду u відкинути
    else
      if (елемент u менший за всі елементи антикоду)
        then
          Поставити елемент на початок антикоду
        else
          Витягнутий елемент коду u додати в кінець антикоду.
        end {if}
    end {if)
 end{while}
 З'єднати ребром останні дві вершини, що залишились в антикоді.
end {TREE}
```

Робота алгоритму відновлення дерева показана на рисунку.



Приклад завдання

1. За кодом Прюфера згенерувати антикод та відновити граф.
2. Малюнок графа.
3. Побудувати його матриці суміжності вершин.

Примітка: *Нумерація вершин починається з 0. Граф має 21 вершину.*

1 частина.

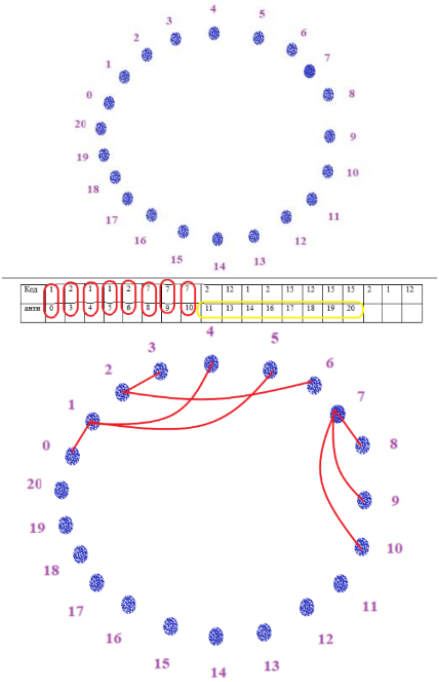
код 1, 2, 1, 1, 2, 7, 7, 7, 2, 12, 1, 2, 15, 12, 15, 15, 2, 1, 12.

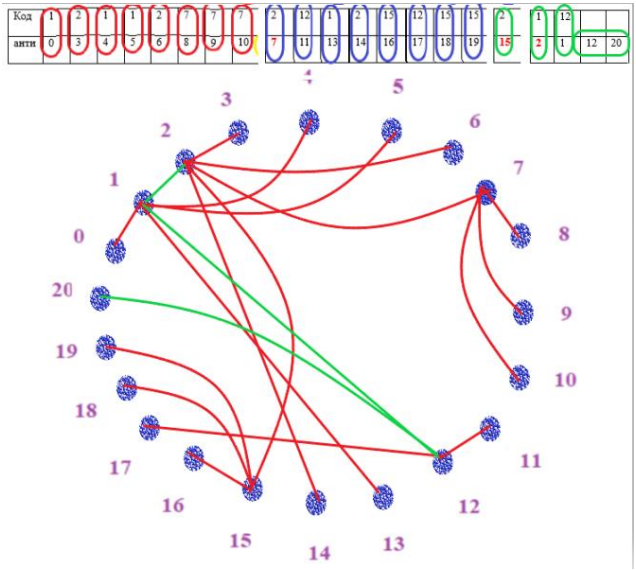
Антикод 0**12**3456**7**8910111**12**1314**15**1617181920

Код	1	2	1	1	2	7	7	**7**	2	12	1	2	15	12	15	**15**	**2**	1	12		
анти	0	3	4	5	6	8	9	10	11	13	14	16	17	18	19	20					

Код	1	2	1	1	2	7	7	**7**	2	12	1	2	15	12	15	**15**	**2**	**1**	**12**		
анти	0	3	4	5	6	8	9	10	**7**	11	13	14	16	17	18	19	**15**	**2**	**1**	**12**	20

2 частина





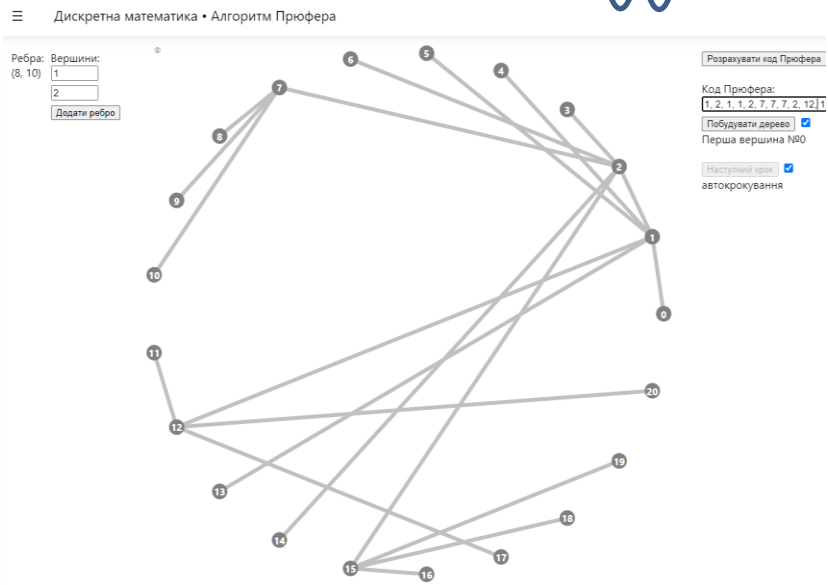
3 частина

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0		**1**																			
1	**1**		**1**		**1**	**1**							**1**	**1**							
2		**1**		**1**			**1**	**1**							**1**	**1**					
3			**1**																		
4		**1**																			
5		**1**																			
6			**1**																		
7			**1**						**1**	**1**	**1**										
8								**1**													
9								**1**													
10								**1**													
11													**1**								
12		**1**										**1**						**1**			**1**
13		**1**																			
14			**1**																		
15			**1**														**1**	**1**	**1**	**1**	
16																**1**					
17													**1**			**1**					
18																**1**					
19																**1**					
20													**1**								

1, 2, 1, 1, 2, 7, 7, 7, 2, 12, 1, 2, 15, 12, 15, 15, 2, 1, 12.

0 **1** **2** 3 4 5 6 **7** 8 9 10 11 **12** 13 14 **15** 16 17 18 19 20

Код	~~1~~	~~2~~	~~1~~	~~1~~	~~2~~	~~7~~	~~7~~	7	~~2~~	~~12~~	~~1~~	~~2~~	~~15~~	~~12~~	~~15~~	15	2	1	12		
анти	0	3	4	5	6	8	9	10	7	11	13	14	16	17	18	19	15	2	1	12	20



Лабораторна робота №8
Тема: Побудова найлегшого бінарного дерева за алгоритмом Хаффмана

Теоретичні відомості.

Подаймо алгоритм побудови бінарного дерева (префіксного коду) з мінімальною вагою за відомими вагами листків (літер).

Алгоритм Хаффмана (Haffman's algorithm)

1. Відсортуйте ваги (відносну частоту) у порядку неспадання.

2. Видаліть найменші елементи f_j і f_k з вагового (частотного) масиву та додайте $(f_j + f_k)$.

3. Створюємо бінарне дерево, в якому корінь і два його прямі нащадки (символи або раніше побудовані бінарні дерева) відповідають витягнутим вагам (частотам) f_j *і* f_k*:*

   – дуга від кореня до найменшого нащадка встановлюється на 0;

   – дуга від кореня до більшого кореня встановлюється відповідно до 1;

   – додаткові ваги (частоти) $(f_j + f_k)$ відповідають кореням побудованого бінарного дерева.

4. Виконуйте кроки 1-3, доки в масиві невидаленої ваги (частоти) не залишиться лише один елемент.

5. Формуємо код Хаффмана, послідовність нулів і одиниць, вирівняних по дузі маршруту (шляху) від кореня останнього побудованого дерева до заданого елемента (коду шляху).

Зауваження. *Для однозначності алгоритму Хаффмана на кроці 3 потрібно задати правило відповідності різним дугам нуля чи одиниці. Наприклад, з використанням алфавітного порядку для символу чи послідовності символів, що є листками побудованих дерев. Наприклад, зверху 0, знизу 1, зліва 0, праворуч 1.*

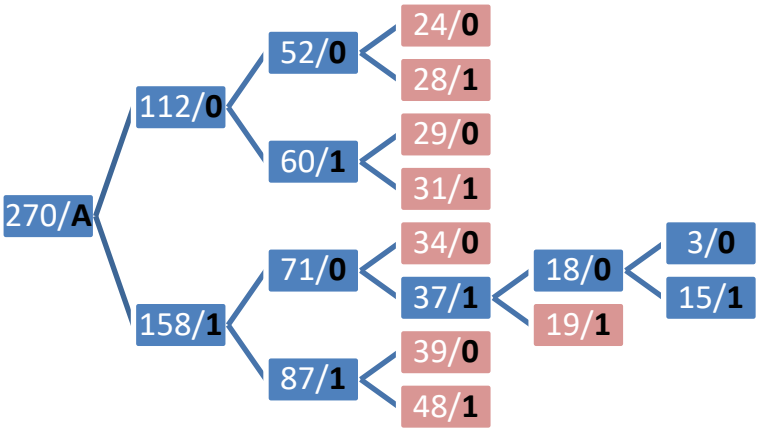
Означення. *Нехай кожна літера* a_j *абетки A має додатну вагу* f_j*, а відповідний елемент двійкового коду має довжину* l_j*. Тоді* ***вагою коду*** *називають суму:* $f_1l_1 + f_2l_2 + f_3l_3 + \cdots$.

Лема 1. *Для довільної скінченної множини листків (символів) і довільного розподілу ваг (частот) серед цих символів існує бінарне дерево мінімальної ваги з цими листками (символами).*

Приклад. За алгоритмом Хаффмана побудуємо бінарне дерево та обчислимо його вагу.

A = { 24, 31, 15, 39, 34, 19, 48, 3, 29, 28 }

1 крок впорядкування A={ 3, 15, 19, 24, 28, 29, 31, 34, 39, 48 } 2 крок сумування A={ 18, 19, 24, 28, 29, 31, 34, 39, 48 } Повторення A={ 37, 24, 28, 29, 31, 34, 39, 48 } A={ 24, 28, 29, 31, 34, 37, 39, 48 } A={ 52, 29, 31, 34, 37, 39, 48 } A={ 29, 31, 34, 37, 39, 48, 52} A={ 60, 34, 37, 39, 48, 52}	A={34, 37, 39, 48, 52, 60} A={71, 39, 48, 52, 60} A={39, 48, 52, 60, 71} A={87, 52, 60, 71} A={52, 60, 71, 87} A={112, 71, 87} A={71, 87,112} A={158, 112} A={112, 158} A={270}



A	код	Довжина коду	Вага
3	10100	5	15
15	10101	5	75
19	1011	4	76
24	000	3	72
28	001	3	84
29	010	3	87
31	011	3	93
34	100	3	102
39	110	3	117
48	111	3	144
		Вага	**865**

При введенні в комп'ютер кожна літера кодується відповідною цифрою. Відбувається це найпростішим способом – кожному символу

відповідає двійкове число, вибране з кодової таблиці. Кодова таблиця – це таблиця, яка задає відповідність між символами та двійковими числами (кодами символів). Кодова таблиця ASCII (Американський стандартний код для обміну інформацією) використовується як основа для кодування символів у персональних комп'ютерах.

Коли ви натискаєте клавішу на клавіатурі, електронні схеми клавіатури генерують двійковий код для цього символу, який визначається кодовою таблицею. Наприклад, коли ви натискаєте клавішу A, генерується код 01000001.

Існують різні стандарти кодування символів, проте всі символи кодуються 8 бітами.

Приклад завдання

За алгоритмом Хаффмана побудувати бінарне дерево та підрахувати його вагу.

1. В звіт малюнок отриманого дерева для кодування.
2. Підстановка, отриманий код та аналіз отриманого коду.

Автобус їде по алеї.
Акація цвіте в садку.
А ми зібралися під нею
Абетку вивчити легку.

а	8	б	З	в	5	г ґ	1	д	З	е	6	є		ж	
з	1	и	5	і	4	ї	2	й		к	4	л	З	м	1
н	2	о	2	п	2	р	1	с	З	т	4	у	4	ф	
х		ц	2	ч	1	ш		щ		ь		ю	1	я	2

Маємо 70 симв * 8 біт = 560 біт, не рахуючи пробілів, точок, абзаців та інше.

Статистика

а8 б3 в5 г1 д3 е6 з1 и5 і4 ї2 к4 л3 м1 н2 о2 п2 р1 с3 т4 у4 ц2 ч1 ю1 я2

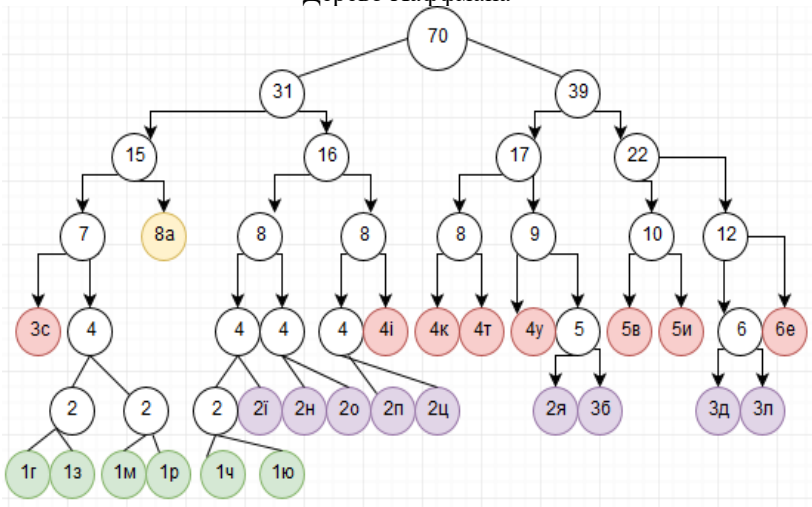
Впорядковуємо 1) за числом входжень та 2) за алфавітом.

г1 з1 м1 р1 ч1 ю1 ї2 н2 о2 п2 я2 ц2 б3 д3 л3 с3 і4 к4 т4 у4 в5 и5 е6 а8

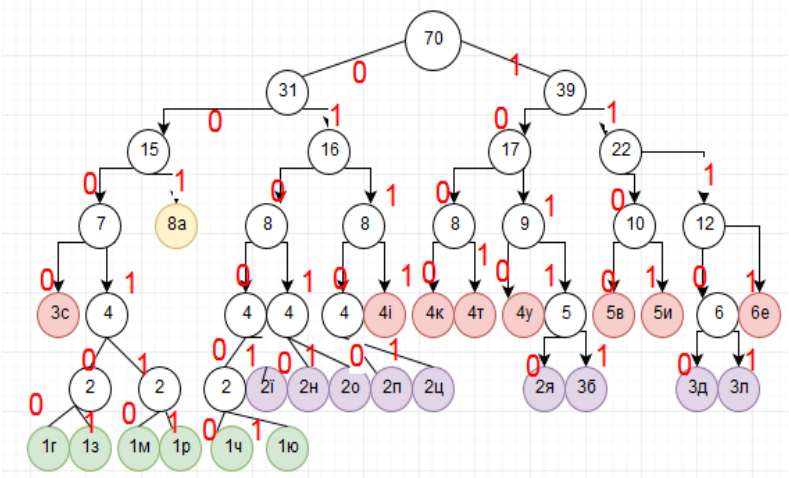
Етапи конкатенації у порядку зростання

{гз2,мр2,чю2,ї2,н2,о2,п2,ц2,я2,б3,д3,л3,с3,і4,к4,т4,у4,в5,и5,е6,а8}
{гзмр4,чюї4,но4,пц4,яб5,длб,с3,і4,к4,т4,у4,в5,и5,е6,а8}
↑ { с3,гзмр4,чюї4,но4,пц4, і4,к4,т4,у4,яб5, в5,и5, длб, е6,а8}
{ сгзмр7,чюїно8,пці8,кт8,уяб9, ви10, дле12,а8}
↑{ сгзмр7, а8, чюїно8,пці8,кт8,уяб9, ви10, дле12}
{ асгзмр15,чюїнопці16,ктуяб17, видле22}
{ асгзмрчюїнопці31,ктуябвидле39}
{ асгзмрчюїнопціктуябвидле70}

Дерево Хаффмана



Кодування: зліва 0, праворуч – 1.



		код	біт	вага
а	8	001	3	24
с	3	0000	4	12
і	4	0111	4	16
к	4	1000	4	16
т	4	1001	4	16
у	4	1010	4	16
в	5	1100	4	20
и	5	1101	4	20
е	6	1111	4	24
ї	2	01001	5	10
н	2	01010	5	10
о	2	01011	5	10

		код	біт	вага
п	2	01100	5	10
ц	2	01101	5	10
б	3	10111	5	15
д	3	11100	5	15
л	3	11101	5	15
я	2	10110	5	10
г	1	000100	6	6
з	1	000101	6	6
м	1	000110	6	6
р	1	000111	6	6
ч	1	010000	6	6
ю	1	010001	6	6

560==100%
305==x%
Висновок. В сумі маємо 305 біт, стиснення 54,5% від початкового

Лабораторна робота №9
Тема: Задача комівояжера (Traveling Salesman Problem (TSP))

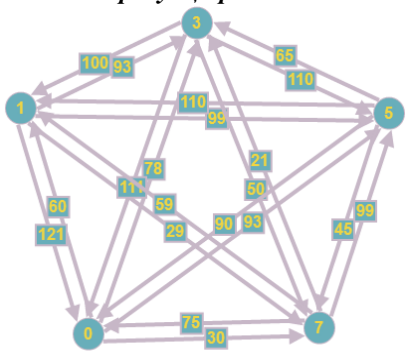
Теоретичні відомості

Незрозуміло, коли вперше було досліджено питання комівояжера. Однак, у відомій книзі, опублікованій у 1832 році під назвою «Комівояжер – як він повинен поводитися і що він повинен робити, щоб доставляти товари й бути успішним у бізнесі, – поради старого посланця» (нім. *DerHandlungsreisende*), – проблему було описано, але математичні інструменти для її вирішення не використовувалися. Натомість надає приклади маршрутів у певних районах Німеччини та Швейцарії Гамільтон Вільям Роуен.

Першим варіантом задачі може розглядатись англ. Icosian Game Вільяма Гамільтона 19 століття, яка полягала в тому, щоб знайти маршрути на графі з 20 вузлами. Першим, хто згадав оптимізацію як математичну проблему, був Карл Менгер (нім. *Karl Menger*), який сформулював її так на математичному семінарі в 1930 році: «Ми називаємо це проблемою планування (оскільки кожен листоноша стикається з цією проблемою, зокрема її вирішують багато мандрівників) (завдання пошуку найкоротшого шляху між кінцевою множиною місць, відстані до яких відомі). Невдовзі з'явилася назва того, що тепер відомо як проблема комівояжера, запропонована Хасслером Вітні з Прінстонського університету – Traveling Salesman Problem (TSP). Насправді, оптимальний шлях є досить складним завданням. З огляду на ці властивості, починаючи з другої половини 20 століття дослідження проблеми комівояжера має менше практичне, ніж теоретичне значення як моделі для розробки нових алгоритмів оптимізації.

На прикладі задачі комівояжера було розроблено багато сучасних поширених дискретних методів оптимізації, таких, як методи поділу площин, розгалужень і границь, а також різні варіації евристичних алгоритмів.

Метод редукції рядків і колонок



У задачі комівояжера процес пошуку оптимального маршруту за допомогою скорочення колони розбивається на (n-2) етапи. Алгоритм розрахунку на кожному етапі однаковий.

Для наочності розглянемо конкретний випадок задачі комівояжера: комерсант має об'їхати 5 міст, побувавши в кожному по одному разу і повернутися у початкове. При цьому витрати на час або пальне повинні бути мінімальними.

Вартість на переміщення між містами знаходяться в наступній таблиці:

	1	2	3	4	5	
1	-	60	78	93	30	
2	121	-	93	99	29	
3	111	100	-	110	50	
4	90	110	65	-	45	
5	75	59	21	99	-	

Перш ніж приступити до 1 етапу, знайдемо потенційну верхню межу функції вартості. Для цього вибираємо довільний маршрут (1.2)>(2.3)>(3.4)>(4.5)>(5.1), для якого обчислюємо значення функції F_b=60+93+110+45+75=383. Значення функції мети F_0 повинно бути меншим за F_b.

Крок 1. Проведемо редукцію рядків, для чого в кожному рядку знаходимо мінімальний елемент і віднімаємо його від всіх елементів даного рядка. В результаті отримуємо таблицю з хоча б одним нулем у рядку:

	1	2	3	4	5	A_i
1	-	30	48	63	0	30
2	92	-	64	70	0	29
3	61	50	-	60	0	50
4	45	65	20	-	0	45
5	54	38	0	78	-	21
					α=∑=	175

де стовпчик A_i містить значення мінімальних елементів кожного рядка, сумуємо його значення це α=175.

Виконуємо редукцію стовпчиків – в кожній колонці знаходимо мінімальний елемент і віднімаємо його від елементів даної колонки (рядок B_j містить мінімальні елементи кожної колонки – суму елементів позначаємо β=135).

	1	2	3	4	5	A_i
1	-	0	48	3	0	30
2	47	-	64	10	0	29
3	16	20	-	0	0	50
4	0	35	20	-	0	45
5	9	8	0	18	-	21
B_j	45	30	0	60	0	α=175 β=135

Після цього, обчислюємо найнижчу можливу межу функції мети.

$$F_{min} = \sum_{i=1}^{n} A_i + \sum_{i=1}^{n} B_j = \alpha + \beta = 175 + 135 = 310;$$

Тобто, функція мети F_0 повинна знаходитись в межах $310 \leq F_0 \leq 383$ $(F_{min} \leq F_0 \leq F_b)$.

Далі перевіряємо, щоб у кожному рядку і стовпчику містилося хоча б по одному нульовому елементу. Якщо умова виконується, тоді розв'язок зупиняється і нульові комірки позначають оптимальний шлях комівояжера з оптимальною функцією мети F_0 . Якщо ж умова не виконується, тоді рішення продовжується – переходимо до визначення наступного з етапів знаходження оптимального маршруту комівояжера. Для цього визначаємо величини a_i та b_j , які вважають «штрафами» рядків і стовпчиків і фіксуємо їх у відповідну колонку і рядок таблиці.

Штрафом i-го рядка є найменше значення даного рядка, крім першого нуля. Якщо *i*-й рядок містить дві чи більше нульові комірки, то $a_i = 0$.

Штрафом j-ї колонки є найменше значення даного стовпчика, крім першого нуля. Якщо *j*-та колонка містить дві чи більше нульові комірки, то $b_j = 0$.

	1	2	3	4	5	A_i
1	-	**0**	48	3	**0**	30
2	47	-	64	10	**0**	29
3	16	20	-	**0**	**0**	50
4	**0**	35	20	-	**0**	45
5	9	8	**0**	18	-	21
B_j	45	30	0	60	0	α=175 β=135

Для кожної нульової комірки визначаємо вторинні штрафи за наступною формулою: $\sigma_{ij} = a_i + b_j$. Після чого заносимо їх в наступну таблицю:

(i,j)	(1.2)	(1.5)	(2.5)	(3.4)	(3.5)	(4.1)	(4.5)	(5.3)
σ_{ij}	8	0	10	3	0	9	0	**28**

Далі знаходимо **максимальне** значення серед знайдених штрафів. В нашому випадку найбільшим є$\sigma_{(5.3)} = 28$. Тобто, у маршрут потрібно ввести ребро (5,3). В результаті отримуємо нову таблицю, в якій викреслюємо з розгляду п'ятий рядок і третю колонку.

	1	2	4	5	A_i
1	-	0	3	0	
2	47	-	10	0	
3	16	20	0	∞	
4	0	35	-	0	
B_j					α= , β=

У будь-якому рядку і у будь-якій колонці таблиці комівояжера повинна існувати одна заборонена комірка. В нашому випадку, така комірка симетрична вилученій (3,5), значення в якій замінюємо на ∞. В результаті отримуємо таблицю, до якої застосовуємо розрахунки на другому етапі.

Крок 2. Аналогічно першому етапу виконуємо редукцію рядків і стовпчиків. Отримуємо:

	1	2	4	5	A_i
1	-	0	3	0	0
2	47	-	10	0	0
3	16	20	0	∞	0
4	0	35	-	0	0
B_j	0	0	0	0	α=0 , β=0

Якщо б значення A_i, B_j відрізнялися від нуля, то повинні були б перевизначити функцію мети:

$$F_{min} = F_{min} + \sum_{i=1}^{n} A_i + \sum_{i=1}^{n} B_j,$$

і врахувати, що значення функції мети повинно знаходитись в межах$F_{min} \leq F_0 \leq F_b$. У нашому випадку нові A_i, B_j дорівнюють нулю, ми цей крок пропускаємо.

Знаходимо штрафи рядків і колонок.

	1	2	4	5	A_i
1	-	**0**	3	**0**	0
2	47	-	10	**0**	0
3	16	20	**0**	∞	0
4	**0**	35	-	**0**	0
B_j	0	0	0	0	α=0 , β=0

Для кожної нульової комірки визначаємо вторинні штрафи.

(i,j)	(1.2)	(1.5)	(2.5)	(3.4)	(4.1)	(4.5)
σ_{ij}	**20**	0	10	19	16	0

Серед знайдених значень вторинних штрафів знаходимо максимальне. Це означає, що комівояжер повинен використати у маршруті ребро (1,2). Вилучаємо рядок і стовпчик, у результаті для третього етапу отримуємо нову таблицю, в якій відсутні рядок i=1 та колонка j=2, а комірка (2, 1) є забороненою.

	1	4	5	A_i
2	∞	10	0	
3	16	0	∞	
4	0	-	0	
B_j				α= , β=

Крок 3. Виконуємо редукцію рядків і колонок, а також знаходимо штрафи рядків і колонок.

	1	4	5	A_i
2	∞	10	**0**	0
3	16	**0**	∞	0
4	**0**	-	**0**	0
B_j	0	0	0	α=0, β=0

Для кожної нульової комірки визначаємо вторинні штрафи.

(i,j)	(2.5)	(3.4)	(4.1)	(4.5)
σ_{ij}	10	**26**	16	0

Серед значень вторинних штрафів вибираємо максимальне і додаємо до оптимального шляху комівояжера ребро (3,4).

Крок 4. У результаті отримали таблицю 2*2, на цьому алгоритм зупиняється, таблиця вказує завершальний маршрут комівояжера (2,5) і (4,1).

	1	5
2	∞	0
4	0	0

Таким чином, ми отримали набір проміжків: (5.3)>(1.2)>(3.4)>(2.5)>(4.1), який упорядковуємо так, щоб кінцева вершина попереднього починала наступний проміжок: (1.2)>(2.5)>(5.3)>(3.4)>(4.1).

Обчислюємо функцію мети: $F_0=60+29+21+110+90=310$.

При цьому виконується умова $310 \leq 310 \leq 383$.

Приклад завдання

1. Знаходимо мінімальні значення в кожному рядку та віднімаємо це значення від усіх елементів рядка. Обчислюємо суму α=36.

i j	**1**	**2**	**3**	**4**	**5**	a
1	1000000	35	45	20	11	11
2	9	1000000	17	6	8	6
3	21	31	1000000	2	11	2
4	30	15	40	1000000	10	10
5	10	9	8	7	1000000	7

2. Знаходимо мінімальні значення в кожному стовпчику та віднімаємо це значення від усіх елементів стовпчика. Обчислюємо суму β=6.

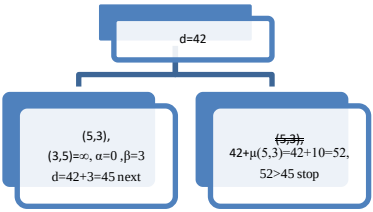
i j	1	2	3	4	5	a
1	999989	24	34	9	0	0
2	3	999994	11	0	2	0
3	19	29	999998	0	9	0
4	20	5	30	999990	0	0
5	3	2	1	0	999993	0
b	3	2	1	0	0	

Знайти суму α+β=36+6=42 – це нижня границя.

3. Отримали зведену матрицю редукції. Для кожного нуля (i,j) обчислюємо штрафну функцію = суму мін в і-рядку та в j-стовпчику, за виключенням самого елементу (i,j).

i j	1	2	3	4	5	a
1	999986	22	33	9	0(9+0)	0
2	0(0+0)	999992	10	0(0+0)	2	0
3	16	27	999997	0(9+0)	9	0
4	17	3	29	999990	0(3+0)	0
5	0(0+0)	0(0+3)	0(0+10)	0(0+0)	999993	0
b	0	0	0	0	0	

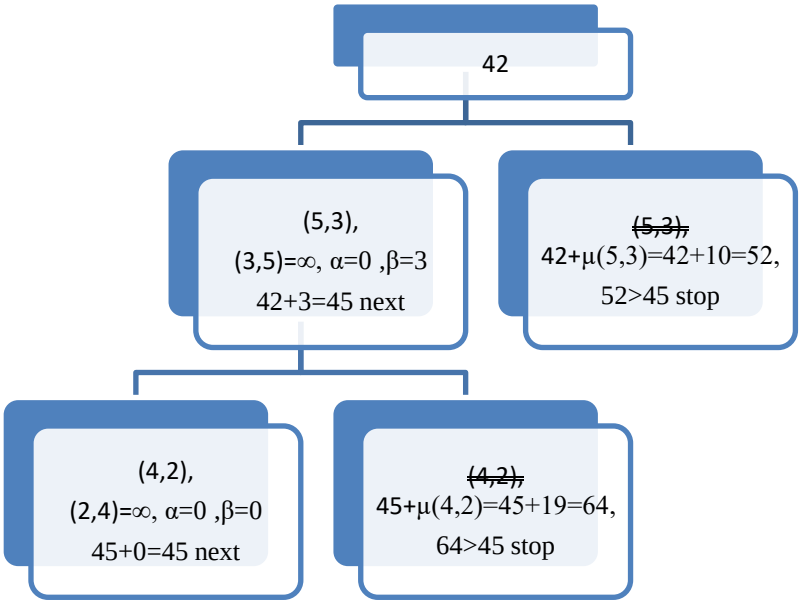
4. Обираємо найбільший штраф. Це точка (5,3). Розгалуження.



i j	1	2	4	5	a
1	999	22	9	0	0
2	0	999	0	2	0
3	16	27	0	999	0
4	17	3	999	0	0
b	0	3	0	0	

Повторюємо етапи 1-4.

i j	1	2	4	5	0
1	999986	19	9	0(9+0)	0
2	0(0+16)	999992	0(0+0)	2	0
3	16	24	0(16+0)	99999	0
4	17	0(0+19)	999990	0(0+0)	0
b	0	0	0	0	



α+β

i j	1	4	5	0
1	999	9	0	0
2	0	999	2	0
3	16	0	999	0
b	0	0	0	

штрафи

i j	1	4	5	0
1	999	9	0(9+2)	0
2	0(2+16)	999	2	0
3	16	0(16+9)	999	0
b	0	0	0	

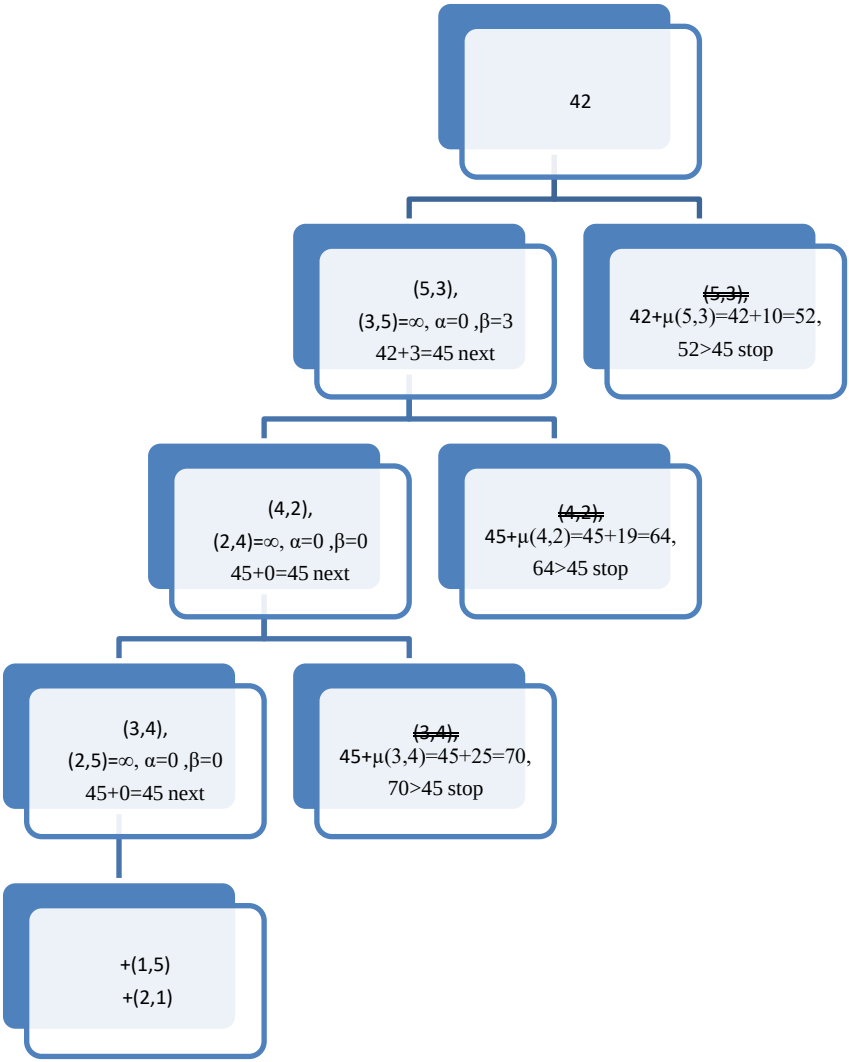
Вибір ребра.
З отриманої матриці 2*2 єдиний набір ребер (1,5), (2,1).

i j	1	5	0
1	999	0	0
2	0	999	0
b	0	0	

Впорядковуємо отриманий список ребер:
(5.3).(4.2).(3.4).(1.5).(2.1)
→(1.5).(5.3).(3.4).(4.2).(2.1)

Обчислюємо відстань по початковій матриці
D=11+8+2+15+9=45 (має збігтися із значенням дерева)

i j	**1**	**2**	**3**	**4**	**5**
1	1000000	35	45	20	11
2	9	1000000	17	6	8
3	21	31	1000000	2	11
4	30	15	40	1000000	10
5	10	9	8	7	1000000


42
(5,3),
(3,5)=∞, α=0 ,β=3
42+3=45 next
~~(5,3),~~
42+μ(5,3)=42+10=52,
52>45 stop
(4,2),
(2,4)=∞, α=0 ,β=0
45+0=45 next
~~(4,2),~~
45+μ(4,2)=45+19=64,
64>45 stop
(3,4),
(2,5)=∞, α=0 ,β=0
45+0=45 next
~~(3,4),~~
45+μ(3,4)=45+25=70,
70>45 stop
+(1,5)
+(2,1)

Лабораторна робота №10
Тема: Пакет Solver Microsoft Excel для задачі комівояжера

Теоретичні відомості.

Задача комівояжера є однією з найвідоміших задач комбінаторної оптимізації і полягає в пошуку найвигіднішого маршруту, який хоча б один раз проходить через задане місто та повертається до міста відправлення.

Сьогодні необхідно вирішувати цю проблему в багатьох сферах, пов'язаних із замкнутими і суворо залежними від часу системами, такими як: конвеєрне виробництво, багатоопераційні технологічні комплекси, системи навантаження суден і залізниць, замкнуті маршрути для вантажних перевезень по маршрутах, калькуляція авіаційних витрат. Тому завдання комівояжера є дуже актуальним у сучасну епоху, адже час, ресурси та фінансові ресурси не безмежні, а розумне та правильне їх використання є ключем до успіху.

Прості методи вирішення проблеми комівояжера: пошук за повним словом, «жадібний» алгоритм (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального скелета дерева. На практиці застосовують різні модифікації найефективніших методів: метод гілок і меж та метод генетичних алгоритмів, та так само алгоритм мурашиної колонії.

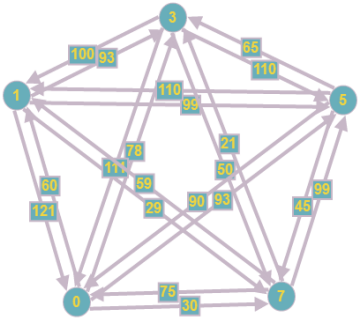
Усі ефективні (такі, що скорочують час на повний перебір) методи розв'язання задачі TSP (комівояжера) – евристичні. У більшості евристичних методів знаходиться не найоптимальніший маршрут, а *наближений розв'язок*. Користуються популярністю так звані ***any-time*** алгоритми, тобто алгоритми, що поступово покращують деякий поточний наближений розв'язок.

Задача TSP (комівояжера) – *NP*-повна. Часто на ній проводять випробування нових підходів до евристичного скорочення повного перебору.

Пошук рішення – це надбудова до Microsoft Excel, яку можна використовувати для аналізу «що, якщо». З його допомогою можна знайти оптимальне значення (максимальне або мінімальне) формули, що міститься в одній комірці (називається **цільовою**), враховуючи обмеження значень в інших комірках аркуша, які містять формули. *SolutionFinderm* використовує набір комірок, які називаються комірками змінних рішень

або просто комірками змінних, для оцінки формул у цільових комірках і комірках обмежень. Надбудова «Пошук рішення» змінює значення в клітинці змінної рішення на основі меж клітинки обмежень і виводить бажаний результат у цільову клітинку.

Розв'язання задачі комівояжера з використанням пакету Solver Microsoft Excel.



Надбудова «Знайти рішення» дозволяє визначати максимальне або мінімальне значення однієї комірки, змінюючи інші комірки.

Для наочності розглянемо конкретний випадок задачі комівояжера, а саме: комівояжер повинен об'їхати 5 міст, побувавши в кожному по одному разу і повернутися у початкове. При цьому витрати повинні бути мінімальними.

Відстані для переміщення між містами знаходяться в таблиці 1.

Таблиця 1.

Відстані між містами

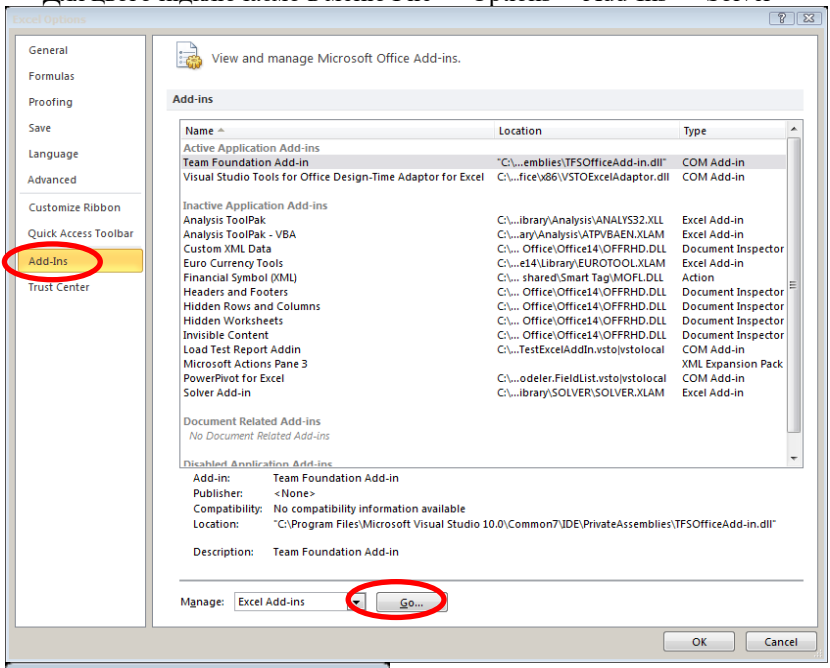
	1	2	3	4	5
1	-	60	78	93	30
2	121	-	93	99	29
3	111	100	-	110	50
4	90	110	65	-	45
5	75	59	21	99	-

Перенесемо таблицю в Microsoft Excel.

	A	B	C	D	E	F
1		1	2	3	4	5
2	1	1000000	60	78	93	30
3	2	121	1000000	93	99	29
4	3	111	100	1000000	110	50
5	4	90	110	65	1000000	45
6	5	75	59	21	99	1000000
7						

Рисунок 1 – Відстані для переміщення між містами

Використаємо надбудови Excel.
Для цього підключаємо в меню File → Options →Add-Ins → Solver



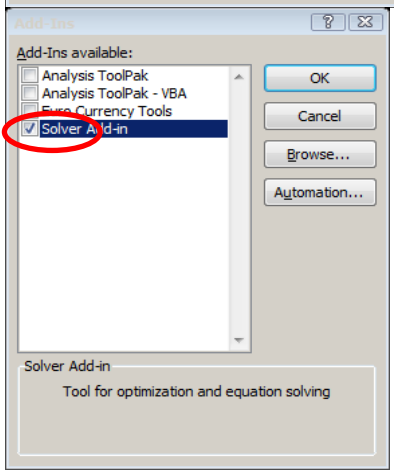


Рисунок 2 – Підключення надбудови пошуку розв'язків

На вкладці Data з'явиться нова кнопка Solver (Рис. 3)

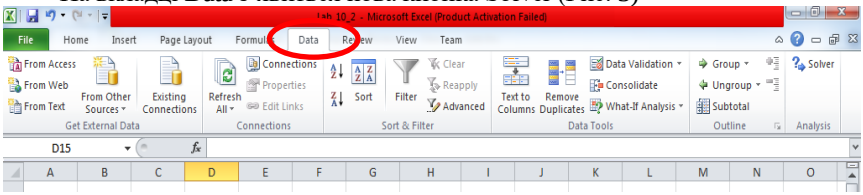


Рисунок 3 – Вкладка Data після підключення Solver

На робочому листі сформуємо дві таблиці, одну для результатів, другу з умовами.

	A	B	C	D	E	F	G
1	Умови	1	2	3	4	5	
2	1	1000000	60	78	93	30	
3	2	121	1000000	93	99	29	
4	3	111	100	1000000	110	50	
5	4	90	110	65	1000000	45	
6	5	75	59	21	99	1000000	
7							
8	Цільова функція (мин)			=СУММПРОИЗВ(B2:F6;B11:F15)			
9							
10	Відповідь	1	2	3	4	5	
11	1						=СУММ(B11:F11)
12	2						=СУММ(B12:F12)
13	3						=СУММ(B13:F13)
14	4						=СУММ(B14:F14)
15	5						=СУММ(B15:F15)
16		=СУММ(B11:B15)	=СУММ(C11:C15)	=СУММ(D11:D15)	=СУММ(E11:E15)	=СУММ(F11:F15)	

Рисунок 4 – Підготовчий етап

В комірки G11:G15, B16:F16 вводимо значення сум для відповідних рядків (стовпчиків).

Комірку D8 використаємо як цільову функцію, введемо формулу:

$$\min L = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} \cdot x_{ij} \ ,$$ D8=SUMPRODUCT(B2:F6;B11:F15).

На вкладці Data викликаємо «Пошук рішення» = Solve. У відповідних клітинках (малюнок 10.6) заповнюємо:

Set objective (max, min, value of) → комірка цільової функції: D8;

By Changing variable cells: комірки, які змінюються: B11:F15;

Subject to the Constraints: порівнюються умови обмежень, знаки з умови системи (в полі обмежень умови додаються за допомогою кнопки Add);

Select a Solving Method: нелінійний, симплекс метод, евристичний.

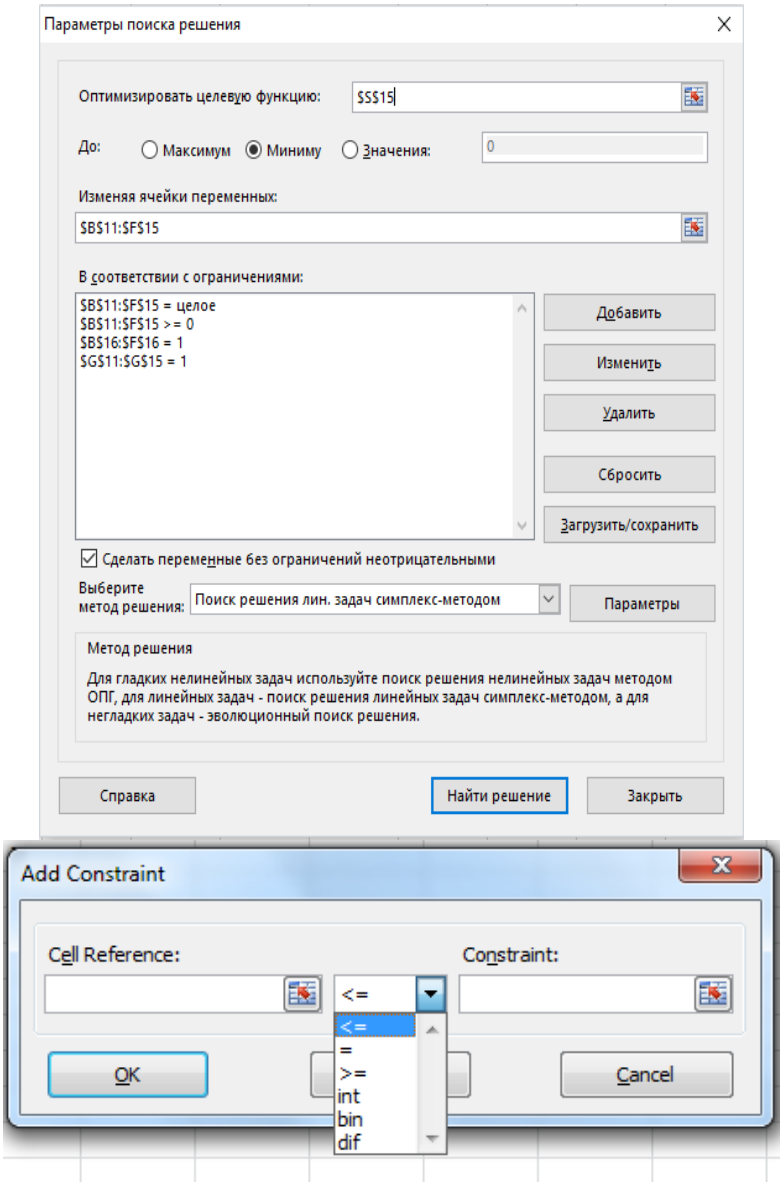


Рисунок 5 – Налагодження пакету Solver

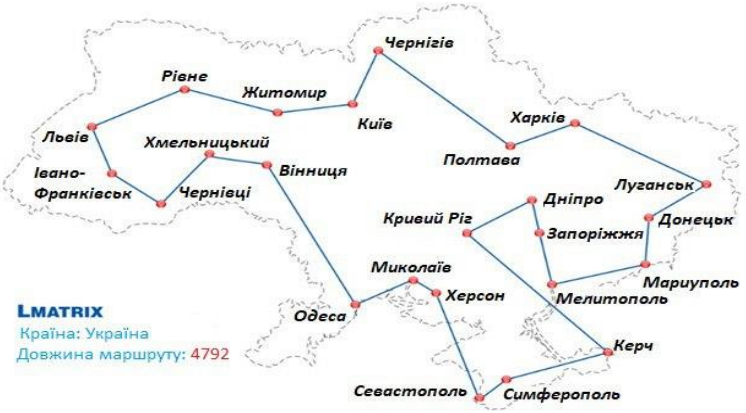
	A	B	C	D	E	F	G
1	Умови	1	2	3	4	5	
2	1	1000000	60	78	93	30	
3	2	121	1000000	93	99	29	
4	3	111	100	1000000	110	50	
5	4	90	110	65	1000000	45	
6	5	75	59	21	99	1000000	
7							
8	Цільова функція (мин)			310			
9							
10	Відповідь	1	2	3	4	5	
11	1	0	1	0	0	0	1
12	2	0	0	0	0	1	1
13	3	0	0	0	1	0	1
14	4	1	0	0	0	0	1
15	5	0	0	1	0	0	1
16		1	1	1	1	1	
17							

Рисунок 6 – Отриманий результат

З отриманого результату виписуємо маршрут 1→2 →5→3→4→1.

Відповідна відстань з початкової матриці d=60+29+21+110+90=310, що збігається зі значенням цільової функції.

Пошукове завдання



Розв’язати задачу комівояжера для міст України.

	1	2	3	4	5	6	7	8	9	10	11	12
1 Київ		476	585	541	740	478	475	340	328	567	324	142
2 Дніпро			940	930	312	219	454	193	802	330	689	581
3 Івано-Франківськ				133	1221	1062	793	923	276	884	253	727
4 Львів					1212	1018	786	880	212	877	237	685
5 Маріуполь						411	635	485	1082	421	972	850
6 Харків							673	144	809	548	801	587
7 Одеса								583	692	218	544	613
8 Полтава									667	474	663	448
9 Рівне										783	193	472
10 Херсон											636	704
11 Хмельницький												467
12 Чернігів												

Є варіант розв’язку з маршрутом довжиною 3858 км. Чи є кращій спосіб? Знайти маршрут.

Лабораторна робота №11
Тема: Орієнтовані графи. Алгоритм Флойда-Воршелла

Теоретичні відомості.

Орієнтованим графом або ***орграфом*** G називається пара множин (V, E), де $E \subseteq V \times V$. Елементи множини V називаються **вершинами** орграфа G, а елементи множини E – ***дугами*** орграфа $G = (V, E)$. Отже, дуга – це впорядкована пара вершин. Відповідно, V називається ***множиною вершин*** і E – ***множиною дуг*** орграфа G.

Якщо $e = (v, w)$ – дуга, то вершина v називається ***початком***, а вершина w – ***кінцем*** дуги e. Кажуть, що дуга e ***веде*** з вершини v у вершину w.

Як і звичайний граф, орграф $G = (V, E)$ може бути заданий шляхом переліку елементів скінченних множин V і E, діаграмою або за допомогою матриць.

Діаграма орграфа відрізняється від діаграми звичайного графа тим, що дуги орграфа зображуються стрілками (відрізками або кривими), що йдуть від початку до кінця дуги. Напрямок лінії позначається стрілкою.

Перенумеруємо вершини орграфа $G = (V, E)$ натуральними числами від 1 до n; дістанемо множину вершин V у вигляді $\{v_1, v_{2,}, \dots, v_n\}$.

Матрицею суміжності A орграфа G є квадратна матриця порядку n, в якій елемент i -го рядка і j -го стовпчика a_{ij} рівний 1, якщо $(v_i, v_j) \in E$; $a_{ij} = 0$.

Перенумеруємо всі вершини орграфа $G = (V, E)$ числами від 1 до n, а дуги – числами від 1 до m.

Матрицею інцидентності B орграфа G називається $n \times m$ – матриця, в якій елемент i -го рядка і j -го стовпчика b_{ij} рівний 1, якщо вершина v_i є початком дуги e_i; $b_{ij} = -1$, якщо вершина v_i є кінцем дуги e_i; $b_{ij} = 0$, якщо вершина v_i і дуга e_i неінцидентні.

Орграфи $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$ називаються ***ізоморфними***, якщо існує таке взаємно однозначне відображення φ множини V_1 на множину V_2, що дуга $(v, w) \in E_1$ тоді і тільки тоді, коли дуга $(\phi(v), \phi(w)) \in E_2$.

Півступенем виходу вершини v (позначається $\delta^+(v)$) орграфаG називається кількість дуг орграфаG, початком яких є вершина v.

Півступенем заходу вершини v (позначається $\delta^-(v)$) орграфаG називається кількість дуг орграфаG, кінцем яких є вершина v.

Маршрутом або ***шляхом*** в орграфі$G = (V, E)$ називається послідовність його вершин і дуг : $v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1}$, така, що $e_i = (v_i, v_{i+1})$, $i = 1,2,\ldots,k$. Кажуть, що цей маршрут ***веде*** з вершини v_1 у вершину v_{k+1}. Число k дуг у маршруті називається його ***довжиною***.

Маршрут, в якому всі дуги попарно різні, називається ***ланцюгом***. Маршрут, в якому всі вершини попарно різні, називається ***простим ланцюгом***. Маршрут називається ***замкненим*** (***циклічним***), якщо $v_1 = v_{k+1}$. Замкнений ланцюг називається ***циклом***, а замкнений простий ланцюг –***простим циклом***, або ***контуром***. Орграф називається ***ациклічним***, якщо він не має жодного циклу.

Якщо існує маршрут, який веде з вершини *v* у вершину *w*, то кажуть, що вершина *w* є ***досяжною*** з вершини *v*. У цьому випадку ***відстанню*** *d* (*v*,*w*) від вершини *v* до вершини *w* називається довжина найкоротшого маршруту, що веде з *v* у *w*. Відстань між вершиною *v* і вершиною *w*, яка є недосяжною з *v*, позначається символом ∞.

Вершина v орграфаG називається ***джерелом***, якщо з v є досяжною будь-яка інша вершина орграфаG. Вершина w називається ***стоком***, якщо вона є досяжною з будь-якої іншої вершини орграфаG.

Алгоритм Флойда-Воршелла

В інформатиці алгоритм Флойда-Воршелла використовується для вирішення проблем найкоротшого шляху у зважених графах з додатніми або від'ємними вагами ребер (але без від'ємних циклів). У звичайній реалізації алгоритм виводить довжину (загальну вагу) найкоротшого шляху між усіма парами вершин, хоча він не виводить інформації про самі шляхи. Різні версії цього алгоритму використовуються для пошуку транзитивних замикань у відношеннях або для пошуку найширшого шляху між усіма парами вершин у зваженому графі (розглянемо метод Шульца) (англійською: проблема найширшого шляху).

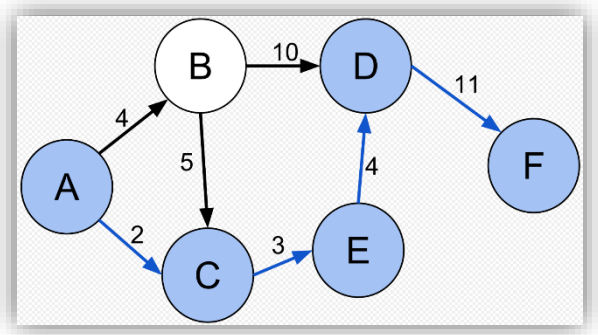
Якщо розвиток загального мистецтва програмування вимагає постійного винаходу та розробки парадигм, розвиток мистецтва окремого програміста вимагає, щоб він розширював свій набір парадигм.
(Роберт Флойд)

Практичне застосування

Алгоритм Флойда-Воршелла – це приклад динамічного програмування.

Книга з описом алгоритму була опублікована Робертом Флойдом у 1962 році у звичному сьогодні вигляді. Однак насправді це той самий алгоритм для знаходження транзитивних замикань у графах, опублікований Бернардом Роєм у 1959 році та Стівеном Воршеллом у 1962 році, і схожий на алгоритм Кліні[ен], опублікований у 1956 році, для перетворення детермінованих скінченних автоматів у регулярні автомати. Сучасне формулювання алгоритму як трьох вкладених циклів було вперше подано Пітером Інгерманом 1962 року.

Алгоритм також називають Алгоритм Флойда, Алгоритм Роя-Воршелла, Алгоритм Роя-Флойда, або Алгоритм WFI.



Алгоритм Воршелла обчислює всі можливі шляхи в графі між кожною парою вершин. Він виконується за $\Theta(|V|^3)$ порівнянь. Це доволі громіздко, враховуючи, що в графі може бути до $\Omega(|V|^2)$ ребер, і кожну комбінацію потрібно перевірити. Процес порівняння виконує це шляхом поступового покращення оцінки за найкоротшим шляхом між двома вершинами, поки оцінка не стає оптимальною.

Розгляньмо граф G з ребрами V, пронумерованими від 1 до N.

Крім того використовують функцію shortestPath (i, j, k), яка повертає найкоротший шлях від i до j, використовуючи вершини з множини {1,2,…,k} як внутрішні у маршруті. Тепер, маючи таку функцію, нам потрібно знайти найкоротший шлях від кожного i до кожного j, використовуючи тільки вершини від 1 до k + 1.

Для кожної з цих пар вершин, найкоротший шлях може бути або (1) – шлях, у якому є тільки вершини з множини {1, …, k}, або (2) – шлях, який проходить від i до k + 1 а потім від k + 1 до j. Найкоротший шлях

від i to j that only uses vertices 1 через k визначається функцією shortestPath(i, j, k), і якщо є коротший шлях від і до (k + 1 до j), то довжина цього шляху буде сумою (конкатенацією) найкоротшого шляху від і до k + 1 (використовуючи вершини {1, …, k}) і найкоротший шлях від k + 1 до j (також використовуючи вершини з {1,…,k}). – це вага ребра між і та j.

Можна визначити shortestPath(i, j, k + 1) наступною **рекурсивною формулою:**

$$\text{shortestPath}(i, j, 0) = w(i, j)$$

рекурсивна частина:

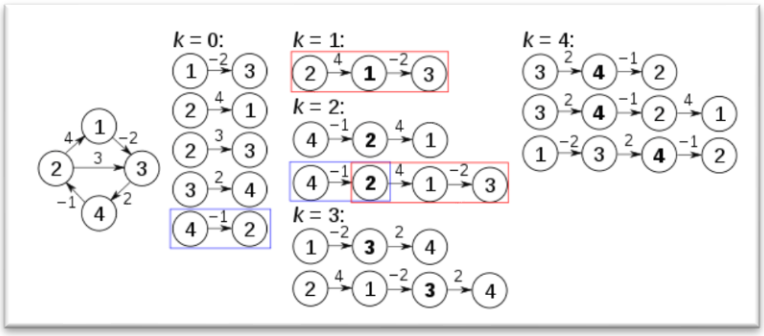
$$\text{shortestPath}(i, j, k+1) = \min(\text{shortestPath}(i, j, k), \text{shortestPath}(i, k+1, k) + \text{shortestPath}(k+1, j, k))$$

Ця формула є основною частиною алгоритму Флойда-Воршелла. Алгоритм спочатку обчислює shortestPath(i, j, k) для всіх пар(i, j) де k = 1, потім k = 2, і т. д. Цей процес продовжується, поки k = N, і поки не знайдено всі найкоротші шляхи для пар (i, j).

Псевдокод для цієї версії алгоритму:

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each vertex v
3    dist[v][v] ← 0
4 for each edge (u,v)
5    dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
6 for k from 1 to |V|
7    for i from 1 to |V|
8       for j from 1 to |V|
9          if dist[i][j] > dist[i][k] + dist[k][j]
10             dist[i][j] ← dist[i][k] + dist[k][j]
11         end if
```

Алгоритм виконується на малюнку нижче:



Перед першою ітерацією циклу k=0 і відомі шляхи відповідають одиночним ребрам у графі.

Коли k=1, знайдено шляхи, які проходять через вершину 1, зокрема: шлях 2→1→3, замінить шлях 2→3, що проходить через меншу кількість ребер, але є довшим.

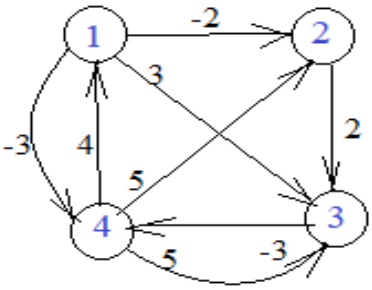
При k=2, знаходяться шляхи, що проходять через вершини {1,2} . Червоні та блакитні квадратики показують, як шлях 4→2→1→3 складається з 4→2 і 2→1→3, визначеними на попередніх ітераціях. Шлях 4→2→3 не розглядається, бо 2→1→3 поки що найкоротший шлях.

При k=3,знаходяться шляхи, що проходять через {1,2,3}.

Нарешті, при k=4, знайдено всі найкоротші шляхи.

Приклади завдань

Приклад 1.Знайти всі найкоротші шляхи для графа:



$$D^0 = \begin{pmatrix} - & -2 & 3 & -3 \\ \infty & - & 2 & \infty \\ \infty & \infty & - & -3 \\ 4 & 5 & 5 & - \end{pmatrix}; Q^0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & - & 2 & 2 \\ 3 & 3 & - & 3 \\ 4 & 4 & 4 & - \end{pmatrix}$$

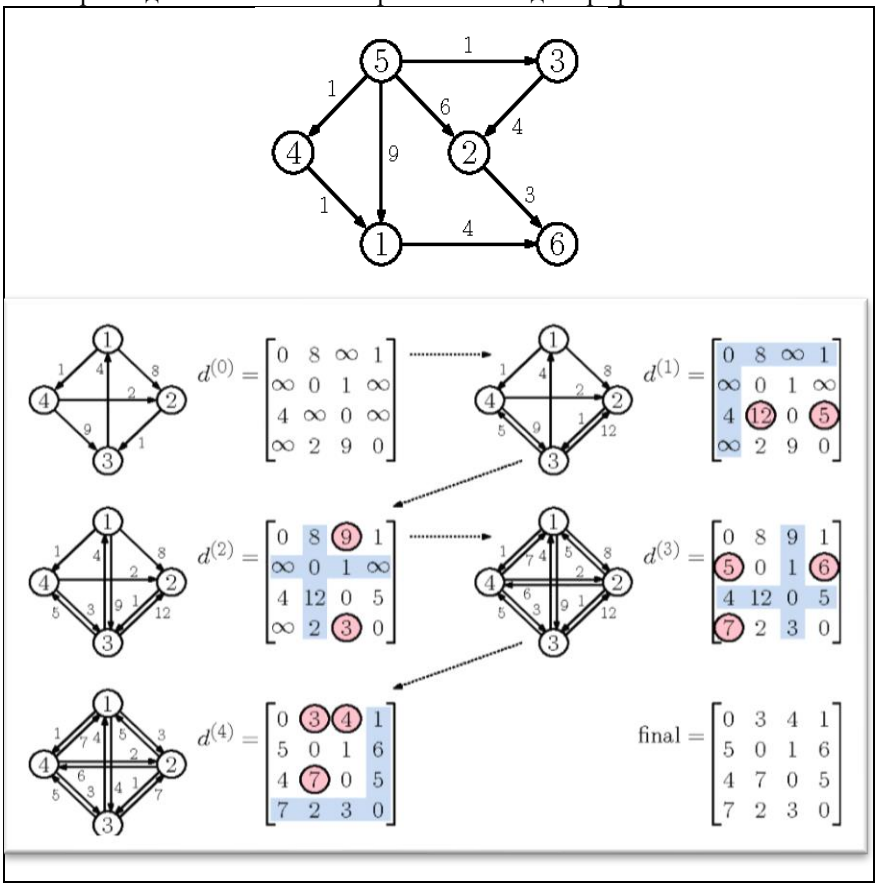
$$D^1 = \begin{pmatrix} 0 & -2 & 3 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 5 & 0 \end{pmatrix}; Q^1 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 4 & 0 \end{pmatrix}$$

$$D^2 = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; Q^2 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix}$$

$$D^3 = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & -1 \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; Q^3 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 3 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix}$$

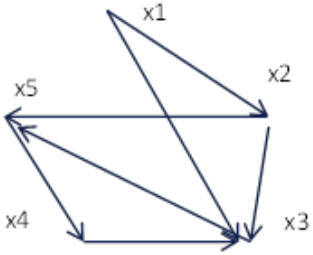
$$D^4 = \begin{pmatrix} 0 & -2 & 0 & -3 \\ 3 & 0 & 2 & -1 \\ 1 & -1 & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; Q^4 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 4 & 0 & 2 & 3 \\ 4 & 4 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix}$$

Приклад 2.Знайти всі найкоротші шляхи для графа:



Приклад 3. Орграф $G(X, U)$ задано списком:
$X=\{x_1, x_2, x_3, x_4, x_5\}$
$U=\{(x_1,x_2),(x_1,x_3),(x_4,x_3),(x_5,x_4),(x_2,x_5),(x_2,x_3),(x_3,x_5)\}$

Задати цей граф у геометричний та матричний способи.

Геометричний (діаграма)	Матричний (суміжність)
	$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 \\ -1 & -1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & -1 & -1 & 1 & 0 \end{pmatrix}$

Лабораторна робота №12
Тема: Мережі. Максимальний потік в мережі. Алгоритм Форда-Фалкерсона

Теоретичні відомості

Мережею називається зв'язний орграф без петель, тобто зважений орграф з двома виділеними вершинами: витоком і стоком. Потоком в мережі називається деяка функція, яка ставить у відповідність дузі деяке число – вагу дуги. Вага дуги означає її пропускну здатність. Потік дуги не перевищує її пропускної здатності та може змінюватися. Потік виходить із витоку й без втрат, в тому ж обсязі заходить до стоку. Якщо потік дорівнює вазі дуги, то ця дуга є насиченою, тобто, через неї можна пройти при розгляді ланцюгів в графі.

Потік називається насиченим, якщо будь-який ланцюг із витоку в стік містить насичену дугу. Умова рівноваги (за обсягом входу і виходу) виконується для кожної вершини мережі.

Завдання про найбільший потік в мережі – не єдина, але, ймовірно, основна задача для потоків в мережі. Очевидна можливість практичного застосування цього завдання для вирішення транспортних проблем (затори на дорогах можна умовно пов'язувати з насиченням мережі або окремої її дуги), проблем транспортування.

Алгоритм Форда-Фалкерсона пошуку максимального потоку в мережі складається з двох частин – насичення потоку і його перерозподілу.

1. Насичення потоку. Розшукуються ланцюги з витоку в стік й всі дуги ланцюга насичуються однаковим найбільшим потоком, що визначаються пропускною здатністю найбільш «тонкої» дуги або найменшою різницею між пропускною спроможністю й потоком в дузі. Різні ланцюги можуть мати спільні дуги. Отриманий потік узгоджений з умовою збереження в вузлах (вершинах). Потік, що входить у вершину, дорівнює потоку, що виходить з неї.

Потік в мережі проходить по ланцюгах із витоку в стік, тобто неприпустимий багаторазовий прохід по окремій дузі. Насичені дуги викреслюються. Перша частина завдання вважається розв'язаною, якщо немає ненасичених ланцюгів з витоку в стік. Перша частина задачі не має однозначної відповіді.

2. Перерозподіл потоку. Перерозподіл потоку виконується, виходячи з умови досягнення загального за мережею максимуму потоку. Для

цього в графі, в якому знята орієнтація дуг, розшукуються маршрути з витоку в стік, що складаються з ребер, які відповідають ненасиченим дугам, спрямованим вперед, і непорожнім дугам, спрямованим назад. Потоки в дугах прямого напрямку збільшуються на величину, на яку зменшуються потоки в зворотних дугах обраного маршруту. При цьому, очевидно, не можна перевищувати пропускну здатність дуг, спрямованих вперед, і допускати негативних потоків в зворотних дугах. У деяких випадках при вдалому виборі ланцюгів в першій частині алгоритму перерозподілу потоку буде не потрібно.

Максимальний потік

Приклад 1. Задана пропускна здатність дуг транспортної мережі (рис. 1) з початком у вершині 1 й кінцем у вершині 8. Використовуючи алгоритм Форда-Фалкерсона, знайти максимальний потік в мережі

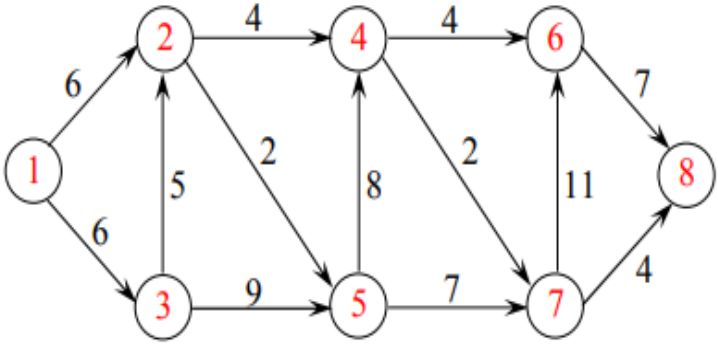


Рисунок 1 – Транспортна мережа

Розв’язання.

Етап 1. Насичення потоку. Розглянемо шлях 1–2–4–6–8. Припустимо через цей шлях потік, рівний 4 (найменша вага). При цьому дуги [2, 4] і [4, 6] будуть насиченими. Аналогічно, шлях 1–3–5–7–8 наситимо потоком 4. Розподіл потоку відзначимо на графі (рис. 2). У чисельнику ставимо пропускну здатність, в знаменнику – потік. Чисельник завжди більше знаменника, а знаменник може бути і нулем. Насичені дуги викреслюємо.

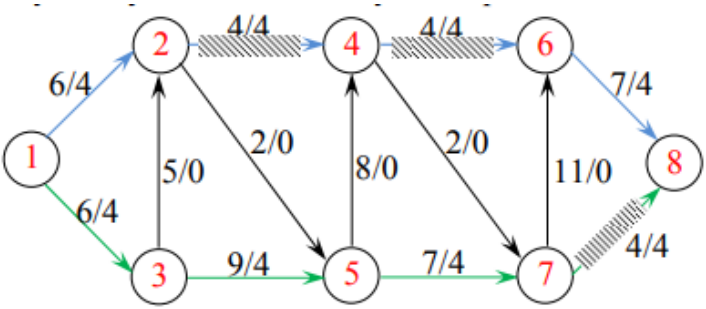


Рисунок 2 – Модифікація транспортної мережі

Зауважимо, що з 1 в 8 є ще ненасичений шлях, 1–3–2–5–4–7–6–8, потік в якому можна збільшити на 2. При цьому наситяться дуги [1, 3], [2, 5] і [4, 7], їх викреслюємо (рис.).

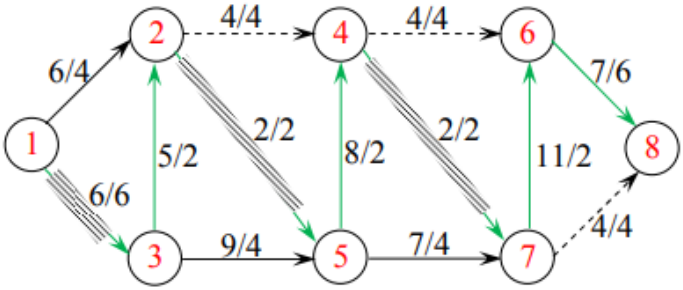


Рисунок 3 – Транспортна мережа

З 1 в 8 більше немає ненасичених шляхів. По дузі [1, 3] рухатися неможна (вона вже насичена), а рух по дузі [1, 2] закінчується в вершині 2, так як обидві дуги, які виходять з неї, насичені.

Етап 2. Перерозподіл потоку. Знайдемо послідовність вершин від 1 до 8, таку,що дуги, які з'єднують сусідні вершини, спрямовані з 1 в 8, не насичені, а дуги,спрямовані в зворотному напрямку, не порожні. Маємо єдину послідовність: 1–2–3–5–7–6–8. Перерозподіляємо потік. Потік в дугах прямого напрямку збільшуємо на 1 (бо найменша різниця 1=7–6), а потік в дугах зворотного напрямку зменшуємо на 1. Процес продовжуємо до тих пір, поки одна з прямих дуг не буде насичена або якась зворотна дуга не буде порожня.

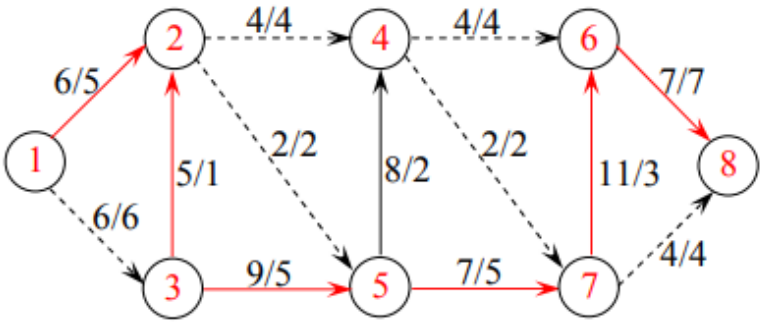


Рисунок 4 – Перерозподіл потоку

Потік в насиченій мережі можна порахувати по потоку, що виходить з витоку 1 або входить в стік 8. Очевидно, ці числа повинні бути рівні. Крім того, для перевірки розв'язку слід перевірити умову збереження потоку по вузлах.

Для кожного вузла сумарний вхідний потік має дорівнювати вихідному потоку. У розглянутому прикладі потік дорівнює 11. Розподіл потоку по дугах при одному і тому ж сумарному мінімальному потоці в мережі не єдиний.

Топологічне сортування мережі.

Нехай у мережі в загальному випадку є кілька витоків і стоків. Стоки і витоки займають в мережі крайні положення. Усі інші вершини можуть бути далі або ближче до них. Розташування вершини в мережі визначається її рівнем. Визначимо це поняття. Вершини рівня 0 – це витоки; вони утворюють множину N_0. Якщо N_i – множина вершин рівня i $\leq k$,то N_{k+1} – множина вершин рівня $k+1$ складається з тих і тільки тих вершин, попередники яких належать до будь-якої з множин N_0, ..., N_k, причому серед цих множин немає порожніх.

Порядковою функцією мережі називають відображення, яке зіставляє кожній вершині мережі її рівень.

Приклад 2. Відсортувати топологічно мережу на рис. 5.

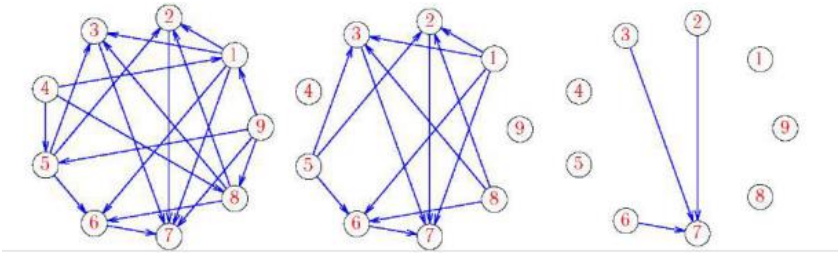


Рисунок 5 – Мережа для топологічного сортування

Розв’язання.

Знаходимо вершини (4 і 9), півступені входу яких дорівнює 0. Видаляємо ці вершини і дуги, що виходять з них. Видалені вершини утворюють перший рівень впорядкованої мережі. Отримуємо нову мережу (рис. 5б). У новій мережі у вершини 1, 5 і 8 не входять дуги. Видаляємо ці вершини (вони утворюють другий рівень) і отримуємо мережу, зображену на рис. 5в. Очевидно, третій рівень складається з вершин 2, 3 і 6, а останній (стік мережі) – з єдиної вершини 7. Зображуємо ту ж мережу за рівнями (рис. 32).

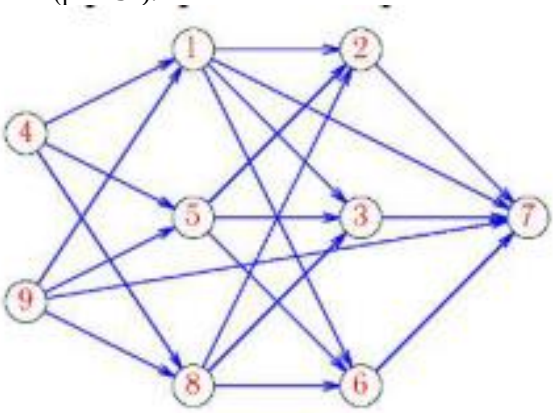
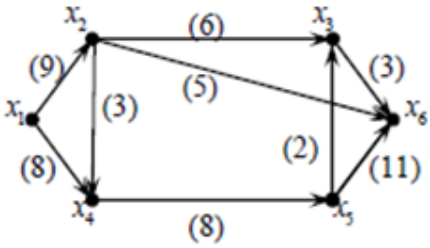


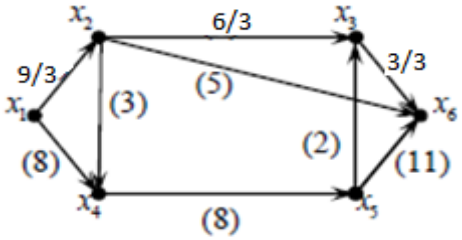
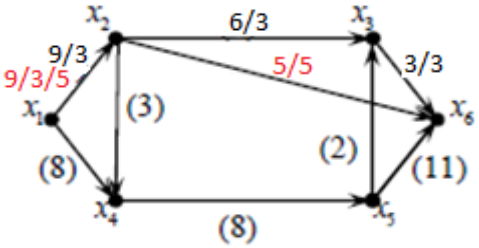
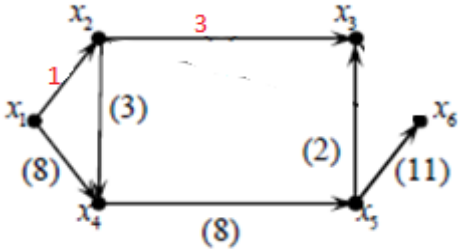
Рисунок 6 – Мережа відсортована за рівнями

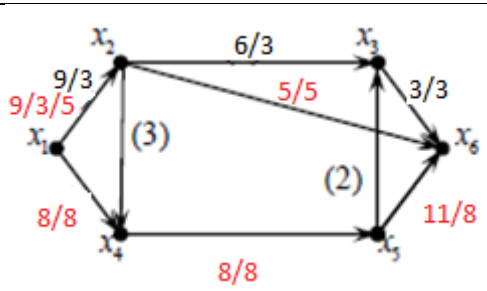
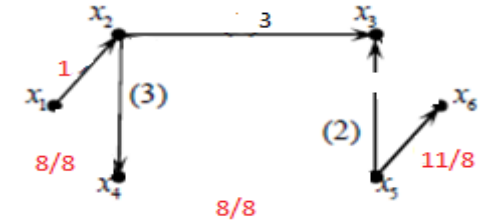
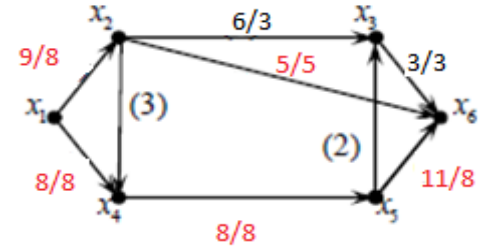
Приклад виконання 1 завдання

Задача 1. Обчислити повний потік в транспортній мережі G (в дужках зазначено припустимі пропускні здатності дуг у вказаних напрямах).



Розв'язок.

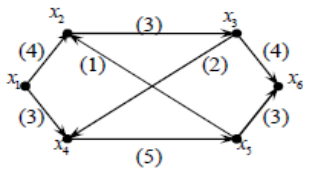
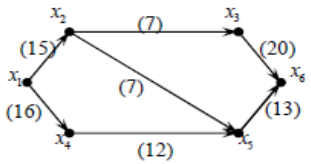
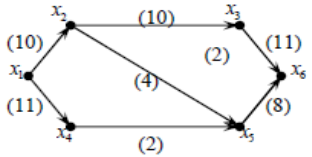
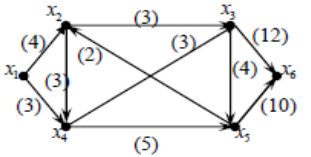
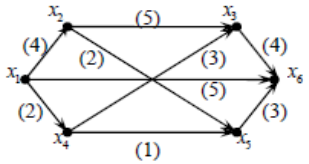
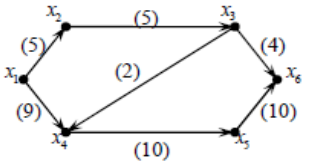
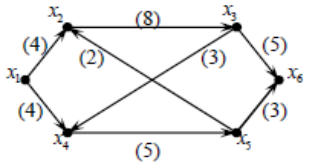
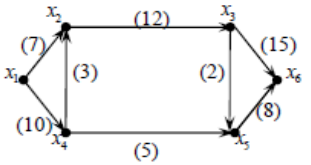
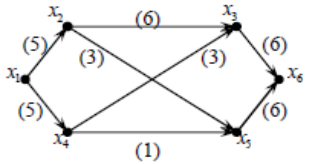
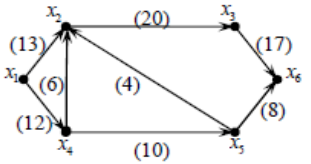
Маршрути та потік	Розрахунки
X1-x2-x3-x6 Min(9-6-3)=3	
X1-x2-x6 Min(6,5)=5	
X1-x2-x4-x5-x3-x6 Min(1,3,8,2,0)=0	

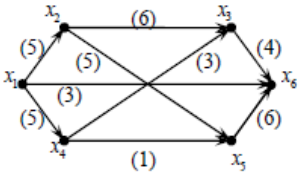
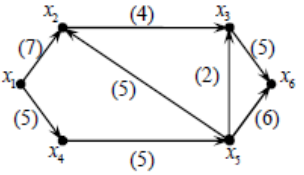
x1-x4-x5-x6 min(8,8,11)=8	
x1-x4-x5-x6	
Джерело W(x1)=8+8=16 Стік W(x6)=3+5+8=16	

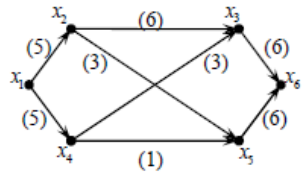
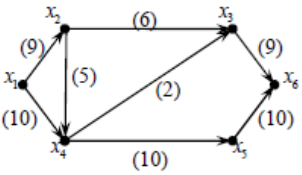
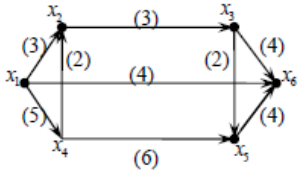
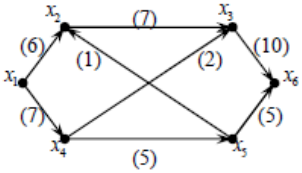
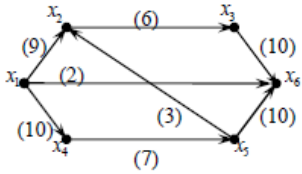
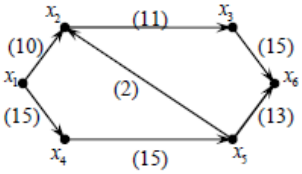
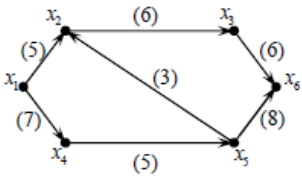
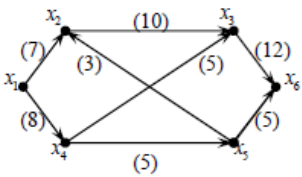
Завдання по варіантам

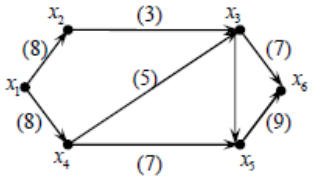
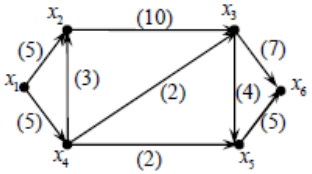
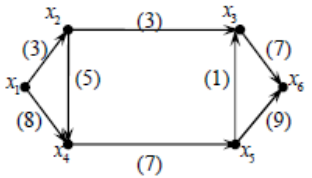
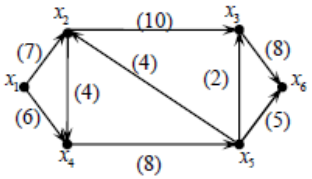
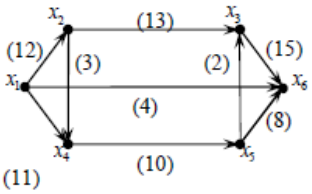
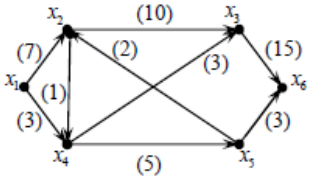
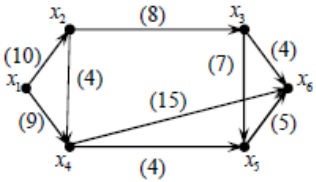
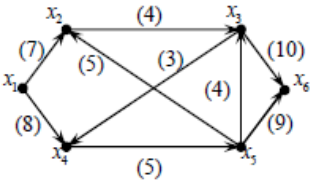
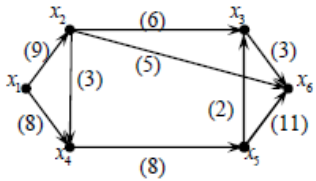
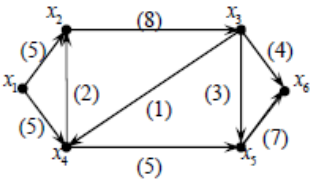
Задача 1.

Обчислити повний потік в транспортній мережі *G* (в дужках зазначено припустимі пропускні здатності дуг).

№ вар.	Завдання	№ вар.	Завдання
01		02	
03		04	
05		06	
07		08	
09		10	

11		12	

№ вар.	Завдання	№ вар.	Завдання
13		14	
15		16	
17		18	
19		20	

№ вар.	Завдання	№ вар.	Завдання
21		22	
23		24	
25		26	
27		28	
29		30	

Завдання 2-6

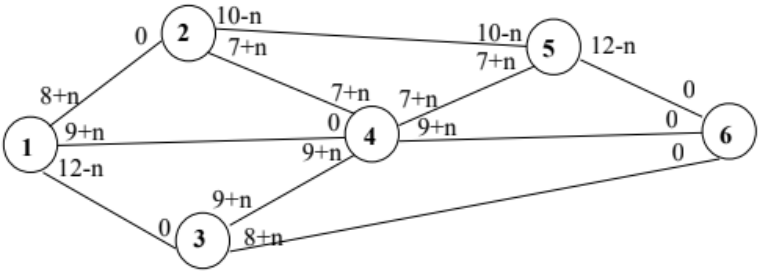
Вираз поряд із вершиною (10-n) дає значення для ***вихідного*** *потоку з вершини,* ***n*** – остання цифра акаунту.

Задача 2.

Система автодоріг, що проходять через область, може забезпечити пропускні спроможності, показані на малюнку (тисячі автомашин на годину).

1) Який максимальний потік через цю систему (тис. автомашин на годину)?

2) Скільки автомашин на годину повинно проїхати по дорозі 5-6, щоб забезпечити максимальний потік?

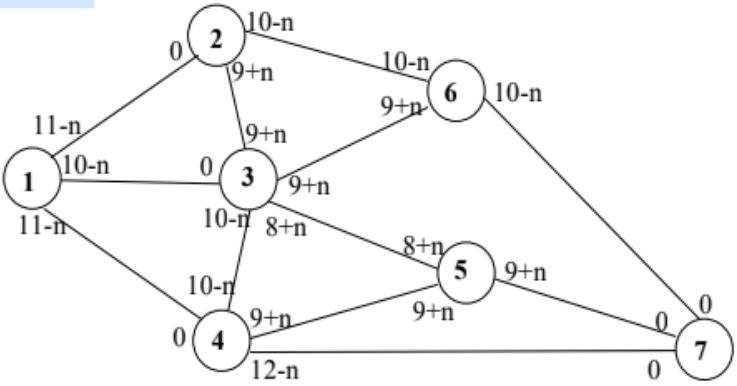


Задача 3.

Телефонна компанія використовує підземну кабельну мережу лінії зв'язку для забезпечення якісного аудіозв'язку між великими містами (вузли 1 і 7 мережі). Переговори здійснюються через серію кабельних ліній і з'єднують їх вузли мережі, як це показано на малюнку. На ньому показано також число телефонних переговорів (тис.), яке допускається одночасно в будь-якій точці часу.

1) Яка максимальна кількість телефонних переговорів між двома містами, яка може бути допущена одночасно (тис. шт.)?

2) Яке число телефонних переговорів має забезпечуватися кабелем 4-7?

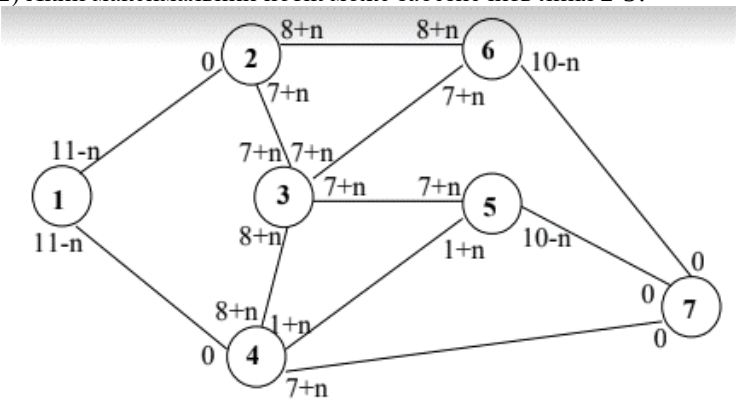


Задача 4.

Нафтова компанія володіє мережею нафтопроводів, через які нафта перекачується від родовищ до нафтосховищ. Частина цієї мережі представлена на малюнку (пропускна здатність нафтопроводів показана в тис. т/год).

1) Якщо компанія хоче поставити нафту в сховище 7 і повністю використовувати пропускну здатність системи, то скільки часу займе поставка 10 тис. т. нафти?

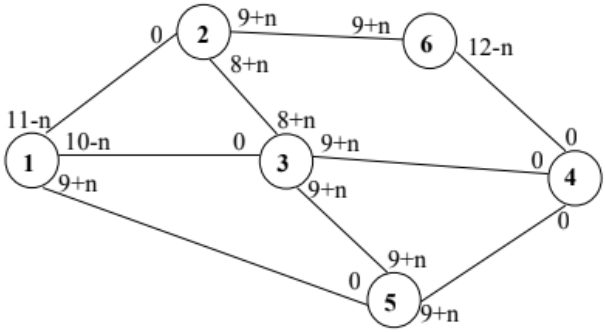
2) Який максимальний потік може забезпечить лінія 2-3?



Задача 5

1) Чому дорівнює максимальний потік автомашин (кількість автомашин на годину) для системи автодоріг, представленої на малюнку?

2) Який максимальний потік може забезпечить дорога 3-4?

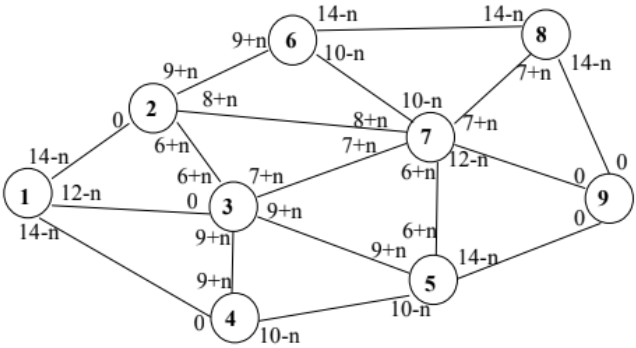


Задача 6

Хімічний завод має мережу труб, призначених для переміщення рідких хімічних продуктів з одних частин підприємства в інші. Мережа труб і пропускні спроможності (т/хв) показані на малюнку.

1) Який максимальний потік для системи, якщо завод збирається перегнати з вузла 1 до вузла 9 стільки рідких хімікатів, скільки це можливо?

2) Скільки хімікатів буде надходити через секцію 3-5?



Рекомендовані джерела інформації

1. Кулаковська І. В. Дискретна математика. Ч.1. Множини, відношення та математичні основи криптографії: метод. вказівки для виконання лабораторних робіт з дисципліни «Дискретна математика» студентами спеціальності 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз» / І. В. Кулаковська . – Миколаїв: Вид-во ЧНУ ім. Петра Могили, 2021. – 100 с. – (Методична серія; вип. 362). https://dspace.chmnu.edu.ua/jspui/handle/123456789/501
2. Кафедра інтелектуальних інформаційних систем ЧНУ. Дискретна математика: відеокурс. - [Електронний ресурс]. - Режим доступу: https://www.youtube.com/@user-lb7ch7lw8e
3. Дискретна математика: Конспект лекцій (Частина 1) [Електронний ресурс]: навч. посіб. для студ. спеціальності 113 «Прикладна математика», освітньої програми «Наука про дані та математичне моделювання» / О. Л. Темнікова ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,97 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 154 с. https://ela.kpi.ua/server/api/core/bitstreams/990893b6-f853-408a-8476-d3dd7c89d2a1/content
4. Темнікова О. Л. Дискретна математика: практикум з дисципліни «Дискретна математика» для студентів спеціальності 113 «Прикладна математика» [Електронне видання] / О. Л. Темнікова – К. : КПІ ім. Ігоря Сікорського, 2018.
5. Капітонова Ю. В., Кривий С. Л., Летичевський О. А., Луцький Г. М., Печорін М. К. Основи дискретної математики. – Київ: Наук. думка, 2002.
6. Журавчак Л. М. Дискретна математика для програмістів. Навчальний посібник. – Львів: Видавництво Львівської політехніки, 2019. – 420 с.
7. Дискретна математика: підручник: гриф МОН України / Ю. М. Бардачов, Н. А. Соколова, В. Є. Ходаков; за ред. В.Є. Ходакова. – 2-ге вид., перероб. і доп. – К.: Вища школа, 2007. – 383 с.
8. Молчановський О. Дискретна математика: курс лекцій. – [Електронний ресурс]. – Режим доступу: http://oim.asu.kpi.ua/courses/discrete-math/.
9. Кулабухов С. Ю. Дискретна математика: конспект лекцій. – [Електронний ресурс]. – Режим доступу: http://window.edu.ru/library/pdf2txt/131/4713/233381.
10. Алексєєв В. Б. Дискретна математика: курс лекцій. – [Електронний ресурс]. – Режим доступу: http://mathcyb.cs.msu.su/paper/books/dmcour.pdf.

11. Кузнецов О. П. Дискретна математика: відеокурс лекцій. – [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/1049/317/info.

12. Пастор А. В., Соколов Д. О. Основи дискретної математики: відеокурс лекцій. – [Електронний ресурс]. – Режим доступу: http://compscicenter.ru/program/course/DiscreteMath2012

ДЛЯ НОТАТОК

Навчальне видання

Інесса Василівна
КУЛАКОВСЬКА

ДИСКРЕТНА МАТЕМАТИКА. Частина 2. ТЕОРІЯ ГРАФІВ, ДЕРЕВА. АЛГОРИТМИ НА ГРАФАХ

Методичні рекомендації для виконання лабораторних робіт з дисципліни «Дискретна математика» студентами спеціальностей 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз»

Методичні рекомендації

Випуск 440

Редактор *О. Михайлова*
Комп'ютерна верстка, дизайн обкладинки *К. Гросу-Грабарчук*
Друк *С. Волинець*. Фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 19.04.2024.
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Ум. друк. арк. 6,2. Обл.-вид. арк. 2,2.
Тираж 50 пр. Зам. № 6792.

Видавець і виготовлювач: ЧНУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.