Прикарпатський національний університет імені Василя Стефаника

Факультет математики та інформатики

Кафедра алгебри та геометрії

ДИПЛОМНА РОБОТА

на здобуття першого (бакалаврського) рівня вищої освіти

на тему "Екстремальні задачі на графах. Програмна реалізація"

Студента 4 курсу, групи М-41

Спеціальність «Математика»

Байляка Назарія Романовича

Керівник: канд. фіз.-мат.наук

Микицей О.Я

Рецензент: канд. фіз.-мат.наук

Копорх К.М.

м. Івано-Франківськ – 2025 рік

ЗМІСТ

АНОТАЦІЯ

Дипломна робота присвячена вивченню теорії графів та її застосуванню для розв’язання прикладних задач, зокрема задачі пошуку найкоротшого шляху. У роботі розглянуто історичні передумови виникнення теорії графів, основні типи графів, поняття ізоморфізму, підграфів, шляхів і циклів. Значну увагу приділено аналізу алгоритмів пошуку найкоротших шляхів, таких як алгоритм Дейкстри, Беллмана-Форда, Флойда–Воршелла та A*. На основі теоретичного аналізу реалізовано програму, що здійснює пошук найкоротшого маршруту між містами з використанням алгоритму Дейкстри. Результати дослідження підтверджують ефективність графових моделей у вирішенні складних задач моделювання та оптимізації в реальних умовах.

ABSTRACT

The thesis is devoted to the study of graph theory and its application to solving practical problems, particularly the shortest path problem. The paper explores the historical background of graph theory, the main types of graphs, as well as the concepts of isomorphism, subgraphs, paths, and cycles. Special attention is given to analyzing algorithms for finding shortest paths, including Dijkstra's algorithm, Bellman-Ford, Floyd-Warshall, and A*. Based on the theoretical research, a software application was developed to find the shortest route between cities using Dijkstra's algorithm. The results confirm the effectiveness of graph-based models for solving complex modeling and optimization problems in real-world scenarios.

ВСТУП

Сьогодні теорія графів — це не просто математичний підхід, а універсальний ключ до розв'язання багатьох задач, таких як оптимізація маршрутів доставки товарів у мегаполісі, аналіз соціальних мереж, які формують нашу інформаційну реальність, пошук найкоротших шляхів, структуризація великих обсягів даних, побудова ефективних алгоритмів у програмному забезпеченні, а також розробка стратегій кіберзахисту. Її застосування виходить далеко за межі інформаційних технологій. Біологія, екологія, економіка — кожна з цих сфер потребує розуміння складних взаємозв’язків, і графи слугують ефективним інструментом для їхнього моделювання. Теорія графів — це не лише засіб розв’язання задач, а спосіб мислення, що дозволяє по-новому аналізувати складні системи і відповідати на виклики сучасного світу.

Метою курсової роботи є продемонструвати застосування теорії графів на різних прикладах, а також реалізувати обрані алгоритми на практиці за допомогою програмної моделі.

Основними завданнями роботи було ознайомлення з основними принципами теорії графів; виявлення ролі графів як математичної моделі для опису складних систем; аналіз прикладів практичного використання графів у різних галузях; розробка програмної реалізації одного з ключових алгоритмів теорії графів та демонстрація її застосування для розв’язання конкретної задачі.

1. Аналіз основних принципів теорії графів.

Розкрити фундаментальні основи: що робить графи такими універсальними? Як вузли та ребра перетворюються на потужні інструменти для вирішення задач? Глибокий аналіз спрямований на те, щоб викристалізувати їхню сутність у простих і складних моделях.

2. Виявлення ролі графів як математичної моделі.

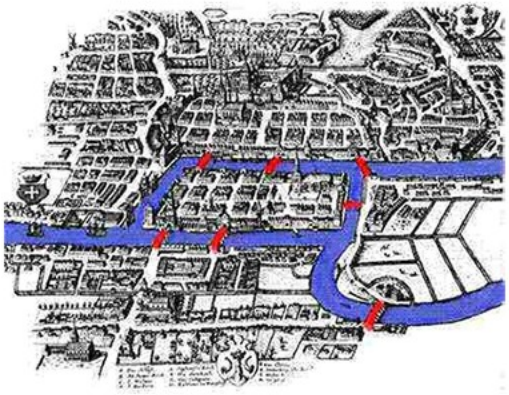
Дослідити, чому саме графи використовуються для моделювання різноманітних систем. Чи дійсно це найкращий спосіб представлення взаємозв’язків, чи є альтернативи, які здатні конкурувати? Визначити, які саме аспекти реальності вони відображають найкраще.

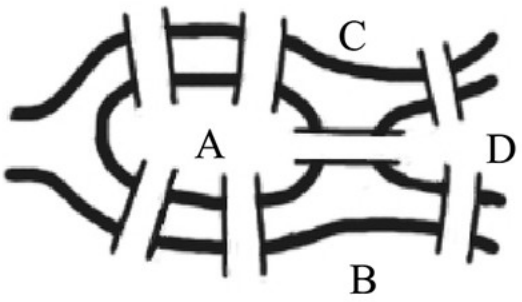
Об’єктом дослідження є графи як універсальна математична модель, що використовується для аналізу, моделювання та вирішення складних задач у різних сферах.

Предметом дослідження є практичне застосування теорії графів для розв'язання задачі знаходження найкоротшого шляху.

ВИНИКНЕННЯ ТЕОРІЇ ГРАФІВ

Витоки теорії графів пов’язують із задачею, сформульованою і розв’язаною Л. Ейлером у XVIII столітті. У місті Кенігсберг існувало сім мостів, що з’єднували чотири частини міста, розташовані по обидва боки річки та на острові. Необхідно було з’ясувати, чи існує маршрут, який дозволяє пройти кожен міст рівно один раз і завершити маршрут у тій самій точці, звідки він починався. [4]





На перший погляд, задача виглядає як така, що може бути розв’язана методом підбору: достатньо спробувати кілька шляхів. Проте Ейлер довів, що незалежно від кількості спроб, знайти такий маршрут неможливо. Сутність його підходу полягала в абстрагуванні реального розташування об’єктів. Він зазначив, що для розв’язання задачі не має значення, як саме розміщені частини міста, — важливо лише, як вони з’єднані між собою мостами.

Для моделювання ситуації Ейлер замінив частини міста на вершини графа, а мости — на ребра між цими вершинами. У такий спосіб було побудовано граф, у якому досліджувалася можливість побудови циклу, що проходить через кожне ребро рівно один раз. Цей підхід поклав початок новій галузі математики — теорії графів.

Важливим спостереженням стало те, що для того, аби з однієї частини міста (тобто з вершини графа) вийти і потім повернутися назад, ця вершина повинна мати парну кількість інцидентних ребер — по одному для входу та

виходу з кожного візиту. Винятком можуть бути лише дві вершини — початкова та кінцева, які можуть мати непарну кількість ребер. Отже, для існування шляху, що проходить через усі ребра графа рівно один раз (так званого ейлерового шляху), граф повинен мати рівно нуль або дві вершини з непарним степенем. Якщо ж усі вершини мають парний степінь і граф зв'язаний, то існує **ейлерів цикл**.

Означення.

Цикл у графі, що проходить через кожне ребро рівно один раз і починається та закінчується в одній і тій самій вершині, називається ***ейлеровим циклом****. Граф, який містить ейлерів цикл, називається* ***ейлеровим графом****.*

Теорема (Ейлера). *Зв'язаний неорієнтований граф має ейлерів цикл тоді й тільки тоді, коли всі його вершини мають парний степінь.*

У графі, що моделює Кенігсберзьку задачу, усі чотири вершини мають непарний степінь. Отже, цей граф не задовольняє умову теореми і не є ейлеровим. Таким чином, побудова маршруту, що проходить через усі сім мостів рівно один раз, є **неможливою**.

Значення цієї задачі виходить за межі її конкретного формулювання. Ейлер не лише відповів на запитання щодо маршруту, але й змінив сам підхід до розв'язання подібних задач. Він показав, що суть задачі полягає не у вимірюванні чи точних обчисленнях, а у вивченні структури зв'язків між об'єктами. Абстрагування задачі до системи точок і ліній дало змогу вивчати її засобами нової галузі математики — теорії графів. Відтоді ця теорія набула широкого розвитку і сьогодні застосовується у фізиці, біології, лінгвістиці, соціології, інформатиці та багатьох інших науках, що мають справу зі складними структурами зв'язків.

ОСНОВНІ ПОНЯТТЯ ТЕОРІЯ ГРАФІВ

2.1 Означення графа

Граф є однією з фундаментальних структур у математиці, що широко використовується в різних галузях науки та техніки. Його зручно уявляти як множину об'єктів, які пов’язані між собою певними відношеннями. Формально граф визначають як пару $G = (V, E)$, де V — скінченна множина вершин, а $E \subseteq V \times V$ — множина ребер, тобто пар вершин, що визначають з'єднання між ними.

У випадку, якщо елементи множини E є невпорядкованими парами, граф називається неорієнтованим, а якщо впорядкованими — орієнтованим. Елементи множини V зазвичай зображають точками на площині, а елементи множини E, тобто ребра, — лініями, що з'єднують відповідні пари вершин. У такому графічному поданні вершини вважають суміжними, якщо існує ребро, що їх з'єднує, а саме ребро є інцидентним обом вершинам. Граф, у якому одна вершина може бути з’єднана з іншою декількома ребрами, тобто в якому допустимі кілька однакових пар у множині E, називають мультиграфом, а такі ребра — кратними. Якщо ребро поєднує вершину саму з собою, його називають петлею, а граф, що містить петлі або кратні ребра, називають псевдографом.

Граф без ребер називають нуль-графом, а якщо у графі немає жодної вершини, такий граф вважається порожнім. Кількість вершин графа називають його порядком, а кількість ребер — розміром графа. Усі ці означення є основою для подальшого вивчення властивостей графів і методів їх аналізу.

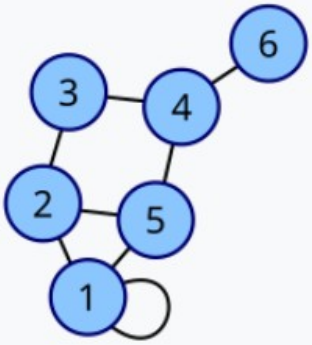
2.2 Способи задання графів

У теорії графів існує багато способів подання, що дають змогу ефективно моделювати структури та аналізувати їхні властивості. Серед них числові способи особливо виділяються своєю формальністю та точністю. Одним із таких є метод, у якому граф представляється у вигляді квадратної булевої матриці, що описує зв’язки між вершинами [2].

Ця матриця, відома як **матриця суміжності**, має розмір n × n, де n — кількість вершин у графі. Елементи матриці вказують, чи існує ребро між парою вершин. Якщо між вершинами i і j є ребро, то відповідний елемент дорівнює 1 (або значенню ваги ребра для зваженого графа), а якщо ребра немає, то записується 0. Для неорієнтованого графа ця матриця є симетричною, що спрощує її аналіз і обробку.

$$a_{ij} = \begin{cases} 1, \text{ коли вершини } v_i \text{ і } v_j \text{ суміжні,} \\ 0 - \text{в протилежному випадку.} \end{cases}$$

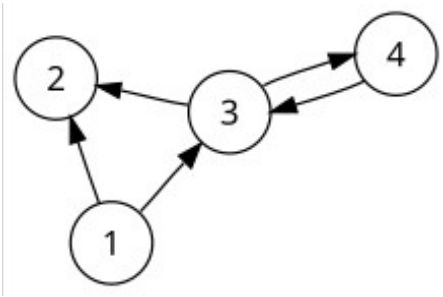
Внаслідок симетричності відношення суміжності матриця суміжностей графа має бути симетричною, а число одиниць в і–му рядку – дорівнювати степені вершини u_i

Маркований граф	Матриця суміжності
	$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ Координати 1-6.

Крім числових методів, існують способи, які зосереджуються на взаємозв’язках між вершинами та ребрами. Одним із них є подання графа у вигляді прямокутної матриці, що враховує інцидентність вершин і ребер.

Такий спосіб, який називається **матрицею інцидентності**, дозволяє описати зв’язки між вершинами та ребрами. Він є зручним для аналізу структури графів, особливо коли важливо знати, які вершини інцидентні певним ребрам. Матриця має розмір n × m, де n — кількість вершин, а m — кількість ребер. У ній кожен стовпець відповідає ребру, а рядок — вершині. Якщо вершина інцидентна ребру, у відповідній клітинці записується 1 (або +1/-1 для орієнтованих графів).

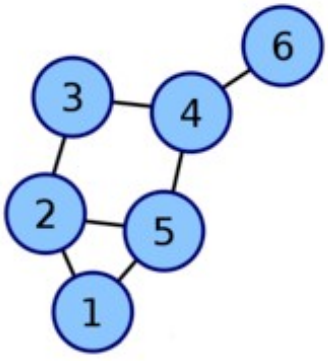
Якщо граф:



- $k_1 = (1, 2)$
- $k_2 = (1, 3)$
- $k_3 = (3, 2)$
- $k_4 = (3, 4)$
- $k_5 = (4, 3)$

то матриця інцидентності виглядатиме так:

$$M = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Неорієнтований граф	Матриця інцидентності
	$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Числові способи подання графів часто доповнюються візуальними методами, які надають більше інтуїтивного розуміння їхньої структури. Найбільш очевидним прикладом такого підходу є представлення **графа у вигляді рисунка.**

У **графічному поданні** вершини зображаються точками, а ребра — лініями, що їх з’єднують. Цей підхід дає змогу швидко виявляти основні властивості графа, такі як зв’язність, наявність циклів або ізольованих вершин. Хоча цей спосіб добре працює для невеликих графів, він стає менш ефективним для великих графів через складність візуалізації.

Окрім графічних методів, для текстового опису графів часто використовуються більш компактні способи подання. Наприклад, для кожної вершини можна зберігати перелік суміжних із нею вершин.

Цей спосіб, відомий як **список суміжних вершин**, дозволяє зберігати інформацію про безпосередні зв’язки кожної вершини. Це ефективний метод для рідких графів, оскільки він використовує значно менше пам’яті порівняно з іншими способами. Він також добре підходить для реалізації алгоритмів, таких як пошук у глибину (DFS) або пошук у ширину (BFS).

Для збереження інформації про зв’язки між вершинами існує ще один метод, що є альтернативою списку суміжних вершин. У цьому методі кожне ребро записується окремо.

Список ребер описує кожне ребро як пару вершин, які воно з’єднує. Цей підхід є компактним і зручним для зважених графів, оскільки до кожної пари можна додати вагу ребра. Список ребер часто використовується у задачах, пов’язаних із роботою з ребрами, таких як пошук мінімального остового дерева або аналіз потоків.

[1,2]

[1,3]

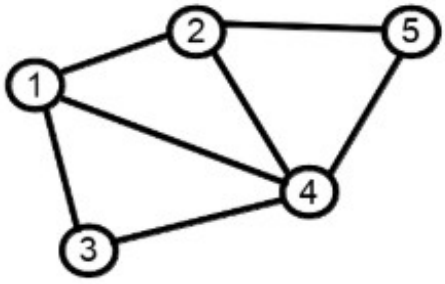
[1,4]

[2,5]

[4,5]

[2,4]

[3,4]

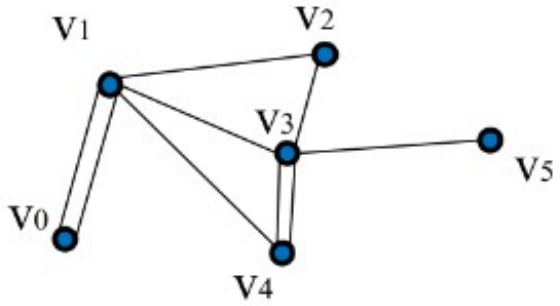


2.3 Основні типи графів

Графи становлять собою ключові математичні моделі, що використовуються для відображення взаємозв’язків між різними об’єктами. Одним із базових різновидів таких структур є простий граф**.** Простий граф визначається як пара (G = (V, E)), де V — це непорожня скінченна множина вершин, а E — множина невпорядкованих пар різних елементів із V, які називаються ребрами. У графах цього типу між кожною парою вершин дозволяється щонайбільше одне ребро, при цьому петлі — тобто ребра, які повертаються до тієї ж самої вершини, — не дозволяються. Наприклад, граф із вершинами V = {v1, v2, v3, v4} і ребрами E = {{v1, v2}, {v1, v3}, {v2, v3}, {v3, v4}} демонструє зв'язки між об'єктами без дублювань. [3]

Аксіома*: У простому графі між будь-якими двома вершинами існує не більше одного ребра, і жодна вершина не з'єднана із собою (петлі не допускаються).*

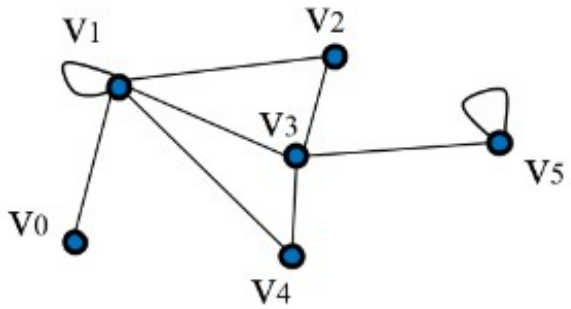
Хоча простий граф зручний для багатьох задач, іноді необхідно враховувати декілька зв’язків між одними й тими ж об’єктами. У подібних ситуаціях доцільно використовувати **мультиграф.** Мультиграф визначається як пара (G = (V, E)), де V — це скінченна множина вершин, а E — сім’я невпорядкованих пар різних елементів множини V. На відміну від простого графа, мультиграф допускає кратні ребра, тобто кілька зв'язків між одними й тими самими вершинами. Наприклад, граф із вершинами V = {v1, v2, v3} і ребрами E = {{v1, v2}, {v1, v2}, {v2, v3}} є мультиграфом, оскільки зв'язок між v1 і v2 дублюється.



***Теорема**: Кожен мультиграф можна подати у вигляді множини простих графів.*

Окрім кратних ребер, у багатьох прикладних задачах важливо враховувати також наявність зворотних зв’язків у структурі об’єкта. У таких випадках доцільно використовувати **псевдограф.** Псевдограф є узагальненням мультиграфа, оскільки він допускає не лише декілька ребер між одними й тими самими вершинами, але й наявність петель. Петля — це таке ребро, що починається і завершується в одній і тій самій вершині.

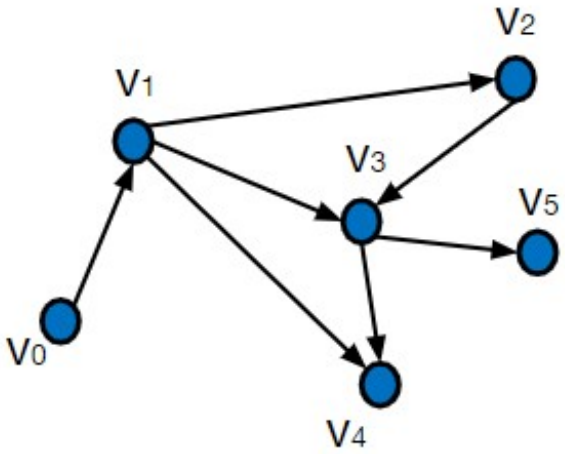
Наприклад, граф із вершинами V = {v1, v2} і ребрами E = {{v1, v1}, {v1, v2}} є псевдографом через наявність петлі у вершині v1.



***Теорема:** Псевдограф є узагальненням простого графа і мультиграфа.*

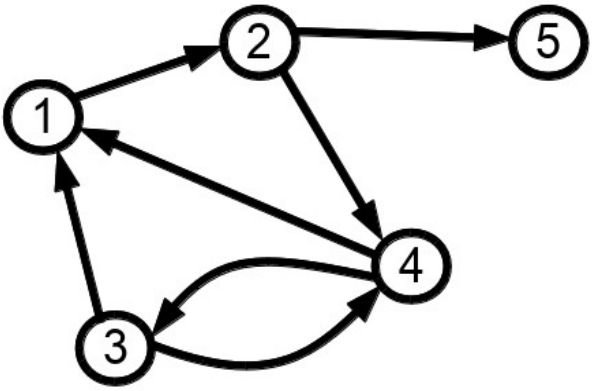
Для багатьох задач важливо враховувати напрямок зв'язків. **Орієнтований граф** дозволяє це зробити завдяки впорядкованості ребер. Кожна дуга має чітко визначений початок і кінець, що робить цей тип графа корисним для моделювання таких систем, як дорожні мережі або потоки інформації.

Наприклад, орієнтований граф із вершинами V = {v1, v2, v3} і дугами E = {(v1, v2), (v2, v3), (v3, v1)} демонструє спрямовані зв'язки між вершинами.



***Аксіома**: У орієнтованому графі кожне ребро має чітко визначений напрямок.*

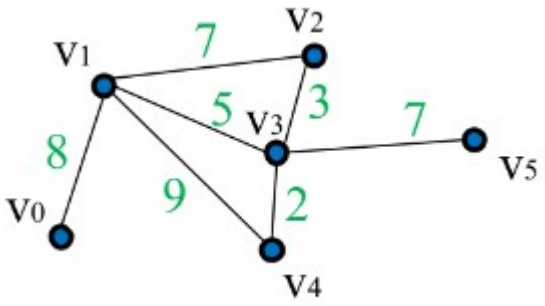
Орієнтовані мультиграфи розширюють можливості орієнтованих графів, дозволяючи кілька дуг із однаковим напрямком між одними й тими самими вершинами. *Визначення: Нехай G = (V, E) — орієнтований мультиграф. Множина E може включати кратні дуги між одними й тими ж вершинами*. Це особливо корисно для моделювання повторюваних зв'язків або паралельних процесів. Наприклад, орієнтований мультиграф із вершинами V = {v1, v2} і дугами E = {(v1, v2), (v1, v2), (v2, v1)} демонструє кілька зв'язків між вершинами.



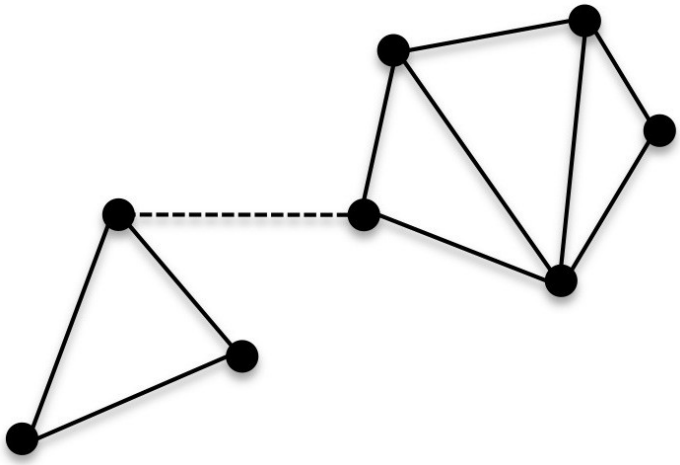
[3]

2.4 Властивості графів

Графи можуть мати різні властивості залежно від типу задач, які вони описують, та умов, що накладаються на їхні вершини й ребра. Одна з поширених характеристик — це наявність ваги на ребрах. **Такі графи називають зваженими.** У них кожне ребро має числове значення, що інтерпретується як вага. Ця вага може означати відстань між пунктами, час подолання маршруту, вартість переходу чи інші параметри. Зважені графи широко застосовуються в транспортних мережах, логістиці, навігаційних системах і задачах оптимізації.

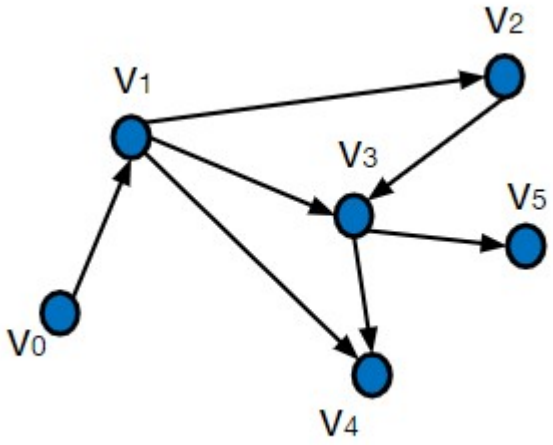


Іншою важливою характеристикою є зв'язність графа. Граф називається з'єднаним, якщо між будь-якою парою його вершин існує хоча б один шлях. Інакше кажучи, можна дістатися з будь-якої вершини до будь-якої іншої, переходячи через послідовність ребер. Ця властивість є ключовою при аналізі структурованості мереж, зокрема соціальних або інформаційних.



Окремо виділяють орієнтовані графи, в яких кожне ребро має напрям. У таких графах кожне ребро подається як впорядкована пара вершин, що означає

наявність чітко визначеного початку та кінця. Напрямленість ребер відіграє важливу роль у моделюванні потоків, алгоритмів маршрутизації та аналізі причинно-наслідкових зв’язків.



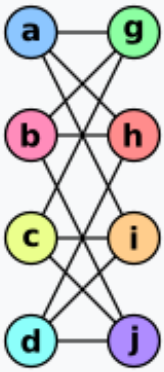
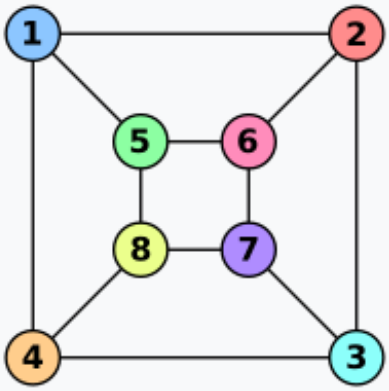
2.5 Ізоморфізм та підграфи

Ізоморфізм графів — це взаємно-однозначне відображення між вершинами двох графів, яке зберігає їх структуру, тобто відповідність ребер. Якщо два графи ізоморфні, вони вважаються еквівалентними за своєю структурою. А підграфом називається граф, вершини та ребра якого є підмножинами відповідних елементів іншого графа. Остовний підграф охоплює всі вершини початкового графа, зберігаючи лише частину його ребер.

Нехай $G = (V, E)$, $H = (V_1, E_1)$ — графи, і існує відображення $h: V \to V_1$, яке є взаємно-однозначною відповідністю (тобто $|V| = |V_1|$). Відображення h називається ізоморфізмом між графами G і H, якщо для будь-яких вершин u і v графа G виконується така умова: їх образи h(u) і h(v) у графі H з'єднані ребром тоді і тільки тоді, коли u і v з'єднані ребром у графі G. Якщо таке відображення h існує, графи G і H називаються ізоморфними. Відношення ізоморфізму є еквівалентністю. Як правило, ізоморфні графи вважаються

ідентичними.

Граф H = (V', E') називається підграфом графа G = (V, E), якщо V' ⊆ V і E' ⊆ E. Якщо H є підграфом G, то H належить графу G. Підграф H графа G називається остовним підграфом, коли V' = V.

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

2.6 Шляхи та цикли.

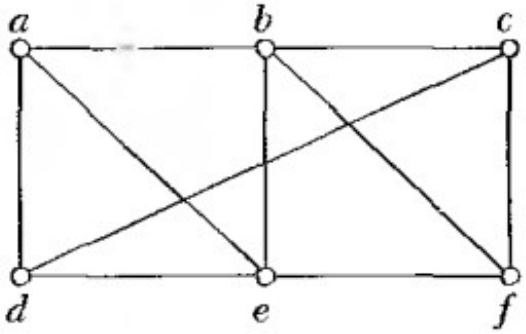
Шляхом у неорієнтованому графі між вершинами u та v називають послідовність ребер довжиною r, яка задається як: $e_1 = \{x_0, x_1\}$, $e_2 = \{x_1, x_2\}$, ..., $e_r = \{x_{r-1}, x_r\}$, де $x_0 = u$, $x_r = v$, а r є натуральним числом. Таким чином, шлях довжини r складається з r ребер, і кожне ребро e_i враховується стільки разів, скільки воно зустрічається в послідовності. Вершини u та v вважаються крайніми, а всі інші вершини, через які проходить шлях, називають внутрішніми. Циклом у графі називається такий шлях, який починається і завершується у тій самій вершині, тобто u = v.

Для простого графа шлях можна описати як послідовність вершин, через які проходить цей шлях:

$x_0, x_1, x_2, \ldots, x_{r-1}, x_r$.

Шлях або цикл вважається простим, якщо у ньому не повторюються одні й ті самі ребра. Якщо шлях з'єднує дві вершини u та v, його позначають як ⟨u, v⟩ і називають ⟨u, v⟩-шляхом.

Приклад. На (рис. 1) представлений простий граф. У цьому графі послідовність вершин a, d, c, f, e є простим шляхом довжиною 4, оскільки пари ребер {a, d}, {d, c}, {c, f}, {f, e} є складовими цього шляху. Проте послідовність d, e, c, b не утворює шляху, адже пара {e, c} не є ребром графа. Також у графі присутній цикл довжиною 4: b, c, f, e, b. Цей цикл утворений послідовністю ребер {b, c}, {c, f}, {f, e}, {e, b}, а його початок і кінець припадають на одну й ту саму вершину b. Послідовність a, b, e, d, a, b, що має довжину 5, не є простим шляхом, оскільки вона двічі проходить через ребро {a, b}.



Теорема. *Між кожною парою різних вершин зв'язного неорієнтованого графа існує простий шлях.*

ЗАСТОСУВАННЯ ГРАФІВ

3.1 Прикладне застосування графів

Використання теорії графів у мережевих технологіях

Теорія графів активно застосовується в галузі мережевих технологій, оскільки дозволяє моделювати та аналізувати різноманітні системи. Серед основних напрямів її використання вирізняються графічне моделювання мереж та їхній подальший аналіз. Представлення даних у вигляді графів дає змогу по-новому підходити до складних задач, спрощуючи їх структуру та забезпечуючи точніше формулювання. Водночас теорія мереж пропонує широкий інструментарій для вивчення властивостей графів за допомогою алгоритмів та аналітичних методів.

Поняття "граф" і "мережа" в основі мають спільну структуру — вершини (вузли) та ребра (зв’язки). Проте вживаються вони в різних контекстах: термін "граф" здебільшого використовують у математичному середовищі, тоді як "мережа" — у фізиці, комп’ютерних науках і соціальних дослідженнях. Завдяки великій кількості типів графів і мереж ці поняття ефективно застосовуються для моделювання як простих зв’язків, так і складних систем взаємодії.

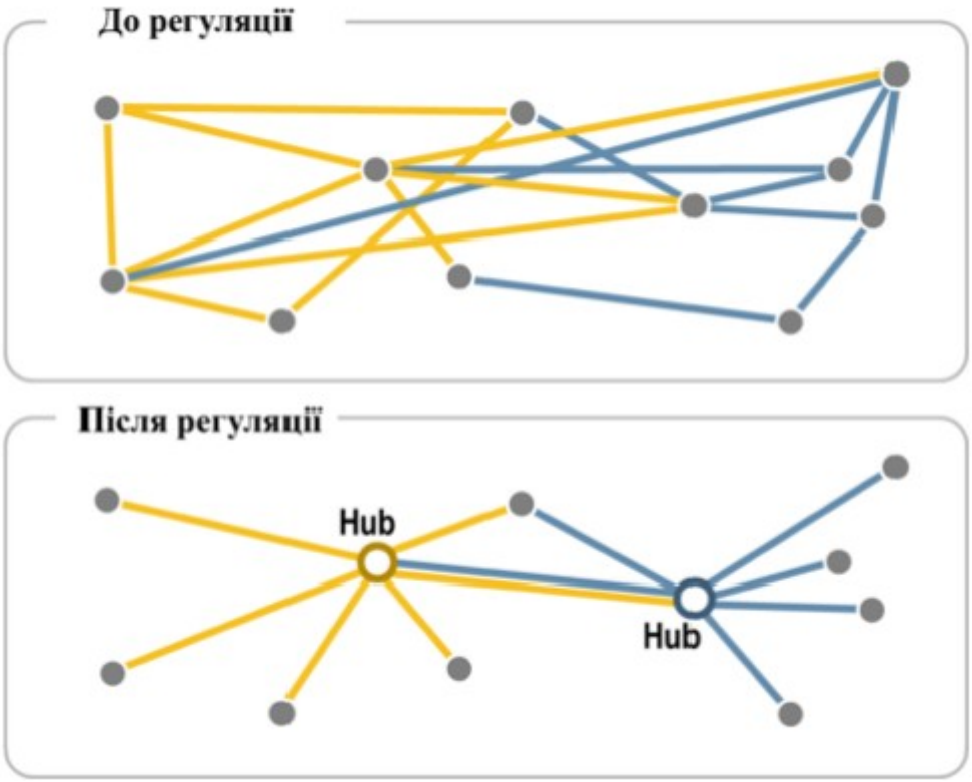
3.2 Розклад авіакомпаній (проблеми з потоком)

Теорія графів ефективно застосовується для розв'язання задач, пов’язаних із потоками, які часто виникають у практичних сферах, зокрема в плануванні авіаперевезень. Авіакомпанії здійснюють рейси між різними містами світу, і кожен із них потребує залучення екіпажу. Оскільки персонал зазвичай закріплений за конкретним містом, неможливо забезпечити всі рейси довільними членами екіпажу. Для раціонального формування розкладу польотів використовуються методи графової теорії.

У цьому контексті рейси розглядаються як інформація для побудови орієнтованого графа: міста виступають у ролі вершин, а напрямлені дуги

представляють маршрути від пункту відправлення до пункту прибуття. Така структура інтерпретується як модель потоку. Кожне ребро має свою вагу — кількість працівників, необхідних для здійснення конкретного рейсу. Щоб завершити модель, додають спеціальні вершини: джерело, що символізує місто, де базується екіпаж, і стік, який узагальнює усі пункти призначення.

Завдяки графовому підходу можна обчислити мінімальний потік, що забезпечує всі маршрути, тобто найменшу кількість екіпажу, необхідну для виконання всіх рейсів. Крім того, беручи до уваги значущість окремих міст, є змога створити оптимізований розклад, що мінімізує ресурси, не враховуючи менш важливі напрямки.



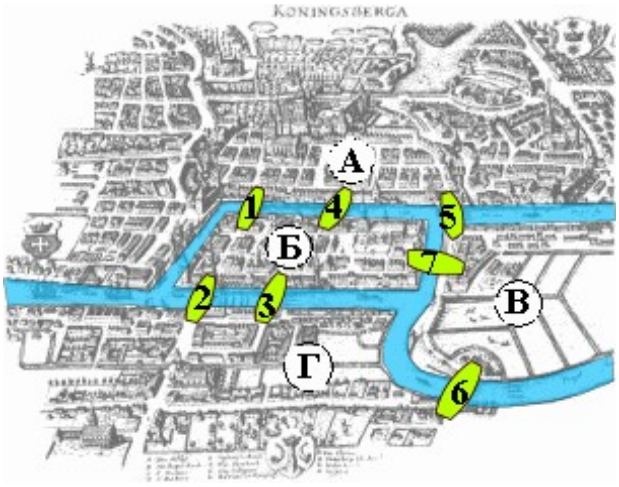
3.3 Напрямки на карті (Найкоротший шлях)

Сучасні смартфони стали незамінними у повсякденному житті, допомагаючи вирішувати безліч рутинних завдань. Наприклад, завдяки ним я можу легко спланувати велосипедний маршрут від свого місцезнаходження до найближчого бару або ресторану. Але як саме відбувається розрахунок такого маршруту? В основі цього процесу лежить теорія графів, яка використовується для пошуку найкоротшого шляху.

Спочатку карта місцевості трансформується у граф. У цьому графі вершинами виступають перехрестя, а дороги між ними утворюють ребра. Кожному ребру призначається певна вага — наприклад, довжина шляху або час, потрібний для його подолання. У містах із одностороннім рухом утворюється орієнтований граф, що відображає напрямки руху транспортних потоків.

Щоб знайти оптимальний маршрут між двома точками, використовується алгоритм, який обирає шлях із найменшою сумою ваг ребер, що з'єднують відповідні вершини. У невеликих мережах це може бути доволі просто, однак у масштабних міських графах із тисячами вершин і зв'язків задача значно ускладнюється. На щастя, для цього існують потужні алгоритми — як-от алгоритм Дейкстри або метод А, які дозволяють знаходити точні або наближені рішення.

Пошук найкоротшого або найшвидшого шляху між двома точками — це, безперечно, одне з найважливіших прикладних застосувань теорії графів. Однак ця задача не обмежується лише навігацією: її активно використовують у соціальних мережах для аналізу зв'язків (зокрема, теорії «шести рукостискань»), а також у сфері телекомунікацій — для оптимального маршруту передавання інформації з мінімальними затримками. [1]



3.4 Методика PageRank

Останніми роками спостерігається стрімке зростання використання теорії графів, особливо в Інтернет-середовищі. Зокрема, графові моделі застосовують для вдосконалення рекламних стратегій. Аналіз взаємозв'язків між користувачами, їхніх уподобань, соціальної активності та взаємодій у таких мережах, як Facebook, Vkontakte чи Twitter, дає змогу суттєво підвищити результативність рекламних кампаній. Згідно з дослідженнями, між будь-якими двома сторінками у Facebook існує ланцюжок із не більше ніж 12 переходів.

Ця ідея має підґрунтя в соціологічних експериментах XX століття, де було висунуто гіпотезу, що будь-які двоє людей у світі з'єднані через шість посередників. Відмінності між кількістю зв'язків у реальному житті та у віртуальних мережах пояснюються тим, що соціальні платформи охоплюють лише частину людства й не охоплюють усі типи взаємин — родинні, професійні, побутові тощо.

У веб-просторі сторінки можна змоделювати у вигляді графа, де кожна сторінка є вершиною, а гіперпосилання між ними — ребрами. Раніше пошукові системи працювали переважно з ключовими словами, ранжуючи сторінки за кількістю переглядів, однак такий підхід не гарантував релевантності результатів пошуку.

Сучасні пошукові системи, як-от Google, впроваджують графові алгоритми для визначення значущості сторінок. Одним із найвідоміших є алгоритм PageRank, розроблений Сергієм Бріном і Ларрі Пейджем. Його основна ідея полягає в тому, що цінність веб-сторінки залежить від кількості та авторитетності ресурсів, які на неї посилаються. Ця концепція стала можливою завдяки попереднім дослідженням, зокрема роботам 1976 року, які мали на меті розробку системи рейтингів для наукових журналів.

Методика PageRank оцінює значущість сторінок, враховуючи кількість і якість посилань, а також додаткові фактори, що дозволяють ранжувати сторінки з високою точністю.

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \cdots + \frac{PR(T_n)}{C(T_n)}\right)$$

де d – коефіцієнт згасання, який може бути в межах від 0 до 1, звичайно дорівнює 0,85;

$C(T_i)$ – число вихідних посилань сторінкиT_i;

PR(Xi) – це випадкові величини, сума яких для всієї мережі дорівнює 1.

Ранг сторінки в пошуковій системі Google залежить від двох основних чинників: кількості вхідних та вихідних посилань. При цьому вхідні посилання на саму сторінку мають нейтральний вплив — вони не знижують її ранг, але й суттєвого підвищення не забезпечують. Щодо вихідних посилань, можна зазначити, що їхня значна кількість на сторінці T_i, яка посилається на сторінку A, або зовсім не вплине, або вплине мінімально на рейтинг PR(A).

Це пояснюється тим, що значення дробу $\frac{PR(T_i)}{C(T_i)}$ у формулі демонструє рівень «автономності» сторінки. Якщо сторінка містить багато посилань на інші джерела, її цінність може знижуватися, адже користувачі можуть звернутися безпосередньо до цих першоджерел. Таким чином, зростання кількості вихідних посилань призводить до зменшення числового значення показника, що впливає на ранг сторінки в загальній системі ранжування. пов'язано з тим, що дріб

$\frac{PR(T_i)}{C(T_i)}$ наближається до 0 ($0 \leq PR(T_i) \leq 1$, при значному збільшенні вихідних посилань у знаменнику C (T_i), дріб зменшується.

PageRank є лише одним із інструментів, які використовуються пошуковою системою Google для визначення порядку відображення інтернет-сторінок у результатах пошуку. Він відіграє роль коефіцієнта, який показує, наскільки сторінка відповідає пошуковому запиту користувачів. Чим вищий показник PageRank, тим більше шансів, що сторінка з’явиться на перших позиціях у списку результатів пошуку. [1]

Практичні приклади є ще одним важливим аспектом цього дослідження. Графи активно використовуються в інформаційних технологіях для визначення маршрутів передавання даних, у логістиці для оптимізації транспортних шляхів, у соціальних мережах для виявлення структур впливу між користувачами, а також у біології — для моделювання складної архітектури молекулярних процесів. Усі ці приклади свідчать про одне: графи — це не просто математичний інструмент, а спосіб пізнання взаємозв’язків у складних системах.

У межах цієї роботи було розглянуто базові положення теорії графів: її історичне становлення, ключові типи графів, поняття ізоморфізму, підграфів, а також особливості шляхів і циклів. Значна увага була приділена способам подання графів та їх прикладному використанню. Проведений аналіз підтверджує широкий потенціал теорії графів як у фундаментальних дослідженнях, так і в практичних застосуваннях, що відкриває широкі горизонти для подальшого впровадження в науково-технічні сфери.

ОГЛЯД АЛГОРИТМІВ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ

Задача пошуку найкоротшого шляху між двома вершинами графа належить до фундаментальних проблем у галузі інформатики та прикладної математики. Вона знаходить широке застосування в таких сферах, як комп'ютерні мережі, геоінформаційні системи, транспортна логістика, розробка навігаційного програмного забезпечення, штучний інтелект, ігрова індустрія тощо. Для розв'язання цієї задачі розроблено низку алгоритмів, які відрізняються за складністю, точністю, сферою застосування та обмеженнями. У цьому розділі буде розглянуто найвідоміші з них, а саме: алгоритм Дейкстри, алгоритм Беллмана-Форда, алгоритм Флойда–Воршелла та алгоритм A* (A-star).

Одним із найбільш поширених і ефективних у випадках, коли всі ваги ребер у графі є невід'ємними, є алгоритм Дейкстри. Він був запропонований нідерландським науковцем Едсгером Дейкстрою у 1956 році. Суть алгоритму полягає в тому, що на кожному кроці вибирається вершина з мінімальною відстанню від початкової точки серед усіх ще не оброблених вершин, після чого відстані до її сусідів оновлюються у разі, якщо знайдено коротший шлях. Такий підхід забезпечує ефективне знаходження найкоротших шляхів у графах з позитивними вагами ребер. Завдяки використанню черги з пріоритетом (наприклад, на основі бінарної купи), алгоритм має складність $O((V + E) \log V)$, де V – кількість вершин, а E – кількість ребер. До переваг цього методу належить висока швидкість та масштабованість на розріджених графах. Проте він не може бути застосований у випадку, якщо в графі присутні ребра з від'ємною вагою.

Іншим поширеним методом є алгоритм Беллмана-Форда, що був незалежно розроблений американськими дослідниками Річардом Беллманом та Лестером Фордом у середині XX століття. На відміну від алгоритму Дейкстри, цей алгоритм дозволяє працювати з графами, що містять від'ємні ваги. Принцип його роботи полягає у поступовому оновленні відстаней до всіх вершин шляхом проходження усіх ребер графа протягом $V - 1$ ітерацій. Після завершення циклів можливе проведення додаткової перевірки на наявність циклів з від'ємною вагою. Основний недолік алгоритму полягає у значно вищій часовій складності – $O(V \times E)$, що ускладнює його використання на великих графах, однак саме його

універсальність і здатність знаходити оптимальні рішення навіть у складних умовах забезпечили йому важливе місце в теорії графів.

Коли постає задача знаходження найкоротших шляхів між усіма парами вершин у графі, доцільним є використання алгоритму Флойда–Воршелла. Цей метод реалізує підхід динамічного програмування, поступово оновлюючи матрицю відстаней між вершинами з урахуванням усіх можливих проміжних вершин. Його обчислювальна складність становить $O(V^3)$, що робить його непридатним для дуже великих графів, проте цілком ефективним у задачах з обмеженою кількістю вершин, зокрема для повних або щільних графів. Він також здатен коректно працювати з від'ємними вагами, якщо в графі відсутні цикли з від'ємною сумою ваг.

Окрему нішу займає алгоритм A*, що є модифікацією алгоритму Дейкстри і активно використовується в задачах пошуку шляху в умовах реального часу – зокрема, в робототехніці, геонавігації та комп'ютерних іграх. Цей алгоритм відрізняється застосуванням евристичної функції, яка оцінює передбачувану відстань від поточної вершини до цільової. Завдяки цьому він здатен істотно скорочувати кількість опрацьованих вузлів, не втрачаючи при цьому оптимальності, за умови, що евристика є допустимою (тобто не переоцінює відстані). Складність роботи A* варіюється залежно від обраної евристики, і в найгіршому випадку вона відповідає складності алгоритму Дейкстри. У разі якісного вибору евристичної функції A* здатен демонструвати вражаючу ефективність.

Підсумовуючи, варто зазначити, що кожен з описаних алгоритмів має свої переваги та доцільність використання залежно від типу графа, наявності від'ємних ваг, необхідності знаходження одного шляху чи усіх, а також від вимог до швидкодії. У таблиці нижче подано порівняльний аналіз характеристик основних алгоритмів пошуку найкоротшого шляху:

Таким чином, алгоритм Дейкстри, обраний у рамках даної дипломної роботи, є обґрунтованим вибором для пошуку найкоротшого шляху в орієнтованому або неорієнтованому графі з невід'ємними вагами. Його ефективність, простота реалізації та широке практичне застосування роблять його одним із базових інструментів у задачах маршрутизації.

ОПИС РЕАЛІЗАЦІЇ ПРОГРАМИ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ

У цьому розділі буде розглянуто програмну реалізацію алгоритму Дейкстри для знаходження найкоротшого шляху між двома містами. Програма написана мовою програмування Python, оскільки ця мова є зручною для реалізації алгоритмів, містить розвинену стандартну бібліотеку та дозволяє швидко створювати інтерфейс командного рядка. Основна мета програми — дати користувачу змогу ввести початкове та кінцеве місто й отримати як довжину найкоротшого шляху, так і сам маршрут.

Програма працює з попередньо визначеним графом, у якому міста представлені у вигляді вершин, а відстані між ними — як ваги ребер. Реалізація використовує структуру даних словника, де ключем є назва міста, а значенням — список кортежів із сусідніми містами та відстанями до них.

5.1. Ініціалізація графа

На початку програми задається граф у вигляді словника Python. Наприклад:

```
graph = {
    "Київ": [("Житомир", 140), ("Чернігів", 150)],
    "Житомир": [("Київ", 140), ("Рівне", 200)],
    "Рівне": [("Житомир", 200), ("Луцьк", 70)],
    "Луцьк": [("Рівне", 70)],
    "Чернігів": [("Київ", 150), ("Суми", 200)],
    "Суми": [("Чернігів", 200), ("Харків", 180)],
    "Харків": [("Суми", 180)],
}
```

Скріншот 1. Фрагмент коду з графом міст

Кожне місто пов’язане з іншими містами, до яких веде безпосередній маршрут. Вага ребра (відстань у кілометрах) зазначається у вигляді числа.

5.2. Функція алгоритму Дейкстри

Основна логіка пошуку найкоротшого шляху реалізована у функції dijkstra(graph, start, end). Алгоритм використовує чергу з пріоритетом (реалізовану за допомогою модуля heapq), щоб завжди обирати місто з найменшою поточною відстанню.

```
def dijkstra(graph, start, end):
    # Відстані від початкової вершини
    distances = {city: float('inf') for city in graph}
    distances[start] = 0
    previous = {city: None for city in graph}
    queue = [(0, start)]

    while queue:
        current_distance, current_city = heapq.heappop(queue)
```

Скріншот 2. Початок реалізації алгоритму

У цій частині ініціалізується словник distances, де кожне місто має початкову відстань ∞, окрім стартового, яке має відстань 0. Словник previous зберігає попереднє місто для кожного, щоб згодом можна було відновити маршрут. Далі розпочинається цикл, у якому на кожній ітерації обробляється місто з найменшою відстанню.

5.3. Оновлення відстаней

У тілі циклу відбувається оновлення відстаней до суміжних міст, якщо знайдено коротший шлях через поточне місто:

```
for neighbor, weight in graph[current_city]:
    distance = current_distance + weight

    if distance < distances[neighbor]:
        distances[neighbor] = distance
        previous[neighbor] = current_city
        heapq.heappush(queue, (distance, neighbor))
```

Скріншот 3. Оновлення відстаней до сусідів

Таким чином, алгоритм "розширює" вже відомі найкоротші шляхи, зберігаючи інформацію про оптимальний маршрут.

5.4. Відновлення шляху

Після завершення роботи основного циклу потрібно побудувати маршрут на основі словника previous:

```
path = []
current = end
while current is not None:
    path.insert(0, current)
    current = previous[current]
```

Скріншот 4. Побудова шляху за допомогою словника previous

Цей код послідовно повертається від кінцевої вершини до початкової, створюючи список вершин (міст), що входять до складу найкоротшого маршруту.

5.5. Взаємодія з користувачем

Користувач вводить початкове і кінцеве міста. Програма перевіряє, чи існують ці міста в графі, та викликає функцію алгоритму:

```
start_city = input("Введіть початкове місто: ")
end_city = input("Введіть кінцеве місто: ")

if start_city in graph and end_city in graph:
    distance, path = dijkstra(graph, start_city, end_city)
    if path:
        print(f"Найкоротший шлях від {start_city} до {end_city}: {' -> '.join(path)}")
        print(f"Загальна довжина: {distance} км")
    else:
        print("Неможливо знайти шлях між цими містами.")
else:
    print("Одне або обидва введені міста відсутні в графі.")
```

Скріншот 5. Приклад запуску програми в терміналі

Результатом роботи є друк найкоротшого шляху та його довжини. У разі відсутності шляху або некоректного вводу користувача програма виведе відповідне повідомлення.

```
Введіть кінцеве місто:
Житомир
Найкоротший шлях від Суми до Житомир: Суми -> Чернігів -> Київ -> Житомир
Загальна довжина: 490 км
```

```
Введіть початкове місто:
Миколаїв
Введіть кінцеве місто:
Київ
Одне або обидва введені міста відсутні в графі.
```

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено задачу пошуку найкоротшого шляху в графі та реалізовано програмне забезпечення, яке вирішує цю задачу за допомогою алгоритму Дейкстри. Проведений теоретичний аналіз засвідчив, що алгоритм Дейкстри є одним з найефективніших способів знаходження найкоротшого шляху в графах з невід'ємними вагами ребер. Він забезпечує високу продуктивність, зокрема при роботі з великими, але розрідженими графами, що часто зустрічаються в реальних задачах навігації, планування маршрутів, транспортної логістики тощо.

У реалізованій програмі користувач має змогу ввести початкове та кінцеве місто, після чого програма виводить найкоротший маршрут і відповідну відстань. Граф представлено у вигляді структури даних «словник» (dictionary), що дозволяє гнучко змінювати або доповнювати список міст і зв'язків між ними. Основна логіка побудована на використанні черги з пріоритетом, яка забезпечує ефективний відбір наступної вершини для обробки.

Покроковий опис функціонування програми, супроводжений поясненнями та фрагментами коду, демонструє простоту і наочність реалізації обраного методу. З огляду на це, програму можна легко модифікувати або розширювати, наприклад, додавши графічний інтерфейс, функцію імпорту графа з файлу або підтримку реальних географічних координат.

Проведене дослідження підтвердило доцільність використання алгоритму Дейкстри в задачах, де необхідно швидко і точно знайти найкоротший маршрут без урахування від'ємних ваг. Завдяки своїй ефективності та логічній простоті алгоритм може бути рекомендований як базовий інструмент при розробці систем навігації, пошукових механізмів у графах і транспортних моделей.

Програму, отриману в дипломній роботі можна адаптувати для інтеграції з візуальними інтерфейсами або розширити для роботи з реальними картографічними даними

ДОДАТОК

```
import heapq


def dijkstra(graph, start, end):
    distances = {city: float('inf') for city in graph}
    distances[start] = 0
    previous = {city: None for city in graph}
    queue = [(0, start)]

    while queue:
        current_distance, current_city = heapq.heappop(queue)
        if current_distance > distances[current_city]:
            continue

        for neighbor, weight in graph[current_city]:
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_city
                heapq.heappush(queue, (distance, neighbor))

    path = []
    current = end
    while current is not None:
        path.insert(0, current)
        current = previous[current]
```

```
    if distances[end] == float('inf'):
        return None, []

    return distances[end], path

graph = {
    "Київ": [("Житомир", 140), ("Чернігів", 150)],
    "Житомир": [("Київ", 140), ("Рівне", 200)],
    "Рівне": [("Житомир", 200), ("Луцьк", 70)],
    "Луцьк": [("Рівне", 70)],
    "Чернігів": [("Київ", 150), ("Суми", 200)],
    "Суми": [("Чернігів", 200), ("Харків", 180)],
    "Харків": [("Суми", 180)],
}

start_city = input("Введіть початкове місто: ")
end_city = input("Введіть кінцеве місто: ")

if start_city in graph and end_city in graph:
    distance, path = dijkstra(graph, start_city, end_city)
    if path:
        print(f"Найкоротший шлях від {start_city} до {end_city}: {' -> '.join(path)}")
        print(f"Загальна довжина: {distance} км")
    else:
```

```
        print("Неможливо знайти шлях між цими містами.")
else:
    print("Одне або обидва введені міста відсутні в графі.")
```

ЛІТЕРАТУРА

1. **Бобрицька Г.С.** Прикладне застосування теорії графів у різних сферах життя суспільства та окремої особистості // **Фізико-математична освіта** : науковий журнал. – 2017. – Випуск 3(13). – С. 26-30.

2. **DOU.** Основні алгоритми в теорії графів [Електронний ресурс] // *DOU* – 2024. – Режим доступу: https://dou.ua/forums/topic/48433/

3. **Тарануха В.Ю.** Задачі на графах для курсу Алгориміка: Навчальний посібник для студентів факультету комп'ютерних наук та кібернетики.

4. **Disted.** Навчальний курс з теорії графів [Електронний ресурс] // *Центр дистанційної освіти* – Режим доступу: https://disted.edu.vn.ua/courses/learn/14042