Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики технологій факультету інформатики

Порівняльний аналіз алгоритмів пошуку найкоротшого шляху в графах з вагами

Текстова частина до курсової роботи
за спеціальністю “Інженерія програмного забезпечення” 121

Керівник курсової роботи
ст. в., Картавий М.О.
_____
(підпис)
“____” _____ 2025 р.
Виконала студентка
Трюхан Т.Г.
“____” _____ 2025 р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

проф., д.ф.-м.н.

_____М. М. Глибовець

(підпис)

„____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Трюхан Тетяні Григорівні факультету інформатики 3-го курсу

ТЕМА: Порівняльний аналіз алгоритмів пошуку найкоротшого шляху в графах з вагами

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Анотація
Вступ
Розділ 1. Теоретичні основи дослідження
Розділ 2. Дослідження роботи алгоритмів пошуку найкоротшого шляху в графах з вагами
Розділ 3. Реалізація практичної частини
Розділ 4. Порівняння алгоритмів
Висновки
Список використаної літератури

Дата видачі „___” _____ 2025 р. Керівник _____

(підпис)

Завдання отримала _____

(підпис)

Календарний план виконання курсової роботи

Тема: Порівняльний аналіз алгоритмів пошуку найкоротшого шляху в графах з вагами

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	11.10.2024	
2.	Пошук тематичної літератури.	01.11.2024	
3.	Ознайомлення з тематичними матеріалами та побудова структурної та теоретичної частин курсової роботи.	30.11.2024	
4.	Написання вступу та змісту роботи.	08.12.2024	
5.	Аналіз алгоритмів пошуку найкоротшого шляху в зважених графах.	25.12.2024	
6.	Написання відповідних розділів курсової роботи.	15.01.2025	
7.	Підготовка набору даних для тестування алгоритмів.	24.01.2025	
8.	Програмна реалізація алгоритмів та розробка застосунку для візуалізації їх роботи.	10.04.2025	
9.	Тестування створеної програми.	11.04.2025	
10.	Узагальнення результатів і написання відповідних розділів курсової роботи.	15.04.2025	
11.	Оформлення роботи відповідно до вимог написання курсової роботи.	19.04.2025	
12.	Узгодження попередньої версії роботи з керівником.	20.04.2025	
13.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника.	01.05.2025	
14.	Створення презентації та написання доповіді для захисту курсової роботи.	10.05.2025	

15.	Захист курсової роботи.	15.05.2025	

Студентка Трюхан Тетяна Григорівна
Керівник Картавий Микола Олексійович
“_____” _____ 2025 р.

ЗМІСТ

АНОТАЦІЯ

У межах цієї курсової роботи досліджуються основні з понять в теорії графів та алгоритми пошуку найкоротшого шляху у зважених графах. Особлива увага приділена принципу роботи алгоритмів A*, Беллмана-Форда, Флойда-Уоршала. У процесі виконання даної роботи було створено демонстраційний застосунок з їх реалізацією, згенеровано набори тестових графів та оцінено й порівняно алгоритми за такими критеріями: застосовність до окремих видів графів, коректність, час виконання, використання пам’яті, кількість перевірених вузлів та простота реалізації.

Ключові слова: граф, зважений граф, алгоритм пошуку найкоротшого шляху, алгоритм A*, алгоритм Беллмана-Форда, алгоритм Флойда-Уоршала, список суміжності, матриця суміжності, евристика, порівняння алгоритмів.

ВСТУП

Усе в світі можливо представити як дані. З того моменту, як людство вступило в епоху диджиталізації, обсяг інформації, яку необхідно опрацьовувати, невпинно зростає. Віддавна і по теперішній час одним з основоположних завдань залишається знаходження оптимального рішення на основі масиву даних. Частина таких задач може бути розв’язана шляхом їх представлення як проблеми пошуку найкоротшого маршруту в графах. Алгоритми пошуку найкоротшого шляху є ключовою складовою вирішення питань, пов’язаних з навігацією, логістикою, маршрутизацією, а тому найчастіше застосовуються, зокрема, в Global Positioning System, протоколах маршрутизації, розробці ігор. Наявні алгоритми працюють з різними типами графів і розв’язують широке коло задач, однак зі збільшенням кількості та ускладненням структури даних, вже згодом їх буде недостатньо. Зрештою настане момент, коли виникне потреба в покращенні або створенні інших алгоритмів, але пошук нових, удосконалених рішень не можливий без аналізу існуючих, розуміння їх особливостей, переваг та недоліків.

Метою даної роботи є визначити, як впливають на швидкість прокладання маршрутів обрані алгоритми пошуку найкоротшого шляху в зважених графах різних типів та створити демонстраційний застосунок.

Об’єктом дослідження даної курсової є алгоритми пошуку найкоротшого шляху в графах.

Предметом дослідження є ефективність алгоритмів пошуку найкоротшого шляху в графах, а саме: A*, Беллмана-Форда, Флойда-Уоршала.

Постановка задачі

1. Дослідити графи: їх структуру, різновиди, особливості та алгоритми пошуку найкоротшого шляху в зважених графах.
2. Проаналізувати особливості роботи алгоритмів A*, Беллмана-Форда, Флойда-Уоршала.
3. Реалізувати демонстраційний застосунок.
4. Проаналізувати та порівняти ефективність роботи обраних алгоритмів на різних графах.

Для виконання завдань наукового будуть застосовані такі методи наукового дослідження: теоретичний для дослідження графів та алгоритмів та емпіричний в аналізі роботи алгоритмів на конкретних графах та порівнянні їх ефективності.

У даній роботі буде використано теоретичні матеріали з мережі Інтернет: з форумів та публічних наукових статей.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1 Поняття та типи графів

Алгоритми пошуку шляху створюються з врахуванням особливостей структури графів, що безпосередньо впливає на обмеження їх застосування, коректність та швидкість виконання. Тому насамперед розглянемо загальні відомості про графи та їх види.

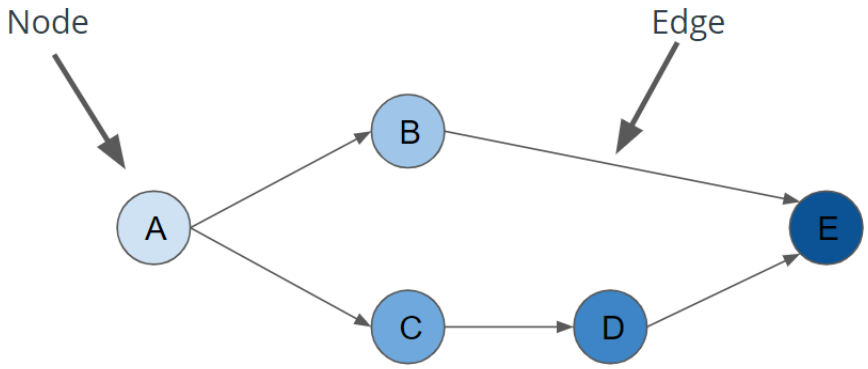


*Рисунок 1.1 – Структура графа, джерело: [https://miro.medium.com/v2/resize:fit:1400/format:webp/0*kLgHKxxJpwbUOlJ5.png]*

Графи – це одна з базових структур даних в програмуванні, яка являє собою скінченну множину вузлів, які називають вершинами, та ребер. Кожне ребро з'єднує між собою пару вершин.

Суміжні вершини – пара вершин, з'єднаних ребром, також називаються сусідніми.

Інцидентність – поняття, що описує зв'язок між ребром та вершиною: інцидентними є пари ребер та вершин, де ребро виходить з даної вершини, або входить в неї.

Маршрут в графі – почергова послідовність з вершин та ребер, яка починається та завершується вершинами, і у якій сусідні елементи є інцидентні.

Довжиною маршруту є кількість пройдених ребер, або для зважених графів – сума ваг усіх ребер з послідовності.

Шлях в графі – це такий маршрут, жодна вершина якого не зустрічається в послідовності двічі. Можливий виняток – цикл, де однакова лише перша та остання вершини.

Цикл в графі – це не нульовий маршрут в графі, де точка початку є також точкою завершення, а ребра та інші вершини не повторюються.

Щільність – відношення кількості ребер в графі, до максимально можливої.

Компонента зв'язності – підграф, множиною вершин якого є найбільша з таких підмножин вершин графа, де між кожними двома вершинами є шлях.

Петля – ребро, яке починається та завершується в одній точці.

Кратні ребра – такі ребра в графі, які з'єднують одну і ту ж пару вершин.

Існують кілька критеріїв, за якими графи можна розділити на різні види, до них належать такі, як орієнтованість, вага ребер, співвідношення вершин та ребер, наявність циклів, зв'язність, наявність петель чи кратних ребер.

Залежно від направленості ребер графи можна розділити на орієнтовані та неорієнтовані. В орієнтованих графах ребра мають напрямок, що вказує на те, що шлях з однієї вершини в іншу існує, а назад – ні. Натомість в неорієнтованих графах, якщо ребро між двома вузлами існує, то це означає, що можна пройти між ними в обох напрямках.

За фактом наявності чи відсутності у ребер ваги, розрізняють зважені та незважені графи. Якщо ребрам присвоюється вага, то такі графи є зваженими, для них алгоритми пошуку найкоротшого шляху враховують не кількість ребер, а суму їх ваг. Окремою підгрупою зважених графів є графи з негативними вагами, де ребрам можуть мати негативні значення.

Також графи можна розглядати як щільні або розріджені. Так, щільними графами є такі, у яких кількість ребер близька до максимальної, тобто такої, де між кожною парою вершин є шлях. Графи, у яких між всіма парами вершин існує шлях, називають повними.

Зокрема вирізняють графи, які мають цикли, тобто такі шляхи, де точка початку є також точкою завершення, і до того ж ребра на шляху не повторюються. Серед таких графів можна виокремити циклічні графи, у яких всі вершини можуть бути об’єднані в єдиний цикл.

Виділяють також незв’язні графи, у яких існує принаймні одна пара вершин, між якими не існує жодного шляху. Такі графи можна розділити на окремі зв’язні компоненти.

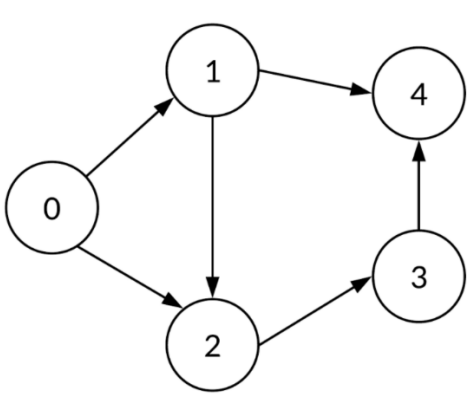
Окремо розглядають графи зі складнішою структурою, мультиграфи та мультиорграфи. Це відповідно неорієнтований та орієнтований графи, які можуть мати кратні ребра і не мають петель.

Також виокремлюють особливі види графів, такі як псевдографи, які дозволяють кратні ребра та петлі, та двочасткові графи – такі що можна розділити на два набори вершин так, щоб між жодною парою вершин в одному наборі не було ребер.

1.2 Представлення графів у програмуванні

Існує кілька варіантів представлення графів у програмуванні. Це зумовлено різноманітністю особливості їх структури, вимогами до обсягу використовуваної пам’яті, складністю реалізації та спектром передбачених для створених графів задач. Обрана структура даних є одним з факторів, які можуть впливати на ефективність роботи алгоритмів, тому важливо розглянути особливості найпоширеніших з підходів.

Список суміжності: з використанням такого підходу граф може бути реалізований як двовимірний масив, де перший вимір відведено на представлення вершин, а відповідають їм масиви з переліком сумісних вузлів. Окрім цього поширеною імплементацією є словник, де ключу та значенню відповідають вершина та масив з сусідніми їй. Варто зауважити, що в таких випадках ребра фактично визначаються як направлені. Тоді, щоб створити неорієнтований граф, для кожного ребра створюється обернена копія, що по суті все ще є орієнтованим графом, де між кожною парою вершин є пара протилежно направлених ребер, але це дозволяє розглядати його як неорієнтований, і працювати з ним без значних змін у алгоритмах. За такого підходу для позначення зважених ребер у списках сусідніх вершин зберігаються не просто їх номери або назви, а пара з назвою та вагою. Списки суміжності доцільно використовувати для представлення розріджених графів.



Adjacency Matrix

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

*Рисунок 1.2 – Приклад матриці суміжності, джерело: [https://miro.medium.com/v2/resize:fit:1400/format:webp/0*XGraRBR3RaxTaWjQ.png]*

Матриця суміжності: така форма подання графа дозволяє імплементувати як орієнтовані та неорієнтовані графи, так і зважені та незважені. Суть полягає у тому, що вершинам графа відповідають певні колонки та рядки матриці розміру V * V (тут V позначає кількість вузлів графа), а в клітинках на їх перетині за допомогою

булевого значення, представленого одиничкою або нулем, зазначається відповідно наявність або відсутність ребра, яке їх пов’язує. Таким чином для неорієнтованого графа матриця буде симетричною відносно діагоналі. Коли потрібно реалізувати зважений граф, це робиться дуже просто, тоді замість булевого значення в клітинку вписується вага ребра. Проте за такого підходу потрібно ретельно перевіряти заповнену матрицю на відповідність структурі графа, оскільки помилково може бути використаний нуль або від’ємна одиниця на позначення відсутності ребра, коли малась на увазі його вага. У таких випадках доцільніше буде використати спеціальні об’єкти на позначення пустого значення або математичної нескінченності. Матриці суміжності – дуже зручний спосіб представлення графів, проте варто зважати на той факт, що він також займає більше місця в пам’яті, а тому для розріджених графів його використовувати не оптимально, якщо немає явних причин для його застосування.

Список ребер: в такому варіанті графи представлені власне списком ребер, де кожне ребро являє собою кортеж або об’єкт, у яких зберігаються значення вихідного та вхідного вузлів, а для зважених графів також – вага присвоєна ребру. Цей спосіб реалізації графів у програмі є доволі простим, але людині отриману структуру важче сприймати.

Матриця інцидентності: ця стратегія для представлення графів відрізняється від вище описаної матриці суміжності тим, що рядкам та колонкам тут ставляться у відповідність вершини та ребра, тобто розмір матриці становить V * E (тут V позначає кількість вершин, а E – кількість ребер графа). На перетині рядків та колонок записуються значення, які вказують, чи інцидентні вершина та ребро. Для позначення направленості ребер у матрицях інцидентності вказують значення як одиницю, якщо ребро виходить з вершини, і від’ємну одиницю, якщо навпаки.

1.3 Види задач на графах

За час існування такого поняття як графи, їх структура ускладнювалася, відображаючи все складніші системи з нашого життя, а тому не дивно, що закономірно і коло задач, які потребували розв’язку, та питань, які лишались без відповідей, розширювалося. Згодом було окреслено проблеми з вищою складністю, що й дотепер очікують на вирішення.

Основою цього дослідження є давнє питання, яке є ключем до розв’язку частини з інших задач, а саме – пошук найкоротшого шляху в графах, втім далі в цьому підрозділі коротко згадаємо й інші.

До найпоширеніших проблем, які розглядаються в теорії графів, зокрема, належать:

- Пошук всіх можливих шляхів між вершинами;
- Пошук кількох найкоротших шляхів;
- Виявлення циклів у графах (зокрема Гамільтонового або Ейлерового);
- Пошук компонент зв’язності графа;
- Пошук найменшого кістякового дерева: мінімальне кістякове дерево графа є підмножиною ребер графа такою, що її розмір або сума ваг ребер для зважених графів є найменшою з можливих за умови збереження зв’язності графа;
- Виявлення мостів у графах: у теорії графів мостом називають таке ребро, видалення якого призведе до появи в графі нових відокремлених компонент зв’язності;
- Розв’язання задач за наявності ребер з негативними вагами;
- Знаходження шляху для передачі потоку;
- Задача комівояжера: полягає у знаходженні такого найкоротшого маршруту, що починається та закінчується в одній вершині, одночасно з тим проходячи інші вузли строго один раз.

1.4 Алгоритми пошуку найкоротшого шляху в зважених графах

З огляду на існування цілого ряду задач на графах, закономірним є те, що послідовно було відкрито численні алгоритми для їх розв'язання. Перші алгоритми були призначені для найпростіших графів, тобто таких, що є неорієнтовані, без ваг та не мають кратних ребер та петель. Проте з часом рішення удосконалювали, вигадували нові, тому насьогодні для більшості видів графів можна знайти алгоритм для розв'язку поставлених задач.

1.4.1 Найвідоміші алгоритми

У даному підрозділі буде коротко оглянуто існуючі алгоритми пошуку найкоротшого шляху в зважених графах, для яких завдань їх використовують, обмеження їх застосування. Найпопулярнішими з них є наступні алгоритми: Дейкстри, Беллмана-Форда, Флойда-Уоршала, A*, Джонсона. На основі даних зібраних з джерел можна порівняти ці алгоритми за таким набором критеріїв: вид розв'язуваної задачі, можливість оперування над графами з від'ємними вагами, і зокрема, виявлення негативних циклів, часова складність за найгіршого випадку.

Таблиця 1.1 – Алгоритми пошуку шляху

Алгоритм	Задача	Робота з від'ємними вагами	Робота з від'ємними циклами	Часова складність
Дейкстри	Найкоротші шляхи з початкової вершини до всіх	Ні	Ні	O((V+E) * log V)
Беллмана-Форда	Найкоротші шляхи з	Так	Може вплинути на	O(V*E)

	початкової вершини до всіх		коректність обчислень	
Флойда-Уоршала	Найкоротші шляхи для всіх пар вершин	Так	Може вплинути на коректність обчислень	O(V^3)
A*	Найкоротший шлях між двома точками	Ні	Ні	Залежить від евристичної функції
Джонсона	Найкоротші шляхи для всіх пар вершин	Так	Ні	O(V^2 log V + V*E)

1.4.2 Покращення алгоритмів

Згадані алгоритми також мають удосконалені версії з підвищеною ефективністю, деякі з них вказано нижче.

Алгоритм Дейкстри: зокрема, в основі наведеного в прикладі алгоритму A* лежить алгоритм Дейкстри, підсилений евристичною функцією з метою оптимізації виконання пошуку. Також існує така варіація, як bidirectional, тобто двонаправлений алгоритм, який одночасно запускає виконання алгоритму з початкової та кінцевої вершин.

Беллмана-Форда: оптимізованим варіантом цього алгоритму є Shortest Path Faster Algorithm. Підвищення ефективності досягається завдяки використанню черги для перевірки тих вузлів, до яких було знайдено коротший шлях.

*A**: існують кілька покращених версій цього алгоритму. На великих графах використовується алгоритм ALT, що є абревіатурою на основі підходів, використаних для розв'язку задачі: A*, landmark, triangle inequality. Його суть полягає у тому, що на графі визначаються вершини, які будуть відігравати роль маячків і для кожного з них визначається найменша відстань до інших вершин, в подальшому використовуються вже результати цих обрахунків. Jump Point Search – прискорений варіант алгоритму A*, який застосовується на сіткових картах і під час виконання перевіряє лише важливі клітинки, ігноруючи такі, до яких шлях від попередньої вершини є коротшим, або такі, для яких існують рівноцінні альтернативні шляхи.

Висновок до розділу 1

У цьому розділі було розглянуто основні терміни та задачі з теорії графів, класифікацію графів та способи їх представлення в програмах. Також, наближаючись до основного завдання цієї роботи, було коротко оглянуто найпоширеніші у використанні алгоритми пошуку найкоротшого шляху у графах та їх покращені версії. Можна побачити, що ідеального алгоритму, який можна буде застосувати для всіх випадків, не існує. Частина з алгоритмів, на приклад, Дейкстри та A* не призначені для опрацювання графів з від'ємними вагами. Тим часом алгоритми Беллмана-Форда та Флойда-Уоршала не лише коректно виконуються за наявності ребер з негативними вагами, а й можуть виявляти негативні цикли. Всі наведені алгоритми мають різну часову складність, але варто зазначити, що вона обрахована для найгірших випадків. Зважаючи на різноманіття можливих особливостей в структурі графів, необхідно досліджувати ефективність роботи таких алгоритмів на графах, які найкраще відображатимуть реальні дані з наданих задач. Також на ефективність та простоту реалізації алгоритмів впливає спосіб представлення графа в програмі. Отже вибір структури даних на позначення графів

та алгоритмів для пошуку шляху в графах є комплексним завданням і має спиратися на особливості в будові графів, їх розміри та очікуваний набір задач.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ РОБОТИ АЛГОРИТМІВ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ В ЗВАЖЕНИХ ГРАФАХ

2.1 Огляд стратегій виконання алгоритмів

За стратегією, що лежить в основі опрацювання графів, серед таких алгоритмів можна виокремити жадібні, динамічні та евристичні.

Жадібними називають такі алгоритми, в основі виконання яких лежить принцип найкращого локального рішення. Для знаходження розв'язку цими алгоритмами, задача розглядається як набір найменших підзадач. Тоді на кожному з кроків обирається оптимальний варіант, не розглядаючи проблему в повному її масштабі. Такі алгоритми у переважній більшості випадків є швидкими і використовують малі обсяги пам'яті, але їх недоліком є можливість повернення не найкращого з рішень, хоч і близького до такого. Яскравим прикладом жадібного алгоритму в графах є алгоритм Дейкстри.

Динамічні алгоритми також поділяють задачу на окремі завдання. Тим часом проміжні результати зберігаються на кожному з етапів виконання і можуть бути використані для подальших обрахунків. Такі алгоритми є дещо повільнішими, але натомість забезпечують коректне рішення і мають ширший спектр проблем, які можуть бути розв'язані з їх використанням. До динамічних належать зокрема й алгоритми Беллмана-Форда, Флойда-Уоршала та Джонсона.

Евристичні алгоритми є найшвидшими з наведених за умови вибору точної евристики. Евристикою називають функцію, результатом виконання якої є приблизна оцінка відстані між кінцевими вершинами в маршруті. Такі алгоритми виконують мінімальну кількість обрахунків, оскільки ігнорують частину можливих шляхів, якщо вони є довшими з урахуванням евристики. Через застосування евристики такі алгоритми не завжди надають найкращий з шляхів, однак

використовують мінімум часу на обробку графів та якнайменше обчислень, тому зазвичай застосовуються на великих графах. Одним з найвідоміших прикладів евристичного алгоритму є A*.

2.2 Дослідження роботи алгоритмів

Із вище зазначених алгоритмів у цій роботі будуть детальніше розглянуті алгоритми Беллмана-Форда, Флойда-Уоршала та A*. У наступних підрозділах описано принцип їх дії.

2.2.1 Алгоритм Беллмана-Форда

Перш за все створюється таблиця, до якої будуть заноситись дані про відстань від вузла, з якого починається шлях, до всіх інших вершин графа. На цьому етапі виконання алгоритму відстань з початкової вершини до самої себе позначається як нульова, а усі інші відстані визначаються як нескінченні. Якщо в задачі достатньо просто знати довжину найкоротшого шляху, то на цьому перший крок можна завершити, проте у більшості випадків на вихід очікується повний маршрут. Тоді необхідно також створити масив, у якому будуть зберігатися попередники вершин.

Основним механізмом в роботі даного алгоритму є релаксація ребер в графі. Для прикладу, надано зважений граф з вершинами A, B, C та ребрами AC, AB, BC. Тоді під час пошуку найкоротших шляхів з точки A у процесі релаксації BC буде визначено, чи можна дістатися з A до C швидше, пройшовши через B. Це легко обраховується шляхом додавання збереженої в таблиці відстані до B та ваги BC і якщо отриманий результат є меншим за зафіксований для шляху AC, то оновлюється значення та вузол-попередник для C. На цьому етапі релаксація усіх ребер графа повторюється V - 1 разів, де V позначає кількість вершин графа.

Останнім кроком є проведення додаткового кола релаксації, що дозволяє виявити присутність негативних циклів. Якщо такий було знайдено, то це означає,

що найкоротший маршрут визначити неможливо, оскільки довжину можна нескінченно зменшувати. Інакше повертається отриманий результат.

2.2.2 Алгоритм Флойда-Уоршала

Першим кроком виконання даного алгоритму для графа з V вузлами є створення матриці розмірністю V * V, до якої буде занесено відстані між вершинами. На початку така матриця подібна матриці суміжності, де рядки та колонки позначають вершини. Відстань вузла до самого себе визначається як нуль. Для відстаней до суміжних вершин береться вага з'єднувального ребра, а для інших встановлюється як нескінченність.

На наступному кроці розглядаються всі можливі проміжні вузли. Тобто для кожної вершини V та усіх пар вершин (V1, V2), на основі даних збережених в матриці, виконується перевірка, чи шлях між V1 та V2 є коротшим, якщо пройти між ними через V. Якщо перевірка була успішною, відстань переписується на нову обраховану. Цей процес вимагає запуску трьох вкладених циклів.

Наприкінці виконується перевірка графа на наявність від'ємних циклів. Якщо в матриці буде принаймні одне від'ємне значення відстані від вузла до себе, то такі існують. Інакше буде отримано коректну матрицю з усіма довжинами шляхів.

Якщо необхідно знайти шлях, а не лише його довжину, то в циклах додатково заповнюється таблиця попередників.

2.2.2 Алгоритм A*

Ключовою частиною цього алгоритму є евристична функція. Оскільки A* найкраще працює на графах з просторовою або геометричною структурою, найпоширеніші евристики ґрунтуються на оцінці відстані між точками. На приклад, для графів, що являють собою сітку застосовують Мангеттенську, діагональну або

Евклідову відстань. Мангеттенська відстань дорівнює сумі модулів різниць координат точок, і застосовується, коли можливий рух лише по вертикалі та горизонталі. Коли рухатись дозволено і по діагоналях, то використовують діагональну відстань, що обраховується як максимуму модулів різниць координат точок. Евклідову відстань використовують, коли можна переміщатись у будь-якому з напрямків.

Алгоритм A* починається зі створення двох списків вершин. У OpenList заносяться відомі вершини, на початку – лише стартова. ClosedList, тобто замкнений список, використовується для збереження досліджених вершин. Усі відомі та досліджені вершини містять вказівник на вузол-попередник, що дозволяє в кінці відтворити пройдений шлях.

На кожному наступному кроці знаходиться вершина, для якої обрахована приблизна відстань до вузла-цілі є найменшою. Орієнтовна відстань для вузла визначається як сума довжини шляху пройденого від початку до нього та евристичної оцінки відстані від вузла до цілі. Тоді вершина заноситься до списку досліджених, а до відкритих додаються її сусіди. Такі суміжні вершини, що вже належать до закритого списку, ігноруються, а для тих, що вже є відкритих, якщо оцінка шляху коротша за наявну, то значення вказівника оновлюється.

Алгоритм зупиняється, коли кінцева вершина потрапляє до списку повністю досліджених.

Висновок до розділу 2

У цьому розділі було розглянуто основні стратегії роботи алгоритмів. Також, з метою поглиблення розуміння процесу роботи обраних алгоритмів перед виконанням практичної частини, було досліджено етапи їх виконання. Усі наведені алгоритми є нескладними для розуміння, але мають свої особливості, на які варто звернути увагу. Оскільки алгоритм A* належить до ряду евристичних, то на його

коректність та час виконання напряму впливає якість обраної евристики, а на коректність шляхів, знайдених алгоритмами Беллмана-Форда та Флойда-Уоршала, може вплинути наявність негативних циклів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРАКТИЧНОЇ ЧАСТИНИ

3.1 Вибір способу представлення графів

Обрані алгоритми мають суттєві відмінності у підході до пошуку шляхів, тому для представлення графів у програмі буде використано кілька варіантів, а саме: список суміжності та матриця суміжності.

Список суміжності краще підходить для розріджених графів, оскільки ігнорує непотрібні дані.

Матриця суміжності є оптимальною для щільних графів і застосовується, коли ефективність алгоритмів залежить від простоти доступу до всіх пар вершин, на приклад, для Флойда-Уоршала.

Додатково можливо реалізувати функції, які дозволятимуть зміну способу представлення графа, якщо такий підхід буде зручнішим для завдання.

3.2 Вибір засобів розробки

3.2.1 Мова програмування

Для реалізації застосунку буде використано мову Python, оскільки вона має вичерпний набір бібліотек та дозволяє мінімізувати обсяг коду.

3.2.2 Бібліотеки та фреймворки

Таблиця 3.1 – Бібліотеки та фреймворки

Бібліотека/Фреймворк	Призначення
NetworkX	Створення графів та виконання функцій над ними

NumPy	Робота з матрицями суміжності
Matplotlib	Візуалізація
Time	Заміри часу
Tracemalloc	Заміри використаного простору
Streamlit	Створення інтерфейсу

3.2.3 Середовище розробки

Застосунок буде створено в середовищі розробки VS Code, оскільки він підтримує всі вище згадані інструменти.

3.4 Опис процесу розробки

3.4.1 Вимоги до функціоналу програми

Щоб забезпечити можливість різностороннього порівняння ефективності алгоритмів застосунок має відповідати наступним вимогам:

1. В програмі має бути реалізовано можливість створювати випадкові зважені графи за заданими параметрами: спосіб представлення (список суміжності, матриця суміжності), розмір (кількість вершин), орієнтованість, щільність, присутність циклів;
2. Додатково варто реалізувати можливість створення графів з вершинами, яким присвоєно координати;

3. В програмі має бути реалізовано алгоритми A*, Беллмана-Форда та Флойда-Уоршала. Варіації Беллмана-Форда: на списку та матриці суміжності, варіації A*: на списку суміжності з використанням Евклідової відстані. При цьому має обраховуватись час виконання та об'єм використаної пам'яті, кількість перевірених вузлів;
4. Має бути можливість демонстрації користувачу створеного графа та запуску на ньому відповідних алгоритмів з виводом результату їх виконання;
5. Результат виконання алгоритму має містити: шлях (перелік вершин), довжину шляху, кількість витраченого часу та використаної пам’яті, кількість перевірених вершин.

Для збору результатів, отриманих на тестових даних очікується наявність таких функцій:

1. Створення заданої кількості випадкових зважених графів на заданій структурі, з заданими параметрами та збереження їх в окремих файлах для можливості повторного використання;
2. Зчитування графів, збережених у графах;
3. Переведення графів в різні способи представлення;
4. Запуск на наборі графів обраного алгоритму та занесення в таблиці таких результатів: коректність (різниця у довжинах отриманого та істинно найкоротшого шляхів), час виконання алгоритму, кількість спожитої пам’яті, кількість перевірених вузлів.

3.4.1 Ключові моменти розробки

Для початку необхідно реалізувати функції для створення випадкових графів. Тут є кілька обмежень, які виникають через об’єднання параметрів:

1. Різна максимальна кількість ребер для орієнтованих та неорієнтованих графів;
2. Щільність зв’язного графа не може бути менша за частку від двох та кількості вершин для неорієнтованого графа, та менша за один відсоток для орієнтованого;
3. Щільність зв’язних графів без циклів знаходиться в межах до п’ятдесяти відсотків для орієнтованих, а для неорієнтованих є сталою, на один меншою від кількості вершин.

Було створено функції для генерації таких типів графів з заданою кількістю вершин та щільністю:

- Неорієнтований без циклів;
- Неорієнтований з циклами;
- Орієнтований слабко-зв'язний без циклів (шлях існує);
- Орієнтований слабко-зв'язний з циклами (шлях існує);
- Орієнтований сильно-зв'язний.

При тестуванні алгоритми шукатимуть шлях між точками 0 та (n - 1). Для орієнтованих графів слабкої зв’язності між цими точками маршрут створюється випадковим чином.

Також для кожного з перелічених варіантів є варіація з координатами. Коли координати не вказані, то вага ребра обирається випадковим чином. Інакше вага ребер обраховується як евклідова відстань з випадковим додатнім відхиленням, що забезпечує коректність спрацювання евристики A*. Ребра розташовуються випадковим чином, що може призвести до того, що реальний найкоротший шлях буде проходити через вершини, які вважаються не оптимальними за евристикою.

Щоб наблизити розташування ребер до такого, за якого евристика буде кращою, було додано такі варіації неорієнтованих графів, які відтворюють системи,

наближені до реальних транспортних. Це досягається завдяки побудові основи графа за мінімальним кістяковим деревом та сортуванню усіх можливих ребер за оцінкою доцільності створення. Така оцінка обраховується з врахуванням довжини існуючого між точками маршруту та Евклідової відстані між ними. Найдоцільнішим вважається створення ребра з найбільшим маршрутом і найменшою Евклідовою відстанню між кінцевими вузлами. До того ж при побудові забороняється додавати ребра, що перетинають наявні.

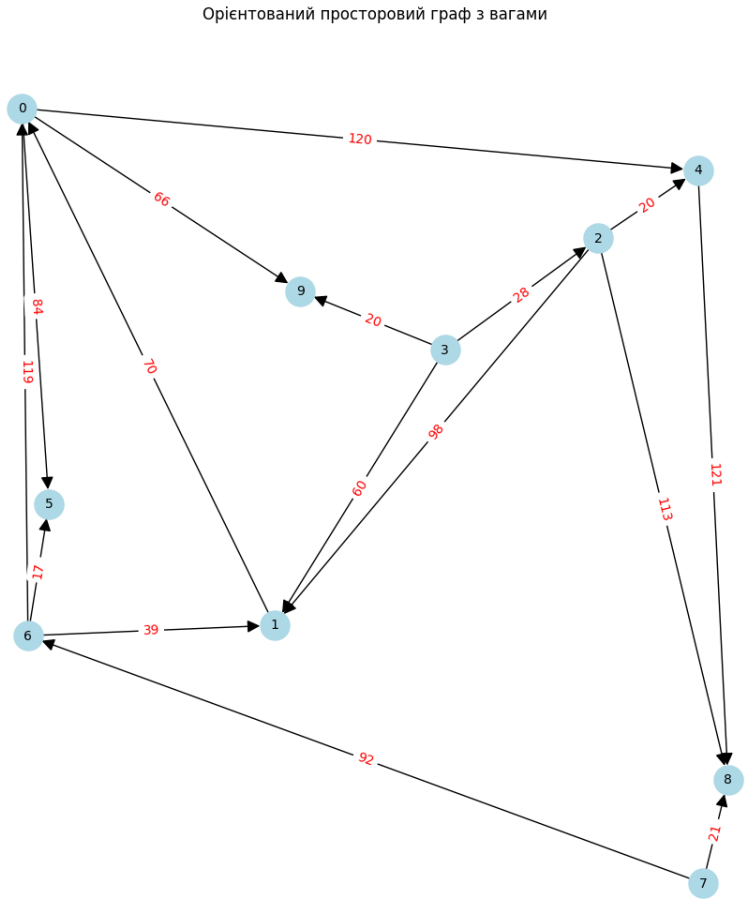


Рисунок 3.1 – Приклад орієнтованого графу

Було додано функції для зміни способу представлення графа з списку суміжності на матрицю і навпаки, функції для створення графів – об’єктів бібліотеки NetworkX на їх основі. Ці функції необхідні для полегшення оперування

набором тестових даних та для можливості порівняння отриманих результатів зі справжніми, обрахованими за допомогою NetworkX.

Необхідно створити такі варіації алгоритмів, щоб порівняти їх ефективність на різних структурах для представлення графів:

1. Алгоритм A* на списку та матриці суміжності з використанням Евклідової відстані;
2. Алгоритм Беллмана-Форд на списку та матриці суміжності;
3. Алгоритм Флойда-Уоршала на матриці суміжності.

В алгоритмах має бути додано підрахунок відвіданих вершин, замір витраченого часу та використаної пам’яті. Не має сенсу реалізовувати Флойда-Уоршала на списку суміжності, оскільки першим кроком буде створення матриці суміжності. Замір часу та пам’яті починається після кроку ініціалізації необхідних списків чи матриць. В реалізаціях алгоритмів необхідно забезпечити можливість відтворення шляху.

Для збору результатів, отриманих на тестових даних очікується наявність таких додаткових функцій:

- Створення заданої кількості графів з заданими параметрами;
- Збереження згенерованих графів до файлів та зчитування;
- Запуск обраного алгоритму на наборі графів зі збереженням результатів до таблиці.

Висновок до розділу 3

Під час роботи над цим розділом було обрано засоби для розробки демонстраційного застосунку: мову, бібліотеки та середовище розробки. Також було описано очікуваний функціонал та ключові моменти роботи над програмою. Було дозволяє створювати випадкові графи з вибором способу представлення та

параметрів, запускати на них алгоритми A*, Беллмана-Форда та Флойда-Уоршала та повертає оцінку коректності, витраченого часу, простору та кількість перевірених у процесі виконання вершин. Також наявні функції для масового тестування: створення наборів зі вказаною кількістю тестових графів з заданими властивостями та збереження інформації про виконання алгоритмів до таблиць.

РОЗДІЛ 4. ПОРІВНЯННЯ АЛГОРИТМІВ

4.1 Критерії оцінки ефективності

Ефективність алгоритмів в цій роботі буде оцінено за такими показниками:

- Застосовність до наданого виду графа – визначаються обмеження роботи алгоритму
- Час виконання – без витрат на підготовку та побудову знайденого шляху.
- Споживання пам’яті
- Кількість перевірених вузлів

Неможливо підібрати такий алгоритм, що матиме найкращі значення за всіма параметрами, тому важливо охопити всі аспекти та обирати ті, які є пріоритетними в поставлених завданнях.

4.2 Підготовка тестових даних

Основою коректної оцінки ефективності алгоритмів є забезпечення достатньо великого набору з тестових даних, які покриватимуть якомога більшу частину випадків. Щоб детальніше проаналізувати алгоритми, їх роботу необхідно перевірити на різних типах графів. Для порівняння ефективності графи буде розділено на групи за такими критеріями: орієнтованість, наявність негативних ваг, щільність, присутність циклів, зв’язність (для орієнтованих) та розмір. Приклади таких груп: “малі, орієнтовані, розріджені, без циклів”, “великі, неорієнтовані, щільні, з циклами”. Також розмежованим буде тестування графів, представлених різними способами. . Незв’язні графи в даній роботі не розглядаються. Набори тестових графів, використаних в межах даної роботи, буде створено автоматично.

Оскільки алгоритм A* дуже відрізняється від інших і на його ефективність впливає якість евристики, то для нього набір тестових графів буде обмежено

такими, що забезпечують умови для хорошої продуктивності. В реальних задачах алгоритм A* найчастіше використовується на системах, наближених до транспортних, тому для його тестування буде згенеровано неорієнтовані просторові графи, з присвоєнням вершинам координат та створенням ребер, які не перетинаються, з вагою наближеною до евклідової відстані між вершинами, при цьому ребра додаються між вершинами, де є імовірність появи в реальних умовах більша. За інших умов використання алгоритму A* не матиме сенсу, оскільки евристика погано оцінюватиме реальну відстань. Оскільки ребра не перетинаються, низька, середня, висока щільність будуть братись як середні значення отриманих на графах заданого розміру.

Інші варіації графів для порівняння алгоритмів Беллмана-Форда та Флойда-Уоршала буде створено без координат, з випадковим розподілом ребер та їх ваг.

4.3 Заміри з виконання алгоритмів на тестових даних

Для кожної тестової множини було створено від тридцяти до п’ятдесяти графів.

Просторові графи (ребра додаються за пріоритетом, перетин заборонено). Графи з цієї категорії як правило мають нижчу щільність, їх буде подано як списки суміжності. Основні алгоритми пошуку – A*, якому передано координати та алгоритм Беллмана-Форда, для алгоритму Флойда-Уоршала створено відповідні матриці суміжності (без зарахування у час виконання алгоритму) :

1. Малі неорієнтовані графи без циклів (на 10 вершин):

   На основі отриманих таблиць даних виведено такі середні значення:

Таблиця 4.1 – Результати на малих орієнтованих графах без циклів

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (с)	4,40E-05	23,1E-05	109,6E-05

Споживання пам’яті (Б)	888,16	688,96	492,36
Час на відновлення шляху (с)	0,41E-05	0,33E-05	0,44E-05
Пам’ять для відновлення шляху (Б)	80,32	80,32	98,08

Алгоритмам Беллмана-Форда та Флойда-Уоршала доводиться обходити усі вершини, тим часом як на подібних графах з логічною структурою, A* знаходить шлях з меншою кількістю додаткових кроків. За отриманими результатами алгоритм відвідує в середньому 12,4 відсотків з непотрібних вершин.

2. Середні неорієнтовані графи без циклів (на 50 вершин)

Таблиця 4.2 – Результати на середніх неорієнтованих графах без циклів

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (с)	93,82E-5	449,53E-5	12976,44E-5
Споживання пам’яті (КБ)	3,69	3,25	14,27
Час на відновлення шляху (с)	1,01E-05	0,62E-05	1,71E-05
Пам’ять для відновлення шляху (КБ)	0,182	0,182	0,184

Для даного набору графів A* відвідує в середньому 21,8 відсотків з непотрібних вершин.

3. Великі неорієнтовані графи без циклів (на 100 вершин)

Таблиця 4.2 – Результати на великих неорієнтованих графах без циклів

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (мс)	0,91	21,95	565,49

Споживання пам’яті (КБ)	7,24	12,49	55,91
Час на відновлення шляху (с)	0,85E-05	1,34E-05	1,81E-05
Пам’ять для відновлення шляху (КБ)	0,254	0,254	0,259

Зайвих відвіданих A* вершин – в середньому 25,7 відсотків.

4. Дуже великі неорієнтовані графи без циклів (на 500 вершин)

Таблиця 4.3 – Результати на дуже великих неор. гр. без циклів

Алгоритм	A*	Беллмана-Форда
Час виконання (мс)	1,538	807,910
Споживання пам’яті (КБ)	38,633	82,769
Час на відновлення шляху (с)	1,228E-05	2,393E-05
Спожита для відновлення шляху пам’ять (КБ)	0,622	0,622

Для даного набору A* відвідує в середньому 30,7 відсотків непотрібних вершин.

5. Малі неорієнтовані графи з циклами (на 10 вершин)
    a. З майже мінімальною щільністю (25 відсотків)

Таблиця 4.4 – Результати на розріджених малих графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (с)	7,21E-05	35,89E-05	100,3E-05
Споживання пам’яті (КБ)	0,734	0,674	0,275

b. З відносно високою щільністю (максимально можливою без перетину на графі)

Таблиця 4.5 – Результати на щільних малих графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (с)	6,358E-05	133,2E-05	217,3E-05
Споживання пам’яті (КБ)	1,17	0,801	0,387

6. Середні неорієнтовані графи з циклами (на 50 вершин)
   a. З майже мінімальною щільністю (10 відсотків)

Таблиця 4.6 – Результати на розріджених середніх графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (с)	24E-05	0,024	0,079
Споживання пам’яті (КБ)	3,812	4,558	8,905

   b. З середньою щільністю (максимально можливою без перетинів на графі)

Таблиця 4.7 – Результати на щільних середніх графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (мс)	1,236	7,09	34,157
Споживання пам’яті (КБ)	5,930	3,086	3,562

7. Великі неорієнтовані графи з циклами (на 100 вершин)
   a. З майже мінімальною щільністю (5 відсотків)

Таблиця 4.8 – Результати на розріджених великих графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (мс)	0,273	36,64	104, 505
Споживання пам’яті (КБ)	6,555	12,693	36,325

b. З середньою щільністю (максимально можливою без перетину на графі)

Таблиця 4.9 – Результати на щільних великих графах

Алгоритм	A*	Беллмана-Форда	Флойда-Уоршала
Час виконання (мс)	5,928	123,6	160,817
Споживання пам’яті (КБ)	18,342	18,216	12,742

Звичайні графи (ребра додаються випадково, перетин дозволено). Це набір тестових графів для порівняння ефективності алгоритмів Беллмана-Форда (на списку суміжності та на матриці суміжності) та Флойда-Уоршала:

Неорієнтовані:

8. Малі графи з циклами (на 10 вершин)
   a. З мінімальною щільністю (25 відсотків)

Таблиця 4.10 – Результати на розріджених непросторових малих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,145	0,264	12,698
Споживання пам’яті (КБ)	0,873	0,946	3,761

b. З високою щільністю (100 відсотків)

Таблиця 4.11 – Результати на щільних непросторових малих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,335	0,39	3,04
Споживання пам’яті (КБ)	0,842	0,710	2,504

9. Середні графи з циклами (на 50 вершин)
    a. З мінімальною щільністю (5 відсотків)

Таблиця 4.12 – Результати на розріджених непросторових середніх графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	3,552	9,815	274,35
Споживання пам’яті (КБ)	2,972	4,132	74,192

    b. З високою щільністю (100 відсотків)

Таблиця 4.13 – Результати на щільних непросторових середніх графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	7,365	11,045	357,713
Споживання пам’яті (КБ)	3,101	3,612	67,692

10. Великі графи з циклами (на 100 вершин)
    a. З мінімальною щільністю (5 відсотків)

Таблиця 4.14 – Результати на розріджених непросторових великих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (с)	0,004	0,017	2,376
Споживання пам’яті (КБ)	11,239	11,752	266,659

    b. З високою щільністю (100 відсотків)

Таблиця 4.15 – Результати на щільних непросторових великих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (с)	0,019	0,018	2,783
Споживання пам’яті (КБ)	12,510	11,964	227,907

11. Дуже великі графи з циклами (на 500 вершин)
    a. З низькою щільністю (5 відсотків)

Таблиця 4.16 – Результати на розріджених непросторових дуже великих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)
Час виконання (с)	0,032	1,743
Споживання пам’яті (КБ)	46,597	93,956

    b. З високою щільністю (100 відсотків)

Таблиця 4.17 – Результати на щільних непросторових дуже великих графах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)
Час виконання (с)	0,051	1,057
Споживання пам’яті (КБ)	47,417	82,372

Орієнтовані:

12. Малі сильно-зв’язні (на 10 вершин)
    a. З низькою щільністю (20 відсотків)

Таблиця 4.18 – Результати на розріджених сильно-зв'язних малих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,572	11,026	2,592
Споживання пам'яті (КБ)	1,109	2,468	2,232

b. З високою щільністю (90 відсотків)

Таблиця 4.19 – Результати на щільних сильно-зв'язних малих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,229	3,95	5,248
Споживання пам'яті (КБ)	0,842	1,381	2,626

13. Середні сильно-зв'язні (на 50 вершин)

a. З низькою щільністю (5 відсотків)

Таблиця 4.20 – Результати на розріджених сильно-зв'язних середніх орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	2,339	55,431	265,764
Споживання пам'яті (КБ)	2,646	9,827	74,669

b. З високою щільністю (90 відсотків)

Таблиця 4.21 – Результати на щільних сильно-зв'язних середніх орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	1,779	55,443	350,245
Споживання пам'яті (КБ)	2,632	12,268	68,987

14. Великі сильно-зв'язні (на 100 вершин)
    a. З низькою щільністю (3 відсотків)

Таблиця 4.22 – Результати на розріджених сильно-зв'язних великих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (с)	0,004	0,267	1,999
Споживання пам'яті (КБ)	10,396	34,035	258,588

    b. З високою щільністю (90 відсотків)

Таблиця 4.23 – Результати на щільних сильно-зв'язних великих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (с)	0,003	0,255	2,792
Споживання пам'яті (КБ)	10,449	34,186	257,561

15. Малі (на 10 вершин) слабко-зв'язні середньої щільності (50 відсотків)

Таблиця 4.24 – Результати на слабко-зв'язних малих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,29	1,113	1,594
Споживання пам'яті (КБ)	0,843	1,052	1,525

16. Середні (на 50 вершин) слабко-зв'язні середньої щільності (50 відсотків)

Таблиця 4.25 – Результати на слабко-зв'язних середніх орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (мс)	0,529	28,547	188,483
Споживання пам'яті (КБ)	2,640	5,856	56,137

17. Великі (на 100 вершин) слабко-зв'язні середньої щільності (50 відсотків)

Таблиця 4.26 – Результати на слабко-зв'язних великих орграфах

Алгоритм	Беллмана-Форда (список)	Беллмана-Форда (матриця)	Флойда-Уоршала
Час виконання (с)	0,001	0,114	1,395
Споживання пам'яті (КБ)	10,125	18,415	234,053

4.4 Порівняння ефективності алгоритмів

На основі зібраних з джерел та отриманих шляхом практичних перевірок даних, помітно що:

- На графах з подібною до транспортних систем будовою, евристика з використанням Евклідової відстані забезпечує колосальний відрив у часі на розріджених графах. При цьому перебираються лише найбільш імовірні до потрапляння в кінцевий шлях вершини. Спостерігається позитивна кореляція між розміром графа та відсотком перебраних вершин, найкращими є результати для графів з низькою щільністю. Також алгоритм A* використовує менше пам’яті для обрахунків, ніж інші;
- Зі збільшенням щільності графа заданого розміру, A* демонструє менший відрив від інших алгоритмів відносно часу;
- Значною є різниця у використанні ресурсів часу та пам’яті алгоритмами Беллмана-Форда та Флойда-Уоршала, що пояснюється різницею природніх для них задач. При цьому обидва алгоритми завжди перевіряють всі ребра (вершини);
- Для алгоритму Флойда-Уоршала час виконання та витрати пам’яті зростають разом зі збільшенням кількості вузлів на графах незалежно від щільності, при цьому вплив зміни щільності за сталого розміру є незначним, що пояснюється основою роботи алгоритма: ітерація по парах вершин з матриці суміжності;
- Алгоритм Беллмана-Форда на матриці суміжності завжди демонструє значно гірші результати з огляду на витрати часу та пам’яті, ніж реалізація на списку суміжності;
- На показники ефективності алгоритмів не впливає орієнтованість графа та наявність циклів (додатніх);

- Алгоритм Флойда-Уоршала не є доцільним для використання на дуже великих графах, від ста вузлів його ефективність різко знижується (зростають витрати часу та пам’яті).

Беручи до уваги всю зібрану інформацію можна зробити такі висновки щодо ефективності алгоритмів:

Умови застосування:

Таблиця 4.27 – Особливості застосування алгоритмів

	A*	Беллмана-Форда	Флойда-Уоршала
Робота з ребрами з від’ємними вагами	Ні	Так	Так
Робота з негативними циклами	Ні	Повертається помилка	Повертається помилка
Можливість побудувати якісну евристику	Необхідна для ефективного пошуку шляху	Не потребує	Не потребує
Простота реалізації	Важчий: евристика, черга з пріоритетами, облік відстаней, попередників	Простий: на основі циклів	Простий: робота з матрицями

В таблиці нижче назви алгоритмів вказано скорочено: БФ – Беллмана-Форда, ФУ – Флойда-Уоршала. Якщо вказується алгоритм A*, то передбачається, що існує якісна евристика, і граф є наближеним до транспортних систем, інакше – наступний вказаний алгоритм. БФ здійснює менше обчислень за умови оптимізації з пропуском вершин.

Таблиця 4.28 – Порівняння результатів алгоритмів

Тип графа	Найшвидший	Використовує найменше пам’яті	Відвідує найменше вершин
Малий граф, низька щільність	A*, БФ	ФУ, БФ	A*, БФ
Малий граф, середня щільність	A*, БФ	ФУ, БФ	A*, БФ
Середній граф, низька щільність	A*, БФ	A*, БФ	A*, БФ
Середній граф, середня щільність	A*, БФ	A*, БФ	A*, БФ
Великий граф, низька щільність	A*, БФ	A*, БФ	A*, БФ
Великий граф, середня щільність	A*, БФ	A*, БФ	A*, БФ

Наведена таблиця вказує найефективніші алгоритми для задач з умовою пошуку шляху між двома вершинами на графах з щільністю, наближеною до транспортних систем. Зважаючи на зменшення відношення часу виконання Беллмана-Форда до A* з наближенням щільності до максимальної щільності, для середніх та великих повних і майже повних графів він буде швидшим за A*.

Для задачі на пошук шляхів з одного вузла в інші стандартним вибором є алгоритм Беллмана-Форда. Однак беручи до уваги на значну різницю часу між A* та Беллмана-Форда, на середніх розріджених графах A*, запущений до кожної з кінцевих вершин, може бути ефективнішим.

В задачах на пошук шляхів між всіма парами вершин для графів з високою щільністю доцільно використовувати алгоритм Флойда-Уоршала. Але для великих розріджених графів, залежно від кількості вершин, може бути ефективнішим запустити Беллмана-Форда з усіх вузлів, особливо, якщо його оптимізувати.

4.5 Додаткові уточнення

Пріоритетним критерієм для оцінки було визначено час виконання алгоритму, тому з метою прискорення доступу до даних алгоритмами, що розглядають графи як суміжні списки, графи реалізовано як словники. Через це затрати пам’яті є вищими, з результатів не виключено ресурси на зберігання графа та обчислення, оскільки критерії оцінки тісно пов’язані. Однак на графах більшої розмірності та з меншою щільністю можна спостерігати очікуваний відрив у використанні пам’яті алгоритмом Флойда-Уоршала від інших.

За результатами наведеними в таблицях можна побачити, що час та витрати пам’яті на побудову шляху за отриманими списками або таблицями результатів для всіх алгоритмів є приблизно однаковими, тому ці значення при порівнянні можна не враховувати.

Висновок до розділу 4

У цьому розділі було відібрано критерії, як входять до оцінки ефективності агоритмів. Також було детально описано набори тестових графів, що використовувались та наведено таблиці з середніми результатами замірів для відповідних алгоритмів. На основі зібраних теоретичних даних та результатів тестувань було проведено порівняння ефективності алгоритмів за зазначеними параметрами: доцільність для виконання вказаних задач, час виконання, використання пам’яті, кількість обчислень (ітерацій/оновлень), простота реалізації.

ВИСНОВКИ

Під час роботи над цією курсовою було досліджено та порівняно алгоритми пошуку найкоротшого шляху в зважених графах, а саме алгоритми A*, Беллмана-Форда та Флойда-Уоршала.

У теоретичній частині наведено основні визначення з теорії графів, розглянуто найпоширеніші види графів та способи їх представлення. Також було коротко оглянуто основні алгоритми пошуку шляху в зважених графах та їх покращені версії. Далі було оглянуто стратегії виконання алгоритмів та детально досліджено, як працюють алгоритми A*, Беллмана-Форда та Флойда-Уоршала, обмеження та особливості їх роботи.

Практичною частиною даної курсової є створення застосунку з реалізацією вищезгаданих алгоритмів, функціями для генерації випадкових графів з заданими властивостями та виведення результату з додатковими параметрами, які містять інформацію, корисну для порівняння ефективності. Було додано інтерфейс з можливістю обрати тип графа, вершини та алгоритм для перевірки, переглянути результат виконання.

Для створення таблиць зі статистикою було додано функції для побудови заданої кількості графів певного типу, функції запису та читання з файлів для графів та результатів. Після цього для всіх створених таблиць було виведено середні результати та на їх основі порівняно алгоритми за визначеними критеріями ефективності. Було визначено, для яких графів, які алгоритми є найшвидшими, витрачають найменше пам’яті, здійснюють найменшу кількість обчислень, для яких задач підходять. В результаті порівняння було підтверджено інформацію про ефективність алгоритмів, зібрану з використаних джерел та розглянуто окремі випадки задач, де вибір алгоритма не є очевидним.

Зважаючи на різноманіття проблем з реального життя, які теоретично можна представити як задачі на графах, та способів до їх розв’язання, немає межі для покращення алгоритмів. Уже зараз існує багато нових алгоритмів з цікавими підходами до вирішення завдань, але шлях до їх створення пролягав через найпростіші, починаючи з алгоритму Дейкстри, пошуку у ширину, пошуку у глибину та розглянутих у цій роботі. Проаналізувавши існуючі алгоритми, поглянувши на них під новим кутом та спробувавши покращити, можна створити щось краще, щось нове та унікальне.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Applications of Dijkstra’s shortest path algorithm [Електронний ресурс] : стаття / GeekForGeeks. — 2022. — Режим доступу: https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm
2. Walks, Trails, Paths, Cycles and Circuits in Graph [Електронний ресурс] : стаття / GeekForGeeks. — 2025. — Режим доступу: https://www.geeksforgeeks.org/walks-trails-paths-cycles-and-circuits-in-graph/
3. Kara Kadyrov, E. İ. Graphs 101: Understanding Graph Data Structures [Електронний ресурс] : блог / Elif İrem Kara Kadyrov. — Medium. — 2023. — Режим доступу: https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af
4. Manoharan, I. Understanding Graph Theory: A Guide to Types of Graphs [Електронний ресурс] : блог / Ilakkuvaselvi Manoharan. — Medium. — 2024. — Режим доступу: https://medium.com/@ilakk2023/330-understanding-graph-theory-a-guide-to-types-of-graphs-0f853477b818
5. Graph Types and Applications [Електронний ресурс] : стаття / GeeksForGeeks. — 2024. — Режим доступу: https://www.geeksforgeeks.org/graph-types-and-applications/
6. @MakeComputerScienceGreatAgain. Demystifying Graph Implementation in Programming [Електронний ресурс] : блог / @MakeComputerScienceGreatAgain. — Medium. — 2023. — Режим доступу: https://medium.com/@MakeComputerScienceGreatAgain/demystifying-graph-implementation-in-programming-a-comprehensive-guide-b856712b2b1c
7. @HeCodesIT. Graph Theory Problems [Електронний ресурс] : блог / @HeCodesIT. — Medium. — 2022. — Режим доступу: https://medium.com/@HeCodesIT/graph-theory-problems-89e6a3c064f9

8. Jose, K. Common Graph Theory Problems [Електронний ресурс] : блог / Kelvin Jose. — Medium. — 2020. — Режим доступу: https://medium.com/data-science/common-graph-theory-problems-ca990c6865f1
9. Mujtaba, M. Common Short Path Algorithms 101 [Електронний ресурс] : блог / Muhammad Mujtaba. — Medium. — 2023. — Режим доступу: https://medium.com/@Muhammadmujtaba_5355/common-short-path-algorithms-101-cfeea73f478e
10. @c0D3M. Shortest Path Algorithms in Graph [Електронний ресурс] : блог / @c0D3M. — Medium. — 2023. — Режим доступу: https://medium.com/@c0D3M/shortest-path-algorithms-in-graph-acd07ead6e91
11. Comparison of Pathfinding Algorithms [Електронний ресурс] : конференційна публікація / ICCS. — 2018. — Режим доступу: https://www.iccs-meeting.org/archive/iccs2018/papers/108620083.pdf
12. Biswas, D. Understanding Jump Point Search (JPS) Algorithm [Електронний ресурс] : блог / Dibyendu Biswas. — Medium. — 2021. — Режим доступу: https://dibyendu-biswas.medium.com/understanding-jump-point-search-jps-algorithm-554c99aab178
13. IEEE Computer Society SBC of IIT. Greedy Algorithms: Strategies and Examples [Електронний ресурс] : блог / IEEE Computer Society SBC of IIT. — Medium. — 2025. — Режим доступу: https://medium.com/@ieeecomputersocietyiit/greedy-algorithms-strategies-and-examples-12e197c8bf28
14. What is Dynamic Programming? Working Algorithms and Examples [Електронний ресурс] : блог / Medium. — 2024. — Режим доступу: https://medium.com/@toyinbosino/what-is-dynamic-programming-working-algorithms-and-examples-93d11b12e609
15. @toyinbosino. What is Dynamic Programming? Working Algorithms and Examples [Електронний ресурс] : блог / @toyinbosino. — Medium. — 2024. — Режим

доступу: https://medium.com/@toyinbosino/what-is-dynamic-programming-working-algorithms-and-examples-93d11b12e609

16. Heuristic Evaluation [Електронний ресурс] : стаття / GeekForGeeks. — 2021. — Режим доступу: https://www.geeksforgeeks.org/walks-trails-paths-cycles-and-circuits-in-graph/
17. Bhuyan, A. Bellman-Ford Algorithm – A Pathfinding Algorithm for Weighted Graphs [Електронний ресурс] : блог / Aditya Bhuyan. — Medium. — 2023. — Режим доступу: https://aditya-sunjava.medium.com/bellman-ford-algorithm-a-pathfinding-algorithm-for-weighted-graphs-647080392326
18. Floyd-Warshall Algorithm [Електронний ресурс] : блог / DevGenius. — Режим доступу: https://blog.devgenius.io/floyd-warshall-algorithm-f004a01ae40e
19. @sridharan_t. Floyd-Warshall Algorithm [Електронний ресурс] : блог / @sridharan_t. — DevGenius. — 2020. — Режим доступу: https://blog.devgenius.io/floyd-warshall-algorithm-f004a01ae40e
20. A* Search Algorithm [Електронний ресурс] : стаття / GeeksForGeeks. — 2024. — Режим доступу: https://www.geeksforgeeks.org/a-search-algorithm/