Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделі та варіантів реалізації задачі оптимізації побудови маршрутів з використанням Google Maps API
(тема)

Виконав:

здобувач 2 року навчання,

групи *ІТПм-24-2*

Нагорний І.А.
(прізвище, ініціали)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва спеціальності)

Тип програми *освітньо-професійна*

Освітня програма *Інформаційні технології проектування*
(повна назва освітньої програми)

Керівник *доц. Петрова Р.В.*
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ *проф. Гребеннік І.В*
(підпис) (прізвище, ініціали)

2025 р.

Я, як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

 «*19» грудня 2025 р.* Нагорний І.А.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено «23» грудня 2025 р.

Керівник кваліфікаційної роботи  доц. Петрова Р.В.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
Кафедра *Системотехніки*
Рівень вищої освіти *другий (магістерський)*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
Освітня програма *Інформаційні технології проектування*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри СТ
проф. Гребеннік І.В
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Нагорному Івану Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження моделі та варіантів реалізації задачі оптимізації побудови маршрутів з використанням Google Maps API
затверджена наказом університету від 24 листопада 2025 р. № 1058Ст
2. Термін подання студентом роботи до екзаменаційної комісії 24 грудня 2025 р.
3. Вихідні дані до роботи *дані про дорожній граф, зокрема вершини, ребра та їх вагові характеристики, а також географічні координати, отримані з використанням Google Maps API. На основі цих даних реалізуються та досліджуються алгоритми пошуку шляху (Дейкстри, двонапрямлений Дейкстри та A*) із фіксацією ітераційних і часових метрик. Розробка та експериментальне дослідження виконуються на платформі .NET з використанням мови C# у середовищі Microsoft Windows 10/11 на IBM-сумісному персональному комп'ютері.*
4. Перелік питань, що потрібно опрацювати в роботі *4.1 Аналіз предметної області, 4.1.1 Загальна характеристика предметної області, 4.1.2 Актуальність дослідження моделі та варіантів реалізації задачі оптимізації побудови маршрутів, 4.1.3 Характеристика обраних алгоритмів, 4.1.4 Постановка задачі, 4.2 Реалізація алгоритмів пошуку шляху, 4.2.1 Реалізація алгоритму Дейкстри, 4.2.2 Реалізація алгоритму двонапрямленого Дейкстри, 4.2.3 Реалізація алгоритму AStar, 4.2.4 Основні відмінності в роботі алгоритмів, 4.3 Порівняльний аналіз алгоритмів пошуку шляху, 4.3.1 Умови та методика проведення експерименту, 4.3.2 Порівняння часових характеристик алгоритмів, 4.3.3 Аналіз кількості відвіданих вершин та опрацьованих ребер, 4.3.4 Порівняння динаміки пошуку за ітераціями, 4.3.5 Інтерпретація результатів, 4.4 Висновки*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій: *5.1 Блок схема алгоритм Дейкстри, 5.2 Блок схема двонапрямлений алгоритм Дейкстри, 5.3 Блок схема алгоритм A*, 5.4 Результати оцінювання малий експеримент, 5.5 Результати оцінювання великий експеримент, 5.6 Структура ітераційних CSV-файлів, 5.7-5.8 Ітераційна динаміка пошуку за метрикою CurrentWeight, 5.9-5.10 Ітераційна динаміка пошуку*

за метрикою BestWeight, 5.11-5.12 Ітераційна динаміка пошуку за метрикою DifferenceFromOptimal

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Аналіз підходів пошуку шляху	*доц. Петрова Р.В.*		*26.11.2025*
Розробка функціональних вимог до системи	*доц.Петрова Р.В.*		*29.11.2025*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	*Отримання завдання кваліфікаційної роботи*	*24.11*	*Виконано*
2.	*Аналіз предметної області та постановка задачі*	*25.11.25-29.11.25*	*Виконано*
3.	*Ознайомлення з літературою*	*30.11.25-03.12.25*	*Виконано*
4.	*Розробка алгоритмів пошуку шляху в графі з невід'ємними вагами ребер*	*03.12.25-4.12.25*	*Виконано*
5.	*Отримання метрик оцінювання та ітераціних даних аналізу алгоритмів пошуку шляху*	*04.12.25-10.12.25*	*Виконано*
6.	*Порівняльний аналіз алгоритмів пошуку шляху*	*10.12.25-12.12.25*	*Виконано*
7.	*Оформлення пояснювальної записки та програмної документації*	*12.12.25-16.12.25*	*Виконано*
8.	*Оформлення графічної частини та презентаційних матеріалів комп'ютерного захисту*	*16.12.25-21.12.25*	*Виконано*
9.	*Представлення на рецензування*	*21.12.25*	*Виконано*
10.	*Представлення кваліфікаційної роботи до ЕК*	*24.12.25*	*Виконано*

Дата видачі завдання 24 листопада 2025 р.

Здобувач _____ Нагорний І.А
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Петрова Р.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра містить: 125 сторінок, 14 рисунків, 19 лістингів, 2 додатки, 20 джерел інформації.

АЛГОРИТМ A*, АЛГОРИТМ ДЕЙКСТРИ, ДВОНАПРЯМЛЕНИЙ АЛГОРИТМ ДЕЙКСТРИ, ГРАФ, ДИНАМІКА ПОШУКУ, ІТЕРАЦІЙНІ МЕТРИКИ, ОПТИМІЗАЦІЯ МАРШРУТІВ, ПОШУК НАЙКОРОТШОГО ШЛЯХУ, ПОРІВНЯЛЬНИЙ АНАЛІЗ, GOOGLE MAPS API, A STAR, BIDIRECTIONAL DIJKSTRA, DIJKSTRA, GRAPH, PATHFINDING.

Об’єкт дослідження – процес побудови та оптимізації маршрутів у дорожніх мережах на основі графових моделей із використанням геоінформаційних сервісів.

Предмет дослідження – моделі та алгоритми пошуку найкоротшого шляху в графах, зокрема алгоритм Дейкстри, двонапрямлений алгоритм Дейкстри та алгоритм A*, а також особливості їх реалізації й поведінки при використанні даних Google Maps API.

Мета роботи – дослідити модель задачі оптимізації побудови маршрутів, реалізувати та проаналізувати різні варіанти алгоритмів пошуку шляху, а також виконати їх порівняльний аналіз за часовими, структурними та ітераційними характеристиками для визначення доцільності використання кожного алгоритму залежно від умов задачі.

Наукова новизна – реалізовано три алгоритми пошуку шляху засобами платформи .NET, розроблено механізм збору ітераційних та часових метрик, сформовано експериментальну базу даних у вигляді CSV-файлів, побудовано графіки динаміки роботи алгоритмів та виконано інтерпретацію отриманих результатів, що дозволило сформувати практичні рекомендації щодо вибору алгоритму для маршрутної оптимізації в задачах різного масштабу.

ABSTRACT

The explanatory note to the master's thesis contains: 125 pages, 14 figures, 19 listings, 2 appendices, 20 sources of information.

A STAR, BIDIRECTIONAL DIJKSTRA, DIJKSTRA, DYNAMIC SEARCH BEHAVIOR, GRAPH, GOOGLE MAPS API, ITERATIVE METRICS, PATHFINDING, ROUTE OPTIMIZATION, SHORTEST PATH SEARCH, COMPARATIVE ANALYSIS

Object of research – the process of route construction and optimization in road networks based on graph models using geoinformation services.

Subject of research – models and shortest path search algorithms in graphs, in particular Dijkstra's algorithm, Bidirectional Dijkstra, and the A* algorithm, as well as the specifics of their implementation and behavior when using Google Maps API data.

Purpose of the work – to investigate the model of the route optimization problem, implement and analyze different variants of pathfinding algorithms, and perform their comparative analysis using temporal, structural, and iterative characteristics in order to determine the feasibility of using each algorithm depending on problem conditions.

Scientific novelty – three pathfinding algorithms were implemented using the .NET platform, a mechanism for collecting iterative and temporal metrics was developed, an experimental dataset in the form of CSV files was formed, graphs illustrating the dynamics of algorithm execution were constructed, and the obtained results were interpreted, which made it possible to formulate practical recommendations for selecting an appropriate algorithm for route optimization problems of different scales.

ЗМІСТ

ПЕРЕЛІК УМОВИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API – програмний інтерфейс для взаємодії між програмними компонентами

A* (A STAR) – евристичний алгоритм пошуку найкоротшого шляху в графі

BestWeight – найкраща відома на поточній ітерації вага маршруту

CSV (Comma-Separated Values) – формат зберігання табличних даних у текстовому вигляді

CurrentWeight – поточна вага маршруту або вузла на ітерації алгоритму

DifferenceFromOptimal – різниця між поточним найкращим значенням та оптимальною вагою маршруту

Graph (граф) – математична структура, що складається з вершин і ребер

Google Maps API – сервіс доступу до картографічних і дорожніх даних Google

Iteration (ітерація) – один крок виконання алгоритму

Node (вузол, вершина) – елемент графа, що представляє точку маршруту

Pathfinding – процес пошуку оптимального шляху в графі

Route Optimization – оптимізація побудови маршруту

TimeInSeconds – вага ребра, що відображає час проходження у секундах

Vertex (вершина) – вузол графа, який представляє точку з’єднання маршрутів

Weight (вага) – числове значення, що характеризує вартість переходу між вершинами

ВСТУП

Розвиток сучасних інформаційних систем тісно пов’язаний із задачами обробки просторових даних та оптимізації маршрутів у складних мережах. Побудова оптимальних шляхів є базовою складовою навігаційних сервісів, логістичних платформ, транспортних інформаційних систем і геоінформаційних застосунків, де швидкість отримання результату та його точність безпосередньо впливають на ефективність роботи всієї системи.

Зростання обсягів просторових даних та ускладнення дорожніх мереж призводять до підвищених вимог до алгоритмів пошуку шляху. Традиційні підходи, які добре працюють для графів невеликого розміру, можуть виявлятися недостатньо ефективними при роботі з великими мережами, що містять тисячі вершин і ребер. У таких умовах актуальним є дослідження та порівняння різних алгоритмів оптимізації маршрутів з метою визначення найбільш доцільних підходів для конкретних сценаріїв використання.

Одним із ключових інструментів для роботи з географічними даними є Google Maps API, який надає доступ до детальної інформації про дорожні мережі, географічні координати та часові характеристики переміщення. Використання такого API створює можливості для побудови та дослідження маршрутів у реальних умовах, що наближені до практичних задач навігації та транспорту.

У контексті оптимізації маршрутів важливу роль відіграють алгоритми пошуку найкоротшого шляху в графах. Серед них особливе місце займає алгоритм Дейкстри, який забезпечує гарантоване знаходження оптимального маршруту для графів з невід’ємними вагами ребер. Разом із тим, його обчислювальна складність може обмежувати ефективність застосування для великих графів, що зумовлює необхідність дослідження покращених та альтернативних підходів.

До таких підходів належать двонапрямлений алгоритм Дейкстри та алгоритм A*, які спрямовані на скорочення області пошуку та зменшення

обчислювальних витрат. Двонапрямлений підхід передбачає одночасний пошук із початкової та кінцевої вершин, тоді як алгоритм A* використовує евристичну функцію для спрямування пошуку в бік цільової вершини. Практична ефективність цих методів значною мірою залежить від структури графа та умов виконання задачі.

Актуальність даної роботи зумовлена необхідністю не лише теоретичного аналізу алгоритмів пошуку шляху, а й їх практичної перевірки в умовах, наближених до реального використання. Порівняння алгоритмів за різними метриками, зокрема часовими характеристиками та ітераційною динамікою, дозволяє глибше зрозуміти їх поведінку та обґрунтувати вибір конкретного алгоритму для практичних застосувань.

Отримані результати будуть мати практичну цінність для розробників навігаційних і логістичних систем, а також можуть бути використані як основа для подальших досліджень у галузі оптимізації маршрутів та обробки просторових даних.

Також під час роботи над кваліфікаційною роботою було опубліковано тези на тему «Research on the Model and Implementation Options for Optimizing Route Construction Using the Google Maps API» на Міжнародній науково-технічній конференції студентів, аспірантів та молодих вчених «COMPUTER SCIENCE, INFORMATION TECHNOLOGIES AND MANAGEMENT SYSTEMS YOUNG SCIENTISTS CONFERENCE CSYSC-2025» у м.Івано - Франківськ 17-19 грудня 2025рік.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика предметної області

Алгоритми пошуку шляху становлять фундаментальний інструмент сучасних інформаційних систем, що працюють із просторовими даними, мережевими структурами або транспортними мережами. Вони забезпечують можливість визначення оптимального маршруту між двома або більшою кількістю точок у графі, де вершини представляють об’єкти або локації, а ребра – можливі шляхи переходу між ними з певною вагою, що відображає відстань, час, вартість чи інший критерій. Задача пошуку найкоротшого шляху є однією з найважливіших у теорії графів та прикладній інформатиці, оскільки відповідає реальним процесам оптимізації, характерним для транспортної логістики, навігаційних систем, робототехніки, комп’ютерних мереж і геоінформаційних сервісів.

У транспортних та картографічних системах алгоритми пошуку шляху виконують роль ядра логічної обробки даних, забезпечуючи можливість формування маршрутів, адаптивне реагування на зміну дорожніх умов і визначення найефективніших напрямків руху. Вони дозволяють інтерпретувати геопросторові дані як зважений граф і виконувати обчислення, спрямовані на мінімізацію певної цільової функції. Такі алгоритми застосовуються не лише для фіксованих графів, а й у динамічних середовищах, де ваги ребер можуть змінюватися залежно від трафіку, дорожніх обмежень або інших факторів, характерних для реальних сценаріїв.

У ширшому контексті алгоритми пошуку шляху є невід’ємною частиною систем підтримки прийняття рішень. Вони допомагають аналізувати альтернативи, порівнювати можливі маршрути та формувати рекомендації на основі об’єктивних даних. Саме завдяки таким алгоритмам навігаційні сервіси,

зокрема ті, що працюють на основі Google Maps API, здатні не лише прокладати шлях до потрібної точки, але й ураховувати зміни дорожнього руху, пропонувати альтернативні варіанти та оптимізувати пересування.

Таким чином, алгоритми пошуку шляху є ключовим елементом інформаційних систем, які працюють з просторовою інформацією. Вони забезпечують можливість ефективного аналізу маршрутів та підвищують якість прийнятих рішень у різних сферах, від міського транспорту до навігації автономних систем.

Розроблюване дослідження ґрунтується на вже створеній інформаційній системі, у якій було реалізовано базовий функціонал побудови маршрутів та впроваджено алгоритм пошуку найкоротшого шляху. У попередній роботі було сформовано структуру транспортного графа, налагоджено взаємодію з Google Maps API та створено механізм визначення оптимального маршруту за допомогою алгоритму Дейкстри. Цей алгоритм забезпечив коректність розрахунків і став фундаментальним компонентом, на якому базується подальше дослідження.

Разом із тим, початкова реалізація має певні обмеження щодо продуктивності та масштабованості, що проявляються при роботі з більшими наборами даних або складнішими маршрутними мережами. Алгоритм Дейкстри, хоча й гарантує знаходження оптимального шляху, не використовує додаткової інформації про простір, не має евристичної складової та характеризується високими витратами часу на опрацювання великих графів. У реальних транспортних системах, де значення ваг ребер можуть змінюватися динамічно, такі обмеження стають суттєвими.

У новому дослідженні планується розширити функціональні можливості існуючої системи шляхом впровадження двох альтернативних алгоритмів – A* та двонапрямленого Дейкстри. Це дозволить провести повноцінне порівняння різних підходів до пошуку найкоротшого шляху, оцінити їхню ефективність у практичних умовах та визначити алгоритмічну стратегію, що найкраще відповідає особливостям транспортних мереж, сформованих на основі даних

Google Maps API. Таким чином, робота продовжує розвиток раніше створеної системи та спрямована на підвищення точності та швидкодії маршрутних обчислень.

У вже існуючій системі, створеній у попередньому дослідженні, ключову роль у побудові маршрутів відіграє алгоритм Дейкстри, який було реалізовано як основний механізм пошуку найкоротшого шляху. Саме на ньому базується логіка формування маршруту між заданими точками, що є фундаментом для подальшої оптимізації та розширення алгоритмічної складової системи. Враховуючи, що система працює з графом, отриманим на основі просторових даних Google Maps, алгоритм Дейкстри був обраний як надійний та універсальний метод, здатний забезпечити коректний результат у широкому спектрі випадків.

Алгоритм Дейкстри ґрунтується на послідовному розширенні множини вершин із відомою мінімальною відстанню від початкової точки. Він передбачає поступове знаходження найкоротших шляхів до всіх вершин графа, використовуючи ваги ребер, які відображають відстань або час руху між суміжними точками. На кожному кроці алгоритм обирає вершину з найменшим поточним значенням відстані, позначає її як опрацьовану і виконує релаксацію всіх суміжних ребер. Такий підхід гарантує, що після завершення роботи будуть визначені оптимальні шляхи від джерела до кожної досяжної вершини графа.

Реалізація цього алгоритму в межах попередньої інформаційної системи враховувала специфіку картографічних даних та особливості побудови транспортної мережі. Для представлення маршруту використовувалася структура графа, де вершинами слугували ключові точки – зупинки, транспортні вузли або координати, а ребра описували можливі напрямки руху між ними. Ваги ребер отримувалися з API Google Maps, що дозволяло прив’язати розрахунок маршруту до реальних відстаней та умов дорожнього руху. Таким чином, алгоритм Дейкстри забезпечував коректність та стабільність побудови маршруту в умовах статичного або слабкодинамічного графа.

У практичній реалізації особливу увагу приділяли організації черги з

пріоритетами, яка дозволяла ефективно визначати наступну вершину для опрацювання. Завдяки цьому вдавалося зменшити час виконання алгоритму на графах середнього розміру, що характерні для задач міського транспортного моделювання. Також було реалізовано механізм відновлення повного маршруту шляхом збереження попередників для кожної вершини, що дозволяло після завершення алгоритму обчислити послідовність точок, які утворюють оптимальний шлях.

Загалом використання алгоритму Дейкстри в попередній системі забезпечило надійний фундамент для задачі пошуку шляху, однак його обчислювальна складність та необхідність опрацювання значної кількості вершин роблять цей підхід менш ефективним у випадках великих або динамічних графів. Це створює передумови для подальшого вдосконалення алгоритмічної частини, що і є ключовою метою успішного розвитку системи в межах магістерського дослідження.

У поточному дослідженні планується реалізувати ще два алгоритми пошуку шляху, що дозволить суттєво розширити можливості існуючої системи та підвищити точність і швидкодію обчислень. Додавання нових алгоритмів обумовлене необхідністю забезпечити порівняння різних підходів до оптимізації маршрутів, оскільки використання лише одного методу не дає змоги об’єктивно оцінити ефективність побудови шляхів у різних типах транспортних мереж. Крім того, у реальних сценаріях дані дорожнього руху можуть бути динамічними, а структура графа – великою та нерівномірною, що вимагає застосування більш гнучких та продуктивних методів.

Першим додатковим алгоритмом є A*, який розширює класичний підхід Дейкстри шляхом використання евристичної функції. Це дозволяє спрямовувати пошук у потрібному напрямку, зменшуючи кількість опрацьованих вершин і, відповідно, скорочуючи час виконання. У задачах транспортної навігації A* є одним із найефективніших алгоритмів, оскільки евристикою може виступати прямолінійна відстань до цільової точки або інші геометричні оцінки, що добре узгоджуються з географічною природою даних Google Maps.

Другим додатковим алгоритмом є двонапрямлений Дейкстри, який оптимізує процес пошуку шляхів за рахунок одночасного виконання двох пошуків – від початкової та кінцевої вершини назустріч один одному. Це дозволяє суттєво скоротити область пошуку, особливо в густих графах, таких як міські транспортні мережі. Алгоритм забезпечує ту саму точність, що й класичний Дейкстри, але потребує значно менше обчислювальних ресурсів.

Обидва алгоритми були обрані завдяки їхній доведеній ефективності у задачах навігації та роботі з великими графами. A* забезпечує прискорення за рахунок евристики, а двонапрямлений Дейкстри – за рахунок скорочення області пошуку. Разом із класичним алгоритмом Дейкстри вони формують три різні, але взаємодоповнюючі стратегії пошуку шляху, що дозволяє отримати ґрунтовні експериментальні дані та визначити найбільш продуктивний підхід для використання в інформаційних системах маршрутизації.

1.2 Актуальніть дослідження моделі та варіантів реалізації задачі оптимізації побудови маршрутів

Сучасний етап розвитку міст характеризується інтенсивним зростанням транспортних потоків, ускладненням вулично-дорожньої мережі та зростанням вимог до якості переміщення пасажирів і вантажів. За таких умов задача оптимізації побудови маршрутів набуває особливої значущості, оскільки від ефективності маршрутних рішень напряму залежать час у дорозі, витрати пального, рівень завантаженості інфраструктури та загальна комфортність пересування для користувачів транспортних послуг. Забезпечення оперативного й обґрунтованого вибору маршруту перетворюється на критично важливу функцію сучасних інформаційних систем підтримки прийняття рішень у транспортній сфері.

Розвиток картографічних сервісів, зокрема Google Maps, створює

принципово нові можливості для побудови маршрутів з урахуванням реальних умов дорожнього руху, дорожніх обмежень, аварійних ситуацій та поточного трафіку. Однак, попри високий рівень готових інструментів, проблема залишається відкритою з точки зору вибору та порівняння алгоритмів пошуку шляху, які найкраще використовують ці дані у конкретній предметній області. Необхідно не лише користуватися зовнішнім сервісом, а й вміти формувати, оптимізувати та інтерпретувати графову модель дорожньої мережі, обираючи алгоритми, здатні забезпечити прийнятну швидкодію при збереженні точності результатів.

Інформаційні системи маршрутизації, що працюють у режимі, наближеному до реального часу, стикаються з проблемою обмежених обчислювальних ресурсів за наявності великої кількості користувачів та складних транспортних мереж. Зростання розмірності графа, що моделює дорожню інфраструктуру, призводить до суттєвого збільшення часу виконання алгоритмів пошуку шляху. У таких умовах використання лише базових підходів, на кшталт класичного алгоритму Дейкстри, часто є недостатнім: хоча він гарантує оптимальність результату, його продуктивності може не вистачати для оперативного відгуку системи при великій кількості запитів.

Додатковим аспектом актуальності є перехід від статичних до динамічних моделей дорожньої мережі. Ваги ребер графа перестають бути фіксованими: вони залежать від часу доби, дорожніх подій, ремонтів, погодних умов та інших факторів. Це означає, що обчислений одного разу маршрут може швидко втратити актуальність, а алгоритми мають бути здатними до багаторазового, швидкого й повторюваного перерахунку. Саме тому порівняння різних варіантів реалізації алгоритмів пошуку шляху на базі однієї й тієї ж моделі дорожньої мережі набуває практичного значення.

Не менш важливим є економічний та екологічний вимір задачі оптимізації маршрутів. Скорочення часу перебування транспортних засобів у дорозі та зменшення довжини маршрутів безпосередньо впливають на зниження витрат пального, зменшення експлуатаційних витрат та скорочення викидів шкідливих

речовин в атмосферу. Для муніципальних транспортних систем, служб доставки, таксі та логістичних компаній навіть незначне покращення середнього часу побудови та якості маршруту може давати відчутний кумулятивний ефект. Це підсилює потребу у формальному дослідженні моделей та алгоритмів, що лежать в основі таких систем.

З теоретичної точки зору задача пошуку найкоротшого шляху є базовою складовою ширшого класу задач оптимізації на графах, включно із задачами маршрутизації транспортних засобів, багатокритеріальними задачами та варіантами з обмеженнями. Вибір конкретного алгоритму пошуку шляху (Dijkstra, A*, двонапрямлений Dijkstra тощо) визначає не лише швидкодію системи, а й можливість подальшого розширення моделі, наприклад, у напрямку багатокритеріальної оптимізації або врахування пріоритетів певних ділянок дороги. Тому дослідження їх властивостей у контексті інтеграції з Google Maps API має методологічну цінність.

Особливу актуальність має проведення порівняльного аналізу алгоритмів у рамках вже існуючої інформаційної системи, яка була попередньо реалізована для формування розкладу маршрутних автобусів із використанням Google Maps[1]. Такий підхід дозволяє переходити від абстрактних оцінок обчислювальної складності до практичних вимірів продуктивності на реальних або наближених до реальних даних. Результати порівняння дають змогу сформувати рекомендації щодо використання певного алгоритму для конкретних сценаріїв – густих міських мереж, міжміських маршрутів, частих перерахунків шляху тощо.

Актуальність роботи також зумовлена загальними тенденціями цифрової трансформації транспортних систем. Інтелектуальні транспортні системи, «розумні» міста, сервіси спільних поїздок та автоматизоване планування перевезень потребують алгоритмічних рішень, здатних працювати у взаємодії з зовнішніми API, такими як Google Maps, та забезпечувати інтеграцію просторових, часових і статистичних даних. Без чітко досліджених моделей і перевірених алгоритмів складно гарантувати стабільність та надійність таких

сервісів.

З практичної точки зору дослідження варіантів реалізації алгоритмів Дейкстри, A* та двонапрямленого Дейкстри дає змогу сформувати інженерні рекомендації для розробників прикладних систем навігації та логістики. Вибір алгоритму часто здійснюється емпірично або за інтуїтивними міркуваннями, без детального аналізу специфіки даних та навантажень. Формалізоване порівняння на базі єдиної програмної платформи дозволяє обґрунтувати такий вибір, знизити ризики неефективних рішень та забезпечити прозору аргументацію у процесі проєктування.

Таким чином, дослідження моделі та варіантів реалізації задачі оптимізації побудови маршрутів із використанням Google Maps API є актуальним як у прикладному, так і в теоретичному аспектах. Воно поєднує аналіз класичних алгоритмів пошуку шляху з їх адаптацією до сучасних картографічних сервісів, дає змогу підвищити ефективність вже існуючих систем та створює підґрунтя для подальшого розвитку інтелектуальних транспортних рішень, зорієнтованих на реальні потреби користувачів та обмеження інфраструктури.

1.3 Характеристика обраних алгоритмів

Алгоритм Дейкстри є одним із найвідоміших і найважливіших алгоритмів теорії графів, призначених для визначення найкоротшого шляху від однієї вершини до всіх інших вершин у графі з невід’ємними вагами ребер. Його значення у сфері маршрутизації є фундаментальним, оскільки він забезпечує гарантоване знаходження оптимального шляху за умови, що всі ваги ребер додатні. Саме через цю властивість алгоритм широко застосовується у транспортних мережах, картографічних сервісах, мережевих протоколах та навігаційних системах, де необхідно забезпечувати точні та передбачувані результати.

Основна ідея алгоритму полягає в поетапному розширенні множини вершин, для яких уже знайдено найкоротший шлях від початкової точки. Алгоритм починає роботу з присвоєння початковій вершині нульової відстані, а всім іншим – нескінченно великої величини. Далі він поступово «покращує» оцінку відстаней шляхом релаксації ребер, доки не буде опрацьовано весь необхідний фрагмент графа. Завдяки детермінованій природі роботи та строгому порядку вибору вершин, алгоритм Дейкстри гарантує, що знайдений шлях є оптимальним.

У процесі роботи алгоритм підтримує дві множини вершин: опрацьовані вершини, для яких найкоротша відстань вже остаточно відома, та неопрацьовані, які все ще можуть бути оновлені. На кожному кроці алгоритм обирає вершину з найменшим значенням поточної відстані серед неопрацьованих. Це забезпечує те, що наступною опрацьовується саме та вершина, шлях до якої на даному етапі гарантовано є найкоротшим. Такий механізм вибору робить алгоритм ефективним та стійким при працю з транспортними графами, де ребра відображають реальні відстані або час поїздки.

Ключовим етапом роботи алгоритму є операція релаксації. Вона полягає у перевірці, чи може шлях від початкової вершини до певної сусідньої вершини бути покращений, якщо пройти через поточну вершину. Якщо нове значення відстані є меншим за попереднє, воно оновлюється. Саме через багаторазове виконання релаксацій для різних ребер алгоритм поступово наближається до оптимального розв’язку. Важливим аспектом є те, що після завершення релаксації для певної вершини та її включення до множини опрацьованих значення її найкоротшої відстані стає остаточним і більше не змінюється.

Для ефективності роботи алгоритму часто використовується черга з пріоритетами, яка дозволяє швидко знаходити вершину з мінімальною поточною відстанню. Така структура даних значно прискорює виконання алгоритму на великих графах, оскільки зменшує час пошуку вершини з найменшим пріоритетом. У практичних реалізаціях також використовується масив або словник для зберігання поточних відстаней до вершин, що дає змогу швидко

оновлювати їх у процесі роботи.

Алгоритм Дейкстри оптимально працює у випадках, коли структура графа є статичною, а ваги ребер не змінюються протягом виконання. Це робить його ідеальним для задач, пов’язаних із побудовою маршрутів у транспортних мережах, де відстані між об’єктами залишаються фіксованими. Проте він має певні обмеження у випадках, коли дані змінюються динамічно або коли необхідно повторно виконувати пошук для численних різних початкових і кінцевих точок.

У контексті інтеграції з Google Maps API алгоритм Дейкстри був адаптований до роботи з графом, утвореним на основі географічних даних. Його ребра відображали можливі дороги або сегменти шляху, а ваги – відповідні відстані або час руху між точками. Така модель дозволяла алгоритму працювати у відповідності з реальною дорожньою інфраструктурою, забезпечуючи максимально наближені до дійсності результати.

У реалізованій системі особлива увага приділялася збереженню інформації про попередників кожної вершини. Це давало змогу після завершення роботи алгоритму відновити повний маршрут – послідовність координат або точок, які формують оптимальний шлях. Цей механізм є важливим, оскільки кінцевою метою пошуку найкоротшого шляху є не лише визначення мінімальної довжини, а й побудова повного маршруту для подальшого використання системою.

Важливою властивістю алгоритму Дейкстри є передбачуваність результатів. На відміну від евристичних методів, він не використовує попередніх припущень щодо розташування цільової точки, а повністю покладається на фактичні ваги ребер. Це робить його особливо корисним у випадках, коли потрібна максимальна точність і недопустимо отримати приблизний результат.

Алгоритм демонструє хорошу продуктивність на графах середнього розміру, однак його обчислювальна складність зростає зі збільшенням кількості вершин і ребер. Це обмежує можливості застосування алгоритму для дуже великих транспортних мереж або сценаріїв, де необхідно швидко обробляти численні запити. Власне, через ці обмеження у поточній роботі планується

порівняти його з іншими, більш ефективними алгоритмами.

Перевагою алгоритму Дейкстри є його універсальність та надійність. Він не потребує додаткової інформації про геометрію простору чи напрямок до цільової точки, що дозволяє застосовувати його у широкому спектрі задач. Проте саме відсутність цієї інформації робить алгоритм менш ефективним у задачах просторової навігації, де такі дані можуть значно прискорити пошук.

Таким чином, алгоритм Дейкстри є базовим та обов'язковим елементом у системі пошуку маршрутів, на основі якого проводиться подальше вдосконалення. Його детермінована структура, гарантована оптимальність результату та простота логіки роблять його ідеальним орієнтиром для порівняння продуктивності інших алгоритмів, таких як A* та двонапрямлений Дейкстри, що будуть впроваджені у рамках подальших етапів дослідження.

Алгоритм Дейкстри належить до класичних методів пошуку найкоротшого шляху на зважених графах із невід'ємними вагами ребер. Його основна ідея полягає у поступовому визначенні найкоротших відстаней від вихідної вершини до всіх інших вершин графа шляхом поетапного покращення їхніх оцінок. Для формального опису алгоритм розглядає граф як пару множин

$$G = (V, E)$$

де V– множина вершин, а E– множина ребер, кожному з яких зіставлена вага w

$$w: E \to \mathbb{R}_{\geq 0}.$$

Першим кроком виконання алгоритму є визначення початкових умов. Вихідна вершина s отримує відстань $d(s) = 0,$ тоді як усім іншим вершинам графа призначається початкове значення $d(v) = +\infty, v \neq s.$

Таке позначення дозволяє трактувати невідомі на початку шляхи як недосяжні та поступово уточнювати їх у процесі виконання алгоритму.

Паралельно з відстанями формується масив попередників $p(v)$, який дозволить згодом відновити знайдений оптимальний маршрут.

Ключовою складовою алгоритму є операція релаксації, яка дозволяє зменшити оцінку відстані до певної вершини, якщо було знайдено коротший шлях. Для кожного ребра (u, v)виконується умова: якщо $d(v) > d(u) + w(u, v)$, тоді значення відстані оновлюється за правилом $d(v) := d(u) + w(u, v)$, а попередником вершини vоголошується вершина u:

$$p(v) := u.$$

Релаксація є фундаментом роботи алгоритму, оскільки саме вона забезпечує покращення оцінок та поступове наближення до оптимальних значень.

Ще однією важливою частиною алгоритму є вибір вершини, яка матиме мінімальну поточну відстань серед тих, що ще не були опрацьовані. Це забезпечує правильний порядок проходження графа, оскільки гарантовано, що обрана вершина вже має найкоротший шлях серед усіх можливих. Формально вибір здійснюється за критерієм:

$$u = \arg\min_{v \notin S} d(v)$$

де S– множина вершини, для яких значення відстаней уже визначені як остаточні. Після додавання вершини до цієї множини алгоритм більше не повертається до неї, що забезпечує детермінованість процесу.

Принцип роботи алгоритму можна охарактеризувати як «радіальне» поширення шуканих відстаней від вихідної точки. На початку хвиля пошуку охоплює лише сусідні вершини, а з кожною наступною ітерацією її область розширюється. Завдяки тому, що на кожному кроці обирається вершина з найменшою поточною відстанню, алгоритм виключає потребу переглядати шляхи, які вже завідомо не можуть бути оптимальними. Це робить обчислення

стабільними й передбачуваними, а сам алгоритм – ефективним для графів зі статичними вагами.

Особливо важливо надати не лише математичну формалізацію, а й наочне уявлення про логіку роботи алгоритму. Для цього представлено блок-схему алгоритму Дейкстри, у якій послідовність дій демонструється у вигляді структурованого графічного опису. Блок-схема, наведена на рисунку 1.1, відображає ключові кроки алгоритму: ініціалізацію даних, вибір вершини з мінімальною відстанню, перевірку сусідів, виконання релаксації та завершення роботи після обробки всіх вершин. Така схема дозволяє легко простежити логіку процесу навіть без детального занурення у формули, а також забезпечує наочність необхідну для документації.

Завдяки поєднанню формального опису, математичних виразів та графічного подання алгоритм Дейкстри може бути представлений максимально доступно та наочно.

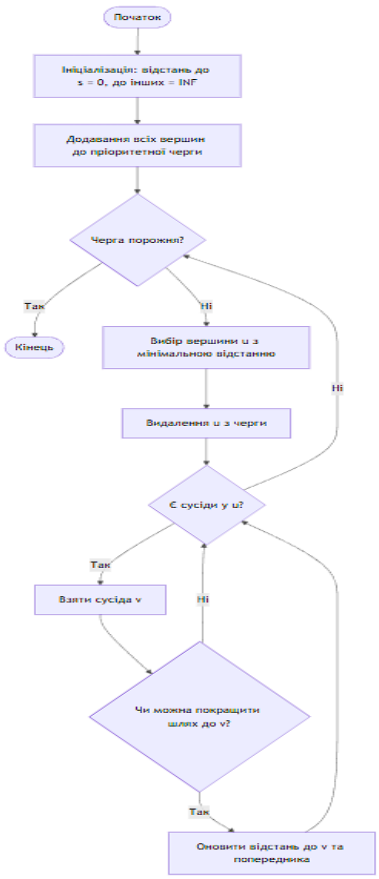

Рисунок 1.1 – Блок-схема алгоритму Дейкстри

Алгоритм двонапрямленого Дейкстри є розвитком класичного підходу та спрямований на суттєве прискорення пошуку найкоротшого шляху за рахунок одночасного виконання двох незалежних пошуків – від початкової вершини та від цільової. Основна ідея полягає в тому, що зустріч двох хвиль пошуку відбувається значно швидше, ніж завершення повного обходу графа в одному напрямку. Завдяки цьому обсяг опрацьованих вершин зменшується в рази, що безпосередньо впливає на зниження обчислювальної складності в реальних сценаріях.

Сутність методу полягає в паралельному виконанні двох варіацій алгоритму Дейкстри. Один із них працює у прямому напрямку – від початкової точки до кінцевої, інший – у зворотному напрямку. Кожен із цих пошуків підтримує власну чергу з пріоритетами, власні структури даних для відстаней та власні множини відвіданих вершин. Після кожної ітерації відбувається перевірка на наявність перетину – моменту, коли вершина, опрацьована у прямому пошуку, зустрічається з вершиною, опрацьованою у зворотному. Саме цей момент є ключовим сигналом, що оптимальний шлях уже може бути знайдений.

Однією з важливих особливостей алгоритму є те, що він оперує тим самим принципом релаксації ребер, що й класичний Дейкстри, але застосовує його одночасно у двох напрямах. У процесі роботи кожна частина пошуку виконує оновлення відстаней до сусідніх вершин, а отримані значення зберігаються окремо. Це дозволяє структурам пошуку незалежно просуватися графом до моменту зустрічі. Після цього виконується синтез результатів – поєднання двох частин маршруту у повний шлях.

Завдяки симетричності роботи алгоритму загальна кількість опрацьованих вершин істотно зменшується. Якщо класичний Дейкстри фактично розширює радіус пошуку у вигляді «хвилі», то двонапрямлений підхід розширює дві менші хвилі, кожна з яких охоплює лише частину графа. Геометрично це означає, що замість пошуку всієї сфери радіуса r алгоритм опрацьовує дві сфери радіуса приблизно r/2, що зменшує загальний обсяг роботи приблизно вчетверо. Це

суттєва перевага у великих транспортних мережах, таких як міські дороги, міжміські траси чи складні полігональні графи.

Практична реалізація двонапрямленого Дейкстри потребує точного синхронізованого контролю над обома пошуками. Потрібно не лише відстежувати зустріч двох «хвиль», а й коректно визначати точку перетину, що забезпечує найкоротший можливий шлях. У різних практичних варіаціях точкою перетину може стати вершина, яка виявляється першою спільною у множинах відвіданих вершин, або вершина, яка забезпечує мінімальну сумарну відстань між двома напрямами пошуку. В обох випадках важливим є правильне комбінування отриманих часткових маршрутів.

У контексті транспортних систем алгоритм двонапрямленого Дейкстри має значну перевагу над класичним варіантом, оскільки реальні дорожні мережі часто є масштабними та містять тисячі вузлів. Пошук у таких графах може бути дорогим для обчислення, і саме в подібних випадках можливість зменшити простір пошуку набуває критичного значення. Завдяки поділу пошуку на два напрями алгоритм здатний значно скоротити час реакції системи, що є важливим при прокладанні маршрутів у режимі, наближеному до реального часу.

Одним із викликів при застосуванні цього алгоритму є необхідність роботи з графами, у яких існує чіткий зворотний напрямок для кожного ребра. У транспортних мережах це не завжди гарантується, оскільки дороги можуть бути односторонніми або мати додаткові обмеження. У таких випадках зворотний пошук потребує побудови додаткової структури – транспонованого графа, який є оберненим щодо напрямів ребер. Це ускладнює реалізацію, але забезпечує коректність роботи алгоритму.

Структура даних, що використовується для реалізації алгоритму, також подібна до класичної: черги з пріоритетами, таблиці найкоротших відстаней, масиви попередників. Проте таких структур у двонапрямленому варіанті завжди дві – одна для прямого пошуку, інша для зворотного. Це збільшує вимоги до пам’яті, але водночас істотно зменшує час роботи, що в більшості практичних застосувань є пріоритетним.

Однією з переваг двонапрямленого Дейкстри є його детермінованість і точність. На відміну від алгоритмів, які використовують евристики, таких як A*, його результати не залежать від якості чи коректності зовнішньої оцінки. Це робить алгоритм цінним інструментом для задач, у яких необхідно отримати гарантовано оптимальний шлях без наближених рішень. У той же час швидкодія залишається вищою, ніж у класичного варіанту, що забезпечує хорошу збалансованість алгоритму між точністю та продуктивністю.

Алгоритм добре підходить для порівняльного аналізу, оскільки дозволяє дослідити, наскільки пошук у двох напрямах перевершує пошук в одному. У межах інформаційної системи для побудови маршрутів двонапрямлений Дейкстри стане важливим елементом експериментальної частини дослідження та дозволить оцінити, чи є цей підхід оптимальнішим у конкретній предметній області.

Важливою особливістю цього алгоритму є чітка логічна структура, що робить його придатним для розширення та модифікацій. Він може бути адаптований до різних типів графів, зокрема й до динамічних або ієрархічних дорожніх мереж. Його концептуальна простота поєднується з високою ефективністю, що робить алгоритм універсальним інструментом у задачах просторової навігації.

Таким чином, двонапрямлений алгоритм Дейкстри поєднує в собі точність класичного методу та суттєве прискорення роботи за рахунок зменшення простору пошуку. Він є природним кандидатом для розширення можливостей системи маршрутизації, оскільки дозволяє забезпечити швидке й гарантовано коректне визначення маршруту навіть у великих та складних транспортних мережах. Його використання у поточному дослідженні дасть змогу всебічно порівняти переваги та недоліки різних підходів до пошуку шляху та визначити оптимальну стратегію для систем, що працюють із географічними даними.

Алгоритм двонапрямленого Дейкстри є модифікацією класичного алгоритму, в якій пошук найкоротшого шляху між початковою вершиною sта цільовою вершиною tвиконується одночасно у двох напрямах: від sдо tта від

t до s. Ідея полягає в тому, що дві «хвилі» пошуку зустрічаються в деякій проміжній вершині, і завдяки цьому загальна кількість опрацьованих вершин є істотно меншою, ніж у випадку одностороннього пошуку. Такий підхід особливо корисний у великих транспортних графах, де повний обхід у одному напрямку є надто дорогим за часом.

Формально розглядається граф

$$G = (V, E), w: E \to \mathbb{R}_{\geq 0},$$

де V – множина вершин, E – множина ребер, $w(u, v)$ – вага ребра між вершинами uта v.

Для реалізації зворотного пошуку використовується або той самий граф (якщо всі ребра двосторонні), або транспонований граф $G^R = (V, E^R)$, де для кожного ребра $(u, v) \in E$існує ребро $(v, u) \in E^R$з тією самою вагою.

У двонапрямленому алгоритмі підтримуються дві системи відстаней: прямі відстані від початкової вершини sта зворотні відстані від цільової вершини t. Початкові умови задаються таким чином:

$$d_f(s) = 0, d_f(v) = \text{INF}, \forall v \in V, v \neq s,$$
$$d_b(t) = 0, d_b(v) = \text{INF}, \forall v \in V, v \neq t,$$

де $d_f(v)$ – поточна оцінка найкоротшої відстані від s до вершини v у прямому пошуку, $d_b(v)$ – відповідна оцінка відстані від t до вершини v у зворотному пошуку.

Робота алгоритму базується на тій самій операції релаксації, що і в класичному алгоритмі Дейкстри, але вона виконується незалежно у двох напрямках. Для прямого пошуку від sдо tрелаксація для ребра (u, v)має вигляд:

якщо $d_f(v) > d_f(u) + w(u, v)$,то

$$d_f(v) \coloneqq d_f(u) + w(u, v),$$
$$p_f(v) \coloneqq u,$$

де $p_f(v)$– попередник вершини vу маршруті від s.

Для зворотного пошуку від tдо s(на транспонованому графі) релаксація формулюється аналогічно:

якщо $d_b(u) > d_b(v) + w(u, v)$,то
$$d_b(u) \coloneqq d_b(v) + w(u, v),$$
$$p_b(u) \coloneqq v,$$

де $p_b(u)$– попередник вершини uу маршруті від t.

На кожному кроці алгоритм обирає по одній вершині з мінімальною поточною відстанню у кожному з напрямів. Для прямого пошуку це вершина $u_f = \arg\min_{v \notin S_f} d_f(v)$, для зворотного – вершина $u_b = \arg\min_{v \notin S_b} d_b(v)$, де S_f і S_b – множини вершин, для яких відстані вже зафіксовані як остаточні у відповідних напрямах. Вершини u_fта u_bвилучаються з пріоритетних черг та переходять до цих множин, після чого для їхніх сусідів виконується релаксація.

Особливістю двонапрямленого алгоритму є необхідність відстежувати «зустріч» двох пошуків. Для цього на кожній ітерації після оновлення відстаней перевіряється, чи існують вершини, які вже опрацьовані в обох множинах S_fі S_b. Якщо така вершина mзнайдена, то формується кандидат на найкоротший шлях з довжиною $L_m = d_f(m) + d_b(m)$.

Алгоритм підтримує поточне значення найкращої знайдено довжини

$$L = \min_{m \in S_f \cap S_b} (d_f(m) + d_b(m)).$$

Відновлення повного маршруту виконується шляхом об’єднання шляху

від sдо m, побудованого за попередниками p_f, та шляху від mдо t, побудованого за попередниками p_b.

Критерій зупинки у двонапрямленому алгоритмі пов'язаний із порівнянням поточного найкращого знайденого шляху Lта значень мінімальних відстаней у чергах. Інтуїтивно алгоритм може завершити роботу тоді, коли у пріоритетних чергах не залишилося вершин з потенційно кращими оцінками, ніж уже знайдена L. У спрощеному варіанті, прийнятному для практичного опису, пошук триває доти, доки обидві черги не спорожніють або доки не буде досягнуто вершини перетину й зафіксовано маршрут, який влаштовує задані умови.

Доцільно доповнити наведені формули блок-схемою, яка наочно демонструє логіку паралельного пошуку в обох напрямах. Така блок-схема подана як Рисунок 1.2 і відображатиме послідовність основних кроків: ініціалізацію структур даних, вибір вершин з мінімальними відстанями у прямому та зворотному напрямах, виконання релаксації, перевірку наявності спільних вершин та завершення роботи після знаходження найкоротшого шляху.

Алгоритм A* є одним із найефективніших алгоритмів пошуку шляху у зважених графах і широко використовується в задачах просторової навігації, робототехніці, комп'ютерній графіці та маршрутизації. Його ефективність полягає у поєднанні двох ключових складових – точності алгоритму Дейкстри та спрямованості пошуку завдяки використанню евристичної функції. Саме це дозволяє A* значно скоротити кількість опрацьованих вершин, фокусуючи пошук у напрямку кінцевої мети.

Основна ідея алгоритму полягає в мінімізації оцінки повної вартості маршруту, яка складається з двох частин: фактичної вартості шляху від початку до поточної вершини та евристичної оцінки відстані до кінцевої вершини. Евристика виступає «підказкою» для алгоритму, спрямовуючи пошук по найбільш перспективному напрямку. Типовими евристиками є евклідова або мангеттенська відстані, які добре працюють у транспортних та географічних графах.

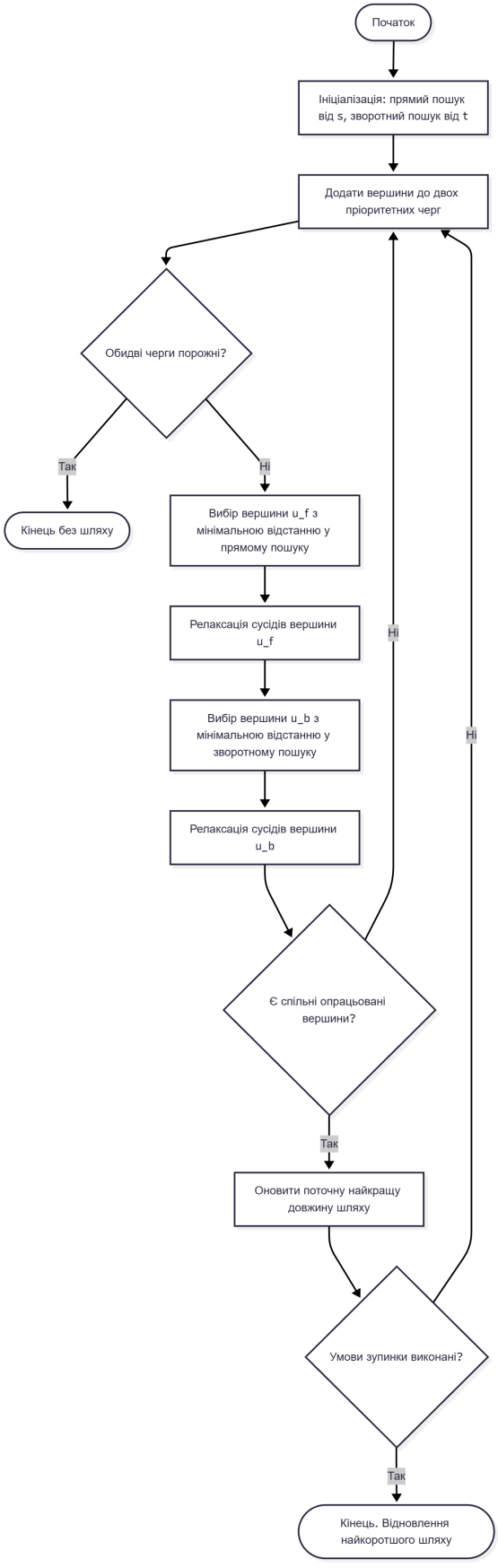

Рисунок 1.2 – Блок-схема алгоритму двонаправленого Дейкстри

У роботі алгоритм підтримує дві основні таблиці оцінок: значення фактичної вартості шляху (g-значення) та значення повної оцінки маршруту (f-значення), де f = g + h. Кожна вершина має власну оцінку, яка визначає її пріоритет у черзі. Алгоритм використовує чергу з пріоритетами для вибору вершини з найменшим f-значенням, що дозволяє фокусувати пошук там, де очікується найшвидше наближення до цілі.

Операція релаксації у A* подібна до релаксації в алгоритмі Дейкстри, але виконується з урахуванням евристики. Якщо новий шлях зменшує фактичну відстань до вершини або покращує повну оцінку маршруту, значення оновлюються. Евристика впливає на те, які вершини будуть опрацьовані раніше, фактично скорочуючи область пошуку. З точки зору простору, якщо Дейкстри рівномірно розширює область навколо початкової вершини, то A* спрямовує пошук у сторону кінцевої, утворюючи витягнуту хвилю, що охоплює меншу кількість точок.

Ключовою властивістю алгоритму є його оптимальність за умови, що евристика є допустимою та монотонною. Допустима евристика ніколи не перевищує справжню відстань до цілі, що гарантує, що A* знайде оптимальний шлях. Монотонність (або узгодженість) забезпечує, що оцінка маршруту не зменшується при переході по графу, що спрощує реалізацію та підвищує стабільність.

У транспортних графах алгоритм A* демонструє особливо високу ефективність, оскільки геометричні відстані між точками відповідають реальному розташуванню місцевості, а дорожня мережа має достатню регулярність. Отже, евклідова або геодезична відстань є якісною евристикою, що дозволяє отримувати оптимальні маршрути з мінімальними обчислювальними витратами. Це робить A* одним із основних методів у картах, навігаторах та мобільних додатках.

Практична реалізація A* вимагає підтримки двох множин вершин – «відкритої» та «закритої». Відкрита множина містить вершини, які ще потрібно опрацювати, тоді як закрита – вершини, для яких значення шляху вже остаточно

визначене. Перехід вершини зі стану відкритої до закритої відбувається після того, як вона обрана чергою та її сусіди пройшли через релаксацію. Такий механізм забезпечує чіткість і передбачуваність обчислень.

Завдяки своїй природі A* добре масштабується на графи великого розміру, на відміну від алгоритму Дейкстри, який змушений опрацьовувати значно більше вершин. Продуктивність A* особливо проявляється у випадках, коли між початковою та кінцевою точками існує єдиний або добре визначений оптимальний напрямок руху. Евристика «звужує» область пошуку, пропускаючи ті частини графа, що очевидно не ведуть до швидкого знаходження оптимального маршруту.

У контексті інтеграції з Google Maps API алгоритм A* може використовувати інформацію про координати та відстані між точками як частину евристичної функції, що дає суттєві переваги. Використання географічних координат дозволяє формувати якісні оцінки наближення до цільової точки, а динамічні дані про трафік можуть адаптувати ваги ребер у живому режимі. Це робить алгоритм придатним до використання в реальних навігаційних системах із мінливими параметрами.

Алгоритм також володіє гнучкою архітектурою, яка дозволяє адаптувати його під конкретні вимоги предметної області. Наприклад, можна використовувати різні типи евристик – від простих геометричних до складних, побудованих на основі історичних даних або прогнозів дорожнього руху. Це розширює прикладні можливості алгоритму та робить його корисним у широкому спектрі завдань.

Незважаючи на значні переваги, A* має й певні обмеження. Зокрема, якість його роботи сильно залежить від обраної евристики. Якщо евристика є неточною або завищує відстань до цілі, алгоритм може втратити оптимальність або навіть працювати повільніше, ніж Дейкстри. Тому правильний підбір евристики є критично важливим для забезпечення ефективності.

Алгоритм A* дає змогу ефективно поєднати точність і швидкодію, що робить його одним із найкращих рішень для задач пошуку маршруту на практиці.

Його застосування у системі, яка працює з даними Google Maps API, забезпечить оптимальний баланс між продуктивністю та достовірністю отриманих маршрутів. У поєднанні з іншими алгоритмами пошуку він дозволяє сформувати комплексне уявлення про ефективність різних підходів у реальних умовах роботи з транспортними графами.

Роботу алгоритму A* доцільно супроводжувати блок-схемою, яка наочно демонструє його логіку. Така схема наведена на рисунку 1.3. На початку відбувається ініціалізація оцінок для стартової вершини та формування відкритої множини. Далі виконується цикл: перевірка, чи не спорожніла відкрита множина; вибір вершини з мінімальним значенням f; перевірка, чи не є ця вершина цільовою; виконання релаксації для всіх сусідніх вершин; оновлення множин та повторення процесу. Після досягнення цільової вершини відновлюється повний маршрут за інформацією про попередників.

Алгоритм A* належить до евристичних алгоритмів пошуку найкоротшого шляху на зважених графах та поєднує в собі ідеї класичного алгоритму Дейкстри з використанням додаткової інформації про наближення до цільової вершини. Нехай задано граф

$$G = (V, E), w: E \to \mathbb{R}_{\geq 0},$$

де V – множина вершин, E – множина ребер, а $w(u, v)$ – вага ребра між вершинами u та v.

Також задано початкову вершину s та цільову вершину t. Метою алгоритму є знаходження шляху з мінімальною сумарною вагою між цими вершинами.

В основі алгоритму A* лежить розділення оцінки шляху на дві складові: фактичну вартість пройденого шляху та евристичну оцінку відстані до цілі. Для кожної вершини v підтримуються три функції. Перша – це $g(v)$, яка описує найменшу відому на поточний момент вартість шляху від початкової вершини s до вершини v. Друга – евристична функція $h(v)$, що оцінює вартість

найкоротшого шляху від вершини vдо цільової вершини t. Третя – узагальнена функція $f(v) = g(v) + h(v)$, яка використовується для пріоритезації вершин під час пошуку.

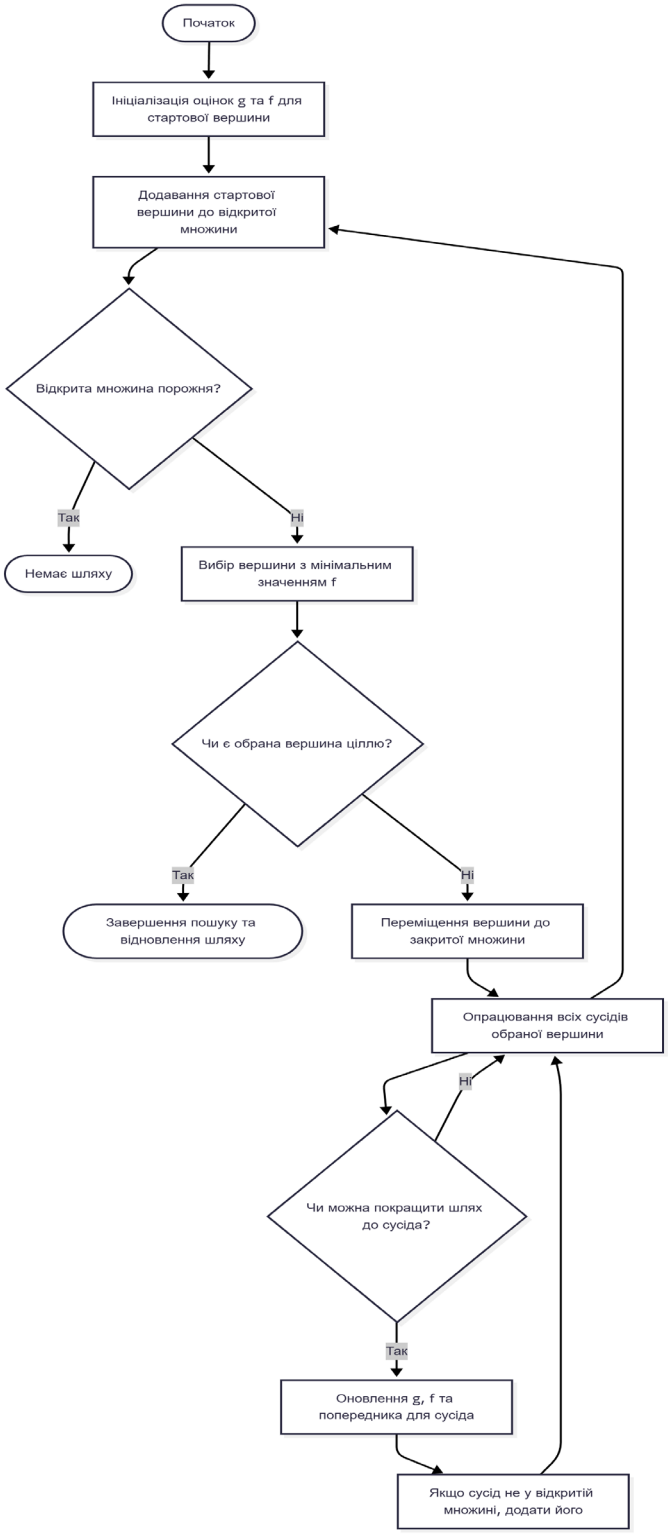

Рисунок 1.3 – Блок-схема алгоритму A*

На початковому етапі задаються стартові умови:

$$g(s) = 0, f(s) = h(s),$$
$$g(v) = \text{INF}, f(v) = \text{INF}, \forall v \in V, v \neq s.$$

Таким чином на початку відомий лише нульовий шлях до стартової вершини, а всі інші вершини вважаються недосяжними. Евристична функція $h(v)$задається окремо й не змінюється в процесі виконання алгоритму. У задачах маршрутизації на географічних графах типовим вибором є геометрична відстань між поточною вершиною та ціллю.

Ключова властивість алгоритму A* пов'язана з вимогами до евристичної функції. Для гарантії оптимальності розв'язку евристика має бути допустимою, тобто не завищувати справжню найкоротшу відстань від vдо t:

$$h(v) \leq d^*(v, t), \forall v \in V,$$

де $d^*(v, t)$– істинна найкоротша відстань між вершинами vі t.

Додатково часто вимагається узгодженість (монотонність) евристики, що означає виконання нерівності $h(u) \leq w(u, v) + h(v)$ для будь-якого ребра (u, v).

Оновлення оцінок у A* здійснюється через модифіковану операцію релаксації. Для кожного ребра (u, v)виконується перевірка, чи не є шлях через вершину uкращим за поточний відомий шлях до вершини v. Формально це записується як

якщо $g(v) > g(u) + w(u, v)$, то
$$g(v) \coloneqq g(u) + w(u, v),$$
$$f(v) \coloneqq g(v) + h(v),$$
$$p(v) \coloneqq u$$

де $p(v)$– попередник вершини vу відновлюваному шляху.

Після кожної успішної релаксації покращується значення фактичної відстані та оновлюється повна оцінка, що враховує наближення до цілі.

Вибір наступної вершини для опрацювання здійснюється не за значенням $g(v)$, як у алгоритмі Дейкстри, а за мінімальним значенням узагальненої функції

$$v_{\text{next}} = \arg\min_{v \in \text{Open}} f(v),$$

де Open – множина вершин, які вже були виявлені, але ще не опрацьовані остаточно.

Такий підхід концентрує пошук у напрямку до цілі: вершини, що одночасно мають малу пройдену відстань і розташовані близько до цільової вершини згідно з евристикою, отримують вищий пріоритет.

У процесі роботи алгоритму підтримуються дві множини вершин – відкрита та закрита. Відкрита множина містить вершини, для яких уже розраховані оцінки gта f, але які ще можуть покращити шляхи до своїх сусідів. Закрита множина містить вершини, які вже були обрані як такі, що мають найменше значення f , і для яких подальше покращення оцінки не передбачається. Перехід вершини з відкритої до закритої множини відбувається в момент її вибору для опрацювання, що забезпечує впорядкованість і завершеність процесу.

1.4 Постановка задачі

Постановка задачі полягає у визначенні способу побудови найкоротшого маршруту між двома географічними точками на основі даних, отриманих через Google Maps API. Дорожня мережа подається у вигляді зваженого графа, де вершини описують точки на карті, а ребра відповідають дорожнім сегментам.

Ваги ребер формуються на основі довжин або часу пересування, що повертається сервісом Google Maps. Метою є побудова маршруту з мінімальною сумарною вагою між заданими точками.

У межах задачі передбачається реалізація трьох алгоритмів пошуку найкоротшого шляху: алгоритму Дейкстри, його двонапрямленої модифікації та алгоритму A*. Вони повинні працювати в єдиному програмному середовищі, використовуючи однакові структури даних, формати вхідної інформації та критерії оцінювання. Такий підхід забезпечує коректне порівняння їхньої ефективності при однакових умовах.

Задача включає формування графа дорожньої мережі на основі API-запитів, нормалізацію отриманих ваг та підготовку даних для роботи алгоритмів. Граф повинен підтримувати можливість обробки великої кількості вершин і ребер, що характерно для реальних маршрутів. Окремо визначаються набір тестових сценаріїв: короткі міські маршрути, маршрути середньої довжини та розширені міжміські шляхи. Кожен сценарій використовується для перевірки поведінки алгоритмів у різних топологіях та масштабах графа.

Реалізовані алгоритми порівнюються за ключовими характеристиками: час виконання, кількість опрацьованих вершин, кількість виконаних релаксацій та потреба в обчислювальних ресурсах. Для кожного маршруту фіксується довжина знайденого шляху, що дозволяє перевірити коректність роботи алгоритмів та відповідність результату реальній картографічній відстані.

На основі отриманих експериментальних даних формуються висновки щодо придатності кожного алгоритму для використання у задачах маршрутизації на базі Google Maps API. Порівняння дозволяє встановити різницю у швидкодії, масштабованості та поведінці алгоритмів на різних типах графів, а також визначити, який з них підходить для інтерфейсів та систем, що працюють у режимі реального часу.

2 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОШУКУ ШЛЯХУ

2.1 Реалізація алгоритму Дейкстри

Алгоритм Дейкстри є одним із базових алгоритмів пошуку найкоротшого шляху у зважених графах з невід’ємними вагами ребер і широко застосовується в задачах маршрутизації та навігації. У межах даної роботи він використовується для побудови оптимального маршруту в моделі дорожньої мережі, сформованої на основі даних Google Maps API. Реалізація цього алгоритму є логічним першим кроком, оскільки він має чітко визначену структуру, гарантовано знаходить оптимальний шлях і добре підходить для використання як еталонний метод.

У програмній реалізації дорожня мережа представляється у вигляді зваженого графа, де вершини відповідають географічним точкам (перехрестям, поворотам або контрольним координатам), а ребра–окремим ділянкам доріг між ними. Вага кожного ребра визначається на основі інформації, отриманої через Google Maps API, і може відповідати довжині ділянки або орієнтовному часу руху. Така модель дозволяє безпосередньо застосовувати алгоритм Дейкстри без додаткових перетворень.

На початковому етапі роботи алгоритму виконується ініціалізація службових структур даних. Для стартової вершини встановлюється відстань, рівна нулю, а для всіх інших вершин–умовно нескінченне значення. Паралельно формується структура для збереження попередників, яка використовується для подальшого відновлення маршруту. Усі вершини додаються до пріоритетної черги, що забезпечує впорядкований доступ до вершин за значенням їх поточної відстані.

Основний цикл алгоритму полягає у послідовному виборі вершини з мінімальною поточною відстанню серед тих, що ще не були остаточно опрацьовані. Така вершина вважається найближчою до стартової на поточному

етапі, і для неї гарантується, що знайдене значення відстані є оптимальним. Після цього вершина вилучається з черги та більше не бере участі у виборі наступних кандидатів.

Для кожної вибраної вершини виконується аналіз усіх суміжних з нею вершин, з якими вона з’єднана ребрами графа. На цьому етапі застосовується операція релаксації, яка полягає у перевірці, чи не є шлях до сусідньої вершини через поточну коротшим за вже відомий. Якщо новий шлях має меншу сумарну вагу, відповідне значення відстані оновлюється, а попередник вершини змінюється.

У реалізації алгоритму особлива увага приділяється ефективному оновленню пріоритетної черги. Після кожної успішної релаксації необхідно змінити пріоритет відповідної вершини в черзі, що забезпечує коректний вибір наступної вершини з мінімальною відстанню. Це є критично важливим для продуктивності алгоритму, особливо при роботі з великими графами, характерними для реальних дорожніх мереж.

Алгоритм Дейкстри у межах даної реалізації може працювати у двох режимах. У першому режимі здійснюється повний обхід графа з обчисленням найкоротших відстаней від стартової вершини до всіх інших. У другому режимі алгоритм завершує роботу після досягнення цільової вершини, що дозволяє скоротити час виконання у задачах пошуку конкретного маршруту між двома точками.

Після завершення основного циклу виконується етап відновлення маршруту. Він ґрунтується на послідовному проходженні масиву попередників, починаючи з цільової вершини та рухаючись у зворотному напрямку до стартової. Отримана послідовність вершин формує оптимальний шлях, який може бути безпосередньо використаний для візуалізації маршруту на карті або подальшого аналізу.

Реалізація алгоритму Дейкстри дозволяє отримати коректний та стабільний результат незалежно від структури графа, за умови що всі ваги ребер є невід’ємними. Це робить його універсальним рішенням для задач

маршрутизації, хоча при великій кількості вершин і ребер він може демонструвати зниження продуктивності через значний обсяг опрацьованих даних.

Використання алгоритму Дейкстри у даній роботі має також методологічне значення. Він слугує базовим підходом, з яким надалі порівнюються двонапрямлений алгоритм Дейкстри та алгоритм A*. Завдяки простій і чітко визначеній логіці роботи результати його реалізації можуть бути легко інтерпретовані та використані для аналізу ефективності інших алгоритмів.

Таким чином, реалізація алгоритму Дейкстри забезпечує надійну основу для побудови системи оптимізації маршрутів. Вона дозволяє перевірити коректність моделі графа, механізмів обробки даних Google Maps API та процесу відновлення маршруту, а також створює базу для подальшого розширення функціональності системи за рахунок використання більш оптимізованих методів пошуку шляху.

Алгоритм Дейкстри вже був реалізований у попередній роботі, на основі якої виконується дане дослідження. У межах цієї реалізації було побудовано програмний модуль пошуку найкоротшого шляху в графі дорожньої мережі з використанням даних Google Maps API. Алгоритм застосовувався для формування оптимальних маршрутів між заданими географічними точками та забезпечував коректне обчислення сумарної ваги шляху.

Наявна реалізація охоплює повний цикл роботи алгоритму, включаючи ініціалізацію відстаней, використання пріоритетної черги для вибору вершини з мінімальною поточною відстанню, виконання операцій релаксації та відновлення маршруту за допомогою масиву попередників. Це дозволяє використовувати реалізований алгоритм як перевірену базу для подальшого розширення системи та слугує вихідною точкою для реалізації двонапрямленого алгоритму Дейкстри та алгоритму A*, що розглядаються у наступних підрозділах.

Код оголошення класу та його основних полів наведено на лістингу 2.1.

Лістинг 2.1 – Код оголошення класу та його основних полів

```
using System.Diagnostics;
using RoadTrip.Application.Enums;
using RoadTrip.Application.Interfaces;
using RoadTrip.Application.Models.Graph;
using RoadTrip.Application.Models.Metrics;

namespace RoadTrip.Application.Services;

public class DijkstraService : IPathfindingService
{
    private List<Node> _nodes;
    private PathfindingMetrics _metrics;
    private Stopwatch _stopwatch;
    private Node _endNode;

    public Dictionary<Node, int> Distances { get; private set; }
    public Dictionary<Node, Node> PreviousNodes { get; private set; }

    public DijkstraService(Graph graph)
    {
        this._nodes = graph.Nodes;
        this._metrics = new PathfindingMetrics { Algorithm =
PathfindingAlgorithm.Dijkstra };
        this._stopwatch = new Stopwatch();
    }
}
```

На лістингу представлено оголошення класу DijkstraService, який реалізує інтерфейс IPathfindingService та інкапсулює логіку виконання алгоритму Дейкстри. Використання окремого сервісного класу відповідає принципам модульної архітектури та дозволяє ізолювати алгоритм пошуку шляху від інших компонентів системи.

Поле _nodes призначене для зберігання списку вершин графа дорожньої мережі, який передається через об’єкт Graph. Це забезпечує доступ алгоритму до структури графа без необхідності повторного звернення до зовнішніх джерел даних. Поле _endNode використовується для збереження цільової вершини, що дозволяє реалізувати пошук маршруту між конкретними початковою та кінцевою точками.

Об’єкт _metrics слугує для збору та збереження метрик виконання алгоритму, зокрема інформації про використаний алгоритм, час виконання та інші показники, необхідні для подальшого аналізу. Ініціалізація цього об’єкта із зазначенням типу алгоритму (PathfindingAlgorithm.Dijkstra) дозволяє уніфікувати процес збору статистики для різних методів пошуку шляху.

Поле _stopwatch використовується для вимірювання часу виконання

алгоритму. Застосування класу Stopwatch з простору імен System.Diagnostics забезпечує точне вимірювання тривалості обчислень і є важливим для подальшого тестування та оцінювання продуктивності реалізації.

Властивості Distances та PreviousNodes призначені для зберігання результатів роботи алгоритму. Перша з них містить поточні найкоротші відстані від початкової вершини до кожної іншої вершини графа, тоді як друга зберігає інформацію про попередні вершини на оптимальному шляху. Обмеження доступу до цих властивостей через модифікатор private set забезпечує контроль над їх оновленням виключно в межах логіки алгоритму.

Конструктор класу приймає об’єкт Graph як параметр, що дозволяє передати сформовану модель дорожньої мережі під час створення сервісу. У конструкторі виконується ініціалізація списку вершин, об’єкта для збору метрик та таймера, що готує клас до подальшого виконання алгоритму пошуку найкоротшого шляху.

Публічний інтерфейс запуску алгоритму та збору метрик виконання наведено на лістингу 2.2.

Лістинг 2.2 – Інтерфейс запуску алгоритму та збору метрик

```
        public List<Node> FindShortestPath(Node start, Node end)
        {
            this._endNode = end;
            this._metrics.StartTime = DateTime.UtcNow;
            this._stopwatch.Start();

            this.Execute(start);
            var path = this.GetShortestPathTo(end);

            this._stopwatch.Stop();
            this._metrics.EndTime = DateTime.UtcNow;
            this._metrics.TotalExecutionTime = this._stopwatch.Elapsed;
            this._metrics.PathFound = path.Count > 1;
            this._metrics.PathLength = path.Count;
            this._metrics.TotalPathWeight      =      this.Distances.ContainsKey(end)      ?
this.Distances[end] : 0;

            // Update all iterations with difference from optimal
            this.UpdateIterationsWithOptimal(this._metrics.TotalPathWeight);

            return path;
        }
```

На даному лістингу представлено публічний метод FindShortestPath, який

є основною точкою входу для запуску алгоритму Дейкстри ззовні. Саме через цей метод ініціюється процес пошуку найкоротшого шляху між заданими початковою (start) та кінцевою (end) вершинами графа. Такий підхід забезпечує чітке розмежування між зовнішнім інтерфейсом сервісу та внутрішньою логікою реалізації алгоритму.

На початку роботи методу зберігається посилання на кінцеву вершину у полі _endNode, що дозволяє використовувати її під час виконання алгоритму та відновлення маршруту. Далі фіксується момент початку виконання алгоритму шляхом збереження часу у властивості StartTime об’єкта метрик, а також запускається таймер _stopwatch, який використовується для точного вимірювання тривалості обчислень.

Основна логіка пошуку шляху виконується у виклику методу Execute(start), який реалізує безпосередній алгоритм Дейкстри, починаючи з початкової вершини. Після завершення обчислень виконується побудова оптимального маршруту за допомогою виклику методу GetShortestPathTo(end), що повертає послідовність вершин, які формують знайдений шлях.

Після отримання результату пошуку робота таймера припиняється, і фіксується час завершення виконання алгоритму. На основі цих даних обчислюється загальний час виконання, який зберігається у полі TotalExecutionTime. Це дозволяє надалі використовувати дану метрику під час тестування та порівняння алгоритмів.

Додатково визначається факт знаходження маршруту, що фіксується у властивості PathFound. Для цього перевіряється кількість вершин у сформованому шляху. Також зберігається довжина маршруту у вершинах (PathLength) та сумарна вага знайденого шляху (TotalPathWeight), яка береться з таблиці найкоротших відстаней для кінцевої вершини.

Окремим етапом виконується оновлення інформації про всі ітерації алгоритму шляхом виклику методу UpdateIterationsWithOptimal. Цей виклик дозволяє зіставити проміжні значення ваг маршрутів із фінальним оптимальним результатом, що є важливим для подальшого аналізу динаміки зближення

алгоритму до оптимального розв’язку.

Таким чином, метод FindShortestPath не лише ініціює та завершує роботу алгоритму Дейкстри, але й централізовано виконує збір усіх ключових метрик виконання. Це робить реалізацію зручною для тестування, аналізу ефективності та подальшого порівняння з іншими алгоритмами пошуку шляху.

Ініціалізація внутрішніх структур даних перед входом до основного циклу алгоритму наведена на лістингу 2.3.

Лістинг 2.3 – Ініціалізація внутрішніх структур даних

```
        private void Execute(Node source)
        {
            var unvisited = new List<Node>(this._nodes);
            this.Distances = this._nodes.ToDictionary(node => node, node => int.MaxValue);
            this.Distances[source] = 0;
            this.PreviousNodes = this._nodes.ToDictionary(node => node, node => 
default(Node));

            int iteration = 0;

            while (unvisited.Any())
            {
                // ...
            }
        }
```

На даному лістингу показано початкову частину методу Execute, який реалізує основну логіку алгоритму Дейкстри. Цей метод є внутрішнім і викликається з публічного API сервісу, отримуючи на вхід початкову вершину source, від якої обчислюються найкоротші шляхи.

Першим кроком формується список unvisited, який містить усі вершини графа та використовується для відстеження вершин, що ще не були остаточно опрацьовані алгоритмом. Ініціалізація цього списку копією повного набору вершин дозволяє контролювати процес обходу та забезпечує коректне завершення алгоритму після обробки всіх доступних вершин.

Далі ініціалізується словник Distances, у якому для кожної вершини зберігається поточна оцінка найкоротшої відстані від початкової точки. На початковому етапі для всіх вершин встановлюється значення int.MaxValue, що інтерпретується як нескінченність та означає відсутність відомого шляху. Для

початкової вершини значення відстані встановлюється рівним нулю, що відповідає формальному означенню алгоритму Дейкстри.

Наступним кроком створюється словник PreviousNodes, який використовується для збереження інформації про попередню вершину на оптимальному шляху. Для кожної вершини на етапі ініціалізації встановлюється значення за замовчуванням, що означає відсутність визначеного попередника. У процесі роботи алгоритму це значення буде оновлюватися під час виконання операцій релаксації.

Змінна iteration вводиться для підрахунку кількості ітерацій основного циклу алгоритму. Вона використовується для збору статистичних даних і подальшого аналізу динаміки роботи алгоритму, зокрема для фіксації проміжних значень ваг шляхів та інших метрик, пов’язаних із кожною ітерацією.

Завершальним елементом фрагмента є умова входу в основний цикл while, який виконується доти, доки у списку unvisited залишаються необроблені вершини. Саме в межах цього циклу відбувається вибір вершини з мінімальною поточною відстанню, виконання операцій релаксації та поступове формування найкоротших шляхів у графі. Таким чином, наведений фрагмент коду відповідає етапу повної підготовки алгоритму до основного обчислювального процесу.

Фрагмент коду, що відповідає вибору наступної вершини для обробки та фіксації метрик ітерації, наведено на лістингу 2.4.

Лістинг 2.4 – Вибір вершин для обробки та фіксації метрик ітерації

```
        while (unvisited.Any())
        {
            iteration++;
            var current = unvisited.OrderBy(node => this.Distances[node]).First();

            // Calculate best known path to end at this moment
            int? bestPathToEnd = this.Distances[this._endNode] == int.MaxValue ? null :
this.Distances[this._endNode];

            // Record iteration metrics
            this._metrics.Iterations.Add(new PathfindingIteration
            {
                IterationNumber = iteration,
                ElapsedTime = this._stopwatch.Elapsed,
                CurrentPathWeight = this.Distances[current] == int.MaxValue ? 0 :
this.Distances[current],
                BestPathWeight = bestPathToEnd,
                CurrentNodeName = current.Name ?? «Unknown»
```

```
        });

        unvisited.Remove(current);
        this._metrics.VisitedNodesCount++;

        if (this.Distances[current] == int.MaxValue)
            break;

        // relaxation loop follows...
    }
```

На цьому лістингу показано основний цикл алгоритму Дейкстри, у межах якого послідовно обирається наступна вершина для обробки та виконується фіксація проміжних результатів роботи алгоритму. Цикл продовжується доти, доки у списку unvisited залишаються необроблені вершини, що відповідає класичній схемі виконання алгоритму.

На початку кожної ітерації збільшується лічильник iteration, який використовується для нумерації кроків алгоритму та подальшого аналізу його динаміки. Далі здійснюється вибір поточної вершини current як тієї, що має мінімальне значення поточної відстані серед усіх необроблених вершин. Такий вибір реалізує ключовий принцип алгоритму Дейкстри, згідно з яким наступною опрацьовується вершина з найменшою відомою відстанню від початкової точки.

Окремо визначається найкраще на поточний момент відоме значення шляху до кінцевої вершини. Якщо для цільового вузла відстань ще не була оновлена і залишається рівною нескінченності, відповідне значення фіксується як null. Це дозволяє відрізнити ситуації, коли шлях до цілі ще не знайдений, від випадків, коли вже існує кандидат на оптимальний маршрут.

Далі формується запис про поточну ітерацію, який додається до колекції Iterations об’єкта метрик. У цьому записі зберігається номер ітерації, час, що минув від початку виконання алгоритму, вага поточного шляху до вибраної вершини, а також найкраща відома вага шляху до кінцевої вершини. Додатково фіксується ідентифікатор або назва поточної вершини, що полегшує подальший аналіз та візуалізацію процесу пошуку.

Після збереження метрик поточна вершина видаляється зі списку необроблених, що означає завершення її обробки та неможливість повторного вибору на наступних ітераціях. Паралельно збільшується лічильник відвіданих

вершин, який використовується як одна з ключових метрик ефективності алгоритму.

Перевірка значення відстані для поточної вершини дозволяє достроково завершити роботу алгоритму у випадку, коли мінімальна відстань серед необроблених вершин дорівнює нескінченності. Це означає, що всі залишкові вершини є недосяжними з початкової точки, і подальше виконання алгоритму не має сенсу.

Таким чином, наведений фрагмент коду поєднує у собі два важливі аспекти реалізації алгоритму Дейкстри: коректний вибір наступної вершини для обробки та детальний збір статистичних даних про перебіг пошуку. Це дозволяє не лише знайти оптимальний маршрут, але й глибоко проаналізувати поведінку алгоритму на кожному етапі його виконання.

Фрагмент коду, що відповідає операції релаксації ребер для поточної вершини, наведено на лістингу 2.5.

Лістинг 2.5 – Операція релаксації ребер для поточної вершини

```
foreach (var edge in current.Edges)
{
    this._metrics.ExploredEdgesCount++;
    var isUnvisited = unvisited.Contains(edge.To);
    if (!isUnvisited) continue;

    var tentativeDistance = this.Distances[current] + edge.TimeInSeconds;
    if (tentativeDistance < this.Distances[edge.To])
    {
        this.Distances[edge.To] = tentativeDistance;
        this.PreviousNodes[edge.To] = current;
    }
}
```

На цьому лістингу представлено ключовий етап алгоритму Дейкстри– операцію релаксації ребер, яка виконується для всіх суміжних вершин поточної обраної вершини current. Саме на цьому етапі відбувається уточнення найкоротших відстаней та формування оптимальної структури маршруту.

Цикл foreach перебирає всі ребра, що виходять із поточної вершини. Кожне ребро представляє можливий перехід до сусідньої вершини графа, а його вага у даній реалізації задається значенням TimeInSeconds, що відповідає часовій

вартості проходження відповідної ділянки дороги. Такий підхід дозволяє оптимізувати маршрут не за геометричною довжиною, а за фактичним або оціненим часом руху.

Під час обробки кожного ребра збільшується лічильник ExploredEdgesCount, який використовується для збору статистики щодо кількості опрацьованих ребер. Ця метрика є важливою для аналізу обчислювальної складності алгоритму та подальшого порівняння з іншими методами пошуку шляху.

Далі виконується перевірка, чи належить кінцева вершина ребра до множини необроблених вершин. Якщо вершина вже була остаточно опрацьована, вона пропускається, що відповідає класичному обмеженню алгоритму Дейкстри та запобігає повторному оновленню відстаней для вже зафіксованих вершин.

Для необроблених вершин обчислюється так звана пробна відстань (tentativeDistance), яка дорівнює сумі найкоротшої відомої відстані до поточної вершини та ваги поточного ребра. Це значення є кандидатом на покращення поточної оцінки відстані до сусідньої вершини.

У випадку, якщо обчислена пробна відстань є меншою за вже збережене значення у словнику Distances, виконується оновлення цієї відстані. Одночасно оновлюється інформація про попередню вершину у словнику PreviousNodes, що дозволяє зафіксувати новий, більш оптимальний шлях до відповідної вершини.

Таким чином, операція релаксації забезпечує поступове покращення оцінок найкоротших шляхів у графі та формує основу для коректного відновлення оптимального маршруту після завершення роботи алгоритму. Саме повторне застосування цього механізму для всіх вершин гарантує знаходження найкоротшого шляху за умови невід’ємних ваг ребер.

Фрагмент коду, що відповідає відновленню (реконструкції) знайденого найкоротшого шляху від кінцевої вершини до початкової, наведено на лістингу 2.6.

Лістинг 2.6 – Відновлення найкоротшого шляху

```
private List<Node> GetShortestPathTo(Node destination)
{
    var path = new List<Node> { destination };
    while (this.PreviousNodes[destination] != null)
    {
        destination = this.PreviousNodes[destination];
        path.Insert(0, destination);
    }
    return path;
}
```

На даному лістингу показано завершальний етап роботи алгоритму Дейкстри, який полягає у формуванні послідовності вершин, що складають оптимальний маршрут між початковою та кінцевою точками. Реконструкція шляху виконується після завершення основних обчислень і ґрунтується на інформації, накопиченій у структурі PreviousNodes.

Метод GetShortestPathTo приймає як параметр кінцеву вершину destination, для якої вже було обчислено найкоротшу відстань від початкової точки. Спочатку створюється список path, до якого одразу додається кінцева вершина. Це дозволяє ініціалізувати маршрут з останнього елемента та надалі поступово доповнювати його попередніми вершинами.

Далі виконується цикл while, у межах якого відбувається рух у зворотному напрямку–від поточної вершини до її попередника. Для кожної вершини перевіряється наявність попередньої вершини у словнику PreviousNodes. Якщо такий попередник існує, він вважається частиною оптимального шляху, і змінна destination оновлюється відповідним значенням.

Кожна знайдена попередня вершина вставляється на початок списку path за допомогою методу Insert(0, destination). Такий підхід забезпечує правильний порядок вершин у фінальному маршруті–від початкової до кінцевої, незважаючи на те, що відновлення шляху відбувається у зворотному напрямку.

Цикл завершується тоді, коли для поточної вершини відсутній попередник, що відповідає досягненню початкової вершини маршруту. Після цього сформований список path містить повну послідовність вершин, які складають найкоротший шлях, знайдений алгоритмом.

Таким чином, наведений метод забезпечує коректне та наочне відновлення

результату роботи алгоритму Дейкстри. Він відокремлює логіку обчислення найкоротших відстаней від логіки формування маршруту, що підвищує зрозумілість реалізації та полегшує подальше використання результатів для візуалізації або аналізу знайденого шляху.

Фрагмент коду, що відповідає завершальному етапу обробки ітераційних метрик після виконання алгоритму, наведено на лістингу 2.7.

Лістинг 2.7 – Завершальний етап обробки ітераційних метрик

```
        private void UpdateIterationsWithOptimal(int optimalWeight)
        {
            foreach (var iteration in this._metrics.Iterations)
            {
                if (iteration.BestPathWeight.HasValue && optimalWeight > 0)
                {
                    iteration.DifferenceFromOptimal = iteration.BestPathWeight.Value -
optimalWeight;
                }
            }
        }

public PathfindingMetrics GetMetrics() => this._metrics;
```

На цьому лістингу показано механізм постобробки зібраних метрик, який виконується після завершення основного пошуку найкоротшого шляху. Метою даного етапу є доповнення кожної ітерації інформацією про відхилення поточного найкращого знайденого шляху від фінального оптимального результату.

Метод UpdateIterationsWithOptimal приймає як параметр вагу оптимального шляху optimalWeight, що була визначена після завершення роботи алгоритму. Далі виконується послідовний обхід усіх записів ітерацій, збережених у колекції Iterations об’єкта метрик. Такий підхід дозволяє ретроспективно проаналізувати кожен крок алгоритму з урахуванням уже відомого оптимального результату.

Для кожної ітерації перевіряється, чи існує на цьому кроці відоме значення найкращого шляху до кінцевої вершини. Якщо таке значення присутнє і фінальна вага оптимального шляху є додатною, обчислюється різниця між поточною найкращою вагою та оптимальною вагою маршруту. Отримане значення

зберігається у полі DifferenceFromOptimal.

Ця метрика дозволяє кількісно оцінити, наскільки швидко алгоритм наближається до оптимального рішення у процесі виконання. Вона є особливо корисною для побудови статистичних таблиць і графіків, що відображають динаміку зближення алгоритму до оптимального маршруту та характер його пошукової поведінки.

Окремо наведено публічний метод GetMetrics, який повертає об’єкт PathfindingMetrics з усіма зібраними даними. Наявність такого методу забезпечує зручний доступ до результатів виконання алгоритму та дозволяє використовувати метрики для подальшого тестування, візуалізації або порівняння з іншими алгоритмами пошуку шляху.

Таким чином, наведений фрагмент коду завершує цикл реалізації алгоритму Дейкстри, доповнюючи результати обчислень аналітичними даними, необхідними для глибокого дослідження ефективності алгоритму та підготовки експериментальної частини роботи.

2.2 Реалізація алгоритму двонаправленого Дейкстри

Двонапрямлений алгоритм Дейкстри є модифікацією класичного алгоритму пошуку найкоротшого шляху, яка спрямована на зменшення обсягу обчислень за рахунок одночасного виконання пошуку з двох напрямків. Він застосовується до зважених графів із невід’ємними вагами ребер і, як і звичайний алгоритм Дейкстри, гарантує знаходження оптимального маршруту між заданими вершинами.

Принцип роботи двонапрямленого алгоритму полягає у паралельному виконанні двох незалежних процесів пошуку. Перший процес починається з початкової вершини та розширює область пошуку у напрямку до цільової, тоді як другий стартує з кінцевої вершини та виконує пошук у зворотному напрямку.

Кожен із цих процесів використовує власні структури даних для збереження відстаней, множин необроблених вершин та попередників, але обидва працюють за тими ж правилами, що й класичний алгоритм Дейкстри.

На кожному кроці алгоритму вибирається вершина з мінімальною поточною відстанню у відповідному напрямку, після чого для неї виконуються операції релаксації. Пошук триває доти, доки області, що досліджуються з обох сторін, не перетнуться. Момент перетину означає, що знайдено спільну вершину або ребро, через яке може бути сформований кандидат на оптимальний шлях.

Головною відмінністю двонапрямленого алгоритму Дейкстри від класичного є значне скорочення кількості опрацьованих вершин і ребер. Замість поступового розширення області пошуку лише від початкової точки, алгоритм одночасно звужує простір пошуку з обох кінців маршруту. Це особливо ефективно у великих графах, де початкова та кінцева вершини розташовані на значній відстані одна від одної.

Завдяки такому підходу двонапрямлений алгоритм дозволяє досягти зменшення часу виконання без втрати точності результату. Саме ця властивість робить його доцільним для використання в системах маршрутизації та навігації, де необхідно швидко знаходити оптимальні маршрути на основі великих дорожніх мереж.

Оголошення сервісу двонапрямленого алгоритму Дейкстри та його конструктора наведено на лістингу 2.8.

Лістинг 2.8 – Оголошення сервісу алгоритму

```
public class BidirectionalDijkstraService : IPathfindingService
{
    private List<Node> _nodes;
    private PathfindingMetrics _metrics;
    private Stopwatch _stopwatch;

    public BidirectionalDijkstraService(Graph graph)
    {
        this._nodes = graph.Nodes;
        this._metrics = new PathfindingMetrics { Algorithm = 
PathfindingAlgorithm.BidirectionalDijkstra };
        this._stopwatch = new Stopwatch();
    }
}
```

На даному лістингу показано оголошення класу BidirectionalDijkstraService, який, аналогічно до реалізації класичного алгоритму Дейкстри, реалізує інтерфейс IPathfindingService. Це забезпечує уніфікований підхід до використання різних алгоритмів пошуку шляху в межах однієї архітектури та дозволяє взаємозамінно застосовувати їх у системі без змін зовнішнього коду.

Структура класу загалом повторює підхід, використаний у реалізації звичайного алгоритму Дейкстри. Зокрема, зберігається список вершин графа _nodes, об’єкт _metrics для збору статистичних даних та таймер _stopwatch для вимірювання часу виконання. Ці компоненти є ідентичними за призначенням і функціональністю, що дозволяє уникнути дублювання пояснень і забезпечує послідовність реалізацій.

Ключова відмінність на цьому етапі полягає в ініціалізації об’єкта метрик. У конструкторі явно вказується тип алгоритму PathfindingAlgorithm.BidirectionalDijkstra, що дозволяє надалі коректно ідентифікувати зібрані дані під час тестування та порівняльного аналізу. Це є важливим для експериментальної частини роботи, оскільки всі алгоритми використовують спільну структуру метрик.

Відсутність полів, пов’язаних із кінцевою вершиною або таблицями відстаней на цьому етапі, пояснюється тим, що двонапрямлений алгоритм Дейкстри оперує двома незалежними наборами структур даних–для прямого та зворотного пошуку. Їх оголошення та ініціалізація виконуються безпосередньо у методах, що реалізують основну логіку алгоритму, і розглядаються у наступних лістингах.

Таким чином, наведений фрагмент коду задає каркас сервісу двонапрямленого алгоритму Дейкстри, зберігаючи спільні архітектурні рішення з класичною реалізацією та створюючи основу для подальшого розгляду специфічних особливостей двонапрямленого пошуку.

Виклик двонапрямленого виконання алгоритму та збір основних метрик наведено на лістингу 2.9.

Лістинг 2.9 – Виклик алгоритму та збір метрик

```
public List<Node> FindShortestPath(Node start, Node end)
{
    this._metrics.StartTime = DateTime.UtcNow;
    this._stopwatch.Start();

    var path = this.Execute(start, end);

    this._stopwatch.Stop();
    this._metrics.EndTime = DateTime.UtcNow;
    this._metrics.TotalExecutionTime = this._stopwatch.Elapsed;
    this._metrics.PathFound = path.Count > 1;
    this._metrics.PathLength = path.Count;

    // Update all iterations with difference from optimal
    this.UpdateIterationsWithOptimal(this._metrics.TotalPathWeight);

    return path;
}
```

На даному лістингу представлено публічний метод FindShortestPath, який є точкою входу для запуску двонапрямленого алгоритму Дейкстри. За своєю структурою цей метод значною мірою повторює відповідний метод у реалізації класичного алгоритму Дейкстри, що дозволяє зберегти єдиний підхід до виклику алгоритмів та збору метрик у системі.

На початку методу фіксується час старту виконання алгоритму та запускається таймер, який використовується для вимірювання тривалості обчислень. Далі відбувається виклик методу Execute(start, end), у якому реалізовано основну логіку двонапрямленого пошуку. На відміну від класичної реалізації, у цьому випадку методу передаються одразу дві вершини–початкова та кінцева, що є принциповою відмінністю і безпосередньо пов’язано з одночасним виконанням пошуку з обох напрямків.

Після завершення виконання основного алгоритму таймер зупиняється, і фіксується час завершення пошуку. На основі цих даних обчислюється загальний час виконання, який зберігається у відповідному полі об’єкта метрик. Це дозволяє безпосередньо порівнювати швидкодію двонапрямленого алгоритму з іншими реалізаціями в однакових умовах.

Аналогічно до класичного алгоритму Дейкстри, визначається факт знаходження маршруту та довжина знайденого шляху у кількості вершин. Вага маршруту при цьому визначається всередині методу Execute і зберігається в

об’єкті метрик, що є характерною особливістю двонапрямленої реалізації.

Завершальним етапом виконується оновлення ітераційних метрик шляхом зіставлення проміжних результатів із фінальною вагою оптимального шляху. Цей крок є ідентичним до відповідного етапу в реалізації класичного алгоритму Дейкстри та забезпечує можливість аналізу динаміки зближення до оптимального маршруту.

Таким чином, наведений метод зберігає загальну структуру публічного API сервісу пошуку шляху, але відрізняється способом запуску основного алгоритму. Це дозволяє реалізувати двонапрямлений пошук без порушення загальної архітектури системи та забезпечує коректне порівняння результатів різних алгоритмів у межах подальшого тестування.

Ініціалізація структур даних для двонапрямленого пошуку наведена на лістингу 2.10.

Лістинг 2.10 – Ініціалізація структур даних

```
// Forward search from start
var forwardDistances = this._nodes.ToDictionary(n => n, n => int.MaxValue);
var forwardPrevious = this._nodes.ToDictionary(n => n, n => default(Node));
var forwardUnvisited = new List<Node>(this._nodes);
forwardDistances[start] = 0;

// Backward search from end
var backwardDistances = this._nodes.ToDictionary(n => n, n => int.MaxValue);
var backwardPrevious = this._nodes.ToDictionary(n => n, n => default(Node));
var backwardUnvisited = new List<Node>(this._nodes);
backwardDistances[end] = 0;

int bestDistance = int.MaxValue;
Node meetingPoint = null;
int iteration = 0;
```

На даному лістингу показано етап ініціалізації, який є ключовою відмінністю двонапрямленого алгоритму Дейкстри від класичної реалізації. На відміну від одностороннього пошуку, тут одночасно готуються дві незалежні множини структур даних–для прямого та зворотного напрямків пошуку.

Для прямого пошуку, що стартує з початкової вершини, створюються словники forwardDistances і forwardPrevious, а також список необроблених вершин forwardUnvisited. Їх призначення повністю відповідає аналогічним

структурам у класичному алгоритмі Дейкстри: збереження поточних найкоротших відстаней, інформації про попередні вершини та контролю необроблених вузлів. Єдиною відмінністю є те, що початкова вершина start отримує нульову відстань саме у прямому напрямку.

Аналогічний набір структур ініціалізується для зворотного пошуку, який стартує з кінцевої вершини end. Словники backwardDistances і backwardPrevious та список backwardUnvisited виконують ті самі функції, але для обходу графа у протилежному напрямку. Встановлення нульової відстані для кінцевої вершини означає початок пошуку від цілі до старту.

Використання двох незалежних наборів структур дозволяє алгоритму виконувати два паралельні процеси пошуку, які розширюють області дослідження назустріч один одному. Це є принциповою відмінністю від класичного алгоритму Дейкстри, де пошук відбувається лише з одного напрямку і поступово охоплює все більшу частину графа.

Змінна bestDistance використовується для збереження найкращої на поточний момент оцінки довжини маршруту, що формується у разі перетину двох пошуків. Її початкове значення встановлюється як нескінченність, що означає відсутність знайденого спільного маршруту на початковому етапі.

Змінна meetingPoint призначена для збереження вершини, у якій відбувається зустріч прямого та зворотного пошуків. Саме через цю вершину згодом формується фінальний оптимальний маршрут, поєднуючи результати обох напрямків.

Лічильник iteration використовується для фіксації кількості ітерацій алгоритму та збору статистичних даних, аналогічно до класичної реалізації. Це дозволяє уніфіковано аналізувати динаміку роботи алгоритмів і порівнювати їх поведінку на однакових наборах даних.

Таким чином, наведений фрагмент коду демонструє основну концептуальну відмінність двонапрямленого алгоритму Дейкстри–поділ процесу пошуку на два симетричні напрями з окремими структурами даних, що створює передумови для зменшення області пошуку та підвищення ефективності

алгоритму.

Фрагмент коду, що реалізує основний цикл двонапрямленого алгоритму Дейкстри з почерговим виконанням прямого та зворотного кроків і перевіркою точки зустрічі пошуків, наведено на лістингу 2.11.

Лістинг 2.11 – Основний цикл алгоритму з почерговим виконанням прямого і зворотнього кроків

```
        while (forwardUnvisited.Any() && backwardUnvisited.Any())
        {
            iteration++;

            // Forward step
            var forwardCurrent = forwardUnvisited.OrderBy(n => forwardDistances[n]).First();
            forwardUnvisited.Remove(forwardCurrent);
            this._metrics.VisitedNodesCount++;

            // Record iteration
            this._metrics.Iterations.Add(new PathfindingIteration
            {
                IterationNumber = iteration,
                ElapsedTime = this._stopwatch.Elapsed,
                CurrentPathWeight = forwardDistances[forwardCurrent] == int.MaxValue ? 0 :
forwardDistances[forwardCurrent],
                BestPathWeight = bestDistance == int.MaxValue ? null : bestDistance,
                CurrentNodeName = $»F:{forwardCurrent.Name}»
            });

            if (forwardDistances[forwardCurrent] == int.MaxValue)
                break;

            foreach (var edge in forwardCurrent.Edges)
            {
                this._metrics.ExploredEdgesCount++;
                if (!forwardUnvisited.Contains(edge.To)) continue;

                var tentative = forwardDistances[forwardCurrent] + edge.TimeInSeconds;
                if (tentative < forwardDistances[edge.To])
                {
                    forwardDistances[edge.To] = tentative;
                    forwardPrevious[edge.To] = forwardCurrent;
                }

                // Check if paths meet
                if (backwardDistances[edge.To] != int.MaxValue)
                {
                    var totalDist = forwardDistances[edge.To] + backwardDistances[edge.To];
                    if (totalDist < bestDistance)
                    {
                        bestDistance = totalDist;
                        meetingPoint = edge.To;
                    }
                }
            }

            // Backward step
            var backwardCurrent = backwardUnvisited.OrderBy(n =>
backwardDistances[n]).First();
            backwardUnvisited.Remove(backwardCurrent);
            this._metrics.VisitedNodesCount++;
```

```
            if (backwardDistances[backwardCurrent] == int.MaxValue)
                break;

            foreach (var edge in this._nodes.Where(n => n.Edges.Any(e => e.To == 
backwardCurrent)))
            {
                this._metrics.ExploredEdgesCount++;
                if (!backwardUnvisited.Contains(edge)) continue;

                var reverseEdge = edge.Edges.FirstOrDefault(e => e.To == backwardCurrent);
                if (reverseEdge == null) continue;

                var tentative = backwardDistances[backwardCurrent] + 
reverseEdge.TimeInSeconds;
                if (tentative < backwardDistances[edge])
                {
                    backwardDistances[edge] = tentative;
                    backwardPrevious[edge] = backwardCurrent;
                }

                // Check if paths meet
                if (forwardDistances[edge] != int.MaxValue)
                {
                    var totalDist = forwardDistances[edge] + backwardDistances[edge];
                    if (totalDist < bestDistance)
                    {
                        bestDistance = totalDist;
                        meetingPoint = edge;
                    }
                }
            }

            // Early termination if we found a path and both searches exceed it
            if (meetingPoint != null &&
                forwardDistances[forwardCurrent] + backwardDistances[backwardCurrent] >= 
bestDistance)
            {
                break;
            }
        }
```

На даному лістингу представлено ключову частину реалізації двонапрямленого алгоритму Дейкстри–основний цикл, у межах якого почергово виконуються кроки прямого та зворотного пошуку. На відміну від класичного алгоритму Дейкстри, де на кожній ітерації опрацьовується лише одна вершина, у цій реалізації кожна ітерація включає два симетричних етапи, що істотно впливає на поведінку алгоритму.

На початку кожної ітерації збільшується лічильник iteration, після чого виконується прямий крок пошуку. Для цього з множини необроблених вершин прямого пошуку вибирається вершина з мінімальною поточною відстанню forwardCurrent. Даний вибір повністю відповідає логіці класичного алгоритму Дейкстри і не потребує додаткових пояснень, оскільки є ідентичним за принципом дії.

Після вибору вершини фіксуються ітераційні метрики, зокрема номер ітерації, поточний час виконання, вага шляху до обраної вершини та найкраща на поточний момент сумарна вага маршруту, якщо така вже відома. Для зручності аналізу назва вершини маркується префіксом F, що дозволяє однозначно ідентифікувати кроки прямого пошуку під час подальшого аналізу журналу ітерацій.

Далі виконується операція релаксації для всіх суміжних ребер поточної вершини прямого пошуку. Цей етап загалом повторює реалізацію класичного алгоритму Дейкстри, однак доповнюється перевіркою на перетин з результатами зворотного пошуку. Якщо для вершини, до якої веде ребро, вже існує обчислена відстань у зворотному напрямку, формується кандидат на повний маршрут шляхом сумування прямих і зворотних відстаней.

У разі, якщо сумарна відстань виявляється меншою за поточне найкраще знайдене значення bestDistance, оновлюється оцінка оптимального шляху та фіксується вершина зустрічі meetingPoint. Саме цей механізм дозволяє алгоритму виявити момент, коли обидва пошуки перетинаються, що є центральною ідеєю двонапрямленого підходу.

Після завершення прямого кроку аналогічним чином виконується зворотний крок пошуку. Вибір поточної вершини backwardCurrent здійснюється за мінімальною відстанню у зворотному напрямку, що повністю відповідає логіці класичного алгоритму Дейкстри. Подальша релаксація ребер виконується з урахуванням зворотної орієнтації графа, що пояснює необхідність додаткового пошуку вхідних ребер.

Під час зворотного кроку також здійснюється перевірка на перетин з результатами прямого пошуку. Якщо для певної вершини вже відома відстань з прямого напрямку, обчислюється сумарна довжина маршруту через цю вершину та, за необхідності, оновлюється найкраща оцінка оптимального шляху.

Завершальним елементом циклу є умова дострокового завершення алгоритму. Якщо точка зустрічі вже знайдена і подальше розширення обох пошуків не може призвести до покращення знайденого результату, виконання

алгоритму припиняється. Такий механізм відсутній у класичному алгоритмі Дейкстри та є важливою оптимізацією двонапрямленого підходу.

Таким чином, наведений фрагмент коду наочно демонструє ключові відмінності двонапрямленого алгоритму Дейкстри від його класичної версії: паралельне виконання пошуку з двох напрямків, виявлення точки зустрічі та можливість дострокового завершення обчислень. Саме ці особливості забезпечують зменшення області пошуку та підвищення ефективності алгоритму на великих графах дорожніх мереж.

Фрагмент коду, що відповідає фіксації фінальної ваги маршруту та перевірці випадку відсутності шляху між заданими вершинами, наведено на лістингу 2.12.

Лістинг 2.12 – Фіксація фінальної ваги маршруту та перевірка відсутності шляху

```
this._metrics.TotalPathWeight = bestDistance;

if (meetingPoint == null)
    return new List<Node> { start };
```

На даному лістингу показано завершальний логічний етап основного алгоритму двонапрямленого пошуку, який виконується після виходу з головного циклу. На цьому етапі приймається рішення щодо наявності коректного маршруту та фіксується його підсумкова вага.

У першому рядку значення змінної bestDistance, яке впродовж виконання алгоритму зберігало найменшу знайдену сумарну вагу маршруту через точку зустрічі двох пошуків, записується у відповідне поле об’єкта метрик. Таким чином, остаточна вага оптимального шляху стає доступною для подальшого аналізу, тестування та порівняння з результатами інших алгоритмів. Цей підхід є аналогічним до фіксації ваги маршруту в класичному алгоритмі Дейкстри, однак значення формується на основі поєднання прямого та зворотного пошуків.

Далі виконується перевірка на випадок, коли точка зустрічі двох напрямків пошуку не була знайдена. Значення meetingPoint, що дорівнює null, означає, що

між початковою та кінцевою вершинами не існує з’єднувального шляху в межах даного графа або ж жоден із пошуків не зміг досягти спільної вершини.

У разі відсутності точки зустрічі метод повертає список, який містить лише початкову вершину. Такий результат сигналізує про неможливість побудови маршруту та дозволяє зовнішнім компонентам системи коректно обробити ситуацію відсутності шляху без виникнення помилок або некоректних даних.

Таким чином, наведений фрагмент коду виконує роль перевірки коректності результату роботи двонапрямленого алгоритму Дейкстри та завершує формування підсумкових метрик. Він забезпечує надійність реалізації у граничних випадках та узгоджує поведінку алгоритму з реалізацією класичного пошуку шляху.

Фрагмент коду, що відповідає реконструкції оптимального маршруту на основі результатів прямого та зворотного пошуків, наведено на лістингу 2.13.

Лістинг 2.13 – Реконструкція оптимального маршруту

```
private List<Node> ReconstructBidirectionalPath(Node start, Node end, Node meeting,
    Dictionary<Node, Node> forwardPrev, Dictionary<Node, Node> backwardPrev)
{
    var path = new List<Node>();

    // Forward path
    var current = meeting;
    var forwardPath = new List<Node>();
    while (current != null)
    {
        forwardPath.Insert(0, current);
        current = forwardPrev[current];
    }

    // Backward path
    current = backwardPrev[meeting];
    var backwardPath = new List<Node>();
    while (current != null)
    {
        backwardPath.Add(current);
        current = backwardPrev[current];
    }

    path.AddRange(forwardPath);
    path.AddRange(backwardPath);

    return path;
}
```

На даному лістингу представлено завершальний етап реалізації двонапрямленого алгоритму Дейкстри–відновлення повного маршруту на основі

двох незалежних ланцюжків попередників, сформованих під час прямого та зворотного пошуків. На відміну від класичного алгоритму Дейкстри, де шлях відновлюється лише з одного набору попередників, у цьому випадку необхідно коректно об’єднати результати двох напрямків.

Метод ReconstructBidirectionalPath приймає як параметри початкову та кінцеву вершини, вершину зустрічі meeting, а також словники попередників для прямого (forwardPrev) і зворотного (backwardPrev) пошуків. Саме вершина зустрічі є спільною точкою, через яку формується оптимальний маршрут, знайдений двонапрямленим алгоритмом.

На першому етапі формується шлях прямого пошуку. Починаючи з вершини зустрічі, алгоритм послідовно переходить до попередніх вершин, використовуючи словник forwardPrev. Оскільки рух відбувається у зворотному напрямку–від точки зустрічі до початкової вершини,–кожна нова вершина вставляється на початок списку forwardPath. Це дозволяє зберегти правильний порядок вершин від початку маршруту до точки зустрічі.

На другому етапі виконується формування зворотної частини маршруту. Для цього використовується словник backwardPrev, який містить інформацію про попередників, отриману під час зворотного пошуку від кінцевої вершини. Початковою точкою цього етапу є вершина, що безпосередньо слідує за точкою зустрічі у напрямку до кінцевої вершини. Вершини додаються до списку backwardPath у прямому порядку, оскільки напрямок обходу вже відповідає кінцевому порядку маршруту.

Після формування обох частин маршруту виконується їх об’єднання. До результуючого списку path спочатку додається послідовність вершин прямого шляху, а потім–послідовність вершин зворотного шляху. Такий підхід забезпечує відсутність дублювання вершини зустрічі та формує цілісний маршрут від початкової до кінцевої точки.

Метод повертає готовий список вершин, що представляє оптимальний маршрут, знайдений двонапрямленим алгоритмом Дейкстри. Відокремлення логіки реконструкції шляху в окремий метод підвищує читабельність реалізації,

спрощує налагодження та дозволяє повторно використовувати цей механізм у разі подальшого розширення або модифікації алгоритму.

Таким чином, наведений фрагмент коду наочно демонструє ключову особливість двонапрямленого підходу–необхідність поєднання результатів двох незалежних пошуків у єдиний оптимальний маршрут, що є принциповою відмінністю від класичного алгоритму Дейкстри.

Реалізація двонапрямленого алгоритму Дейкстри, наведена у даній роботі, демонструє логічне розширення класичного підходу до пошуку найкоротшого шляху з метою зменшення області пошуку та підвищення ефективності обчислень. На відміну від звичайного алгоритму Дейкстри, який виконує пошук лише з початкової вершини, двонапрямлений варіант одночасно задіює два процеси пошуку, що принципово змінює характер виконання алгоритму.

Ключовою відмінністю реалізації є використання двох незалежних наборів структур даних для прямого та зворотного напрямків пошуку. Окремі таблиці відстаней, списки необроблених вершин і словники попередників дозволяють кожному з пошуків розвиватися автономно, зберігаючи при цьому класичну логіку алгоритму Дейкстри в межах кожного напрямку.

На відміну від класичної реалізації, де на кожній ітерації опрацьовується лише одна вершина, у двонапрямленому алгоритмі кожна ітерація включає два послідовних кроки–прямий і зворотний. Це призводить до швидшого звуження області пошуку та зменшення кількості вершин, які необхідно опрацювати до знаходження оптимального маршруту.

Особливо важливою особливістю реалізації є механізм виявлення точки зустрічі двох пошуків. Перевірка перетину результатів прямого та зворотного напрямків дозволяє сформувати кандидата на оптимальний маршрут ще до повного обходу графа. У класичному алгоритмі Дейкстри така можливість відсутня, оскільки оптимальний шлях визначається лише після завершення опрацювання всіх релевантних вершин.

Додатковою відмінністю є використання змінної bestDistance, яка динамічно оновлюється у процесі виконання алгоритму. Вона зберігає найменшу

знайдену сумарну вагу маршруту через точку зустрічі, що дозволяє контролювати якість поточного рішення та використовувати його для дострокового завершення обчислень.

Механізм дострокового завершення є ще однією суттєвою відмінністю двонапрямленого алгоритму від класичного. Якщо подальше розширення областей пошуку не може призвести до покращення знайденого результату, алгоритм припиняє роботу. У звичайному алгоритмі Дейкстри така оптимізація відсутня, що часто призводить до зайвих обчислень.

Реалізація зворотного пошуку потребує додаткової обробки графа, зокрема пошуку вхідних ребер для поточної вершини. Це ускладнює реалізацію порівняно з класичним алгоритмом, але є необхідним для коректного функціонування двонапрямленого підходу. Така особливість підкреслює компроміс між складністю реалізації та виграшем у продуктивності.

Процедура реконструкції маршруту також суттєво відрізняється від класичної реалізації. Замість одного ланцюжка попередників використовується поєднання двох шляхів, отриманих у прямому та зворотному напрямках. Це вимагає додаткової логіки об’єднання результатів, але дозволяє коректно сформувати повний маршрут через точку зустрічі.

Інтеграція збору ітераційних метрик у двонапрямлений алгоритм дозволяє порівнювати його поведінку з класичним алгоритмом Дейкстри на однакових умовах. При цьому аналіз ітерацій демонструє, що оптимальний маршрут, як правило, знаходиться значно раніше, ніж завершуються всі можливі обходи графа.

З точки зору практичного застосування, реалізований двонапрямлений алгоритм Дейкстри є більш придатним для великих дорожніх мереж, де відстань між початковою та кінцевою точками є значною. Саме в таких умовах виграш у кількості опрацьованих вершин і часу виконання стає найбільш помітним.

Разом із тим, реалізація двонапрямленого алгоритму є більш складною порівняно з класичним варіантом, що потребує уважного контролю коректності реалізації обох напрямків пошуку. Проте отримані переваги у швидкодії та

зменшенні області пошуку обґрунтовують доцільність використання цього підходу в системах маршрутизації.

У підсумку, реалізація двонапрямленого алгоритму Дейкстри в межах даної роботи наочно демонструє, як модифікація класичного алгоритму може суттєво підвищити ефективність пошуку найкоротшого шляху. Вона створює логічний міст між базовим підходом і більш складними евристичними алгоритмами, такими як A*, реалізація яких розглядається у наступному підрозділі.

2.3 Реалізація алгоритму AStar

Алгоритм A* є евристичним методом пошуку найкоротшого шляху у зважених графах, який поєднує властивості класичного алгоритму Дейкстри з додатковою оцінкою відстані до цільової вершини. Він широко застосовується у задачах маршрутизації та навігації, особливо у випадках, коли граф має просторову інтерпретацію, зокрема при роботі з географічними координатами. За умови коректного вибору евристичної функції алгоритм A* гарантує знаходження оптимального маршруту.

Принцип роботи алгоритму A* ґрунтується на використанні функції оцінювання, яка для кожної вершини визначає сумарну вартість маршруту як суму вже пройденої відстані від початкової точки та евристичної оцінки відстані до цілі. На кожному кроці алгоритм обирає для обробки вершину з мінімальним значенням цієї функції, що дозволяє спрямовувати пошук у бік кінцевої вершини, а не рівномірно розширювати область пошуку.

На відміну від класичного алгоритму Дейкстри, який не враховує інформацію про положення цільової вершини і поступово досліджує всі можливі напрями, алгоритм A* використовує додаткову евристичну інформацію для фокусування пошуку. Це дозволяє істотно зменшити кількість опрацьованих

вершин і ребер, особливо у великих графах дорожніх мереж.

Порівняно з двонапрямленим алгоритмом Дейкстри, A* не виконує пошук з двох сторін, а натомість використовує одну спрямовану область пошуку, яка динамічно коригується на основі евристичної функції. Такий підхід спрощує структуру реалізації, водночас зберігаючи високу ефективність за рахунок зменшення простору пошуку.

Основною відмінністю алгоритму A* від двох попередніх методів є наявність евристичної складової, яка напряму впливає на порядок обробки вершин. У випадку правильно підібраної евристики, зокрема на основі евклідової або геодезичної відстані між координатами, алгоритм демонструє значну перевагу в швидкодії без втрати точності результату. Саме ця особливість робить алгоритм A* особливо придатним для використання у сучасних навігаційних системах, що працюють з реальними картографічними даними.

Оголошення сервісу алгоритму A* та його основних структур даних наведено на лістингу 2.14.

Лістинг 2.14 – Оголошення сервісу алгоритму A*

```
public class AStarService : IPathfindingService
{
    private List<Node> _nodes;
    private PathfindingMetrics _metrics;
    private Stopwatch _stopwatch;
    private Node _target;

    public Dictionary<Node, int> GScore { get; private set; }
    public Dictionary<Node, int> FScore { get; private set; }
    public Dictionary<Node, Node> PreviousNodes { get; private set; }

    public AStarService(Graph graph)
    {
        this._nodes = graph.Nodes;
        this._metrics = new PathfindingMetrics { Algorithm = 
PathfindingAlgorithm.AStar };
        this._stopwatch = new Stopwatch();
    }
}
```

На даному лістингу показано оголошення класу AStarService, який, аналогічно до реалізацій алгоритму Дейкстри та двонапрямленого алгоритму Дейкстри, реалізує інтерфейс IPathfindingService. Це забезпечує єдиний програмний інтерфейс для запуску різних алгоритмів пошуку шляху та

уніфікований механізм збору метрик, що є важливим для подальшого тестування та порівняльного аналізу.

Базові поля класу _nodes, _metrics та _stopwatch є ідентичними за призначенням до відповідних елементів у попередніх реалізаціях. Вони використовуються для зберігання графа дорожньої мережі, фіксації статистичних даних виконання алгоритму та вимірювання часу роботи. Оскільки їх функціональність повністю збігається з уже розглянутими реалізаціями, додаткового пояснення ці елементи не потребують.

Ключовою відмінністю реалізації алгоритму A* є наявність поля _target, яке зберігає цільову вершину пошуку. На відміну від класичного та двонапрямленого алгоритмів Дейкстри, де інформація про кінцеву вершину використовується переважно для завершення або реконструкції маршруту, в алгоритмі A* цільова вершина безпосередньо бере участь у процесі вибору наступної вершини для обробки через евристичну функцію.

Принциповою особливістю реалізації A* є використання двох окремих словників оцінок–GScore та FScore. Словник GScore зберігає фактичну вартість шляху від початкової вершини до кожної вершини графа. За своєю роллю він є аналогом таблиці відстаней у класичному алгоритмі Дейкстри та не потребує додаткових концептуальних пояснень.

Словник FScore, на відміну від GScore, є специфічним для алгоритму A* і зберігає значення функції оцінювання, яка поєднує фактичну вартість шляху з евристичною оцінкою відстані до цільової вершини. Саме використання FScore дозволяє алгоритму A* спрямовувати пошук у бік цілі та зменшувати кількість опрацьованих вершин.

Словник PreviousNodes використовується для збереження інформації про попередні вершини на оптимальному шляху та повністю відповідає за призначенням аналогічній структурі у реалізаціях алгоритму Дейкстри. Його логіка використання та реконструкції маршруту є ідентичною, що дозволяє уникнути дублювання коду та пояснень.

У конструкторі класу виконується ініціалізація списку вершин графа,

об’єкта метрик із зазначенням типу алгоритму PathfindingAlgorithm.AStar та таймера. Це забезпечує коректну ідентифікацію алгоритму під час збору статистичних даних і зберігає єдиний підхід до організації сервісів пошуку шляху в системі.

Таким чином, наведений фрагмент коду відображає основні архітектурні відмінності алгоритму A* від реалізацій алгоритму Дейкстри та його двонапрямленого варіанту. Ключова різниця полягає у введенні евристичної складової та додаткових структур оцінювання, які визначають спрямований характер пошуку та забезпечують підвищення ефективності алгоритму на просторово орієнтованих графах.

Реалізація евристичної функції, що використовується в алгоритмі A* для оцінювання відстані до цільової вершини, наведена на лістингу 2.15.

Лістинг 2.15 – Евристична функція

```
private int HeuristicCostEstimate(Node from, Node to)
{
    // Haversine formula for geographical distance
    var lat1 = from.Location.Latitude * Math.PI / 180;
    var lat2 = to.Location.Latitude * Math.PI / 180;
    var dLat = (to.Location.Latitude - from.Location.Latitude) * Math.PI / 180;
    var dLon = (to.Location.Longitude - from.Location.Longitude) * Math.PI / 180;

    var a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
            Math.Cos(lat1) * Math.Cos(lat2) *
            Math.Sin(dLon / 2) * Math.Sin(dLon / 2);

    var c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
    var distance = 6371000 * c; // Earth radius in meters

    // Convert to approximate time (assuming 50 km/h average speed)
    return (int)(distance / 50000 * 3600);
}
```

На даному лістингу показано реалізацію евристичної функції, яка є принциповою відмінністю алгоритму A* від алгоритму Дейкстри та його двонапрямленого варіанту. Саме наявність евристичної оцінки визначає спрямований характер пошуку та впливає на порядок обробки вершин у процесі виконання алгоритму.

Метод HeuristicCostEstimate призначений для оцінювання наближеної вартості шляху між поточною вершиною from та цільовою вершиною to. На

відміну від фактичної вартості шляху, яка обчислюється на основі реальних ваг ребер графа, евристична оцінка не використовує інформацію про структуру графа, а ґрунтується виключно на географічному положенні вершин.

У даній реалізації для обчислення географічної відстані між двома точками використовується формула гаверсинусів (Haversine formula). Вона дозволяє визначити довжину найкоротшої дуги між двома точками на поверхні сфери за їх географічними координатами. Використання цієї формули є доцільним у задачах маршрутизації, оскільки координати вершин отримуються з картографічних даних Google Maps API.

На початковому етапі координати широти та довготи переводяться з градусів у радіани, що є необхідним для коректного застосування тригонометричних функцій. Далі обчислюється допоміжне значення a, яке враховує різницю координат і використовується для визначення центрального кута між двома точками. Значення c відповідає цьому куту в радіанах.

Отриманий кут множиться на радіус Землі, який у реалізації прийнято рівним 6 371 000 метрів, у результаті чого обчислюється географічна відстань між двома вершинами у метрах. Це значення є евклідовою оцінкою мінімально можливої довжини маршруту між точками за відсутності будь-яких обмежень дорожньої мережі.

Оскільки у даній системі ваги ребер графа виражені у секундах часу руху, отримана відстань додатково перетворюється у часову оцінку. Для цього використовується припущення про середню швидкість руху 50 км/год, що дозволяє привести евристичну оцінку до тієї ж розмірності, що й фактичні ваги ребер.

Застосована евристична функція є допустимою, оскільки вона ніколи не переоцінює реальну вартість оптимального маршруту. Це забезпечує збереження головної властивості алгоритму A*–гарантії знаходження оптимального шляху за умови коректної реалізації.

Таким чином, наведена евристична функція є ключовим елементом реалізації алгоритму A*, що принципово відрізняє його від алгоритмів Дейкстри.

Вона дозволяє спрямовувати процес пошуку у бік цільової вершини, зменшуючи кількість опрацьованих вершин і підвищуючи ефективність алгоритму при роботі з реальними географічними даними.

Ініціалізація основних структур даних для виконання алгоритму A* наведена на лістингу 2.16.

Лістинг 2.16 – Ініціалізація основних структур даних

```
        private List<Node> Execute(Node start, Node goal)
        {
            var openSet = new List<Node> { start };

            this.GScore = this._nodes.ToDictionary(node => node, node => int.MaxValue);
            this.GScore[start] = 0;

            this.FScore = this._nodes.ToDictionary(node => node, node => int.MaxValue);
            this.FScore[start] = this.HeuristicCostEstimate(start, goal);

            this.PreviousNodes = this._nodes.ToDictionary(node => node, node => 
default(Node));

            int iteration = 0;
            // ...
        }
```

На даному лістингу представлено ініціалізацію структур, які визначають специфіку алгоритму A* та відрізняють його від алгоритму Дейкстри і двонапрямленого алгоритму Дейкстри. Хоча загальна ідея підготовки таблиць відстаней і попередників зберігається, в A* ключовою є робота з множиною відкритих вершин та двома типами оцінок вартості.

Першим кроком формується список openSet, який містить вершини, доступні для подальшого розгляду алгоритмом. На відміну від класичного алгоритму Дейкстри, де зазвичай використовується множина необроблених вершин, що ініціалізується всіма вершинами графа, в A* пошук починається лише зі стартової вершини. Надалі до openSet додаються лише ті вершини, що були досягнуті в процесі релаксації і становлять потенційний інтерес для продовження пошуку.

Далі ініціалізується словник GScore, який зберігає фактичну вартість найкращого відомого шляху від стартової вершини до кожної вершини графа. За своєю суттю GScore є аналогом таблиці Distances у реалізації алгоритму

Дейкстри, оскільки саме він відображає реальну накопичену вагу маршруту. Початковій вершині присвоюється значення нуль, а для решти вершин встановлюється нескінченність.

Наступним кроком формується словник FScore, який є специфічним для A* і відсутній у двох попередніх алгоритмах. Значення FScore для вершини визначає пріоритет її обробки та обчислюється як сума фактичної вартості шляху GScore і евристичної оцінки відстані до цілі. Для стартової вершини FScore одразу ініціалізується значенням евристики HeuristicCostEstimate(start, goal), оскільки фактична вартість на старті дорівнює нулю.

Словник PreviousNodes ініціалізується аналогічно до реалізацій алгоритму Дейкстри. Він призначений для збереження попередників на оптимальному шляху та використовується для реконструкції маршруту після завершення пошуку. Оскільки механізм відновлення маршруту базується на тій самій ідеї, структура даних є ідентичною та не потребує повторного концептуального пояснення.

Лічильник iteration вводиться для фіксації номерів ітерацій алгоритму та збору проміжних метрик. За своєю роллю він узгоджується з попередніми реалізаціями та забезпечує можливість аналізу динаміки роботи A* у порівнянні з алгоритмами Дейкстри.

Таким чином, наведений фрагмент коду відображає ключову відмінність A*–заміну повної множини необроблених вершин на множину відкритих вершин openSet, а також використання двох функціонально різних оцінок вартості (GScore і FScore). Саме ці елементи забезпечують спрямований характер пошуку та потенційне зменшення кількості опрацьованих вузлів у порівнянні з класичними алгоритмами без евристики.

Фрагмент коду, що реалізує основний цикл алгоритму A* з вибором вершини за мінімальною F-оцінкою та перевіркою досягнення цілі, наведено на лістингу 2.17.

Лістинг 2.17 – Основий цикл алгоритму з вибором вершими за мінімальною F-оцінкою

```
        while (openSet.Any())
        {
            iteration++;
            var current = openSet.OrderBy(node => this.FScore[node]).First();

            // Record iteration metrics
            this._metrics.Iterations.Add(new PathfindingIteration
            {
                IterationNumber = iteration,
                ElapsedTime = this._stopwatch.Elapsed,
                CurrentPathWeight = this.GScore[current] == int.MaxValue ? 0 :
this.GScore[current],
                BestPathWeight = this.FScore[current] == int.MaxValue ? null :
this.FScore[current],
                CurrentNodeName = current.Name ?? «Unknown»
            });

            if (current == goal)
            {
                return this.ReconstructPath(goal);
            }

            openSet.Remove(current);
            this._metrics.VisitedNodesCount++;
            // ...
        }
```

На даному лістингу представлено центральний елемент реалізації алгоритму A*–основний цикл пошуку, у межах якого відбувається вибір наступної вершини для обробки та перевірка досягнення цільової вершини. Саме на цьому етапі найбільш чітко проявляється принципова відмінність A* від алгоритму Дейкстри та його двонапрямленого варіанту.

На початку кожної ітерації лічильник iteration збільшується, після чого з множини відкритих вершин openSet обирається вершина з мінімальним значенням FScore. На відміну від алгоритму Дейкстри, де вибір здійснюється виключно на основі накопиченої вартості шляху, в A* використовується комбінована оцінка, що враховує як вже пройдений шлях, так і евристичну оцінку відстані до цілі. Це забезпечує спрямований характер пошуку.

Далі виконується фіксація ітераційних метрик. Записуються номер ітерації, поточний час виконання, фактична вага маршруту до вибраної вершини (GScore), а також її F-оцінка, яка відображає найкращу на поточний момент оцінку повного маршруту через цю вершину. Такий набір даних дозволяє проаналізувати, як алгоритм поступово наближається до цільової вершини.

Після цього виконується перевірка, чи є поточна вершина цільовою. У разі досягнення вершини goal алгоритм негайно завершує роботу та переходить до реконструкції маршруту. Це є суттєвою відмінністю від алгоритму Дейкстри, який продовжує виконання до повного вичерпання множини необроблених вершин або до остаточного закріплення оптимальних відстаней.

Дострокове завершення пошуку є характерною рисою алгоритму A* і безпосередньо пов’язане з використанням допустимої евристичної функції. За цієї умови перше досягнення цільової вершини гарантує оптимальність знайденого маршруту, що дозволяє уникнути зайвих обчислень.

У випадку, якщо цільова вершина ще не досягнута, поточна вершина видаляється з множини openSet і вважається обробленою. Лічильник відвіданих вершин збільшується, що є ідентичним за призначенням до відповідних етапів у реалізаціях алгоритму Дейкстри та двонапрямленого алгоритму Дейкстри.

Таким чином, наведений фрагмент коду демонструє ключову логічну відмінність A*–вибір вершини за F-оцінкою та можливість негайного завершення пошуку після досягнення цілі. Саме ці особливості забезпечують високу ефективність алгоритму при роботі з просторово орієнтованими графами та роблять його придатним для використання у навігаційних системах на основі реальних картографічних даних.

Фрагмент коду, що реалізує операцію релаксації сусідніх вершин з оновленням F-оцінки в алгоритмі A*, наведено на лістингу 2.18.

Лістинг 2.18 – операція релаксації сусідів з оновленням оцінки

```
        foreach (var edge in current.Edges)
        {    this._metrics.ExploredEdgesCount++;
            var neighbor = edge.To;
            var tentativeGScore = this.GScore[current] + edge.TimeInSeconds;
            if (tentativeGScore < this.GScore[neighbor])
            {
                this.PreviousNodes[neighbor] = current;
                this.GScore[neighbor] = tentativeGScore;
                this.FScore[neighbor]          =          this.GScore[neighbor]          +
this.HeuristicCostEstimate(neighbor, goal);
                if (!openSet.Contains(neighbor))
                {.Add(neighbor);
                }    }
        }
```

На даному лістингу представлено етап релаксації сусідніх вершин, який є спільним за ідеєю для всіх розглянутих алгоритмів, проте має принципові відмінності у реалізації в алгоритмі A*. Як і в алгоритмі Дейкстри, релаксація полягає у перевірці можливості покращення поточної оцінки шляху до сусідньої вершини, однак критерій вибору та подальшої обробки вершин у цьому випадку є іншим.

Для кожного ребра, що виходить з поточної вершини current, збільшується лічильник опрацьованих ребер, що є ідентичним підходом до реалізацій алгоритму Дейкстри та двонапрямленого алгоритму Дейкстри. Це дозволяє коректно порівнювати кількість виконаних операцій релаксації між різними алгоритмами.

Далі визначається сусідня вершина neighbor та обчислюється пробна фактична вартість маршруту tentativeGScore, яка відповідає сумі вже накопиченої вартості шляху до поточної вершини та ваги ребра. За своїм змістом це значення повністю відповідає поняттю пробної відстані у класичному алгоритмі Дейкстри.

У випадку, якщо нова фактична вартість є меншою за раніше збережене значення для сусідньої вершини, виконується оновлення інформації про оптимальний шлях. Зокрема, у словнику PreviousNodes фіксується новий попередник, а в GScore зберігається покращена фактична вартість маршруту.

Ключовою відмінністю від алгоритмів без евристики є оновлення словника FScore. Для сусідньої вершини обчислюється нове значення F-оцінки як сума фактичної вартості шляху та евристичної оцінки відстані до цільової вершини. Саме це значення надалі використовується для визначення пріоритету вершини у множині відкритих вершин.

Після оновлення оцінок виконується перевірка на наявність сусідньої вершини у множині openSet. Якщо вершина ще не була додана до цієї множини, вона включається до списку кандидатів на подальшу обробку. На відміну від алгоритму Дейкстри, де зазвичай оперується повною множиною необроблених вершин, A* працює з динамічною множиною відкритих вершин, що

безпосередньо впливає на ефективність пошуку.

Таким чином, наведений фрагмент коду ілюструє ключову особливість алгоритму A*–поєднання класичної релаксації з евристичним оцінюванням. Саме на цьому етапі формується спрямований характер пошуку, який дозволяє алгоритму концентрувати обчислення у напрямку цільової вершини та істотно зменшувати кількість опрацьованих вершин у порівнянні з алгоритмами Дейкстри.

Фрагмент коду, що відповідає реконструкції оптимального маршруту в алгоритмі A*, наведено на лістингу 2.19.

Лістинг 2.19 – Реконструкція оптимального маршруту

```
private List<Node> ReconstructPath(Node current)
{
    var path = new List<Node> { current };
    while (this.PreviousNodes[current] != null)
    {
        current = this.PreviousNodes[current];
        path.Insert(0, current);
    }
    return path;
}
```

На даному лістингу показано завершальний етап роботи алгоритму A*, який полягає у відновленні послідовності вершин, що формують знайдений оптимальний маршрут. Логіка реконструкції шляху повністю збігається з відповідною процедурою, використаною в реалізації класичного алгоритму Дейкстри, що дозволяє уникнути дублювання концептуально однакових механізмів.

Метод ReconstructPath приймає поточну вершину current, яка на момент виклику відповідає цільовій вершині маршруту. Саме досягнення цієї вершини в основному циклі алгоритму A* є умовою завершення пошуку, що відрізняє його від алгоритму Дейкстри, де завершення обчислень не завжди пов’язане з негайним досягненням цілі.

На початковому етапі формується список path, до якого одразу додається цільова вершина. Далі у циклі while виконується послідовний перехід до попередніх вершин, збережених у словнику PreviousNodes. Кожна наступна

вершина вставляється на початок списку, що забезпечує правильний порядок маршруту–від початкової до кінцевої вершини.

Оскільки структура PreviousNodes заповнюється під час операцій релаксації та оновлюється лише у разі покращення шляху, ланцюжок попередників однозначно визначає оптимальний маршрут. За цієї умови процедура реконструкції є коректною та не потребує додаткових перевірок.

Використання окремого методу для відновлення маршруту підвищує читабельність реалізації та дозволяє ізолювати логіку пошуку від логіки формування результату. Такий підхід є спільним для реалізацій алгоритму Дейкстри та алгоритму A* і забезпечує узгодженість програмної архітектури.

Таким чином, наведений фрагмент коду завершує реалізацію алгоритму A*, формуючи фінальний маршрут на основі накопиченої інформації про попередні вершини. Він демонструє, що попри суттєві відмінності в механізмах вибору та оцінювання вершин, етап реконструкції шляху в A* залишається концептуально ідентичним класичному підходу, використаному в алгоритмі Дейкстри.

Реалізація алгоритму A* у межах даної роботи демонструє принципово інший підхід до організації пошуку найкоротшого шляху порівняно з класичним алгоритмом Дейкстри. Основна відмінність полягає у використанні евристичної інформації про наближеність вершин до цільової точки, що дозволяє спрямувати процес пошуку та зменшити обсяг непотрібних обчислень.

На відміну від алгоритму Дейкстри, який розглядає всі вершини графа як потенційні кандидати для обробки та поступово розширює область пошуку рівномірно в усіх напрямках, алгоритм A* працює з динамічною множиною відкритих вершин. До цієї множини потрапляють лише ті вершини, які були досягнуті в процесі релаксації і мають перспективу формування оптимального маршруту.

Ключовою відмінністю реалізації A* є розділення оцінки вартості маршруту на дві складові–фактичну вартість шляху від початкової вершини (GScore) та евристичну оцінку відстані до цілі. У класичному алгоритмі

Дейкстри використовується лише одна величина, яка відображає вже накопичену вартість маршруту, без урахування положення цільової вершини.

Використання F-оцінки, що поєднує GScore та евристику, змінює критерій вибору наступної вершини для обробки. У результаті алгоритм A* віддає перевагу тим вершинам, які одночасно мають низьку фактичну вартість шляху та розташовані ближче до цільової точки. Саме цей механізм забезпечує більш цілеспрямований характер пошуку.

Ще однією суттєвою відмінністю є можливість негайного завершення роботи алгоритму після досягнення цільової вершини. За умови використання допустимої евристичної функції перше знаходження цілі гарантує оптимальність маршруту. У класичному алгоритмі Дейкстри таке дострокове завершення не завжди можливе, оскільки оптимальність шляху підтверджується лише після завершення відповідних обчислень.

Реалізація евристичної функції на основі географічної відстані між вершинами є важливим практичним аспектом алгоритму A*. На відміну від алгоритму Дейкстри, який не використовує інформацію про координати вершин, A* безпосередньо інтегрує просторові дані у процес пошуку, що є особливо актуальним для задач маршрутизації з використанням картографічних сервісів.

Інтеграція евристики потребує додаткових обчислень на кожному кроці релаксації, що ускладнює реалізацію алгоритму порівняно з класичним Дейкстрою. Проте зменшення кількості оброблених вершин і ребер зазвичай компенсує ці витрати, особливо на великих графах дорожніх мереж.

Процедура релаксації в A* зберігає загальну логіку класичного підходу, але доповнюється обов’язковим оновленням F-оцінки для кожної вершини. Це створює тісний зв’язок між локальними рішеннями та глобальною ціллю пошуку, чого не спостерігається в алгоритмі Дейкстри.

Попри суттєві відмінності в механізмах вибору та обробки вершин, етап реконструкції маршруту в A* повністю збігається з класичною реалізацією алгоритму Дейкстри. Це підтверджує, що основні відмінності між алгоритмами зосереджені саме у фазі пошуку, а не у формуванні результату.

З точки зору збору метрик, реалізація A* дозволяє детально проаналізувати вплив евристичної складової на процес пошуку. Фіксація значень G- та F-оцінок на кожній ітерації створює можливість дослідити динаміку наближення алгоритму до оптимального маршруту та порівняти її з поведінкою алгоритму Дейкстри.

У практичному контексті реалізований алгоритм A* є більш придатним для систем навігації, що працюють з великими просторовими графами та мають доступ до координат вершин. Саме в таких умовах його спрямований характер дозволяє досягти істотного виграшу у швидкодії порівняно з класичним алгоритмом Дейкстри.

У підсумку, реалізація алгоритму A* у межах даної роботи наочно демонструє, як використання евристичної інформації змінює парадигму пошуку найкоротшого шляху. Вона суттєво відрізняється від класичного алгоритму Дейкстри не лише з точки зору продуктивності, але й за концепцією прийняття рішень, що робить A* важливим елементом сучасних систем оптимізації маршрутів.

2.4 Основні відмінності в роботі алгоритмів

Алгоритми Дейкстри, двонапрямлений алгоритм Дейкстри та алгоритм A* мають спільну мету–знаходження найкоротшого шляху між двома вершинами зваженого графа, однак принципи їх роботи та підходи до організації пошуку істотно відрізняються. Ці відмінності проявляються як на концептуальному рівні, так і на рівні практичної реалізації.

Класичний алгоритм Дейкстри базується на рівномірному розширенні області пошуку від початкової вершини. На кожному кроці обирається вершина з мінімальною поточною відстанню, після чого виконується релаксація всіх суміжних ребер. Такий підхід не враховує розташування цільової вершини, що

призводить до обробки значної кількості вершин, які не лежать на оптимальному маршруті.

Двонапрямлений алгоритм Дейкстри змінює цю парадигму шляхом одночасного виконання двох незалежних пошуків–з початкової та з кінцевої вершин. Це дозволяє суттєво зменшити простір пошуку, оскільки кожен з напрямків опрацьовує лише частину графа до моменту їх перетину. Принциповою відмінністю є поява точки зустрічі, через яку формується оптимальний маршрут.

У реалізації двонапрямленого алгоритму з’являється необхідність підтримки двох наборів структур даних для відстаней, попередників та необроблених вершин. Це ускладнює програмну реалізацію порівняно з класичним алгоритмом Дейкстри, але забезпечує можливість дострокового завершення пошуку, чого не передбачає стандартний підхід.

Алгоритм A* використовує принципово інший механізм оптимізації пошуку, заснований на евристичній оцінці відстані до цільової вершини. На відміну від двох варіантів алгоритму Дейкстри, A* не потребує виконання пошуку з двох напрямків, а натомість спрямовує єдиний процес пошуку у бік цілі.

Ключовою відмінністю A* є використання комбінованої функції оцінювання, яка поєднує фактичну вартість вже пройденого шляху з евристичною оцінкою відстані до кінцевої точки. Це змінює критерій вибору вершини для обробки та дозволяє алгоритму концентрувати обчислення у перспективних напрямках.

На відміну від алгоритму Дейкстри, який зазвичай оперує повною множиною необроблених вершин, алгоритм A* працює з динамічною множиною відкритих вершин. Це зменшує кількість кандидатів для обробки та сприяє більш ефективному використанню обчислювальних ресурсів.

Важливою відмінністю у роботі алгоритмів є умови завершення пошуку. Класичний алгоритм Дейкстри не завжди може завершити роботу одразу після досягнення цільової вершини, тоді як двонапрямлений алгоритм використовує

умови дострокового завершення на основі найкращої знайденої ваги маршруту. Алгоритм A* за наявності допустимої евристики завершує роботу одразу після досягнення цільової вершини.

З точки зору використання просторової інформації, алгоритм Дейкстри є повністю незалежним від географічних координат вершин. Натомість алгоритм A* безпосередньо інтегрує просторові дані у процес пошуку, що робить його більш придатним для задач маршрутизації з використанням картографічних сервісів.

Процедури реконструкції маршруту в усіх трьох алгоритмах мають спільну ідею, але різну складність реалізації. У класичному алгоритмі Дейкстри та A* використовується один ланцюжок попередників, тоді як у двонапрямленому алгоритмі необхідно поєднати результати двох напрямків пошуку через точку зустрічі.

З точки зору продуктивності, класичний алгоритм Дейкстри є надійним і простим у реалізації, але може бути неефективним для великих графів. Двонапрямлений алгоритм забезпечує зменшення кількості опрацьованих вершин за рахунок симетричного пошуку, а A* досягає ефективності завдяки евристичному спрямуванню.

У підсумку, принципові відмінності між алгоритмами полягають у способі організації пошуку, використанні додаткової інформації про цільову вершину та механізмах завершення обчислень. Кожен з розглянутих алгоритмів має власні переваги та обмеження, що визначає доцільність його використання залежно від характеристик графа та вимог до швидкодії системи маршрутизації.

3 ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ПОШУКУ ШЛЯХУ

3.1 Умови та методика проведення експерименту

У межах порівняльного аналізу алгоритмів пошуку шляху експериментальні дослідження виконувалися з метою оцінювання їх поведінки та ефективності за різних масштабів вхідних даних. Для забезпечення коректності результатів усі алгоритми працювали в однакових умовах, використовуючи спільну модель графа дорожньої мережі та ідентичні вагові характеристики ребер.

Експеримент було проведено для двох різних сценаріїв пошуку маршруту, що відрізнялися розмірністю графа. Перший сценарій відповідав короткому маршруту, у якому кількість вершин графа становила приблизно 200–250. Другий сценарій моделював пошук маршруту у великому графі, що містив декілька тисяч вершин, імітуючи умови реальної міської або міжміської дорожньої мережі.

Граф у кожному сценарії формувався на основі картографічних даних, отриманих із використанням Google Maps API. Вершини графа відповідали географічним точкам дорожньої мережі, а ребра відображали можливі напрямки руху між ними. Ваги ребер задавалися у вигляді часу проходження відповідних ділянок маршруту, що дозволяло оцінювати алгоритми з точки зору практичної навігаційної задачі.

Для кожного з двох сценаріїв послідовно запускалися три алгоритми пошуку шляху: класичний алгоритм Дейкстри, двонапрямлений алгоритм Дейкстри та алгоритм A*. Початкові та кінцеві вершини для всіх алгоритмів у межах одного сценарію були однаковими, що забезпечувало порівнюваність отриманих результатів та виключало вплив випадкових факторів.

Під час виконання алгоритмів здійснювався детальний збір метрик, що

характеризують процес пошуку. До основних показників належали загальний час виконання алгоритму, кількість відвіданих вершин, кількість опрацьованих ребер, довжина знайденого маршруту та сумарна вага оптимального шляху. Окремо фіксувалися ітераційні дані, які відображали динаміку зміни поточних та найкращих оцінок маршруту на кожному кроці виконання.

Ітераційні метрики для кожного алгоритму зберігалися у вигляді окремих файлів у форматі CSV. Такий підхід дозволив виконати подальший аналіз поведінки алгоритмів у процесі пошуку та порівняти швидкість їх зближення до оптимального рішення. Для забезпечення об’єктивності всі вимірювання проводилися в однаковому програмному середовищі та на одній апаратній платформі.

Кожен експериментальний запуск виконувався одноразово для кожного сценарію, оскільки всі алгоритми є детермінованими та за однакових вхідних даних формують ідентичні результати при повторних запусках. Це дозволило зосередити увагу на аналізі відмінностей у роботі алгоритмів, а не на усередненні статистичних даних.

Обрана методика експерименту має забезпечити коректне та наочне порівняння алгоритмів пошуку шляху за різних масштабів задачі. Вона має створити підґрунтя для подальшого аналізу часових характеристик, обсягу виконаних обчислень та динаміки формування оптимального маршруту у кожному з розглянутих підходів.

3.2 Порівняння часових характеристик алгоритмів

Порівняння часових характеристик алгоритмів пошуку шляху є одним з ключових аспектів оцінювання їх ефективності, оскільки саме час виконання безпосередньо впливає на придатність алгоритму для використання в реальних навігаційних та інформаційних системах. У межах даного дослідження часові

показники аналізувалися для двох різних сценаріїв, що відрізнялися масштабом графа та складністю маршруту.

Перший експеримент відповідав задачі пошуку короткого маршруту в графі з кількістю вершин близько 200–250. Такий сценарій моделює локальні задачі навігації, наприклад прокладання маршруту в межах невеликого району або окремої частини міста. Результати виконання алгоритмів у цьому випадку наведено на рисунку 3.1.

Algorithm	Time (ms)	Visited
Dijkstra	0,21	39
AStar	2,85	35
BidirectionalDijkstra	4,14	28

Рисунок 3.1 – Результат швидкості виконання експерименту з малою кількістю вершин

Аналіз результатів для короткого маршруту показує, що всі три алгоритми демонструють дуже малий час виконання, який вимірюється частками мілісекунди. Це свідчить про те, що для невеликих графів обчислювальні витрати на пошук оптимального шляху є незначними незалежно від обраного алгоритму.

Класичний алгоритм Дейкстри у цьому сценарії показав найменший час виконання. Це пояснюється простотою його реалізації та відсутністю додаткових обчислень, пов’язаних із евристиками або підтримкою двох напрямків пошуку. За невеликої кількості вершин накладні витрати алгоритму є мінімальними, що забезпечує високу швидкодію.

Алгоритм A* у першому експерименті продемонстрував більший час виконання порівняно з алгоритмом Дейкстри. Незважаючи на використання евристичної функції, обчислення географічної відстані та додаткове оновлення F-оцінок створюють певні накладні витрати, які в умовах малого графа не компенсуються зменшенням області пошуку.

Двонапрямлений алгоритм Дейкстри у випадку короткого маршруту показав найбільший час виконання серед трьох алгоритмів. Це зумовлено тим, що переваги двонапрямленого підходу практично не проявляються на малих графах, тоді як необхідність підтримки двох наборів структур даних і виконання двох кроків пошуку на кожній ітерації збільшує загальні витрати часу.

Таким чином, результати першого експерименту свідчать про те, що для задач малої розмірності використання складніших алгоритмів не дає виграшу в часі, а класичний алгоритм Дейкстри є найбільш доцільним з точки зору швидкодії. Цей висновок добре узгоджується з теоретичними очікуваннями.

Другий експеримент був спрямований на аналіз часових характеристик алгоритмів у значно складніших умовах. У цьому сценарії розмір графа складав декілька тисяч вершин, що моделює пошук маршруту в межах великої міської або міжміської дорожньої мережі. Результати виконання алгоритмів для цього випадку наведено на рисунку 3.2.

Algorithm	Time (ms)	Visited
AStar	12,83	336
Dijkstra	34,17	351
BidirectionalDijkstra	39,82	76

Рисунок 3.2 – Результат швидкості виконання експерименту з великою кількістю вершин

У другому експерименті різниця у часі виконання між алгоритмами стає значно більш помітною. Це підтверджує тезу про те, що масштаб задачі є критичним фактором при виборі алгоритму пошуку шляху. Для великих графів обчислювальні витрати зростають, і ефективність алгоритмів починає істотно відрізнятися.

Алгоритм A* у даному сценарії продемонстрував найменший час виконання серед усіх розглянутих алгоритмів. Це свідчить про ефективність евристичного спрямування пошуку у випадку великих графів, коли правильний вибір евристики дозволяє значно зменшити кількість оброблених вершин і

прискорити досягнення цільової точки.

Час виконання алгоритму Дейкстри у другому експерименті суттєво зріс порівняно з першим сценарієм. Оскільки алгоритм не використовує жодної інформації про напрямок до цілі, він змушений рівномірно розширювати область пошуку, що призводить до значного збільшення кількості ітерацій та, відповідно, часу виконання.

Двонапрямлений алгоритм Дейкстри у другому експерименті продемонстрував менший час виконання порівняно з класичним алгоритмом Дейкстри, але все ж поступився алгоритму A*. Це пояснюється тим, що двонапрямлений підхід дійсно зменшує область пошуку, однак не використовує евристичну інформацію, яка дозволяє ще більш ефективно спрямовувати пошук.

Порівняння двох експериментів наочно демонструє, що відносна ефективність алгоритмів змінюється залежно від масштабу задачі. Якщо для коротких маршрутів різниця у часі виконання є незначною і часто визначається накладними витратами реалізації, то для великих маршрутів вирішальну роль відіграє стратегія пошуку.

Особливу увагу слід звернути на те, що у другому експерименті виграш алгоритму A* у часі є суттєвим порівняно з класичним алгоритмом Дейкстри. Це підтверджує доцільність використання евристичних алгоритмів у задачах маршрутизації з великим простором пошуку.

Водночас результати показують, що двонапрямлений алгоритм Дейкстри займає проміжне положення між класичним Дейкстрою та A*. Він демонструє кращі часові характеристики, ніж односторонній пошук, але поступається евристичному підходу за рахунок відсутності спрямованості до цілі.

Отримані часові показники також свідчать про те, що для малих графів вибір алгоритму має другорядне значення з точки зору продуктивності. У таких випадках простота реалізації та надійність можуть бути важливішими критеріями, ніж абсолютний час виконання.

Для великих графів ситуація є протилежною: навіть відносно невеликі оптимізації у виборі вершин для обробки можуть призводити до значного

скорочення часу виконання. Саме тому алгоритм A* демонструє найкращі результати у другому експерименті.

Таким чином, порівняння часових характеристик алгоритмів у двох сценаріях дозволяє зробити висновок про залежність ефективності алгоритмів від розмірності задачі. Класичний алгоритм Дейкстри є ефективним для малих графів, але погано масштабується.

Двонапрямлений алгоритм Дейкстри покращує часові характеристики для великих графів порівняно з класичним підходом, однак його потенціал обмежений відсутністю евристичного спрямування.

Алгоритм A* демонструє найкращу масштабованість і найменший час виконання у випадку великих маршрутів, що робить його найбільш придатним для практичного використання в навігаційних системах, які працюють з великими дорожніми мережами.

Отже, аналіз часових характеристик підтверджує теоретичні властивості розглянутих алгоритмів і наочно показує, як вибір стратегії пошуку впливає на продуктивність системи маршрутизації за різних умов експлуатації.

3.3 Аналіз кількості відвіданих вершин та опрацьованих ребер

Аналіз кількості відвіданих вершин та опрацьованих ребер є важливою складовою порівняльного дослідження алгоритмів пошуку шляху, оскільки ці показники безпосередньо відображають обсяг виконаних обчислень. На відміну від часу виконання, який залежить також від апаратних і програмних факторів, кількість відвіданих вершин і опрацьованих ребер дозволяє більш об’єктивно оцінити алгоритмічну ефективність кожного підходу.

У межах даного дослідження аналіз зазначених метрик було проведено для двох сценаріїв: короткого маршруту з кількістю вершин близько 200–250 та великого маршруту з кількома тисячами вершин. Це дозволяє простежити, як

змінюється поведінка алгоритмів залежно від масштабу задачі та складності графа.

Результати першого експерименту для короткого маршруту наведено на рисунку 3.3. У цьому випадку всі алгоритми демонструють відносно невелику кількість відвіданих вершин і опрацьованих ребер, що є очікуваним для графа малої розмірності.

Algorithm	Visited	Edges
Dijkstra	39	225
AStar	35	209
BidirectionalDijkstra	28	231

Рисунок 3.3 – Результат швидкості виконання експерименту з малою кількістю вершин

Алгоритм Дейкстри у першому сценарії відвідав 39 вершин і опрацював 225 ребер. Це відображає характерну для нього властивість рівномірного розширення області пошуку без урахування розташування цільової вершини. Навіть для короткого маршруту алгоритм змушений досліджувати певну кількість альтернативних напрямків.

Алгоритм A* у цьому ж сценарії продемонстрував дещо меншу кількість відвіданих вершин–35, а також меншу кількість опрацьованих ребер–209. Це свідчить про те, що навіть у випадку невеликого графа евристичне спрямування пошуку дозволяє зменшити кількість зайвих обчислень, хоча виграш є відносно невеликим.

Двонапрямлений алгоритм Дейкстри для короткого маршруту відвідав найменшу кількість вершин–28. Це пояснюється тим, що пошук виконується одночасно з двох напрямків, і області пошуку зустрічаються досить швидко. Водночас кількість опрацьованих ребер у цьому випадку є дещо більшою–231, що зумовлено необхідністю виконання релаксації в обох напрямках.

Таким чином, для короткого маршруту можна спостерігати, що мінімальна

кількість відвіданих вершин не завжди корелює з мінімальною кількістю опрацьованих ребер. Це підкреслює важливість аналізу обох метрик одночасно, а не ізольовано.

Другий експеримент, результати якого наведено на рисунку 3.4, демонструє принципово іншу картину. У випадку великого графа різниця між алгоритмами стає значно більш вираженою, що дозволяє чітко оцінити їх масштабованість.

Algorithm	Visited	Edges
AStar	336	13293
Dijkstra	351	13405
BidirectionalDijkstra	76	6853

Рисунок 3.4 – Результат швидкості виконання експерименту з великою кількістю вершин

Алгоритм Дейкстри у другому сценарії відвідав 351 вершину та опрацював 13 405 ребер. Така велика кількість операцій є наслідком рівномірного характеру пошуку, коли алгоритм поступово охоплює значну частину графа, незалежно від фактичного напрямку до цільової вершини.

Алгоритм A* у цьому ж сценарії відвідав 336 вершин і опрацював 13 293 ребра. Незважаючи на використання евристичної функції, кількісна різниця порівняно з алгоритмом Дейкстри виявляється відносно невеликою. Це свідчить про те, що в умовах складної дорожньої мережі евристика не завжди може кардинально обмежити простір пошуку, але все ж забезпечує певне скорочення обсягу обчислень.

Найбільш показові результати у другому експерименті демонструє двонапрямлений алгоритм Дейкстри. Кількість відвіданих вершин у цьому випадку становить лише 76, що є у декілька разів меншим значенням порівняно з двома іншими алгоритмами. Це підтверджує ефективність симетричного пошуку з двох напрямків у великих графах.

Кількість опрацьованих ребер для двонапрямленого алгоритму у другому сценарії також суттєво менша–6 853. Це майже вдвічі менше, ніж у алгоритмів Дейкстри та A*, і безпосередньо відображає зменшення області пошуку за рахунок ранньої зустрічі двох фронтів пошуку.

Порівняння результатів двох експериментів дозволяє зробити висновок, що масштаб задачі суттєво впливає на ефективність різних підходів. Для коротких маршрутів різниця у кількості відвіданих вершин і опрацьованих ребер є незначною, і вибір алгоритму має обмежений вплив на загальний обсяг обчислень.

Для великих маршрутів ситуація змінюється кардинально. Алгоритм Дейкстри демонструє різке зростання кількості опрацьованих елементів, що робить його менш придатним для використання у масштабних навігаційних задачах без додаткових оптимізацій.

Алгоритм A* частково компенсує цю проблему за рахунок евристичного спрямування, однак його ефективність значною мірою залежить від якості евристичної функції та структури графа. У розглянутому експерименті виграш у кількості операцій є помірним.

Двонапрямлений алгоритм Дейкстри демонструє найкращі показники за кількістю відвіданих вершин і опрацьованих ребер у великому графі. Це підтверджує теоретичні очікування щодо його здатності істотно скорочувати простір пошуку без використання евристичної інформації.

Водночас слід зазначити, що зменшення кількості відвіданих вершин досягається ціною більш складної реалізації та необхідності підтримки двох напрямків пошуку. Тому доцільність використання цього алгоритму залежить від вимог до продуктивності та складності програмної реалізації.

Отримані результати також пояснюють часові характеристики, проаналізовані у попередньому підрозділі. Зменшення кількості відвіданих вершин і опрацьованих ребер безпосередньо впливає на час виконання алгоритму, особливо у великих графах.

Таким чином, аналіз кількості відвіданих вершин та опрацьованих ребер

наочно демонструє принципові відмінності у роботі алгоритмів пошуку шляху. Класичний алгоритм Дейкстри є стабільним, але малоефективним у масштабних задачах, алгоритм A* забезпечує помірне скорочення обчислень за рахунок евристики, а двонапрямлений алгоритм Дейкстри показує найменший обсяг виконаних операцій у великих графах.

Отримані результати створюють основу для подальшого аналізу динаміки пошуку та якості зближення алгоритмів до оптимального рішення, що розглядається у наступних підрозділах порівняльного аналізу.

3.4 Порівняння динаміки пошуку за ітераціями

У даному підрозділі виконується порівняння динаміки роботи алгоритмів пошуку шляху в ітераційному розрізі. Аналіз зосереджено не лише на кінцевому результаті, а й на тому, як саме формується рішення в процесі виконання алгоритмів, тобто на характері зміни основних метрик від ітерації до ітерації. Такий підхід дозволяє оцінити поведінку алгоритмів на різних етапах пошуку та виявити відмінності у швидкості зближення до оптимального маршруту.

У межах цього підрозділу порівнюється робота трьох алгоритмів: алгоритму Дейкстри, двонапрямленого алгоритму Дейкстри та алгоритму A*. Порівняння проводиться для двох експериментальних сценаріїв, які відрізняються розміром графа: для короткого маршруту з відносно невеликою кількістю вершин та для великого маршруту, що містить декілька тисяч вершин. Це дає змогу простежити, як змінюється динаміка пошуку при зростанні масштабів задачі.

Для наочного аналізу результатів у даному розділі будується шість графіків. Для кожної з обраних ітераційних метрик формується по два графіки: один для малого графа, другий–для великого графа. На кожному з графіків одночасно відображено криві для всіх трьох алгоритмів, що забезпечує

безпосереднє візуальне порівняння їх поведінки.

Першою ітераційною метрикою є BestWeight. Вона характеризує найкращу з відомих на поточній ітерації сумарну вагу маршруту, знайденого алгоритмом. Зміна цього показника відображає моменти, коли алгоритм знаходить маршрут з меншою вартістю, та дозволяє оцінити, наскільки швидко відбувається наближення до оптимального рішення.

Другою метрикою є CurrentWeight, яка відображає вагу маршруту або вузла, що обробляється алгоритмом на конкретній ітерації. Цей показник характеризує поточний стан пошуку та демонструє, які значення ваг розглядаються алгоритмом у процесі дослідження графа. Аналіз цієї метрики дозволяє оцінити характер обходу графа та інтенсивність перегляду альтернативних шляхів.

Третьою метрикою є DifferenceFromOptimal, що показує різницю між поточним найкращим знайденим маршрутом та фінальним оптимальним значенням. Дана метрика є індикатором віддаленості алгоритму від оптимального рішення на кожному кроці виконання. Її динаміка дозволяє оцінити швидкість зближення алгоритму до оптимуму та стабільність процесу оптимізації.

Під час виконання експериментальних досліджень для кожного з обраних алгоритмів було здійснено детальний збір ітераційних даних. У результаті виконання алгоритмів для двох експериментальних сценаріїв–малого та великого графів–було сформовано шість CSV-файлів, по одному для кожної комбінації алгоритму та розміру графа. Таким чином, для алгоритму Дейкстри, двонапрямленого алгоритму Дейкстри та алгоритму A* було отримано по два файли, що відповідають різним масштабам задачі.

Кожен CSV-файл містить повну інформацію про перебіг роботи алгоритму в ітераційному розрізі. Дані фіксувалися на кожній ітерації виконання та дозволяють відтворити динаміку процесу пошуку шляху від початкового стану до знаходження оптимального маршруту. Такий підхід забезпечує можливість глибокого аналізу не лише кінцевого результату, але й проміжних станів

алгоритмів.

Структура сформованих CSV-файлів є уніфікованою для всіх алгоритмів і наведена на рисунку 3.5. Кожен запис у файлі відповідає окремій ітерації виконання алгоритму. Це дозволяє безпосередньо порівнювати значення метрик між різними алгоритмами на однакових етапах їх роботи.

Iteration	Time Ms	Current Weight	Best Weight	Difference Fron	Node Name
1	0.01	0			Start (24. Лозова. Харківська область. Україна. 64600)
2	0.03	52	2135	1359	R2.1 (48.8746. 36.2962)
3	0.04	68	2135	1359	R6.1 (48.8745. 36.2964)
4	0.04	115	2135	1359	R6.2 (48.8788. 36.2974)
5	0.04	125	1799	1023	R2.1 (48.8798. 36.2960)
6	0.05	187	1781	1005	R2.2 (48.8791. 36.2897)
7	0.05	207	1781	1005	R6.3 (48.8791. 36.2894)
8	0.05	284	1781	1005	R2.3 (48.8822. 36.2792)
9	0.06	415	1781	1005	R2.4 (48.8898. 36.2835)
10	0.06	446	1781	1005	R6.4 (48.8890. 36.2861)
11	0.06	565	1781	1005	R2.5 (48.8912. 36.3069)
12	0.07	669	1102	326	R2.6 (48.8906. 36.3141)
13	0.07	776	776	0	End (вулиця Соборна. 2. Лозова. Харківська область. Україна. 61000)

Рисунок 3.5 – Структура ітераційних файлів CSV

Поле Iteration містить порядковий номер ітерації, що відображає послідовність виконання алгоритму. Значення TimeMs фіксує накопичений час виконання на момент відповідної ітерації, що дозволяє співвідносити логічні кроки алгоритму з реальними часовими витратами. Поле CurrentWeight відображає вагу маршруту або вузла, який обробляється на поточній ітерації, характеризуючи поточний стан пошуку.

Поле BestWeight містить найкращу з відомих на цій ітерації вагу маршруту, знайденого алгоритмом, та використовується для аналізу процесу покращення рішення. Значення DifferenceFromOptimal показує різницю між поточним найкращим маршрутом і фінальним оптимальним значенням, що дає змогу оцінити ступінь наближення алгоритму до оптимального розв’язку на кожному кроці виконання.

Додатково у файлах присутнє поле NodeName, яке містить ідентифікаційну інформацію про вузол, що обробляється на відповідній ітерації,

включно з географічними координатами. Хоча це поле безпосередньо не використовується для побудови графіків, воно забезпечує трасування маршруту та можливість детального аналізу проходження алгоритму по графу.

На рисунках 3.6 та 3.7 наведено графіки ітераційної зміни метрики CurrentWeight для трьох алгоритмів пошуку шляху в умовах малого та великого експериментів відповідно. Дані графіки відображають вагу маршруту або вузла, який обробляється алгоритмом на кожній ітерації, що дозволяє простежити характер обходу графа та послідовність розгляду альтернативних шляхів.

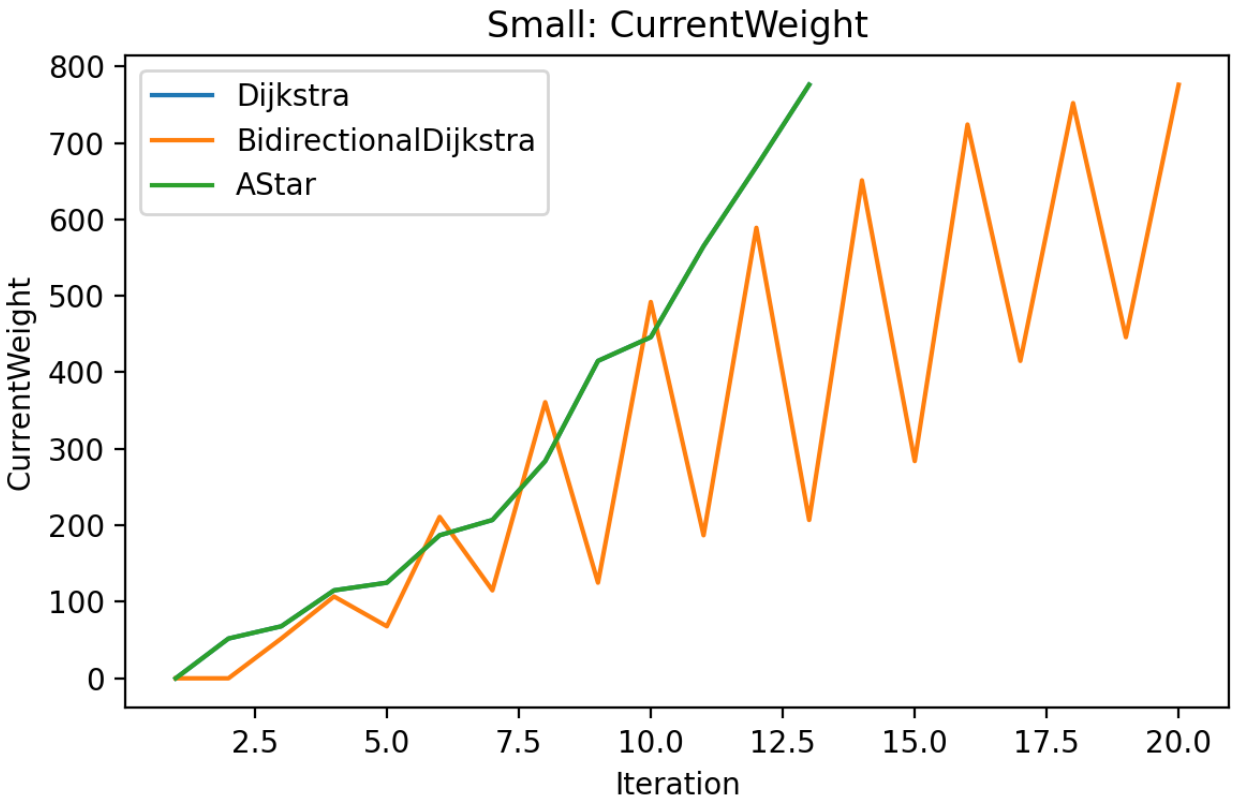

Рисунок 3.6 – Графік CurrentWeight для малого експерименту

У випадку малого експерименту (рисунок 3.6) спостерігається практично повне накладання кривих алгоритмів Дейкстри та A*. Лінія алгоритму Дейкстри на графіку візуально не відрізняється та фактично не сприймається окремо, оскільки вона повністю повторює траєкторію алгоритму A*. Це свідчить про те, що для графа невеликого розміру евристичне спрямування алгоритму A* не

призводить до істотної зміни порядку розгляду вершин у порівнянні з класичним алгоритмом Дейкстри, і обидва алгоритми проходять через однакові або дуже близькі за вагою стани.

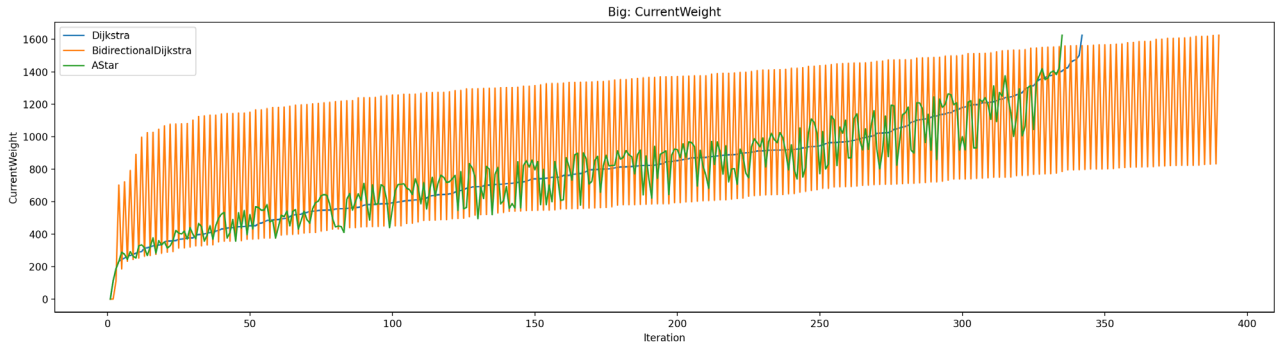

Рисунок 3.7 – Графік CurrentWeight для великого експерименту

При цьому двонапрямлений алгоритм Дейкстри у малому експерименті демонструє відмінну поведінку. Його крива має більш виражений ступінчастий характер із різкими коливаннями значень CurrentWeight, що зумовлено почерговим виконанням пошуку з початкової та кінцевої вершин. Такі коливання відображають зміну напрямку активного фронту пошуку та чергування вузлів, що розглядаються з різних сторін маршруту.

На рисунку 3.7, що відповідає великому експерименту, різниця між алгоритмами проявляється значно чіткіше. Алгоритм Дейкстри демонструє плавне, майже монотонне зростання значень CurrentWeight, що відповідає послідовному розширенню області пошуку без урахування напряму до цільової вершини. Така поведінка є типовою для класичного алгоритму та відображає його повний перебір вершин у порядку зростання відстані від початкової точки.

Алгоритм A* у великому експерименті характеризується менш рівномірною кривою з помітними коливаннями навколо загального зростаючого тренду. Це зумовлено впливом евристичної функції, яка спрямовує пошук у бік цілі, але водночас призводить до періодичного перегляду альтернативних маршрутів із різними значеннями оцінки. У результаті значення CurrentWeight змінюються більш динамічно, ніж у випадку класичного алгоритму Дейкстри.

Двонапрямлений алгоритм Дейкстри у великому експерименті демонструє найбільш виражені коливання метрики CurrentWeight. Часті піки та спади на графіку пов’язані з паралельним розширенням двох фронтів пошуку та переходами між ними. Така поведінка свідчить про інтенсивний перегляд вершин з обох напрямків та є характерною особливістю двонапрямленого підходу.

Загалом аналіз графіків CurrentWeight показує, що для задач невеликого розміру відмінності між алгоритмами є мінімальними, і їхня ітераційна поведінка практично збігається. Натомість зі збільшенням розміру графа різниця в динаміці пошуку стає суттєвою, що дозволяє чітко простежити вплив евристики в алгоритмі A* та двонапрямної стратегії в алгоритмі двонапрямленого Дейкстри.

На рисунках 3.8 та 3.9 наведено графіки ітераційної зміни метрики BestWeight для трьох алгоритмів пошуку шляху в умовах малого та великого експериментів відповідно. Дана метрика відображає найкращу з відомих на поточній ітерації сумарну вагу маршруту та є ключовим показником процесу наближення алгоритму до оптимального розв’язку.

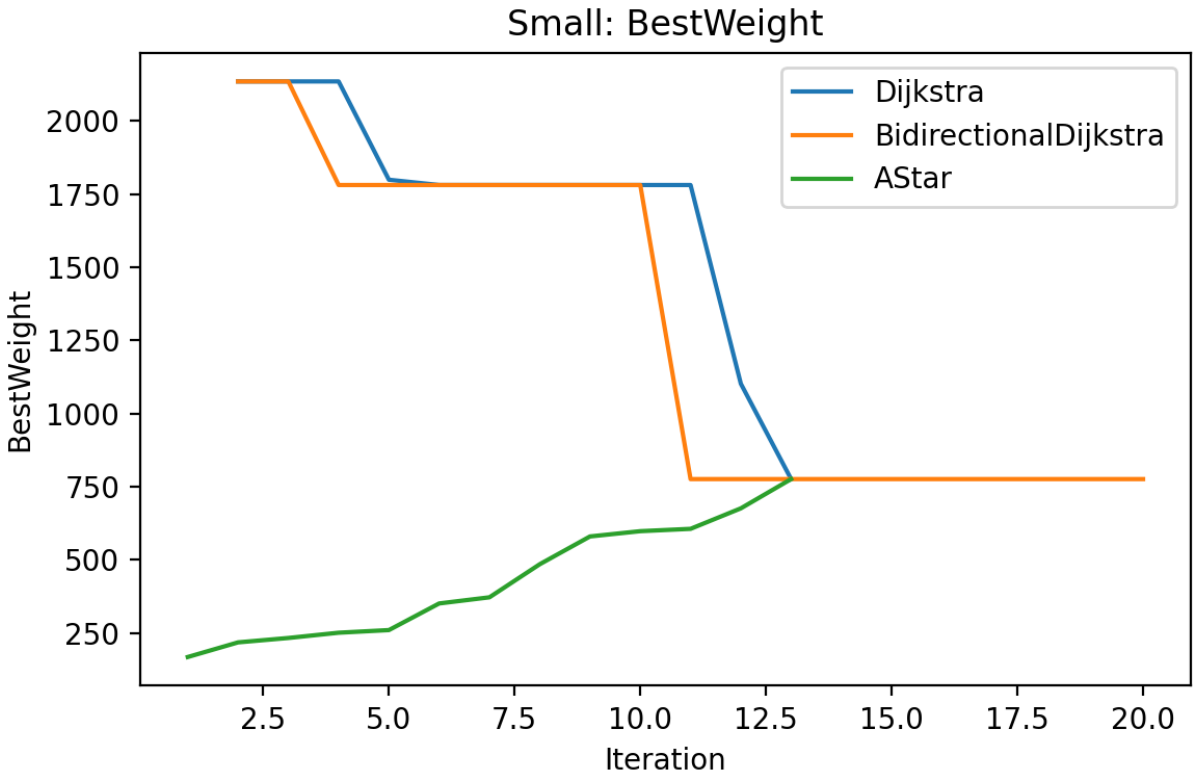

Рисунок 3.8 – Графік BestWeight для малого експерименту

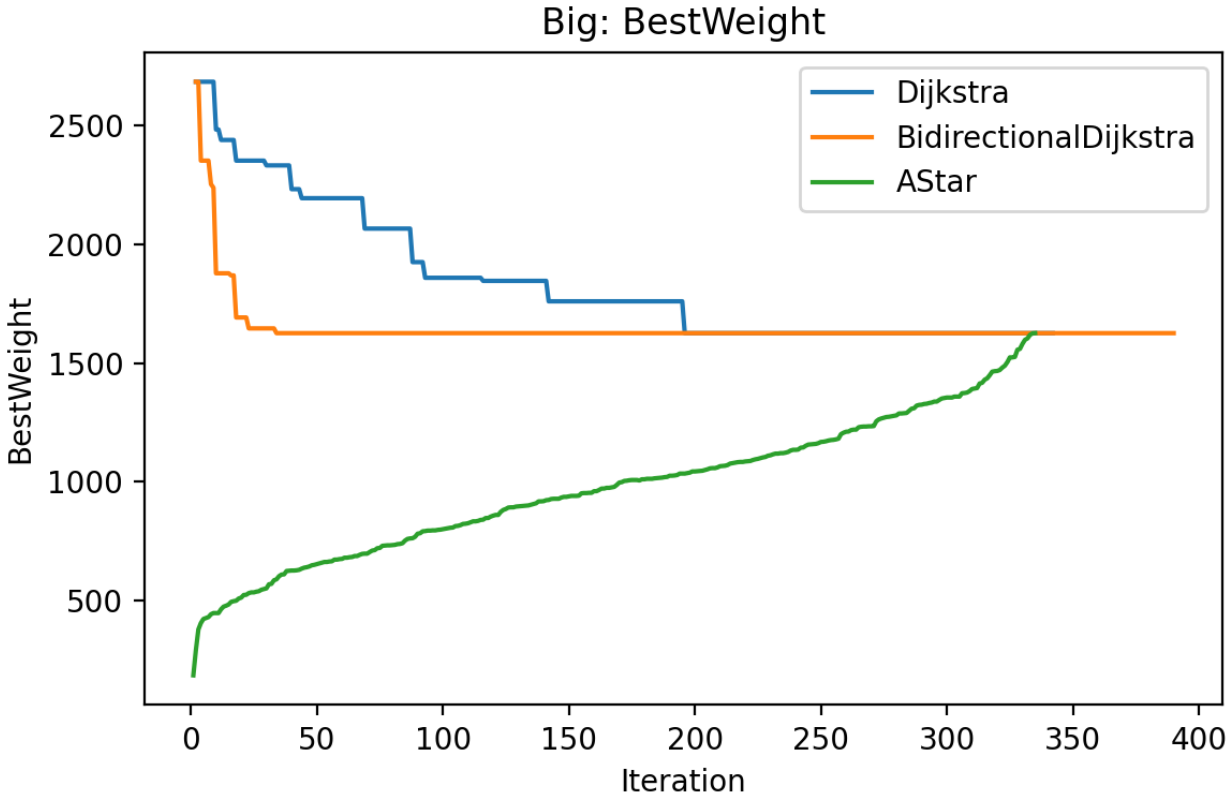

Рисунок 3.9 – Графік BestWeight для великого експерименту

У випадку малого експерименту (рисунок 3.8) чітко простежується ступінчастий характер кривих для алгоритмів Дейкстри та двонапрямленого алгоритму Дейкстри. На початкових ітераціях обидва алгоритми мають однакове або близьке значення BestWeight, що відповідає першому знайденому допустимому маршруту з відносно великою вагою. Подальші різкі зниження значень відображають моменти, коли алгоритми знаходять кращі альтернативні шляхи.

Для алгоритму Дейкстри зменшення значення BestWeight відбувається поступово та досягає оптимального значення лише на пізніших ітераціях. Це узгоджується з принципом його роботи, згідно з яким пошук рівномірно розширюється від початкової вершини без урахування напряму до цілі. Двонапрямлений алгоритм Дейкстри, навпаки, демонструє більш різке зниження BestWeight, що пояснюється зустріччю двох фронтів пошуку та швидшим виявленням маршруту з меншою сумарною вагою.

Алгоритм A* у малому експерименті характеризується принципово іншою поведінкою. Його крива BestWeight зростає поступово від невеликих значень до фінального оптимального рівня. Така форма графіка зумовлена особливістю реалізації метрики, де значення формується на основі евристичної оцінки та уточнюється в процесі пошуку. Попри інший характер зміни, фінальне значення BestWeight для всіх трьох алгоритмів збігається, що підтверджує коректність отриманих результатів.

На рисунку 3.9, що відповідає великому експерименту, відмінності між алгоритмами стають ще більш вираженими. Алгоритм Дейкстри демонструє класичну ступінчасту криву з численними етапами покращення рішення. Значення BestWeight зменшується поступово у міру розширення області пошуку, що свідчить про значні обчислювальні витрати до досягнення оптимального маршруту.

Двонапрямлений алгоритм Дейкстри у великому експерименті показує значно швидше зниження BestWeight на ранніх ітераціях. Після досягнення оптимального або близького до оптимального значення крива переходить у горизонтальний відрізок, що означає відсутність подальших покращень. Це підтверджує ефективність двонапрямленої стратегії для графів великого розміру.

Алгоритм A* у великому експерименті знову демонструє відмінну форму кривої. Значення BestWeight змінюється більш плавно та майже монотонно, що відображає поступове уточнення оцінки маршруту під впливом евристичної функції. Остаточне оптимальне значення досягається пізніше, ніж у двонапрямленого алгоритму Дейкстри, але при значно меншій кількості розглянутих альтернативних шляхів.

Загалом аналіз графіків BestWeight показує, що дана метрика чітко відображає моменти покращення розв’язку та дозволяє порівняти швидкість зближення алгоритмів до оптимального маршруту. Для малих графів різниця між алгоритмами є менш суттєвою, тоді як для великих графів переваги двонапрямленого алгоритму Дейкстри та евристичного підходу A* проявляються значно виразніше.

На рисунках 3.10 та 3.11 наведено графіки ітераційної зміни метрики DifferenceFromOptimal для трьох алгоритмів пошуку шляху в умовах малого та великого експериментів відповідно. Дана метрика відображає різницю між найкращою відомою на поточній ітерації вагою маршруту та фінальним оптимальним значенням, що дозволяє безпосередньо оцінити ступінь наближення алгоритму до оптимального розв'язку в процесі виконання.

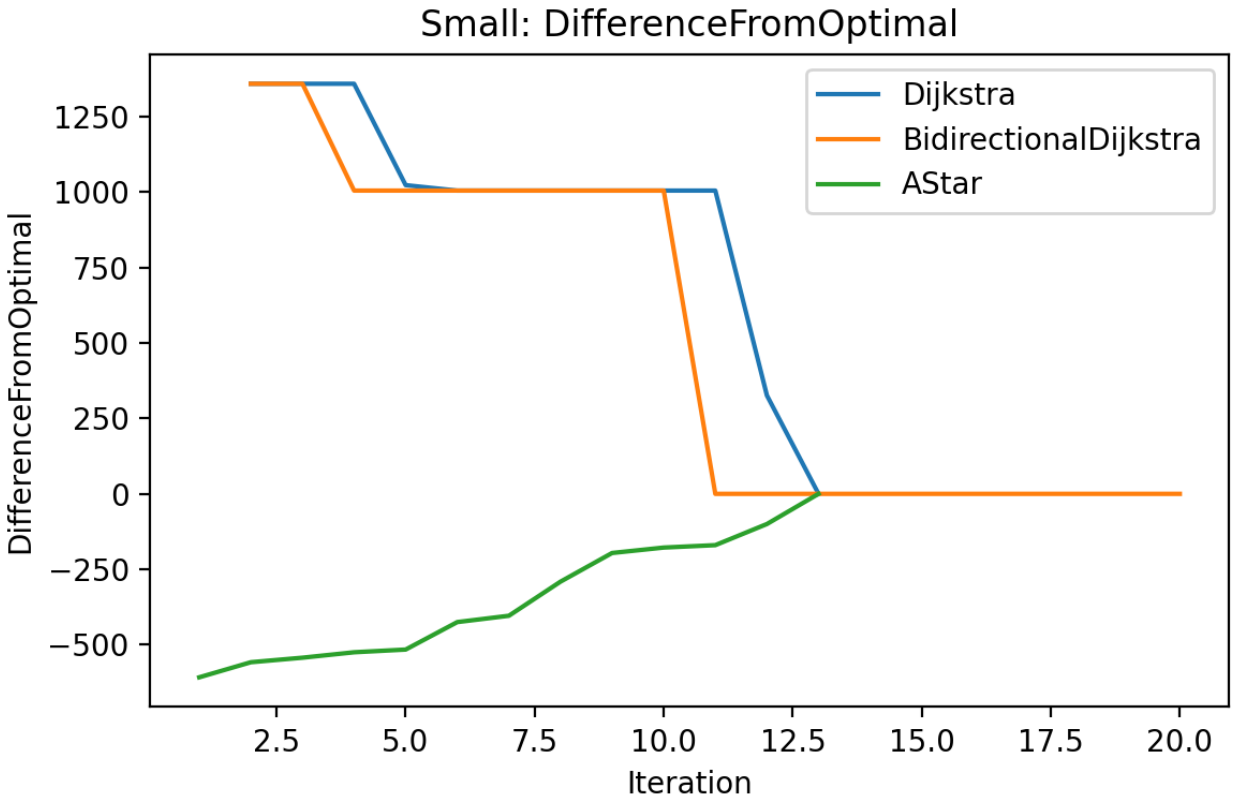

Рисунок 3.10 – Графік DifferenceFromOptimal для малого експерименту

У випадку малого експерименту (рисунок 3.10) для алгоритмів Дейкстри та двонапрямленого алгоритму Дейкстри спостерігається позитивне значення DifferenceFromOptimal на початкових ітераціях, що свідчить про значну віддаленість від оптимального маршруту. Значення метрики зменшується стрибкоподібно, відображаючи моменти, коли алгоритми знаходять кращі маршрути з меншою сумарною вагою. Для двонапрямленого алгоритму Дейкстри перехід до нульового значення відбувається раніше, ніж для

класичного алгоритму Дейкстри, що вказує на швидше досягнення оптимального розв’язку.

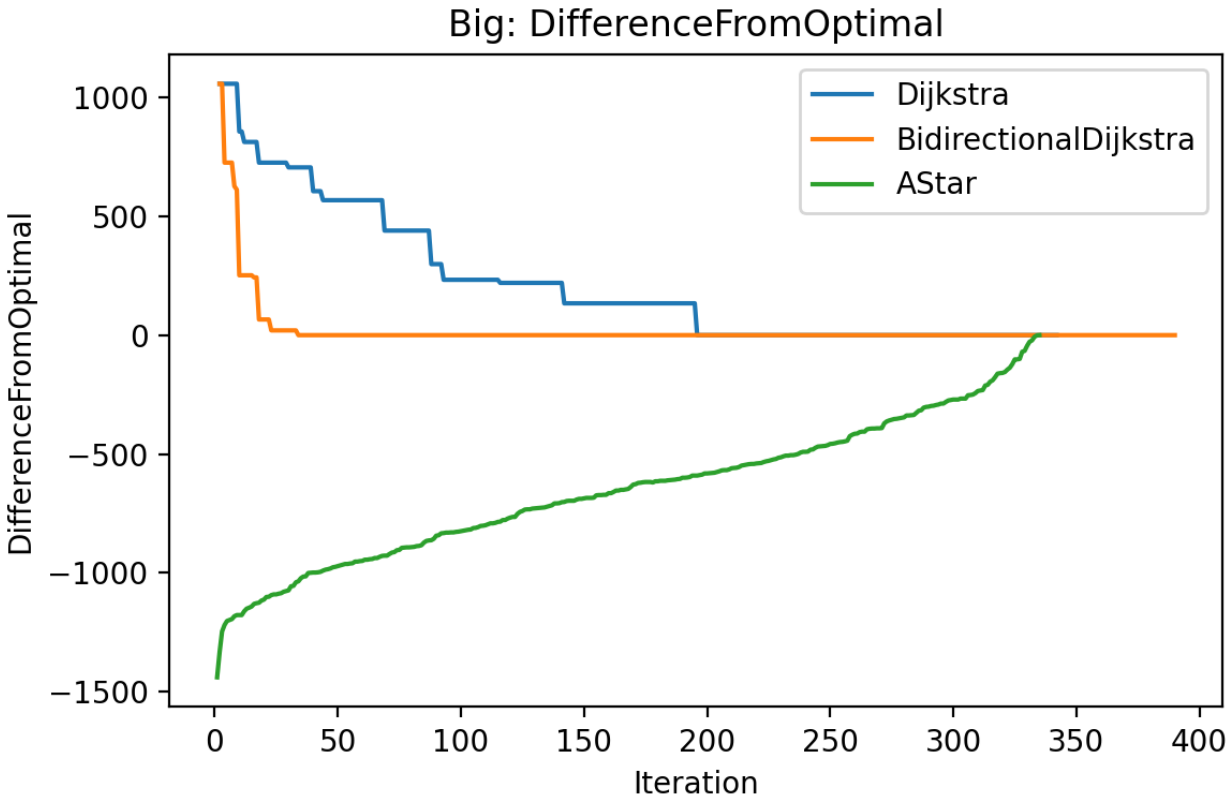

Рисунок 3.11 – Графік DifferenceFromOptimal для великого експерименту

Для алгоритму A* у малому експерименті метрика DifferenceFromOptimal має від’ємні значення на початкових ітераціях. Така поведінка зумовлена особливостями формування евристичної оцінки та способу обчислення різниці, коли поточна оцінка маршруту може бути меншою за фінальне оптимальне значення. У міру виконання алгоритму значення метрики монотонно зростає та наближається до нуля, що означає уточнення оцінки та збіжність до оптимального маршруту.

На рисунку 3.11, що відповідає великому експерименту, відмінності між алгоритмами проявляються значно чіткіше. Алгоритм Дейкстри демонструє поступове, ступінчасте зменшення значення DifferenceFromOptimal від великих додатних значень до нуля. Це свідчить про тривалий процес пошуку

оптимального маршруту та характерну для алгоритму Дейкстри поступову збіжність без використання евристичних обмежень.

Двонапрямлений алгоритм Дейкстри у великому експерименті показує різке зменшення DifferenceFromOptimal вже на ранніх ітераціях. Після досягнення нульового значення метрика залишається незмінною, що означає знаходження оптимального маршруту та відсутність подальших покращень. Така поведінка підтверджує ефективність двонапрямленого підходу для задач великого масштабу.

Алгоритм A* у великому експерименті знову характеризується від'ємними значеннями DifferenceFromOptimal на початкових етапах, після чого спостерігається плавне та майже монотонне наближення до нуля. Це відображає поступове уточнення евристичної оцінки маршруту та стабільну збіжність алгоритму до оптимального розв'язку. Досягнення нульового значення відбувається пізніше, ніж у двонапрямленого алгоритму Дейкстри, проте без різких стрибків і з більш передбачуваною динамікою.

Загалом аналіз графіків DifferenceFromOptimal дозволяє чітко оцінити швидкість зближення алгоритмів до оптимального маршруту. Для малих графів різниця між алгоритмами є відносно незначною, тоді як для великих графів двонапрямлений алгоритм Дейкстри демонструє найшвидше досягнення оптимуму, а алгоритм A*–найбільш плавний та стабільний процес збіжності. Алгоритм Дейкстри, у свою чергу, характеризується повільнішою, але гарантовано коректною збіжністю до оптимального рішення.

Дослідження показало, що для графів невеликого розміру відмінності в ітераційній поведінці алгоритмів є мінімальними. Зокрема, алгоритми Дейкстри та A* демонструють практично ідентичну динаміку за метрикою CurrentWeight, що свідчить про обмежений вплив евристики в умовах малої кількості вершин. У таких сценаріях усі алгоритми досить швидко доходять до оптимального рішення, а їх внутрішня логіка пошуку істотно не відрізняється.

Для графів великого розміру різниця між алгоритмами проявляється значно чіткіше. Алгоритм Дейкстри характеризується поступовою та повільною

збіжністю до оптимального рішення, що відображається у ступінчастій зміні метрик BestWeight та DifferenceFromOptimal. Це підтверджує його повний перебір простору пошуку без використання додаткових обмежень або напрямних механізмів.

Двонапрямлений алгоритм Дейкстри продемонстрував найшвидше досягнення оптимального маршруту за метриками BestWeight та DifferenceFromOptimal, особливо у великому експерименті. Раннє зменшення різниці до нульового значення свідчить про ефективність двонапрямленого підходу, який дозволяє значно скоротити область пошуку за рахунок зустрічі двох фронтів.

Алгоритм A* відрізняється найбільш плавною та стабільною динамікою збіжності. Хоча досягнення оптимального значення за метрикою DifferenceFromOptimal відбувається пізніше, ніж у двонапрямленого алгоритму Дейкстри, процес пошуку характеризується меншою кількістю різких змін та більш передбачуваною поведінкою. Це підтверджує ефективність евристичного спрямування пошуку, особливо в умовах великих графів.

Загалом результати порівняльного аналізу свідчать, що жоден з алгоритмів не є універсально найкращим для всіх сценаріїв. Алгоритм Дейкстри забезпечує стабільну та гарантовано оптимальну збіжність, але поступається за швидкістю. Двонапрямлений алгоритм Дейкстри демонструє найкращі показники швидкості зближення до оптимального рішення на великих графах. Алгоритм A* забезпечує збалансоване поєднання керованості процесу пошуку та ефективності, що робить його доцільним для практичних задач маршрутизації з використанням географічних даних.

3.5 Інтерпретація результатів

У даному розділі здійснюється узагальнення результатів

експериментальних досліджень та інтерпретація отриманих даних з позиції практичного застосування алгоритмів пошуку шляху. Аналіз ітераційної динаміки, часових характеристик та кількісних показників дозволяє сформувати рекомендації щодо доцільності використання кожного з розглянутих алгоритмів залежно від умов задачі маршрутизації.

Отримані експериментальні результати показали, що ефективність алгоритмів суттєво залежить від масштабу задачі та структури графа. Аналіз часових характеристик засвідчив, що алгоритм Дейкстри продемонстрував найменший час виконання у випадку малого експерименту, що пояснюється відносно невеликою кількістю вершин і ребер. За таких умов додаткові механізми оптимізації, зокрема евристика в алгоритмі A*, не дають суттєвої переваги, а простота класичного підходу забезпечує мінімальні накладні витрати.

У випадку великого експерименту найкращі часові показники продемонстрував алгоритм A*. Використання евристичної функції дозволило суттєво обмежити область пошуку та зосередити обчислення в напрямку цільової вершини, що призвело до зменшення часу виконання порівняно з іншими алгоритмами. Це підтверджує доцільність застосування алгоритму A* для задач маршрутизації великого масштабу, зокрема при роботі з дорожніми графами, що містять тисячі вершин.

Водночас двонапрямлений алгоритм Дейкстри виявився найповільнішим у обох експериментах. Незважаючи на теоретичні переваги двонапрямного підходу, практична реалізація зумовила значні накладні витрати, пов’язані з підтримкою двох фронтів пошуку, додатковими перевірками умов зустрічі та ускладненим керуванням структурами даних. У результаті ці витрати перевищили потенційний виграш від скорочення області пошуку.

Таким чином, результати дослідження свідчать, що двонапрямлений алгоритм Дейкстри не завжди забезпечує покращення часових характеристик у практичних реалізаціях, особливо у порівнянні з ефективно реалізованим алгоритмом A*. Це підкреслює важливість не лише теоретичного аналізу

алгоритмів, а й оцінки їх реальної поведінки в конкретних умовах експлуатації.

Загалом можна зробити висновок, що алгоритм Дейкстри є доцільним для задач малого масштабу, алгоритм A*–для задач великого масштабу, тоді як двонапрямлений алгоритм Дейкстри потребує додаткової оптимізації або спеціалізованих умов застосування, щоб виправдати свою складність у практичних сценаріях.

Результати показують, що алгоритм A* особливо ефективний у задачах з великими графами, коли необхідно швидко отримати прийнятне або близьке до оптимального рішення. Його застосування є доцільним у системах навігації, сервісах побудови маршрутів та інтерактивних додатках, де час відгуку є критичним параметром.

Важливим аспектом інтерпретації результатів є вплив вибору евристичної функції на ефективність алгоритму A*. Коректно підібрана евристика забезпечує оптимальність результату та істотно скорочує область пошуку. Натомість некоректна або надто груба евристична оцінка може знизити ефективність алгоритму або призвести до небажаних обчислювальних витрат.

Порівняльний аналіз також показав, що для задач малого масштабу різниця між алгоритмами є незначною, і вибір конкретного методу може базуватися на простоті реалізації або зручності інтеграції в існуючу систему. У таких умовах класичний алгоритм Дейкстри або A* можуть бути рівноцінними з практичної точки зору.

Для задач великого масштабу, зокрема при роботі з дорожніми графами, що містять тисячі вершин, доцільним є використання алгоритмів з додатковими механізмами оптимізації пошуку. У таких сценаріях алгоритм A* демонструє суттєву перевагу над класичним підходом.

ВИСНОВКИ

У межах даної роботи було виконано комплексне дослідження задачі оптимізації побудови маршрутів у графових структурах з використанням сучасних підходів до пошуку найкоротшого шляху. Робота охоплює як теоретичний аналіз, так і практичну реалізацію алгоритмів, що дозволило всебічно розглянути проблему та оцінити ефективність різних методів її розв’язання.

На початковому етапі було здійснено аналіз предметної області, пов’язаної з задачами маршрутизації та пошуку оптимальних шляхів у графах. Розглянуто особливості дорожніх графів, їхню структуру та специфіку застосування в геоінформаційних системах.

У роботі визначено актуальність дослідження, яка зумовлена зростанням обсягів просторових даних, ускладненням дорожніх мереж та підвищеними вимогами до швидкодії систем навігації. Обґрунтовано необхідність пошуку ефективних алгоритмічних рішень, здатних забезпечити швидке та коректне формування маршрутів у реальному часі.

Далі було здійснено детальну характеристику обраних алгоритмів пошуку шляху, а саме алгоритму Дейкстри, двонапрямленого алгоритму Дейкстри та алгоритму A*. Для кожного з них розглянуто основні принципи роботи, сильні та слабкі сторони, а також умови, за яких їх використання є доцільним.

Окрему увагу приділено формалізації алгоритмів. Було визначено основні логічні етапи їх роботи, сформульовано відповідні математичні співвідношення та побудовано блок-схеми, що наочно відображають процес пошуку найкоротшого шляху. Такий підхід дозволив забезпечити чітке розуміння внутрішньої логіки кожного алгоритму та підготувати їх до практичної реалізації.

Практична частина роботи полягала в реалізації обраних алгоритмів із використанням платформи .NET. Реалізація включала розробку окремих сервісів

для кожного алгоритму, організацію структур даних для зберігання графа, а також механізмів збору детальних метрик виконання. Особливу увагу приділено уніфікації підходів до реалізації, що забезпечило коректність подальшого порівняльного аналізу.

У процесі виконання алгоритмів було реалізовано збір ітераційних даних, на основі яких сформовано CSV-файли з детальними логами роботи алгоритмів.

Подальший аналіз результатів було проведено за низкою метрик, зокрема часом виконання, кількістю ітерацій, кількістю відвіданих вершин та опрацьованих ребер, а також ітераційними метриками.

Порівняльний аналіз часових характеристик показав, що алгоритм Дейкстри продемонстрував найкращі результати у випадку малого експерименту, що пояснюється відносно невеликим розміром графа та мінімальними накладними витратами на обчислення.

У випадку великого експерименту найшвидшим виявився алгоритм A*. Використання евристичної функції дозволило істотно обмежити область пошуку та спрямувати обчислення в бік цільової вершини, що призвело до зменшення часу виконання порівняно з іншими алгоритмами.

Інтерпретація отриманих результатів дозволила сформувати практичні рекомендації щодо вибору алгоритму залежно від умов задачі.

У підсумку виконана робота досягла поставленої мети та продемонструвала можливість ефективного дослідження й оптимізації задачі побудови маршрутів шляхом поєднання теоретичного аналізу та практичної реалізації. Отримані результати можуть бути використані як основа для подальших досліджень, зокрема для розширення набору алгоритмів, удосконалення евристичних функцій та інтеграції більш складних моделей дорожнього руху.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Нагорний І. А. Дослідження алгоритмів пошуку шляхів у задачі формування графіку маршрутів автобусів із використанням Google Maps. Кваліфікаційна робота бакалавра. – Харків, ХНУРЕ, 2024.

2. Dijkstra, E. W. A Note on Two Problems in Connexion with Graphs // Numerische Mathematik. – 1959. – Режим доступу: https://doi.org/10.1007/BF01386390 (дата звернення: 18.11.2025).

3. Hart, P. E., Nilsson, N. J., Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths // IEEE Transactions on Systems Science and Cybernetics. – 1968. – Режим доступу: https://ieeexplore.ieee.org/document/4082128 (дата звернення: 18.11.2025).

4. Goldberg, A., Harrelson, C. Computing the Shortest Path: A* Search Meets Graph Theory // Proceedings of the 16th Annual ACM–SIAM Symposium. – 2005. – Режим доступу: https://dl.acm.org/doi/10.5555/1070432.1070455 (дата звернення: 19.11.2025).

5. Pijls, W., Post, H. Yet Another Bidirectional Algorithm for Shortest Paths // Proceedings of the 8th International Symposium on Experimental Algorithms. – 2009. – Режим доступу: https://link.springer.com/chapter/10.1007/978-3-642-02011-7_30 (дата звернення: 19.11.2025).

6. Google Maps Platform Documentation. – Google Developers, 2025. – Режим доступу: https://developers.google.com/maps/documentation (дата звернення: 19.11.2025).

7. Mehlhorn, K., Sanders, P. Algorithms and Data Structures: The Basic Toolbox. – Springer, 2020. – Режим доступу: https://link.springer.com/book/10.1007/978-3-540-77978-0 (дата звернення: 20.11.2025).

8. Кормен, Т., Лейзерсон, Ч., Рівест, Р., Штайн, К. Алгоритми. Побудова і аналіз : навч. посіб. – К.: Вільямс, 2021. – Режим доступу: https://library.kh.ua (дата звернення: 20.11.2025).

9. Саметов, Р. В. Методи оптимізації та пошуку найкоротших шляхів у графах // Вісник КНУ. – 2022. – Режим доступу: https://periodicals.knu.ua (дата звернення: 20.11.2025).

10. Zhan, F. Three Fastest Shortest Path Algorithms on Real Road Networks // Journal of Geographic Information and Decision Analysis. – 1997. – Режим доступу: http://www.geoinfojournal.com (дата звернення: 20.11.2025).

11. Ahuja, R. K., Magnanti, T. L., Orlin, J. B. Network Flows: Theory, Algorithms, and Applications. – Prentice Hall, 1993. – Режим доступу: https://web.mit.edu/orlin/www/book.pdf (дата звернення: 18.11.2025).

12. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. – MIT Press, 2022. – Режим доступу: https://mitpress.mit.edu/9780262046305 (дата звернення: 18.11.2025).

13. Pohl, I. Bi-Directional Search // Machine Intelligence. – 1971. – Режим доступу: https://www.sciencedirect.com/science/article/pii/B9780444008129500150 (дата звернення: 18.11.2025).

14. Geisberger, R., Sanders, P., Schultes, D., Delling, D. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks // Experimental Algorithms. – 2008. – Режим доступу: https://link.springer.com/chapter/10.1007/978-3-540-68552-4_24 (дата звернення: 19.11.2025).

15. Delling, D., Goldberg, A. V., Werneck, R. F. Faster Batched Shortest Paths in Road Networks // Journal of Experimental Algorithmics. – 2011. – Режим доступу: https://dl.acm.org/doi/10.1145/1963190.1963194 (дата звернення: 19.11.2025).

16. Bast, H., Funke, S., Sanders, P., Schultes, D. Fast Routing in Road Networks with Transit Nodes // Science. – 2007. – Режим доступу: https://www.science.org/doi/10.1126/science.1139851 (дата звернення: 19.11.2025).

17. Hansen, E. A., Zhou, R. Anytime Heuristic Search // Journal of Artificial Intelligence Research. – 2007. – Режим доступу: https://www.jair.org/index.php/jair/article/view/10318 (дата звернення: 20.11.2025).

18. Zhou, R., Hansen, E. A. Breadth-First Heuristic Search // Artificial Intelligence. – 2006. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S0004370206000481 (дата звернення: 20.11.2025).

19. LaValle, S. M. Planning Algorithms. – Cambridge University Press, 2006. – Режим доступу: http://planning.cs.uiuc.edu (дата звернення: 20.11.2025).

20. Chen, Z., Guo, L., Liu, Y. Efficient Path Planning Algorithms for Large-Scale Road Networks // IEEE Access. – 2021. – Режим доступу: https://ieeexplore.ieee.org/document/9352943 (дата звернення: 20.11.2025).