ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

АНАЛІЗ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗНАХОДЖЕННЯ НАЙКОРОТШОГО ШЛЯХУ ГРАФА

зі спеціальності 122 Комп'ютерні науки освітня
програма «Комп'ютерні науки» ступеня магістра

Виконавець роботи Коцюба Родіон Олегович
_____«____»_____ 202_ р.
(підпис)

Науковий керівник к. ф.-м. н., доцент, Парфьонова Тетяна Олександрівна
_____«____»_____ 202_ р.
(підпис)

Рецензент

ПОЛТАВА 202_

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
CurrentNode	Це змінна, яка використовується в алгоритмі Дейкстри для представлення поточної вершини, з якої алгоритм розглядає можливі ребра до сусідніх вершин.
For Each	Використовується для ітерації (перебору) через елементи колекції, такі як масиви або колекції. У кожній ітерації цикл присвоює змінній значення наступного елемента в колекції і виконує блок коду.
List	Є динамічною структурою даних, що представляє собою список об'єктів. Вона дозволяє зберігати колекцію об'єктів одного типу та додавати, видаляти або змінювати елементи в будь-якому місці списку.
While	Ключове слово у мові програмування Visual Basic, яке використовується для виконання блоку коду, допоки вираз, що перевіряється, є істинним.

ВСТУП

У сучасному світі, де швидко розвиваються технології, методи програмної реалізації знаходження найкоротшого шляху в графах стають важливою складовою прогресу. Ці методи мають широкі застосування в різних галузях, починаючи від транспорту та логістики і закінчуючи інформаційними системами та соціальними мережами.

Одним із найважливіших завдань в області обчислювальної науки є знаходження найкоротшого шляху у графі. Кваліфікаційна робота виконується згідно вимог.[1] Розробка програмного забезпечення для знаходження найкоротшого шляху в графах вирішує реальні завдання, такі як аналіз складних мереж і знаходження оптимальних маршрутів для пересування об'єктів або інформації. Від розробки маршрутів доставки до пошукових алгоритмів у мережах, ці методи грають ключову роль у вирішенні різноманітних завдань у сучасному світі. Завдяки постійному розвитку комп'ютерних технологій, програмні реалізації цих методів стають все швидшими, точнішими та потужнішими. Використання оптимізованих алгоритмів дозволяє обробляти величезні об'єми даних у реальному часі, що стає ключовим для розвитку сучасних інформаційних технологій та наукових досліджень. [2]

Мета кваліфікаційної роботи – порівняти, проаналізувати, реалізувати програмне забезпечення методів знаходження найкоротшого шляху графа.

Об’єкт розробки – методи знаходження найкоротшого шляху графа, їх програмна реалізація.

Предмет розробки – створення програмного забезпечення алгоритмів Дейкстри, Беллмана-Форда, Евристичного алгориму.

Актуальність роботи – алгоритми для знаходження найкоротшого шляху в графах є надзвичайно актуальними в наш час через, їх широке застосування в різних сферах. Наприклад, системи GPS і картографічні додатки використовують ці алгоритми для визначення оптимальних

маршрутів між точками, що є критично важливим для транспортних засобів, логістики та служб доставки, адже це дозволяє мінімізувати час і витрати на пересування. В соціальних мережах і пошукових системах вони допомагають знаходити найкоротші зв’язки між користувачами або веб-сторінками, покращуючи якість пошуку та рекомендацій. Алгоритм Дейкстри шукає найкоротший шлях від однієї вершини до всіх інших у графах з негативними вагами, використовуючи жадібний підхід. Алгоритм Беллмана-Форда дозволяє знаходити найкоротші шляхи у графах з від’ємними вагами, перевіряючи всі ребра на кожній ітерації, що робить його повільнішим, але універсальнішим.

Програмне забезпечення для знаходження найкоротшого шляху в графі, реалізоване мовою програмування C#, передбачає використання алгоритмічних методів для вирішення задач теорії графів.

Методи дослідження – при дослідженні та реалізації алгоритму для знаходження найкоротшого шляху в графах використовуються різні програмні методи та підходи. Одним із основних методів є імплементація алгоритмів, таких як алгоритм Дейкстри для графів з невід'ємними вагами ребер.

Головне завдання – створення програмного продукту, що реалізує методи знаходження найкоротшого шляху графа. Кваліфікаційна робота складається з чотирьох розділів. У першому розділі описано постановку задачі для реалізації програмного забезпечення. У другому розділі здійснюється огляд алгоритмів пошуку найкоротших шляхів. Третій розділ містить опис алгоритмів методів, які реалізуються відповідно до теми, детально описаний алгоритм роботи програми та блок-схеми. У четвертому розділі описано процес програмної реалізації програмного забезпечення та інструкція користувача. Обсяг пояснювальної записки: 57 стор., основна частина – 54 стор., джерела – 20 назв.

1. ПОСТАНОВКА ЗАДАЧІ

Головна мета кваліфікаційної роботи – створення програмного забезпечення і порівняльний аналіз методів знаходження найкоротшого шляху графа та їх ефективності.

Наступні завдання розробки:

1. Ознайомлення з теоретичним матеріалом, на тему методів знаходження найкоротшого шляху графа.

2. Розробка алгоритму функціонування програмного забезпечення.

3. Створення блок-схеми програмного забезпечення.

4. Програмна реалізація програми.

5. Опис результатів програмної реалізації програмного продукту.

Список завдань для реалізації проекту:

1. Визначення вимог:

- Описати, які дані про граф необхідно вводити.
- Вибрати алгоритм для знаходження найкоротшого шляху.

2. Проектування:

- Спроектувати структуру даних для зберігання графа.
- Визначити клас або структури для представлення графа, вузлів та ребер.

3. Реалізація:

- Створити додаток з графічним інтерфейсом на C#.
- Реалізувати клас для представлення графа.
- Реалізувати метод для додавання вузлів та ребер.
- Реалізувати введення даних графа користувачем через текстові поля та кнопки.
- Реалізувати алгоритм Дейкстри для знаходження найкоротшого шляху.

4. Введення та виведення даних:

- Реалізувати методи для введення даних про граф з текстових полів.

- Реалізувати методи для виведення результату у текстове поле.

5. Тестування:

- Перевірити роботу додатку з різними вхідними даними.

6. Оптимізація та відлагодження:

- Оптимізувати код для підвищення ефективності роботи алгоритму.

- Відлагодити програму для усунення можливих помилок.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд методів пошуку найкоротших шляхів графа

Задача пошуку найкоротшого шляху є однією з основних у теорії графів. Вона полягає у визначенні найкоротшого маршруту між двома вершинами графа таким чином, щоб загальна сума ваг ребер на цьому шляху була мінімальною. Ця задача, також відома як задача мінімального шляху, є ключовою та класичною проблемою в теорії графів. Її практична значущість полягає в тому, що вона виступає основою для розв’язання складніших задач та слугує моделлю для створення ефективних алгоритмів у графовій теорії.

Важливість розв'язання цієї задачі пояснюється її численними практичними застосуваннями. Наприклад, у GPS-навігаторах знаходиться найкоротший маршрут між двома перехрестями: перехрестя виступають вершинами графа, а дороги – ребрами, що їх з'єднують. Якщо сума довжин доріг є мінімальною, то знайдений шлях вважається найкоротшим.

Варто зазначити, що на практиці ця задача застосовується не лише для обчислення довжини маршруту, але й для розв’язання фінансових, матеріальних та паливно-енергетичних завдань. Зокрема, в економічних задачах довжина ребра може відображати такі параметри, як час, вартість або кількість використаних ресурсів. При проходженні маршруту ці характеристики додаються аналогічно до довжин у класичному обчисленні загального шляху. Це дає змогу мінімізувати вищевказані показники. Теоретично цілком допустиме дослідження графів із ребрами, у яких значення ваг є від’ємними. Однак задачі такого типу, як правило, не мають простого економічного змісту, і складно вирішуються. Задача про найкоротший шлях має рішення для будь-якого зв’язного графа. При цьому рішенням задачі вважається найкоротший шлях і його довжина. Зрозуміло, що довжина найкоротшого шляху визначається однозначно, в той час як найкоротших шляхів може бути декілька. Рішення задачі про

найкоротший шлях може бути проведено двома способами: на основі алгоритму повного перебору усіх можливих шляхів або методом динамічного програмування на графі.

При розв’язанні задачі про найкоротший шлях за допомогою алгоритму повного перебору будуються всі можливі маршрути між початковою та кінцевою точками, після чого обчислюються довжини всіх отриманих шляхів, і вибирається шлях із мінімальною довжиною. Обраний шлях (або кілька шляхів) вважається розв’язком задачі.

У процесі перебору можна застосовувати певні оптимізації, що дозволяють відкинути явно невигідні варіанти. Наприклад, якщо довжина поточного шляху вже перевищує мінімальну довжину знайдених раніше маршрутів, подальші обчислення для цього варіанта можна припинити. Однак основним недоліком такого підходу є його низька ефективність і велика обчислювальна складність для графів зі значною кількістю вершин і ребер.

На відміну від цього, метод динамічного програмування базується на розбитті складної задачі на простіші підзадачі. Якщо вихідну задачу можна розв’язати оптимально, розділяючи її на підзадачі й рекурсивно визначаючи їх оптимальні рішення, то така задача має оптимальну підструктуру, що робить її придатною для динамічного програмування.1959 р. Дейкстри запропонував алгоритм пошуку у графі, який можна використати для вирішення задачі знаходження найкоротшого шляху між двома вершинами для будь-якого графа з додатними вагами. Цей алгоритм пізніше був модифікований Лі в 2006 році і застосований до системи наведення автомобіля. Він має дві модифікації, а саме: найкоротший шлях та найшвидший алгоритм. Алгоритм визначення найкоротшого шляху обчислює найкоротший маршрут між кожною парою вершин графа, в той час як алгоритм найшвидшого шляху визначає шлях з мінімальним часом його проходження. Майбутній час проходження шляху можна передбачити на основі моделей прогнозування, використовуючи

дані про час проходження маршруту розглянув алгоритм Дейкстри та алгоритм Беллмана-Форда для пошуку найкоротшого шляху в графі. Прикладом може бути платформа, де користувачі виконують запити, щоб знайти найкоротший шлях до «соціальних посилань», яка пов’язує їх із певним користувачем. Цей тип запитів є складним завданням для графів середнього розміру, і стає обчислювально нездійсненним для графів, які моделюють сучасні соціальні мережі, більшість з яких містять мільйони вузлів і мільярди ребер. Автори пропонують про-граму Atlas – новий підхід до масштабованих наближених найкоротших шляхів між вузлами графа, використовуючи колекцію охоплюючих дерев.

Описано методи поступового оновлення, що враховують динамічні зміни соціальних графів з часом. Для фіксації динаміки графів було використано 35 щоденних знімків мережі Facebook, що дозволило показати здатність Atlas амортизувати вартість оновлення дерев у довгостроковій перспективі. Atlas було також застосовано до низки графічних додатків, де отримані результати виявилися близькими до ідеальних.

Дослідники вирішили задачу проєктування мережі, відому як проблема кабелю та траншеї, яка балансує між витратами на використання мережі та капітальними витратами на її будівництво. У цій задачі дерево найкоротшого шляху може призвести до вищих витрат на будівництво, але при цьому зменшити експлуатаційні витрати завдяки оптимальнішим маршрутам для відправлення й призначення. З іншого боку, мінімальне охоплююче дерево є менш витратним для будівництва, але може збільшити витрати на використання. Для розв'язання цієї задачі було запропоновано евристичний підхід, який забезпечує оптимальні або близькі до оптимальних рішення.

Окрім цього, Pallottino і Scutella детально описали алгоритми пошуку найкоротшого шляху в транспортних моделях, розглянувши як класичні, так і інноваційні аспекти цієї проблеми. Вони розглянули алгоритми найкоротшого шляху при перевезеннях у двох аспектах. Перший – це

класичні алгоритми, які є найбільш цікавими у транспортних задачах або щодо теоретичних міркувань, або щодо їхньої ефективності, а також з огляду на їхнє практичне використання в транспортних моделях. Другий аспект вказує на динамічні проблеми щодо найкоротшого шляху, які часто виникають у транспортній галузі. Автори проаналізували основні особливості проблем, які існують у відповідних умовах щодо часу транспортування та функцій витрат, загальної «хронологічної» алгоритмічної парадигми, яка називається Chrono-SPT. Ahmat докладно вивчав комунікаційні мережі. У дослідженні описані основні поняття теорії графів та їхній зв’язок із комунікаційними мережами, представлено деякі проблеми оптимізації, пов’язані з протоколами маршрутизації та моніторингом мережі, і показано, що багато проблем оптимізації є NP-повними або NP-складними.

Більшість людей знайомі з проблемою найкоротшого шляху лише в її базовій формі — знаходження найкоротшого маршруту між двома точками A і B. Однак для фахівців з комп’ютерних наук ця задача є значно ширшою та глибшою, оскільки для різних варіантів проблеми потрібні різні алгоритми.

Алгоритми пошуку найкоротшого шляху зазвичай працюють з графами, які складаються з вершин і ребер, що їх з’єднують. Графи можуть мати різні властивості: бути орієнтованими або неорієнтованими, зваженими чи незваженими. Саме ці характеристики визначають, який алгоритм буде найбільш ефективним для конкретного типу графа та задачі. Алгоритми найкоротшого шляху мають різні застосування, найвідомішим з яких є програмне забезпечення для планування маршрутів, таке як Google Maps, і ми в MyRouteOnline використовуємо алгоритми найкоротшого шляху для генерування ваших маршрутів.

Що таке алгоритм найкоротшого шляху?

Алгоритм - це, по суті, набір інструкцій або дуже специфічна процедура кроків для комп'ютера, яких він повинен дотримуватися, щоб

вирішити певну проблему або конкретне завдання. У випадку тих, хто шукає допомоги з ефективними картами маршрутів, алгоритми пошуку найкоротших шляхів - це те рішення, якого вони прагнуть. Використовуючи навігаційні програми для планування маршруту, думайте про них як про план створення алгоритму найкоротшого шляху, спеціально розроблений для комп'ютера, щоб він міг його прочитати, зрозуміти, а потім діяти, щоб створити найкоротший можливий маршрут на карті для користувача. Багато користувачів не дуже розуміють тонкощі роботи цих алгоритмів, адже їм не обов'язково розуміти все це, щоб отримати велику користь від таких додатків.

Алгоритм Дейкстри.

Алгоритм Дейкстри вирізняється з-поміж інших своєю здатністю знаходити найкоротший шлях від однієї вершини до кожної іншої вершини в межах однієї структури даних графа. Це означає, що замість того, щоб просто знайти найкоротший шлях від початкової вершини до іншої конкретної вершини, алгоритм шукає найкоротший шлях до кожної досяжної вершини - за умови, що граф не змінюється.

Алгоритм працює до тих пір, поки не будуть відвідані всі доступні вершини. Таким чином, вам потрібно буде запустити алгоритм Дейкстри лише один раз і зберегти результати, щоб використовувати їх знову і знову без повторного запуску алгоритму - знову ж таки, якщо структура даних графа не змінилася жодним чином.

У випадку змін у графі, потрібно буде перезапустити граф, щоб переконатися, що є найновіші найкоротші шляхи для вашої структури даних.

Візьмемо приклад маршрутизації, якщо потрібно дістатися з пункту А в пункт Б найкоротшим шляхом, але відомо, що деякі дороги сильно перевантажені, заблоковані, ведуться роботи і тому подібне, при використанні Дейкстри алгоритм знайде найкоротший шлях, уникаючи ребер з більшою вагою, таким чином знайшовши найкоротший маршрут.

Алгоритм Беллмана-Форда.

Подібно до алгоритму Дейкстри, алгоритм Беллмана-Форда шукає найкоротший шлях між заданою вершиною та всіма іншими вершинами графа. Хоча він повільніший за перший, Беллмана-Форда компенсує цей недолік своєю універсальністю. На відміну від алгоритму Дейкстри, Беллмана-Форда може працювати з графами, в яких деякі ваги ребер від'ємні.

Важливо зазначити, що якщо в графі є від'ємний цикл, в якому ребра в сумі дають від'ємне значення, то в ньому не існує найкоротшого або найдешевшого шляху. Це означає, що алгоритм не може знайти правильний маршрут, оскільки він закінчується на від'ємному циклі. Метод Беллмана-Форда вміє виявляти від'ємні цикли і повідомляти про їх існування.

Алгоритм Флойда-Уоршалла.

Алгоритм Флойда-Уоршалла вирізняється тим, що на відміну від попередніх двох алгоритмів, він не є алгоритмом з одним джерелом. Це означає, що він обчислює найкоротшу відстань між кожною парою вершин у графі, а не лише від однієї вершини. Він працює, розбиваючи основну задачу на менші, а потім об'єднує відповіді, щоб вирішити основну задачу про найкоротший шлях.

Метод Флойда-Уоршалла надзвичайно корисний, коли мова йде про створення маршрутів для поїздок з кількома зупинками, оскільки він обчислює найкоротший шлях між усіма відповідними вузлами. З цієї причини багато програм для планування маршрутів використовують цей алгоритм, оскільки він надасть вам найбільш оптимізований маршрут з будь-якого місця. Таким чином, незалежно від того, де ви зараз знаходитесь, Флойд-Уоршалл визначить найкоротший шлях до будь-якої іншої вершини на графіку.

Алгоритм Джонсона.

Алгоритм Джонсона найкраще працює з розрідженими графами, тобто з меншою кількістю ребер, оскільки час його роботи залежить від кількості ребер. Чим менше ребер, тим швидше він згенерує маршрут.

Цей алгоритм відрізняється від інших тим, що для визначення найкоротшого шляху він спирається на два інших алгоритми. По-перше, він використовує метод Беллмана-Форда для виявлення від'ємних циклів та усунення всіх від'ємних ребер. Потім, з цим новим графом, він покладається на алгоритм Дейкстри для обчислення найкоротших шляхів у початковому графі, який було введено.

Історія алгоритму короткого шляху MyRouteOnline

Для тих, хто «в темі», MyRouteOnline відомий своїм конкурентним, складним і високоефективним алгоритмом. Щоб зрозуміти більше про нього і про те, як він з'явився, нам потрібно повернутися до ранніх днів MyRouteOnline. Понад 30 років тому, коли люди все ще використовували для навігації складені паперові карти.

MyRouteOnline почався з доктора Баруха Аксельрода. Він був комп'ютерним науковцем, сім'янином та ентузіастом подорожей. Якщо скласти все це разом, стає зрозуміло, як з колекції пристрастей цієї людини з'явився MyRouteOnline. У перші дні, коли MyRouteOnline не був таким, яким він є сьогодні, доктор Барух Аксельрод і його команда долали великі відстані, щоб фізично встановити свій програмний продукт для багатьох великих компаній і корпорацій по всій території Сполучених Штатів. Вони навіть поверталися, щоб впроваджувати оновлення програмного забезпечення.

Однак це сталося лише через багато років, коли дочка доктора Баруха Аксельрода звернулася до нього з професійною проблемою. Вона не могла знайти найкращий спосіб вчасно виконати всі доставки для свого флористичного бізнесу. Саме тоді MyRouteOnline перетворився на алгоритм і продукт, яким сьогодні користуються тисячі людей.

Частіше за все, найкращий алгоритм для використання не буде залежати від нас, а буде залежати від типу графа, який ви використовуєте, та задачі про найкоротший шлях, яку ви розв'язуєте. Наприклад, для задач з ребрами з від'ємною вагою ви звернетесь до алгоритму Беллмана-Форда, тоді як для розріджених графів без від'ємних ребер ви звернетесь до алгоритму Дейкстри.

Коли справа доходить до суті, багато аспектів цих алгоритмів однакові, однак, розглядаючи продуктивність і використання, саме тут виявляються відмінності. Тому, замість того, щоб питати, який алгоритм найкращий, слід подумати, який з них підходить для типу графа і задачі пошуку найкоротшого шляху. [3]

3.1 АЛГОРИТМИ МЕТОДІВ ПОШУКУ НАЙКОРОТШИХ ШЛЯХІВ ГРАФА

3.1.1 Алгоритм Дейкстри

Алгоритм Дейкстри розв'язує задачу про найкоротший шлях для будь-якого зваженого орієнтованого графа з невід'ємними вагами. Він може працювати з графами, що складаються з циклів, але від'ємні ваги призводять до неправильних результатів.

Алгоритм підтримує пріоритетну чергу minQ, яка використовується для зберігання необроблених вершин з їх оцінками найкоротшого шляху est(v) як ключових значень. Потім алгоритм багаторазово витягує вершину u з мінімальним est(u) з minQ і розслабляє всі ребра, що виходять з u до будь-якої вершини в minQ. Після того, як одна вершина буде вилучена з minQ і всі ребра, що проходять через неї, будуть розслаблені, алгоритм вважатиме цю вершину обробленою і більше не торкатиметься її. Алгоритм Дейкстри зупиняється або коли minQ порожній, або коли кожна вершина буде розглянута рівно один раз. Як маршрутизатор S вирішує, як направити пакет до заданого пункту призначення D? Кожне з'єднання у мережі має свою вартість, і маршрутизатори, такі як S, часто обчислюють найкоротші (тобто найдешевші) шляхи до пунктів призначення у локальному домені. Припустимо, що вартість є невеликим цілим числом. Пригадайте з Розділу 2, що найпоширенішим протоколом маршрутизації у домені є OSPF (Open Shortest Path First), який базується на маршрутизації за станом каналу зв'язку. При маршрутизації за станом з'єднання кожен маршрутизатор у підмережі надсилає пакет стану з'єднання (LSP) з переліком своїх з'єднань усім своїм сусідам. Кожен LSP надсилається кожному іншому маршрутизатору в підмережі. Кожен маршрутизатор надсилає свій LSP іншим маршрутизаторам, використовуючи примітивний протокол переповнення. Після того, як кожен маршрутизатор отримає LSP від кожного маршрутизатора, кожен маршрутизатор матиме повну карту

мережі. Припускаючи, що топологія залишається стабільною, кожен маршрутизатор тепер може обчислити свій найкоротший шлях до кожного іншого вузла в мережі, використовуючи стандартний алгоритм пошуку найкоротшого шляху, наприклад, алгоритм Дейкстри. Розрахунок найкоротшого шляху базується на алгоритмі Дейкстри. Як тільки маршрутизатор отримує нову LSP, він чекає 5 секунд, перш ніж запустити розрахунок найкоротшого шляху. Між двома послідовними обчисленнями найкоротшого шляху в межах однієї області існує 10-секундний таймер затримки. Однак маршрутизатори L1/L2, які знаходяться в області L1, повинні виконувати окремі розрахунки найкоротшого шляху, один для області L1, а інший для області L2. Метрика каналу в IS-IS спочатку була обмежена 6 бітами і, таким чином, її значення коливається від 0 до 63, а загальна вартість шляху в домені IS-IS може мати максимальне значення 1023. Ця 6-бітова метрика відома як вузька метрика. Розширення широкої метрики тепер доступне через розширення трафіку для IS-IS, яке дозволяє 24-бітну метрику, що дозволяє діапазон від 0 до 16,777,215. На відміну від алгоритму Флойда-Уоршалла, алгоритм Дейкстри використовує розрідженість графа для зменшення його складності. Алгоритм досліджує вихідні ребра графа з вихідної вершини, починаючи з ребра з найменшою вагою, і поступово будує найкоротші шляхи до всіх інших вершин. Алгоритм Дейкстри може бути використаний для розв'язання всіх трьох представлених задач пошуку найкоротшого шляху, якщо в графі не існує від'ємних ваг ребер. В Алгоритмі 2 ми представляємо варіант розв'язання задачі SSSP за алгоритмом Дейкстри. Для того, щоб розв'язати задачу APSP, нам просто потрібно застосувати той самий алгоритм, використовуючи кожну вершину графа як вихідну вершину. Для розв'язання задачі SPSP ми можемо використати той самий алгоритм і зупинити дослідження, як тільки досягнемо цільової вершини. Після того, як даний вузол має копію LSP від кожного іншого вузла, він може обчислити повну карту топології мережі, і на основі цієї карти він може

вибрати найкращий маршрут до кожного пункту призначення. Питання полягає в тому, як саме він розраховує маршрути на основі цієї інформації. Рішення базується на відомому алгоритмі з теорії графів - алгоритмі найкоротшого шляху Дейкстри.

В алгоритмі Дейкстри ми починаємо з дослідження вершин, найближчих до початкової вершини, і записуємо відстань між ними та початковою вершиною, рухаючись вперед. Алгоритм гарантує, що ми відвідаємо кожну вершину графа в порядку зростання відстані від початкової вершини. Для зберігання невідвіданих вершин та відстаней між ними ми використовуємо структуру даних пріоритетної черги. Спочатку відстань для всіх невідвіданих вершин дорівнює нескінченності (або великому значенню), окрім вихідної вершини, яка дорівнює нулю. Розбиваємо цей процес на менші кроки: Встановлення відстані до початкової вершини рівною нулю, а до всіх інших вершин - нескінченності. Вибір початкової вершини і відвідування всіх сусідніх вершин, які ще не були відвідані. Оновлення відстані до відвіданих вершин, враховуючи, що щойно пройдений шлях має мінімальну відстань. Продовження тих самих дій для інших невідвіданих вершин, поки не буде відвідано кінцеву вершину. Найкоротший шлях і відстань буде отримано після дослідження всіх можливих шляхів. Існує багато типів алгоритмів найкоротшого шляху, але для цього дослідження ми також очікуємо найбезпечніший шлях для евакуації. Алгоритм Дейкстри — це алгоритм пошуку графа, який розв’язує проблему найкоротшого шляху з одним джерелом для графа з невід’ємною вартістю шляху ребра. З 1959 року алгоритм Дейкстри був визнаний найкращим алгоритмом і використовувався як метод пошуку найкоротшого шляху. Найкоротший шлях — це шлях пошуку мінімальної ваги, що з’єднує дві вказані вершини, або між усіма парами вершин, або від вказаної вершини до всіх інших. Крім того, час, необхідний для вирішення, набагато менший у порівнянні з динамічним або лінійним програмуванням.

3.1.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда - алгоритм пошуку найкоротшого шляху у зваженому графі. Алгоритм знаходить найкоротші шляхи від однієї вершини графа до всіх інших. На відміну від алгоритму Дейкстри, алгоритм Беллмана-Форда допускає ребра з від'ємною вагою. Алгоритм носить ім'я двох американських учених: Річарда Беллмана (Richard Bellman) і Лестера Форда (Lester Ford). Форд фактично винайшов цей алгоритм у 1956 р. під час вивчення іншого математичного завдання, підзавдання якого звелося до пошуку найкоротшого шляху в графі, і Форд дав начерк алгоритму, що розв'язує це завдання. Беллман у 1958 р. опублікував статтю, присвячену конкретно завданню знаходження найкоротшого шляху, і в цій статті він чітко сформулював алгоритм у тому вигляді, в якому він відомий нам зараз. Алгоритм маршрутизації RIP (алгоритм Беллмана-Форда) вперше розробили 1969 року, як основний для мережі ARPANET. Дано орієнтований або неорієнтований граф G зі зваженими ребрами. Довжина шляху - сума ваг ребер, що входять у цей шлях. Знайти найкоротші шляхи від виділеної вершини s до всіх вершин графа (у графі, що містить цикл з від'ємною сумарною вагою, існує як завгодно короткий шлях від однієї вершини цього циклу до іншої) Цикл, сума ваг ребер якого від'ємна, називається від'ємним циклом. Для знаходження найкоротших шляхів від однієї вершини до всіх інших, скористаємося методом динамічного програмування. Побудуємо матрицю Aij, елементи яких позначатимуть таке: Aij - це довжина найкоротшого шляху з s в i, що містить не більше j ребер. Шлях, що містить 0 ребер, існує тільки до вершини s. Т. ч., Ai0 дорівнює 0 за i = s, і $+\infty$ в іншому випадку. Загалом, алгоритм Дейкстри, порівняно з алгоритмом Беллмана-Форда, забезпечує більш реальну оцінку ситуації в мережі, більш швидку реакцію. Беллмана-Форда, забезпечує реальнішу оцінку ситуації в мережі, швидшу реакцію на важливі зміни в мережі (такі, як увімкнення нової лінії

зв'язку) і зменшує зациклення пакетів; однак алгоритм Дейкстри складніший у реалізації і потребує в рази більше пам'яті.

3.1.3 Евристичний алгоритм

У світі науки, техніки та повсякденного життя ми часто стикаємося з ситуаціями, які потребують швидкого прийняття рішень. Інколи немає можливості чи ресурсів використовувати складні розрахункові методи або систематичний підхід. У таких випадках доцільно звернутися до евристичних методів — способів вирішення завдань, що спираються на інтуїцію, досвід, спрощення та адаптацію. Евристичний підхід часто є ефективним для розв'язання складних задач у ситуаціях невизначеності або дефіциту інформації. Евристичний метод — це підхід до вирішення завдань або прийняття рішень, який базується на використанні наближених, але практичних правил, що ґрунтуються на попередньому досвіді або інтуїції. Його назва походить від грецького слова heuriskein, що означає "знаходити" або "відкривати". Цей метод не завжди гарантує оптимальне чи правильне рішення, але часто дозволяє знайти задовільний результат швидше та з меншими витратами. Його використовують у багатьох галузях, таких як психологія, математика, інженерія, програмування, медицина тощо.

Основні принципи евристичного підходу

1. Спрощення. Евристика зводить складну задачу до простішої форми. Це дозволяє зосередитися на ключових аспектах проблеми.
2. Аналіз аналогій. Метод базується на попередньому досвіді або рішеннях схожих задач.
3. Ітеративність. Рішення досягається шляхом поступового вдосконалення попередніх наближень.
4. Пріоритетність. Зосередження на найважливіших аспектах задачі, ігноруючи менш значущі деталі.
5. Інтуїтивність. Розрахунок на інтуїцію як додатковий інструмент у процесі прийняття рішень.

Приклади евристичних методів

1. Правило великого пальця. Прості правила чи рекомендації, які зазвичай працюють у повсякденному житті.

2. Метод проб і помилок. Випробування різних варіантів до знаходження підходящого рішення.

3. Метод аналогій. Пошук вирішення задачі через порівняння з уже відомими сценаріями.

4. Жадібні алгоритми. Використовуються в інформатиці для оптимізації. Кожен крок базується на виборі локально найкращого варіанту.

5. Метод "розділяй і володарюй". Завдання розбивається на менші частини, кожна з яких вирішується окремо.

Переваги евристичного підходу

1. Швидкість. Евристика дозволяє швидко знайти рішення, що особливо важливо в ситуаціях, коли час є критичним фактором.

2. Простота. Використання евристичних правил не потребує складних розрахунків чи глибокого аналізу.

3. Гнучкість. Може бути адаптована до широкого спектра задач та умов.

4. Ефективність при невизначеності. Підхід добре працює в умовах браку інформації.

Недоліки евристичних методів

1. Неточність. Евристика може привести до субоптимального рішення або навіть до помилок.

2. Залежність від контексту. Метод може не працювати в нових чи непередбачуваних умовах.

3. Схильність до когнітивних упереджень. Наприклад, люди можуть переоцінювати свою інтуїцію або досвід, що впливає на об'єктивність.

4. Відсутність універсальності. Евристичні методи, які працюють у певних умовах, не завжди придатні для інших.

Застосування евристичного методу

1. Наука та техніка. евристика використовується для моделювання складних систем, оптимізації процесів або прогнозування.
2. Штучний інтелект і машинне навчання. Алгоритми, такі як генетичні чи еволюційні, є прикладом евристичних методів для пошуку оптимальних рішень.
3. Психологія. У психології евристика допомагає пояснювати, як люди приймають рішення в умовах невизначеності.
4. Освіта. У навчанні евристичний метод допомагає студентам знаходити рішення через самостійний аналіз, досвід і експерименти.
5. Бізнес і економіка. Використовується для оцінки ризиків, прийняття стратегічних рішень або створення інновацій.

Евристичний метод — це потужний інструмент, який дозволяє швидко та ефективно вирішувати завдання, особливо в умовах невизначеності чи обмежених ресурсів. Хоча цей підхід має свої недоліки, його переваги роблять його незамінним у багатьох галузях. Поєднання евристики з аналітичними методами дозволяє досягати оптимальних результатів, зберігаючи баланс між точністю та ефективністю.

Вивчення та вдосконалення евристичних методів відкриває нові можливості для розвитку технологій, науки та управління, що робить їх важливим елементом сучасного світу.

3.2 Алгоритми програмної реалізації знаходження найкоротшого шляху графа

Алгоритм Дейкстри - це алгоритм пошуку найкоротшого шляху в графі з невід'ємними вагами ребер. Основна ідея полягає у тому, щоб знайти найкоротший шлях від вихідного вузла до всіх інших вузлів у графі. Для розробки програмного забезпечення пошуку найкоротшого шляху, було обрано алгоритм Дейкстри. Алгоритм Дейкстри використовується для знаходження найкоротшого шляху в графі з невід'ємними вагами ребер.

1. Вибір початкової вершини: Почнемо з вибору вершини, з якої будемо шукати найкоротші шляхи до всіх інших вершин. Назвемо її стартовою вершиною.

2. Ініціалізація: Для кожної вершини в графі встановимо початкову відстань. Відстань до стартової вершини буде 0, а до всіх інших вершин - нескінченність (або дуже велике число). Створимо набір відвіданих вершин, спочатку порожній.

3. Вибір вершини: Знайдемо не відвідану вершину з найменшою відстанню (на початку це буде стартова вершина).

4. Оновлення сусідів: Для кожного сусіда поточної вершини обчислимо нову відстань до нього через поточну вершину. Якщо нова відстань менша за раніше записану відстань, оновимо її.

5. Позначення вершини відвіданою: Після обробки всіх сусідів, позначимо поточну вершину як відвідану. Це означає, що ми знайшли найкоротший шлях до цієї вершини і більше не будемо її перевіряти.

6. Повторення: Повторюємо кроки 3-5, поки не відвідаємо всі вершини або не знайдемо найкоротші шляхи до всіх потрібних вершин.

7. Завершення: Коли всі вершини відвідані або більше не залишилося вершин з відомими найменшими відстанями, алгоритм завершується. В результаті у нас є найкоротші відстані від стартової вершини до всіх інших вершин.

Приклад роботи алгоритму:

- Граф має вершини: A, B, C, D
- Вага ребер: A-B (1), A-C (4), B-C (2), B-D (5), C-D (1)

1. Почнемо з вершини A:
   - Відстань до A = 0
   - Відстань до B = нескінченність
   - Відстань до C = нескінченність
   - Відстань до D = нескінченність

2. Вибираємо A, оновлюємо відстані до її сусідів:

- Відстань до B = 1

- Відстань до C = 4

3. Вибираємо B (найменша відстань серед не відвіданих), оновлюємо відстані:

- Відстань до C = 3 (1 + 2, бо через B швидше, ніж напряму від A)

- Відстань до D = 6 (1 + 5)

4. Вибираємо C, оновлюємо відстані:

- Відстань до D = 4 (3 + 1, бо через C швидше)

5. Вибираємо D (остання не відвідана вершина).

Результат: найкоротший шлях від A до B = 1, до C = 3, до D = 4.

Алгоритм Беллмана-Форда використовується для знаходження найкоротших шляхів від однієї початкової вершини до всіх інших у графі, навіть якщо в ньому є ребра з від'ємною вагою. Алгоритм починається з ініціалізації: для кожної вершини встановлюється початкова відстань як нескінченність, оскільки ми ще не знаємо, як до неї дістатися, а для початкової вершини відстань задається як нуль, тому що ми вже знаходимося в цій точці. Після цього алгоритм виконує основний процес оновлення, який триває рівно V-1 раундів, де V — кількість вершин у графі. У кожному раунді перевіряються всі ребра графа. Якщо через ребро можна знайти коротший шлях до кінцевої вершини, то відстань до цієї вершини оновлюється. Наприклад, якщо ми вже знаємо мінімальну відстань до вершини A, і з неї є ребро до вершини B з певною вагою, ми перевіряємо, чи сума відстані до A і ваги ребра менша за поточну відстань до B. Якщо це так, ми оновлюємо значення відстані до B. Після завершення V-1 раунду ми маємо найкоротші відстані для всіх вершин, оскільки шлях до кожної вершини у графі з V вершинами може складатися максимум із V-1 ребра. Після цього алгоритм проводить додатковий раунд перевірки для виявлення негативних циклів. Негативний цикл — це цикл у графі, де сума ваг ребер є від'ємною. Якщо на цьому етапі виявляється, що хоча б одну відстань ще можна зменшити, це означає, що у графі є

негативний цикл, і задача пошуку найкоротших шляхів стає нерозв'язною, оскільки можна нескінченно зменшувати відстані, проходячи по цьому циклу. У підсумку, якщо негативних циклів немає, алгоритм завершує роботу, і ми отримуємо найкоротші відстані від початкової вершини до всіх інших.

Уявімо, що ми маємо орієнтований граф із чотирма вершинами: A, B, C та D. Граф описується такими ребрами з відповідними вагами:

- Від A до B із вагою 4,
- Від A до C із вагою 2,
- Від B до C із вагою -3,
- Від C до D із вагою 2,
- Від D до B із вагою 1.

Мета алгоритму — знайти найкоротший шлях від вершини A до всіх інших.

1. Ініціалізація:

На початку встановлюємо відстані до всіх вершин як нескінченність (∞), крім початкової вершини A, відстань до якої дорівнює нулю. Тобто:

- Відстань до A = 0,
- Відстань до B = ∞,
- Відстань до C = ∞,
- Відстань до D = ∞.

2. Перший раунд:

Перевіряємо кожне ребро та оновлюємо відстані:

- Ребро A→B (вага 4): відстань до B оновлюється до (0 + 4 = 4).

- Ребро A→C (вага 2): відстань до C оновлюється до (0 + 2 = 2).

- Ребро B→C (вага -3): оскільки поточна відстань до B дорівнює 4, нова відстань до C буде (4 - 3 = 1), тому її оновлюємо.

- Ребро C→D (вага 2): поточна відстань до C дорівнює 1, тому відстань до D оновлюється до (1 + 2 = 3).

- Ребро D→B (вага 1): поточна відстань до D дорівнює 3, тому відстань до B оновлюється до (3 + 1 = 4) (у цьому раунді вона не змінюється).

Тепер відстані виглядають так:

- (A = 0), (B = 4), (C = 1), (D = 3).

3. Другий раунд:

Знову перевіряємо всі ребра:

- Ребро A→B: поточна відстань до B не змінюється (0 + 4 = 4).
- Ребро A→C: відстань до C теж не змінюється (0 + 2 = 2).
- Ребро B→C: нічого не змінюється (4 - 3 = 1).
- Ребро C→D: нічого не змінюється (1 + 2 = 3).
- Ребро D→B: нічого не змінюється (3 + 1 = 4).

Відстані залишаються такими ж:

- (A = 0), (B = 4), (C = 1), (D = 3).

4. Третій раунд:

Перевіряємо всі ребра втретє, але жодна відстань більше не змінюється. Це означає, що ми знайшли найкоротші відстані.

5. Перевірка на негативний цикл:

Ще раз перевіряємо всі ребра. Якщо після цього раунду хоча б одна відстань змінюється, це свідчить про наявність негативного циклу. У нашому прикладі всі відстані залишаються без змін.

6. Результат:

Найкоротші відстані від (A) до всіх інших вершин:

- До (B) — (4),
- До (C) — (1),
- До (D) — (3).

Таким чином, алгоритм успішно знайшов найкоротші шляхи.

3.3 Блок-схема роботи алгоритму програми

Блок-схема алгоритму програми (рис. 3.1)

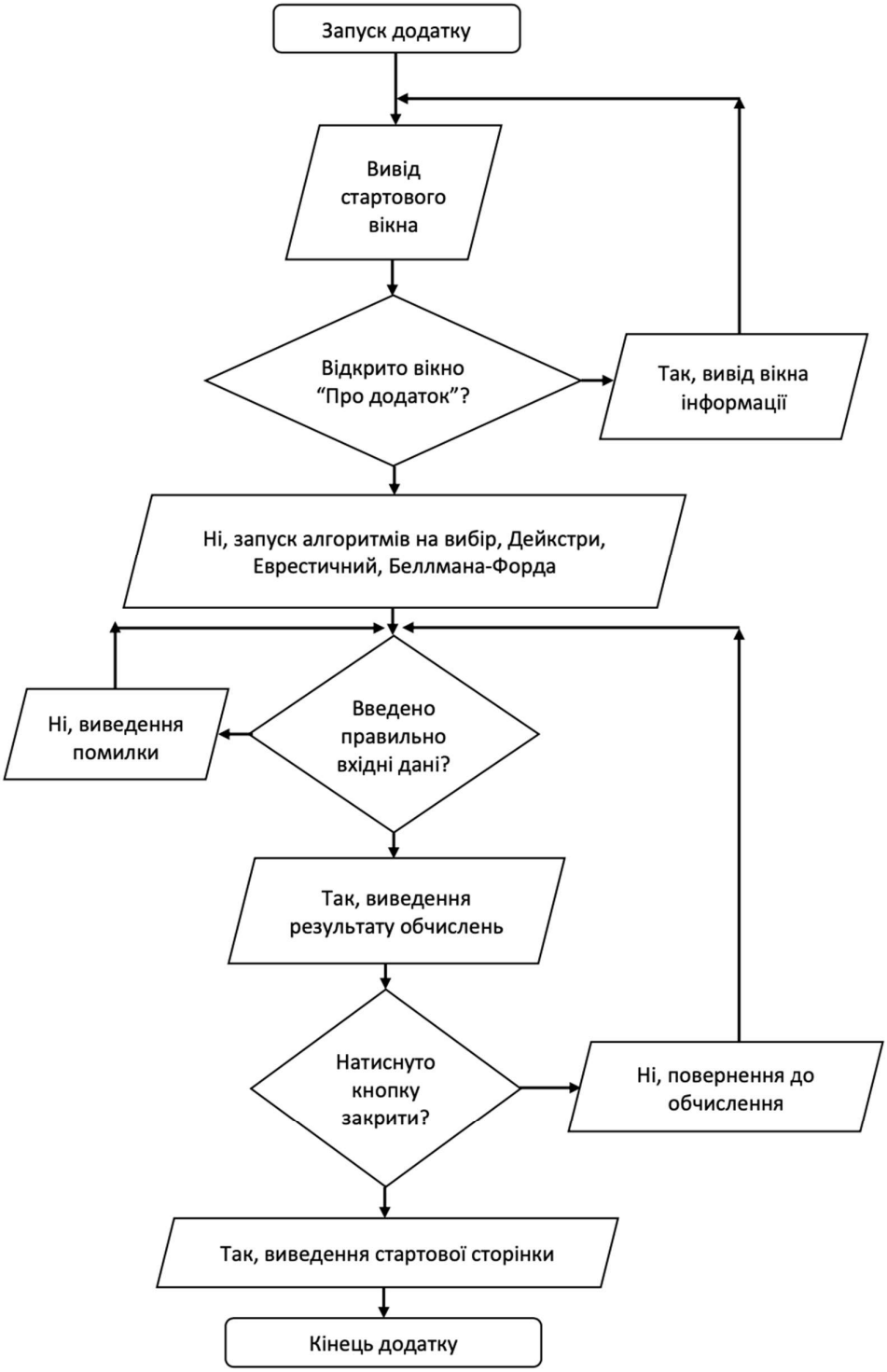

Рисунок 3.1 – Блок-схема алгоритму програми

3.4 Порівняльний аналіз методів

Алгоритми пошуку найкоротшого шляху є важливими елементами в комп'ютерній науці, зокрема в області графів. Вони широко застосовуються у сфері навігації, транспортного планування, телекомунікацій та ігрової індустрії. Серед найбільш популярних алгоритмів виділяються алгоритм Дейкстри, алгоритм Беллмана-Форда та евристичні підходи.

1. Алгоритм Дейкстри є одним із найвідоміших алгоритмів для пошуку найкоротшого шляху в графах з невід'ємними вагами ребер.

Основні характеристики:

Обмеження: Працює лише з невід'ємними вагами.

Переваги:

1. Ефективність на графах з невеликою кількістю вершин або при використанні спеціалізованих структур даних.
2. Гарантує коректний результат для графів з невід'ємними вагами.
3. Недоліки:
4. Не може працювати з графами, що містять ребра з від'ємною вагою.
5. Менш ефективний на розріджених графах порівняно з деякими іншими методами.

Практичне застосування:

1. Системи GPS-навігації.
2. Планування мережі передачі даних.
3. Ігрові алгоритми для пошуку найкоротших шляхів.

2. Алгоритм Беллмана-Форда розширює функціонал Дейкстри, дозволяючи працювати з графами, що містять ребра з від'ємними вагами.

Основні характеристики:

Обмеження: Чутливий до наявності циклів з від'ємною вагою.

Переваги:

1. Може обробляти графи з від'ємними вагами.
2. Виявляє цикли з від'ємною вагою, що дозволяє використовувати його для аналізу аномалій у графах.

Недоліки:

1. Повільніший порівняно з Дейкстрою на великих графах.

2. Не підходить для графів із великою кількістю ребер через високу обчислювальну складність.

Практичне застосування:

1. Фінансові системи для аналізу арбітражних можливостей.
2. Визначення критичних шляхів у транспортних системах.
3. Задачі оптимізації потоків у мережах.

3. Евристичний алгоритм на прикладі A*

Алгоритм A* є прикладом евристичного підходу, який комбінує переваги пошуку за графом та евристик.

Основні характеристики:

Обмеження: Потребує добре визначеної евристичної функції для ефективної роботи.

Переваги:

1. Гнучкість у налаштуванні за допомогою евристичних функцій.
2. Ефективність на практиці для задач із великими графами.
3. Можливість зосереджувати пошук лише в "перспективних" напрямках.

Недоліки:

1. Ефективність сильно залежить від якості евристичної функції.
2. Високі вимоги до пам'яті для великих графів.

Практичне застосування:

1. Пошук шляхів у робототехніці.
2. Планування руху автономних транспортних засобів.
3. Ігрова індустрія для моделювання поведінки персонажів.

Алгоритм Дейкстри є ідеальним вибором для задач із графами, що не містять від'ємних ваг, завдяки своїй ефективності. Алгоритм Беллмана-Форда підходить для задач, де важливо враховувати від'ємні ваги, хоча він менш ефективний для великих графів. Евристичні алгоритми, такі як A*, забезпечують чудову продуктивність у практичних сценаріях, проте вимагають ретельного підбору евристики.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Обґрунтування вибору програмних засобів

C# є сучасною, об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft, яка забезпечує високий рівень продуктивності, надійності та гнучкості. Вибір C# як основного програмного засобу обумовлений низкою причин.

По-перше, ця мова є частиною екосистеми .NET, що забезпечує широку підтримку бібліотек, інструментів та фреймворків, таких як ASP.NET для веб-розробки, Xamarin для мобільних додатків і WPF для настільних програм. Це дозволяє створювати масштабовані, кросплатформенні рішення.

По-друге, C# має вбудовані механізми управління пам’яттю, зокрема автоматичне збирання сміття, що значно знижує ризики витоків пам’яті та забезпечує стабільність роботи додатків. Крім того, підтримка типізації на рівні компілятора сприяє виявленню помилок на ранніх етапах розробки.

Третім важливим фактором є інтеграція C# із середовищем розробки Visual Studio, яке пропонує потужні інструменти для налагодження, тестування та рефакторингу коду. Це скорочує час розробки і сприяє підвищенню продуктивності команди.

Крім того, мова C# підтримує сучасні концепції програмування, такі як асинхронність, паралельні обчислення, LINQ для роботи з даними та модульність через механізм зборів (assemblies). Це робить її зручною для реалізації складних систем із багатошаровою архітектурою.

Важливим аргументом на користь вибору C# є його велика популярність у спільноті розробників, доступність якісної документації, прикладів коду та активна підтримка з боку Microsoft. Усе це робить мову перспективним вибором для створення програмного забезпечення різного масштабу та призначення.

4.2. Опис програмної реалізації

Програмне забезпечення для пошуку найкоротшого шляху графу реалізовано на мові програмування C#, в середовищі розробки Visual Studio. (рис. 4.1)

Рисунок 4.1 – Середовище Visual Studio

Після вибору мови для розробки, розпочинається процес створення нового проєкту на мові програмування C# . (рис. 4.2)

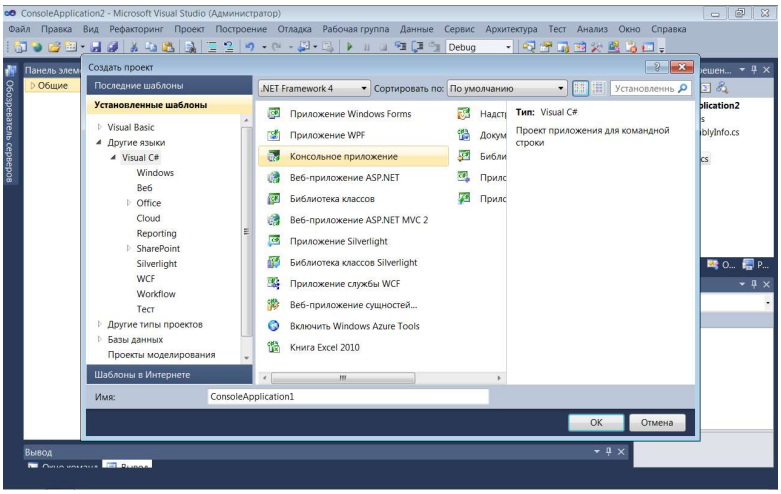

Рисунок 4.2 – Створення нового консольного додатку.

Розглянемо алгоритм роботи:

Крок 1. Визначення структури графа:

Потрібно визначити структуру графа. Для цього будемо використовувати матрицю суміжності.

```
Dim graph(,) As Integer = {
    {0, 10, 20, 0, 0, 0},
    {10, 0, 0, 50, 10, 0},
    {20, 0, 0, 20, 33, 0},
    {0, 50, 20, 0, 20, 2},
    {0, 10, 33, 20, 0, 1},
    {0, 0, 0, 2, 1, 0}
}
```

Крок 2. Ініціалізація:

Створюємо масиви для зберігання відстаней та відвіданих вершин.

```
Dim numVertices As Integer = 6
Dim distances(numVertices - 1) As Integer
Dim visited(numVertices - 1) As Boolean
```

```
Dim previous(numVertices - 1) As Integer
For i As Integer = 0 To numVertices - 1
   distances(i) = Integer.MaxValue
   visited(i) = False
   previous(i) = -1
Next

distances(0) = 0 ' Початкова вершина
```

Крок 3. Головний цикл алгоритму:

Основний цикл алгоритму, який виконується до тих пір, поки не буде оброблено всі вершини.

```
For i As Integer = 0 To numVertices - 1
   Dim u As Integer = FindMinDistance(distances, visited, numVertices)
   visited(u) = True
   For v As Integer = 0 To numVertices - 1
      If Not visited(v) And graph(u, v) <> 0 And distances(u) <> Integer.MaxValue And distances(u) + graph(u, v) < distances(v) Then
         distances(v) = distances(u) + graph(u, v)
         previous(v) = u
      End If
   Next
Next
```

Крок 4. Допоміжна функція для знаходження вершини з мінімальною відстанню:

```
Function FindMinDistance(ByRef distances() As Integer, ByRef visited() As Boolean, ByVal numVertices As Integer) As Integer
   Dim min As Integer = Integer.MaxValue
   Dim minIndex As Integer = -1
   For i As Integer = 0 To numVertices - 1
      If Not visited(i) And distances(i) <= min Then
```

```
                min = distances(i)
                minIndex = i
            End If
        Next
        Return minIndex
    End Function
```

Крок 5. Виведення результатів:

Виведемо найкоротші відстані та шляхи до кожної вершини.

```
    For i As Integer = 0 To numVertices - 1
        Console.WriteLine("Відстань від вершини 0 до вершини " & i & " дорівнює " & distances(i))
        Console.Write("Шлях: " & i)
        PrintPath(previous, i)
        Console.WriteLine()
    Next
    Sub PrintPath(ByVal previous() As Integer, ByVal j As Integer)
        If previous(j) = -1 Then
            Return
        End If
        PrintPath(previous, previous(j))
        Console.Write(" <- " & previous(j))
    End Sub
```

Наступний крок розробки – програмна реалізація алгоритму Дейкстри в середовищі Visual Studio. (рис. 4.4)

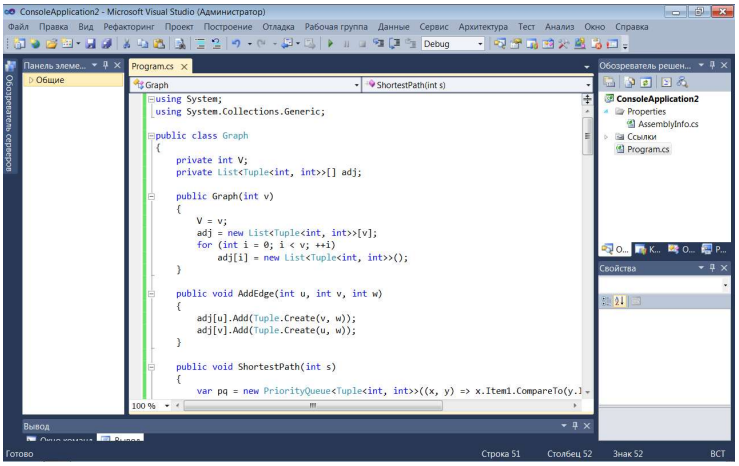

Рисунок 4.4 – Програмна реалізація на мові програмування C#

Додатково, в консольному додатку відкориговано конфігурації збірки додатку. (рис. 4.5)

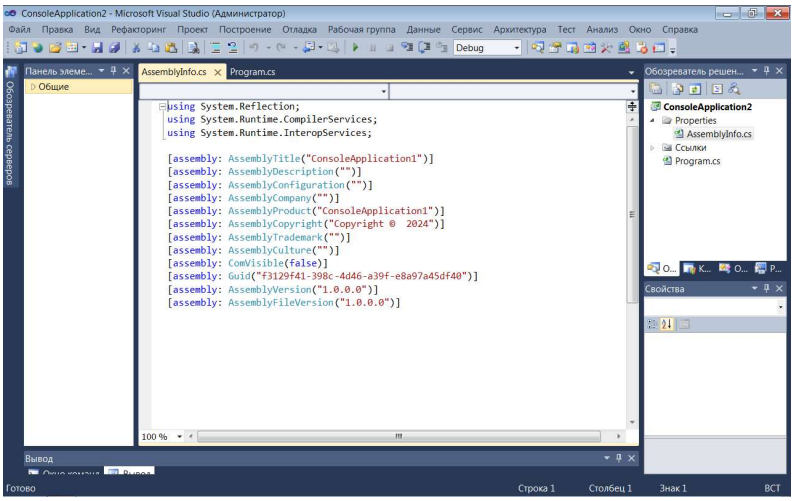

Рисунок 4.5 – Конфігурації збірки додатку C#

При реалізації методу в середовищі програмування, використано наступні об’єкти. (рис. 4.6)

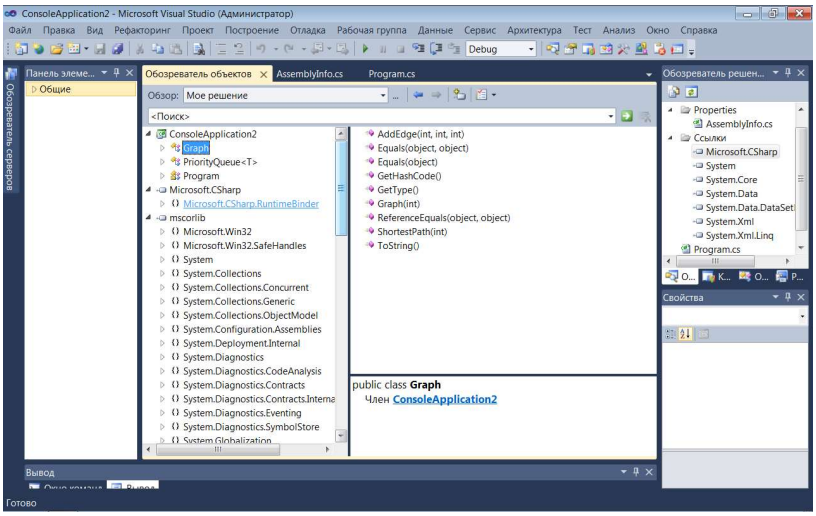

Рисунок 4.6 – Об’єкти додатку C#

Після програмування, виконано перший пробник запуск додатку та перевірка його функціоналу, користувачу виводиться інструкція щодо введення умови в додаток. (рис. 4.7)

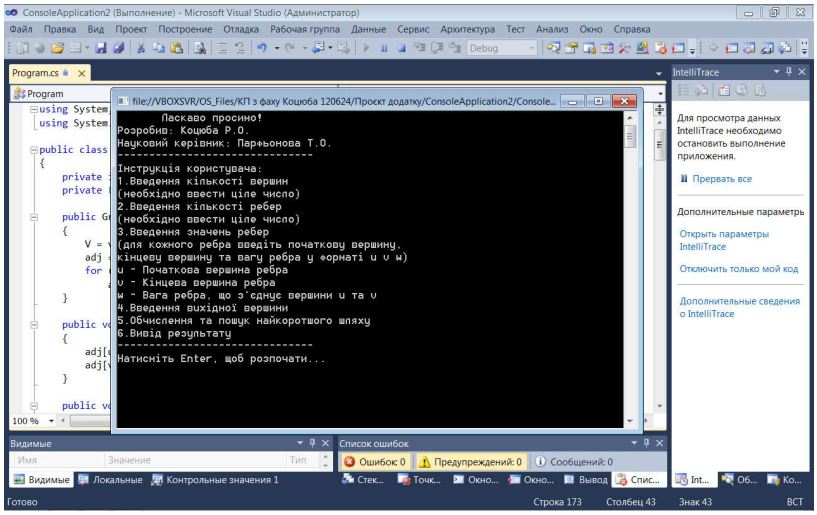

Рисунок 4.7 – Пробний запуск додатку C#

Наступним кроком, після тестування консольної версії, необхідно перенести програмне забезпечення на візуальний інтерфейс, для цього створено новий проект Windows Forms та чисту форму графічного інтерфейсу. (рис. 4.8)

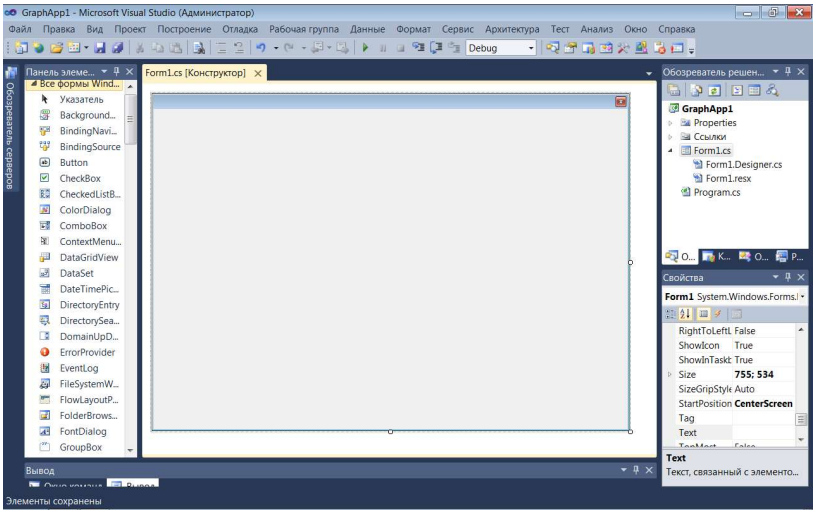

Рисунок 4.8 – Чиста форма

Потім, потрібно внести зміни в програмний код, для правильної роботи алгоритму. (рис. 4.9)

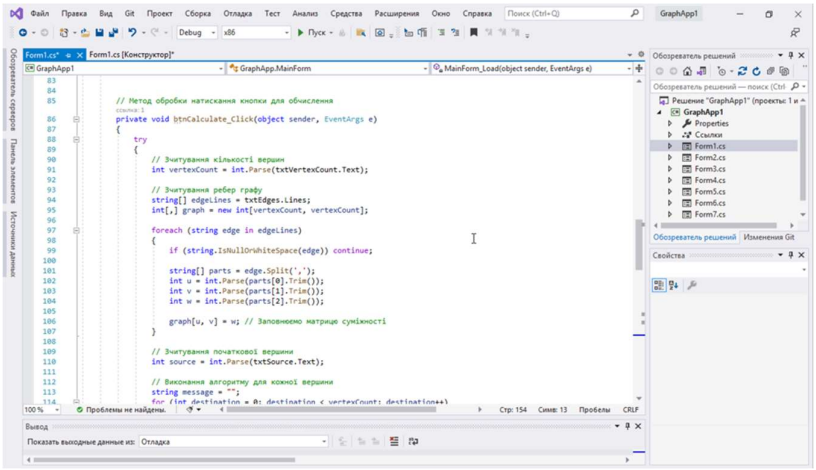

Рисунок 4.9 – Програмування додатку

Після внесення змін, розміщуються елементи керування на формі та додатково налаштовуються їх параметри. Виконується адаптація алгоритму до цих елементів. (рис. 4.10)

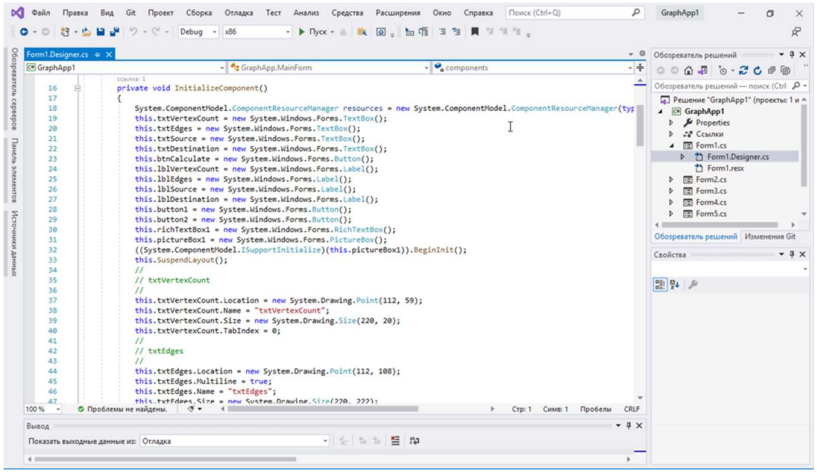

Рисунок 4.10 – Налаштування елементів керування

Далі формується дизайн додатку, більш точніше та симетричніше підганяються до розміру форми елементи керування додатком. (рис. 4.11)

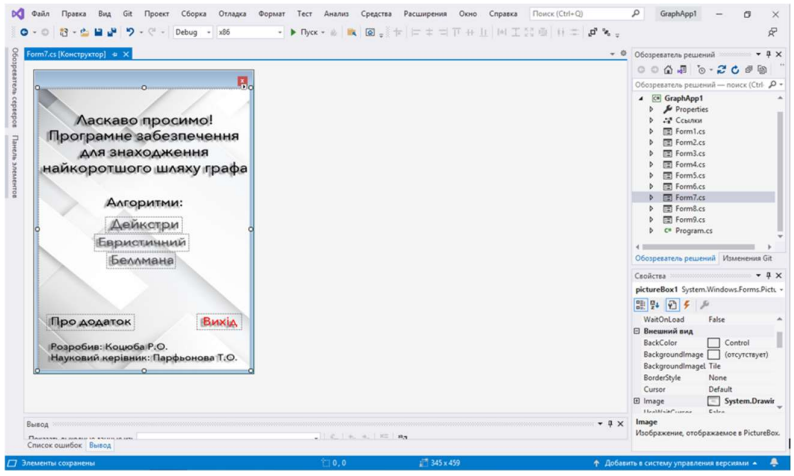

Рисунок 4.11 – Дизайн додатку

Також розроблено вікно інформації, у якому розміщено загальну інформацію про додаток. (рис. 4.12)

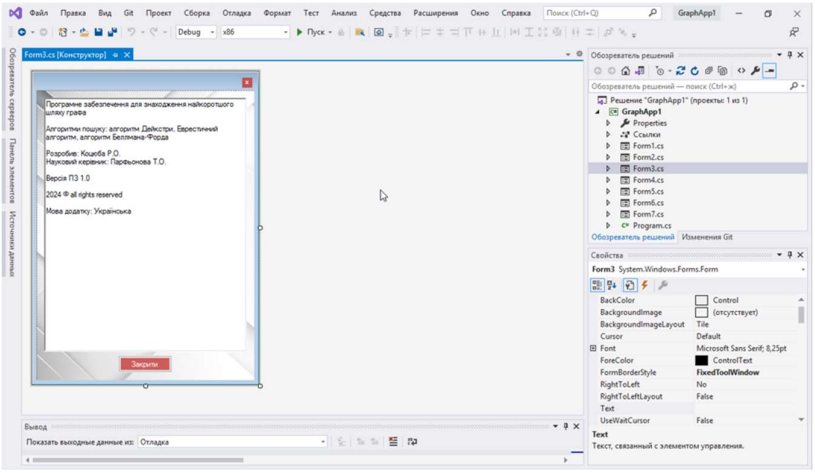

Рисунок 4.12 – Вікно інформації

В кінці розробки проведено декілька тестових запусків програмного забезпечення та виконано тестування на виявлення помилок. (рис. 4.13)

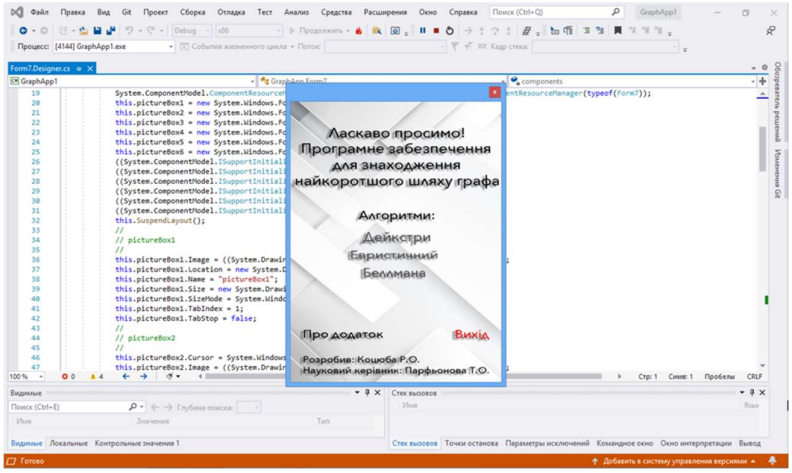

Рисунок 4.13 – Пробний запуск додатку та тестування

Далі розпочинається розробка інтерфейсу алгоритму Дейкстри. (рис. 4.14)

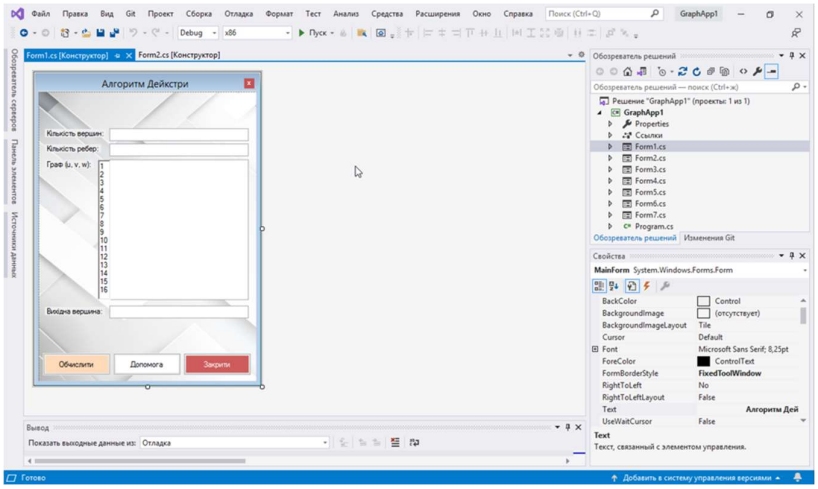

Рисунок 4.14 – Алгоритм Дейкстри

Для зручності користувача, додатково розроблено вікно допомоги, яке містить інструкцію користування до алгоритму Дейкстри. (рис. 4.15)

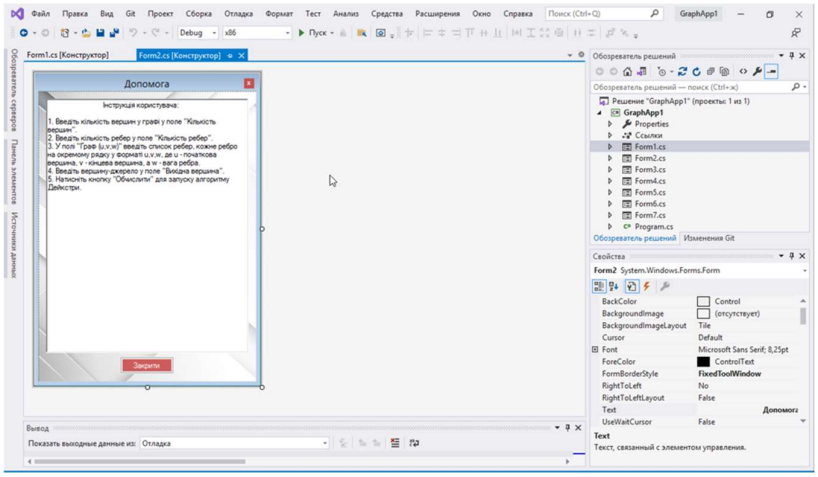

Рисунок 4.15 – Вікно допомоги

Далі розпочинається розробка інтерфейсу та логіки роботи алгоритму Беллмана-Форда. (рис. 4.16)

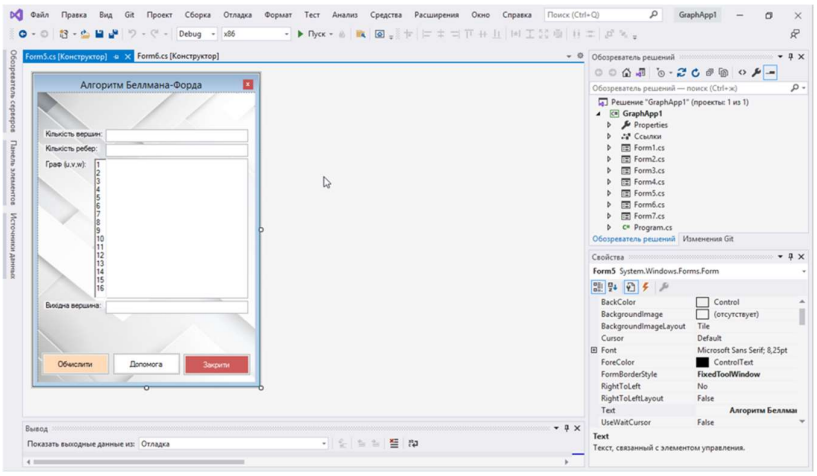

Рисунок 4.16 – Алгоритм Беллмана-Форда

Також зручності користувача, теж додатково розроблено вікно допомоги, яке містить інструкцію користування до алгоритму Беллмана-Форда. (рис. 4.17)

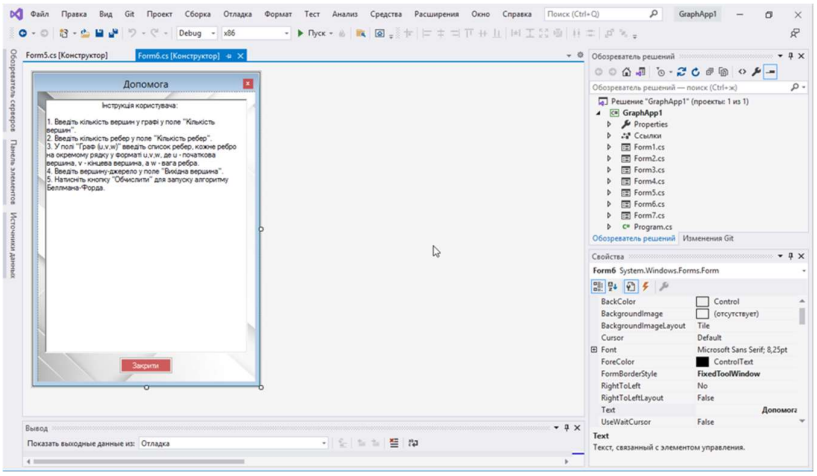

Рисунок 4.17 – Вікно допомоги

Додатково для візуалізації розроблено та доопрацьовано Евристичний алгоритм (рис. 4.18)

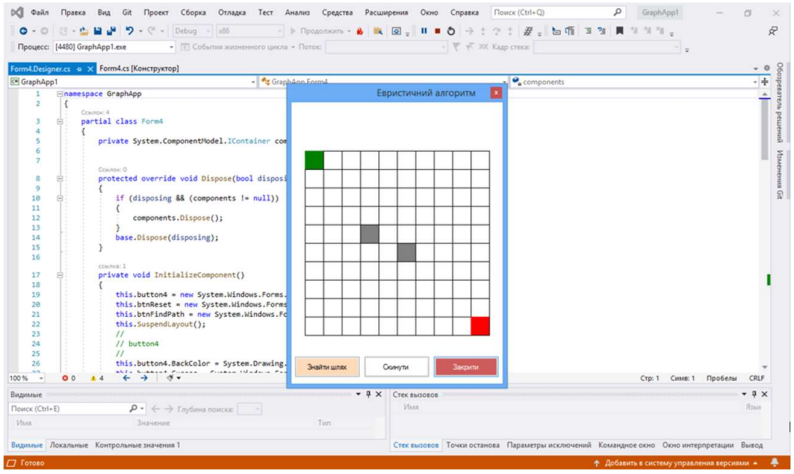

Рисунок 4.18 – Евристичний алгоритм

В кожному алгоритмі додатково оброблено кожну можливу помилку, яка буде виводитись користувачеві у спливаючому сповіщенні. (рис. 4.19)

```
                }
                else
                {
                    message += "Шлях до вершини " + destination + ": недосяжна\n";
                }

                message += "Відстань: " + (distance == int.MaxValue ? "недосяжна" : distance.ToString()) + "\n\n";
            }

            // Виведення результату
            MessageBox.Show(message, "Результати обчислення", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка: " + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    ссылка: 1
    private void button1_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    ссылка: 1
    private void button2_Click(object sender, EventArgs e)
    {
        Form2 Form2 = new Form2();
        Form2.Show();
    }

    ссылка: 1
    private void MainForm_Load(object sender, EventArgs e)
```

Рисунок 4.19 – Обробка помилок

4.3. Опис роботи програмного забезпечення

Після запуску програмного забезпечення, користувачу висвічується вікно, в якому відображається головний інтерфейс додатку з вибором алгоритмів. (рис. 4.20)

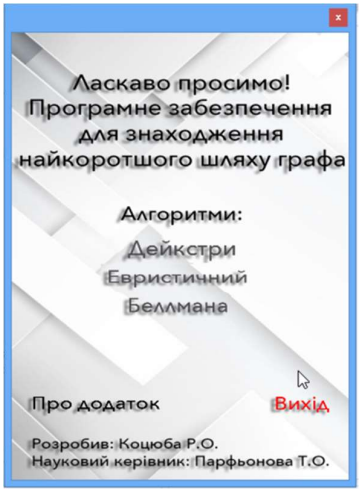

Рисунок 4.20 – Головне вікно додатку

В користувача є можливість подивитися інформацію про додаток, натиснувши кнопку на головному екрані “Про додаток”. (рис. 4.21)

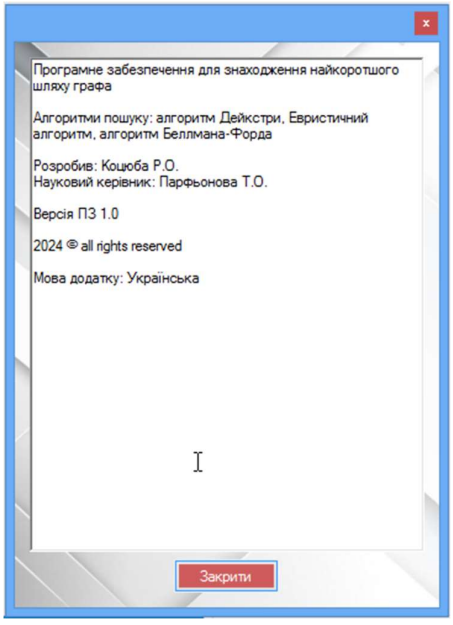

Рисунок 4.21 – Вікно інформації

Перший алгоритм який користувач може обрати – алгоритм Дейкстри. Він має простий та інтуїтивно зрозумілий інтерфейс. (рис. 4.22)

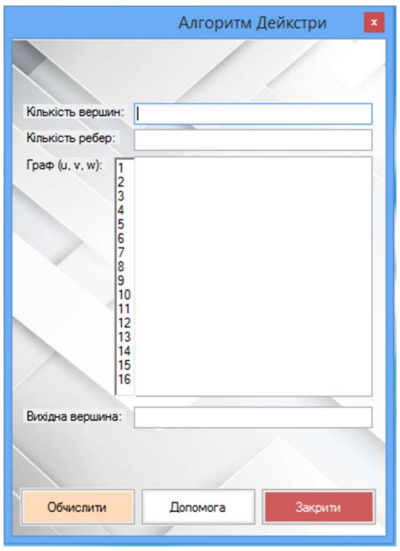

Рисунок 4.22 – Алгоритм Дейкстри

Для того щоб почати роботу з додатком, користувачу необхідно ввести з клавіатури в поля кількість вершин, кількість ребер, граф у форматі u, v, w та натиснути кнопку “Обчислити”. (рис. 4.23)

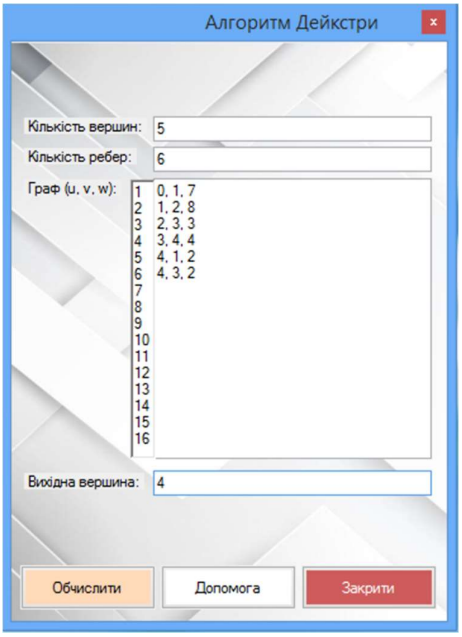

Рисунок 4.23 – Введення вхідних даних

Якщо користувачу необхідна додаткова інструкція, користувач натискає кнопку “Допомога”, де більш детально описано правила введення вхідних даних. (рис. 4.24)

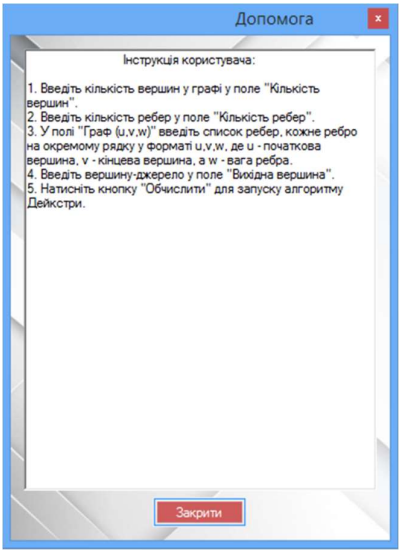

Рисунок 4.24 – Інструкція користувача

В алгоритмі Дейкстрі оброблено додатково помилки. Якщо користувач помилився з введенням вхідних даних, йому відобразиться помилка. (рис. 4.25)

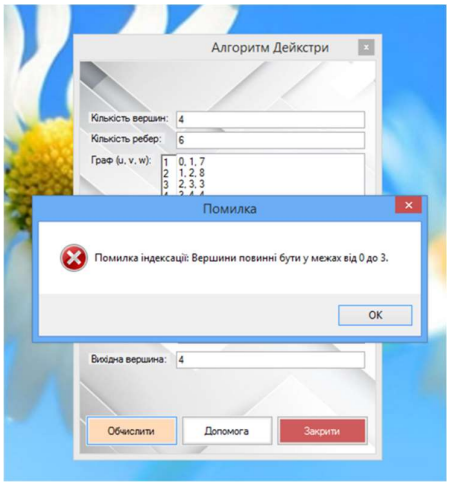

Рисунок 4.25 – Виведення помилки

Ще один приклад помилки з введенням вхідних даних (рис. 4.26)

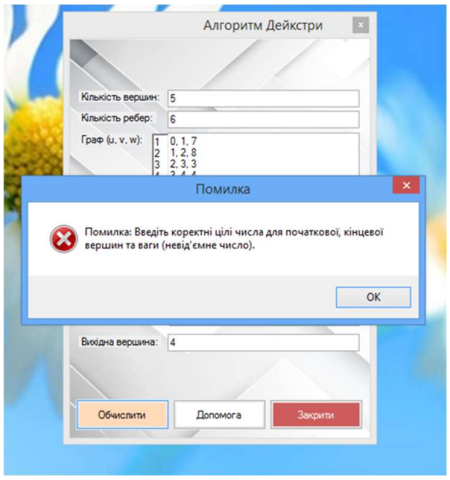

Рисунок 4.26 – Виведення помилки

Наступний алгоритм – Евристичний. Цей алгоритм реалізовано у додатку візуально. Евристичний алгоритм пошуку найкоротшого шляху у двовимірному полі з використанням A-Star. Алгоритм ефективно визначає маршрут від початкової до кінцевої точки, враховуючи перешкоди на карті, що представлені у вигляді сітки. Точки в евристичному методі розташовані на кутах клітинок у сітці, тому що при обчисленнях координат точок позиція вузла (або клітинки) представляється у вигляді центральних координат сіткової системи.

1. Інтерфейс програми дозволяє користувачу задавати карту шляхом встановлення перешкод на сітці через клік миші. Стартова і кінцева точки фіксовані, що забезпечує стабільність пошуку.

2. Візуалізація результатів:

- Поле відображається у вигляді клітинок, де перешкоди позначаються сірим кольором.

- Стартова точка позначена зеленим, кінцева точка — червоним, а знайдений шлях відображається синіми клітинками, що робить результат наочним і зрозумілим.

3. Реалізація алгоритму:

- Алгоритм використовує евристичну функцію, що обчислює манхеттенську відстань до цільової точки, що дозволяє оптимізувати шлях у сітці.

- Під час роботи програма перевіряє сусідні клітинки у чотирьох основних напрямках (вгору, вниз, вліво, вправо) та уникає перешкод.

- Шлях будується шляхом зворотного відстеження (backtracking) від кінцевої точки до початкової через вузли, що мають найменшу вартість шляху.

4. Обробка помилок: якщо шлях не існує (наприклад, коли кінцева точка оточена перешкодами), програма коректно обробляє цей випадок, повертаючи порожній результат.

Таким чином, розроблений застосунок надає функціональність для інтерактивного пошуку найкоротшого шляху на карті з перешкодами, візуалізує результат та дозволяє користувачеві змінювати карту для тестування алгоритму. Евристичний алгоритм забезпечує ефективний і швидкий пошук шляху, що робить програму надійною для розв'язання задач такого типу. (рис. 4.27)

Рисунок 4.27 – Евристичний алгоритм

У користувача є можливість виставити курсором точки, вони грають роль перешкод на шляху, потім при натискання кнопки “Знайти шлях”, алгоритм автоматично прорахує шлях і візуально виведе його в додатку. (рис. 4.28)

Рисунок 4.28 – Евристичний алгоритм

Якщо шлях повністю заблоковано, висвітиться вікно помилки. (рис. 4.29)

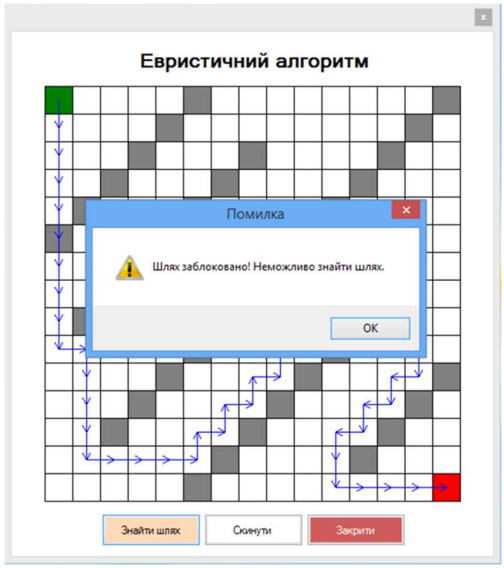

Рисунок 4.29 – Вікно помилки

Також після висвічування вікна на піксельній карті відобразяться хрестики. (рис. 4.30)

Рисунок 4.30 – Хрестики

Третій останній алгоритм додатку – алгоритм Беллмана-Форда. По дизайну він розроблений так само як алгоритм Дейкстри, але запрограмований по алгоритму Беллмана-Форда. (рис. 4.31)

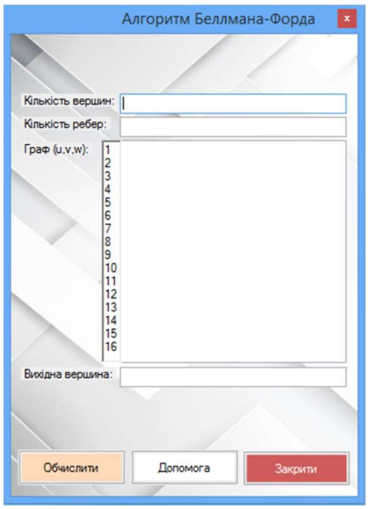

Рисунок 4.31 – Алгоритм Беллмана-Форда

Якщо користувачу необхідна додаткова інструкція, користувач натискає кнопку “Допомога”, де більш детально описано правила введення вхідних даних в алгоритмі Беллмана-Форда. (рис. 4.32)

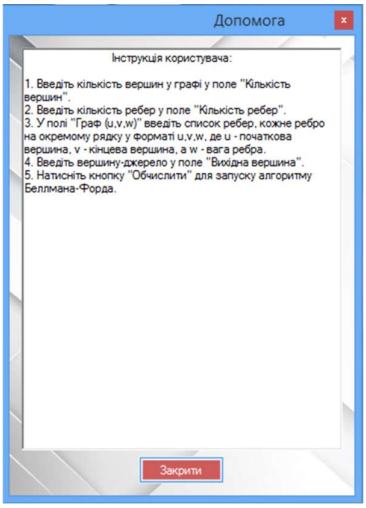

Рисунок 4.32 – Інструкція до алгоритму

Аналогічно, як в алгоритмі Дейкстри, для того щоб почати роботу з додатком, користувачу необхідно ввести з клавіатури в поля кількість вершин, кількість ребер, граф у форматі u, v, w та натиснути кнопку “Обчислити”. (рис. 4.33)

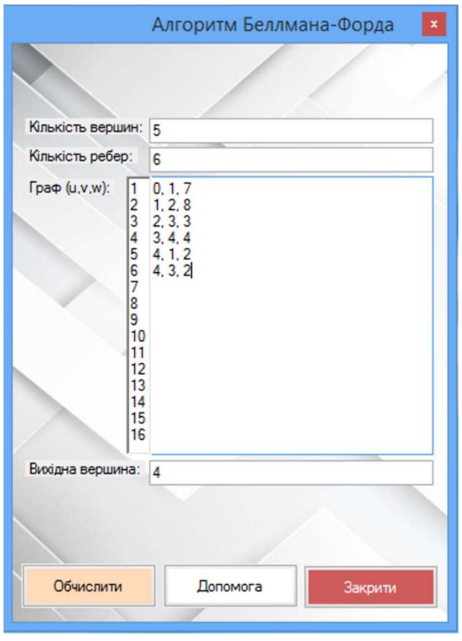

Рисунок 4.33 – Введення вхідних даних

В алгоритмі Беллмана-Форда оброблено велику кількість можливих помилок та багів. Наприклад, якщо користувач помилився при введенні вхідних даних та ввів неправильну кількість ребер, йому висвітиться наступна помилка. (рис. 4.34)

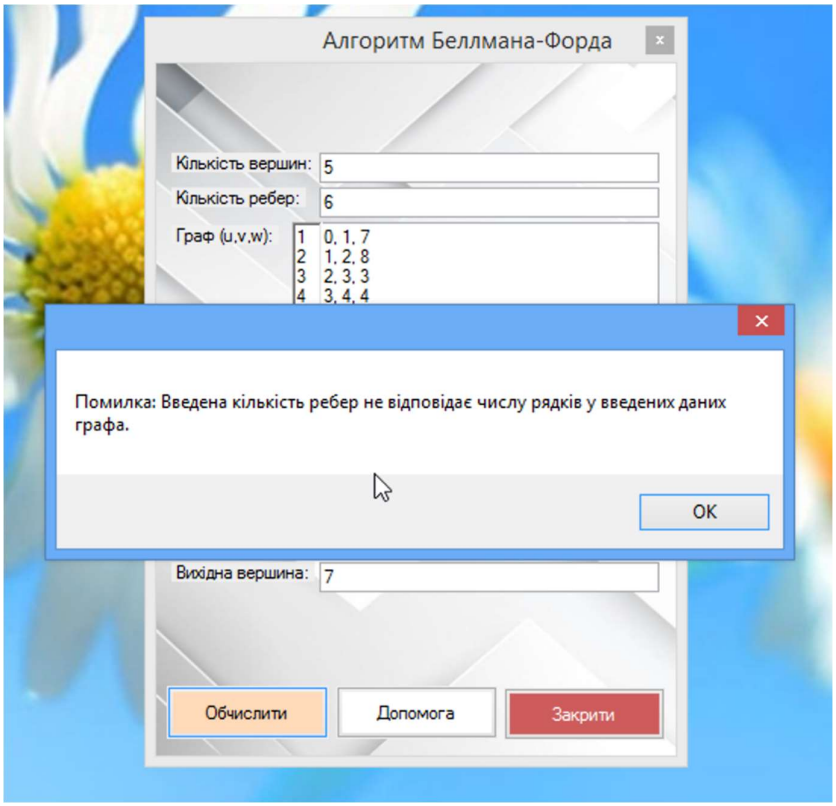

Рисунок 4.34 – Виведення помилки

Також оброблено такий тип помилок, коли користувач або не заповнив поля або недозаповнив їх вхідними даними. (рис. 4.35)

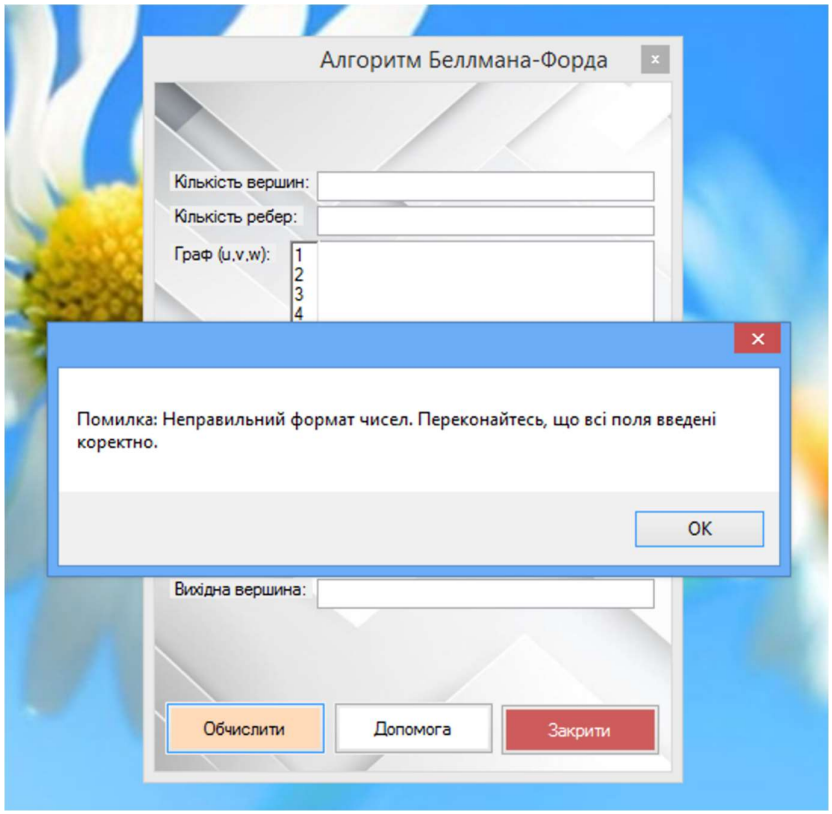

Рисунок 4.35 – Виведення помилки порожніх полів

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено програмне забезпечення знаходження найкоротшого шляху графа з використанням алгоритмів Дейкстри, Беллмана-Форда, Евристичний алгоритм. Застосунок забезпечує інтерфейс для введення даних про кількість вершин графа, ребер у форматі (початкова вершина, кінцева вершина, вага), а також початкової вершини для обчислень.

Головним результатом є можливість отримання детальної інформації про найкоротші шляхи від заданої початкової вершини до всіх інших вершин графа. Результати обчислень виводяться у зрозумілому вигляді, де для кожної вершини показується послідовність вершин (шлях), через які пролягає маршрут, а також загальна відстань. Якщо шлях відсутній, програма вказує про неможливість досягнення конкретної вершини.

Завдяки реалізації алгоритму Дейкстри, програма є ефективною для графів з невід'ємними вагами ребер і надає точні результати. Передбачено обробку виняткових ситуацій, таких як некоректне введення даних, вихід за межі допустимих значень або помилки формату. Таким чином, робота дозволила реалізувати коректний алгоритм пошуку найкоротших шляхів у зручному програмному інтерфейсі з можливістю багаторазового використання та обчислень для різних графів.

Кваліфікаційну роботу було успішно виконано, розроблено зручне програмне забезпечення знаходження найкоротшого шляху графа.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні рекомендації щодо виконання кваліфікаційної роботи студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня магістра [Електронний ресурс]. – http://dspace.puet.edu.ua/handle/123456789/12865
2. Алгоритми пошуку найкоротших шляхів [Електронний ресурс]. – https://www.myrouteonline.com/blog/shortest-path-algorithms
3. Алгоритм Дейкстри [Електронний ресурс]. – Режим доступу: https://www.studysmarter.co.uk/explanations/math/decision-maths/dijkstras-algorithm/
4. Технології дистанційного навчання [Електронний ресурс]. – Режим доступу: https://allreferat.com.ua/uk/pedagogika_metoduka_vukladanny/referat/4324
5. Y. Kim, Y. Kim, S. Lee, J. Kang, and J. An, "Portable Fire Evacuation Guide Robot System," pp. 2789–2794.
6. Modification-of-Dijkstras-algorithm [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/profile/Abd-Samad-Basari/publication/282187479_Modification_of_Dijkstra's_algorithm_for_safest_and_shortest_path_during_emergency_evacuation/links/5d01aec5299bf13a38510b97/Modification-of-Dijkstras-algorithm-for-safest-and-shortest-path-during-emergency-evacuation.pdf
7. Апаратне й програмне забезпечення мереж. [Електронний ресурс]. – Режим доступу: https://informatik.pp.ua/uroky/9-klas/prezentatsii/urok-4-aparatne-i-prohramne-zabezpechennia-merezh-adresatsiia-v-merezhakh/
8. Y. Sato and Y. Osana, "Office Layout Plan Evaluation System Using Evacuation Simulation Considering Other Agents' Action," in IEEE International Conference on Systems, Man, and Cybernetics, vol. 2, pp. 1911–1916.
9. T. Meilinger, J. Frankenstein, and H. H. Bülthoff, "Learning to Navigate: Experience Versus Maps," Cognition, vol. 129, no. 1, pp. 24–30, Oct. 2013.

10. C. Hölscher, T. Tenbrink, and J. M. Wiener, “Would You Follow Your Own Route Description? Cognitive Strategies in Urban Route Planning,” Cognition, vol. 121, no. 2, pp. 228–47.
11. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання: ДСТУ 7.1-2006. – [Чинний від 2007-07-01]. – К. : Держспоживстандарт України, 2007. – 47 с.
12. Microsoft Visual Studio [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
13. C# development with Visual Studio [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/?view=vs-2022
14. C# Get Started [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/cs/cs_getstarted.php
15. Visual Studio and C# [Електронний ресурс]. – Режим доступу: https://www.halvorsen.blog/documents/programming/csharp/csharp.php
16. Створення програми на C# в Visual Studio 2015 [Електронний ресурс]. – Режим доступу: https://learn.ztu.edu.ua/mod/page/view.php?id=9976
17. Getting Started With C# on Visual Studio Code [Електронний ресурс]. – Режим доступу: https://medium.com/@adebiyiadedotun9/getting-started-with-c-on-visual-studio-code-5a76dcca1110
18. Randolph Visual Studio 2010 for professionals // Randolph, Nick, Gardner, David, Minutillo, Michael, Anderson, Chris.: Trans. with English - M.: LLC "I.D. Williams", 2011. - 1184 p.
19. ReSharper: The Visual Studio Extension for .NET [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/resharper/
20. Install Visual Studio [Електронний ресурс]. – Режим доступу: https://www.csharptutorial.net/csharp-tutorial/install-visual-studio/

ДОДАТОК А. КОД ПРОГРАМИ