

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та
інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: «Розроблення програмного застосунку для візуалізації та аналізу
графів у навчальній дисципліні «Дискретна математика»»

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні системи
та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_2022
Коваленко Максим Віталійович
Керівник: Одарущенко Олена Борисівна
Рецензент: Стрюк Олексій Юрійович

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та
інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ **Юрій УТКІН**
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Коваленка Максима Віталійовича

1. Тема кваліфікаційної роботи: Розроблення програмного застосунку для візуалізації та аналізу графів у навчальній дисципліні «Дискретна математика»»,

Керівник роботи: к. т. н., доцент, доцент кафедри інформаційних систем та технологій Одарущенко Олена Борисівна.

Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 383-ст

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року.

3. Вихідні дані до роботи: наукова та навчальна література з теорії графів і дискретної математики; документація мови програмування Python: <https://docs.python.org/>, документація бібліотеки NetworkX: <https://networkx.org/documentation/>, документація бібліотеки matplotlib: <https://matplotlib.org/stable/>, інтегроване середовище розробки PyCharm: <https://www.jetbrains.com/pycharm/>

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

Розділ 1 Теоретичні основи теорії графів

Розділ 2. Проектування і реалізація програмного застосунку

Розділ 3. Тестування та аналіз роботи програмного застосунку

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об'єктом дослідження

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Дивнич О. Д., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Оцінювання економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р. - 18.06.2026 р.	

Здобувач вищої освіти

Максим КОВАЛЕНКО

Керівник роботи

Олена ОДАРУЩЕНКО

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	6
ВСТУП.....	7
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ТЕОРІЇ ГРАФІВ	9
1.1 Поняття графа та його основні характеристики	9
1.2 Способи представлення графів.....	11
1.3 Основні алгоритми роботи з графами.....	13
РОЗДІЛ 2 ПРОЄКТУВАННЯ І РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ	16
2.1 Вибір засобів розробки програмного застосунку	16
2.2 Архітектура, принцип роботи та графічний інтерфейс програмного застосунку	17
2.3 Реалізація взаємодії з графами та алгоритмів їх обробки.....	24
2.4 Формування матриці суміжності та збереження даних	33
РОЗДІЛ 3 ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ	37
3.1 Тестування функціональних можливостей програмного застосунку	37
3.2 Аналіз результатів роботи програмного застосунку	39
3.3 Переваги та перспективи розвитку програмного застосунку.....	41
3.4 Оцінка вартості розробки програмного застосунку	43
ВИСНОВКИ.....	44
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	46
ДОДАТКИ.....	49

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BFS – Breadth-First Search.

CSV – Comma-Separated Values.

DFS – Depth-First Search.

GUI – Graphical User Interface.

JSON – JavaScript Object Notation.

PNG – Portable Network Graphics.

UI – User Interface.

UX – User Experience.

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій важливого значення набувають методи обробки, аналізу та візуалізації даних. Одним із фундаментальних розділів дискретної математики, що широко застосовується в інформатиці та суміжних галузях, є теорія графів [1]. Графові моделі використовуються для опису та дослідження складних систем, у яких необхідно враховувати взаємозв'язки між окремими елементами. До таких систем належать комп'ютерні мережі, транспортні системи, соціальні мережі, інформаційні структури, а також різноманітні задачі оптимізації.

У процесі вивчення теорії графів студенти стикаються з рядом труднощів, пов'язаних із високим рівнем абстрактності матеріалу. Значна частина понять і алгоритмів, що розглядаються в межах дисципліни «Дискретна математика», складно сприймається без наочного представлення. Зокрема, це стосується таких задач, як обходи графів, пошук найкоротших шляхів, аналіз зв'язності та інші алгоритмічні процедури. Відсутність зручних інструментів для візуалізації ускладнює розуміння логіки роботи алгоритмів і знижує ефективність навчального процесу.

У зв'язку з цим виникає необхідність створення програмних засобів, які б забезпечували можливість інтерактивної роботи з графами, їх візуального представлення та аналізу. Використання таких інструментів дозволяє не лише покращити засвоєння теоретичного матеріалу, але й сприяє формуванню практичних навичок роботи з алгоритмами та структурами даних.

Актуальність теми кваліфікаційної роботи полягає у необхідності розробки програмного застосунку, який поєднує можливості побудови графів, їх візуалізації та виконання основних алгоритмів аналізу, що використовуються в курсі «Дискретна математика». Особливу увагу доцільно приділити використанню сучасних мов програмування та бібліотек, які дозволяють ефективно реалізувати зазначений функціонал.

Мета роботи – розробка Python-застосунку для візуалізації та аналізу графів, який може бути використаний як допоміжний інструмент у навчальному процесі. Для досягнення поставленої мети необхідно вирішити ряд задач, що включають аналіз теоретичних основ теорії графів, дослідження існуючих алгоритмів їх обробки, вибір інструментальних засобів розробки, проектування структури програмного застосунку та реалізацію його основного функціоналу.

Об’єкт дослідження – процес використання графових моделей у навчанні та аналізі дискретних структур.

Предмет дослідження – методи візуалізації та алгоритмічного аналізу графів, а також програмні засоби їх реалізації.

Методи дослідження: аналіз наукових та навчальних джерел, узагальнення існуючих підходів до представлення графів, а також методи програмування та моделювання для реалізації функціональних можливостей застосунку.

Інформаційну базу дослідження становили наукові та навчальні джерела з дискретної математики та теорії графів, документація мови програмування Python, матеріали офіційної документації бібліотек NetworkX та matplotlib, а також інформаційні ресурси, присвячені розробці програмного забезпечення для аналізу та візуалізації графових структур.

Практична значущість отриманих результатів полягає у створенні програмного продукту, який може бути використаний у навчальному процесі для демонстрації роботи алгоритмів на графах, а також для самостійного вивчення відповідного матеріалу студентами.

Структура роботи включає вступ, три розділи, висновки та список використаних джерел. Обсяг роботи становить 61 сторінку, 3 таблиці, 6 рисунків, 2 додатки.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ТЕОРІЇ ГРАФІВ

1.1 Поняття графа та його основні характеристики

Теорія графів є одним із фундаментальних розділів дискретної математики, що досліджує властивості графових структур, їх формальне представлення та методи застосування для розв'язання широкого кола теоретичних і прикладних задач. Її розвиток пов'язаний із необхідністю опису складних систем, у яких ключову роль відіграють не лише самі елементи, а й зв'язки між ними. Саме можливість формалізувати такі зв'язки зробила графи універсальним інструментом у сучасній науці та техніці.

У сучасному світі графи широко застосовуються для моделювання різноманітних процесів і структур. Вони використовуються в інформатиці для опису комп'ютерних мереж і структур даних, у транспортних системах для моделювання маршрутів і потоків, у соціології для аналізу соціальних зв'язків, а також у багатьох інших галузях. Така універсальність пояснюється тим, що практично будь-яку систему, яка складається з об'єктів та зв'язків між ними, можна представити у вигляді графа.

Граф у загальному вигляді визначається як математична структура, що складається з множини вершин і множини ребер, які з'єднують ці вершини між собою [2]. Вершини графа інтерпретуються як окремі об'єкти або елементи системи, тоді як ребра відображають взаємозв'язки між ними. У деяких випадках ребра можуть мати додаткові характеристики, що дозволяє більш точно описувати властивості зв'язків.

Однією з ключових особливостей графів є різноманітність їх типів, що визначається характером зв'язків між вершинами. Зокрема, важливим є поділ графів на орієнтовані та неорієнтовані. В орієнтованих графах кожне ребро має напрямок, що задає впорядкований перехід від однієї вершини до іншої. Такий тип графів використовується для моделювання процесів, у яких важливим є напрям

руху або залежності, наприклад у мережах передачі даних або системах керування. У неорієнтованих графах зв'язки є взаємними, що означає відсутність напрямку, і такі графи застосовуються для опису симетричних відношень.

Ще однією важливою характеристикою є наявність ваг у ребер. У зважених графах кожному ребру ставиться у відповідність числове значення, яке може відображати довжину, вартість, час або інші параметри [3]. Це дозволяє використовувати графи для розв'язання задач оптимізації, зокрема пошуку найкоротших шляхів або мінімізації витрат. Незважені графи, у свою чергу, використовуються у випадках, коли всі зв'язки вважаються рівнозначними, і важливим є лише сам факт їх існування.

Окрім зазначених характеристик, важливими є також такі властивості графів, як ступінь вершини, зв'язність та наявність циклів. Ступінь вершини визначає кількість ребер, що інцидентні до неї, і є важливим показником при аналізі структури графа. Зв'язність графа характеризує можливість досягнення будь-якої вершини з будь-якої іншої, що має значення при дослідженні цілісності системи. Наявність циклів визначає, чи існують у графі замкнені шляхи, що повертаються до початкової вершини, і це також впливає на вибір методів аналізу [4].

Графи можуть мати додаткові властивості, такі як планарність, регулярність або дерево-подібна структура, що дозволяє класифікувати їх за різними критеріями та обирати відповідні методи дослідження. Наприклад, дерева є окремим класом графів, які не містять циклів і мають особливе значення у багатьох алгоритмічних задачах.

Таким чином, поняття графа є базовим для розуміння великої кількості процесів і структур, що використовуються в сучасній інформатиці та інших галузях. Різноманітність типів графів та їх властивостей дозволяє ефективно моделювати складні системи та застосовувати математичні методи для їх аналізу, а також створювати програмні засоби, орієнтовані на дослідження та візуалізацію відповідних структур. Використання графових моделей сприяє більш глибокому розумінню взаємозв'язків між елементами системи та забезпечує можливість формалізації задач, що виникають у практичній діяльності. Крім того, графи є

зручним інструментом для подальшого застосування алгоритмічних підходів, що робить їх важливою складовою сучасних інформаційних технологій.

1.2 Способи представлення графів

Для практичного використання графів у програмуванні необхідно визначити спосіб їх представлення у пам'яті комп'ютера. Від вибору відповідної структури даних значною мірою залежить ефективність виконання алгоритмів, швидкість обробки інформації та обсяг використаних ресурсів. Саме тому питання представлення графів є одним із ключових при розробці програмних застосунків, що працюють із графовими структурами.

Існує декілька підходів до представлення графів, кожен із яких має свої переваги та недоліки. Найбільш поширеними є матриця суміжності та список суміжності, які відрізняються за способом зберігання інформації про зв'язки між вершинами.

Матриця суміжності являє собою двовимірну структуру даних, у якій рядки та стовпці відповідають вершинам графа. Значення елемента матриці визначає наявність або відсутність ребра між відповідними вершинами [5]. У випадку зваженого графа в матриці можуть зберігатися ваги ребер, що дозволяє використовувати її для реалізації алгоритмів пошуку оптимальних шляхів. Такий спосіб представлення є інтуїтивно зрозумілим і зручним для реалізації, оскільки дозволяє безпосередньо перевіряти існування зв'язку між будь-якими двома вершинами за сталий час.

Разом із тим використання матриці суміжності має суттєві обмеження, пов'язані з витратами пам'яті. Розмір матриці зростає пропорційно квадрату кількості вершин, що робить цей підхід неефективним для великих графів, особливо у випадках, коли кількість ребер є значно меншою за максимально можливу. Такі графи називають розрідженими, і для них доцільніше використовувати інші способи представлення.

Альтернативним підходом є список суміжності, який передбачає зберігання для кожної вершини переліку всіх суміжних з нею вершин. У цьому випадку структура даних організовується таким чином, що кожна вершина має пов'язаний із нею список або іншу колекцію, яка містить інформацію про її сусідів. Такий спосіб дозволяє зберігати лише існуючі зв'язки, що значно зменшує витрати пам'яті порівняно з матрицею суміжності.

Список суміжності є більш ефективним для роботи з великими та розрідженими графами, оскільки дозволяє швидко отримувати інформацію про суміжні вершини та виконувати обходи графа. Водночас перевірка наявності конкретного ребра між двома вершинами може потребувати додаткового часу, оскільки необхідно переглянути відповідний список.

Окрім матриці та списку суміжності, існують і інші способи представлення графів. Зокрема, іноді використовується список ребер, який містить перелік усіх зв'язків у графі. Такий підхід є простим у реалізації та зручним для деяких задач, однак він не завжди забезпечує швидкий доступ до інформації про суміжні вершини.

Таблиця 1.1 – Порівняння способів представлення графів

Спосіб представлення	Переваги	Недоліки
Матриця суміжності	Простота реалізації, швидка перевірка зв'язків	Великі витрати пам'яті
Список суміжності	Економне використання пам'яті	Повільніша перевірка наявності ребра
Список ребер	Простота структури	Незручність аналізу суміжних вершин

У більш складних випадках можуть застосовуватися комбіновані або спеціалізовані структури даних, які поєднують переваги різних підходів. Наприклад, у деяких алгоритмах використовуються додаткові структури для зберігання інформації про ваги ребер, попередників або інші характеристики, що необхідні для ефективного виконання обчислень.

Вибір конкретного способу представлення графа залежить від багатьох факторів, серед яких розмір графа, щільність зв'язків, тип задачі та вимоги до

продуктивності. У практичних застосуваннях часто використовуються різні підходи залежно від конкретної ситуації, що дозволяє досягти оптимального співвідношення між швидкістю роботи та використанням ресурсів.

Тому, способи представлення графів є важливим елементом їх програмної реалізації. Розуміння особливостей кожного підходу дозволяє обрати найбільш ефективне рішення для конкретної задачі та забезпечити високу продуктивність програмного застосунку.

1.3 Основні алгоритми роботи з графами

Алгоритми обробки графів є невід'ємною складовою теорії графів і відіграють ключову роль у її практичному застосуванні. Вони дозволяють виконувати аналіз структури графа, визначати взаємозв'язки між його елементами, знаходити оптимальні рішення та розв'язувати широкий спектр задач, що виникають у різних галузях науки і техніки. Розробка та використання ефективних алгоритмів є основою для створення програмних систем, які працюють із графовими структурами.

Одними з базових алгоритмів є алгоритми обходу графа, які дозволяють послідовно відвідати всі вершини та дослідити структуру графа. До таких алгоритмів належать обхід у ширину та обхід у глибину, які мають різні принципи роботи та застосовуються залежно від поставленої задачі [6].

Обхід у ширину передбачає поступове дослідження графа за рівнями, починаючи з початкової вершини. Спочатку відвідуються всі вершини, безпосередньо пов'язані з початковою, після чого відбувається перехід до вершин наступного рівня. Реалізація цього алгоритму зазвичай здійснюється з використанням черги, що дозволяє зберігати порядок обробки вершин і гарантує правильну послідовність їх відвідування.

Обхід у глибину, на відміну від обходу у ширину, базується на принципі максимального заглиблення в структуру графа. Алгоритм переходить від поточної

вершини до однієї з суміжних і продовжує цей процес доти, доки це можливо. Після досягнення вершини, з якої неможливо виконати подальший перехід, відбувається повернення до попередньої вершини для дослідження інших можливих шляхів. Такий підхід дозволяє ефективно аналізувати структуру графа та використовується для розв'язання задач, пов'язаних із пошуком циклів, визначенням компонент зв'язності та побудовою топологічного порядку [7].

Важливою особливістю алгоритмів обходу є їх відносно невисока обчислювальна складність, що робить їх ефективними навіть при роботі з великими графами. У більшості випадків їх складність визначається кількістю вершин і ребер, що забезпечує можливість їх використання у практичних програмних системах.

Окрім алгоритмів обходу, значне місце займають алгоритми пошуку найкоротших шляхів, які дозволяють знаходити оптимальні маршрути між вершинами графа. Одним із найбільш відомих і широко застосовуваних алгоритмів цього типу є алгоритм Дейкстри. Він використовується для знаходження найкоротших відстаней від однієї вершини до всіх інших у зваженому графі з невід'ємними вагами ребер.

Принцип роботи алгоритму Дейкстри полягає у поступовому визначенні найменших відстаней до вершин графа [8]. На кожному кроці обирається вершина з мінімальним поточним значенням відстані, після чого виконується оновлення відстаней до суміжних вершин. Такий підхід дозволяє послідовно уточнювати результати та забезпечує отримання оптимального рішення.

Ефективність алгоритму значною мірою залежить від способу його реалізації. Використання додаткових структур даних, таких як пріоритетні черги, дозволяє значно зменшити час виконання та підвищити продуктивність при роботі з великими графами. У загальному випадку складність алгоритму визначається кількістю вершин і ребер та може варіюватися залежно від обраної реалізації.

Алгоритм Дейкстри широко застосовується у практичних задачах, зокрема в системах навігації, мережевих протоколах, логістичних розрахунках та інших сферах, де необхідно визначати оптимальні шляхи. Водночас важливо враховувати,

що він коректно працює лише у випадку, коли всі ваги ребер є невід’ємними. У протилежному випадку необхідно використовувати інші алгоритми.

Таблиця 1.2 – Порівняння алгоритмів роботи з графами

Алгоритм	Призначення	Особливості
BFS	Обхід графа в ширину	Використовується для пошуку найкоротших шляхів у незважених графах
DFS	Обхід графа в глибину	Дозволяє аналізувати структуру графа та знаходити цикли
Алгоритм Дейкстри	Пошук найкоротшого шляху	Працює зі зваженими графами

Порівняння основних алгоритмів роботи з графами дозволяє визначити особливості їх використання залежно від типу задачі та структури графа. Кожен із алгоритмів має власні переваги та застосовується для розв’язання певного класу задач у сфері аналізу графових структур.

У контексті розробки програмного застосунку для візуалізації та аналізу графів реалізація зазначених алгоритмів має особливе значення. Вона дозволяє не лише виконувати обчислення, але й демонструвати процес їх роботи у наочній формі, що сприяє кращому розумінню теоретичного матеріалу. Візуалізація алгоритмів обходу та пошуку шляхів дозволяє користувачу спостерігати послідовність обробки вершин, що значно підвищує ефективність навчання.

Сучасні алгоритми аналізу графів активно використовуються у різних сферах інформаційних технологій. Вони застосовуються у системах навігації, аналізі комп’ютерних мереж, соціальних сервісах, логістичних системах, пошукових алгоритмах та інших програмних рішеннях, де необхідно виконувати аналіз взаємозв’язків між об’єктами. Завдяки високій ефективності алгоритми роботи з графами залишаються важливим інструментом сучасного програмування та обробки даних.

Отже, алгоритми роботи з графами є важливим інструментом для аналізу складних систем і становлять основу для створення програмних засобів, що використовуються як у наукових дослідженнях, так і в освітньому процесі.

РОЗДІЛ 2

ПРОЄКТУВАННЯ І РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Вибір засобів розробки програмного застосунку

Одним із важливих етапів розробки програмного застосунку є вибір інструментальних засобів, які будуть використовуватися для його реалізації. Від правильності цього вибору залежить ефективність розробки, продуктивність програмного продукту, а також можливість його подальшого розширення та підтримки. У сучасних умовах існує велика кількість мов програмування та бібліотек, які можуть бути використані для роботи з графовими структурами, що потребує обґрунтованого підходу до їх вибору.

Для реалізації програмного застосунку, призначеного для візуалізації та аналізу графів, було обрано мову програмування Python. Даний вибір зумовлений рядом переваг цієї мови, серед яких варто відзначити простоту синтаксису, що дозволяє швидко реалізовувати алгоритми, високу читабельність коду, а також наявність великої кількості бібліотек, орієнтованих на обробку даних та наукові обчислення [9]. Python широко використовується у сфері освіти та досліджень, що робить його доцільним вибором для створення навчальних застосунків. Для розробки програмного застосунку було використано інтегроване середовище розробки PyCharm, яке забезпечує зручні засоби написання, налагодження та тестування програмного коду.

Однією з ключових переваг Python є підтримка спеціалізованих бібліотек для роботи з графами. Зокрема, бібліотека NetworkX надає широкий набір інструментів для створення, модифікації та аналізу графових структур. Вона дозволяє легко реалізовувати основні операції над графами, такі як додавання вершин і ребер, визначення їх властивостей, а також застосування алгоритмів обходу та пошуку шляхів [10]. Використання цієї бібліотеки значно спрощує процес розробки та дозволяє зосередитися на реалізації функціональних можливостей застосунку.

Для візуалізації графів доцільно використовувати бібліотеку `matplotlib`, яка забезпечує можливість побудови графічних зображень на основі даних. Вона дозволяє відображати структуру графа у зрозумілому вигляді, що є особливо важливим для навчального застосування [11]. Використання засобів візуалізації дозволяє наочно демонструвати роботу алгоритмів, що сприяє кращому засвоєнню матеріалу.

Крім того, Python має розвинену екосистему інструментів для розробки програмного забезпечення, включаючи інтегровані середовища розробки, системи керування версіями та засоби тестування. Це дозволяє організувати процес розробки на високому рівні та забезпечити якість кінцевого продукту.

Вибір Python як основної мови програмування також обумовлений його кросплатформеністю, що дозволяє запускати розроблений застосунок на різних операційних системах без необхідності внесення значних змін у код. Це є важливою перевагою, особливо у випадку створення навчального програмного забезпечення.

Отже, обрані засоби розробки забезпечують необхідний функціонал для створення програмного застосунку, який дозволяє виконувати візуалізацію та аналіз графів. Використання мови програмування Python разом із відповідними бібліотеками дозволяє ефективно реалізувати поставлені задачі та створити зручний інструмент для навчального процесу.

2.2 Архітектура, принцип роботи та графічний інтерфейс програмного застосунку

Під час розробки програмного застосунку для візуалізації та аналізу графів особлива увага приділялася побудові зручної та зрозумілої архітектури системи. Правильно організована структура програмного забезпечення забезпечує стабільність роботи застосунку, спрощує процес його розробки та дозволяє легко розширювати функціональні можливості у майбутньому. Архітектура програмного

застосунку була побудована з урахуванням необхідності реалізації графічного інтерфейсу, роботи з графовими структурами та виконання алгоритмів аналізу графів.

Оснoву програмного застосунку становить модульна структура, у якій окремі компоненти відповідають за виконання конкретних функцій. Такий підхід дозволяє розділити логіку роботи програми на незалежні частини, що значно спрощує підтримку програмного коду та його подальшу модифікацію [12]. Кожен модуль виконує окремий набір задач, пов'язаних із побудовою графа, обробкою даних, візуалізацією або реалізацією алгоритмів.

Центральним елементом системи є модуль роботи з графом. Даний компонент відповідає за створення графових структур, додавання вершин та ребер, зміну параметрів графа, а також збереження інформації про зв'язки між вершинами. Для реалізації графової структури використовується бібліотека NetworkX, яка забезпечує набір готових засобів для роботи з графами різних типів.

У процесі роботи програми користувач взаємодіє із графічним інтерфейсом, який забезпечує передачу команд до відповідних функціональних модулів. Після виконання певної дії, наприклад запуску алгоритму або створення графа, програма виконує обробку даних та оновлює графічне представлення результатів. Такий підхід забезпечує інтерактивність застосунку та дозволяє користувачу спостерігати зміни у структурі графа у режимі реального часу.

Одним із ключових компонентів архітектури є модуль реалізації алгоритмів аналізу графів. Даний модуль забезпечує виконання алгоритмів обходу графа в ширину, обходу в глибину та алгоритму Дейкстри для пошуку найкоротшого шляху. Реалізація алгоритмів у вигляді окремого функціонального компонента дозволяє спростити структуру програмного коду та забезпечує можливість подальшого додавання нових алгоритмів без необхідності суттєвої зміни інших частин системи.

Після запуску алгоритму програма отримує структуру графа та виконує необхідні обчислення відповідно до обраного методу аналізу. Отримані результати передаються до модуля візуалізації, який відповідає за графічне відображення

графа та результатів роботи алгоритмів. У процесі візуалізації можуть змінюватися кольори вершин, виділятися певні ребра або відображатися знайдені маршрути.

Важливим компонентом архітектури є модуль візуалізації графа. Його основним призначенням є побудова графічного представлення графових структур та оновлення зображення під час взаємодії користувача із програмою. Для реалізації візуалізації використовується бібліотека `matplotlib`, яка дозволяє створювати графічні об'єкти та забезпечує їх відображення у вікні застосунку.

Принцип роботи програмного застосунку полягає у послідовній взаємодії між користувачем, графічним інтерфейсом та функціональними модулями системи. Після запуску програми користувач може створювати граф, додавати вершини та ребра, запускати алгоритми аналізу та виконувати інші операції. Програма обробляє введені дані, виконує необхідні обчислення та відображає результати у графічному вигляді.

Взаємодія користувача із програмним застосунком реалізована у інтерактивному режимі, що дозволяє оперативно змінювати структуру графа та миттєво отримувати результати виконання алгоритмів. Під час роботи користувач має можливість редагувати графову структуру, переміщувати вершини, задавати ваги ребер та виконувати аналіз побудованого графа за допомогою реалізованих алгоритмів.

Обробка даних у програмному застосунку здійснюється поетапно. Після отримання дій користувача система виконує оновлення структури графа, обробляє зміни та формує нове графічне представлення елементів. Такий підхід забезпечує стабільність роботи програмного застосунку та коректне відображення результатів аналізу графових структур.

Крім основних алгоритмів аналізу графів, застосунок підтримує функції генерації випадкових графів, формування матриці суміжності, збереження та завантаження графових структур, а також експорт результатів роботи у різних форматах. Наявність таких функцій розширює можливості використання програмного застосунку та підвищує його практичну цінність.

Реалізований принцип взаємодії між окремими компонентами системи забезпечує зручність роботи користувача та дозволяє ефективно використовувати програмний застосунок для аналізу та візуалізації графових структур.

На рисунку 2.1 наведено загальний вигляд програмного застосунку та основні елементи його інтерфейсу.

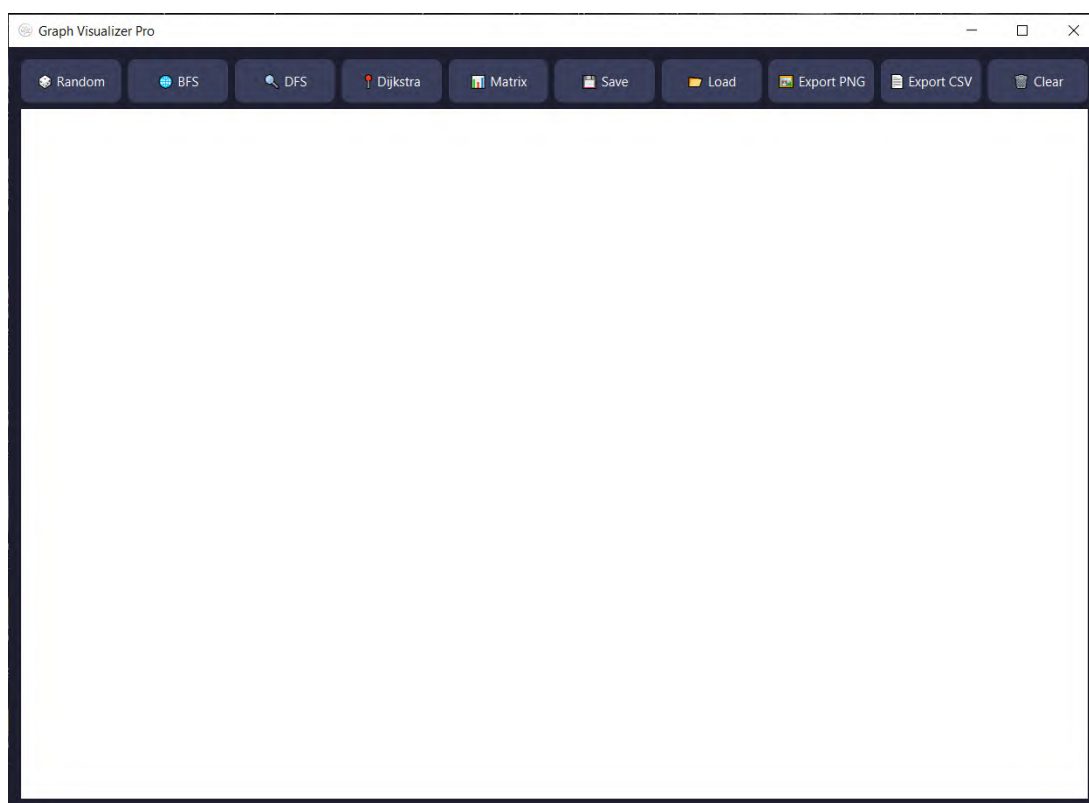


Рисунок 2.1 – Загальний вигляд програмного застосунку

На рисунку представлено головне вікно застосунку, яке містить панель керування, кнопки запуску алгоритмів та область візуалізації графа. Інтерфейс забезпечує швидкий доступ до основних функціональних можливостей системи та дозволяє виконувати аналіз графів у зручному графічному середовищі.

Таким чином, реалізована архітектура програмного застосунку забезпечує ефективну взаємодію між окремими компонентами системи, дозволяє виконувати аналіз графів, підтримує стабільну обробку даних, підвищує зручність використання програмного продукту та забезпечує наочне й зручне графічне представлення результатів роботи алгоритмів.

Графічний інтерфейс користувача є важливою складовою будь-якого програмного застосунку, оскільки саме він забезпечує взаємодію користувача із системою та визначає зручність роботи із програмою. У процесі розробки застосунку для візуалізації та аналізу графів особлива увага приділялася створенню простого, зрозумілого та функціонального інтерфейсу, який дозволяє ефективно працювати з графовими структурами та алгоритмами їх аналізу.

Під час проектування інтерфейсу було враховано необхідність швидкого доступу до основних функцій програми. Усі елементи керування розташовані у верхній частині головного вікна, що забезпечує зручність взаємодії користувача із застосунком. Такий підхід дозволяє мінімізувати кількість зайвих дій під час роботи та забезпечує швидкий запуск основних функцій аналізу графів.

Інтерфейс програмного застосунку реалізовано у вигляді єдиного робочого вікна, яке складається із панелі керування та області візуалізації графа. Панель керування містить набір кнопок для виконання основних операцій над графом, а центральна область використовується для побудови та відображення графових структур.

Для оформлення інтерфейсу було використано темну кольорову схему. Використання темних кольорів дозволяє покращити сприйняття графічних елементів та створює сучасний вигляд програмного застосунку [13]. Крім того, темна тема дозволяє зменшити навантаження на зір користувача під час тривалої роботи із програмою. Робоча область графа виконана у світлому кольорі, що забезпечує контрастність між фоном та графічними елементами графа.

Кнопки керування мають єдиний стиль оформлення та містять текстові позначення разом із графічними іконками. Такий підхід покращує сприйняття функціональних можливостей програми та дозволяє швидше орієнтуватися в інтерфейсі [14]. Для кожної кнопки реалізовано окрему функцію, що забезпечує виконання відповідної операції над графом. Завдяки цьому кожен елемент керування відповідає за виконання визначеної дії, що забезпечує коректну роботу програмного застосунку та підвищує зручність взаємодії користувача з графовою структурою.

У програмному застосунку реалізовано кнопку генерації випадкового графа, яка дозволяє автоматично створювати графові структури із довільним розташуванням вершин та ребер. Наявність такої функції спрощує процес тестування алгоритмів та дозволяє швидко отримувати приклади графів для аналізу.

Окремі кнопки призначені для запуску алгоритмів обходу графа в ширину та в глибину. Після запуску алгоритму користувач може спостерігати зміну кольорів вершин та ребер, що забезпечує наочне представлення процесу обходу графа. Така візуалізація дозволяє краще зрозуміти принципи роботи алгоритмів та підвищує ефективність вивчення теорії графів.

Для пошуку найкоротших шляхів у програмі реалізовано кнопку запуску алгоритму Дейкстри. Після виконання алгоритму програма виділяє знайдений маршрут на графі, що дозволяє користувачу наочно оцінити результати роботи алгоритму.

Інтерфейс також містить кнопку відображення матриці суміжності графа. Використання матричного представлення дозволяє аналізувати структуру графа у табличному вигляді та забезпечує додатковий спосіб дослідження зв'язків між вершинами.

Процес створення графа у програмному застосунку реалізовано таким чином, щоб користувач міг швидко формувати необхідну графову структуру без виконання складних додаткових дій. Інтерактивний підхід до створення вершин та ребер спрощує роботу із системою та забезпечує більш зручну взаємодію із програмним застосунком.

Важливим елементом інтерфейсу є реалізація функцій збереження та завантаження графів. Користувач має можливість зберігати створені графові структури у файл та повторно відкривати їх у програмі. Це дозволяє спростити процес роботи із застосунком та забезпечує можливість повторного використання результатів.

Крім того, у програмі реалізовано функції експорту даних у формати PNG та CSV. Експорт у формат PNG дозволяє зберігати графічне представлення графа у

вигляді зображення [15], а експорт у формат CSV забезпечує можливість подальшої обробки даних у табличних редакторах або інших програмних засобах [16].

На рисунку 2.2 наведено приклад роботи програмного застосунку із побудованим графом.

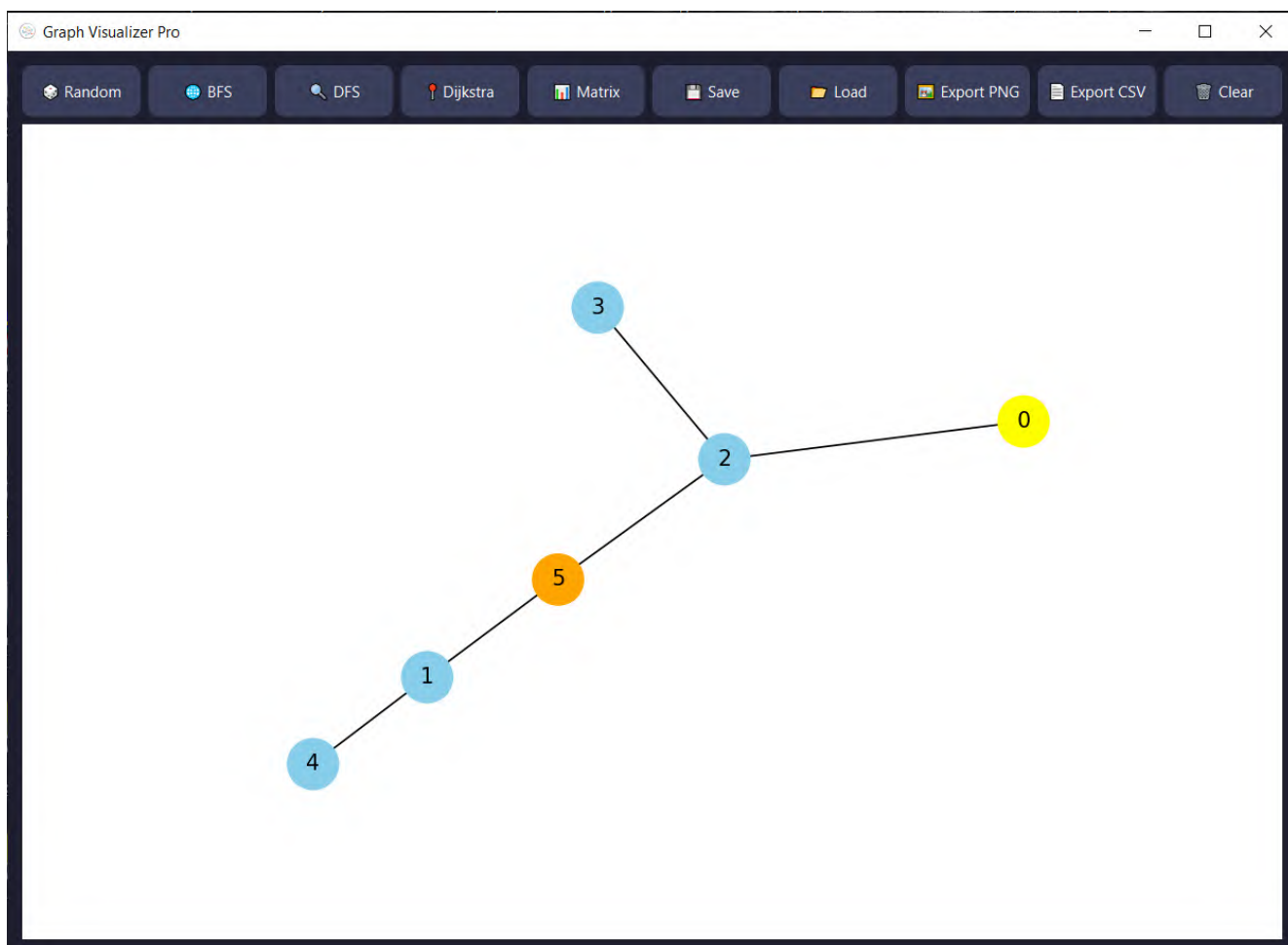


Рисунок 2.2 – Приклад побудованого графа у програмному застосунку

На рисунку представлено результат роботи функції генерації випадкового графа. Вершини та ребра графа відображаються у робочій області застосунку, що дозволяє користувачу аналізувати структуру створеного графа та виконувати подальші алгоритмічні операції.

Отож, реалізований графічний інтерфейс користувача забезпечує зручну взаємодію із програмним застосунком та створює комфортні умови для візуалізації й аналізу графових структур.

2.3 Реалізація взаємодії з графами та алгоритмів їх обробки

Однією з важливих складових розробленого програмного застосунку є реалізація інтерактивної взаємодії користувача із графовими структурами. У процесі створення програмного забезпечення особлива увага приділялася забезпеченню простого та зрозумілого механізму роботи з графом, який дозволяє виконувати основні операції без використання складних команд або додаткових налаштувань. Реалізований підхід забезпечує швидке створення графів та спрощує процес їх подальшого аналізу.

Взаємодія користувача із графом реалізована за допомогою дій миші та клавіатури. Такий підхід дозволяє забезпечити інтуїтивно зрозуміле керування програмним застосунком та створює зручні умови для роботи із графовими структурами у межах графічного інтерфейсу [17].

Створення нової вершини графа здійснюється за допомогою натискання лівої кнопки миші у межах робочої області застосунку. Після виконання даної дії у відповідній точці автоматично створюється нова вершина графа. Кожна вершина має власний унікальний номер, що дозволяє ідентифікувати її під час виконання алгоритмів аналізу графів та побудови зв'язків між вершинами.

Для видалення вершини використовується натискання правої кнопки миші на відповідному елементі графа. Після видалення вершини автоматично видаляються також усі ребра, пов'язані із нею. Реалізація такого механізму дозволяє підтримувати коректність структури графа та забезпечує правильну роботу алгоритмів аналізу.

Створення ребер між вершинами реалізовано шляхом послідовного вибору двох вершин за допомогою лівої кнопки миші. Після вибору першої та другої вершини у програмному застосунку автоматично відкривається додаткове діалогове вікно для введення ваги ребра. Користувач має можливість задати числове значення ваги, яке надалі використовується у процесі виконання алгоритмів аналізу графів, зокрема алгоритму Дейкстри.

Реалізація механізму задання ваг ребер дозволяє використовувати програмний застосунок для роботи зі зваженими графами. Це значно розширює функціональні можливості програми та забезпечує можливість демонстрації практичного використання алгоритмів пошуку найкоротших шляхів.

Окрему увагу під час розробки було приділено можливості зміни розташування вершин у робочій області. Для переміщення вершини використовується комбінація клавіші Shift та лівої кнопки миші. Після вибору вершини користувач може перетягувати її у необхідну область робочого поля, змінюючи структуру візуального представлення графа.

Можливість переміщення вершин дозволяє покращити наочність графового представлення та уникнути накладання елементів графа під час роботи із великими структурами. Крім того, дана функція забезпечує більш зручну взаємодію користувача із системою та дозволяє адаптувати вигляд графа відповідно до потреб користувача.

Реалізовані механізми взаємодії користувача із графом забезпечують простий та ефективний процес створення графових структур, їх редагування та подальшого аналізу. Використання інтерактивного підходу значно покращує зручність роботи із застосунком та дозволяє ефективно використовувати програму у навчальному процесі під час вивчення теорії графів.

Однією з основних функціональних можливостей розробленого програмного застосунку є реалізація алгоритму обходу графа в ширину. Даний алгоритм належить до базових алгоритмів теорії графів та широко використовується у задачах аналізу графових структур, пошуку шляхів, визначення зв'язності графа та дослідження структури мереж. Завдяки своїй ефективності алгоритм обходу графа в ширину є одним із найбільш поширених засобів дослідження графових структур. Реалізація даного алгоритму є важливою складовою функціональних можливостей програмного застосунку, призначеного для аналізу та дослідження графів.

Алгоритм обходу в ширину, або Breadth-First Search (BFS), виконує послідовне дослідження вершин графа за рівнями [18]. Принцип роботи алгоритму полягає у тому, що після відвідування поточної вершини виконуються переходи до

всіх суміжних вершин, після чого здійснюється перехід до наступного рівня графа. Такий підхід дозволяє забезпечити рівномірне дослідження структури графа та визначити найкоротший шлях у незважених графах.

У програмному застосунку запуск алгоритму здійснюється за допомогою окремої кнопки BFS, розташованої на панелі керування головного вікна. Після запуску алгоритму програма виконує аналіз структури графа та послідовно обробляє вершини відповідно до принципу обходу в ширину.

Для реалізації алгоритму використовується структура даних типу черга, яка забезпечує правильний порядок обробки вершин. Під час виконання алгоритму кожна відвідана вершина додається до черги, після чого виконується перевірка всіх суміжних вершин. Такий механізм дозволяє гарантувати послідовне дослідження графа відповідно до рівнів його структури.

Важливою особливістю реалізації алгоритму у програмному застосунку є наявність графічної візуалізації процесу обходу графа. Під час виконання алгоритму вершини та ребра можуть змінювати колір, що дозволяє користувачу наочно спостерігати порядок обходу графа та аналізувати процес роботи алгоритму у реальному часі.

Використання візуалізації має важливе значення у навчальному процесі, оскільки дозволяє краще зрозуміти принципи роботи алгоритму та підвищує ефективність засвоєння теоретичного матеріалу. Наочне представлення обходу графа значно спрощує сприйняття структури графових алгоритмів та демонструє особливості їх роботи.

Алгоритм обходу в ширину широко застосовується у різних сферах інформаційних технологій. Зокрема, він використовується у системах навігації, аналізі мережевих структур, пошуку маршрутів, дослідженні соціальних мереж та інших задачах, пов'язаних із графовими моделями.

У процесі тестування програмного застосунку було перевірено коректність роботи алгоритму BFS на графах різної структури. Результати тестування підтвердили правильність реалізації алгоритму та стабільність його роботи під час обробки графових структур із різною кількістю вершин та ребер.

На рисунку 2.3 наведено приклад роботи алгоритму обходу графа в ширину у програмному застосунку.

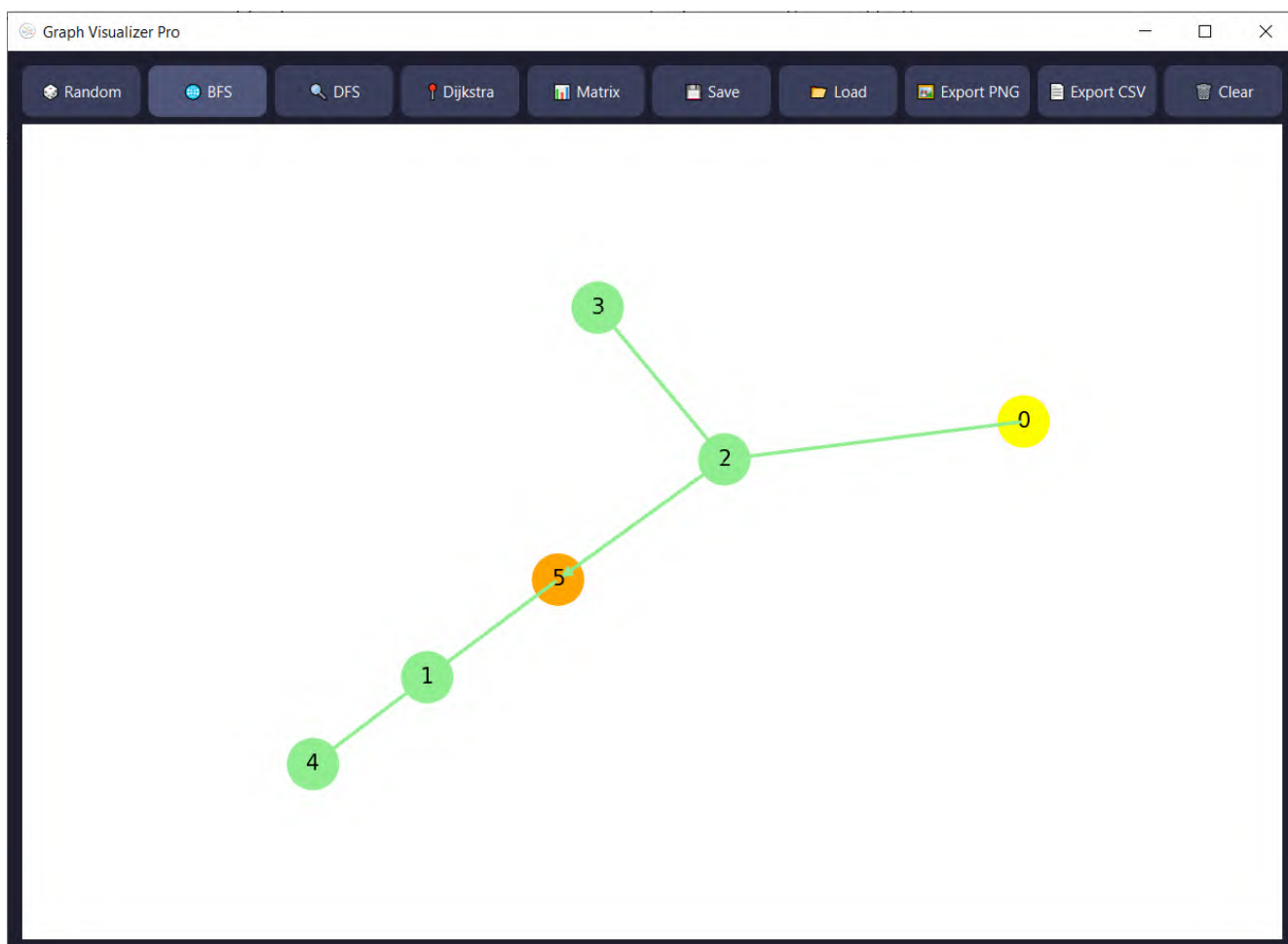


Рисунок 2.3 – Візуалізація алгоритму обходу графа в ширину

На рисунку представлено процес обходу графа в ширину із графічним виділенням вершин та ребер. Візуалізація дозволяє простежити порядок дослідження вершин та забезпечує наочне представлення роботи алгоритму.

Таким способом, реалізація алгоритму обходу графа в ширину забезпечує можливість ефективного аналізу графових структур, дозволяє визначати послідовність відвідування вершин, підвищує наочність представлення процесу обходу графа, сприяє кращому розумінню принципів роботи алгоритмів теорії графів та створює умови для практичного дослідження й вивчення особливостей функціонування графових алгоритмів у межах програмного застосунку.

У межах розробленого програмного застосунку реалізовано алгоритм обходу графа в глибину, який є одним із базових алгоритмів теорії графів та широко використовується у задачах аналізу графових структур. Алгоритм обходу в глибину, або Depth-First Search (DFS), застосовується для дослідження структури графа, пошуку компонент зв'язності, виявлення циклів та розв'язання багатьох інших задач, пов'язаних із графовими моделями [19].

Принцип роботи алгоритму DFS полягає у послідовному переході від поточної вершини до однієї із суміжних вершин із максимальним заглибленням у структуру графа. Після досягнення вершини, з якої неможливо виконати подальший перехід, алгоритм повертається до попередніх вершин для продовження обходу іншими шляхами. Такий підхід дозволяє ефективно досліджувати структуру графа та аналізувати його зв'язки.

У програмному застосунку запуск алгоритму здійснюється за допомогою кнопки DFS, розташованої на панелі керування головного вікна. Після запуску алгоритму програма виконує аналіз структури графа та послідовно обробляє вершини відповідно до принципу обходу в глибину.

Для реалізації алгоритму використовується механізм послідовного відвідування суміжних вершин із фіксацією вже оброблених елементів графа. Такий підхід дозволяє уникнути повторного відвідування вершин та забезпечує коректність роботи алгоритму навіть у випадку складних графових структур.

Важливою особливістю реалізації алгоритму у програмному застосунку є графічне представлення процесу обходу графа. Під час виконання алгоритму вершини та ребра, які беруть участь в обході, змінюють свій колір. Це дозволяє користувачу наочно спостерігати послідовність переходів між вершинами та аналізувати принцип роботи алгоритму.

Візуалізація алгоритму обходу в глибину має важливе значення у процесі вивчення теорії графів, оскільки дозволяє краще зрозуміти логіку роботи алгоритму та особливості його поведінки у різних графових структурах. Наочне представлення результатів роботи алгоритму спрощує сприйняття навчального матеріалу та підвищує ефективність практичного аналізу графів.

Алгоритм DFS широко застосовується у сучасних інформаційних технологіях. Зокрема, він використовується у задачах аналізу мереж, пошуку компонент зв'язності, побудови маршрутів, аналізу залежностей між об'єктами та інших задачах, що базуються на використанні графових моделей [20].

У процесі тестування програмного застосунку було перевірено коректність роботи алгоритму DFS на графах різної структури. Отримані результати підтвердили правильність реалізації алгоритму та стабільність його роботи під час обробки графових структур із різною кількістю вершин і ребер.

На рисунку 2.4 наведено приклад роботи алгоритму обходу графа в глибину у програмному застосунку.

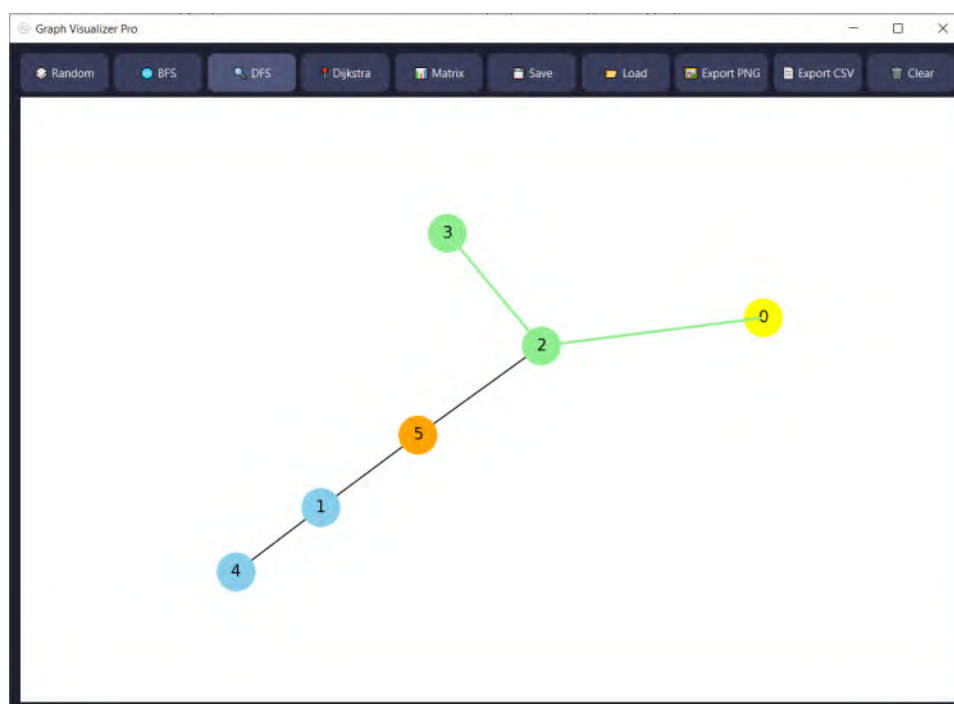


Рисунок 2.4 – Візуалізація алгоритму обходу графа в глибину

На рисунку представлено процес обходу графа в глибину із графічним виділенням вершин та ребер, які були оброблені алгоритмом. Візуалізація дозволяє простежити послідовність переходів між вершинами та забезпечує наочне представлення роботи алгоритму.

Тому, реалізація алгоритму обходу графа в глибину забезпечує можливість ефективного аналізу графових структур та дозволяє наочно демонструвати принципи роботи алгоритмів теорії графів у межах програмного застосунку.

У межах розробленого програмного застосунку реалізовано алгоритм Дейкстри, який використовується для пошуку найкоротшого шляху у зважених графах. Даний алгоритм є одним із найбільш відомих та ефективних алгоритмів теорії графів і широко застосовується у задачах маршрутизації, навігації, аналізу мереж та оптимізації шляхів.

Алгоритм Дейкстри призначений для визначення найкоротшого шляху між вершинами графа з урахуванням ваг ребер. Принцип роботи алгоритму полягає у послідовному виборі вершини з найменшою поточною відстанню та подальшому оновленні значень відстаней до суміжних вершин. У результаті виконання алгоритму формується оптимальний маршрут між заданими вершинами графа [21].

У програмному застосунку алгоритм запускається за допомогою кнопки Dijkstra, розташованої на панелі керування головного вікна. Після запуску алгоритму програма виконує аналіз структури графа, враховує ваги ребер та обчислює найкоротший шлях між вершинами.

Для коректної роботи алгоритму у програмі реалізовано підтримку зважених графів. Під час створення ребра користувач має можливість задавати вагу зв'язку між вершинами. Введені значення використовуються алгоритмом у процесі пошуку оптимального маршруту. Такий підхід дозволяє моделювати реальні задачі, у яких зв'язки між об'єктами мають різну вартість або довжину.

Важливою особливістю реалізації алгоритму у програмному застосунку є графічне представлення результатів роботи. Після завершення обчислень знайдений найкоротший шлях виділяється іншим кольором, що дозволяє користувачу наочно спостерігати результат роботи алгоритму та аналізувати побудований маршрут. Такий спосіб відображення результатів підвищує зручність використання програмного застосунку та спрощує сприйняття отриманих даних, забезпечуючи більш наочне представлення результатів виконання алгоритму для користувача.

Візуалізація алгоритму Дейкстри має важливе значення у навчальному процесі, оскільки дозволяє краще зрозуміти принципи пошуку найкоротших шляхів у графах [22]. Наочне представлення результатів роботи алгоритму значно спрощує сприйняття теоретичного матеріалу та дозволяє демонструвати практичне застосування методів аналізу графів.

Алгоритм Дейкстри широко використовується у сучасних інформаційних системах. Зокрема, він застосовується у навігаційних системах, телекомунікаційних мережах, логістичних системах, транспортних моделях та інших задачах, пов'язаних із пошуком оптимальних маршрутів.

Використання даного алгоритму дозволяє ефективно виконувати аналіз графових структур та визначати найкоротші шляхи між вершинами графа. Завдяки підтримці роботи зі зваженими графами алгоритм Дейкстри є одним із найбільш поширених алгоритмів пошуку оптимальних маршрутів у сучасному програмному забезпеченні.

У процесі роботи алгоритму здійснюється послідовний аналіз вершин графа та обчислення мінімальної вартості переходу між ними. Отримані результати відображаються у графічному вигляді, що дозволяє користувачу наочно спостерігати процес пошуку найкоротшого шляху та аналізувати результати роботи алгоритму.

У процесі тестування програмного застосунку було перевірено правильність роботи алгоритму на графах різної структури та складності. Результати тестування підтвердили коректність реалізації алгоритму та правильність обчислення найкоротших шляхів між вершинами графа.

Крім того, було перевірено стабільність роботи алгоритму під час виконання повторних обчислень та зміни структури графа. Отримані результати показали, що реалізований алгоритм коректно працює із різними типами графових структур та забезпечує правильне визначення оптимальних маршрутів між вершинами графа.

На рисунку 2.5 наведено приклад роботи алгоритму Дейкстри у програмному застосунку.

Реалізований механізм візуалізації результатів дозволяє користувачу не лише отримати числовий результат роботи алгоритму, але й наочно оцінити структуру побудованого маршруту. Це підвищує зручність використання програмного застосунку та робить процес аналізу графів більш зрозумілим і наочним.

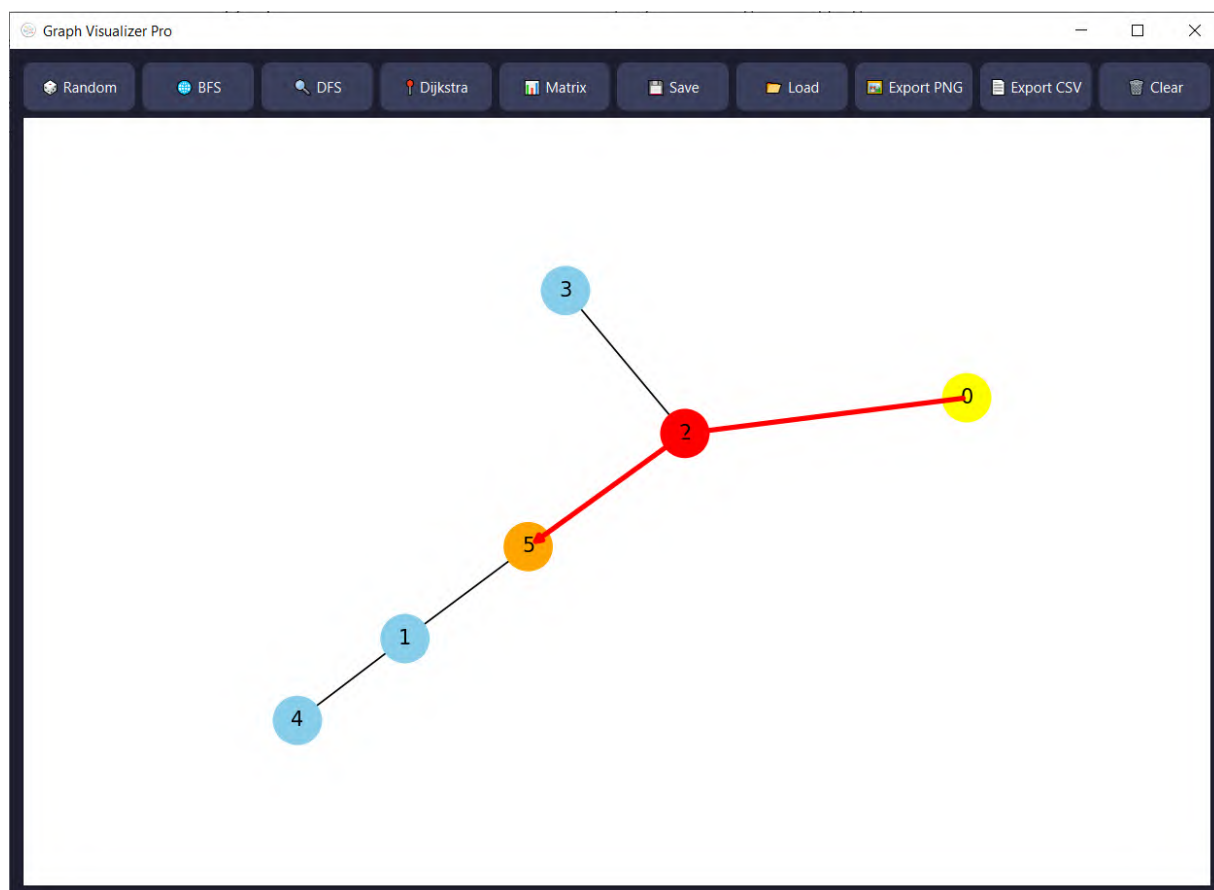


Рисунок 2.5 – Візуалізація роботи алгоритму Дейкстри

На рисунку представлено результат роботи алгоритму пошуку найкоротшого шляху. Знайдений маршрут виділено кольором, що дозволяє наочно продемонструвати результат виконання алгоритму та структуру оптимального шляху між вершинами графа.

Таким чином, реалізація алгоритму Дейкстри забезпечує можливість ефективного пошуку найкоротших шляхів у зважених графах, підвищує практичну цінність програмного застосунку, сприяє кращому розумінню принципів роботи графових алгоритмів та створює умови для наочного дослідження принципів роботи алгоритмів оптимізації у межах програмного застосунку.

2.4 Формування матриці суміжності та збереження даних

У процесі розробки програмного застосунку було реалізовано функцію формування матриці суміжності графа. Матриця суміжності є одним із основних способів представлення графових структур у вигляді таблиці та широко використовується у задачах аналізу графів, реалізації алгоритмів і дослідження зв'язків між вершинами.

Матриця суміжності являє собою двовимірну таблицю, у якій рядки та стовпці відповідають вершинам графа [23]. Значення у комірках матриці визначають наявність зв'язку між відповідними вершинами, а у випадку зважених графів можуть містити ваги ребер. Використання матричного представлення дозволяє спростити аналіз структури графа та забезпечує зручний спосіб роботи із графовими даними.

У програмному застосунку формування матриці суміжності здійснюється автоматично на основі поточної структури графа. Після натискання кнопки Matrix програма виконує аналіз усіх вершин та ребер графа, після чого генерує таблицю суміжності із відповідними значеннями.

Для відображення матриці у програмі використовується окреме діалогове вікно, у якому дані подаються у табличному вигляді. Рядки та стовпці таблиці містять номери вершин графа, а значення у комірках відповідають вагам зв'язків між вершинами. Якщо між вершинами відсутнє ребро, у відповідній комірці відображається нульове значення.

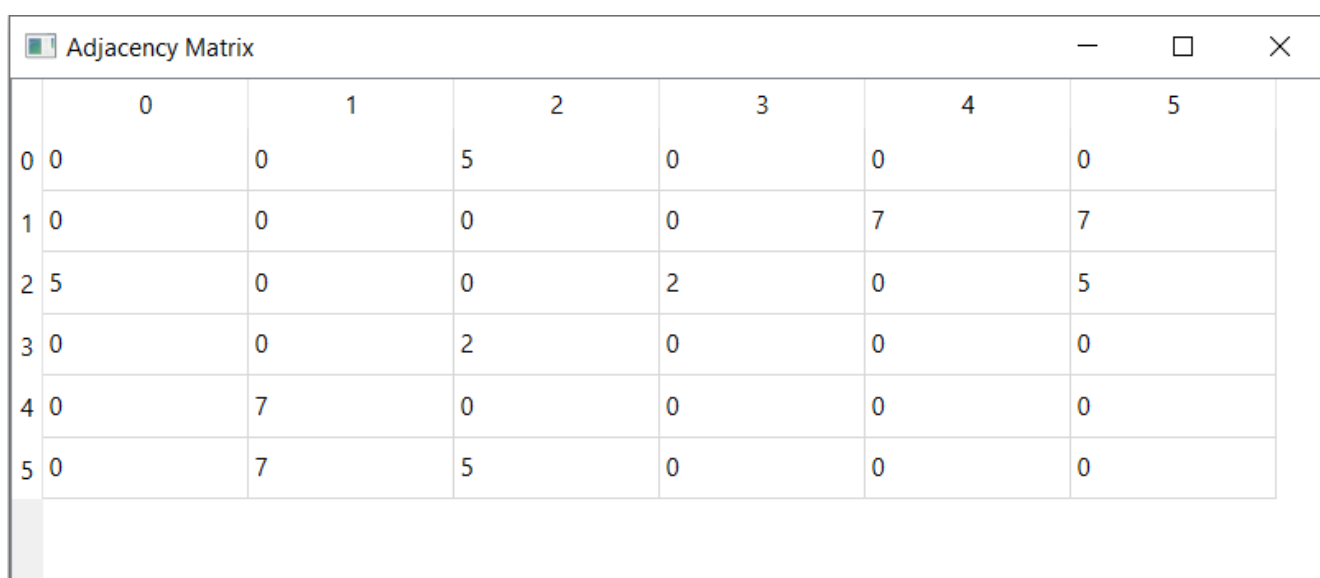
Реалізація функції формування матриці суміжності дозволяє користувачу отримати додаткове представлення структури графа та виконувати аналіз зв'язків між вершинами не лише у графічному, але й у табличному вигляді. Такий підхід є особливо корисним під час вивчення теорії графів та аналізу складних графових структур [24].

Використання матриці суміжності має важливе значення у процесі реалізації алгоритмів теорії графів, оскільки багато алгоритмів працюють саме із матричним представленням графа [25]. Наявність функції автоматичного формування матриці

значно спрощує процес дослідження графових структур та дозволяє користувачу швидко отримувати необхідну інформацію про граф.

У процесі тестування програмного застосунку було перевірено правильність формування матриці суміжності для графів різної структури. Результати тестування підтвердили коректність відображення зв'язків між вершинами та правильність формування вагових значень у таблиці.

На рисунку 2.6 наведено приклад сформованої матриці суміжності у програмному застосунку.



	0	1	2	3	4	5
0	0	0	5	0	0	0
1	0	0	0	0	7	7
2	5	0	0	2	0	5
3	0	0	2	0	0	0
4	0	7	0	0	0	0
5	0	7	5	0	0	0

Рисунок 2.6 – Матриця суміжності графа

На рисунку представлено табличне представлення графової структури у вигляді матриці суміжності. Використання такого способу подання даних дозволяє виконувати аналіз зв'язків між вершинами та забезпечує додатковий спосіб представлення графа у програмному застосунку [26].

Отже, реалізована функція формування матриці суміжності забезпечує можливість представлення графових структур у табличному вигляді, що значно спрощує процес аналізу взаємозв'язків між вершинами, підвищує зручність роботи користувача з даними та суттєво розширює функціональні можливості програмного застосунку для аналізу, дослідження й візуалізації графів, забезпечуючи зручне представлення структури зв'язків між елементами.

У процесі розробки програмного застосунку було реалізовано функції збереження, завантаження та експорту графових структур. Наявність таких можливостей дозволяє підвищити зручність використання програми, забезпечує можливість повторного використання створених графів та спрощує процес збереження результатів роботи користувача.

Функція збереження графа дозволяє записувати структуру створеного графа у файл для подальшого використання. Під час збереження програма виконує обробку інформації про вершини, ребра та ваги зв'язків між вершинами, після чого формує файл із відповідними даними. Реалізація даної функції забезпечує можливість повторного відкриття графа без необхідності його повторного створення.

Для збереження графових структур у програмному застосунку використовується формат JSON. Використання даного формату забезпечує зручне зберігання структурованих даних та спрощує процес обробки інформації. Формат JSON широко застосовується у сучасних програмних системах завдяки простоті структури та високій сумісності із різними мовами програмування [27].

Функція завантаження графа дозволяє відкривати раніше збережені файли та автоматично відновлювати структуру графа у робочій області застосунку. Після вибору необхідного файлу програма виконує зчитування інформації та формує відповідну графову структуру із вершинами, ребрами та вагами зв'язків.

Окрему увагу під час розробки було приділено реалізації функцій експорту результатів роботи. У програмному застосунку реалізовано можливість експорту графа у формат PNG. Дана функція дозволяє зберігати графічне представлення графової структури у вигляді зображення, що може використовуватися у звітах, презентаціях або навчальних матеріалах.

Крім експорту графічного представлення, у програмі реалізовано функцію експорту даних у формат CSV. Використання формату CSV дозволяє зберігати інформацію про структуру графа у табличному вигляді а також забезпечує можливість подальшої обробки даних у табличних редакторах або інших програмних системах [27].

Реалізація функцій збереження та експорту даних значно розширює функціональні можливості розробленого програмного застосунку та суттєво підвищує його практичну цінність. Користувач отримує можливість не лише виконувати аналіз графів у межах програми, але й зберігати отримані результати роботи для їх подальшого використання, опрацювання та перегляду.

У процесі тестування програмного застосунку було ретельно перевірено правильність роботи функцій збереження, завантаження та експорту даних. Результати проведеного тестування підтвердили коректність формування відповідних файлів, правильність обробки даних та стабільність роботи реалізованих механізмів програмного забезпечення в різних режимах використання.

Таким способом, реалізовані функції збереження та експорту даних забезпечують більш зручну та ефективну роботу із графовими структурами, а також створюють можливість раціонального використання результатів роботи програмного застосунку у навчальному процесі та практичній діяльності. Завдяки цьому підвищується загальна зручність використання програмного забезпечення та розширюються можливості роботи користувача з отриманими даними.

РОЗДІЛ 3

ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Тестування функціональних можливостей програмного застосунку

Після завершення основного етапу розробки програмного застосунку було проведено тестування його функціональних можливостей. Основною метою тестування була перевірка коректності роботи реалізованих алгоритмів, стабільності функціонування графічного інтерфейсу та правильності взаємодії між окремими компонентами системи.

У процесі тестування перевірялася робота всіх основних функцій програмного застосунку, зокрема створення та видалення вершин, побудова ребер між вершинами, задання ваг зв'язків, переміщення елементів графа, генерація випадкових графів, формування матриці суміжності, а також робота алгоритмів обходу графа та пошуку найкоротших шляхів.

Тестування функцій створення графа підтвердило правильність роботи механізмів взаємодії користувача із графічним інтерфейсом. Під час додавання вершин та створення ребер програма коректно формувала структуру графа та відображала зміни у робочій області. Видалення вершин також виконувалося правильно, при цьому автоматично видалялися пов'язані із ними ребра.

Окрему увагу було приділено тестуванню функції переміщення вершин. У процесі перевірки було встановлено, що переміщення графічних елементів здійснюється коректно, а структура графа автоматично оновлюється відповідно до нових координат вершин. Це забезпечує зручність роботи користувача та покращує візуальне представлення графових структур.

Під час тестування алгоритму обходу графа в ширину було перевірено правильність послідовності відвідування вершин та коректність візуального відображення результатів роботи алгоритму. Результати тестування підтвердили, що алгоритм BFS правильно обробляє структуру графа та забезпечує коректне виділення відвіданих вершин і ребер.

Аналогічно було проведено тестування алгоритму обходу графа в глибину. У процесі перевірки алгоритм DFS продемонстрував правильність роботи на графах різної структури та складності. Візуалізація результатів дозволила підтвердити коректність послідовності обходу графа та стабільність роботи алгоритму.

Під час тестування алгоритму Дейкстри особлива увага приділялася правильності обчислення найкоротших шляхів у зважених графах. Результати перевірки підтвердили, що алгоритм правильно враховує ваги ребер та коректно визначає оптимальний маршрут між вершинами графа.

Також було проведено тестування функції формування матриці суміжності. У процесі перевірки встановлено, що програма правильно формує табличне представлення графа та коректно відображає зв'язки між вершинами й вагові значення ребер.

Функції збереження та експорту даних також були перевірені у процесі тестування. Збереження графів у форматі JSON, а також експорт даних у формати PNG та CSV виконувалися без помилок. Завантаження раніше збережених графів дозволяло повністю відновлювати структуру графа та результати роботи користувача.

У процесі тестування програмного застосунку не було виявлено критичних помилок, які б впливали на стабільність роботи системи або коректність виконання алгоритмів. Реалізовані функціональні можливості продемонстрували стабільну роботу та правильність взаємодії між окремими компонентами програмного забезпечення.

Під час перевірки роботи програмного застосунку також було протестовано коректність обробки дій користувача під час створення, редагування та видалення елементів графа. Отримані результати підтвердили правильність оновлення структури графа та стабільність роботи графічного інтерфейсу користувача під час виконання різних операцій.

Окрему увагу було приділено перевірці роботи функцій візуалізації графів та відображення результатів роботи алгоритмів. У процесі тестування було встановлено, що програмний застосунок коректно виконує побудову графових

структур та забезпечує наочне відображення результатів аналізу графів у графічному вигляді.

Отже, результати тестування підтвердили коректність реалізації основних функцій програмного застосунку та стабільність роботи алгоритмів. Реалізований функціонал забезпечує зручну взаємодію користувача із системою та дозволяє ефективно використовувати програмний застосунок для створення, візуалізації та аналізу графових структур.

3.2 Аналіз результатів роботи програмного застосунку

Після завершення тестування програмного застосунку було проведено аналіз результатів його роботи та оцінку ефективності реалізованих функціональних можливостей. Основною метою аналізу було визначення рівня коректності роботи алгоритмів, зручності використання графічного інтерфейсу та ефективності реалізованих механізмів візуалізації графових структур.

У процесі аналізу встановлено, що програмний застосунок забезпечує коректну роботу із графовими структурами та дозволяє ефективно виконувати основні операції аналізу графів. Реалізовані механізми створення, редагування та видалення елементів графа працюють стабільно та забезпечують зручну взаємодію користувача із системою.

Особливу увагу було приділено оцінці роботи алгоритмів обходу графа та пошуку найкоротших шляхів. Результати тестування підтвердили правильність роботи алгоритмів BFS, DFS та Дейкстри на графах різної структури та складності. У процесі виконання алгоритмів не було виявлено критичних помилок або порушень логіки обробки графових структур. Отримані результати тестування підтвердили стабільність роботи реалізованих алгоритмів та ефективність їх використання у межах програмного застосунку для аналізу графових структур.

Важливою перевагою програмного застосунку є наявність графічної візуалізації результатів роботи алгоритмів. Використання кольорового виділення

вершин та ребер дозволяє користувачу наочно спостерігати процес обходу графа або побудови найкоротшого шляху. Такий підхід значно покращує сприйняття інформації та забезпечує більш ефективне засвоєння принципів роботи алгоритмів теорії графів [28].

Проведений аналіз показав, що реалізований графічний інтерфейс є достатньо простим та зрозумілим для користувача. Основні функції програми доступні через систему кнопок та не потребують складних налаштувань або додаткових дій. Інтерактивний підхід до роботи із графом дозволяє швидко створювати графові структури та виконувати їх подальший аналіз.

Позитивним аспектом програмного застосунку є підтримка роботи зі зваженими графами. Можливість задання ваг ребер дозволяє використовувати програму для демонстрації алгоритмів оптимізації маршрутів та дослідження задач пошуку найкоротших шляхів [29].

Окрему увагу під час аналізу було приділено функції формування матриці суміжності. Використання табличного представлення графа дозволяє отримати додатковий спосіб аналізу графових структур та забезпечує більш глибоке розуміння принципів представлення графів у програмуванні [30].

Функції збереження та експорту даних також позитивно впливають на практичну цінність програмного застосунку. Можливість експорту графів у формати PNG та CSV дозволяє використовувати результати роботи програми у навчальних матеріалах, звітах та інших програмних засобах.

Під час аналізу результатів роботи застосунку було встановлено, що програма може ефективно використовуватися як допоміжний інструмент у процесі вивчення дисципліни «Дискретна математика». Використання візуалізації графів та алгоритмів дозволяє покращити сприйняття навчального матеріалу та підвищити рівень розуміння принципів роботи графових структур.

Тому, результати аналізу підтверджують ефективність реалізованого програмного застосунку та доцільність його використання для візуалізації та аналізу графів у навчальному процесі а також під час виконання практичних завдань.

3.3 Переваги та перспективи розвитку програмного застосунку

Розроблений програмний застосунок для візуалізації та аналізу графів має низку переваг, які забезпечують його ефективне використання у навчальному процесі та під час дослідження графових структур. Реалізований функціонал дозволяє не лише виконувати аналіз графів, але й забезпечує наочне представлення результатів роботи алгоритмів, що є важливим у процесі вивчення дисципліни «Дискретна математика».

Однією з основних переваг програмного застосунку є наявність інтерактивного графічного інтерфейсу користувача. Використання графічного способу взаємодії із системою дозволяє значно спростити процес роботи з графами та забезпечує швидкий доступ до основних функцій програми. [31] Користувач має можливість створювати графові структури без використання складних команд або додаткових налаштувань.

Важливою перевагою застосунку є підтримка візуалізації алгоритмів аналізу графів. Графічне відображення процесу обходу графа та пошуку найкоротших шляхів дозволяє наочно демонструвати принципи роботи алгоритмів BFS, DFS та Дейкстри. Використання кольорового виділення вершин і ребер покращує сприйняття інформації та забезпечує більш ефективне засвоєння навчального матеріалу.

Позитивним аспектом програмного застосунку є можливість роботи зі зваженими графами. Реалізація механізму задання ваг ребер дозволяє використовувати програму для демонстрації задач оптимізації маршрутів та аналізу транспортних або мережевих структур [32].

Окремою перевагою є підтримка генерації випадкових графів. Дана функція спрощує процес тестування алгоритмів та дозволяє швидко створювати нові графові структури для подальшого аналізу. Використання випадкової генерації графів також розширює можливості практичного використання програмного застосунку.

Важливим елементом функціоналу є підтримка збереження та експорту даних. Користувач має можливість зберігати графові структури у форматі JSON [33], експортувати графічне представлення графа у формат PNG [34] та виконувати експорт даних у формат CSV [35]. Це забезпечує можливість подальшого використання результатів роботи програми у звітах, презентаціях та інших програмних засобах.

Під час розробки програмного застосунку використовувалися сучасні бібліотеки Python, що дозволило забезпечити стабільність роботи системи та спростити реалізацію алгоритмів і механізмів візуалізації. Використання бібліотек NetworkX та matplotlib дозволило значно скоротити складність реалізації окремих функціональних компонентів програми.

Незважаючи на реалізацію основного функціоналу, програмний застосунок має можливості для подальшого розвитку та вдосконалення. Одним із перспективних напрямів є розширення набору алгоритмів аналізу графів. У майбутньому до програми можуть бути додані алгоритми пошуку мінімального остовного дерева, алгоритми Флойда–Воршелла, Беллмана–Форда та інші методи аналізу графових структур.

Перспективним напрямом розвитку також є вдосконалення графічного інтерфейсу користувача. У подальших версіях програми може бути реалізована підтримка світлої та темної теми оформлення, багатомовного інтерфейсу, а також додаткових засобів анімації процесу роботи алгоритмів.

Ще одним напрямом розвитку є створення вебверсії програмного застосунку або реалізація підтримки роботи із базами даних для збереження графових структур. Це дозволить розширити сферу використання програми та забезпечити можливість роботи із графами у мережевому середовищі.

Отже, розроблений програмний застосунок має достатній набір функціональних можливостей для ефективної роботи із графовими структурами та може бути використаний як допоміжний інструмент у навчальному процесі. Наявність можливостей подальшого розвитку забезпечує перспективність

удосконалення програмного продукту у майбутньому та розширення його функціональних можливостей.

3.4 Оцінка вартості розробки програмного застосунку

Вартість розробки наведена у таблиці 3.1. Оцінка витрат часу на створення програмного застосунку була виконана на основі середніх показників погодинної оплати праці, представлених на тематичних вебресурсах. Під час розрахунку було враховано час, витрачений на проектування, реалізацію та тестування програмного застосунку. Оскільки для розробки використовувалося програмне забезпечення з відкритим доступом, додаткові витрати на придбання програмних засобів відсутні. Отримана оцінка дозволяє визначити орієнтовну вартість створення програмного застосунку та характеризує трудові витрати, необхідні для його розробки.

Таблиця 3.1 – Вартість розробки програмного застосунку

Складова розробки	Вартість	Пояснення
Вартість програмного забезпечення, використаного для розробки	0 грн	Програмне забезпечення, яке використовувалося під час розробки, є безкоштовним
Вартість часу, витраченого на розробку	10 год x 300 грн	Усереднене значення на основі даних тематичних вебресурсів для розробників початкового рівня.

Отже, проведене тестування функціональних можливостей застосунку підтвердило стабільність та коректність роботи всіх реалізованих алгоритмів; детальний аналіз результатів засвідчив високу ефективність та педагогічну цінність програми, наочність якої досягається завдяки інтерактивному графічному інтерфейсу та кольоровому виділенню елементів; проведена оцінка вартості розробки підтверджує його економічну ефективність, зважаючи на використання виключно безкоштовного програмного забезпечення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено теоретичні основи теорії графів, розглянуто основні способи представлення графових структур та проаналізовано алгоритми, що використовуються для роботи з графами. Проведений аналіз показав, що графи є важливим інструментом для моделювання складних систем та широко застосовуються у різних галузях інформаційних технологій, математики та програмування.

У межах роботи було розроблено програмний застосунок для візуалізації та аналізу графів мовою програмування Python. Для реалізації проєкту використано бібліотеки NetworkX та matplotlib, які забезпечили можливість створення графових структур, реалізації алгоритмів та графічного відображення результатів роботи.

Під час розробки програмного забезпечення було реалізовано інтерактивний графічний інтерфейс користувача, що дозволяє створювати вершини та ребра, задавати ваги зв'язків, переміщувати елементи графа та виконувати їх видалення. Реалізовані механізми взаємодії забезпечують просту та зрозумілу роботу із графовими структурами у межах програмного застосунку.

У програмі реалізовано алгоритми обходу графа в ширину та в глибину, а також алгоритм Дейкстри для пошуку найкоротших шляхів у зважених графах. Наявність графічної візуалізації дозволяє наочно демонструвати процес роботи алгоритмів та покращує сприйняття навчального матеріалу під час вивчення дисципліни «Дискретна математика».

Окрему увагу під час розробки було приділено реалізації додаткових функціональних можливостей програмного застосунку. Зокрема, у програмі реалізовано генерацію випадкових графів, формування матриці суміжності, збереження та завантаження графових структур, а також експорт результатів роботи у формати PNG та CSV.

У процесі тестування програмного застосунку було перевірено коректність роботи реалізованих алгоритмів та стабільність функціонування системи. Отримані

результати підтвердили правильність роботи механізмів аналізу графів, графічного інтерфейсу та функцій візуалізації.

Практичне значення розробленого програмного застосунку полягає у можливості його використання як допоміжного інструменту у процесі вивчення теорії графів та дисципліни «Дискретна математика». Використання графічної візуалізації та інтерактивної взаємодії із графом дозволяє покращити розуміння принципів роботи алгоритмів і підвищити ефективність навчального процесу. Розроблений програмний застосунок поєднує засоби побудови, редагування та аналізу графових структур в єдиному програмному середовищі, що забезпечує зручність роботи користувача та розширює можливості практичного дослідження графів. Реалізований функціонал дозволяє виконувати основні операції над графовими структурами та отримувати результати у наочному графічному вигляді.

Отже, поставлену мету кваліфікаційної роботи досягнуто, а всі основні завдання, визначені на початковому етапі розробки, успішно виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Graph Theory | Brilliant Math & Science Wiki. URL: <https://brilliant.org/wiki/graph-theory/> (дата звернення: 23.03.2026).
2. Тема 5. Теорія графів Лекції Історична довідка. URL: <https://studfile.net/preview/9331189/> (дата звернення: 23.03.2026).
3. Способи представлення графів. URL: <https://disted.edu.vn.ua/courses/learn/250> (дата звернення: 23.03.2026).
4. Graph Data Structure. URL: <https://www.programiz.com/dsa/graph> (дата звернення: 23.03.2026).
5. Список суміжності та матричне представлення графа. URL: <https://www.guru99.com/uk/adjacency-list-matrix-representation-graph.html> (дата звернення: 23.03.2026).
6. Обхід графа вглиб. URL: <https://disted.edu.vn.ua/courses/learn/277> (дата звернення: 23.03.2026).
7. Тонконогий В.М., Вайсман В.О., Бовнегра Л.В., Кіркопуло К.Г. Системний аналіз: навчальний посібник для здобувачів спеціальності 122 – Комп’ютерні науки. Одеса: Національний університет «Одеська політехніка», 2022. 84 с.
8. Задача про найкоротший шлях - алгоритм Дейкстри. URL: <https://www.mathros.net.ua/znahodzhennja-najkorotshogo-shljahu-v-orijentovanomu-grafi-za-algorytmom-dejkstry.html> (дата звернення: 23.03.2026).
9. Welcome to Python.org. URL: <https://www.python.org/about/> (дата звернення: 23.03.2026).
10. Software for Complex Networks — NetworkX 3.6.1 documentation. URL: <https://networkx.org/documentation/stable/> (дата звернення: 23.03.2026).
11. Matplotlib documentation — Matplotlib 3.10.9 documentation. URL: <https://matplotlib.org/stable/> (дата звернення: 23.03.2026).
12. Python Application Layouts: A Reference – Real Python. URL: <https://realpython.com/python-application-layouts/> (дата звернення: 23.03.2026).

13. Nielsen Norman Group. Dark Mode vs. Light Mode: Which Is Better? URL: <https://www.nngroup.com/articles/dark-mode/> (дата звернення: 23.03.2026).
14. Icon Usability. URL: <https://www.nngroup.com/articles/icon-usability/> (дата звернення: 23.03.2026).
15. matplotlib.pyplot.savefig — Matplotlib 3.10.9 documentation. URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html (дата звернення: 23.03.2026).
16. CSV File Reading and Writing. URL: <https://docs.python.org/3/library/csv.html> (дата звернення: 23.03.2026).
17. Usability 101: Introduction to Usability. URL: <https://www.nngroup.com/articles/usability-101-introduction-to-usability/> (дата звернення: 23.03.2026).
18. Пошук в ширину | Algoua. URL: <https://algoua.com/algorithms/graphs/bfs/> (дата звернення: 23.03.2026).
19. Алгоритм обходу в глибину для орієнтованого графа. URL: <https://www.mathros.net.ua/obhid-orijentovanogo-grafa-v-glybynu.html> (дата звернення: 23.03.2026).
20. Depth First Search (DFS) Algorithm. URL: <https://www.programiz.com/dsa/graph-dfs> (дата звернення: 23.03.2026).
21. 12.3: Алгоритм Дейкстри для найкоротших шляхів. URL: <https://lnk.ua/xkO1a4Uzs> (дата звернення: 23.03.2026).
22. Задача про найкоротший шлях - алгоритм Дейкстри. URL: <https://www.mathros.net.ua/znahodzhennja-najkorotshogo-shljahu-v-orijentovanomu-grafi-za-algorytmom-dejkstry.html> (дата звернення: 23.03.2026).
23. Graph Adjacency Matrix (With code examples in C++, Java and Python). URL: <https://www.programiz.com/dsa/graph-adjacency-matrix> (дата звернення: 23.03.2026).
24. Representation of Graph — GeeksforGeeks. URL: <https://www.geeksforgeeks.org/dsa/graph-and-its-representations/> (дата звернення: 23.03.2026).

25. Graph Theory – Adjacency Matrix. URL: https://www.tutorialspoint.com/graph_theory/graph_theory_adjacency_matrix.htm (дата звернення: 23.03.2026).
26. Baeldung. Adjacency Matrix and Adjacency List Complexity. URL: <https://www.baeldung.com/cs/adjacency-matrix-list-complexity> (дата звернення: 23.03.2026).
27. Working with csv files in Python – GeeksforGeeks. URL: <https://www.geeksforgeeks.org/python/working-csv-files-python/> (дата звернення: 23.03.2026).
28. Gallery – NetworkX 3.6.1 documentation. URL: https://networkx.org/documentation/stable/auto_examples/index.html (дата звернення: 23.03.2026).
29. Graph Theory – Weighted Graphs. URL: https://www.tutorialspoint.com/graph_theory/graph_theory_weighted_graphs.htm (дата звернення: 23.03.2026).
30. Graph Representation using Java ArrayList – GeeksforGeeks. URL: <https://www.geeksforgeeks.org/java/graph-representation-using-java-arraylist/> (дата звернення: 23.03.2026).
31. What are Graphical User Interfaces (GUIs)? URL: <https://ixdf.org/literature/topics/graphical-user-interfaces> (дата звернення: 23.03.2026).
32. Graph Theory – Applications. URL: https://www.tutorialspoint.com/graph_theory/graph_theory_applications.htm (дата звернення: 23.03.2026).
33. JSON. URL: <https://www.json.org/json-en.html> (дата звернення: 23.03.2026).
34. PNG (Portable Network Graphics) Home Site. URL: <https://libpng.org/pub/png/> (дата звернення: 23.03.2026).
35. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files | RFC Editor. URL: <https://www.rfc-editor.org/info/rfc4180/> (дата звернення: 23.03.2026).