Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут комп’ютерних наук та інформаційних технологій
Кафедра автоматизованих систем управління

Курсова робота

з дисципліни

“Алгоритми і структури даних”

на тему:

ПОШУК ЦИКЛІВ У ОРІЄНТОВАНИХ ГРАФАХ

Виконав:

студент ОІ-25

Басала Валентин

Керівник:

Скрибайло-Леськів Д.Ю.

Львів – 2023

Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра автоматизованих систем управління

Завдання до курсової роботи

з дисципліни **“Алгоритми і структури даних”**

Прізвище, ім'я студента Басала Валентин
Група ОІ-25
Тема курсової роботи Пошук циклів у орієнтованих графах
Спеціальна частина завдання:

1. У розділі огляду літератури розглянути основні принципи та методики використання фреймворку Qt для створення графічного інтерфейсу, а також аспекти їх поєднання з алгоритмами, реалізованими на мові програмування C++. Також розглянути алгоритми, які використовуються для ефективного пошуку циклів у орієнтованих графах.
2. Реалізувати можливість пошуку циклів у орієнтованих графах двома алгоритмами:
    1) DFS з повторенням вершин;
    2) DFS без повторення вершин.
3. Розробити інтерфейс для вибору файлів з вхідними даними, формування вручну даних, для відображення графа та анімації алгоритму його обходу, також для виведення всіх циклів та відомостей про використаний алгоритм (шлях обходу графа, час виконання алгоритму).
4. Середовище функціонування програми – Qt Creator.
5. Термін завершення роботи – 14 грудня 2023р.

Завдання видано: 13 вересня 2023 р.

Керівник (_____) Скрибайло-Леськів Д.Ю.

Студент (_____) Басала В.Р.

Зміст

Вступ

Орієнтовані графи є одними з основних об’єктів в теорії графів та знаходять широке застосування у різних галузях науки та техніки. Орієнтований граф - це граф, ребрам якого присвоєно напрямок. Однією з важливих задач, пов’язаних з орієнтованими графами, є пошук циклів - замкнутих шляхів, які повертаються у початкову вершину через певну кількість кроків [4].

Актуальність дослідження пошуку циклів в орієнтованих графах полягає у його важливості для виявлення аномалій у мережах, визначенні вразливих місць в програмному забезпеченні, оптимізацію та моделювання різноманітних систем, оптимізації маршрутів у транспортних системах та ін.

Традиційно для вирішення задач пошуку циклів у графах використовуються алгоритми на основі пошуку в глибину (DFS) [2] та його модифікації. Ці алгоритми базуються на матрицях суміжності та списках суміжності.

Отже, дана курсова робота має на меті дослідити та реалізувати алгоритми пошуку циклів в орієнтованих графах з використанням мови програмування C++ та фреймворку Qt. Серед основних цілей роботи є проаналізувати існуючі методи пошуку циклів, реалізувати власні алгоритми для ефективного виявлення циклів у графі, а також створити інтерактивний інтерфейс користувача для візуалізації та використання розроблених алгоритмів.

1. Огляд літератури

1.1 Використання фреймворку Qt для графічного інтерфейсу в поєднанні з алгоритмами на C++

Qt надає можливість розробки крос-платформеного програмного забезпечення, що може працювати на різних операційних системах, таких як Windows, macOS, Linux тощо. Це важливо враховувати при розробці інтерфейсу, оскільки дизайн може відрізнятися на різних платформах.

Використання C++ з Qt надає можливість впроваджувати потужні можливості мови програмування C++ разом з засобами Qt для швидкої розробки графічного інтерфейсу та реалізації функціональності програм. Qt надає спеціальні класи та функції, які доповнюють стандартні можливості C++. Таке поєднання дозволяє розширити можливості розробки програм та забезпечити їх швидку та ефективну реалізацію.

Під час використання фреймворку Qt для створення графічного інтерфейсу важливо використовувати класи та об’єкти Qt для створення вікон, кнопок, текстових полів та інших елементів інтерфейсу. Встановлення взаємозв'язків між цими віджетами необхідно для обробки подій та забезпечення взаємодії користувача з програмою. Механізми сигналів та слотів в Qt дозволяють ефективно організувати взаємодію між об'єктами.

Важливо виконувати тестування для переконання, що графічний інтерфейс функціонує як очікується та є зручним для користувача. Реалізація сценаріїв користувача допомагає забезпечити оптимальну взаємодію.

1.2 Алгоритми для пошуку циклів у орієнтованих графах

Алгоритми пошуку циклів у орієнтованих графах - це набір методів та стратегій, спрямованих на виявлення та аналіз замкнених шляхів у графі, де вершина початку співпадає з вершиною закінчення. Ці алгоритми мають за мету знайти відповідний цикл або цикли, які утворюють замкнуті ланцюжки вершин та ребер.

Серед таких алгоритмів є алгоритм DFS (Depth-First Search). Алгоритм пошуку в глибину, що використовується для обходу графа та виявлення циклів. Під час обходу, якщо зустрічається вершина, яка вже була відвідана і не є батьківською для поточної вершини, це свідчить про існування циклу. Алгоритм DFS дозволяє проникнути в структуру графа, що допомагає виявити зв'язність між вершинами та визначити наявність циклів, що є корисним для багатьох аспектів аналізу графів. За допомогою цього алгоритму можна визначити всі цикли в орієнтованому графі.

Також є алгоритм пошуку SCC (Strongly Connected Components). Алгоритм сильно зв’язних компонентів виявляє максимальні підграфи, у яких кожна вершина доступна з будь-якої іншої, та допомагають виявити цикли в графі. Проте даний алгоритм не гарантовано виявить всі цикли. SCC виявляють компоненти, які є сильно зв'язними в межах себе, але це не означає, що вони містять всі цикли графа. Граф може містити цикли, які перетинають декілька SCC.

2. Постановка задачі

Основним класом задачі є виявлення циклів у вже заданому орієнтованому графі. Метою розв'язання цієї задачі є ідентифікація наявності циклів в графі, а також визначення їхньої структури, тобто послідовності вершин, що утворюють цикл.

Результатом програмної реалізації має бути визначення всіх циклів, якщо вони є, та їхнє подання: перелік вершин, що складають цикл, а також ребра, які його утворюють.

Умовою існування циклу в графі є наявність послідовності вершин, яку можна пройти та повернутися до початкової вершини. Для вирішення цієї задачі, необхідно мати вхідні дані, що можуть представляти граф у форматі списку ребер. Формат вхідних даних може бути у вигляді текстових файлів.

Критерієм ефективності розв'язку може бути час виконання алгоритму, а також точність виявлення циклів у орієнтованих графах.

3. Алгоритми розв’язування задачі

3.1 Алгоритм DFS з повторенням вершин

Словесний опис

1. Для кожної вершини графа використовуємо рекурсивний обхід в глибину.
2. Позначаємо вершини, що вже були відвідані, та додаємо їх у поточний шлях.
3. При знаходженні вершини, яка вже є частиною поточного шляху, виявляємо цикл та зберігаємо його.
4. Фільтруємо знайдені цикли за умовою наявності більше ніж однієї вершини, щоб відкинути петлі (петля не являється циклом).

Блок-схема (1-3 пункти)

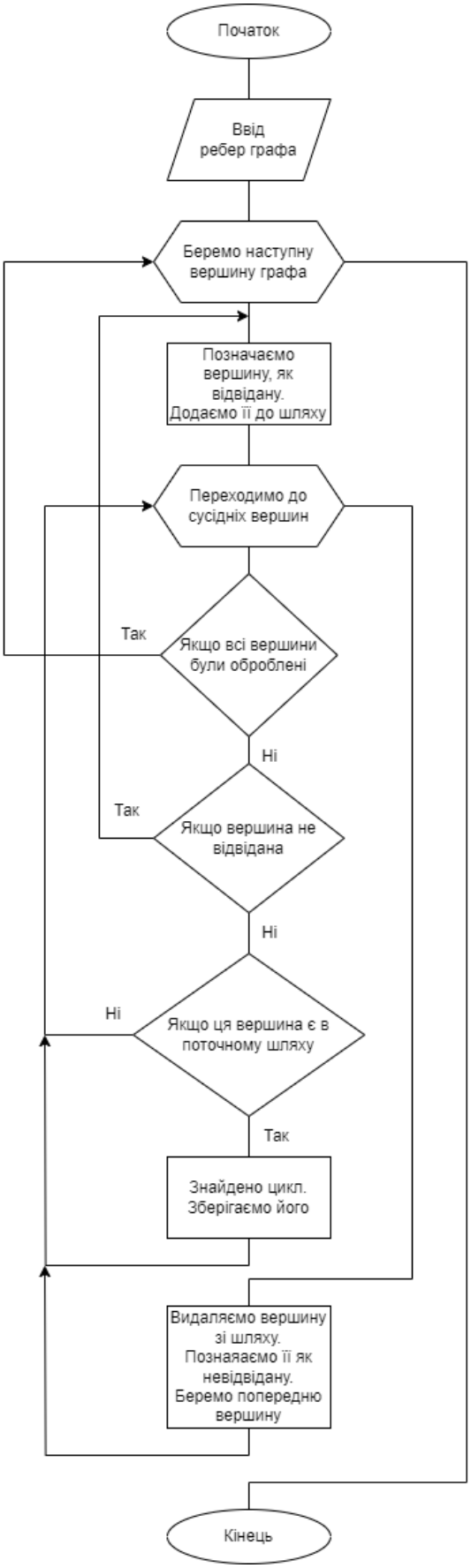

Рис. 1 – Блок-схема алгоритму DFS з повторенням вершин

Псевдокод

```
DFS (v, graph, visited, path, cycles):
    visited[v] = true
    path.push(v)

    for each u in graph[v]:
        if !visited[u]:
            DFS(u, graph, visited, path, cycles)
        else if u in path:
            cycle = []
            it = find(u in path)
            for each vertex in path starting from it:
                cycle.push(vertex)
            cycles.insert(cycle)

    path.pop()
    visited[v] = false


FindAllCycles(graph):
    n = size(graph)
    cycles = set()
    visited = array of size n initialized to false
    path = empty array

    for i from 1 to n:
         DFS(i, graph, visited, path, cycles)

    uniqueCycles = []

    for each cycle in cycles:
        if size(cycle) > 1:
                uniqueCycles.push(cycle)

    return uniqueCycles
```

3.2 Алгоритм DFS без повторення вершин

Словесний опис

1. Для кожної вершини графа, яка не була відвідана (під час обходу в глибину), використовуємо рекурсивний обхід в глибину.
2. Під час обходу в глибину переходимо до сусідньої вершини (напрямок ребра), тільки тоді, коли вона не була відвідана в минулих рекурсивних обходах в глибину.
2. Позначаємо вершини, що вже були відвідані, та додаємо їх у поточний шлях.
3. При знаходженні вершини, яка вже є частиною поточного шляху, виявляємо цикл та зберігаємо його.
4. Фільтруємо знайдені цикли за умовою наявності більше ніж однієї вершини, щоб відкинути петлі (петля не являється циклом).

Псевдокод

```
DFS (v, graph, visited, visit, used, path, cycles):
   visited[v] = true
   used[v] = true
   path.push(v)

   for each u in graph[v]:
      if !used[u] && !visit[u]:
         DFS(u, graph, visited, visit, used, path, cycles)
      else if u in path:
         cycle = []
         it = find(u in path)
         for each vertex in path starting from it:
            cycle.push(vertex)
         cycles.insert(cycle)

   path.pop()
   used[v] = false


FindAllCycles(graph):
   n = size(graph)
   cycles = set()
   used = array of size n initialized to false
```

```
visited = array of size n initialized to false
visit = array of size n initialized to false
path = empty array

for i from 1 to n:
    if !visited[i]:
        DFS(i, graph, visited, visit, used, path, cycles)
    visit[i] = true

uniqueCycles = []

for each cycle in cycles:
    if size(cycle) > 1:
            uniqueCycles.push(cycle)

return uniqueCycles
```

4. Програмна реалізація алгоритму

4.1 Загальна характеристика та можливості програми

Структура програми

Програма містить 4 класи, кожен з яких в окремому файлі та має заголовний файл:

Клас MainWindow - у файлі mainwindow.cpp (розмір 17.8 КБ), заголовний файл - mainwindow.h.

Клас SearchCycles - у файлі search_cycles.cpp (розмір 6.33 КБ), заголовний файл – search_cycles.h.

Клас SearchCyclesUpdated - у файлі search_cycles_2.cpp (розмір 3 КБ), заголовний файл – search_cycles_2.h.

Клас DataCreation - у файлі data_creation.cpp (розмір 1.54 КБ), заголовний файл – data_creation.h.

Також містить точку входу, а саме функцію main() у файлі main.cpp (розмір – 183 байт).

Основні функції і можливості програми:

Вибір файлів з даними графа: Користувач може вибирати файли з вхідними даними графа для подальшого аналізу. Ці файли можуть містити інформацію про зв'язки між вершинами графа.

Ручне формування та редагування даних: Користувачеві доступна можливість створювати власні графи шляхом вручну введення даних про вершини та їхні зв'язки, або генерувати випадково граф на основі введення кількості вершин та ребер. Присутня можливість редагування файлів.

Візуалізація графа: Програма надає можливість відображення графа у вигляді візуального зображення, що полегшує розуміння структури графа.

Анімація алгоритму обходу графа: Користувач може спостерігати за процесом обходу графа та виявленням циклів у ньому через візуальну анімацію.

Вибір алгоритму: Користувач може вибрати алгоритм для пошуку циклів.

Виведення всіх циклів: Програма надає змогу вивести всі виявлені цикли у графі.

Інформація про використаний алгоритм: Користувачеві надається інформація про використаний алгоритм пошуку циклів, а саме: шлях обходу графа, а також час, необхідний для виконання цього алгоритму.

4.2 Вхідна та вихідна інформація

Формат вхідних даних: Програма може приймати дані з файлу, де міститься вказані дані про кількість вершин та ребер, а також список суміжних вершин (ребер). Наприклад:

```
7 11
1 2
2 3
2 4
3 1
4 1
4 3
4 4
5 6
5 7
```

```
6 2
7 4
```

Також можна створити випадковий граф на основі вказаної кількості вершин та ребер.

В програмі передбачено перевірку правильності вказання ребер, тобто чи вершина не дорівнює 0 та невід’ємна, також чи вершина не перевищує вказану кількість вершин.

Формат вихідних даних: Результати можуть бути представлені у вигляді списку циклів, виведених у текстовому вигляді або графічно на візуальному вікні програми, також у вигляді анімації обходу графа. Буде доступна для виведення інформація про шлях проходження графа, та час виконання алгоритму. Проміжні повідомлення про стан виконання програми або етапи обробки будуть виведені користувачеві внизу вікна.

Наприклад, при введені попередніх даних, буде виведено таку інформацію про цикли:

```
Граф має наступні цикли:
----------
1->2->3->1
1->2->4->1
1->2->4->3->1
```

Таблиця 1

Таблиця для порівняння часу виконання алгоритмів для різного набору вхідних даних

Кількість вершин	Кількість ребер	Час виконання (с) DFS з повторенням вершин	Час виконання (с) DFS без повторення вершин
50	80	0.213	0.034
100	150	2.073	0.195
200	300	13.048-46.891	0.296-11.084

4.3 Структура програми, опис основних функцій

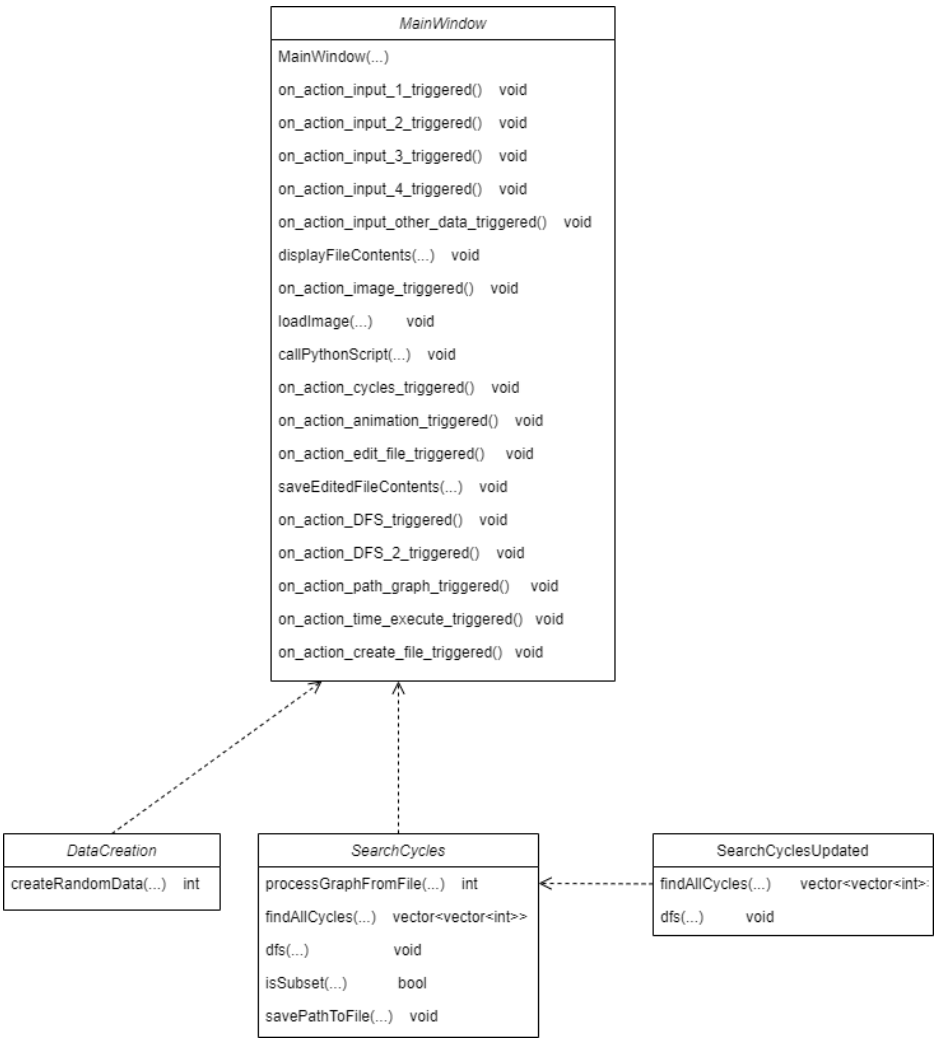

Рис. 2 – Діаграма класів

Файл main.cpp

У файлі main.cpp міститься функція *int main(int argc, char *argv[])* – ця функція є точко входу, яка створює об'єкт додатку, ініціалізує головне вікно програми, відображає його на екрані.

Файл search_cycles.cpp

Файл містить клас SearchCycles та наступні методи:

int processGraphFromFile(const QString &filePath, int selectionAlgorithm) – метод, який зчитує дані з файлу та за, параметром є шлях до файлу, та номер вибраного алгоритму.

vector<vector<int>> findAllCycles(vector<vector<int>>& graph) – метод, в якому викликається функція dfs, а також формуються унікальні цикли, повертає унікальні цикли.

void dfs(int v, vector<vector<int>>& graph, vector<bool>& visited, vector<int>& path, set<vector<int>>& cycles) – метод, для обходу в глибину графа та пошуку циклів, параметрами є вершина, ребра графа, вектор для визначення відвіданих вершин, вектор для шляху проходження, структура множин для зберігання циклів.

bool isSubset(const vector<int>& a, const vector<int>& b) - метод, для перевірки чи є цикл підмножино іншого циклу.

void savePathToFile(const vector<int>& path) – метод для запису у файл пройденого шляху, параметром є пройдений шлях.

Файл search_cycles_2.cpp

Файл містить клас SearchCyclesUpdated та наступні методи:

vector<vector<int>> findAllCycles(vector<vector<int>>& graph) – метод, в якому викликається функція dfs, а також формуються унікальні цикли, повертає унікальні цикли.

void dfs(int v, vector<vector<int>>& graph, vector<bool>& visited, vector<bool>& visit, vector<bool>& used, vector<int>& path, set<vector<int>>& cycles)– метод, для обходу в глибину графа та пошуку циклів, параметрами є вершина, ребра графа, вектори для визначення відвіданих вершин, вектор вже відвіданих вершин, вектор для позначення відвіданих вершин в методі, для шляху проходження, структура множин для зберігання циклів.

Файл data_creation.cpp

Файл містить клас DataCreation та наступні методи:

int createRandomData(int numVertices, int numEdges) – метод, генерування випадкових ребер на основі вказаної кількості вершин та ребер, та занесення цих даних у файл, параметрами є кількість вершин та ребер.

Файл mainwindow.cpp

Файл містить клас MainWindow та наступні методи:

MainWindow(QWidget *parent) - конструктор класу MainWindow в Qt, призначений для ініціалізації головного вікна програми та налаштування його інтерфейсу.

Наступні методи призначенні для вибору шляху до різних файлів з даними:

void on_action_input_1_triggered()

void on_action_input_2_triggered()

void on_action_input_3_triggered()

void on_action_input_4_triggered()

void on_action_input_other_data_triggered()

void displayFileContents(const QString &fileName) – метод для відображення даних у вікні, параметром є вибраний файл з даним.

void on_action_image_triggered() – метод, який викликає функції для генерування та відображення зображення.

void loadImage(const QString& fileName) – метод для вставлення зображення у праву частину вікна.

void callPythonScript(const QString &graphFile) – метод, який викликає функцію Python для генерування зображення графу та анімацію його проходження.

void on_action_cycles_triggered() – метод для виведення циклів в окремому вікні.

void on_action_animation_triggered() – метод для вставлення анімації проходження графу у вікні.

void on_action_edit_file_triggered() – метод для редагування відкритого файлу.

void saveEditedFileContents(const QString &fileName, const QString &content) – метод для збереження редагованого файлу.

void on_action_DFS_triggered() – метод для вибору алгоритму DFS з повторенням вершин.

void on_action_DFS_2_triggered() – метод для вибору алгоритму DFS без повторення вершин.

void on_action_path_graph_triggered() – метод для виведення шляху обходу графа в окремому вікні.

void on_action_time_execute_triggered() – метод для виведення часу виконання алгоритму.

void on_action_create_file_triggered() – метод для виведення вікна для ведення кількості вершин та кількості ребер для формування випадкових ребер графа.

4.4 Середовище реалізації програми

Програма розроблена на Windows 11 на мові C++ з використанням фреймворку Qt, вона може виконуватися у відповідних інтегрованих середовищах розробки, таких як Qt Creator або Microsoft Visual Studio, налаштованих для роботи з Qt.

5. Інструкція користувачеві програми

Користувач під час запуску програми, в першу чергу, повинен вибрати в меню вхідні дані. Існує три варіанти. Першим варіантом є можливість вибору наявного файлу, який містить вхідні дані про граф. Другий варіант - це вибрати інший файл для використання його як вхідних даних. Третій - створення нового файлу, де користувач може вказати кількість вершин та ребр для створення графа (вибрані дані з файлу виводять у лівій частині вікна).

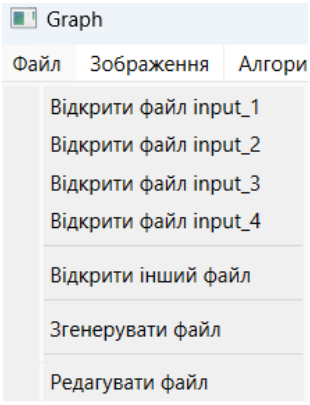

Рис. 3 – Вибір вхідних даних

Після введення даних, користувач має можливість в меню обрати алгоритм для пошуку циклів у орієнтованому графі серед запропонованих варіантів.

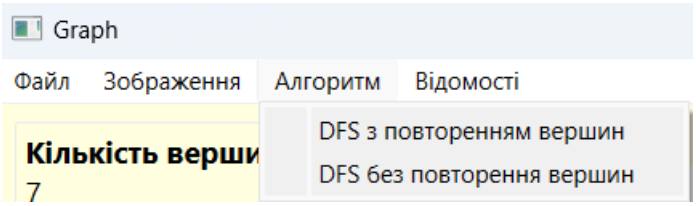

Рис. 4 – Вибір алгоритму для пошуку циклів

Після введення даних та вибору алгоритму, користувач має можливість вивести зображення графа (в правій частині вікна поряд з вхідними даними), побудованого на основі введених даних, а також переглянути анімацію, яка демонструє проходження графа за допомогою обраного алгоритму.

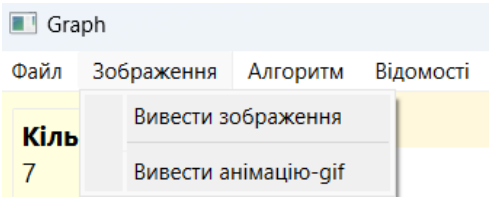

Рис. 5 – Можливість виведення зображення та анімації

Після цього користувачу стає доступною інформація-відомості про знайдені цикли в графі, шлях проходження графа та час, який зайняв обраний алгоритм для виконання. Ця інформація виводиться в окремому вікні для зручного перегляду.

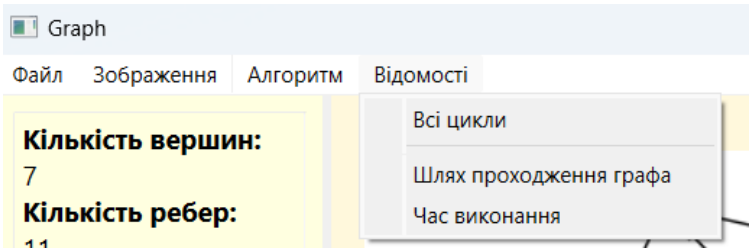

Рис. 6 – Вибір інформації про роботу алгоритму

Також, користувач може редагувати файли з вхідними даними в окремому вікні програми, щоб вносити зміни та оновлювати інформацію, яка використовується для аналізу графа. Це надає можливість контролювати та змінювати дані, що використовуються для обробки графів у програмі.

В нижній частині вікна програми присутня область, де відображається корисна інформація про виконання програми, стан обробки або робочого процесу, а також додаткові підказки чи повідомлення для користувача.

У правій частині вікна програми розташоване меню швидкого доступу, яке надає зручний доступ до основних опцій програми, а саме: вивести інформацію з файлу, вивести зображення графа, анімацію проходження графа, редагування файлу, пошук циклів в графі.

6. Контрольні приклади та результати їх реалізації

При запуску програми користувач бачить вікно з наступним розташуванням елементів: у верхній частині вікна розміщене меню, де можна здійснити різні дії; ліворуч під меню розташована область для виведення вибраних даних; праворуч від неї розміщене вікно для відображення зображення графа та анімації його проходження.

Праворуч від основної області знаходиться область швидкого доступу, яка містить список функцій для швидкого виклику. У нижній частині вікна розташована область, де виводиться інформація про обрану операцію або корисні підказки для користувача."

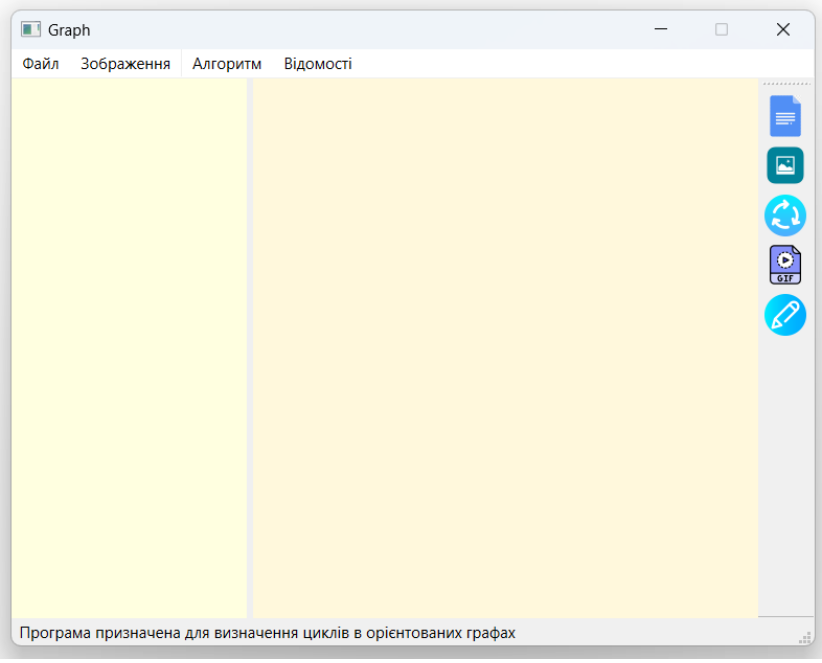

Рис. 7 – Головне вікно програми

Для вибору файлу з вхідними даними, користувач може скористатися опцією 'Файл' у меню програми.

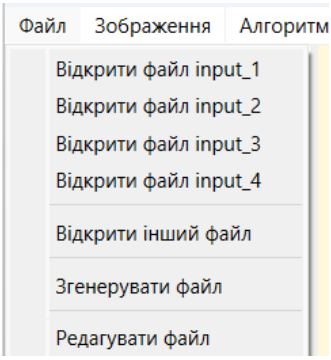

Рис. 8 – Опція Файл в меню програми

Відбувається виведення вибраних даних.

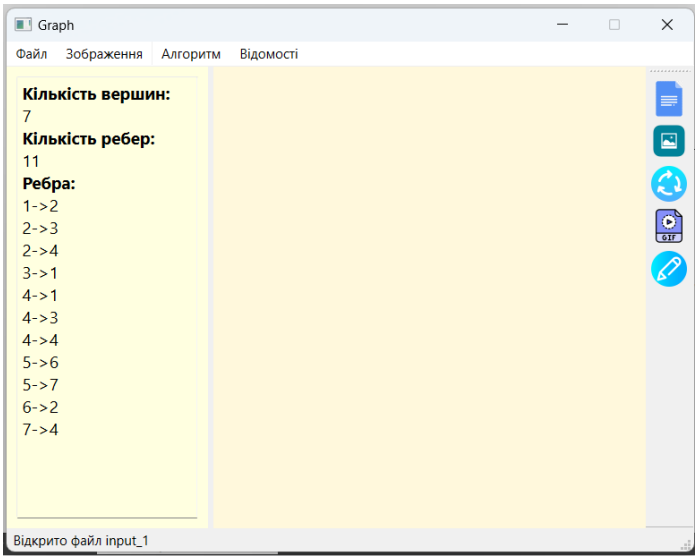

Рис. 9 – Відображення вхідних даних

Якщо користувач вибрав функцію згенерувати файл, відкривається вікно для введення даних.

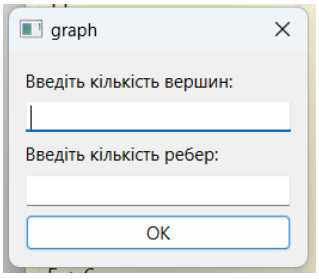

Рис. 10 – Створення файлу з даними

Далі вибір алгоритму для пошуку графа опція 'Алгоритм' у меню програми.

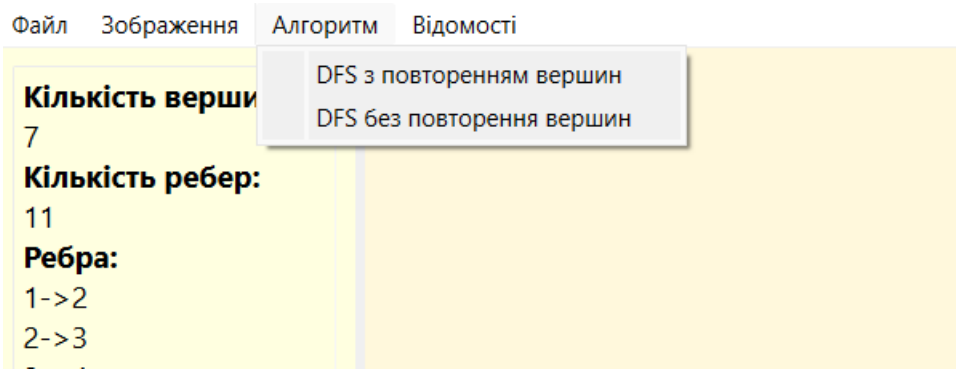

Рис. 11 – Вибір алгоритму в меню

Відображається кожна дія користувача в нижній області:

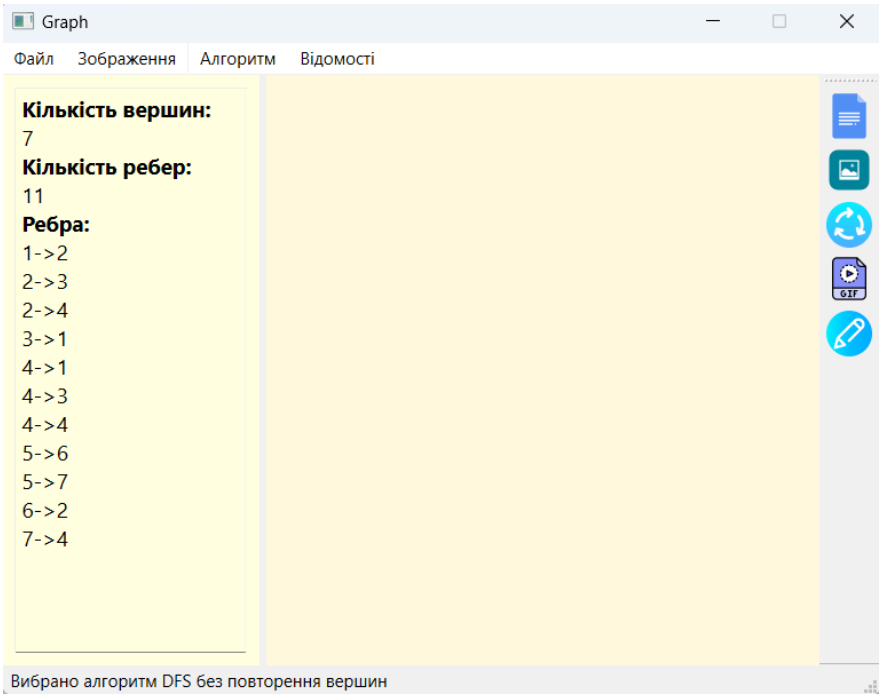

Рис. 12 – Вибір алгоритму в меню

Користувач має можливість вивести зображення графа в правій частині вікна через опцію 'Зображення' в меню, або меню швидкого доступу.

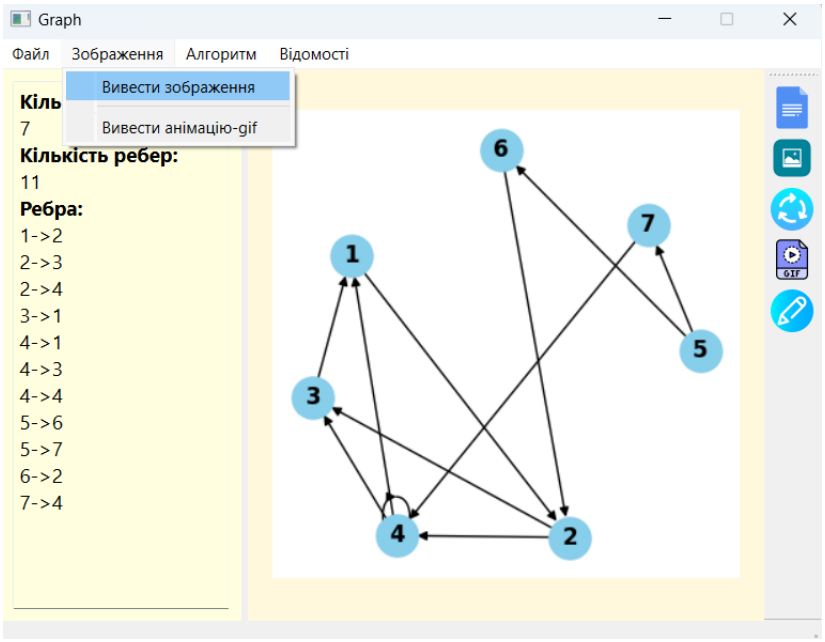

Рис. 13 – Вивід зображення графа

Присутня можливість виведення анімації проходження графа в правій частині вікна через опцію 'Зображення – Вивести анімацію-gif' в меню, або меню швидкого доступу.

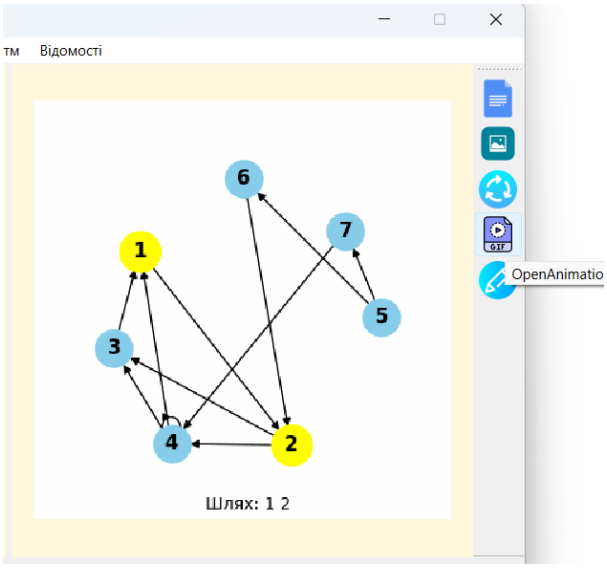

Рис. 14 – Відображення анімації проходження графа

Можливість переглянути відомості, а саме: всі цикли, шлях проходження графа, час виконання алгоритму на основі вибраного алгоритму через опцію 'Відомості' в меню, які будуть виведені в окремому вікні.

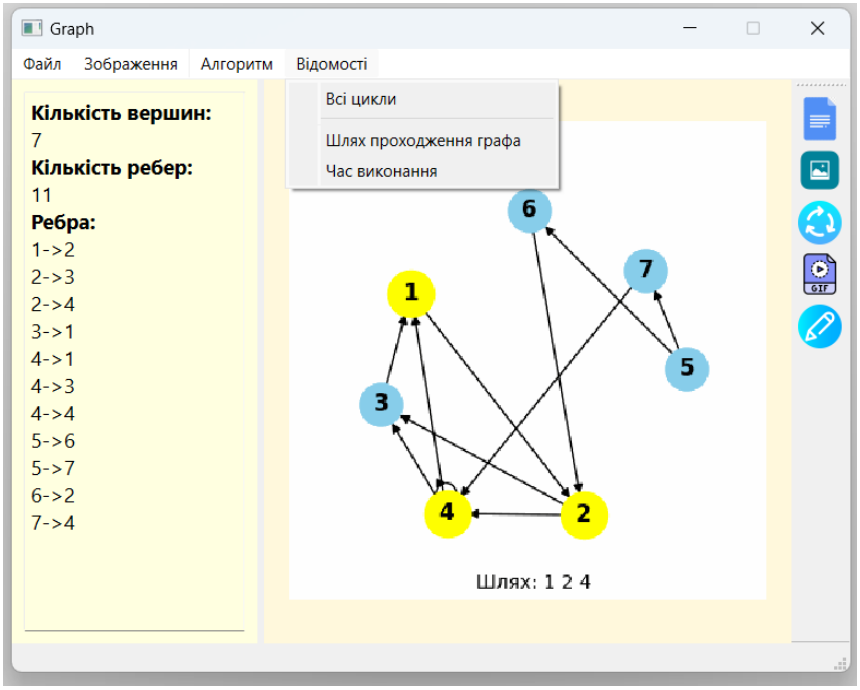

Рис. 15 – Опція Відомості в меню програми

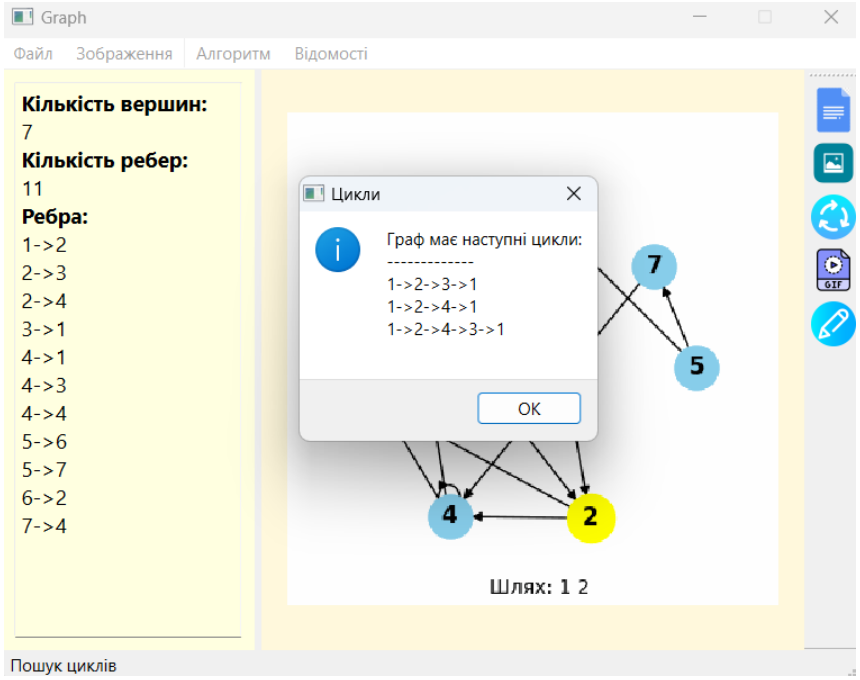

Рис. 16 – Вікно зі знайденими циклами

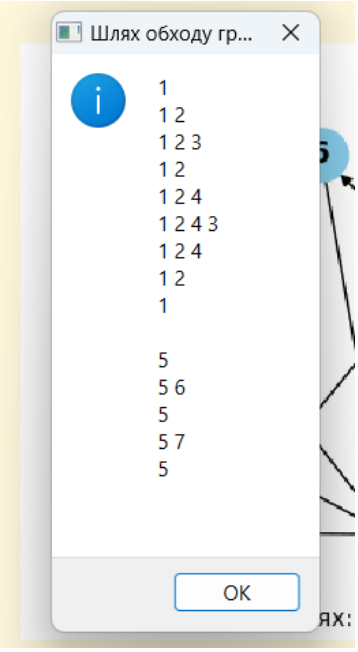

Рис. 17 – Вікно зі шляхом проходження графа

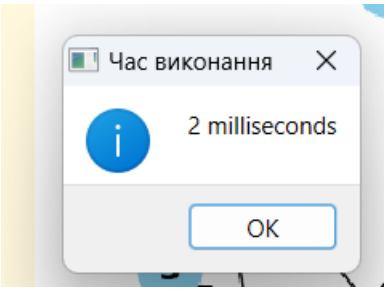

Рис. 18 – Вікно з часом виконання алгоритму

Можливість редагування файлу в опції ‘Файл-Редагувати файл’.

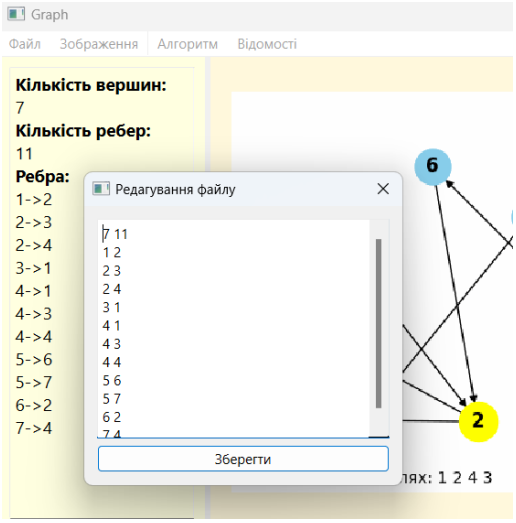

Рис. 19 – Вікно для редагування файлу

Результати роботи програми для інших вхідних даних:

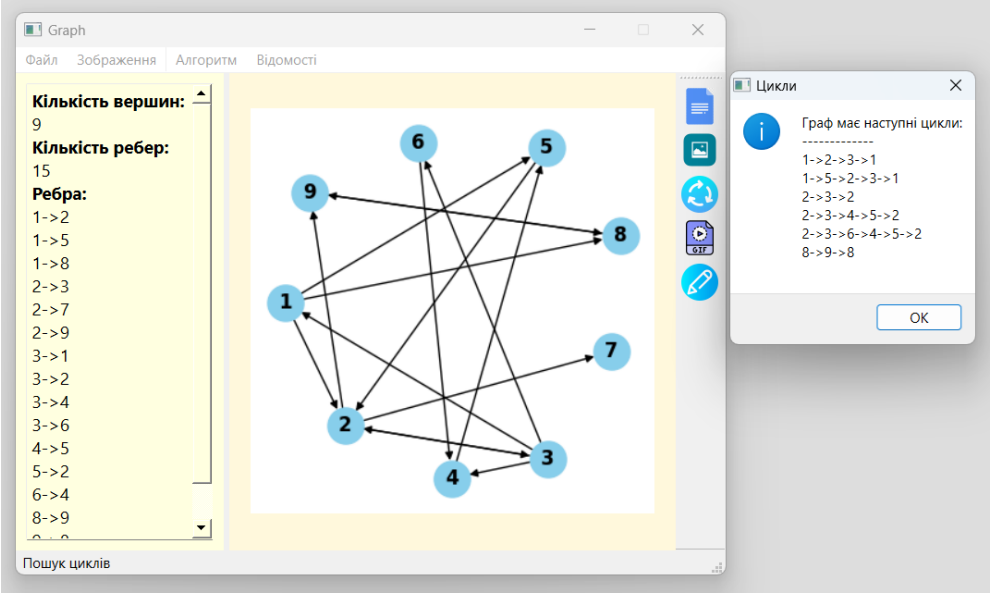

Рис. 20 – Виконання програми для файлу input_4

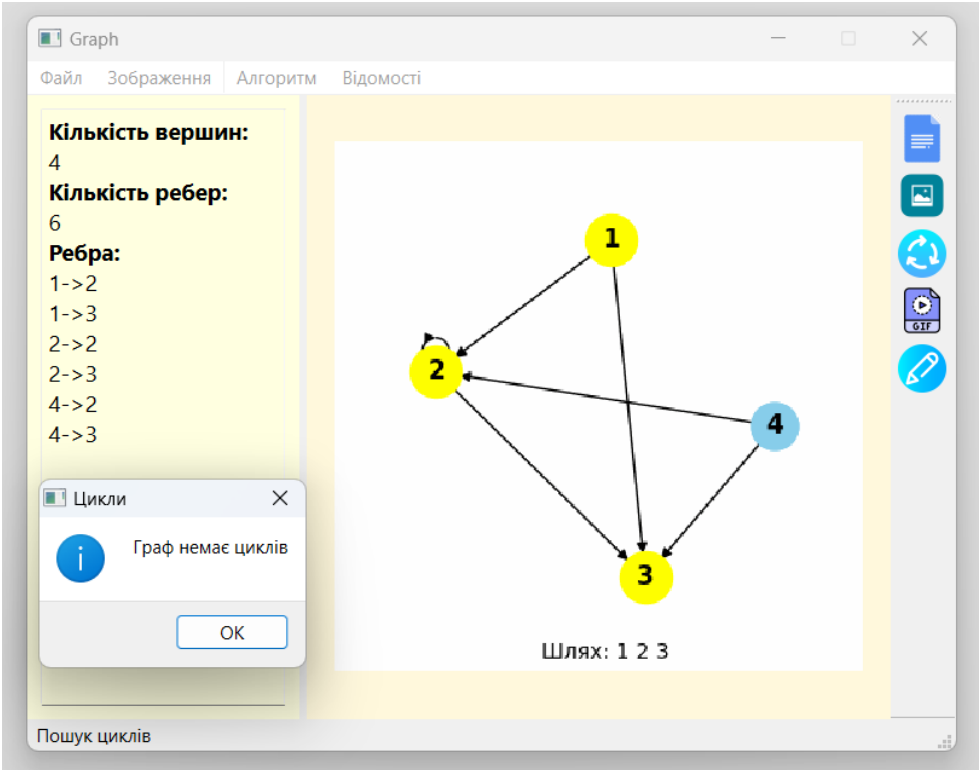

Рис. 21 – Виконання програми для файлу input_3

Висновки

Представлений програмний продукт є інструментом для виявлення всіх циклів у орієнтованих графах. Однією з ключових особливостей цього програмного продукту є поєднання можливостей мови програмування C++ та фреймворку Qt для створення крос-платформеного програмного забезпечення, що сприяє швидкій реалізації функціоналу та забезпечує зручний інтерфейс користувача.

Програма використовує алгоритми на основі DFS (Depth-First Search) для ефективного виявлення циклів у графах. Використання цих алгоритмів дозволяє ретельно проходити всі вершини графа, шукаючи цикли і виявляючи послідовності вершин, які утворюють ці цикли.

Можливі області застосування цієї програми включають оптимізацію та моделювання різноманітних систем, оптимізацію транспортних систем. Подальше вдосконалення продукту включає розширення функціоналу програми, додавання нових алгоритмів для пошуку циклів та покращення інтерфейсу користувача для забезпечення більшої зручності та функціональності.

Усі ці аспекти роблять розроблений програмний продукт важливим інструментом для аналізу орієнтованих графів, який має потенціал для подальшого розвитку.

Список використаної літератури

1. "Тім Рафгарден. Досконалий алгоритм. Графові алгоритми та структури даних: Пер. з англ., 2019. – 256 с."

2. Перевірка орієнтованого графа на наявність циклів. Режим доступу: https://www.mathros.net.ua/perevirka-orijentovanogo-grafa-na-najavnist-cykliv.html

3. "Sartaj Sahni. Data Structures, Algorithms, And Applications In C++: 2-nd ed. – Silicon Pr., 2004. – 792 p."

4. Цикл (теорія графів). Режим доступу: https://jak.koshachek.com/articles/cikl-teorija-grafiv.html

5. "Ніклаус Вірт. Алгоритми та структури даних: Пер. з англ. – СПб.: Невський діалект, 2014. – 274 с."

6. Пошук в глибину. Режим доступу: https://algoua.com/algorithms/graphs/dfs/

7. "John Paul Mueller, Luca Massaron. Algorithms For Dummies (For Dummies (Computers)): 1st ed. – For Dummies; 2017. – 432 p."

Додатки

Додаток 1. Текст всіх програмних файлів

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл data_creation.h

```
#ifndef DATA_CREATION_H
#define DATA_CREATION_H

class DataCreation
{
public:
    DataCreation();
    static int createRandomData(int numVertices, int numEdges);
};

#endif // DATA_CREATION_H
```

Файл data_creation.cpp

```
#include "data_creation.h"
```

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <set>
using namespace std;

DataCreation::DataCreation() {}

int DataCreation::createRandomData(int numVertices, int numEdges) {

    srand(time(0));

    set<pair<int, int>> edges;

    while (edges.size() < numEdges) {
        int from = rand() % numVertices + 1; // Випадкова вершина від 1 до numVertices
        int to = rand() % numVertices + 1;

        if (from != to) {
            edges.insert({from, to});
        }
    }

    ofstream outputFile("D:/Qt/graphtestproject/graph.txt");

    if (outputFile.is_open()) {
        outputFile << numVertices << " " << numEdges << endl;

        for (const auto &edge : edges) {
            outputFile << edge.first << " " << edge.second << endl;
        }

        outputFile.close(); // Закриття файлу
        cout << "Файл 'graph.txt' створено з успіхом." << endl;
    } else {
```

```
        cout << "Неможливо відкрити файл для запису." << endl;
    }


    return 0;
}
```

Файл search_cycles.h

```
#ifndef SEARCH_CYCLES_H
#define SEARCH_CYCLES_H

#include <vector>
#include <set>
#include <QString>

class SearchCycles
{
public:
    SearchCycles();
    static void dfs(int v, std::vector<std::vector<int>>& graph, std::vector<bool>& visited, std::vector<int>& path, std::set<std::vector<int>>& cycles);
    static bool isSubset(const std::vector<int> &a, const std::vector<int> &b);
    static bool isLoop(const std::vector<int> &cycle);
    static std::vector<std::vector<int>> findAllCycles(std::vector<std::vector<int>> &graph);
    static int processGraphFromFile(const QString &filePath, int selectionAlgorithm);
    static void savePathToFile(const std::vector<int>& path);
};

#endif // SEARCH_CYCLES_H
```

Файл search_cycles.cpp

```
#include <QString>

#include <algorithm>

#include <iostream>

#include <set>

#include <vector>

#include <fstream>

#include <chrono>


#include "search_cycles.h"

#include "search_cycles_2.h"
```

```
using namespace std;


//Збереження шляху до файлу
void SearchCycles::savePathToFile(const vector<int>& path) {
    ofstream file("D:/Qt/graphtestproject/path.txt", ios::app);


    if (file.is_open()) {
        for (int node : path) {
            file << node << " ";
        }
        file << endl;
        file.close();
    } else {
        cout << "Не вдалося відкрити файл" << endl;
    }
}


//Обхід в глибину
void SearchCycles::dfs(int v, vector<vector<int>>& graph, vector<bool>& visited, vector<int>& path,
set<vector<int>>& cycles) {
    visited[v] = true;
    path.push_back(v);


    SearchCycles::savePathToFile(path);


    for (int u : graph[v]) {
        if (!visited[u]) { // Якщо вершина не відвідана, запускаємо DFS для неї
            dfs(u, graph, visited, path, cycles);
        } else if (find(path.begin(), path.end(), u) != path.end()) {
            // Якщо вершина вже в поточному шляху, знайдено цикл
            vector<int> cycle;
            auto it = find(path.begin(), path.end(), u);
            for (auto iter = it; iter != path.end(); ++iter) {
                cycle.push_back(*iter);
                cout << *iter;
            }
```

```
            cout << "\n";
            cycles.insert(cycle);
        }
    }
    path.pop_back(); // Після проходження, видаляємо вершину з поточного шляху
    SearchCycles::savePathToFile(path);
    visited[v] = false; // Позначаємо вершину як невідвідану для інших шляхів
}

// Перевірка, чи один вектор є підмножиною іншого
bool SearchCycles::isSubset(const vector<int>& a, const vector<int>& b) {
    if (a.size() > b.size()) {
        return false;
    }

    for (int vertex : a) {
        if (find(b.begin(), b.end(), vertex) == b.end()) {
            return false;
        }
    }

    return true;
}

// Функція знаходження усіх циклів у графі
vector<vector<int>> SearchCycles::findAllCycles(vector<vector<int>>& graph) {
    int n = graph.size();
    set<vector<int>> cycles; // Множина для зберігання унікальних циклів
    vector<bool> visited(n, false); // Масив для відстеження відвіданих вершин
    vector<int> path; // Шлях DFS

    for (int i = 1; i < n; ++i) {
        dfs(i, graph, visited, path, cycles); // Запуск DFS для кожної вершин
    }

    vector<vector<int>> uniqueCycles;
```

```
    // Фільтрація унікальних циклів та перевірка на замкнутість
    for (const auto& cycle : cycles) {
        if (cycle.size() > 1) {
            bool isSubsetOfOthers = false;

            // Перевірка, чи цей цикл є підмножиною іншого циклу
            for (const auto& existingCycle : uniqueCycles) {
                if (SearchCycles::isSubset(cycle, existingCycle) && cycle.size() == existingCycle.size()) {
                    isSubsetOfOthers = true;
                    break;
                }
            }

            // Якщо цей цикл не є підмножиною іншого циклу, додаємо його
            if (!isSubsetOfOthers) {
                uniqueCycles.push_back(cycle);
            }
        }
    }

    return uniqueCycles;
}

int SearchCycles::processGraphFromFile(const QString &filePath, int selectionAlgorithm)
{
    auto start_time = chrono::steady_clock::now();

    std::string file = filePath.toStdString();
    std::ifstream input(file);

    ofstream path("D:/Qt/graphtestproject/path.txt", ios::trunc);
    path.close();

    if (!input.is_open()) {
        cerr << "Error: Не вдалося відкрити файл" << endl;
        return 1;
    }
```

```
std::ofstream output("D:/Qt/graphtestproject/output.txt");

int vertexCount, edgeCount;
input >> vertexCount >> edgeCount;
//cerr << vertexCount << edgeCount;

vector<vector<int>> graph(vertexCount + 1);

for (int i = 0; i < edgeCount; ++i) {
    int from, to;
    input >> from >> to;
    //cerr << from << to;

    if (from < 1 || from > vertexCount || to < 1 || to > vertexCount) {
        cerr << "Error Неправильно вказані вершини!" << endl;
        return 1; // Помилка
    }

    graph[from].push_back(to);
}
vector<vector<int>> cycles;

if (selectionAlgorithm == 1)
    cycles = findAllCycles(graph);
else
    cycles = SearchCyclesUpdated::findAllCycles(graph);

if (!output.is_open()) {
    cerr << "Error: Не вдалося відкрити файл для запису" << endl;
    return 1;
}
freopen("D:/Qt/graphtestproject/output.txt", "w", stdout);

if (cycles.empty()) {
    output << "Граф немає циклів" << endl;
} else {
```

```
        output << "Граф має наступні цикли:" << endl;
        output << "-------------\n";
        for (const auto &cycle : cycles) {
            for (size_t i = 0; i < cycle.size(); ++i) {
                output << cycle[i] << "->";
            }
            output << cycle[0] << endl; // Додано виведення першої вершини на кінець
        }
    }

    std::ofstream out("D:/Qt/graphtestproject/time_execution.txt");
    if (!out.is_open()) {
        cerr << "Error: Не вдалося відкрити файл для запису" << endl;
        return 1;
    }
    freopen("D:/Qt/graphtestproject/time_execution.txt", "w", stdout);

    auto end_time = chrono::steady_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(end_time - start_time);
    out << duration.count() << " milliseconds";
    cerr << "Час виконання алгоритму: " << duration.count() << " milliseconds" << endl;

    return 0;
}
```

Файл search_cycles_2.h

```
#ifndef SEARCH_CYCLES_2_H
#define SEARCH_CYCLES_2_H

#include <vector>
#include <set>

class SearchCyclesUpdated
```

```
{
public:
    SearchCyclesUpdated();
    static void dfs(int v, std::vector<std::vector<int>>& graph, std::vector<bool>& visited, std::vector<bool>& visit,
                    std::vector<bool>& used, std::vector<int>& path, std::set<std::vector<int>>& cycles);
    static std::vector<std::vector<int>> findAllCycles(std::vector<std::vector<int>> &graph);
};

#endif // SEARCH_CYCLES_2_H
```

Файл search_cycles_2.cpp

```
#include "search_cycles_2.h"
#include "search_cycles.h"

#include <algorithm>
#include <iostream>
#include <set>
#include <vector>

using namespace std;

// Обхід в глибину
void SearchCyclesUpdated::dfs(int v, vector<vector<int>>& graph, vector<bool>& visited, vector<bool>& visit,
vector<bool>& used, vector<int>& path, set<vector<int>>& cycles) {
    visited[v] = true;
    used[v] = true;
    path.push_back(v);

    SearchCycles::savePathToFile(path);

    for (int u : graph[v]) {
        if (!used[u] && !visit[u]) { // Якщо вершина не відвідана, запускаємо DFS для неї
            dfs(u, graph, visited, visit, used, path, cycles);
        } else if (find(path.begin(), path.end(), u) != path.end()) {
            // Якщо вершина вже в поточному шляху, знайдено цикл
            vector<int> cycle;
```

```
            auto it = find(path.begin(), path.end(), u);
            for (auto iter = it; iter != path.end(); ++iter) {
                cycle.push_back(*iter);
                cout << *iter;
            }
            cout << "\n";
            cycles.insert(cycle);
        }
    }
    path.pop_back();
    SearchCycles::savePathToFile(path);
    used[v] = false;
}

//Пошук всіх циклів без повторення вершин
vector<vector<int>> SearchCyclesUpdated::findAllCycles(vector<vector<int>>& graph) {
    int n = graph.size();
    set<vector<int>> cycles; // Множина для зберігання унікальних циклів
    vector<bool> used(n, false);
    vector<bool> visited(n, false);
    vector<bool> visit(n, false);
    vector<int> path; // Шлях DFS

    for (int i = 1; i < n; ++i) {
        if (!visited[i]) {
            dfs(i, graph, visited, visit, used, path, cycles);
        }
        visit[i] = true;
    }

    vector<vector<int>> uniqueCycles;

    // Фільтрація унікальних циклів та перевірка на замкнутість
    for (const auto& cycle : cycles) {
        if (cycle.size() > 1) {
            bool isSubsetOfOthers = false;
```

```
            for (const auto& existingCycle : uniqueCycles) {
                if (SearchCycles::isSubset(cycle, existingCycle) && cycle.size() == existingCycle.size()) {
                    isSubsetOfOthers = true;
                    break;
                }
            }

            if (!isSubsetOfOthers) {
                uniqueCycles.push_back(cycle);
            }
        }
    }

    return uniqueCycles;
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSplitter>
#include <QLabel>

#include <QDialog>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void displayFileContents(const QString &fileName);
    void callPythonScript(const QString &filePath);
    void loadImage(const QString& fileName);
    void saveEditedFileContents(const QString &fileName, const QString &content);
    void onOkButtonClicked();


private slots:
    void on_action_input_1_triggered();

    void on_action_input_2_triggered();

    void on_action_input_other_data_triggered();

    void on_action_input_3_triggered();

    void on_action_input_4_triggered();

    void on_action_edit_file_triggered();

    void on_action_DFS_triggered();

    void on_action_DFS_2_triggered();

    void on_action_path_graph_triggered();

    void on_action_time_execute_triggered();

    void on_action_create_file_triggered();
```

```
    void on_action_image_triggered();

    void on_action_animation_triggered();

    void on_action_cycles_triggered();

private:
    Ui::MainWindow *ui;
    int change;
    QLabel *textLabel;
    QFrame *line_down;
    QSplitter *splitter;
    QWidget *leftWidget;
    QWidget *rightWidget;
    QLabel *imageLabel;
    QWidget *imageWidget;
    QString filePath;
    int numberImage = 0;
    int selectionAlgorithm = 0;
    QMovie *movie;

    QLabel *verticesLabel;
    QLabel *edgesLabel;
    QLineEdit *verticesLineEdit;
    QLineEdit *edgesLineEdit;
    QPushButton *okButton;
    QVBoxLayout *layout;
    int vertices;
    int edges;

};

#endif // MAINWINDOW_H
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "search_cycles.h"
#include "data_creation.h"
#include "./ui_mainwindow.h"
#include <QFileDialog>
#include <QFile>
#include <QTextStream>
#include <QTextEdit>
#include <QLabel>
#include <QHBoxLayout>
#include <QPainter>
#include <QSplitter>
#include <QToolBar>
#include <QProcess>
#include <QString>
#include <QMessageBox>
#include <QMovie>
#include <QPushButton>



MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setFixedSize(614, 456);
    statusBar()->showMessage(tr("Програма призначена для визначення циклів в орієнтованих графах"));

    QSplitter *splitter = new QSplitter(Qt::Horizontal, this);
    splitter->setObjectName("image_splitter");

    leftWidget = new QWidget(splitter);
    rightWidget = new QWidget(splitter);


    // Встановлення фонового кольору тільки для центрального віджета
    leftWidget->setStyleSheet("background-color: #FFFFE0;");
    rightWidget->setStyleSheet("background-color: #FFF8DC;");


    splitter->addWidget(leftWidget);
    splitter->addWidget(rightWidget);

    leftWidget->setMaximumWidth(180);

    setCentralWidget(splitter);

    change = 0;


    imageLabel = new QLabel(this);
    imageWidget = nullptr;

    // Створення тулбару та його налаштування
    QToolBar *toolbar = new QToolBar("MyToolBar", this);
    toolbar->setIconSize(QSize(32, 32)); // Налаштування розміру піктограм
    addToolBar(Qt::RightToolBarArea, toolbar); // Додавання тулбару
```

```
    QAction *actionOpenFile = new QAction(QIcon("D:/Qt/graphtestproject/pictures/file_2.png"), tr("OpenFile"),
this);
    connect(actionOpenFile, &QAction::triggered, this, &MainWindow::on_action_input_1_triggered);
    toolbar->addAction(actionOpenFile);

    QAction *actionOpenImage = new QAction(QIcon("D:/Qt/graphtestproject/pictures/picture_2.png"),
tr("OpenImage"), this);
    connect(actionOpenImage, &QAction::triggered, this, &MainWindow::on_action_image_triggered);
    toolbar->addAction(actionOpenImage);

    QAction *actionSearchCycle = new QAction(QIcon("D:/Qt/graphtestproject/pictures/cycle.png"),
tr("SearchCycle"), this);
    connect(actionSearchCycle, &QAction::triggered, this, &MainWindow::on_action_cycles_triggered);
    toolbar->addAction(actionSearchCycle);

    QAction *actionNextImage = new QAction(QIcon("D:/Qt/graphtestproject/pictures/animation.png"),
tr("OpenAnimation"), this);
    connect(actionNextImage, &QAction::triggered, this, &MainWindow::on_action_animation_triggered);
    toolbar->addAction(actionNextImage);

    QAction *actionBackImage = new QAction(QIcon("D:/Qt/graphtestproject/pictures/edit.png"), tr("EditFile"), this);
    connect(actionBackImage, &QAction::triggered, this, &MainWindow::on_action_edit_file_triggered);
    toolbar->addAction(actionBackImage);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::callPythonScript(const QString &graphFile) {
    QString program = "python";  // Шлях до встановленого Python на комп'ютері
    QStringList arguments;

    arguments << "D:/Qt/graphtestproject/video_graph.py" << graphFile;

    QProcess *process = new QProcess(this);
    process->start(program, arguments);
    process->waitForFinished(-1);  // Чекаємо завершення процесу Python
}



//----------------Відкриття файлів--------------
void MainWindow::on_action_input_1_triggered()
{
    filePath = "D:/Qt/graphtestproject/input_1.txt";
    ui->statusbar->showMessage("Відкрито файл input_1");
    displayFileContents(filePath);
    change = 1;
}


void MainWindow::on_action_input_2_triggered()
{
    filePath = "D:/Qt/graphtestproject/input_2.txt";
    ui->statusbar->showMessage("Відкрито файл input_2");
    displayFileContents(filePath);
    change = 2;
}

void MainWindow::on_action_input_3_triggered()
```

```
{
    filePath = "D:/Qt/graphtestproject/input_3.txt";
    ui->statusbar->showMessage("Відкрито файл input_3");
    displayFileContents(filePath);
    change = 3;
}

void MainWindow::on_action_input_4_triggered()
{
    filePath = "D:/Qt/graphtestproject/input_4.txt";
    ui->statusbar->showMessage("Відкрито файл input_4");
    displayFileContents(filePath);
    change = 4;
}

void MainWindow::on_action_input_other_data_triggered()
{
    filePath = QFileDialog::getOpenFileName(this, tr("Оберіть файл"), "", tr("Текстові файли (*.txt)"));
    if (!filePath.isEmpty()) {
        ui->statusbar->showMessage(tr("Відкрито файл: ") + filePath);
        displayFileContents(filePath);
        change = 5;
    } else {
        qDebug() << "Скасовано вибір файлу";
    }
}

//Відображення з файлу у лівій частині
void MainWindow::displayFileContents(const QString &fileName) {
    QFile file(fileName);

    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream in(&file);

        int vertexCount, edgeCount;
        in >> vertexCount >> edgeCount;
        QString result = "<b>Кількість вершин:</b><br>" + QString::number(vertexCount) + "<br>";

        result += "<b>Кількість ребер:</b><br>" + QString::number(edgeCount) + "<br>";
        result += "<b>Ребра:</b><br>";

        // Зчитуємо решту рядків (ребра) та додаємо до результуючого рядка
        for (int i = 0; i < edgeCount; ++i) {
            int from, to;
            in >> from >> to;
            result += QString::number(from) + "->" + QString::number(to) + "<br>";
        }

        QTextEdit *textEdit = findChild<QTextEdit*>("data_graph");

        if (!textEdit) {
            textEdit = new QTextEdit(this);
            textEdit->setObjectName("data_graph");
            textEdit->setReadOnly(true);

            QFont font = textEdit->font();
            font.setPointSize(11);
            textEdit->setFont(font);

            QSplitter *splitter = findChild<QSplitter*>("image_splitter");
            if (splitter) {
                QWidget *leftWidget = splitter->widget(0); // Лівий віджет
```

```
            QVBoxLayout *leftLayout = new QVBoxLayout(leftWidget);
            leftLayout->addWidget(textEdit);
            leftWidget->setLayout(leftLayout);

            leftWidget->setMaximumWidth(180);
            leftWidget->setMaximumHeight(QWIDGETSIZE_MAX);
        } else {
            qDebug() << "QSplitter не знайдено";
        }
    }

    textEdit->setHtml(result);

    file.close();
  }
}
//-----------------------------------------------------------------

//---------------------Вставлення зображення------------------
void MainWindow::on_action_image_triggered() {
  if (change > 0){
  QString fileName;
  fileName = "D:/Qt/graphtestproject/graph_image.png";
  if (selectionAlgorithm > 0){
    SearchCycles::processGraphFromFile(filePath, selectionAlgorithm);
    callPythonScript(filePath);
    loadImage(fileName);
  } else
     ui->statusbar->showMessage("Виберіть алгоритм!");
  } else
    ui->statusbar->showMessage("Виберіть файл з даними!");

}

void MainWindow::loadImage(const QString& fileName) {
  if (!fileName.isEmpty()) {
    QPixmap image(fileName);
    if (!image.isNull()) {
      if (!imageWidget) {
        imageWidget = new QWidget(this);
        QVBoxLayout *imageLayout = new QVBoxLayout(imageWidget);
        imageLabel->setPixmap(image);
        imageLayout->addWidget(imageLabel);

        QSplitter *splitter = findChild<QSplitter*>("image_splitter");
        if (splitter) {
          QWidget *rightWidget = splitter->widget(1); // Правий віджет

          QVBoxLayout *rightLayout = new QVBoxLayout(rightWidget);
          rightLayout->addWidget(imageWidget);
          rightWidget->setLayout(rightLayout);
          ui->statusbar->showMessage("Вставлено зображення");
        } else {
          qDebug() << "QSplitter не знайдено";
        }
      } else {
        imageLabel->setPixmap(image);
        ui->statusbar->showMessage("Оновлено зображення");
      }
    } else {
      qDebug() << "Помилка завантаження зображення";
```

```
        }
    }
}
//-----------------------------------------------------------------

//-----------------Пошук циклів і виведення в окремому вікні------------------
void MainWindow::on_action_cycles_triggered() {
    if (selectionAlgorithm > 0){
        SearchCycles::processGraphFromFile(filePath, selectionAlgorithm);
        ui->statusbar->showMessage("Пошук циклів");

        QString filename = "D:/Qt/graphtestproject/output.txt";
        QFile file(filename);

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::critical(this, "Помилка", "Не вдалося відкрити файл");
            return;
        }

        QTextStream in(&file);
        QString fileData = in.readAll();

        QMessageBox::information(this, "Цикли", fileData);
        in.flush(); // Очищення буфера вводу-виводу
        file.close();
    } else
        ui->statusbar->showMessage("Виберіть алгоритм!");
}
//-------------------------------------------------------------------------


//-------------Всталення анімацію gif--------------------------------
void MainWindow::on_action_animation_triggered()
{
    if (change > 0){
        QString fileName;
        fileName = "D:/Qt/graphtestproject/graph_animation.gif";

        movie = new QMovie(fileName);

        if (imageLabel) {
            imageLabel->setMovie(movie);
            movie->start();
            ui->statusbar->showMessage("Вставлено гіфку");
        } else {
            // Створення QLabel для відображення гіфки, якщо вона ще не створена
            imageLabel = new QLabel(this);
            QVBoxLayout *layout = new QVBoxLayout(rightWidget);
            layout->addWidget(imageLabel);
            rightWidget->setLayout(layout);

            movie->setScaledSize(imageLabel->size());

            imageLabel->setMovie(movie);
            movie->start();
            ui->statusbar->showMessage("Вставлено гіфку");
        }
    }
    else
        ui->statusbar->showMessage("Не вибрано файл або не вставлено зображення");
}
//-------------------------------------------------------------------------
```

```
//-------------------Редагування файлу-----------------------------------
void MainWindow::on_action_edit_file_triggered() {
    // Створення нового вікна для редагування файлу
    QDialog *editDialog = new QDialog(this);
    editDialog->setWindowTitle("Редагування файлу");

    QVBoxLayout *layout = new QVBoxLayout(editDialog);

    QTextEdit *textEdit = new QTextEdit(editDialog);
    textEdit->setObjectName("edit_text");
    textEdit->setReadOnly(false);

    // Відкриття файлу для редагування та відображення вмісту у QTextEdit
    if(change > 0){
        QFile file(filePath);
        if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QTextStream in(&file);
            textEdit->setPlainText(in.readAll());
            file.close();
        } else {
            delete editDialog; // Видалення вікна в разі невдалого відкриття файлу
            delete layout;
            return;
        }

        layout->addWidget(textEdit);

        QPushButton *saveButton = new QPushButton("Зберегти", editDialog);
        layout->addWidget(saveButton);
        editDialog->setLayout(layout);

        connect(saveButton, &QPushButton::clicked, this, [this, textEdit, editDialog]() {
            saveEditedFileContents(filePath, textEdit->toPlainText());
            editDialog->close();
        });

        editDialog->exec(); // Відобразити вікно редагування
    } else{
        ui->statusbar->showMessage("Не вдалося відкрити файл для редагування! Потрібно вибрати файл!");
        delete layout;
    }
}

void MainWindow::saveEditedFileContents(const QString &fileName, const QString &content) {
    QFile file(fileName);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        out << content; // Зберігаємо зміни до файлу
        file.close();
        ui->statusbar->showMessage("Файл успішно збережено");
    } else {
        qDebug() << "Не вдалося відкрити файл для запису";
    }
}
//-----------------------------------------------------------------------


//-----------------------Вибір алгоритму-----------------------------
void MainWindow::on_action_DFS_triggered()
{
```

```
    selectionAlgorithm = 1;
    ui->statusbar->showMessage("Вибрано алгоритм DFS з повторенням вершин");
}


void MainWindow::on_action_DFS_2_triggered()
{
    selectionAlgorithm = 2;
    ui->statusbar->showMessage("Вибрано алгоритм DFS без повторення вершин");
}
//-------------------------------------------------------------------------

//------------------------Вікно з пройденим шляхом-----------------------------
void MainWindow::on_action_path_graph_triggered()
{
    if (selectionAlgorithm > 0){
        SearchCycles::processGraphFromFile(filePath, selectionAlgorithm);
        ui->statusbar->showMessage("Шлях обходу графа");

        QString filename = "D:/Qt/graphtestproject/path.txt";
        QFile file(filename);

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::critical(this, "Помилка", "Не вдалося відкрити файл");
            return;
        }

        QTextStream in(&file);
        QString fileData = in.readAll(); // Зчитування даних з файлу

        QMessageBox::information(this, "Шлях обходу графа", fileData);
        in.flush();
        file.close();
    } else
        ui->statusbar->showMessage("Виберіть алгоритм!");
}



//------------------------Вікно з часом виконання алгоритму-----------------------------
void MainWindow::on_action_time_execute_triggered()
{
    if (selectionAlgorithm > 0){
        SearchCycles::processGraphFromFile(filePath, selectionAlgorithm);
        ui->statusbar->showMessage("Час виконання алгоритму");

        QString filename = "D:/Qt/graphtestproject/time_execution.txt";
        QFile file(filename);

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::critical(this, "Помилка", "Не вдалося відкрити файл");
            return;
        }

        QTextStream in(&file);
        QString fileData = in.readAll();

        QMessageBox::information(this, "Час виконання", fileData);
        in.flush();
        file.close();
    } else
        ui->statusbar->showMessage("Виберіть алгоритм!");
```

```
}


//--------------------Введення для формування рандомного графа------------
void MainWindow::on_action_create_file_triggered()
{
    // Створення діалогового вікна
    QDialog *dialog = new QDialog(this);

    QLabel *verticesLabel = new QLabel("Введіть кількість вершин:", dialog);
    QLabel *edgesLabel = new QLabel("Введіть кількість ребер:", dialog);
    QLineEdit *verticesLineEdit = new QLineEdit(dialog);
    QLineEdit *edgesLineEdit = new QLineEdit(dialog);
    QPushButton *okButton = new QPushButton("OK", dialog);

    QVBoxLayout *layout = new QVBoxLayout(dialog);
    layout->addWidget(verticesLabel);
    layout->addWidget(verticesLineEdit);
    layout->addWidget(edgesLabel);
    layout->addWidget(edgesLineEdit);
    layout->addWidget(okButton);

    // Підключення слота до сигналу clicked кнопки "OK"
    connect(okButton, &QPushButton::clicked, this, [=]() {
        int vertices = verticesLineEdit->text().toInt();
        int edges = edgesLineEdit->text().toInt();

        // Ваша логіка обробки даних тут
        DataCreation::createRandomData(vertices, edges);
        filePath = "D:/Qt/graphtestproject/graph.txt";
        displayFileContents(filePath);
        dialog->accept(); // Закриття діалогового вікна після отримання даних
    });

    dialog->setLayout(layout);
    dialog->exec(); // Показати діалогове вікно та очікувати його закриття
}
```

Додаток 2. Файли вхідних даних

Файл input_1.txt

```
7 11
1 2
2 3
2 4
3 1
4 1
4 3
4 4
5 6
5 7
6 2
7 4
```

Файл input_2.txt

```
4 5
1 2
2 3
3 1
4 1
4 2
```

Файл input_3.txt

```
4 6
1 2
1 3
2 2
2 3
4 2
4 3
```

Файл input_4.txt

```
9 15
1 2
1 5
1 8
2 3
2 7
2 9
3 1
3 2
3 4
3 6
4 5
5 2
6 4
8 9
9 8
```

Файл graph.txt

```
100 150
4 73
4 99
5 81
6 14
6 32
6 36
7 34
7 58
8 7
8 82
9 12
9 23
10 27
10 60
```

10 69
11 65
11 88
13 35
13 56
14 96
15 45
16 29
16 50
16 51
16 60
16 85
17 3
18 89
19 37
20 7
21 30
23 89
24 31
25 52
26 49
27 49
28 67
29 11
29 24
29 82
29 97
30 24
30 52
30 70
30 100
31 88
31 89
32 70
34 50
34 85
35 59

35 97

36 1

37 51

37 53

37 61

37 70

37 71

37 79

38 20

38 39

38 69

40 39

41 25

41 63

42 54

43 25

43 54

44 8

44 27

44 28

44 68

44 82

45 71

46 20

47 61

49 35

49 74

49 99

50 86

51 100

52 49

53 21

53 32

54 21

54 65

55 84

56 37

56 45
56 83
56 89
57 46
57 48
58 2
58 76
61 3
61 96
61 97
64 8
64 39
65 19
65 61
65 64
66 38
66 94
67 31
67 43
69 12
70 26
70 30
70 53
71 13
71 90
72 8
72 76
73 43
73 93
74 10
75 50
76 10
76 62
76 72
80 66
82 11
83 11

83 50

83 91

84 2

84 15

84 40

87 70

87 74

87 97

88 38

88 96

89 27

91 89

92 26

92 67

93 9

93 56

94 53

95 62

96 35

96 46

96 69

96 70

97 64

97 93

100 73