МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

КУРСОВА РОБОТА

з дисципліни «Алгоритмізація та програмування»

на тему «визначення найкоротших відстаней від наданої вершини до інших (алгоритм Дейкстри)»

Виконав: студент 1 курсу групи КІ-ХХ
спеціальність 122 «Комп'ютерні науки»
_____
Керівник: к. е. н., доцент Просянкіна-Жарова Т. І.
Національна оцінка _____
Кількість балів: _____

Прийняли:

_____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Засвідчую, що у цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань студент
_____

Київ - 2024

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»					
Навчально-науковий інститут прикладного системного аналізу					
Кафедра	***Штучного інтелекту***				
Дисципліна	***Алгоритмізація та програмування***				
Галузь знань	***12 Інформаційні технології***				
Курс	***перший***	Група	***КІ-32***	Семестр	***другий***

ЗАВДАННЯ

на курсовий проект(роботу) студента

Ільїн Сергій Анатолійович	
(прізвище, ім’я, по батькові)	
1. Тема проекту(роботи)	визначення найкоротших відстаней від наданої вершини до інших (алгоритм Дейкстри)
2. Строк здачі студентом закінченого проекту(роботи)	***XX.05.2024 р.***
3. Вихідні дані до проекту(роботи)	
Середовище програмування Qt Creator	
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)	
1. Постановка задачі.	
2. Метод розв’язку задачі	
3. Загальна блок-схема алгоритму та опис алгоритму	
4. Опис програмного продукту.	
5. Результати роботи.	
6. Висновки.	
7. Список використаної літератури.	
Додаток А. Текст програми.	
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)	
1. Загальна блок-схема алгоритму.	
2. Ілюстрації роботи програми.	
6. Дата видачі завдання	

КАЛЕНДАРНИЙ ПЛАН

№/п	Назва етапів курсового проекту (роботи)	Строк виконання етапів роботи	Примітка
1.	Вибір теми курсової роботи. Опрацювання відповідної літератури. Оформлення листа Завдання.	05.02.2024-18.02.2024	
2.	Аналіз постановки задачі. Узгодження з керівником попереднього плану роботи.	19.02.2024-03.03.2024	
3.	Вибір та дослідження методів та структур даних, Розробка загального алгоритму	04.03.2024-17.03.2022	
4.	Перше узгодження з керівником.	18.03.2024-24.03.2024	
5.	Проектування інтерфейсу. Розробка алгоритмів окремих блоків.	25.03.2024-07.04.2024	
6.	Друге узгодження з керівником.	08.04.2024-14.04.2024	
7.	Програмна реалізація.	15.04.2024-05.05.2024	
8.	Демонстрація першого варіанту. Третє узгодження з керівником.	06.05.2024-12.05.2024	
9.	Доопрацювання програми. Заключне тестування програми.	13.05.2024-19.05.2024	
10.	Аналіз результатів. Оформлення пояснювальної записки.	20.05.2024-02.06.2024	
11.	Захист та демонстрація курсової роботи.	03.06.2024-07.06.2024	

Студент		
	(підпис)	

Керівник			
	(підпис)		(прізвище, ім’я, по батькові)

(дата)	

ЗМІСТ

ВСТУП

Питання пошуку найкоротшого шляху було актуальним майже всю історію людства. Коли розвиток світового суспільства спричинив міжнародну торгівлю, це питання стало особливо важливим [1], так як оптимальний маршрут торгового шляху як мінімум робить доставлення довільних пакунків набагато коротшим в часі, як максимум зберігає здоров'я та здоровий глузд кур'єрів. Таке питання загалом досить важливе в логістиці (логістика - мистецтво та наука управління потоками товарів, енергії, інформації та інших ресурсів від точки походження до точки споживання [13]), навігації, програмному забезпечені, навіть в туризмі.

Вирішення цієї проблеми математичним (системним) шляхом з'явилось тільки в XX ст. в вигляді різноманітних алгоритмів [3]. Один з таких, який буде розглядатись в цій роботі, називається алгоритмом Дейкстри. Алгоритм Дейкстри — алгоритм на графах, розроблений Дейкстрою (повне ім'я "Е́дсгер Ві́бе Де́йкстра") . Знаходить найкоротший шлях від однієї вершини графу до всіх інших вершин [3].

Метою курсового проєкту є реалізація цього алгоритму.

Завдання роботи – на основі здобутих навичок розробити застосунок з реалізацією алгоритму Дейкстри за допомогою відповідних вбудованих та сторонніх бібліотек для реалізації алгоритму та графічного інтерфейсу.

Основним практичним значенням даної роботи є покращення розуіння алгоритму Дейкстри. Окремі алгоритми та класи створеного додатку можна буде використовувати у майбутніх навчальних проєктах.

При виконанні роботи було використано інтегроване середовище розробки Qt Creator [9] та підключено фреймворк Qt[11] для роботи з інтерфейсом. Програма розробляється на операційній системі Windows 11. Веб-браузер, вибраний для пошуку інформації – Mozilla Firefox [17]. Для рисування ілюстрацій був використаний GIMP [18]. Для побудови блок-схем алгоритмів онлайн-застосунок draw.io. Для оформлення пояснювальної записки було використано текстовий процесор Microsoft Word 2016.

Вибір теми для курсової роботи зумовлений корисністю навичок при її виконані. Реалізація алгоритмів пошуку найкоротшого шляху та створення інтерфейсу може застосовуватись у різноманітних сферах, наприклад, в створенні власних алгоритмів, розробці певних обчислювальних програм, інше.

Пояснювальна записка курсової роботи складається з вступу, висновків та трьох розділів, у яких викладено основний текст роботи. У першому розділі було оглянуто поверхнево предметну область, обрано інструменти розробки та виконано постановку задачі дослідження. У другому розділі розглянуто побудову програми, алгоритм пошуку найкоротшого шляху (тобто Дейкстри), опис розробленого в цій курсовій роботі програмного забезпечення представлений у третьому розділі.

Основою для дослідження є інформаційні ресурси мережі Інтернет, праці вітчизняних та закордонних вчених та програмістів-практиків, чинне законодавство України, державні та закордонні стандарти в галузі інформаційних технологій.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз доступної інформації

Алгоритм Дейкстри є класичним алгоритмом для знаходження найкоротших шляхів у графах з невід'ємними вагами ребер. Його широке використання і популярність обумовлені ефективністю і простотою реалізації, що дозволяє застосовувати його у різних галузях, таких як мережевий трафік, навігація, планування маршруту тощо. Дослідження та аналіз існуючої інформації про алгоритм Дейкстри дозволяють зрозуміти його сильні та слабкі сторони, а також сфери застосування [2][3]. Алгоритм Дейкстри був запропонований нідерландським вченим Едсгером Дейкстрою у 1956 році і опублікований у 1959 році [3]. Цей алгоритм став основоположним у теорії графів та комп'ютерній науці, відкривши нові можливості для вирішення задач на графах. Спочатку алгоритм був розроблений для використання у маршрутизації і мережевих протоколах, однак його універсальність дозволила застосовувати його в інших галузях [4]. Алгоритм Дейкстри належить до категорії жадібних алгоритмів, які на кожному кроці вибирають оптимальний локальний варіант, сподіваючись, що глобальний результат також буде оптимальним. Основна ідея полягає у поступовому збільшенні множини вершин з відомими найкоротшими шляхами, поки всі вершини графа не будуть оброблені [2]. Класична реалізація алгоритму Дейкстри має часову складність O(V^2), де V — кількість вершин у графі [3]. Використання пріоритетної черги з бінарною купою дозволяє знизити складність до O((V+E)logV), де E — кількість ребер [4]. Для подальшої оптимізації використовуються фібоначчієві купи, що зменшує складність до O(VlogV+E) [4]. Аналіз різних реалізацій та структур даних показує, що вибір оптимальної реалізації залежить від специфіки задачі та розмірів графа [2][5]. Алгоритм Дейкстри активно використовується у сучасних технологіях. У мережевій маршрутизації він допомагає знайти найкоротші

шляхи між вузлами мережі, що забезпечує ефективну передачу даних [2]. У географічних інформаційних системах (ГІС) алгоритм використовується для планування маршрутів на карті, дозволяючи знаходити оптимальні шляхи між пунктами [3]. Також він застосовується в логістиці для оптимізації перевезень та мінімізації витрат [6]. Алгоритм Дейкстри має певні обмеження, які слід враховувати при його застосуванні. Основне обмеження — невід'ємність ваг ребер. Для графів з від'ємними вагами більш підходящим є алгоритм Беллмана-Форда [4]. Також алгоритм може бути неефективним для дуже великих графів з великою кількістю вершин і ребер, де пріоритетними стають інші алгоритми [5]. Сучасні дослідження зосереджуються на адаптації алгоритму Дейкстри до паралельних і розподілених обчислень, що дозволяє використовувати його для великих графів у реальному часі [5]. Розглядаються також гібридні підходи, які комбінують Дейкстру з іншими алгоритмами для підвищення ефективності. Такі дослідження відкривають нові перспективи для застосування алгоритму у високонавантажених системах [4][6].

Таким чином, аналіз доступної інформації про алгоритм Дейкстри демонструє його важливість і актуальність у сучасній комп'ютерній науці. Незважаючи на певні обмеження, алгоритм залишається одним з найбільш ефективних і широко використовуваних методів для знаходження найкоротших шляхів у графах з невід'ємними вагами ребер [2][3][4].

1.2 Обґрунтування вибору інструментів, обраних для завдання

У якості мови програмування вибрано мову C++. Важливою перевагою вибраної мови є багатофункціональна стандартна бібліотека та оптимізація коду для більшості платформ. Це одна з найпопулярніших мов для написання комп’ютерних програм і найважливіша для створення складних додатків, включаючи системні програми та ігри [7].

Мова C++ була розроблена Б'ярном Страуструпом і безпосередньо реалізована на мові С. Фактично майже вся мова С збереглася у C++ у вигляді своєрідної підмножини. Однак, C++ пропонує покращені способи вирішення багатьох задач і включає інноваційні методи, яких не було в мові С [8].

Існує ряд причин, завдяки яким мова C++ використовується для розробки додатків: Однією з основних проектних характеристик мови C++ була висока продуктивність, тому швидкість виконання функцій одна з найвищих серед таких мов [7]. Це мультипарадигмова мова програмування, що підтримує різні стилі програмування, у тому числі і об’єктно-орієнтоване програмування [7]. Оскільки мова C++ давно використовується в індустрії програмування, по ній доступно багато ресурсів, включаючи численні бібліотеки та інструменти для різних завдань [8].

Середовищем розробки обрано Qt Creator, що є потужною IDE, створеною компанією The Qt Company. Однією з причин її вибору є можливість безкоштовного використання з відкритим вихідним кодом, що пропонує ряд високорівневих функціональних можливостей, які виходять за межі базового управління кодом [9]. Важливими перевагами є інтуїтивно зрозуміле форматування коду та підсвічування помилок з пропонуванням можливих способів їх вирішення. Qt Creator підтримує останні стандарти C++, включаючи C++17 та C++20 [10].

Графічна частина додатку може бути реалізована за допомогою фреймворку Qt – кросплатформного інструментарію для розробки програмного забезпечення. Qt надає потужні засоби для створення як графічного інтерфейсу користувача, так і обробки даних [11]. Для створення графіки обрано Qt з таких причин: Qt є об’єктно-орієнтованим фреймворком, що забезпечує високу продуктивність та простоту у використанні завдяки зручному API [12]. Qt підтримує кросплатформну розробку, дозволяючи створювати додатки, що працюють на різних

операційних системах без значних змін у коді [9]. Бібліотека надає розширені можливості для роботи з графікою, включаючи двовимірний і тривимірний рендеринг, що дозволяє створювати складні візуальні ефекти [11].

1.3 Уточнена постановка задачі

Після дослідження предметної області та аналізу інформаційних джерел щодо реалізації алгоритму Дейкстри, а також врахування умов технічного завдання з приводу необхідного функціоналу для реалізації програмного забезпечення, можна сформулювати задачі, які необхідно вирішити під час проєктування:

1. Провести дослідження інформаційних джерел щодо предметної області, що відноситься до технічного завдання.
2. Розробити свою реалізацію алгоритму Дейкстри у обраному середовищі, мовою програмування C++.
3. Спроектувати графічний інтерфейс.
4. Перевірити коректність роботи програми.
5. Розробити інструкцію користувача, описати розроблений додаток та його інтерфейс.
6. Зробити висновки щодо виконання технічного завдання та шляхів удосконалення розробки.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз предметної області. Загальні відомості про застосунок.

В кожному програмному забезпеченні, яке вирішує певні поставлені задачі, які задані певним набором символів в окремо виділеній області програми, має бути певний алгоритм, який буде коректно вирішувати ці задачі, та результат якого має висвічуватись також в окремо виділеній області програми. Також має бути можливість зберігання та завантажування вхідних даних для алгоритму. З цього ми можемо зробити висновок, що наша програма буде побудована за такими принципами:

- Надійна реалізація алгоритму Декйстри.
- Можливість зберігання та завантажування вхідних даних алгоритму Дейкстри.
- Виділення окремої області під введення вхідних даних вручну в програмі.
- Виділення окремої області під виведення результату алгоритму Дейкстри матричним та графічним способом (за допомогою графа).

Програма реалізована за даними принципами, які використовуються в більшості програмного забезпечення такого роду. Для введення вхідних даних в програмі використовується віджет квадратної матриці (матриця 1 на рис. 2.2) для задання матриці суміжності графа, який задається подібно до класичної матриці суміжності. Розмір цього віджета можна змінити. Також передбачене завантаження матриці суміжності з текстового файлу (File -> Open рис. 2.3). Для виведення результату алгоритму Дейкстри використовується віджет матриці з одним рядком (матриця 2 на рис. 2.2), де в кожній комірці на індексі, який рівний номеру вершини, буде мінімальний шлях в текстовому вигляді (тобто по яким вершинам треба йти, щоб пройти мінімальну відстань до вершини) та мінімальний шлях в умовних одиницях (рис. 2.5). Також для виведення

результату буде використовуватись віджет для графічного представлення графа, тобто граф буде намальований після прорахунку алгоритму Дейкстри в вигляді вершин та ребер між ними, де вершини будуть мати два кольори: зелений та червоний. Зелений означає, що алгоритм дійшов до вершини, а червоний – не дійшов.

Провідною ланкою даної розробки є алгоритм Дейкстри, без якого неможливе знаходження найкоротшого шляху в програмі цієї курсової роботи, тому треба розібратись, як він працює. На рисунку 2.1 зображений псевдокод алгоритму Дейкстри [3].

```
1  function Dijkstra(Graph, source):
2
3      for each vertex v in Graph.Vertices:
4          dist[v] ← INFINITY
5          prev[v] ← UNDEFINED
6          add v to Q
7      dist[source] ← 0
8
9      while Q is not empty:
10         u ← vertex in Q with minimum dist[u]
11         remove u from Q
12
13         for each neighbor v of u still in Q:
14             alt ← dist[u] + Graph.Edges(u, v)
15             if alt < dist[v]:
16                 dist[v] ← alt
17                 prev[v] ← u
18
19     return dist[], prev[]
```

Рисунок 2.1 – псевдокод алгоритму Дейкстри.

Як ми бачимо з рисунку 2.1 вхідні дані алгоритму – це початкова вершина, з якої алгоритм буде знаходити відстань до кожної точки графа, та сам граф, заданий певним чином. На початковому етапі алгоритм створює масив dist, який відповідає за загальну дистанцію від початкової вершини до певної вершини, де по замовчуванню з початку для початкової вершини ставиться дистанція 0, а для решти – нескінченність, масив prev, який відповідає за збереження пройденого шляху по вершинах в текстовій формі від початкової вершини до певної вершини, та множину Q, яка містить в собі всі вершини графа, але без ребер. В

основній частині алгоритм входить в цикл умови, який ітерує допоки множина Q не порожня. На початку ітерації цикла умови створюється локальна змінна u, в яку записується вершина з мінімальною дистанцією від початкової вершини, яка потім видаляється з множини Q. Далі в умовному циклу алгоритм входить в цикл, який ітерує по всім вершинам v, які ще знаходяться в множині Q, які є сусідами вершині u. Далі в цьому внутрішньому циклі знаходиться альтернативна відстань до вершини v, яка дорівнює сумі дистанції від початкової вершини до вершини u та дистанції від вершини u до v. Якщо ця альтернативна дистанція менша за попередню, то замість попередньої дистанції в масиві dist для вершини v записується тільки що знайдена альтернативна дистанція. Також вершина u записується до шляху в масив prev на індексі v. Після чого цикл продовжує роботу. Після закінчення алгоритму функція повертає масив з мінімальними відстанями від початкової вершини до решти вершин та масив з шляхами від початкової вершини до решти вершин в письмовій формі.

У коді застосунку використано елементи ООП, такі як наслідування, інкапсуляція та поліморфізм. Були створені окремі класи для реалізації інтерфейсу та групування функціоналу даного програмного забезпечення. Ці класи представлені в таблиці 2.1.

Таблиця 2.1 – Використані у застосунку класи

Назва класу	Призначення
window	Клас головного вікна, в якому зосереджуються всі віджети.
GraphWidget	Клас віджета графічного представлення графу.
MatrixModel1	Модель для зберігання вхідних даних в вигляді матриці суміжності .
MatrixModel2	Модель для виведення результату алгоритма Дейкстри.

2.2 Проектування графічної та звукової частини

Тип інтерфейсу – графічні вікна Windows. Перед початком розробки інтерфейсу, був створений прототип головного вікна, що зображений на рисунку 2.2, та прототип меню, що зображений на рисунку 2.3. На рисунку головного вікна (рис. 2.2) схематично зображені віджети з їхніми назвами. На рисунку 2.3 схематично зображена тільки область меню з всіма її підменю.

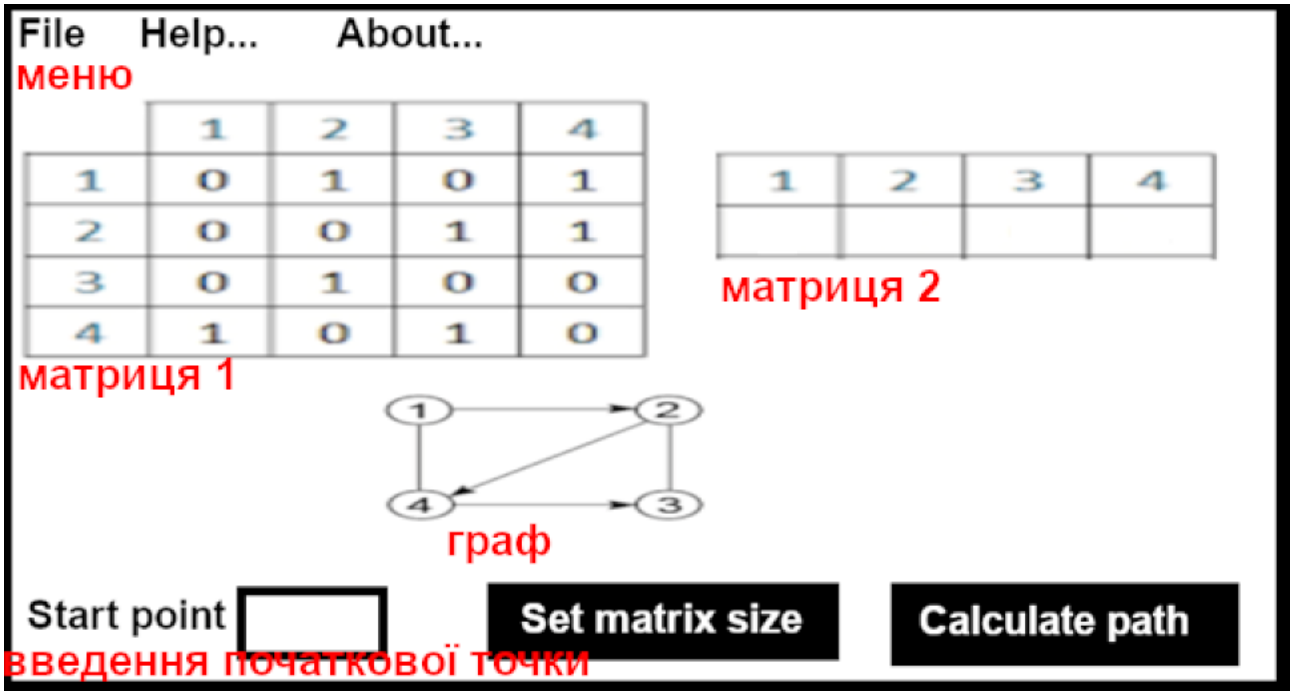

Рисунок 2.2 – Прототип вікна головного меню

Додаючи до інформації про інтерфейс застосунку з попереднього пункту, користувач ще має задати початкову точку в відповідному текстовому полі поряд з текстом “Start point” після чого, щоб прорахувати маршрути до решти точок, натиснути кнопку “Calculate path”.

На рисунку 2.3 зображено меню розглядаємої програми, де в нас є 3 головних кнопок: About (відповідає за появу малого діалогового вікна, в якому буде просто малий текст, який буде в собі містити коротеньку інформацію про программу), Help (відповідає за відкриття мануалу цієї програми, який

зберігається в вигляді word документу в кореневій папкі програми), File (відкриває підменю з ще 4 кнопками)

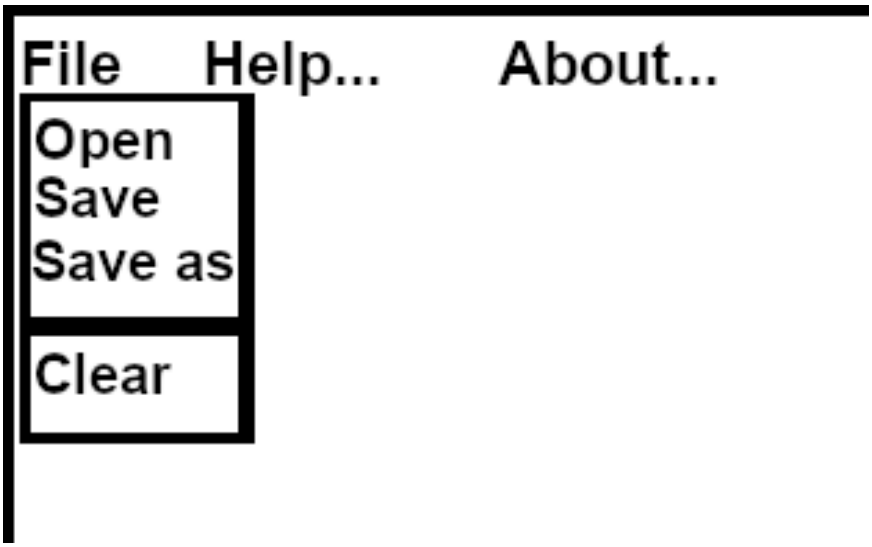

Рисунок 2.3 – прототип меню.

В підменю кнопки File знаходиться ще 4 кнопки: Open (відповідає за відкриття файлу, в якій має знаходиться матриця суміжності), Save (відповідає за зберігання матриці суміжності в вже відкритий файл), Save as (відповідає за зберігання матриці суміжності в окремо створений файл), Clear (відповідає за повне очищення матриці суміжності та графічного представлення графу). Зверніть увагу на те, що кнопка Clear відокремлена лінією, щоб показати користувачеві, що ця кнопка “небезпечна” і її не треба просто так натискати.

Натискання будь яких кнопок буде запускати відповідні “механізми”, в кінці яких буде звучати звук повідомлення windows, яке є в цій операційній системі по замовчуванню.

Кінцева реалізація інтерфейсу представлена на рисунках 2.4-2.6.

В кінцевій реалізації можна побачити незначні зміни в інтерфейсі відносно прототипу: змінені деяки назви кнопок, додане текстове поле внизу програми для показу стану програми.

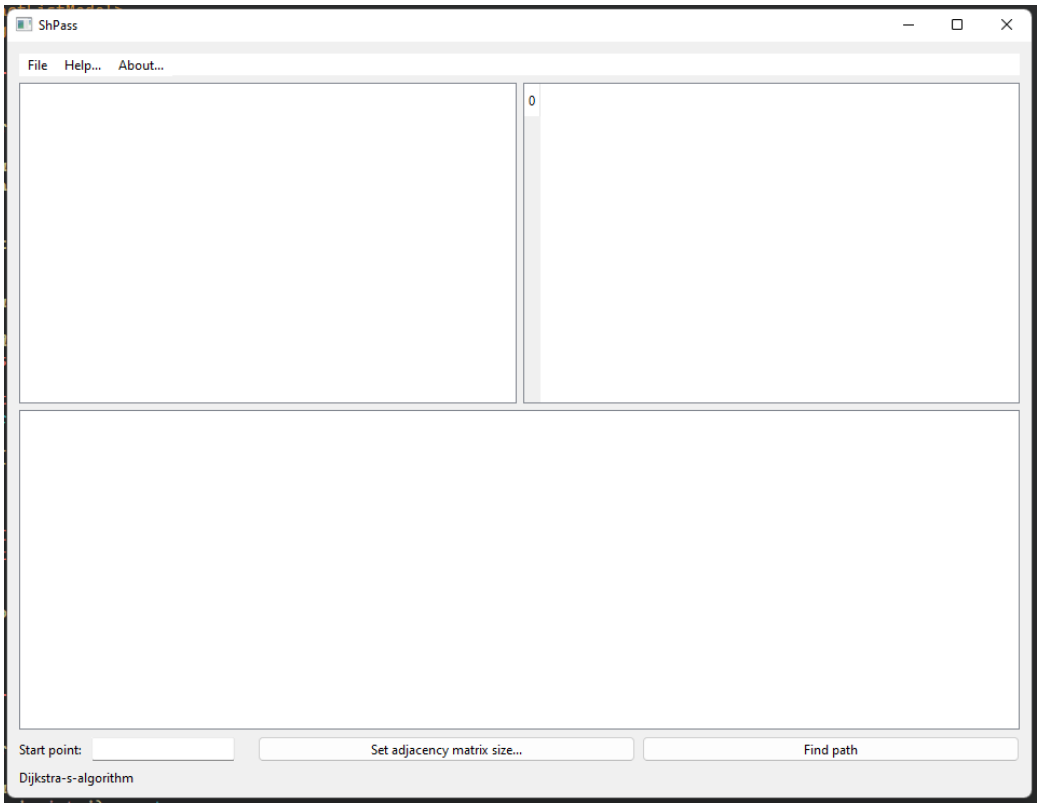

Рисунок 2.4 – кінцева реалізація інтерфейсу без відкритого файлу.

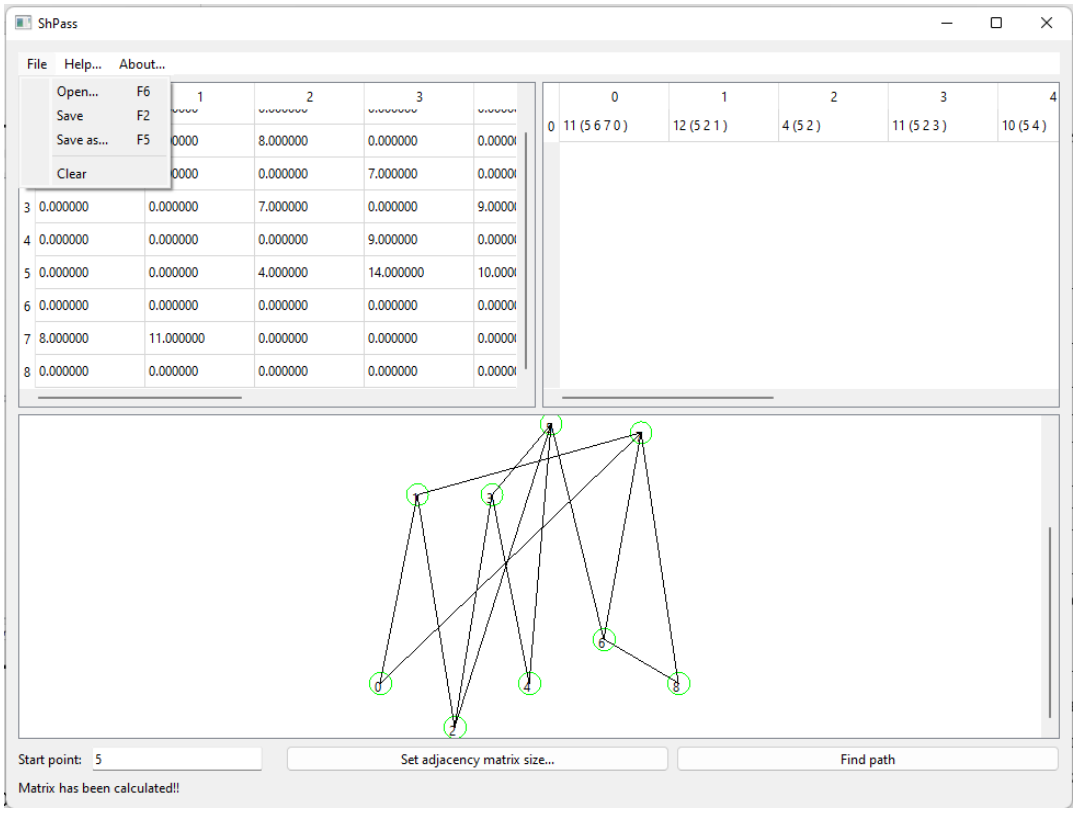

Рисунок 2.5 – кінцева реалізація інтерфейсу з відкритим файлом та прорахованим шляхом від початкової вершини 5.

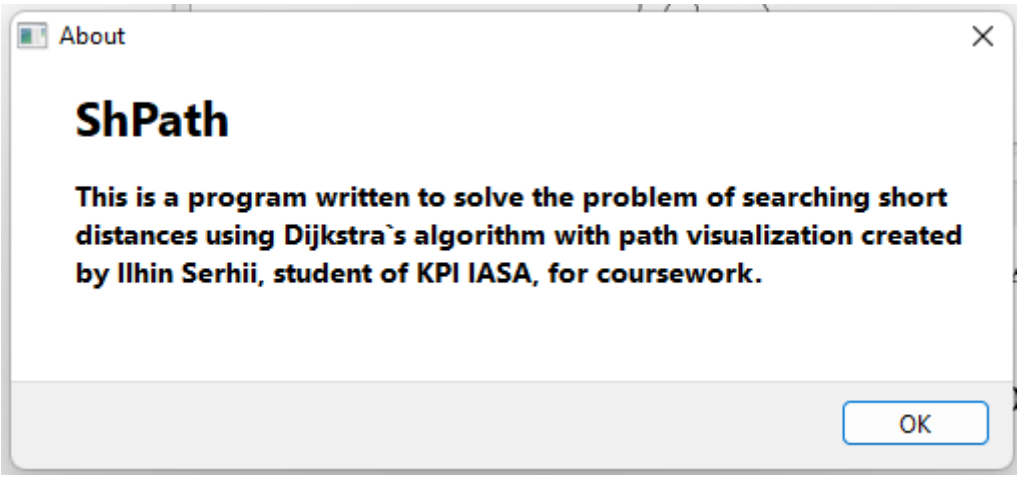

Рисунок 2.6 – Діалогове вікно About.

РОЗДІЛ 3
ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

3.1 Опис розробленого програмного забезпечення

При виконанні завдання курсової роботи був розроблений додаток за прототипом Windows-вікна. Враховуючи, що метою розробки було створення застосунку з графічною частиною, основна увага приділена реалізації класів та методів, які забезпечують роботу графічного інтерфейсу.

Алгоритм дій користувача під час використання застосунку представлений на рисунку 3.1.

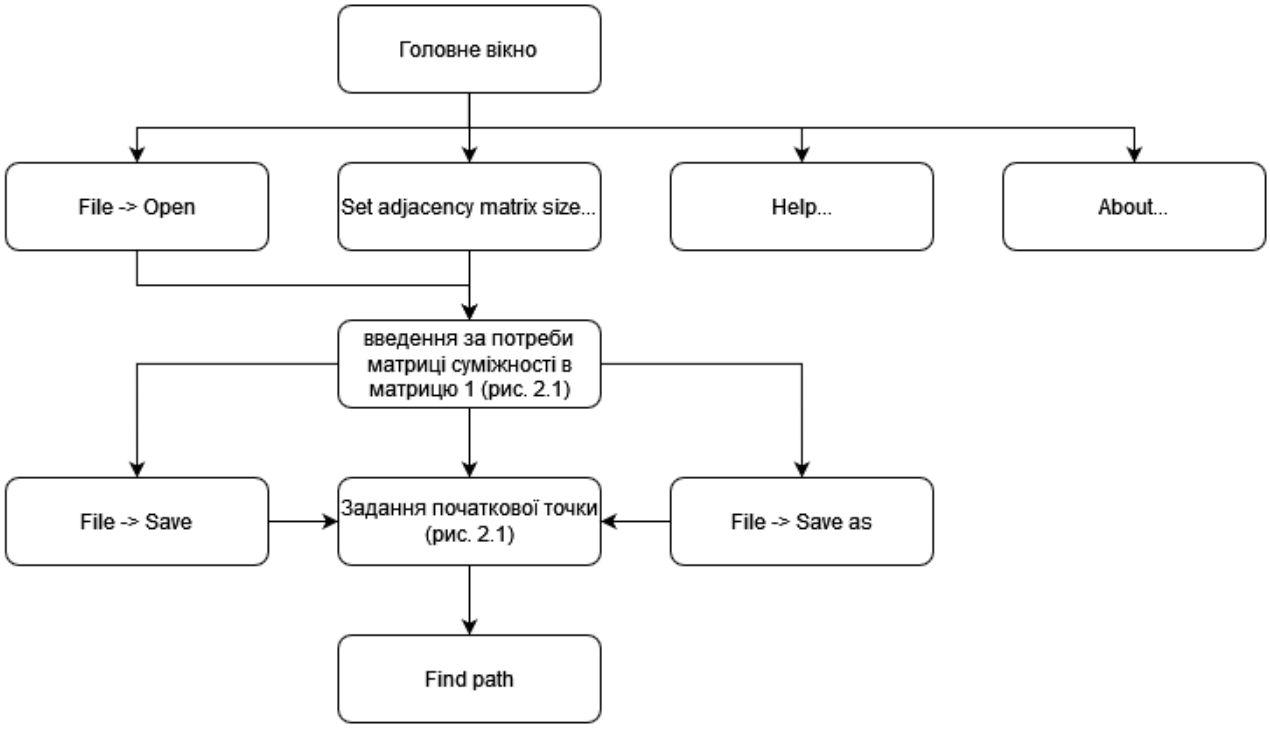

Рисунок 3.1 – Структура меню програми

З рисунку 3.1 видно, що користувач має два шляхи введення вхідних даних, про які було детальніше написано в пункті 2.1, два шляхи збереження вхідних даний: збереження матриці суміжності в вже відкритий файл та збереження матриці суміжності в тільки що створений файл користувачем.

Загалом, з рисунку 3.1 видно, що інтерфейс несильно складно спроектований. Мала кількість розгалуджень в алгоритмі використання програми забезпечує інтуїтивність для користувача, який знає англійську мову на базовому рівні та який розуміє, що таке графи та матриця суміжності. Тобто програма була націлена саме на такий тип користувачів.

Зазначимо, що після розрахунку мінімальних шляхів користувач може продовжити інтеракцію з програмою, а саме ще раз пройти алгоритм, зображений на рисунку 3.1, тим самим розрахувавши найкоротші відстані для нового графу, або можна пересунути вершини графа в його графічному представленні, щоб для користувача це ж графічне представлення було більш зрозумілим.

Лістинг (код) програми мовою C++ наведений у додатку Б.

3.2 Опис головних класів, методів, полів програми

У даній програмі всі функції використовуються як методи відповідних класів.

Методи завантаження та зберігання матриці суміжності з txt-файлу створені для того, щоб користувач міг власноруч задати вхідну матрицю у текстовому файлі. В свою чергу в даній програмі матриця суміжності зберігається в дуже легкому для розуміння форматі: щоб утворити нові колонки треба писати числа через пробіл, щоб утворити нові рядки треба просто перейти на наступний рядок. Це все робиться в текстовому файлі з розширенням “.txt”. З формату зберігання матриці суміжності зрозуміло, який алгоритм методу зберігання: зовнішній цикл, який ітерує по рядках, внутріщній цикл, який ітерує по стовпчиках, всередині внутрішнього циклу в відкритий файл записуються числа через пробіл, після внутрішнього циклу, тобто в кінці зовнішнього циклу, запишеться перехід на наступний рядок. Алгоритм відкриття файлу майже аналогічний, тільки також додається перевірка на коректність даних в файлі.

Назви цих методів: window::saveAdjacencyMatrix та window::loadAdjacencyMatrix.

Програма використовує метод моделі-представлення [14]. Якщо коротко, то це такий метод, коли існує два об'єкти: об'єкт представлення, який відповідає за висвітлення та приймання даних, та об'єкт моделі, який відповідає за зберігання даних. Цей метод використовується для представлення в стилі віджетів списків, таблиць, ієрархічних дерев.

В коді програмі є класи MatrixModel1 та MatrixModel2, їхні призначення можна подивитись в таблиці 2.1. Вони реалізують матриці, а якщо конкретніше, то таблиці. Ці два класи наслідуються від класа QAbstractTableModel [15], що означає, що MatrixModel1 та MatrixModel2 реалізують модель віджета представлення таблиці. Представлення таблиці використовується вбудоване в конструкторі класа window. В об’єкти цих представлень передаються MatrixModel1 та MatrixModel2, щоб прив’язати дані з моделі до представлення. Всі класи, які наслідуються від абстрактних класів, мають реалізувати всі абстрактні методи базового класу [8], тому MatrixModel1 та MatrixModel2 реалізовують ті методи, які представленні в таблиці 3.1 (Для чого треба ці методи можна подивитись в таблицях 3.101, 3.102, 3.111 та 3.112. Як саме треба реалізовувати такі методи і за що саме вони відповідають можна почитати в [15]).

Назва абстрактного методу	Параметри абстрактного методу	Значення повернення абстрактного методу
data	const QModelIndex& index, int role	QVariant (тип даних, який в собі може тримати різні типи даних)

headerData	int section, Qt::Orientation orientation, int role = Qt::DisplayRole	Bool (логічний тип даних)
flags	const QModelIndex&	Qt::ItemFlags (Тип з обмеженими значеннями)
setData	const QModelIndex& index, const QVariant& data, int role	Bool (логічний тип даних)
columnCount	const QModelIndex& index = QModelIndex()	Int (ціле число)
rowCount	const QModelIndex& index = QModelIndex()	Int (ціле число)

Таблиця 3.1 – поверхневий огляд абстрактних методів QAbstractTableModel, які треба перевизначити при його наслідуванні.

Так як MatrixModel1 зберігає матрицю суміжності, то в його приватному полі є змінна, названна VectoredMat, типу двовимірного динамічного масиву, який в собі зберігає числа з плаваючою крапкою (QVector<QVector<double>> VectoredMat). MatrixModel2 зберігає в собі в вигляді тексту результат алгоритму Дейкстри, тому в його приватному полі є змінна, названа mat, типу динамічного масиву, який зберігає в собі рядки (QVector<QString> v).

Для графічного представлення графа використовується клас GraphWidget, який наслідується від класа QGraphicsView [16], який в свою чергу слугує полотном для графіки, яка непередбачена звичайними віджетами. Такою графікою є графи. В GraphWidget є два основні методи: метод задання матриці суміжності та метод, який відстежує інтеракії представлення з курсором. В метод задання матриці суміжності, в параметри якої треба передавати саму матрицю суміжності, яку ми хочемо графічно представити, відбувається в такий спосі: спочатку матриця суміжності дублюється в локальну матрицю суміжності класу GraphWidget, яка об’явлена в приватному полі цього класу, після чого створюються вершини, а потім ребра між ними. Підмічу той факт,

що випадкове розташування вершин досить незручне, так як підібрати такі параметри для гернерації чисел, щоб координати всіх вершин графа були комфортні для ока, дуже тяжко, тому для генерації x вершини графа я використав формулу $x = i * 34$, де i – це лічільник цикла, який ітерує по всім вершинам. Ця формула робить так, щоб кожні сусідні вершини по x відрізнялись в розташуванні на 34 пікселі. А для генерації висоти вершини я використав спеціальну формулу, яку підібрав емпіричним методом: $y = 100\cos(i\pi) + 40\sin\left(\frac{i\pi}{4}\right)$, де i – це лічільник цикла, який ітерує по всім вершинам. На мій погляд таке розташування доволі комфортне для ока. Результат такого розташування представлений на рисунку 2.5.

Метод в класі GraphWidget з назвою mouseMoveEvent, який відстежує інтеракії представлення з курсором, дає можливість користувачеві так виставити вершини на графічному представлені, як він хоче. Підмічу те, що такої б можливості не було б, якби в методі задання матриці суміжності під назвою setMat в класі GraphWidget ми не задавали спеціальний прапорець (QGraphicsItem::ItemIsMovable [16]) для створеної вершини, то ми б не могли реалізувати mouseMoveEvent належним чином.

Найголовнішим методом цієї програми є конструктор класа window, тому що саме в ньому будується весь інтерфейс програми та задається весь функціонал програми.

Опис полів та методів класів можна подивитись в таблицях 3.11 – 3.112.

Отже, варто сказати, що при використанні такого сильного інструменту як фреймворк Qt [11] код значно спрощується як в його розумінні, так і в його об'ємах. Також його перевагою є те, що він підтримує кросплатформу, тому написаний мною інтерфейс та всю програму загалом можна перенести на будь-яку сучасну операційну систему. Але якби моя прогрма для курсової роботи мала б використовувати якісь специфічні для операційної системи або для архітектури процессора операції, то треба було б використовувати бібліотеки нижчого рівня.

Таблиця 3.11 – Поля класу window (змінні).

Ідентифікатор	Тип	Модифікатор доступу	Призначення
model1	MatrixModel1*	public	Таблиця для введення матриці суміжності
Model2	MatrixModel2*		Таблиця для виведення результату алгоритму Дейкстри
graph1	GraphWidget*		Графічне представлення для рисування графу
bCalculate	QPushButton*		Кнопка для запуску алгоритму Дейкстри
startPointA	QLineEdit*		Текстове поле для введення початкової вершини, з якої буде йти пошук інших вершин
resLabel	QLabel*		Текстове відображення, яке відображає стан програми
bSetNewMatrixSize	QPushButton*		Кнопка, яка визиває діалогове вікно задання нового розміру матриці model1

Таблиця 3.12 – Поля класу window (змінні).

Ідентифікатор	Тип	Модифікатор доступу	Призначення
dioSetNewMatSize	QDialog*	public	Діалогове вікно вибору нового розміру матриці model1
bSetV1	QLineEdit*		Текстове поле діалогового вікна dioSetNewMatSize
enterMatSize	QPushButton*		Кнопка, після натискання якої задається новий розмір матриці model1
menu	QMenuBar*		Меню головного вікна
ifOpened	Bool		Чи відкритий файл для зберігання матриці суміжності в застосунку
filePath	QString		Шлях до відкритого файлу

Таблиця 3.2 – Інкапсульовані однакові поля класів MatrixModel1 та MatrixModel2.

Ідентифікатор	Тип	Призначення
mat	QHash<QModelIndex, QString>	Зберігає матрицю суміжності в вигялді ключ значення, де ключ має тим індексу моделі, а значення – це рядок.
rows	int	Рядки матриці суміжності
cols	int	Колонки матриці суміжності

Таблиця 3.3 – Інкапсульовані Поля класу GraphWidget.

Ідентифікатор	Тип	Призначення
adjacencyMatrix	QVector<QVector<double>>	Зберігання матриці суміжності графу
scene	QGraphicsScene*	Об’єкт сцени, який в собі зберігає графічні об’єкти, такі як вершини та ребра графу
nodes	QVector<QGraphicsEllipseItem*>	Вершини графу задані графічно
nodeLabels	QVector<QGraphicsSimpleTextItem*>	Номери (ярлики) вершин графів
edges	QMap<QPair<int, int>, QGraphicsLineItem*>	Ребра графу задані графічно

Таблиця 3.4 – Методи в класі window.

Індекс	Прототип методів в класі
1	inline void setMatSize(const int &size);
2	void keyPressEvent(QKeyEvent *event) override;
3	void slotCalculateClicked();
4	void slotSetNewMatrixSizeFromDialog();
5	void slotDioSetNewMatSizeShow();
6	void slotMenuTriggered(QAction*);

Таблиця 3.5 – Методи в класі GraphWidget.

Індекс	Прототип методів в класі
1	void setMat(QVector<QVector<double>> m);
2	void pointFindedNodes(QVector<bool> v);
3	void pointAllBlack();
4	void clear();
5	void mouseMoveEvent(QMouseEvent *event) override;

Таблиця 3.6 – Методи в класі MatrixModel1.

Індекс	Прототип методів в класі
1	QVariant data (const QModelIndex& index, int role) const override;
2	QVariant data (int i, int j) const;
3	bool setData (const QModelIndex& index, const QVariant& data, int role) override;
4	void setData (int i, int j, const QVariant& data);
5	void clearData();
6	int rowCount (const QModelIndex& index = QModelIndex()) const override;
7	int columnCount (const QModelIndex& parent = QModelIndex()) const override;
8	QVariant headerData (int section, Qt::Orientation orientation, int role = Qt::DisplayRole) const override;
9	Qt::ItemFlags flags (const QModelIndex&) const override;
10	QVector<QVector<double>> GetVectoredMat();
11	void setMat(const QVector<QVector<double>>& mat);
12	void setSize(int rowCount);
13	void clear();

Таблиця 3.7 – Методи в класі MatrixModel2.

Індекс	Прототип методів в класі
1	QVariant data (const QModelIndex& index, int role) const override;
2	QVariant data (int i, int j) const;
3	bool setData (const QModelIndex& index, const QVariant& data, int role) override;
4	void setData (int i, int j, const QVariant& data);
5	void clearData();
6	int rowCount (const QModelIndex& index = QModelIndex()) const override;
7	int columnCount (const QModelIndex& parent = QModelIndex()) const override;
8	QVariant headerData (int section, Qt::Orientation orientation, int role = Qt::DisplayRole) const override;
9	Qt::ItemFlags flags (const QModelIndex&) const override;
10	QVector<QString> GetVectoredMat();
11	void setRes(int Vertice, int pathLenght, QString path);
12	void setColumnCount (int);
13	void clear();

Таблиця 3.8 – Опис методів класу window (індекси див. табл. 3.4).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
1	Константне посилання на ціле число, size, задаваємий розмір матриці	Задає новий розмір матриці
2	Посилання на об’єкт події натискання клавіші на клавіатурі, event, об’єкт події натискання клавіші на клавіатурі	Робить можливим роботу гарячих клавіш
3	Параметри відсутні	При натисканні кнопки bCalculate запускає алгоритм Дейкстри
4	Параметри відсутні	При натисканні кнопки enterMatSize задає новий розмір матриці
5	Параметри відсутні	При натисканні кнопки bSetNewMatrixSize запускає діалогове вікно зчитування нового розміру матриці
6	Посилання на об’єкт події меню, a, об’єкт події меню	Запускає певні команди при натисканні меню

Таблиця 3.9 – Опис методів класу GraphWidget (індекси див. табл. 3.5).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
1	Двовимірний динамічний масив чисел з плавуючою точкою, m, передаваєма матриця суміжності	Графічне відображення графа, задаваємого передаваємим в метод параметром матрицею суміжності
2	Динамічний масив логічних змінних, v, зберігає в собі інформацію які вершини знайшлися	Помічає заленим кольором знайдені вершини, а червоним кольором – незнайдені вершини
3	Параметри відсутні	Помічає всі вершини чорним кольором
4	Параметри відсутні	Очищує весь нарисований граф
5	Об’єкт події руху курсора по графічному представленню	Робить адекватне переміщення країв ребер при переміщенні вершин

Таблиця 3.101 – Опис методів класу MatrixModel1 (індекси див. табл. 3.6).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
1	Константне посилання на модельний індекс, index, індекс комірки; Ціле число, role, задаваєма роль комірки	Задання комірці спеціальної ролі (це може бути як комірка з доступом зміни даних в ній, так і з такою заблокованою ролю)
2	Ціле число, i, індекс рядка комірки; Ціле число, j, індекс колонки комірки	Повертає значення в заданій комірці
3	Константне посилання на модельний індекс, index, індекс комірки; константне посилання на універсальний тип, data, дані, які користувач тільки що ввів ; Ціле число, role, задаваєма роль комірки	Перевіряє та записує дані, які вводить користувач в комірки
4	Ціле число, i, індекс рядка комірки; Ціле число, j, індекс колонки комірки; константне посилання на універсальний тип, data, дані, які треба передати в зазначену індексами комірку	Записує текстово дані в комірку
5	Параметри відсутні	Очищує матрицю від чисел
6	Константне посилання на модельний індекс, index, індекс комірки	Повертає кількість рядків таблиці
7	Константне посилання на модельний індекс, index, індекс комірки	Повертає кількість стовпців таблиці

Таблиця 3.102 – Опис методів класу MatrixModel1 (індекси див. табл. 3.6).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
8	Ціле число, section, комірка індексу рядка або колонки в боковому рядку або стовпчику; Ціле число, orientation, визначає яке положення бокової рядка або колонки	Ставить бокові індекси
9	Константне посилання на модельний індекс, index, індекс комірки	Повертає прапорець передаваємої комірки
10	Параметри відсутні	Повертає двовимірний динамічний масив, який зберігає матрицю суміжності
11	Константне посилання по двовимірний динамічний масив чисел з плавуючою точкою, mat, передаваєма матриця суміжності	Задає нову матрицю в таблицю
12	Ціле число, rowCount, новий розмір матриці, яка є квадратною	Задає новий розмір матриці
13	Параметри відсутні	Очищує матрицю від даних

Таблиця 3.111 – Опис методів класу MatrixModel2 (індекси див. табл. 3.7).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
1	Константне посилання на модельний індекс, index, індекс комірки; Ціле число, role, задаваєма роль комірки	Задання комірці спеціальної ролі (це може бути як комірка з доступом зміни даних в ній, так і з такою заблокованою ролю)
2	Ціле число, i, індекс рядка комірки; Ціле число, j, індекс колонки комірки	Повертає значення в заданій комірці
3	Константне посилання на модельний індекс, index, індекс комірки; константне посилання на універсальний тип, data, дані, які користувач тільки що ввів ; Ціле число, role, задаваєма роль комірки	Перевіряє та записує дані, які вводить користувач в комірки
4	Ціле число, i, індекс рядка комірки; Ціле число, j, індекс колонки комірки; константне посилання на універсальний тип, data, дані, які треба передати в зазначену індексами комірку	Записує текстово дані в комірку
5	Параметри відсутні	Очищує матрицю від чисел
6	Константне посилання на модельний індекс, index, індекс комірки	Повертає кількість рядків таблиці
7	Константне посилання на модельний індекс, index, індекс комірки	Повертає кількість стовпців таблиці

Таблиця 3.112 – Опис методів класу MatrixModel1 (індекси див. табл. 3.6).

індекс	Тип, ідентифікатор, та семантика параметрів	Призначення функції
8	Ціле число, section, комірка індексу рядка або колонки в боковому рядку або стовпчику; Ціле число, orientation, визначає яке положення бокової рядка або колонки	Ставить бокові індекси
9	Константне посилання на модельний індекс, index, індекс комірки	Повертає прапорець передаваємої комірки
10	Параметри відсутні	Повертає двовимірний динамічний масив, який зберігає матрицю суміжності
11	Ціле число, Verticie, індекс вершини; Ціле число, pathLenght, довжина знайденого шляху в умовних одиницях; Рядок, path, шлях по вершинах в вигляді рядка	Додає інформацію про знайдений алгоритмом Дейкстри шлях для заданої вершини
12	Ціле число, colCount, новий розмір матриці, яка є рядковою	Задає новий розмір матриці
13	Параметри відсутні	Очищує матрицю від даних

3.3. Опис інтерфейсу та інструкція користувача

Повний інтерфейс програми з індексацією віджетів для того, щоб посилатись на ці віджети в тексті, представлений на рисунку 3.2. Повне меню програми представлене на рисунку 3.3.

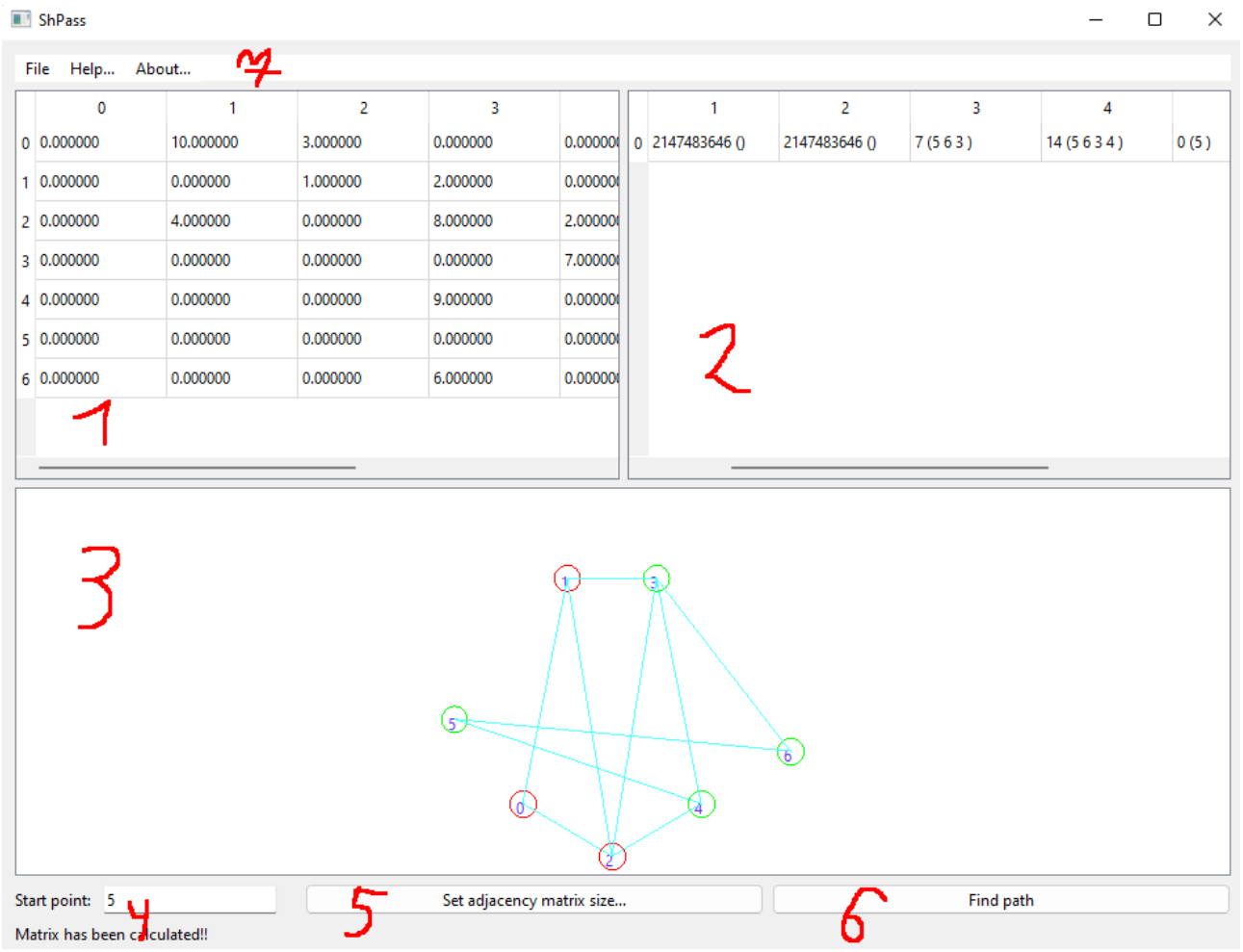

Рисунок 3.2 - Повний інтерфейс програми з індексацією віджетів.

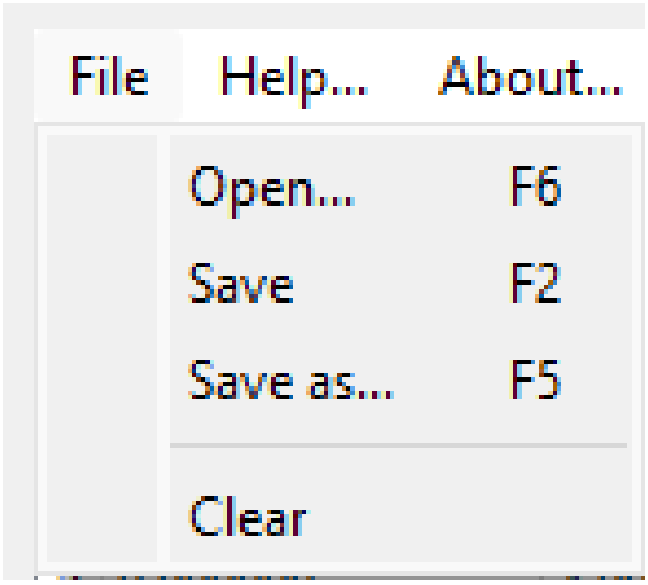

Рисунок 3.3 - повне меню програми.

Для введення вхідних даних в програмі використовується віджет квадратної матриці (1 на рис. 3.2) для задання матриці суміжності графа, який задається подібно до класичної матриці суміжності. Розмір матриці можна змінити кнопкою 5 на рис. 3.2. Також передбачене завантаження матриці суміжності з текстового файлу (File в меню на 7 індексі на рис. 3.3 та Open… на рис. 3.3). Для виведення результату алгоритму Дейкстри використовується віджет матриці з одним рядком (2 на рис. 3.2), де в кожній комірці на індексі, який рівний номеру вершини, буде мінімальний шлях в текстовому вигляді (тобто по яким вершинам треба йти, щоб пройти мінімальну відстань до вершини) та мінімальний шлях в умовних одиницях. Якщо в умовних одиницях задано число 2147483646 та в дужках поряд з ним немає вершин, то шляху до цієї вершини не знайдено. Для підрахунку коротшого шляху після введення або завантаження матриці спочатку треба ввести початкову вершину в 4 на рис. 3.2, після чого натиснути кнопку 6 на рис. 3.2. Також для виведення результату буде використовуватись віджет для графічного представлення графа, тобто граф буде намальований після прорахунку алгоритму Дейкстри в вигляді вершин та ребер між ними, де вершини будуть мати два кольори: зелений та червоний. Зелений означає, що алгоритм дійшов до вершини, а червоний – не дійшов (3 на рис. 3.2).

Індекси комірок матриці суміжності на 1 рис. 3.2 стоять зліва та зверху. Верхні індекси комірок матриці результату на 2 рис 3.2 означають те, для якої вершини представлений розрахований шлях та його відстань від заданої початкової точки в комірці цього індексу.

Кнопки Help… та About… викликають вікно допомоги та вікно з інформацією про програму відповідно.

Алгоритм роботи з програмою запропонований розробником представлений на рисунку 3.4 в вигляді блок-схеми.

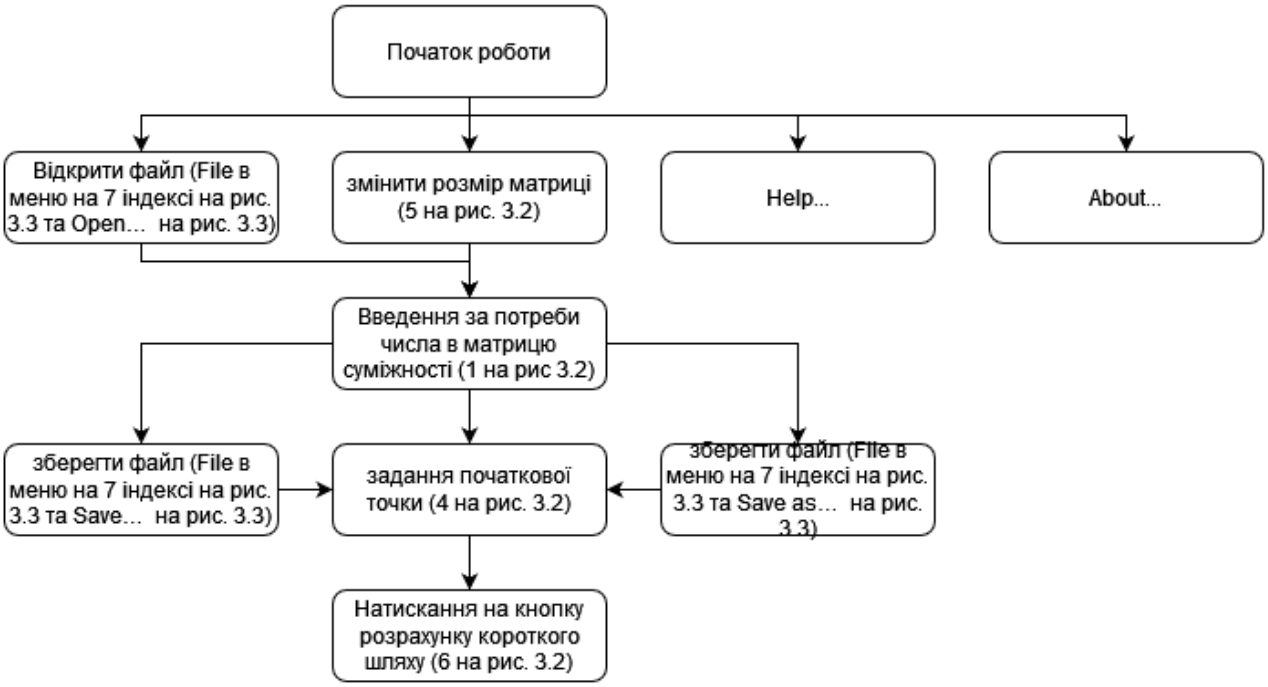

Рисунок 3.4 – запропонований розробником алгоритм роботи з програмою.

3.4 Результати роботи програмного продукту

В ході перевірки працездатності програми було досліджено, що додаток працює повністю коректно. Виявлені помилки графічного інтерфейсу та логіки алгоритму Дейкстри були виправлені на етапі розробки та тестування. Алгоритм правильно рахуя шлях та відстані, не робить зависання програми. Враховано, що при неправильному введені користувача даних в матрицю суміжності чи початкову вершину програма видає помилку де саме ми помилились.

Перевірка програми показала, що додаток не видає помилок та працює коректно.

Найголовнішою перевагою застосунку, розробленого у ході виконання курсової роботи, є простий неперевантажений інтерфейс. Також одна з переваг програми – це вибір користувача де саме йому зручніше вводити вхідні файли: в окремому текстовому файлі чи в самому додатку.

Найважливішим недоліком можна назвати повторюваність деякого коду, який не використовується в програмі. Його можна помітити в класі

MatrixModel2. Також є проблема в виведенні нумерації вершин, іноді ребра повністю закривають номер вершини. До цього ж можу додати, що розташування вершин при кожному прорахунку шляху буде однаковим і його не можна задати так, щоб було якесь конкретне розташування, а не розташування прораховане по формулі.

Отже, додаток, розроблений у ході виконання курсової роботи працює повністю коректно, має зручний інтерфейс та хорошу продуктивність. Для покращення застосунку варто звернути увагу на недоліки, які були зазначені вище.

ВИСНОВКИ

В ході виконання курсової роботи було створено програму, яка обчислює найкоротші відстані від однієї точки до всіх інших за допомогою алгоритму Дейкстри. Особливістю цього розроблення є наявність графічного представлення графа.

Пояснювальна записка до курсової роботи складається з трьох основних розділів, що охоплюють актуальність проекту, методи розв'язання задачі, детальний опис алгоритму та структури програми.

У першому розділі проведено огляд інформаційних джерел, обґрунтовано вибір мови програмування та середовища розробки, а також уточнено постановку задачі.

У другому розділі здійснено аналіз алгоритму Дейкстри, описано інтерфейс програми та загальний принцип його роботи. Особлива увага приділена прототипу інтерфейсу, який ілюструє концепцію застосунку.

У третьому розділі детально розглянуто структуру програми, описано взаємодію користувача з додатком, а також наведено опис класів та методів. У результаті було підтверджено виконання всіх вимог технічного завдання.

У майбутньому розроблений застосунок можна вдосконалити, додавши нові алгоритми пошуку шляхів, покращивши графічне відображення графа та забезпечивши можливість взаємодії з мапою світу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Історія торгівлі. URL:https://en.wikipedia.org/wiki/Timeline_of_international_trade (дата звернення: 01.03.2024).
2. GeeksforGeeks - Dijkstra's shortest path algorithm. URL:https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/ (дата звернення: 29.05.2024).
3. Wikipedia - Dijkstra's algorithm. URL:https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (дата звернення: 29.05.2024).
4. Brilliant - Dijkstra’s Algorithm. URL:https://brilliant.org/wiki/dijkstras-short-path-finder/ (дата звернення: 29.05.2024).
5. Towards Data Science - Understanding Dijkstra’s Algorithm. URL: https://towardsdatascience.com/shortest-paths-and-dijkstras-algorithm-68c9ec30eff0 (дата звернення: 29.05.2024).
6. Programiz - Dijkstra's Algorithm. URL: https://www.programiz.com/dsa/dijkstra-algorithm (дата звернення: 29.05.2024).
7. GeeksforGeeks - C++ Overview. URL: https://www.geeksforgeeks.org/c-plus-plus/ (дата звернення: 29.05.2024).
8. Wikipedia - C++. URL: https://en.wikipedia.org/wiki/C%2B%2B (дата звернення: 29.05.2024).
9. Qt Creator - Official Website. URL: https://www.qt.io/product/development-tools (дата звернення: 29.05.2024).
10. Qt Documentation - Getting Started with Qt Creator. URL: https://doc.qt.io/ (дата звернення: 29.05.2024).

11. Qt - Official Website. URL: https://www.qt.io/ (дата звернення: 29.05.2024).
12. Wikipedia - Qt (software). URL: https://en.wikipedia.org/wiki/Qt_(software) (дата звернення: 29.05.2024).
13. Logistics. URL: https://en.wikipedia.org/wiki/Logistics (дата звернення: 29.05.2024).
14. Model/View Programming. URL: https://doc.qt.io/qt-6/model-view-programming.html (дата звернення 01.06.2024).
15. QAbstractTableModel Class. URL: https://doc.qt.io/qt-6/qabstracttablemodel.html (дата звернення 01.06.2024).
16. QGraphicsView Class. URL: https://doc.qt.io/qt-6/qgraphicsview.html (дата звернення 01.06.2024).
17. Firefox. URL: https://en.wikipedia.org/wiki/Firefox (дата звернення 02.06.2024).
18. GIMP. URL: https://en.wikipedia.org/wiki/GIMP (дата звернення 02.06.2024).

ДОДАТКИ

Додаток А

Лістинг програми

```
#include <QtWidgets>
#include <fstream>
#include <cmath>

Class MatrixModel1;
Class MatrixModel2;
Class GraphWidget;

class window : public QWidget {
   Q_OBJECT
private:
   MatrixModel1* model1;
   MatrixModel2* model2;
   GraphWidget* graph1;
   //GraphWidget* graph2;
   QPushButton* bCalculate;
   QLineEdit* startPointA;
   QLabel* resLabel;
   QPushButton* bSetNewMatrixSize;

   QDialog* dioSetNewMatSize;
   QLineEdit* bSetV1;
   QPushButton *enterMatSize;

   QMenuBar* menu;
```

```
    bool ifOpened = false;


//    std::ofstream fileO;
//    std::ifstream fileI;
    QString filePath = "";
public:
    window(QWidget *parent = nullptr);
protected:
    void saveAdjacencyMatrix(const QVector<QVector<double>>& matrix, const
QString& filename);
    QVector<QVector<double> > loadAdjacencyMatrix(const QString &filename,
bool &isOk);
    inline void setMatSize(const int &size);


    void keyPressEvent(QKeyEvent *event) override;
protected slots:
    //void slotShortcutTriggered(QAction *a);
    void slotCalculateClicked();
    void slotSetNewMatrixSizeFromDialog();
    void slotDioSetNewMatSizeShow();
    void slotMenuTriggered(QAction*);


};


window::window(QWidget *parent) : QWidget(parent) {


    // model init
    QTableView* view = new QTableView;
```

```
    model1 = new MatrixModel1(0, 0);
    view->setModel(model1);
    view->setToolTip("It is an adjacency matrix");

    QTableView* view2 = new QTableView;
    model2 = new MatrixModel2(1, 0);
    view2->setModel(model2);
    view2->setToolTip("It is a result row matrix");

    // widgets init
    bCalculate = new QPushButton("Find path");
    bCalculate->setToolTip("Press this button to calculate set adjacency matrix");
    bSetNewMatrixSize = new QPushButton("Set adjacency matrix size...");
    bSetNewMatrixSize->setToolTip("Press this button to set adjacency matrix size");
    resLabel = new QLabel("Dijkstra-s-algorithm");
    resLabel->setToolTip("This label is for showing program status");
    startPointA = new QLineEdit;
    startPointA->setToolTip("Enter there a start vertex from wich algorithm will
calculate a path to other vertices");


    ////////////////////////////////////// dialog init
    dioSetNewMatSize = new QDialog(this);
    enterMatSize = new QPushButton("Enter matrix size");
    bSetV1 = new QLineEdit;
    bSetV1->setValidator(new QIntValidator());
    QHBoxLayout *hbl1d = new QHBoxLayout;
    hbl1d->addWidget(new QLabel("Verticies number: "));
    hbl1d->addWidget(bSetV1);
```

```
QVBoxLayout *vbld = new QVBoxLayout;
vbld->addLayout(hbl1d);
vbld->addWidget(enterMatSize);
dioSetNewMatSize->setLayout(vbld);
////////////////////////////////

// connection
connect(bSetNewMatrixSize, &QPushButton::clicked, this,
        &window::slotDioSetNewMatSizeShow);
connect(enterMatSize, &QPushButton::clicked, this,
        &window::slotSetNewMatrixSizeFromDialog);
connect(bCalculate, &QPushButton::clicked, this,
        &window::slotCalculateClicked);

QMenu* mFile = new QMenu("File");
mFile->addAction("Open...", Qt::Key_F6);
mFile->addAction("Save", Qt::Key_F2);
mFile->addAction("Save as...", Qt::Key_F5);
mFile->addSeparator();
mFile->addAction("Clear");


menu = new QMenuBar;
menu->addMenu(mFile);
menu->addAction("Help...",Qt::Key_F8);
menu->addAction("About...");

connect(menu, &QMenuBar::triggered, this, &window::slotMenuTriggered);
```

```
bSetNewMatrixSize->setMinimumWidth(350);
bCalculate->setMinimumWidth(350);

graph1 = new GraphWidget();
graph1->setToolTip("It is a graph of your adjacency matrix");

// layout setup
QHBoxLayout* hbl = new QHBoxLayout;
hbl->addWidget(new QLabel("Start point: "));
hbl->addWidget(startPointA);
hbl->addSpacing(15);
hbl->addWidget(bSetNewMatrixSize);
hbl->addWidget(bCalculate);
QHBoxLayout* matHbl = new QHBoxLayout;
matHbl->addWidget(view);
matHbl->addWidget(view2);
QHBoxLayout* graphHbl = new QHBoxLayout;
graphHbl->addWidget(graph1);
//graphHbl->addWidget(graph2);
QVBoxLayout* vbl = new QVBoxLayout;
vbl->addWidget(menu);
vbl->addLayout(matHbl);
vbl->addLayout(graphHbl);
vbl->addLayout(hbl);
vbl->addWidget(resLabel);

setLayout(vbl);
```

```
    setFocusPolicy(Qt::StrongFocus);
}

void window::saveAdjacencyMatrix(const QVector<QVector<double> > &matrix,
const QString &filename){
    std::ofstream fileO;
    fileO.open(filename.toStdString());
    filePath = "";
    if (fileO.is_open()) {
        for (const auto& row : matrix) {
            for (int i = 0; i < row.size(); ++i) {
                fileO << row[i];
                if (i < row.size() - 1) {
                    fileO << " ";
                }
            }
            fileO << "\n";
        }
        filePath = filename;
        fileO.close();

        resLabel->setText("File saved.");
    } else {
        resLabel->setText("Unable to open file for writing.");
    }
}
```

```
QVector<QVector<double>> window::loadAdjacencyMatrix(const QString&
filename, bool& isOk) {
    QVector<QVector<double>> matrix;
    std::ifstream fileI;
    fileI.open(filename.toStdString());
    std::string line;

    int lineCount = 0;
    if (fileI.is_open()) {
        try {
            while (std::getline(fileI, line)) {
                ++lineCount;
                std::istringstream stream(line);
                QVector<double> row;
                std::string value;
                while (stream >> value) {
                    row.push_back(std::stod(value));
                }
                if(row.length() != 0) matrix.push_back(row);
            }
            ifOpened = true;
//            if(fileO.is_open()) fileO.close();
//            fileO.open(filename.toStdString());
            filePath = filename;
            resLabel->setText("File loaded.");
        } catch (...) {
            QApplication::beep();
```

```
            QMessageBox::information(0, "Error file reading",
QString::fromStdString("File has reading error on " + std::to_string(lineCount) + "
line"));
            resLabel->setText("Try fix your previous file or open another file");
            isOk = false;
            matrix.clear();
        }

        fileI.close();

    } else {
        resLabel->setText("Unable to open file for reading.");
    }

    isOk = true;
    return matrix;
}
void window::slotDioSetNewMatSizeShow() {
    dioSetNewMatSize->show();
}

void window::slotMenuTriggered(QAction *a)
{
    if(a->text() == "Open...") {
        try {
            bool isOk;
            QVector<QVector<double>> mat =
loadAdjacencyMatrix(QFileDialog::getOpenFileName(this, "Pick your save file", "",
"*.txt"), isOk);
            if(!isOk) {
```

```
            return;
        }
        setMatSize(mat.size());
        model1->setMat(mat);
    } catch (...) {
        resLabel->setText("ERROR");
    }
    QApplication::beep();
    resLabel->setText("File Loaded!");

} else if(a->text() == "Save"){
    auto matrix = model1->GetVectoredMat();
    std::ofstream fileO;
    fileO.open(filePath.toStdString());
    if (fileO.is_open()) {
        fileO.clear();
        for (const auto& row : matrix) {
            for (int i = 0; i < row.size(); ++i) {
                fileO << row[i];
                if (i < row.size() - 1) {
                    fileO << " ";
                }
            }
            fileO << "\n";
        }
        fileO.close();
    } else {
        resLabel->setText("Unable to open file for writing.");
```

```
        }
        QApplication::beep();
        resLabel->setText("File saved!");
    } else if(a->text() =="Save as..."){
        saveAdjacencyMatrix(model1->GetVectoredMat(),
QFileDialog::getSaveFileName(0, "Save file", "", "*.txt"));
        QApplication::beep();
        resLabel->setText("File saved!");
    } else if(a->text() == "Help..."){
          QDesktopServices::openUrl(QUrl::fromLocalFile("Help....docx"));
    } else if(a->text() == "About...") {
        QMessageBox::about(0, "About", "<h1>ShPath</h1>\n<h3>This is a program
written to solve the problem of searching short distances using Dijkstra`s algorithm
with path visualization created by Ilhin Serhii, student of KPI IASA, for
coursework.<h3>");
    } else if(a->text() == "Clear") {
        if(QMessageBox::warning(0, "Warning", "Do you really want to clear adjacency
matrix?",
                                QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes)
== QMessageBox::Yes){
            model1->clear();
            model2->clear();
            graph1->clear();
            QApplication::beep();
            resLabel->setText("Adjacency matrix cleared");
        }
    }
}
```

```
void window::slotCalculateClicked() {
    bool isAok = true;
    int A = startPointA->text().toInt(&isAok);
    QVector<QVector<double>> mat = model1->GetVectoredMat();
    int s = mat.size();
    if(s == 0){
        QApplication::beep();
        QMessageBox::information(0, "Error message", "Your adjacency matrix is
empty");
        return;
    }
    if(!isAok || A < 0){
        QApplication::beep();
        QMessageBox::information(0, "Error message", "You must enter a positive
integer as start vertex");
        return;
    }
    if(A>=s){
        QApplication::beep();
        QMessageBox::information(0, "Error message", QString::fromStdString("your
graph has "+std::to_string(s)+" vertices, but you enter vertex with number
"+std::to_string(A)));
        return;
    }
    if(model1->columnCount() <= 0){
        QApplication::beep();
        QMessageBox::information(0, "Error message", "Unexpected error");
        return;
    }
```

```
    graph1->setMat(mat);


    QVector<bool> map(s, 0);
    QVector<std::pair<double, std::string>> resMat(s,
std::pair<double,std::string>(INF, ""));
    resMat[A] = std::pair<double,std::string>(0, std::to_string(A)+" ");


    while (true) {
        int min = -1;
        double minValue = INF;
        for (int i = 0; i < s; ++i) {
            if (!map[i] && resMat[i].first < minValue) {
                minValue = resMat[i].first;
                min = i;
            }
        }

        if (min == -1) break;


        map[min] = true;


        for (int i = 0; i < s; ++i) {
            double item = mat[min][i];
            if (item != 0 && !map[i] && resMat[min].first + item < resMat[i].first) {
                resMat[i].first = resMat[min].first + item;
                resMat[i].second = resMat[min].second + std::to_string(i) + " ";
            }
        }
```

```
    }

    graph1->pointAllBlack();
    QVector<bool> matForResGraph(s, 0);
    for (const auto& i : resMat) {
        std::istringstream iss(i.second);
        std::string vert = "";
        while (iss >> vert) {
            matForResGraph[std::stoi(vert)] = 1;
        }
    }
    graph1->pointFindedNodes(matForResGraph);
    for (int i = 0; i < s; ++i) {
        model2->setRes(i, resMat[i].first, QString::fromStdString(resMat[i].second));
    }

    QApplication::beep();
    resLabel->setText("Matrix has been calculated!!");
}

void window::slotSetNewMatrixSizeFromDialog() {
    setMatSize(bSetV1->text().toInt());

}

inline void window::setMatSize(const int& size){
    model1->setSize(size);
    model2->setColumnCount(size);
```

```
}


void window::keyPressEvent(QKeyEvent *event)
{
    if (event->key() == Qt::Key_F6) {
        try {
            bool isOk;
            QVector<QVector<double>> mat =
loadAdjacencyMatrix(QFileDialog::getOpenFileName(this, "Pick your save file", "",
"*.txt"), isOk);
            if(!isOk) {
                return;
            }
            setMatSize(mat.size());
            model1->setMat(mat);
        } catch (...) {
            resLabel->setText("ERROR");
        }
        QApplication::beep();
    } else if (event->key() == Qt::Key_F2) {
        auto matrix = model1->GetVectoredMat();
        std::ofstream fileO(filePath.toStdString());
        if (fileO.is_open()) {
            for (const auto& row : matrix) {
                for (int i = 0; i < row.size(); ++i) {
                    fileO << row[i];
                    if (i < row.size() - 1) {
                        fileO << " ";
                    }
```

```
            }
            fileO << "\n";
        }
        fileO.close();
    } else {
        resLabel->setText("Unable to open file for writing.");
    }
    QApplication::beep();
  } else if (event->key() == Qt::Key_F5) {
    saveAdjacencyMatrix(model1->GetVectoredMat(),
QFileDialog::getSaveFileName(this, "Save file", "", "*.txt"));
    QApplication::beep();
  }

  QWidget::keyPressEvent(event);
}

class GraphWidget : public QGraphicsView {
  Q_OBJECT
public:
  GraphWidget(QWidget *parent = nullptr);
  void setMat(QVector<QVector<double>> m);
  void pointFindedNodes(QVector<bool> v);
  void pointAllBlack();
  void clear();
protected:
  void mouseMoveEvent(QMouseEvent *event) override;

private:
```

```
    QVector<QVector<double>> adjacencyMatrix;
    QGraphicsScene* scene;
    QVector<QGraphicsEllipseItem*> nodes;
    QVector<QGraphicsSimpleTextItem*> nodeLabels;
    QMap<QPair<int, int>, QGraphicsLineItem*> edges;
};
GraphWidget::GraphWidget(QWidget *parent) : QGraphicsView(parent) {
    scene = new QGraphicsScene(this);
    setScene(scene);



}

void GraphWidget::setMat(QVector<QVector<double>> m) {
    adjacencyMatrix = m;
    for (int i = 0; i < adjacencyMatrix.size(); ++i) {
        for (int j = 0; j < adjacencyMatrix.size(); ++j) {
            if(adjacencyMatrix[i][j] != 0) adjacencyMatrix[j][i] = adjacencyMatrix[i][j];
            else if(adjacencyMatrix[j][i] != 0) adjacencyMatrix[i][j] =
adjacencyMatrix[j][i];
        }
    }

    scene->clear();
    int nodeCount = adjacencyMatrix.size();
    nodes.resize(nodeCount);
    nodeLabels.resize(nodeCount);

for (int i = 0; i < nodeCount; ++i) {
```

```
        nodes[i] = scene->addEllipse(-10, -10, 20, 20);
        nodes[i]->setPos(i*34, std::cos(i*M_PI)*100+std::sin(i*M_PI/4)*40);
        nodes[i]->setFlag(QGraphicsItem::ItemIsMovable, true);

        nodeLabels[i] = new QGraphicsSimpleTextItem((QString::number(i)));
        nodeLabels[i]->setPos(nodes[i]->pos() - QPointF(5, 5));
        nodeLabels[i]->setBrush(QBrush(QColor(92, 39, 254)));
        scene->addItem(nodeLabels[i]);
    }

    for (int i = 0; i < nodeCount; ++i) {
        for (int j = i + 1; j < nodeCount; ++j) {
            if (adjacencyMatrix[i][j] != 0) {
                QGraphicsLineItem *edge = scene->addLine(nodes[i]->pos().x(), nodes[i]-
>pos().y(), nodes[j]->pos().x(), nodes[j]->pos().y(), QPen(QColor(82, 255, 255), 1));
                edges[QPair<int, int>(i, j)] = edge;
            }
        }
    }
}

void GraphWidget::pointAllBlack()  {
    for (int i = 0; i < nodes.size(); ++i) {
        nodes[i]->setPen(QPen(Qt::black));
    }
}

void GraphWidget::clear()
{
```

```
    scene->clear();
}

void GraphWidget::pointFindedNodes(QVector<bool> v) {
    for (int i = 0; i < v.size(); ++i) {
        if(v[i]) nodes[i]->setPen(QPen(Qt::green));
        else nodes[i]->setPen(QPen(Qt::red));
    }
}

void GraphWidget::mouseMoveEvent(QMouseEvent *event) {
    QGraphicsView::mouseMoveEvent(event);

    for (int i = 0; i < nodes.size(); ++i) {
        nodeLabels[i]->setPos(nodes[i]->pos() - QPointF(5, 5));
        for (int j = i + 1; j < nodes.size(); ++j) {

            QGraphicsLineItem *edge = edges.value(QPair<int, int>(i, j), nullptr);
            if (edge) {
                QPointF p1 = nodes[i]->scenePos();
                QPointF p2 = nodes[j]->scenePos();
                edge->setLine(QLineF(p1, p2));
            }
        }
    }
}
class MatrixModel1 : public QAbstractTableModel {
    Q_OBJECT
```

```
public:
    explicit MatrixModel1 (int rows, int cols, QObject* obj = nullptr);

    QVariant data (const QModelIndex& index, int role) const override;
    QVariant data (int i, int j) const;
    bool setData (const QModelIndex& index, const QVariant& data, int role)
override;
    void setData (int i, int j, const QVariant& data);
    void clearData();

    int rowCount (const QModelIndex& index = QModelIndex()) const override;
    int columnCount ( const QModelIndex& parent = QModelIndex()) const override;

    QVariant headerData (int section, Qt::Orientation orientation, int role =
Qt::DisplayRole) const override;
    Qt::ItemFlags flags (const QModelIndex&) const override;

    QVector<QVector<double>> GetVectoredMat();
    void setMat(const QVector<QVector<double>>& mat);

    void setSize(int rowCount);

    void clear();
private:
    QHash<QModelIndex, QString> mat;
    QVector<QVector<double>> VectoredMat;
    int rows;
    int cols;
};
```

```
class MatrixModel2 : public QAbstractTableModel {
    Q_OBJECT
public:
    explicit MatrixModel2 (int rows, int cols, QObject* obj = nullptr);

    QVariant data (const QModelIndex& index, int role) const override;
    int data (int i, int j) const;
    bool setData (const QModelIndex& index, const QVariant& data, int role)
override;
    void setData (int i, int j, const double& data);
    void clearData();

    int rowCount (const QModelIndex& index = QModelIndex()) const override;
    int columnCount ( const QModelIndex& parent = QModelIndex()) const override;

    QVariant headerData (int section, Qt::Orientation orientation, int role =
Qt::DisplayRole) const override;
    Qt::ItemFlags flags (const QModelIndex&) const override;

//    void setRowCount (int);
    void setColumnCount (int);

    QVector<QString> GetVectoredMat();

    void clear();

    void setRes(int Vertice, int pathLenght, QString path);
```

```
private:
    QHash<QModelIndex, QString> mat;
    QVector<QString> v;
    int rows;
    int cols;
};
MatrixModel1::MatrixModel1(int rowsCount, int colsCount, QObject* obj)
    : QAbstractTableModel(obj), rows(rowsCount), cols(colsCount)  { }

QVariant MatrixModel1::data(const QModelIndex &index, int role) const {
    if(!index.isValid() || index.row() > rows || index.column() > cols) return
QVariant();
    return (role == Qt::DisplayRole || role == Qt::EditRole) ? mat.value(index, 0) :
QVariant();
}

QVariant MatrixModel1::data(int i, int j) const {
    return mat.value(index(i, j));
}

Qt::ItemFlags MatrixModel1::flags( const QModelIndex& index ) const {
    Qt::ItemFlags flags = QAbstractTableModel::flags( index );
    return index.isValid() ? (flags | Qt::ItemIsEditable) : flags;
}

bool MatrixModel1::setData( const QModelIndex& index, const QVariant& value,
int role ) {
    if( !index.isValid() || role != Qt::EditRole || index.row() > rows || index.column() >
cols) return false;
```

```
    QString toStr = value.toString();
    if(toStr == "") return false;
    bool isOk;
    double buf = toStr.toDouble(&isOk);
    if(!isOk || buf < 0) {
        QMessageBox::information(0, "Error", "You must write some positive integer (like 3) or positive floating point number using dot(like 3.14)");
        return false;
    }

    VectoredMat[index.row()][index.column()] = buf;

    mat[index] = toStr;
    emit dataChanged(index, index);
    return true;
}

int MatrixModel1::rowCount( const QModelIndex& parent ) const {
    Q_UNUSED(parent)
    return rows;
}

int MatrixModel1::columnCount( const QModelIndex& parent ) const {
    Q_UNUSED(parent)
    return cols;
}
```

```
QVariant MatrixModel1::headerData (int section, Qt::Orientation orientation, int role) const {
    if( role != Qt::DisplayRole ) return QVariant();
    if( orientation == Qt::Vertical || orientation == Qt::Horizontal) return section;
    return QVariant();
}

void MatrixModel1::setSize(int rowCount) {
    this->rows = rowCount;
    this->cols = rowCount;
    VectoredMat = QVector<QVector<double>>(rowCount, QVector<double>(rowCount, 0));
    emit layoutChanged();
}

void MatrixModel1::clear()
{
    mat.clear();
    VectoredMat.clear();
}

void MatrixModel1::setData(int i, int j, const QVariant& data) {
    QModelIndex ind = index(i, j);
    beginInsertColumns(ind,0, 1);
    mat.insert(ind, data.toString());
    endInsertColumns();
}

void MatrixModel1::clearData() {
```

```
    beginResetModel();
    mat.clear();
    endResetModel();
    setSize(0);
}

QVector<QVector<double>> MatrixModel1::GetVectoredMat() {
    return VectoredMat;
}

void MatrixModel1::setMat(const QVector<QVector<double> > &matrix) {
    for (int i = 0; i < VectoredMat.size(); ++i) {
        for (int j = 0; j < VectoredMat.size(); ++j){
            mat[index(i, j)] = QString::fromStdString(std::to_string(matrix[i][j]));
            VectoredMat[i][j] = matrix[i][j];
        }
    }
}

MatrixModel2::MatrixModel2(int rowsCount, int colsCount, QObject* obj)
    : QAbstractTableModel(obj), rows(rowsCount), cols(colsCount)  { }

QVariant MatrixModel2::data(const QModelIndex &index, int role) const {
    if(!index.isValid() || index.row() > rows || index.column() > cols) return
QVariant();
    return (role == Qt::DisplayRole || role == Qt::EditRole) ? mat.value(index, 0) :
QVariant();
}
```

```
//double MatrixModel2::data(int i, int j) const {
//    return mat.value(index(i, j)).first;
//}

Qt::ItemFlags MatrixModel2::flags( const QModelIndex& index ) const {
   Qt::ItemFlags flags = QAbstractTableModel::flags( index );
   return index.isValid() ? (flags | Qt::ItemIsEditable) : flags;
}

bool MatrixModel2::setData( const QModelIndex& index, const QVariant& value,
int role ) {
   if( !index.isValid() || role != Qt::EditRole || index.row() > rows || index.column() >
cols) return false;
   mat[index] = value.toString();
   v[index.column()] = value.toString();
   emit dataChanged(index, index);
   return true;
}

int MatrixModel2::rowCount( const QModelIndex& parent ) const {
   Q_UNUSED(parent)
   return rows;
}

int MatrixModel2::columnCount( const QModelIndex& parent ) const {
   Q_UNUSED(parent)
   return cols;
}
```

```
QVariant MatrixModel2::headerData (int section, Qt::Orientation orientation, int role) const {
    if( role != Qt::DisplayRole ) return QVariant();
    if( orientation == Qt::Vertical || orientation == Qt::Horizontal) return section;
    return QVariant();
}

//void MatrixModel2::setRowCount(int rowCount) {
//    this->rows = rowCount;
//    emit layoutChanged();
//}
void MatrixModel2::setColumnCount(int colCount) {
    v = QVector<QString>(colCount);
    this->cols = colCount;
    emit layoutChanged();
}

QVector<QString> MatrixModel2::GetVectoredMat() {
    return v;
}

void MatrixModel2::clear()
{
    mat.clear();
    setColumnCount(0);
}

void MatrixModel2::setRes(int Vertice, int pathLenght, QString path) {
    QModelIndex ind = index(0, Vertice);
```

```
    beginInsertColumns(ind,0, 1);
    v[Vertice] = QString::number(pathLenght) + " (" + path + ")";
    mat.insert(ind, v[Vertice]);
    endInsertColumns();
    emit layoutChanged();
}
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    window w;
    w.setMinimumSize(895, 700);
    w.show();
    return a.exec();
}
```