МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з Основ програмування
(назва дисципліни)
на тему: пошук найкоротших шляхів в мережі

Студента 1 курсу, групи ІП-13
Бондаренко Максима Вікторовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник Головченко Максим Миколайович
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

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

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

Київ- 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-13 Семестр 2

ЗАВДАННЯ
на курсову роботу студента
Бондаренко Максима Вікторовича

(прізвище, ім'я, по батькові)

1. Тема роботи Пошук найкоротших шляхів в мережі

2. Строк здачі студентом закінченої роботи 25.08.2022

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Дата видачі завдання 10.02.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	04.06.2022	
4.	Розробка сценарію роботи програми	04.06.2022	
6.	Узгодження сценарію роботи програми з керівником	04.06.2022	
5.	Розробка (вибір) алгоритму рішення задачі	06.06.2022	
6.	Узгодження алгоритму з керівником	06.06.2022	
7.	Узгодження з керівником інтерфейсу користувача	06.06.2022	
8.	Розробка програмного забезпечення	09.06.2022	
9.	Налагодження розрахункової частини програми	09.06.2022	
10.	Розробка та налагодження інтерфейсної частини програми	10.06.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	23.06.2022	
12.	Тестування програми	24.06.2022	
13.	Підготовка пояснювальної записки	03.07.2022	
14.	Здача курсової роботи на перевірку	25.08.2022	
15.	Захист курсової роботи	26.08.2022	

Студент _____
(підпис)

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

"___" _____2022 р.

АНОТАЦІЯ

Пояснювальна записка до курсової: 55 сторінок, 16 таблиць, 20 рисунків.

Об’єкт дослідження: Пошук найкоротших шляхів в мережі.

Мета роботи: дослідження алгоритмів пошуку найкоротших шляхів в мережі, створення програмного забезпечення для пошуку найкоротшого шляху між усіма вершинами в мережі.

Опановано розробку програмного забезпечення з використанням ООП. Приведені змістовну постановку задачі, індивідуальні математичні моделі, а також описано детальний процес розв’язання задачі.

Виконана програмна реалізація пошуку найкоротшого шляху між усіма вершинами мережі.

ГРАФ, МАТРИЦЯ СУМІЖНОСТІ, ПОШУК НАЙКОРОТШИХ ШЛЯХІВ, МЕТОД ФЛОЙДА, МЕТОД ДАНЦИГА.

ЗМІСТ

Кафедра інформатики та програмної інженерії 3
ВСТУП 5
1 ПОСТАНОВКА ЗАДАЧІ 6
2 ТЕОРИТИЧНІ ВІДОМОСТІ 7
2.1 Алгоритм Флойда-Уоршелла. 7
2.2 Алгоритм Данцига. 8
3 ОПИС АЛГОРИТМІВ 9
3.1 Загальний алгоритм 9
3.2 Алгоритм Флойда-Уоршелла. 10
3.3 Алгоритм Данцига. 11
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 12
4.1 Діаграма класів програмного забезпечення 12
4.2 Опис методів частин програмного забезпечення 13
4.2.1 Стандартні методи 13
4.2.2 Користувацькі методи 16
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 18
5.1 План тестування 18
5.2 Приклади тестування 19
6 ІНСТРУКЦІЯ КОРИСТУВАЧА 24
6.1 Робота з програмою 24
6.2 Формат вхідних та вихідних даних 28
6.3 Системні вимоги 28
7 АНАЛІЗ РЕЗУЛЬТАТІВ 29
ВИСНОВОК 34
ПЕРЕЛІК ПОСИЛАНЬ 35
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 36
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ 39
main.py 40
MainWindow.py 40
Graph.py 48
SolutionWindow.py 53

ВСТУП

Пошук найкоротшого шляху в мережі – дуже потрібна та практична задача, вона має багато практичних застосувань. Наприклад, прокладання найкоротшого маршруту для водіїв на картах, що неймовірно важливо для усієї транспортної інфраструктури.

На рівні програмного коду цю задачу ми розглядаємо як пошук найкоротшого шляху графу, що є однією з найважливіших класичних задач теорії графів, для вирішення якої існує багато алгоритмів, два з яких будуть використані для реалізації цієї роботи (алгоритм Флойда-Уоршела, алгоритм Данцига).

В рамках пояснювальної записки буде описано задачу, теоретичні відомості по заданим алгоритмам, самі алгоритми будуть повністю описані. На наступномі кроці буде здійснено опис програмного забезпечення (класи та їх методи). Буде описано теступання програми за наведеним планом. Також буде створено інструкцію для користувача з повним описанням усіх можливостей програми. Наприкінці проаналізуємо результати роботи та опишемо кінцевий висновок.

За допомогою застосунку користувач може легко виконувати операції пов’язані з пошуком найкоротшого шляху мережі. Простий та інтуітивний інтерфейс допоможе розібратися з програмою достатньо швидко. Її можна використовувати не тільки зі своїми даними, а й з тими що генерує вона сама.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити всі найкоротші шляхи в мережі наступними методами:

а) метод Флойда;

б) метод Данцига;

Вхідними даними для даної роботи є зважений орієнтований граф та матриця суміжності з вагами ребер, яка задає його,

$$A = \begin{matrix} 0 & d_{i,j} & \dots \\ \infty & 0 & \dots \\ d_{i,j} & \dots & 0 \end{matrix}$$, де

$d_{i,j}$ – довжина ребра, що йде від вершини *i* до вершини *j*;

i, j – індекси елементів матриці;

∞ – ребро з напрямком $i \rightarrow j$ відсутнє;

0 – довжина петлі, що завжди дорівнює нулю в головній діагоналі.

Програмне забезпечення повинно оброобробляти матрицю, розмірність якої знаходиться в межах від 1 до 10.

Вихідними даними для даної роботи є зображення заданого у матриці графа, усі найкоротші шляхи мережи, матриця з довжинами найкоротших шляхів між вершинами. Програмне забезпечення видає розв’язок за умови, що граф не містить контуру від’ємної довжини. У цьому випадку виводиться відповідне повідомлення.

2 ТЕОРИТИЧНІ ВІДОМОСТІ

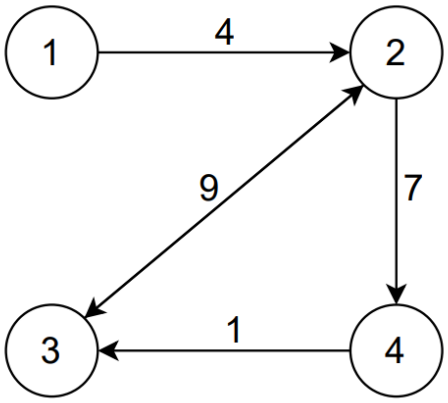
Існують різні способи задання графа. У нашому випадку для виконання задачі використаємо задання графу матрицею суміжності, елементи якої – вага ребер, що з'єднують вершини.

Приклад. Матриця суміжності з вагами ребер (таблиця 2.1) для зваженого орграфа (рисунок 2.1):

Таблиця 2.1 – матриця суміжності

	1	**2**	**3**	**4**
1	0	4	∞	∞
2	∞	0	9	7
3	∞	9	0	∞
4	∞	∞	1	0

Рисунок 2.1 – зважений орграф



2.1 Алгоритм Флойда-Уоршелла.

Алгоритм Флойда-Уоршелла виконує пошук найкоротших шляхів між усіма вершинами зваженого орграфа, його суть полягає в порівнянні всіх можливих шляхів в графі та виборі найкоротших з них.

Основною частиною алгоритму Флойда-Уоршелла є дана нерівність:

$d_{i,k} + d_{k,j} < d_{i,j}$ $(i \neq j, i \neq k, j \neq k)$, яка і являє собою порівняння усіх можливих шляхів між двома вершинами, поки таким чином не досягнеться найменший.

На початковому етапі ми маємо матрицю суміжности D_0 розмірності N×N, елементи якої є вагою ребер графа. Для цілого k, що поступово набуває значення 1...N, алгоритм почергово перевіряє за елементами матриці D_{k-1}, чи справджується основна нерівність для кожного елементу, та записує цю суму в нову матрицю D_n, якщо нерівність справдилась. Також на кожній ітерації потрібно перевіряти справдження даної нерівності $D[i,i] < 0$, що свідчить про наявність від'ємного контуру в графі та неможливість застосування алгоритму.

В результаті обробки алгоритмом отримаємо нову матрицю D_n з довжиною всіх найкоротших шляхів між вершинами i та j.

2.2 Алгоритм Данцига.

Алгоритм Данцига також виконує пошук найкоротших шляхів між усіма вершинами зваженого орієнтованого графа та є дуже схожим на алгоритм Флойда, відрізняючись тільки іншим порядком виконання тих самих операцій

На початковому етапі ми маємо матрицю суміжности D_0 розмірності N×N, елементи якої є вагою ребер графа. Для цілого m, що поступово набуває значення 1...N, алгоритм почергово перевіряє за елементами матриці $D_{m-1,}$ дані формули:

$$d_{mj}^{m} = min_{i=1,2,\dots,m-1}(d_{mi}^{0} + d_{mi}^{m-1})(j = 1,2,\dots,m-1)$$

$$d_{im}^{m} = min_{i=1,2,\dots,m-1}\left(d_{ij}^{m-1} + d_{jm}^{0}\right)(i = 1,2,\dots,m-1)$$

$$d_{ij}^{m} = \min\left(d_{im}^{m} + d_{mj,}^{m} d_{ij,}^{m-1}\right)(i,j = 1,2,\dots,m-1),$$

визначаючи елементи матриці D_m.

Алгоритм так само потребує перевірок на від’ємні контури на кожній ітерації.

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено у таблиці 3.1.

Таблиця 3.1 – основні змінні та їх призначення.

Змінна	Призначення
matrix	Задана користувачем матриця суміжності з вагами ребер.
matrix_size	Розмір матриці суміжності.
path	Матриця для зберігання послідовності вершин графу.
cost	Матрия для зберігання довжини найкоротших шляхів між усіма вершинами.
i, j, k, m	Проміжні індекси елементів.
n	Розмірність матриці (кількість вершин графу).
min	Проміжний елемент для визначення мінімального шляху за алгоритмом Данцига.
way	Строка для зберігання всіх вершин шляху.
start	Вершина початку шляху.
end	Вершина кінця шляху.
new_start, new_end	Тимчасові допоміжні змінні для знаходження усіх вершин шляху.

3.1 Загальний алгоритм

1. ПОЧАТОК
2. Зчитати розмір матриці
   2.1. ЯКЩО розмір не належить проміжку (1-10), ТО вивести помилку та повернутися до пункту 2
3. Зчитати матрицю.
   3.1. ЯКЩО виявлено значення відмінне від цілого числа або «Inf», ТО вивести помилку та повернутися до пункту 3
   3.2. ЯКЩО на головній діагоналі виявлено 0, ТО вивести помилку та повернутися до пункту 3

3.3. ЯКЩО виявлено пусте поле, ТО вивести помилку та повернутися до пункту 3

4. Зчитати вибір методу
   4.1. ЯКЩО метод не обрано, ТО вивести помилку та повернутися до пункту 4
   4.2. ЯКЩО вибрано метод Флойда, ТО використати алгоритм Флойда (3.2)
   4.3. ЯКЩО вибрано метод Данцига, ТО використати алгоритм Данцига (3.3)
5. ЯКЩО виявлено від'ємний контур, ТО повернутися до пункту 3
6. Вивести граф, побудований на основі матриці.
7. Вивести список усіх шляхів між усіма парами вершин.
8. Вивести матрицю довжин найкоротших шляхів між вершинами.
9. Вивести кількість ітерацій алгоритму.
10. Записати результуючу матрицю, список шляхів та кількість ітерацій у зовнішній .txt файл.
11. КІНЕЦЬ.

3.2 Алгоритм Флойда-Уоршелла.

```
ПОЧАТОК
cost = matrix
ВІД i = 0 ДО matrix_size
        ВІД j = 0 ДО matrix_size
                path[i][j] = j
ВІД k = 0 ДО matrix_size
        minPath(matrix_size, k)
КІНЕЦЬ

minPath(m, k):
ПОЧАТОК
ВІД i = 0 до m
        ВІД j = 0 до m
                ЯКЩО cost[i][k] + cost[k][j] < cost[i][j] ТО
                        cost[i][j] = cost[i][k] + cost[k][j]
                        path[i][j] = k
                ЯКЩО cost[i][i] < 0 ТО
                        ВИКЛЮЧЕННЯ
```

3.3 Алгоритм Данцига.

```
ПОЧАТОК
cost = matrix
ВІД i = 0 ДО matrix_size
        ВІД j = 0 ДО matrix_size
                path[i][j] = j
ВІД m = 2 ДО matrix_size
        ВІД k = 0 ДО m
                minPath(m, k)
        findMinMj(m)
        findMinIm(m)
ВІД k = 0 ДО matrix_size
        minPath(matrix_size, k)
КІНЕЦЬ

findMinIm(m):
ПОЧАТОК
ВІД i = 0 ДО m
        min = cost[i][m]
        ВІД k = 0 ДО m
                ЯКЩО i != k && cost[i][k] + cost[k][m] < min ТО
                        path[i][m] = k
                        min = cost[i][k] + cost[k][m]
        cost[i][m] = min
КІНЕЦЬ

findMinMj(m):
ПОЧАТОК
ВІД i = 0 ДО m
        min = cost[m][j]
        ВІД k = 0 ДО m
                ЯКЩО j != k && cost[m][k] + cost[k][j] < min ТО
                        path[m][j] = k
                        min = cost[m][k] + cost[k][j]
        cost[m][j] = min
КІНЕЦЬ
```

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

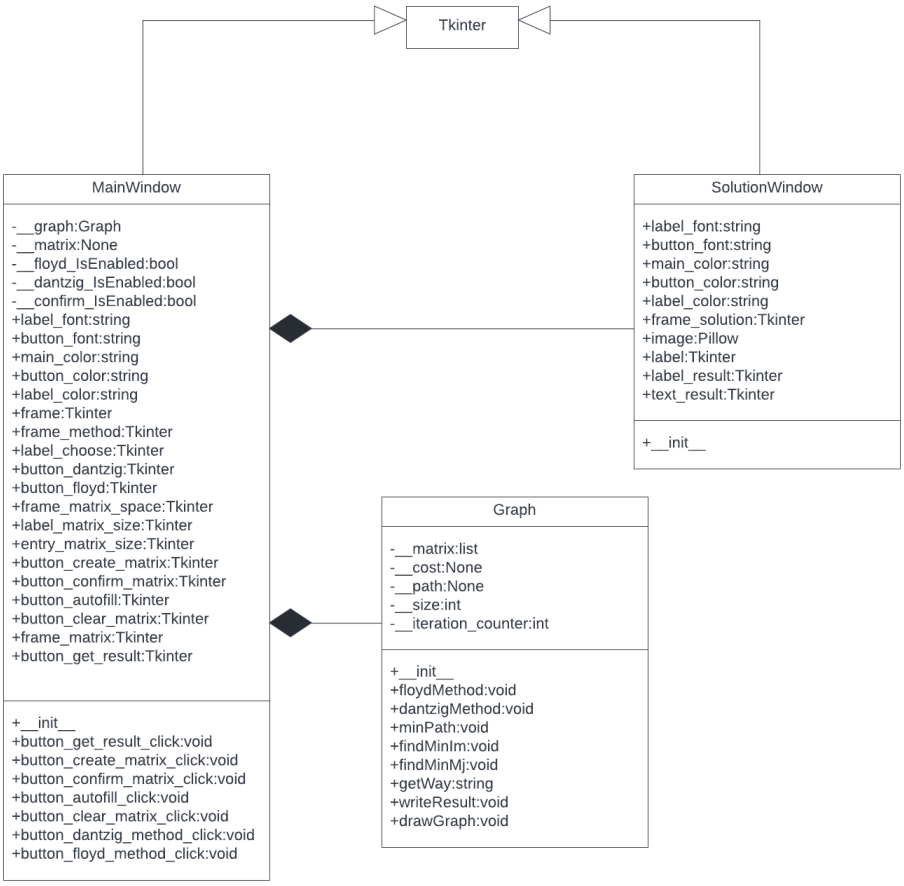


Рисунок 4.1 – Діаграма класів.

Програмне забезпечення включає в себе 3 класси. Клас MainWindow повністю відповідає за графічний інтерфейс з яким взаємодіє користувач. Клас Graph відповідає за основний функціонал програми, роботу з графом та алгоритмами. Клас SolutionWindow відповідає за вікно, яке містить усю результуючу інформацію програми.

4.2 Опис методів частин програмного забезпечення

4.2.1 Стандартні методи

У таблиці 4.1 наведено стандартні методи, які було використано при розробці програмного забезпечення.

Таблиця 4.1 – Стандартні методи

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	Tkinter	pack	Додавання віджета у вікно	віджет	
2	Tkinter	place	Розміщення віджета за певними координатами та з заданими розмірами	віджет	
3	Tkinter	insert	Додавання тексту у віджет	віджет, int, str	
4	Tkinter	get	Отримання тексту з віджету	віджет	string
5	Tkinter	delete	Видалення тексту з віджету	віджет, int, int	
6	Tkinter	destroy	Видалення віджета з вікна	віджет	
7	Tkinter	place	Вказання розмірів та координат віджета	віджет	
8	Tkinter.messagebox	showwarn-ing	Виведення повідомлення про помилку	string, string	

Продовження таблиці 4.1

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
9	Tkinter.messagebox	showinfo	Виведення інформативного повідомлення	string, string	
10	Tkinter	geometry	Встановлення розміру та позиції вікна на екрані	string	
11	Tkinter	title	Встановлення назви вікна	string	
12	Tkinter	resizable	Надання можливості змінювати розміри вікна	bool, bool	
13	Tkinter	mainloop	Указує на запуск вікна	Tkinter object	
14	Random	choice	Вибір випадкового елементу зі списку	list	Елемент списку
15	File	read	Зчитування файлу в текст	file	string
16	File	write	Записк тексту в файл	string	
17	File	open	Відкриття файлу в певному режимі	string, string, string	File object
18	PIL.Image	open	Відкриття зображення для подальшого використання	string	Image object

Продовження таблиці 4.1

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
19	PIL.ImageTk	PhotoImage	Зображення картинки на віджетах	Image object	віджет
20		append	Додавання елементу в кінець списку	list, елемент для додавання	list
21		isdigit	Перевірка елемента, чи є він числом	string	bool
22		range	Повертає послідовність чисел, що ітеруються	int, int, int	Range object
23		int	Переведення строки, байтового об’єкту або числа в цілочисельний тип даних	String/double/object, int	int
24	networkx	add_node	Додання вершини у граф	int	
25	networkx	add_edge	Додання ребра у граф	int, int	
26	networkx	draw	Виведення графа	Граф, позиція	
27	matplotlib	savefig	Збереження картинки	string	

4.2.2 Користувацькі методи

У таблиці 4.2 наведено усі користувацькі методи, що використовувались у процесі розробки програми, та їх опис.

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	MainWindow	button_get_result_click	Отримання кінцевого результату		
2	MainWindow	button_create_matrix_click	Створення простору для матриці		
3	MainWindow	button_confirm_matrix_click	Підтвердження заповненої матриці		
4	MainWindow	button_autofill_click	Автоматичне заповнення матриці		
5	MainWindow	button_clear_matrix_click	Очищення всіх уведених даних		
6	MainWindow	button_dantzig_method_click	Застосування методу Данцига		
7	MainWindow	button_floyd_method_click	Застосування методу Флойда		

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
8	Graph	floydMethod	Застосування алгоритму Флойда для вирішення задачі		
9	Graph	dantzigMethod	Застосування алгоритму Данцига для вирішення задачі		
10	Graph	minPath	Порівняння довжин шляхів	int, int	
11	Graph	findMinIm	Пошук мінімального значення, поки j=m	int	
12	Graph	findMinMj	Пошук мінімального значення, поки i=m	int	
13	Graph	getWay	Побудова найкоротшого шляху між двома точками	int, int	string
14	Graph	writeResult	Запис результучих даних		
15	Graph	drawGraph	Побудова графу		

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

При розробці об’ємного програмного забезпечення можна легко впустити деякі моменти, які можуть впливати на працездатність програми, тому важливо проводити тестування різними ситуаціями, що можуть призвести до конфліктів, для подальшого виправлення цих багів.

Для перевірки коректоності роботи програми, на основі наявної інформації про додаток та його функціонал, складемо тест-план.

а) Тестування коректності введених значень розміру.

б) Тестування коректності введених значень матриці суміжності.

   1) Тестування при введенні значень головної діагоналі, які не дорівнюють 0.
   2) Тестування при введенні некоректних значень.

в) Тестування коректності роботи програми при виконанні не всіх вимог програми.

   1) Тестування роботи програми при відсутності вибору алгоритму.

г) Тестування роботи програми при заданні некоректного графу.

   1) Тестування роботи програми при заданні графу, що має від’ємний контур.

д) Тестування коректності роботи методів Флойда та Данцига для графу з додатними значеннями ребер.

   1) Перевірка коректності роботи методу Флойда.
   2) Перевірка коректності роботи методу Данцига.

е) Тестування коректності роботи методів Флойда та Данцига для графу з від’ємними значеннями ребер

   1) Перевірка коректності роботи методу Флойда.
   2) Перевірка коректності роботи методу Данцига.

ж) Тестування побудови графу.

5.2 Приклади тестування

Проведемо тестування програмного забезпечення згідно з розробленим планом та запишемо очікуваний результат і стан програми після проведення випробувань кожного тесту в окрему таблицю (таблиці 5.1-5.10).

Таблиця 5.1 – Тестування при введенні некоректних значень розміру матриці

Мета тесту	Перевірити можливість введення некоректних значень розміру матриці.
Початковий стан програми	Відкрите вікно програми.
Вхідні дані	Розмір: 0
Схема проведення тесту	Натискаємо кнопку «Створити матрицю».
Очікуваний результат	Видано помилку «Введіть коректний розмір (1-15)», простір для матриці не створюється.
Стан програми після проведення випробувань	Видано помилку «Введіть коректний розмір (1-15)», простір для матриці не створено.

Таблиця 5.2 – Тестування при введенні значень головної діагоналі, які не дорівнюють 0.

Мета тесту	Перевірити можливість введення значень головної діагоналі, відмінних від 0.
Початковий стан програми	Відкрите вікно програми, створено простір для матриці.
Вхідні дані	Матриця: 1 1 2 0
Схема проведення тесту	Натискаємо кнопку «Підтвердити»
Очікуваний результат	Видано помилку «Основна діагональ може складатися лише з 0», матриця не підтверджується.

Продовження таблиці 5.2

Стан програми після проведення випробувань	Видано помилку «Основна діагональ може складатися лише з 0», матриця не підтверджена.

Таблиця 5.3 – Тестування роботи програми при введенні некоректних значень в матрицю суміжності.

Мета тесту	Перевірити можливість введення некоректних даних у матрицю суміжності
Початковий стан програми	Відкрите вікно програми, простір для матриці створено.
Вхідні дані	Матриця: 0 a 2 0
Схема проведення тесту	Натискаємо на кнопку «Підтвердити».
Очікуваний результат	Видано помилку «Введіть коректні значення (Inf або ціле число)», матриця не підтверджується.
Стан програми після проведення випробувань	Видано помилку «Введіть коректні значення (Inf або ціле число)», матриця не підтверджена.

Таблиця 5.4 – Тестування роботи програми при відсутності вибору алгоритма роботи.

Мета тесту	Перевірити можливість роботи програми без вибору алгоритму.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, матриця підтверджена.
Вхідні дані	Алгоритм не обрано
Схема проведення тесту	Натискаємо на кнопку «Отримати результат».

Продовження таблиці 5.4

Очікуваний результат	Видано помилку «Оберіть метод розв’язання», граф не будується.
Стан програми після проведення випробувань	Видано помилку «Оберіть метод розв’язання», граф не побудувався.

Таблиця 5.5 – Тестування роботи програми з графом, який має від’ємний контур.

Мета тесту	Перевірити можливість опрацювання графу з від’ємним контуром
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм, матриця підтверджена.
Вхідні дані	Матриця графу з від’ємним контуром
Схема проведення тесту	Натискаємо на кнопку «Отримати результат».
Очікуваний результат	Видано помилку «Знайдений негативний контур». Введіть нову матрицю», результат не видається
Стан програми після проведення випробувань	Видано помилку «Знайдений негативний контур». Введіть нову матрицю», результат не видався.

Таблиця 5.6 – Перевірка коректності роботи методу Флойда для графу з додатними значеннями ребер.

Мета тесту	Перевірити роботу алгоритму Флойда для графу з додатними значеннями ребер.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм.
Вхідні дані	Матриця суміжності графу з додатними значеннями.

Продовження таблиці 5.6

Схема проведення тесту	Натискаємо на кнопку «Отримати результат».
Очікуваний результат	Виведення списку найкоротших шляхів та матриці з довжинами цих шляхів.
Стан програми після проведення випробувань	Виведено відповідну матрицю та шляхи.

Таблиця 5.7 – Перевірка коректності роботи методу Данцига для графу з додатними значеннями ребер.

Мета тесту	Перевірити роботу алгоритму Данцига для графу з додатними значеннями ребер.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм.
Вхідні дані	Матриця суміжності графу з додатними значеннями.
Схема проведення тесту	Натискаємо на кнопку «Отримати результат».
Очікуваний результат	Виведення списку найкоротших шляхів та матриці з довжинами цих шляхів.
Стан програми після проведення випробувань	Виведено відповідну матрицю та шляхи.

Таблиця 5.8 – Перевірка коректності роботи методу Флойда для графу з від’ємними значеннями ребер.

Мета тесту	Перевірити роботу алгоритму Флойда для графу з від’ємнихми значеннями ребер.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм.
Вхідні дані	Матриця суміжності графу з від’ємними значеннями.
Схема проведення тесту	Натискаємо на кнопку «Отримати результат»

Продовження таблиці 5.8

Очікуваний результат	Виведення списку найкоротших шляхів та матриці з довжинами цих шляхів.
Стан програми після проведення випробувань	Виведено відповідну матрицю та шляхи.

Таблиця 5.9 – Перевірка коректності роботи методу Данцига для графу з від’ємними значеннями ребер.

Мета тесту	Перевірити роботу алгоритму Данцига для графу з від’ємнихми значеннями ребер.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм.
Вхідні дані	Матриця графу з від’ємними значеннями.
Схема проведення тесту	Натискаємо на кнопку «Отримати результат»
Очікуваний результат	Виведення списку найкоротших шляхів та матриці з довжинами цих шляхів.
Стан програми після проведення випробувань	Виведено відповідну матрицю та шляхи.

Таблиця 5.10 – Перевірка правильності побудови графа.

Мета тесту	Перевірити правильність побудови графа.
Початковий стан програми	Відкрите вікно програми, створено та заповнено матрицю, обрано алгоритм.
Вхідні дані	Довільна матриця суміжності.
Схема проведення тесту	Натискаємо на кнопку «Отримати результат».
Очікуваний результат	Правильно побудований граф.
Стан програми після проведення випробувань	Правильно побудований граф.

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe, відкривається головне вікно програми (рисунок 6.1):

Рисунок 6.1 – Головне вікно програми

Далі потрібно створити простір для введення матриці, зазначивши її розмір у відповідному полі (користувач вводить певне число з клавіатури) (рисунок 6.2) та натиснувши кнопку (рисунок 6.3), що відповідає за саме створення простору (рисунок 6.4).

Рисунок 6.2 – Поле для введення розміру матриці

Рисунок 6.3 – Кнопка для створення простору для матриці

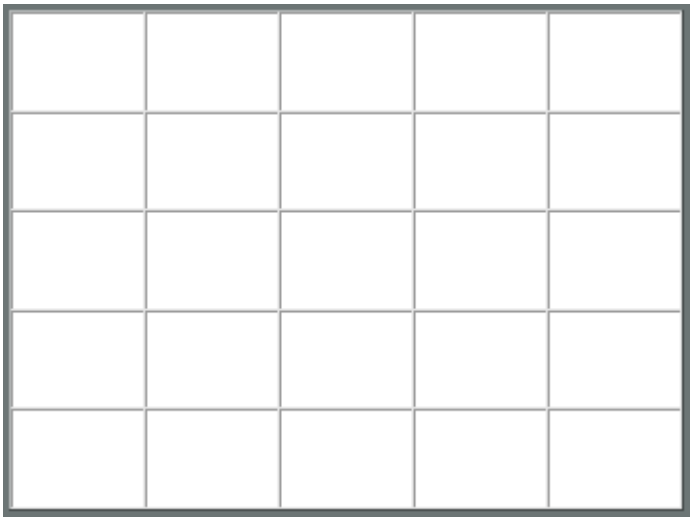


Рисунок 6.4 – Простір для вводу матриці суміжності

Після потрібно заповнити матрицю суміжності значеннями ваг ребер між ними. Це можна зробити двома способами: ручний ввід і автозаповнення. У першому випадку користувач заповнює матрицю з клавіатури, дотримуючись певних правил: усі елементи головної діагоналі дорівнюють 0, усі виділені для матриці суміжності поля мають бути заповнені числами або позначками нескінченності (для позначення нескінченності використовуються ключові слова Inf або inf). У другому випадку користувач натискає кнопку автозаповнення (Рисунок 6.5), і матриця заповнюється самостійно. Для підтвердження заповненої матриці необхідно натиснути відповідну кнопку (Рисунок 6.6).

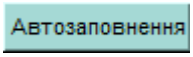


Рисунок 6.5 – Кнопка для автоматичного заповнення матриці

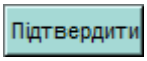


Рисунок 6.6 – Кнопка підтвердження введеної матриці

Обов’язковим кроком є вибір метода, який буде опрацьовувати матрицю. Вибір можна здійснити шляхом натискання потрібної кнопки (рисунок 6.7).

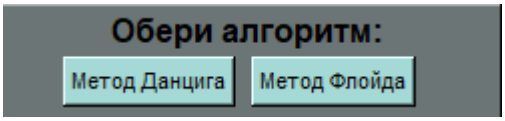


Рисунок 6.7 – Кнопка вибору метода

Останнім кроком є отримання результату, яке виконується шляхом натискання відповідної кнопки (рисунок 6.8), коли всі попередні кроки були зроблені. У підсумку відкривається нове вікно (рисунок 6.9), яке містить двя

поля для результатів: поле для графу (рисунок 6.10) та поле для текстових даних (рисунок 6.11).

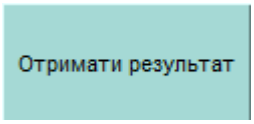


Рисунок 6.8 – Кнопка для отримання кінцевого результату.

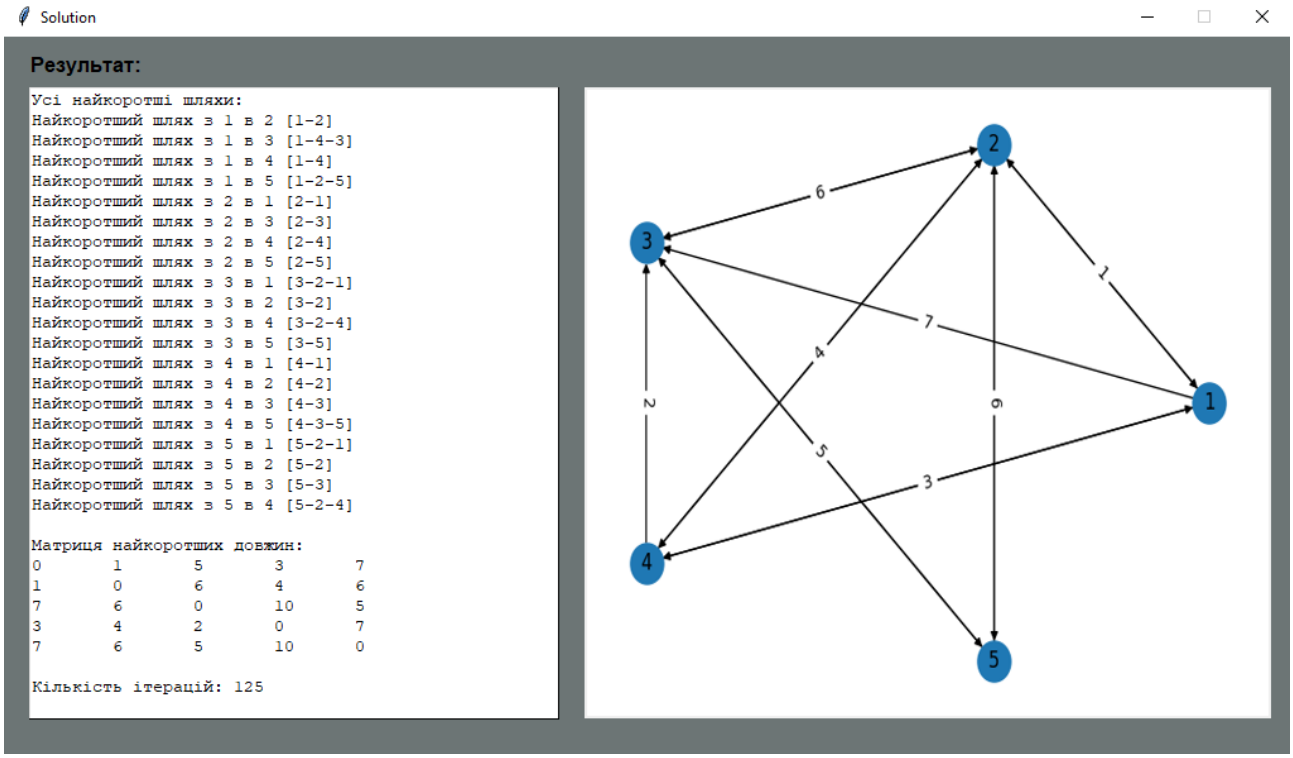


Рисунок 6.9 – Вікно з результатами

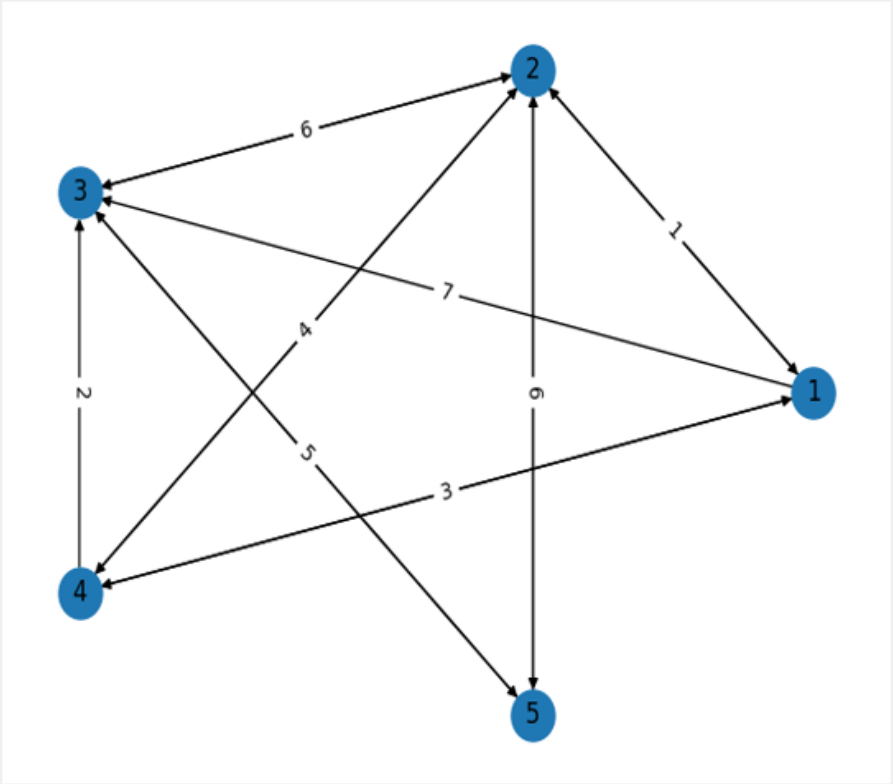


Рисунок 6.10 – Поле для графу

```
Усі найкоротші шляхи:
Найкоротший шлях з 1 в 2 [1-2]
Найкоротший шлях з 1 в 3 [1-4-3]
Найкоротший шлях з 1 в 4 [1-4]
Найкоротший шлях з 1 в 5 [1-2-5]
Найкоротший шлях з 2 в 1 [2-1]
Найкоротший шлях з 2 в 3 [2-3]
Найкоротший шлях з 2 в 4 [2-4]
Найкоротший шлях з 2 в 5 [2-5]
Найкоротший шлях з 3 в 1 [3-2-1]
Найкоротший шлях з 3 в 2 [3-2]
Найкоротший шлях з 3 в 4 [3-2-4]
Найкоротший шлях з 3 в 5 [3-5]
Найкоротший шлях з 4 в 1 [4-1]
Найкоротший шлях з 4 в 2 [4-2]
Найкоротший шлях з 4 в 3 [4-3]
Найкоротший шлях з 4 в 5 [4-3-5]
Найкоротший шлях з 5 в 1 [5-2-1]
Найкоротший шлях з 5 в 2 [5-2]
Найкоротший шлях з 5 в 3 [5-3]
Найкоротший шлях з 5 в 4 [5-2-4]

Матриця найкоротших довжин:
0       1       5       3       7
1       0       6       4       6
7       6       0       10      5
3       4       2       0       7
7       6       5       10      0

Кількість ітерацій: 125
```

Рисунок 6.11 – Поле для текстових результатів

Також існує можливість очищення всієї введеної інформації в головному вікні, для цього потрібно натиснути відповідну кнопку (рисунок 6.12).

Рисунок 6.12 – Очищення введених даних

6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подається розмір матриці суміжності, яка задає граф (кількість його вершин), значення матриці, які відповідають певним умовам та метод, який буде опрацьовувати дані.

Результатом виконання програми є візуалізований граф, список найкоротших шляхів між усіма вершинами графу, матриця найкоротших шляхів між усіма вершинами, кількість ітерацій виконання алгоритму.

6.3 Системні вимоги

Таблиця 6.1 – Системні вимоги

	Мінімальні	Рекомендовані
Операційна система	Windows 8/Windows 10/Windows 11 (з останніми обновленнями)	Windows 10/Windows 11 (з останніми обновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	2 GB RAM	4 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	1280x720	1920x1080
Прилади введення	Клавіатура, комп’ютерна миша	
Додаткове програмне забезпечення	Бібліотеки Pillow, MatplotLib, NetworkX, Tkinter	

7 АНАЛІЗ РЕЗУЛЬТАТІВ

Метою курсової роботи була реалізація програми для пошуку найкоротших шляхів мережі методами Флойда та Данцига.

Критичних ситуацій у роботі програми не було. Усі можливі помилки при некоректному вводі даних користувачем проходять валідацію, яка на дає їм бути опрацьованими програмою, якщо вони не відповідають потрібним умовам. Помилки при неможливості застосування алгоритма до матриць певних типів також усунено, шляхом постійної перевірки матриці на наявність цієї проблеми.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаємось сервісом GraphOnline.

Для початку створимо випадкову матрицю суміжності методом автозаповнення в нашій програмі та побудуємо граф за нею. (рисунок 7.1):

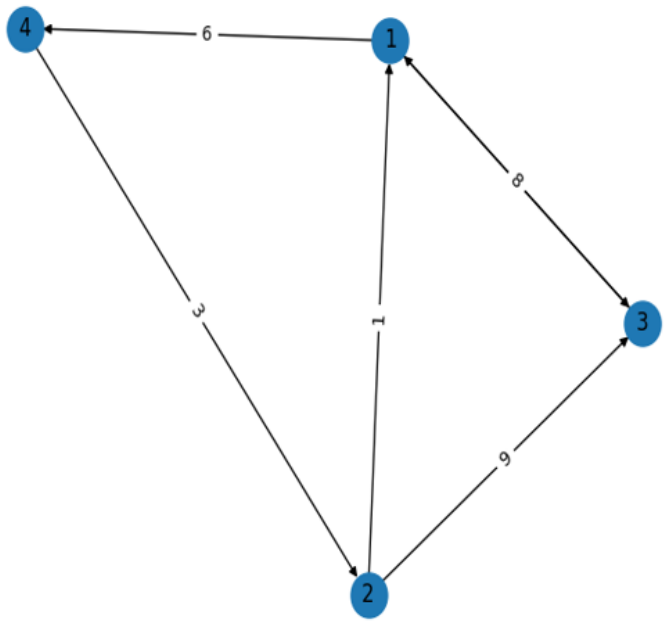


Рисунок 7.1 – Граф, побудований програмою.

Так само введемо матрицю в сервісі й отримаємо той же граф (рисунок 7.2).

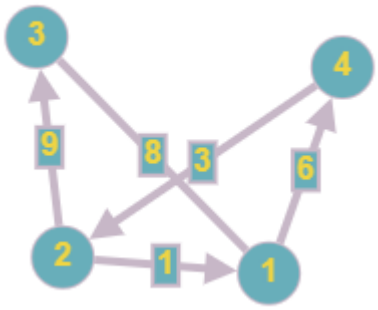


Рисунок 7.2 – Зображення графа у сервісі.

Після виконання метода Флойда порівняємо матрицю найкоротшив шляхів, що вивела програма (рисунок 7.3) та сервіс (рисунок 7.4).

```
Усі найкоротші шляхи:
Найкоротший шлях з 1 в 2 [1-4-2]
Найкоротший шлях з 1 в 3 [1-3]
Найкоротший шлях з 1 в 4 [1-4]
Найкоротший шлях з 2 в 1 [2-1]
Найкоротший шлях з 2 в 3 [2-3]
Найкоротший шлях з 2 в 4 [2-1-4]
Найкоротший шлях з 3 в 1 [3-1]
Найкоротший шлях з 3 в 2 [3-1-4-2]
Найкоротший шлях з 3 в 4 [3-1-4]
Найкоротший шлях з 4 в 1 [4-2-1]
Найкоротший шлях з 4 в 2 [4-2]
Найкоротший шлях з 4 в 3 [4-2-3]

Матриця найкоротших довжин:
0       9       8       6
1       0       9       7
8       17      0       14
4       3       12      0

Кількість ітерацій: 64
```

Рисунок 7.3 – Результат виконання програмою.

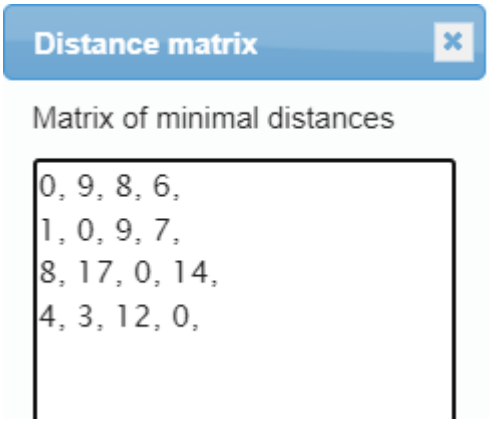


Рисунок 7.4 – Результат виконання сервісом.

Тією ж самою матрицею перевіримо метод Данцига (рисунок 7.5):

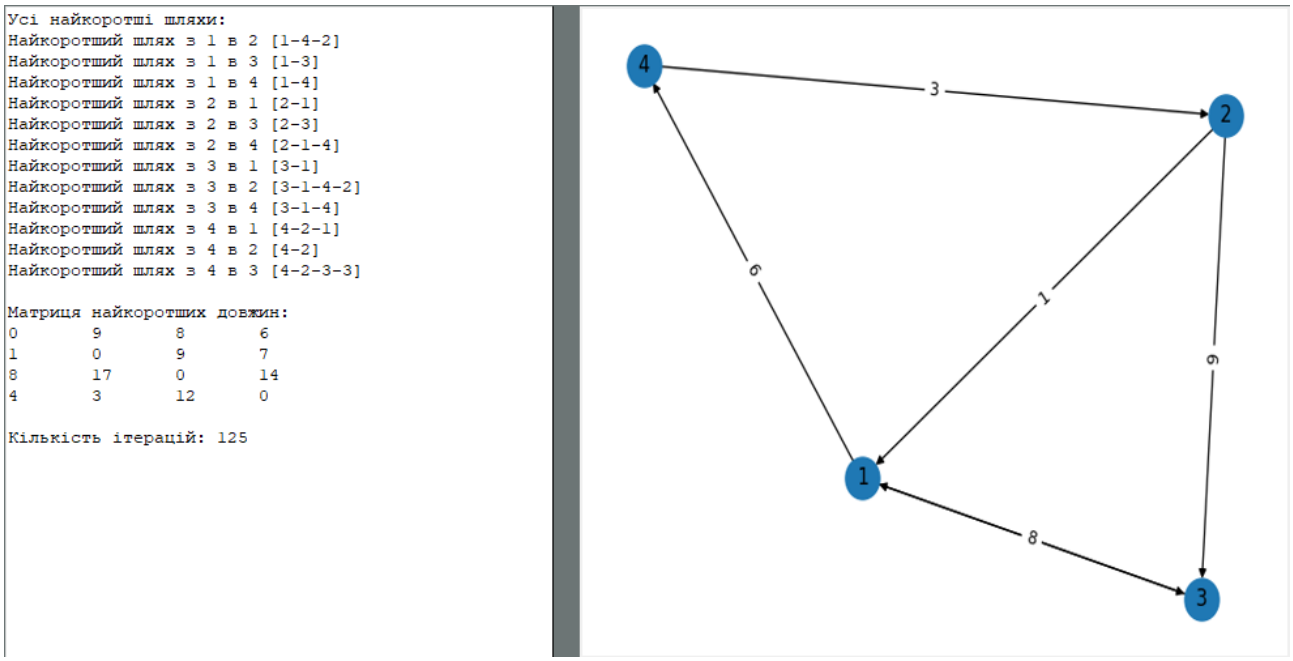


Рисунок 7.5 – Результат виконання програми методом Данцига.

Можемо побачити, що відповіді повністю однакові, отже програма працездатна.

Проведемо тестування ефективності алгоритмів за допомогою графу з матрицею суміжності виду:

$$G = \begin{matrix} 0 & d_{i,j} & \dots \\ \infty & 0 & \dots \\ d_{i,j} & \dots & 0 \end{matrix}$$

де $d_{i,j}$ – це вага дуги між вершинами і та j;

∞ – відсутність ребра;

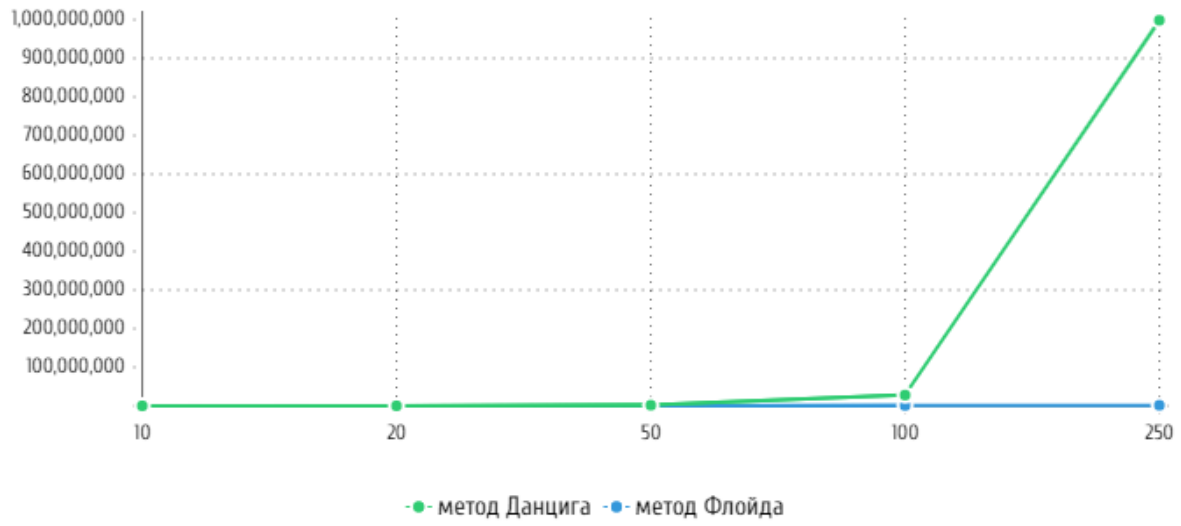
0 – величина петлі.

Результати тестування ефективності алгоритмів пошуку найкоротших шляхів у графі наведено у таблиці 7.1:

Таблиця 7.1 – Тестування ефективності алгоритмів.

Розмірність матриці суміжності	Параметри тестування	Метод	
		Флойда	Данцига
10	Кількість ітерацій	1000	3592
20	Кількість ітерацій	8000	52629
50	Кількість ітерацій	125000	1759101
100	Кількість ітерацій	1000000	27918298
250	Кількість ітерацій	15625000	998522642

Наглядне порівняння кількості ітерацій у двох алгоритмах зображено на діаграмі 7.1.



Діаграма 7.1 – Порівняння результатів тестування алгоритмів.

За результатом тестування можна зробити такі висновки:

1) Обидва алгоритми впорюються з вирішенням задачі знаходження найкоротших шляхів у мережі;
2) Математична складність алгоритму Флойда – $O(k * n^3)$;

3) Математична складність алгоритму Данцига – $O(k*n^3)$, на практиці ми мали велику кількість ітерацій.
4) Алгоритм Флойда є більш зручним та ефективним за рахунок меншого обсягу ітерацій та використаної пам’яті.

ВИСНОВОК

Курсова робота була присвячена дослідженню алгоритмів пошуку найкоротших шляхів у мережі.

У ході роботи було сформовано мету та описано алгоритм її досягнення. Описано теоретичні відомості, необхідні для вирішення даної задачі. Описано загальний алгоритм програми та алгоритми Флойда і Данцига з їх допоміжними функціями. Було описано класи програми, зображено діаграму класів та надано короткий опис усіх методів. Далі було розроблено та виконано тестування готового програмного забезпечення з метою перевірки коректності роботи. Додано інструкцію користувача з рисунками та описом дій для коректної роботи з програмою. Наприкінці було проведено аналіз результатів, під час якого було визначено основні результати роботи та надано оцінку роботи обох алгоритмів.

Програма повністю працездатна та виконує весь функціонал правильно.

ПЕРЕЛІК ПОСИЛАНЬ

1. Білоус Н. В., Бондаренко М. Ф., Руткас А. Г. Комп'ютерна дискретна математика. Харків : Компанія СМІТ, 2004. 299 с.
2. Майника Е. Алгоритми оптимізації на мережах та графах. Нью-Йорк, 1978. 53 с.
3. Майника Е. Алгоритми оптимізації на мережах та графах. Нью-Йорк, 1978. 58 с.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив
Керівник Головченко М. М.
«05» травня 2022 р.

Виконавець:
Студент Бондаренко М. В.
«05» травня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Пошук найкоротших шляхів в мережі»

з дисципліни:

«Об'єктно-орієнтоване програмування»

Київ 2022

1. *Мета:* Метою курсової роботи є розробка ефективного програмного забезпечення для розв'язання задач пошуку найкоротших шляхів у мережі за допомогою певних методів.
2. *Дата початку роботи*: «15» березня 2022 р.
3. *Дата закінчення роботи*: «25» серпня 2022 р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      - Можливість задання розміру мережі до 15 вершин.
      - Можливість задання мережі.
      - Можливість автоматичного заповнення даних.
      - Перевірка коректності введених даних.
      - Можливість обирати метод пошуку найкоротших шляхів (метод Флойда або Данцига).
      - Можливість зображення графу за допомогою графічного інтерфейсу.
      - Збереження та виведення всіх статистичних даних роботи алгоритмів пошуку.

   2) Нефункціональні вимоги:
      – Можливість запуску ПЗ на ОС Windows 10.
      – Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:
        ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.
        ГОСТ 19.106 - 78 - Вимоги до програмної документації.
        ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації
5. *Стадії та етапи розробки*:
   1) Об'єктно-орієнтований аналіз предметної області задачі (до 25.06.2022 р.)

2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 26.06.2022р.)
3) Розробка програмного забезпечення (до 24.08.2022 р.)
4) Тестування розробленої програми (до 24.08.2022р.)
5) Розробка пояснювальної записки (до 25.08.2022 р.).
6) Захист курсової роботи (до 26.08.2022 р.).

6. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду програмного забезпечення
Пошуку найкоротших шляхів в мережі

(Найменування програми (документа))

Електронний носій
(Вид носія даних)

16 арк,20 Кб
(Обсяг програми (документа), арк., Кб)

студента групи ІП-13 І курсу
Бондаренко М.В.

main.py

```
from MainWindow import *

win = MainWindow()
win.mainloop()
```

MainWindow.py

```
import random
from tkinter import messagebox
from Graph import *
from SolutionWindow import *
import math


class MainWindow(Tk):
    def __init__(self):
        super().__init__()

        self.__graph = Graph()              # створюємо об'єкт графу, щоб взаємодіяти в меню з ним
        self.__matrix = None                # матриця для збереження даних, що вводить користувач
        self.__floyd_IsEnabled = False      # перевірка кнопки активації методу Флойда
        self.__dantzig_IsEnabled = False    # перевірка кнопки активації методу Данцига
        self.__confirm_IsEnabled = False    # перевірка кнопки підтвердження

        # БАЗОВІ ПАРАМЕТРИ ІНТЕРФЕЙСУ
        self.label_font = ("Mazzard Soft L", 12, 'bold')
```

```
        self.button_font = ("Mazzard Soft M", 8, 'normal')
        self.main_color = '#6c7575'
        self.button_color = '#a5d9d5'
        self.label_color = 'black'

        self.title("Main window")
        self.geometry("375x400+448+240")
        self.resizable(False, False)

        # ПРОСТІР ОСНОВНОГО ВІКНА
        self.frame = Frame(self, bg=self.main_color)
        self.frame.place(x=0, y=0, width=375, height=400)

        # ПРОСТІР ДЛЯ КНОПОК ВИБОРУ АЛГОРИТМА
        self.frame_method = Frame(self.frame, bg=self.main_color, relief='raised',
bd=1)
        self.frame_method.place(x=0, y=340, width=250, height=60)

        self.label_choose = Label(self.frame_method, text='Обери алгоритм: ',
font=self.label_font, bg=self.main_color,
                                  anchor='n')
        self.label_choose.pack()

        self.button_dantzig = Button(self.frame_method, text='Метод Данцига',
font=self.button_font,
                                     bg=self.button_color,
command=self.button_dantzig_method_click)
        self.button_dantzig.place(relx=0.12, rely=0.425)

        self.button_floyd = Button(self.frame_method, text='Метод Флойда',
font=self.button_font, bg=self.button_color,
```

```
                                command=self.button_floyd_method_click)
        self.button_floyd.place(relx=0.5, rely=0.425)

        # ПРОСТІР ДЛЯ КНОПОК РОБОТИ З МАТРИЦЕЮ
        self.frame_matrix_space = Frame(self.frame, bg=self.main_color, relief='raised')
        self.frame_matrix_space.place(x=0, y=0, width=375, height=340)

        self.label_matrix_size = Label(self.frame_matrix_space, text='Розмір матриці:
', font=self.label_font,
                                       bg=self.main_color)
        self.label_matrix_size.place(relx=0.05, rely=0.04)

        self.entry_matrix_size = Entry(self.frame_matrix_space, width=2,
font=self.label_font)
        self.entry_matrix_size.place(relx=0.4, rely=0.04)

        self.button_create_matrix = Button(self.frame_matrix_space, text='Створити
матрицю', font=self.button_font,
                                           bg=self.button_color,
command=self.button_create_matrix_click)
        self.button_create_matrix.place(relx=0.475, rely=0.04, relwidth=0.275)

        self.button_confirm_matrix = Button(self.frame_matrix_space,
text='Підтвердити', font=self.button_font,
                                            bg=self.button_color, height=1,
                                            command=self.button_confirm_matrix_click)
        self.button_confirm_matrix.place(relx=0.765, rely=0.04, relwidth=0.185)

        self.button_autofill = Button(self.frame_matrix_space, text='Автозаповнення',
font=self.button_font,
                                      bg=self.button_color, height=1,
```

```
command=self.button_autofill_click)
        self.button_autofill.place(relx=0.05, rely=0.9, relwidth=0.25)

        self.button_clear_matrix = Button(self.frame_matrix_space, text='Очистити',
font=self.button_font,
                                          bg=self.button_color, height=1,
command=self.button_clear_matrix_click)
        self.button_clear_matrix.place(relx=0.75, rely=0.9, relwidth=0.2)

        # ПРОСТІР ДЛЯ МАТРИЦІ
        self.frame_matrix = Frame(self.frame_matrix_space, bg=self.main_color,
relief='ridge', bd=1)
        self.frame_matrix.place(relx=0.05, rely=0.14, height=250, relwidth=0.9)

        # КНОПКА ОТРИМАННЯ РЕЗУЛЬТАТУ
        self.button_get_result = Button(self.frame, text='Отримати результат',
font=self.button_font, bg=self.button_color,
                                        command=self.button_get_result_click)
        self.button_get_result.place(x=250, y=340, height=60, width=125)

    @property
    def graph(self):
        return self.__graph

    @graph.setter
    def graph(self, graph):
        self.__graph = graph

    @property
    def matrix(self):
        return self.__matrix
```

```
    @matrix.setter
    def matrix(self, matrix):
        self.__matrix = matrix

    # КНОПКА ДЛЯ ОТРИМАННЯ РЕЗУЛЬТАТУ ТА ЙОГО ВИВЕДЕННЯ
    def button_get_result_click(self):
        if self.__confirm_IsEnabled == False:
            messagebox.showwarning(title='Попередження', message='Підтвердіть створену матрицю')
            return
        self.__graph.drawGraph()
        if self.__floyd_IsEnabled == True:
            self.__graph.floydMethod()
            try:
                self.__graph.floydMethod()
            except:
                messagebox.showwarning(title='Попередження', message='Знайдений негативний контур. Введіть нову матрицю')
                return
        elif self.__dantzig_IsEnabled == True:
            self.__graph.dantzigMethod()
            try:
                self.__graph.dantzigMethod()
            except:
                messagebox.showwarning(title='Попередження', message='Знайдений негативний контур. Введіть нову матрицю')
                return
        else:
            messagebox.showwarning(title="Попередження", message="Оберіть метод розв'язання")
```

```
            return

        solution_window = SolutionWindow()
        solution_window.mainloop(1)
        with open("data.txt", 'r', encoding='utf-8') as file:
            text = file.read()
        solution_window.text_result.insert(1.0, text)

        self.__floyd_IsEnabled = False
        self.__dantzig_IsEnabled = False
        self.__confirm_IsEnabled = False

    # СТВОРЕННЯ ПРОСТОРУ ДЛЯ МАТРИЦІ
    def button_create_matrix_click(self):
        size = self.entry_matrix_size.get()
        if not size.isdigit() or int(size) < 1 or int(size) > 15:
            messagebox.showwarning(title="Попередження", message="Введіть
коректний розмір (1-15) ")
            self.entry_matrix_size.delete(0, END)
            return

        self.__graph.size = int(size)
        self.__matrix = [[None for x in range(self.__graph.size)] for y in
range(self.__graph.size)]

        for i in range(self.__graph.size):
            for j in range(self.__graph.size):
                entry = Entry(self.frame_matrix, bg='white', font=50, justify='center')
                entry.place(relx=j / self.__graph.size, rely=i / self.__graph.size, relwidth=1
/ self.__graph.size,
                            relheight=1 / self.__graph.size)
```

```
                self.__matrix[i][j] = entry
        self.__confirm_IsEnabled = False

    # ПІДТВЕРДЖЕННЯ СТВОРЕНОЇ МАТРИЦІ ТА ВАЛІДАЦІЯ ДАНИХ
    def button_confirm_matrix_click(self):
        if self.__matrix == None:
            messagebox.showwarning(title='Попередження', message='Спочатку створіть матрицю')
            return

        self.__graph.matrix = [[None for x in range(self.__graph.size)] for y in range(self.__graph.size)]
        for i in range(self.__graph.size):
            for j in range(self.__graph.size):
                element = self.__matrix[i][j].get()
                if i == j:
                    if element != '0':
                        messagebox.showwarning(title='Попередження',
                                               message='Основна діагональ може складатися лише з 0')
                        return 0
                    else:
                        self.__graph.matrix[i][j] = int(element)
                else:
                    if element == 'Inf' or element == 'inf':
                        self.__graph.matrix[i][j] = math.inf
                    elif element != '' and element[0] == '-' and element[1:].isdigit():
                        self.__graph.matrix[i][j] = int(element)
                    elif element == '' or not element.isdigit():
                        messagebox.showwarning(title='Попередження', message="Введіть коректні значення (Inf або ціле число)")
```

```
                    return
                else:
                    self.__graph.matrix[i][j] = int(element)
        messagebox.showinfo(title='Інфо', message='Матриця успішно створена')
        self.__confirm_IsEnabled = True

    # АВТОЗАПОВНЕННЯ ВИПАДКОВИМИ ЗНАЧЕННЯМИ
    def button_autofill_click(self):
        if self.__matrix == None:
            messagebox.showwarning(title='Попередження', message='Спочатку
створіть матрицю')
            return

        list_elements = [x for x in range(1, 10)]

        for j in range(0, self.__graph.size):
            for i in range(j, self.__graph.size):
                random_value = []
                entry = self.__matrix[i][j]
                entry.delete(0, END)
                if i == j:
                    entry.insert(0, 0)
                else:
                    random_value.append(random.choice(list_elements))
                    random_value.append('Inf')
                    entry.insert(0, random.choice(random_value))
                    entry = self.__matrix[j][i]
                    entry.delete(0, END)
                    entry.insert(0, random.choice(random_value))

    # ОЧИСТКА ВСІХ УВЕДЕНИХ ДАННИХ
```

```
    def button_clear_matrix_click(self):
        self.entry_matrix_size.delete(0, END)

        self.__matrix = None
        self.__confirm_IsEnabled = False

        self.frame_matrix.destroy()
        self.frame_matrix = Frame(self.frame_matrix_space, bg=self.main_color,
relief='ridge', borderwidth=1)
        self.frame_matrix.place(relx=0.05, rely=0.14, height=250, relwidth=0.9)

    # ПЕРЕМИКАЧ КНОПОК АКТИВАЦІЇ МЕТОДІВ
    def button_dantzig_method_click(self):
        self.__floyd_IsEnabled = False
        self.__dantzig_IsEnabled = True
        return

    # ПЕРЕМИКАЧ КНОПОК АКТИВАЦІЇ МЕТОДІВ
    def button_floyd_method_click(self):
        self.__floyd_IsEnabled = True
        self.__dantzig_IsEnabled = False
        return
```

Graph.py

```
import math
import copy
import matplotlib.pyplot as plt
import networkx as nx
```

```
class Graph:

    def __init__(self, matrix=None, size=None):
        self.__matrix = matrix
        self.__cost = None
        self.__path = None
        self.__size = size
        self.__iteration_counter = 0

    @property
    def matrix(self):
        return self.__matrix

    @matrix.setter
    def matrix(self, matrix):
        self.__matrix = matrix

    @property
    def size(self):
        return self.__size

    @size.setter
    def size(self, size):
        self.__size = size

    # АЛГОРИТМ ФЛОЙДА
    def floydMethod(self):
        self.__cost = copy.deepcopy(self.__matrix)
        self.__path = [[x for x in range(self.__size)] for y in range(self.__size)]

        for k in range(self.__size):
```

```
            self.minPath(self.__size, k)

        self.writeResult()
        self.__iteration_counter = 0

    # АЛГОРИТМ ДАНЦИГА
    def dantzigMethod(self):
        self.__cost = copy.deepcopy(self.__matrix)
        self.__path = [[x for x in range(self.__size)] for y in range(self.__size)]

        for m in range (2, self.__size):
            for k in range(m):
                self.minPath(m, k)
            self.findMinMj(m)
            self.findMinIm(m)

        for k in range(self.__size):
            self.minPath(self.__size, k)

        self.writeResult()
        self.iterationCount = 0

    # ПОРІВНЯННА ШЛЯХІВ ДЛЯ ПОШУКУ НАЙКОРОТШОГО +
ПЕРЕВІРКА НА НЕГАТИВНИЙ КОНТУР
    def minPath(self, m, k):
        for i in range(m):
            for j in range(m):
                if self.__cost[i][k] + self.__cost[k][j] < self.__cost[i][j]:
                    self.__cost[i][j] = self.__cost[i][k] + self.__cost[k][j]
                    self.__path[i][j] = k
```

```
            self.__iteration_counter += 1

            if self.__cost[i][i] < 0:
                raise Exception

    # ПОШУК МІНІМАЛЬНОГО, КОЛИ j=m
    def findMinIm(self, m):
        for i in range(m):
            min = self.__cost[i][m]
            for k in range(m):
                if (i != k and self.__cost[i][k] + self.__cost[k][m] < min):
                    self.__path[i][m] = k
                    min = self.__cost[i][k] + self.__cost[k][m]
                self.__iteration_counter += 1
            self.__cost[i][m] = min

    # ПОШУК МІНІМАЛЬНОГО, КОЛИ i=m
    def findMinMj(self, m):
        for j in range(m):
            min = self.__cost[m][j]
            for k in range(m):
                if (j != k and self.__cost[m][k] + self.__cost[k][j] < min):
                    self.__path[m][j] = k
                    min = self.__cost[m][k] + self.__cost[k][j]
                self.__iteration_counter += 1
            self.__cost[m][j] = min

    # ПОБУДОВА НАЙКОРОТОШОГО ШЛЯХУ (ЧЕРЕЗ ЩО ВІН
ПРОХОДИТЬ)
    def getWay(self, start, end):
        self.way = ""
```

```
        if (self.__cost[start][end] == math.inf):
            self.way = "Неможливо знайти шлях"
            return self.way

        self.way = f'{start + 1}-'
        while (self.__path[start][end] != end):
            new_end = self.__path[start][end]
            while (self.__path[start][new_end] != new_end and
self.__cost[start][new_end] != math.inf):
                new_start = self.__path[start][new_end]
                self.way += f"{self.__path[start][new_end] + 1}-"
                start = new_start
            self.way += f'{self.__path[start][end] + 1}-'
            start = self.__path[self.__path[start][end]][end]
        self.way += f'{end + 1}'
        return self.way

    # ЗАПИС РЕЗУЛЬТАТІВ В ФАЙЛ
    def writeResult(self):
        text = "Усі найкоротші шляхи:\n"
        for i in range(self.__size):
            for j in range(self.__size):
                if i != j and self.__path[i][j] != -1:
                    text += f'Найкоротший шлях з {i + 1} в {j + 1} [{str(self.getWay(i,
j))}]\n'

        text += "\nМатриця найкоротших довжин:\n"
        for i in range(self.__size):
            for j in range(self.__size):
                text += str(self.__cost[i][j]) + '\t'
            text += '\n'
```

```
        text += f"\nКількість ітерацій: {self.__iteration_counter}"

        with open("data.txt", 'w', encoding='utf-8') as file:
            file.write(text)

    # ВИВЕДЕННЯ ГРАФУ
    def drawGraph(self):
        graph = nx.DiGraph()

        for i in range(self.__size):
            graph.add_node(i + 1)

        for i in range(self.__size):
            for j in range(self.__size):
                if self.__matrix[i][j] < math.inf and self.__matrix[i][j] != 0:
                    graph.add_edge(i + 1, j + 1, weight=self.__matrix[i][j])

        pos = nx.spring_layout(graph)
        nx.draw(graph, pos=pos, node_size=500, arrows=True, with_labels=True)
        labels = nx.get_edge_attributes(graph, 'weight')
        nx.draw_networkx_edge_labels(graph, pos, edge_labels=labels)
        plt.savefig('graph.png')
        plt.clf()
```

SolutionWindow.py

```
from tkinter import *
from PIL import Image, ImageTk

class SolutionWindow(Toplevel):
    def __init__(self):
```

```
        super().__init__()

        self.label_font = ("Mazzard Soft L", 12, 'bold')
        self.button_font = ("Mazzard Soft M", 8, 'normal')
        self.main_color = '#6c7575'
        self.button_color = '#a5d9d5'
        self.label_color = 'black'

        self.title("Solution")
        self.geometry("1020x570+850+175")
        self.resizable(False, False)

        self.frame_solution = Frame(self, bg=self.main_color)
        self.frame_solution.place(x=0, y=0, width=1020, height=570)

        # ІМПОРТ ФОТО ГРАФУ З ПК
        self.image = Image.open("graph.png")
        photo = self.image.resize((540, 495))
        self.image = ImageTk.PhotoImage(photo)
        self.label = Label(self.frame_solution, image=self.image)
        self.label.place(x=460, y=40)

        self.label_result = Label(self.frame_solution, text='Результат: ',
font=self.label_font, bg=self.main_color)
        self.label_result.place(x=18, y=10)
        self.text_result = Text(self.frame_solution, bg='white', relief='raised', bd=1)
        self.text_result.place(x=20, y=40, width=420, height=500)
```