

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 дисципліни «Основи програмування»

(назва дисципліни)

на тему: пошук найкоротшого шляху в графі

Студента першого курсу, групи ПП-13
Замкового Дмитра Володимировича
Спеціальності 121 «Інженерія програмного
забезпечення»

Керівник старший викладач Головченко М.М.
(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 "ПІЗ"

Курс 1 Група ПІ-13

Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Замкового Дмитра Володимировича

1. Тема роботи знаходження найкоротшого шляху в графі методами Дейкстри та Беллмана-Форда

2. Строк здачі студентом закінченої роботи _____

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 _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	02.02.2022	
2.	Підготовка ТЗ	14.02.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	21.02.2022	
4.	Розробка сценарію роботи програми	22.02.2022	
6.	Узгодження сценарію роботи програми з керівником	13.04.2022	
5.	Розробка (вибір) алгоритму рішення задачі	13.04.2022	
6.	Узгодження алгоритму з керівником	20.04.2022	
7.	Узгодження з керівником інтерфейсу користувача	20.04.2022	
8.	Розробка програмного забезпечення	01.06.2022	
9.	Налагодження розрахункової частини програми	07.06.2022	
10.	Розробка та налагодження інтерфейсної частини програми	31.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	----	
12.	Тестування програми	08.06.2022	
13.	Підготовка пояснювальної записки	08.06.2022	
14.	Здача курсової роботи на перевірку	12.06.2022	
15.	Захист курсової роботи	17.06.2022	

Студент _____

(підпис)

Керівник _____

(підпис)

_____ Муха І. П.

(прізвище, ім'я, по батькові)

"__" _____ 20__ р.

Анотація

Пояснювальна записка до курсової роботи: 61 сторінка, 17 рисунків, 23 таблиці, 4 посилання.

Об'єкт дослідження: задача пошуку найкоротшого шляху у графі.

Мета роботи: дослідження методів пошуку найкоротшого шляху у графі, створення програмного забезпечення для пошуку найкоротшого шляху у графі, що задається користувачем з початкової до кінцевої точки.

Вивчено метод параметричного програмування задач з параметром у цільовій функції та векторі обмежень. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв'язання кожної з них.

Виконана програмна реалізація алгоритму параметричного програмування ЗЛП з параметром у цільовій функції або векторі обмежень.

**ЗВАЖЕНИЙ ОРІЄНТОВАНИЙ ГРАФ, МЕТОД ДЕЙКСТРИ, МЕТОД
БЕЛЛМАНА-ФОРДА**

ЗМІСТ

ВСТУП.....	5
1 ПОСТАНОВКА ЗАДАЧІ.....	6
2 ТЕОРЕТИЧНІ ВІДОМОСТІ.....	7
2.1. Метод Дейкстри.....	7
2.2. Метод Беллмана-Форда.....	7
3 ОПИС АЛГОРИТМІВ	8
3.1. Загальний алгоритм	8
3.2. Алгоритм методу Дейкстри	9
3.3. Алгоритм Беллмана-Форда	10
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	11
4.1. Діаграма класів програмного забезпечення	11
4.2. Опис методів частин програмного забезпечення.....	13
4.2.1. Користувацькі методи	13
4.2.2. Стандартні методи.....	19
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	23
5.1. План тестування.....	23
5.2. Приклади тестування.....	24
6 ІНСТРУКЦІЯ КОРИСТУВАЧА	33
6.1. Робота з програмою	33
6.2. Формат вхідних та вихідних даних	39
6.3. Системні вимоги	39
7 АНАЛІЗ РЕЗУЛЬТАТІВ.....	40
ВИСНОВКИ	46
ПЕРЕЛІК ПОСИЛАНЬ	47
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ	48
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ	51

ВСТУП

За деяких обставин у людей з'являється потреба знайти найкоротший шлях у зваженому графі. Наприклад, для вирішення задач з дискретної математики.

Зазвичай, знаходження найкоротшого шляху у графі – відносно проста задача, проте іноді бувають такі випадки, досить важко, або взагалі – неможливо.

Пошук найкоротшого шляху передбачає обходження усіх вершин графа, проте різні методи по-різному це роблять. Наприклад, метод Беллмана-Форда – підходить для графа з від'ємною вагою ребер, тому є більш універсальним. А метод Дейкстри підходить тільки для графа з додатною вагою ребер, проте він набагато ефективніший за метод Беллмана-Форда.

Моя програма дозволяє знаходити найкоротший шлях для графів з від'ємними або додатною вагою ребер двома способами на вибір: метод Дейкстри або метод Беллмана-Форда.

Програма поєднує в собі зручний інтерфейс із багатьма, а також швидке та надійне вирішення поставленої задачі.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити рішення для пошуку найкоротшого шляху наступними методами:

- а) метод Дейкстри;
- б) метод Беллмана-Форда

Вхідними даними для даної роботи є мережа G , з початковою та кінцевою точкою, задана матрицею вагів

$$G = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0j} \\ a_{10} & a_{11} & \cdots & a_{1j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i0} & a_{i1} & \cdots & a_{ij} \end{bmatrix}$$

де G – квадратна матриця, де рядки(стовпці) відповідають вершинам графа. Елементи матриці визначають вагу відповідних ребер. Програмне забезпечення повинно обробляти мережу, задану матрицею ваг в межах від 2 до 20 вершин.

Вихідними даними для даної роботи являється знайдений найкоротший шлях в заданій мережі від заданого початкової і кінцевої вершини. Програмне забезпечення повинно видавати розв’язок за умови, що всі дані задані коректно. Якщо це не так, то програма повинна вивести відповідне повідомлення. Якщо задача не має розв’язків, то програма повинна видати відповідне повідомлення.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Мережу можна задати таким способом:

$$G = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0j} \\ a_{01} & a_{11} & & a_{1j} \\ & \vdots & \ddots & \vdots \\ a_{i0} & a_{i1} & \cdots & a_{ij} \end{bmatrix}$$

Тоді якщо всі елементи – числа та граф не має від’ємних циклів то мережу G вважаємо заданою і має розв’язок.

2.1.Метод Дейкстри

Сутність метода Дейкстри полягає в пошуку в ширину з підрахунком довжини пройденого шляху. Метод Дейкстри має рішення при умові, що всі довжини ребр - додатні. Для всі вершин, окрім початкової, ставимо довжину $(-\infty)$, на початкову - ставимо 0. Вибираємо початкову вершину, як поточну. Рахуємо вагу всіх сусідніх вершин до початкової та запишемо їх значення. Помічаємо початкову вершину, як пройдену. Серед вершин, які не пройдені і мають довжину шляху, що не дорівнює ∞ шукаємо вершину з найменшим шляхом. Помічаємо її як поточну та робимо теж саме для нової поточної вершини, допоки номер поточної вершини не буде відповідати кінцевої вершини

2.2.Метод Беллмана-Форда

Метод Беллмана-Форда є схожим на метод Дейкстри, але розрахований також на ребра від’ємної довжини. Також відмінність полягає в тому, що позначки оновлюються також для вже перевірених вершин, та перевірені вершини, що включені до рішення, можуть перераховуватись та змінювати свою вагу. Також за методом Беллмана-Форда є розв’язки при умові, що граф не має від’ємних контурів. Алгоритм закінчується тоді, коли всі вершини графа включені до рішення та і ще одне проходження алгоритма не змінить рішення.

3 ОПИС АЛГОРИТМІВ

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

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
<i>matrix_val</i>	Матриця ваг графа
tops_num	В методі Дейкстри масив, в якому лежать номери сусідніх елементів
edge	В методі Беллмана-Форда масив списків у форматі (початок_ребра, кінець_ребра, вага_ребра), який показує всі ребра у цьому графі

3.1. Загальний алгоритм

3.1.1. ПОЧАТОК

3.1.2. Зчитати кількість вершин графа

3.1.3. Зчитати метод рішення графа.

3.1.4. Зчитати матрицю ваг:

3.1.4.1. Зчитати матрицю системи:

3.1.4.2. Цикл проходу по всіх рядках матриці системи (a_i – поточна строка):

3.1.4.2.1. Цикл проходу всіх стовпців матриці системи (a_{ij} – поточний елемент):

3.1.4.2.2. ЯКЩО поточний елемент матриці – вірно записане число, ТО записати його в відповідну комірку *matrix_val*. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 4.

3.1.5. Зчитати початкову вершину

3.1.6. Зчитати кінцеву вершину

- 3.1.7. ЯКЩО обраний метод Дейкстри, ТО обробити дані згідно алгоритму методу Дейкстри (пункт 1.2)
- 3.1.8. ЯКЩО обраний метод Беллмана-Форда, ТО обробити дані згідно алгоритму методу Беллмана-Форда (пункт 1.3)
- 3.1.9. Зчитати початкову точку для пошуку
- 3.1.10. Зчитати кінцеву точку для пошуку
- 3.1.11. КІНЕЦЬ

3.2. Алгоритм методу Дейкстри

3.2.1. ПОЧАТОК

3.2.2. Задати поточний елемент, як початковим

3.2.2.1. ЦИКЛ проходу по всіх вершинах графа:

3.2.2.2. ЯКЩО поточний елемент дорівнює кінцевому, ТО Закінчити метод (пункт 4)

3.2.2.2.1. Задати масив сусідніх вершин до поточної вершини tops_num

3.2.2.2.2. ЦИКЛ проходу по всіх елементах tops_num:

3.2.2.2.2.1. Вибірємо мінімальне значення між вагою уже існуючого шляху у перевіряемого елемента вершини та сумою ваги поточного елемента та вагою ребра між поточним і перевіряємим елементом

3.2.2.2.2.2. ЯКЩО вага перевіряємої вершини змінилась, ТО додаємо до шляху перевіряємої вершини поточний елемент

3.2.2.2.3. Помітити поточний елемент, як пройдений

3.2.2.2.4. Вибрати елемент з мінімальною вагою шляху серед непройдених

3.2.3. КІНЕЦЬ

3.3. Алгоритм Беллмана-Форда

3.3.1. ПОЧАТОК

3.3.2. Задати масив (edge) списків ребер графа за принципом (початок_ребра, кінець_ребра, вага_ребра)

3.3.3. ЦИКЛ проходу по вершинах графа – 1

3.3.3.1. ЦИКЛ проходу по масиву списків ребер графа (edge)

3.3.3.1.1. ЯКЩО вага шляху вершини початку ребра не дорівнює ∞ і сума ваги шляху вершини початку ребра та ваги ребра менше суми ваги шляху вершини кінця ребра, ТО задати вагу шляху вершини кінця ребра сумою ваги шляху вершини початку ребра та вагою ребра і задати шлях вершини кінця ребра шляхом вершини початку ребра та додати до нього вершину кінця ребра

3.3.3.2. ЦИКЛ проходу по масиву списків ребер графа (edge)

3.3.3.2.1. ЯКЩО вага шляху вершини початку ребра не дорівнює ∞ і сума ваги шляху вершини початку ребра та ваги ребра менше суми ваги шляху вершини кінця ребра, ТО задати вагу шляху вершини кінця ребра $-\infty$ і видалити шлях до цієї вершини

3.3.4. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення

Клас Main – головний клас програми. Створює в собі клас App, тому має відношення асоціації

Клас App призначений для малювання та отримання даних від користувача. Створює в собі об'єкт класу TGraph та передає йому вхідні дані від користувача, тому має відношення асоціації з ним. Його об'єкт створюється в класі Main, тому має відношення асоціації з ним.

Клас TGraph призначений для побудови та обчислення найкоротшого шляху. Має атрибути – масив від 2 до 20 об'єктів класу TTop та створює об'єкт класу Draw, тому має відношення асоціації з ним.

Клас TTop призначений для зберігання відомостей про вершину графа. Його об'єкт створюється в класі TGraph, тому має відношення асоціації з ним.

Клас Draw призначений для малювання графу. Його об'єкт створюється у класі TGraph, тому має відношення асоціації з ним.

Діаграма класів зображена на рисунку 4.1:

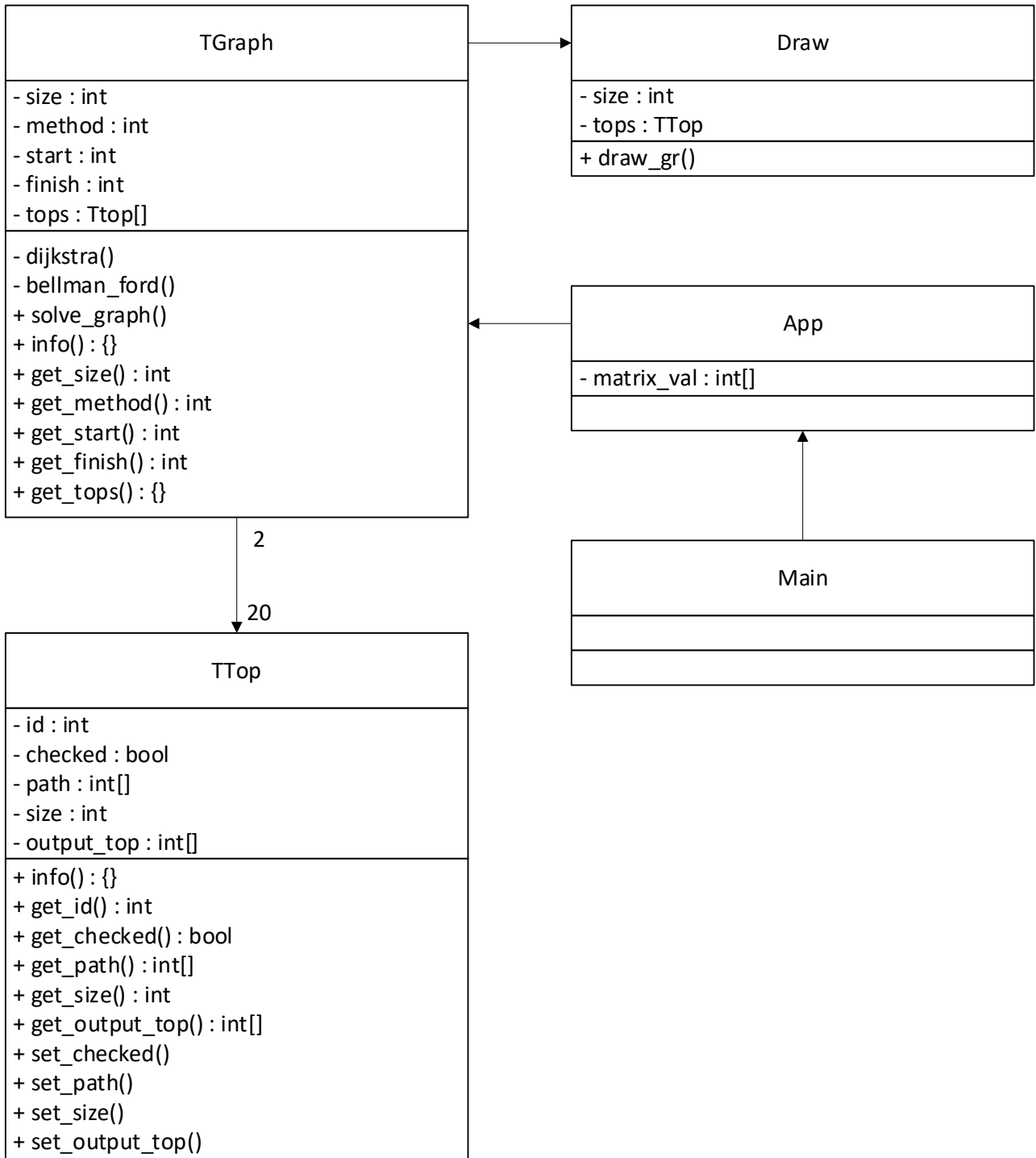


Рисунок 4.1 – Діаграма класів

4.2.Опис методів частин програмного забезпечення

4.2.1. Користувацькі методи

У таблиці 1.2 наведено перелік користувацьких методів використаних в курсовій роботі

Таблиця 1.1 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
1	App	gr_output	Виведення графа	---	---	view_menu
2	App	save_res	Збереження результату	---	---	view_menu
3	App	finish	Закінчення вводу даних	---	---	view_menu
4	App	start_finish_btn	Вивід повзунків «Початок» і «Кінець»	---	---	view_menu
5	App	check_btn_gen	Перевірка правильності введених кількості вершин і методу рішення	---	---	view_menu

Продовження таблиці 1.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
6	App	check_btn_set	Генерація верхньої форми	---	---	view_menu
7	App	top_deleted	Перевірка правильност і задання матриці ваг	---	---	view_menu
8	TGraph	dijkstra	Метод Дейкстри	---	---	Graph_class
9	TGraph	bellman_ford	Метод Беллмана- Форда	---	---	Graph_class
10	TGraph	solve_graph	Вирішити граф в залежності від методу	---	---	Graph_class
11	TGraph	info	Вивід інформації про граф	---	Масив даних	Graph_class

Продовження таблиці 1.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
12	TGraph	size	Вивід інформації про кількість вершин графа	---	Число – кількість вершин	Graph_class
13	TGraph	method	Вивід інформації про метод вирішення графа	---	Число - метод	Graph_class
14	TGraph	start	Вивід інформації про номер початкової вершини	---	Число – номер вершини	Graph_class
15	TGraph	finish	Вивід інформації про номер кінцевої вершини	---	Число – номер вершини	Graph_class

Продовження таблиці 1.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
16	TGraph	tops	Вивід інформації про вершини в графі	---	Масив об'єктів TTop	Graph_class
17	Draw	draw_gr	Генерація картинки графу	---	---	Graph_class
18	TTop	info	Вивід інформації про граф	---	Масив даних	Top_class
19	TTop	id	Вивід інформації про номер вершини	---	Число – номер вершини	Top_class
20	TTop	checked	Вивід інформації про те, чи вершина зафарбован а	---	True/False	Top_class

Продовження таблиці 1.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
21	ТТор	path	Вивід інформації про шлях до вершини	---	Масив – шлях	Top_class
22	ТТор	size	Вивід інформації про вагу шляху вершини	---	Число – вага шляху вершини	Top_class
23	ТТор	output_top	Вивід інформації про сусідні вершини до даної	---	Масив – номера сусідніх вершин	Top_class
24	ТТор	checked	Змінити значення про те, чи вершина зафарбована	Значення на яке потрібно змінити	---	Top_class
25	ТТор	path	Змінити значення шляху вершини	Значення на яке потрібно змінити	---	Top_class
26	ТТор	size	Змінити значення ваги шляху вершини	Значення на яке потрібно змінити	---	Top_class

Продовження таблиці 1.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Файл
27	TTop	output_top	Змінити значення сусідніх вершин	Значення на яке потрібно змінити	---	Top_class
28	---	minimum	Знайти індекс мінімальног о елемента	Масив в якому треба шукати	Число – індекс елементу	help_modul
29	---	save_res_in_file	Записати масив у файл	Масив, який потрібно записати	---	help_modul

4.2.2. Стандартні методи

У таблиці 1.1 наведено перелік стандартних методів всіх стандартних функцій використаних в курсовій роботі

Таблиця 1.2 – Стандартні методи

№ п/п	Назва класу	Назва функції	Призначенн я функції	Опис вхідних параметрів	Опис вихідних параметрі в	Пакет
1	Tk	Tk	Створити форму	---	Об'єкт Tk	tkinter
2	Tk	mainloop	Головний цикл форми	---	---	tkinter
3	Tk	title	Встановле ння заголовок форми	Текст	---	tkinter
4	Tk	geometry	Задання розміру і розташува ння форми	Рядок формату (WxH+X +Y)	---	tkinter
5	Tk	destroy	«Зламати» форму	---	---	tkinter
6	Tk	winfo_screenwidth	Отримати ширину екрана	---	Число – ширина екрану	tkinter
7	Tk	winfo_screenheight	Отримати висоту екрана	---	Число – довжина екрану	tkinter

Продовження таблиці 1.1

№ п/ п	Назва класу	Назва функції	Призначенн я функції	Опис вхідних параметрі в	Опис вихідних параметрів	Пакет
8	Tk	resizable	Задати можливість редагувати розміри форми користувач у	Висота та/або ширина дорівнює True/Fals е	---	tkinter
9	Toplevel	Toplevel	Створити верхню форму	Форма	Об'єкт Toplevel	tkinter
10	Toplevel	protocol	Відслідкову вати подію	Подія, функція	---	tkinter
11	Toplevel	destroy	«Зламати» верхню форму	Верхня форма	---	tkinter
12	Frame	Frame	Створити рамку	Форма	Об'єкт Frame	tkinter
13	Frame	place	Додати рамку на форму	х, у	---	tkinter
14	Frame	destroy	«Зламати» рамку	---	---	tkinter
15	Label	Label	Додати текстове поле	Форма/па мка	Об'єкт Label	tkinter

Продовження таблиці 1.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Пакет
16	Label	pack	Публікувати текстове поле	---	Об'єкт Label	tkinter
17	Label	place	Публікувати текстове поле	х, у	---	tkinter
18	Scale	Scale	Створити повзунок	Форма/рамка	Об'єкт Scale	tkinter
19	Scale	pack	Публікувати повзунок	---	---	tkinter
20	Entry	Entry	Створити поле для вводу даних	Форма/рамка	Об'єкт Entry	tkinter
21	Entry	get	Отримати значення	---	Рядок - значення	tkinter
22	Entry	grid	Публікувати поле для вводу даних	Колонка, рядок	---	tkinter
23	---	dumps	Перетворити об'єкт в json формат	Масив даних	Масив json	json
24	---	range	Створити об'єкт чисел	Початкове, кінцеве числа	Масив чисел	вбудований
25	---	len	Довжина елементу	Елемент	Число – довжину елемента	вбудований
26	---	str	Перетворити число у рядок	Число	Рядок	вбудований

Продовження таблиці 1.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Пакет
27	---	int	Перетворити рядок у ціле	Рядок	Число	вбудований
28	---	abs	Взяти число по модулю	Число	Модуль числа	вбудований
29	---	append	Додати елемент в кінець масиву	Елемент	---	вбудований
30	---	float	Перетворити рядок в число з плаваючою крапкою	Рядок	Число	вбудований

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. План тестування

При розробці ПЗ деякі моменти можуть упускатися чи просто про них можна забути. Тому дуже важливо провести тестування ПЗ, аби не було помилок, якого-небудь роду, на різні критичні моменти для своєчасного виправлення багів.

Для перевірки коректності роботи програми проведемо тестування за наступними критеріями

- а) Тестування правильності введення даних.
 - 1) Тестування при введенні недозволених символів в матрицю ваг
 - 2) Тестування при введенні від'ємних чисел в методі Дейкстри
 - 3) Тестування при введенні від'ємних чисел в методі Беллмана-Форда
 - 4) Тестування при не коректному виборі методу вирішення
 - 5) Тестування при незаповненні матриці ваг
 - 6) Тестування при частковому незаповненні матриці
- б) Тестування коректної роботи при деяких значеннях.
 - 1) Тестування роботи методу Дейкстри на графі з від'ємним контуром
 - 2) Тестування роботи методу Беллмана-Форда на графі з від'ємним контуром
 - 3) Тестування роботи методу Дейкстри при однаковому значенні початкової і кінцевої точки
 - 4) Тестування роботи методу Беллмана-Форда при однаковому значенні початкової і кінцевої точки
 - 5) Тестування роботи методу Дейкстри при неможливості дійти з початкової точки в кінцеву
 - 6) Тестування роботи методу Беллмана-Форда при неможливості дійти з початкової точки в кінцеву
- в) Тестування коректності роботи методів Дейкстри та Беллмана-Форда.
 - 1) Перевірка коректності роботи метода Дейкстри
 - 2) Перевірка коректності роботи метода Беллмана-Форда
- г) Тестування побудови графа

5.2. Приклади тестування

Проведемо по декілька тестувань для кожного пункту плану, результати запишемо до таблиць 5.1-5.15:

Таблиця 5.1 - Приклад роботи програми при введенні недозволених символів в матрицю ваг

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите вікно для вводу матриці ваг
Вхідні дані	и 7 4 5 4 7 2 0 3
Схема проведення тесту	Введення невалідних значення в матрицю ваг
Очікуваний результат	Повідомлення про помилку невалідності значень (підсвічення елемента з некоректними даними)
Стан програми після проведення випробувань	Підсвічено елемент матриці, який є невалідним червоним кольором

Таблиця 5.2 - Приклад роботи програми при введенні від'ємних чисел в методі Дейкстри

Мета тесту	Перевірити можливість введення від'ємних чисел у матрицю ваг в методі Дейкстри
Початковий стан програми	Відкрите вікно для вводу матриці ваг
Вхідні дані	$\begin{matrix} -5 & -7 & 4 \\ 3 & -4 & 5 \\ 2 & 2 & 3 \end{matrix}$
Схема проведення тесту	Введення від'ємних значень в матрицю ваг
Очікуваний результат	Програма приймає всі значення та бере їх по модулю
Стан програми після проведення випробувань	Програма взяла всі значення по модулю

Таблиця 5.3 - Приклад роботи програми при введенні від'ємних чисел в методі Беллмана-Форда

Мета тесту	Перевірити можливість введення від'ємних чисел у матрицю ваг в методі Беллмана-Форда
Початковий стан програми	Відкрите вікно для вводу матриці ваг
Вхідні дані	$\begin{matrix} -5 & -7 & 4 \\ 3 & -4 & 5 \\ 2 & 2 & 3 \end{matrix}$
Схема проведення тесту	Введення від'ємних значень в матрицю ваг
Очікуваний результат	Програма приймає всі значення без помилок
Стан програми після проведення випробувань	Програма прийняла всі значення

Таблиця 5.4 - Приклад роботи програми при не коректному виборі методу вирішення

Мета тесту	Перевірити можливість не вибирати метод вирішення задачі
Початковий стан програми	Відкрите вікно вибору метода
Вхідні дані	- (метод не вибраний)
Схема проведення тесту	Не вибирати метод рішення
Очікуваний результат	Повідомлення про помилку
Стан програми після проведення випробувань	Видано помилку «Виберіть метод»

Таблиця 5.5 - Приклад роботи програми при незаповнені матриці ваг

Мета тесту	Перевірити можливість не заповнювати матрицю ваг
Початковий стан програми	Відкрите вікно матриці ваг
Вхідні дані	(пуста матриця)
Схема проведення тесту	Не вводити дані в матрицю ваг
Очікуваний результат	Програма приймає значення
Стан програми після проведення випробувань	Програма прийняла значення та рахує, що ребер не існує

Таблиця 5.6 - Приклад роботи програми при частковому незаповненні матриці ваг

Мета тесту	Перевірити можливість частково не заповнювати матрицю ваг
Початковий стан програми	Відкрите вікно матриці ваг
Вхідні дані	$\begin{matrix} 5 & 1 \\ & 2 \\ 4 \end{matrix}$
Схема проведення тесту	Частково не вводити дані в матрицю ваг
Очікуваний результат	Програма приймає значення, а пусті клітинки заміняє на 0
Стан програми після проведення випробувань	Програма прийняла значення та рахує

Таблиця 5.7 - Приклад роботи програми при методі Дейкстри на графі з від'ємним контуром

Мета тесту	Перевірити можливість вирішення задачі методом Дейкстри з від'ємним контуром
Початковий стан програми	Відкрите вікно матриці ваг
Вхідні дані	$\begin{matrix} 0 & -1 & -3 \\ -2 & 0 & -1 \\ -4 & -2 & 0 \end{matrix}$
Схема проведення тесту	Введення графа через матрицю ваг з від'ємним контуром
Очікуваний результат	Заданий граф з усіма значеннями, взятими по модулю
Стан програми після проведення випробувань	Взяті значення в матриці ваг по модулю

Таблиця 5.8 - Приклад роботи програми при методі Беллмана-Форда на графі з від'ємним контуром

Мета тесту	Перевірити можливість вирішення задачі методом Беллмана-Форда з від'ємним контуром
Початковий стан програми	Відкрите вікно матриці ваг
Вхідні дані	$\begin{matrix} 0 & -1 & -3 \\ -2 & 0 & -1 \\ -4 & -2 & 0 \end{matrix}$
Схема проведення тесту	Введення графа через матрицю ваг з від'ємним контуром
Очікуваний результат	Виведення, як результат, шлях - (-), а довжину ($-\infty$)
Стан програми після проведення випробувань	Взяті значення в матриці ваг по модулю

Таблиця 5.9 - Приклад роботи програми методу Дейкстри при однаковому значенні початкової і кінцевої точки

Мета тесту	Перевірити можливість задавати однакову кінцеву і початкову точку в методі Дейкстри
Початковий стан програми	Очкування введення
Вхідні дані	3, 3
Схема проведення тесту	Вибрати однакову кінцеву і початкову точку
Очікуваний результат	Довжина – 0, Шлях – 3(номер вершини)
Стан програми після проведення випробувань	Результат: довжина – 0, шлях – 3(номер вершини)

Таблиця 5.10 - Приклад роботи програми методу Беллмана-Форда при однаковому значенні початкової і кінцевої точки

Мета тесту	Перевірити можливість задавати однакову кінцеву і початкову точку в методі Беллмана-Форда
Початковий стан програми	Очкування введення
Вхідні дані	3, 3
Схема проведення тесту	Вибрати однакову кінцеву і початкову точку
Очікуваний результат	Довжина – 0, Шлях – 3(номер вершини)
Стан програми після проведення випробувань	Результат: довжина – 0, шлях – 3(номер вершини)

Таблиця 5.11 - Приклад роботи програми при неможливості дійти з початкової точки в кінцеву методом Дейкстри

Мета тесту	Перевірити можливість задання параметрів, при яких не можливо дійти з початкової в кінцеву точки методом Дейкстри
Початковий стан програми	Очкування введення матриці ваг
Вхідні дані	$\begin{matrix} 0 & 0 & 3 \\ 0 & 0 & 2 \text{ початок} - 1, \text{ кінець} - 2 \\ 5 & 0 & 0 \end{matrix}$
Схема проведення тесту	Задати значення з неможливим розв'язком
Очікуваний результат	Довжина - ∞ , шлях - ∞
Стан програми після проведення випробувань	Результат: довжина – ∞ , шлях – ∞

Таблиця 5.12 - Приклад роботи програми при неможливості дійти з початкової точки в кінцеву методом Беллмана-Форда

Мета тесту	Перевірити можливість задання параметрів, при яких не можливо дійти з початкової в кінцеву точки методом Беллмана-Форда
Початковий стан програми	Очкування введення матриці ваг
Вхідні дані	0 0 3 0 0 2 початок – 1, кінець – 2 5 0 0
Схема проведення тесту	Задати значення з неможливим розв'язком
Очікуваний результат	Довжина - ∞ , шлях - ∞
Стан програми після проведення випробувань	Результат: довжина – ∞ , шлях – ∞

Таблиця 5.13 - Приклад роботи програми при заданих коректних даних для метода Дейкстри

Мета тесту	Перевірити роботу метода Дейкстри
Початковий стан програми	Очікування задання усіх даних
Вхідні дані	Вершини – 3, початок – 1, кінець – 2, метод – Дейкстри 5 4 0 2 0 3 0 2 -6
Схема проведення тесту	Введення коректних даних при методі Дейкстри
Очікуваний результат	Довжина – 4, шлях – (1 -> 2)
Стан програми після проведення випробувань	Результат: довжина – 4, шлях – (1 -> 2)

Таблиця 5.14 - Приклад роботи програми при заданих коректних даних для метода Беллмана-Форда

Мета тесту	Перевірити роботу метода Беллмана-Форда
Початковий стан програми	Очікування задання усіх даних
Вхідні дані	Вершини – 3, початок – 1, кінець – 2, метод – Беллмана-Форда <div style="text-align: center;"> 5 4 0 2 0 3 0 2 6 </div>
Схема проведення тесту	Введення коректних даних при методі Беллмана-Форда
Очікуваний результат	Довжина – 4, шлях – (1 -> 2)
Стан програми після проведення випробувань	Результат: довжина – 4, шлях – (1 -> 2)

Таблиця 5.15 - Приклад роботи програми введенні некоректних даних на побудову графа

Мета тесту	Перевірити правильність побудови графу
Початковий стан програми	Очікування введення даних у вікні
Вхідні дані	Вершини – 3, початок – 1, кінець – 2, метод – Беллмана-Форда <div style="text-align: center;"> 5 4 0 2 0 3 0 2 6 </div>
Схема проведення тесту	Задати вхідні дані і перевірити правильність побудови графіку
Очікуваний результат	правильно побудований граф (рисунок 5.1)
Стан програми після проведення випробувань	виведення графа (рисунок 5.2)

Рисунок 5.1 – правильно побудований граф

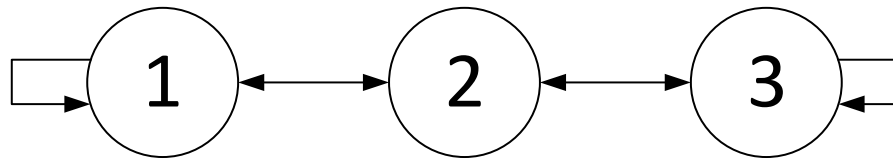
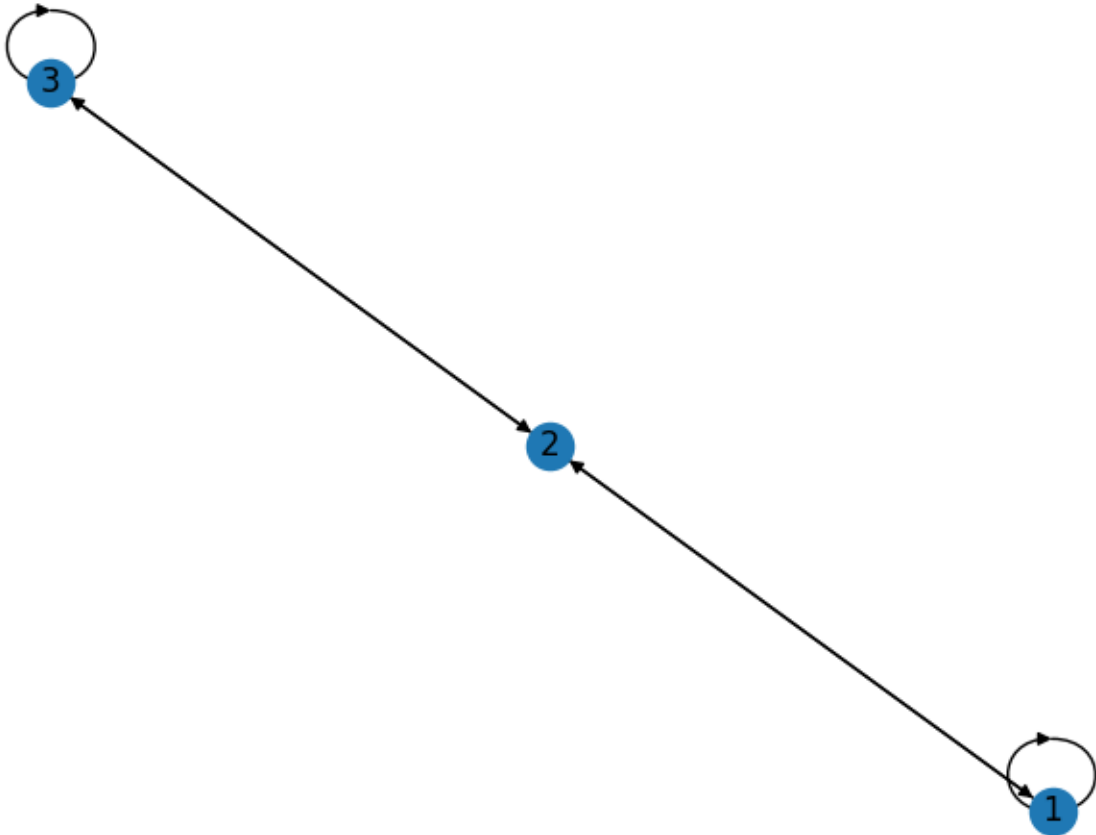


Рисунок 5.2 – граф, побудований програмою



Отже, в ході тестування ми перевірили різноманітні варіанти входу для програми та переконалися у її коректності роботи.

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1.Робота з програмою

6.1.1. Встановлення

Для запуску програми потрібно установити Python 3.7 з двома додатковими пакетами: `matplotlib`, `networkx`. Їх можна завантажити на офіційних сайтах: [Python](#), [matplotlib](#), [networkx](#). Для встановлення потрібно ввести у командний рядок два рядка коду:

```
pip install matplotlib
```

```
pip install networkx
```

Потрібно переконатися, що пакети додано до системних змінних.

Після цього завантажте код з [GitHub](#) на ваш комп'ютер

6.1.2. Запуск

Програму можна запустити подвійним натиском лівої кнопки миші по файлу `main.py`

6.1.3. Початок роботи

Після запуску програми відкривається головне вікно програми (Рисунок 6.1):

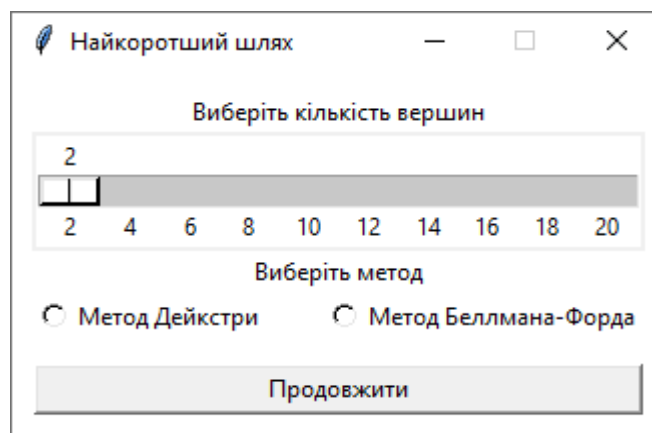


Рисунок 6.1 – Головне вікно програми

Далі за допомогою повзунка з текстом «Виберіть кількість вершин» шляхом пересування повзунка задати кількість вершин графа (рисунок 6.2):

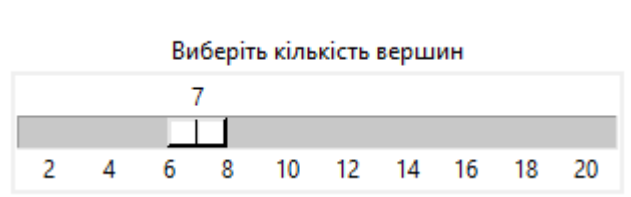


Рисунок 6.2 – Вибір кількості вершин графа

Потім вибираємо метод розв’язку на перемикачі з назвою «Виберіть метод» шляхом відмічення потрібного методу. Вибрати можна тільки один із двох запропонованих методів – Дейкстри і Беллмана-Форда (рисунок 6.3):

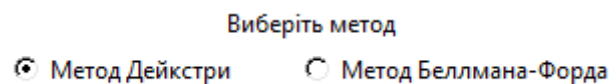


Рисунок 6.3 – Вибір методу розв’язку

Далі натискаємо кнопку продовжити. Якщо всі данні на цьому етапі було введено правильно, то відкриється наступна частина вводу даних та заблокуються попередні елементи для редагування (рисунок 6.4):

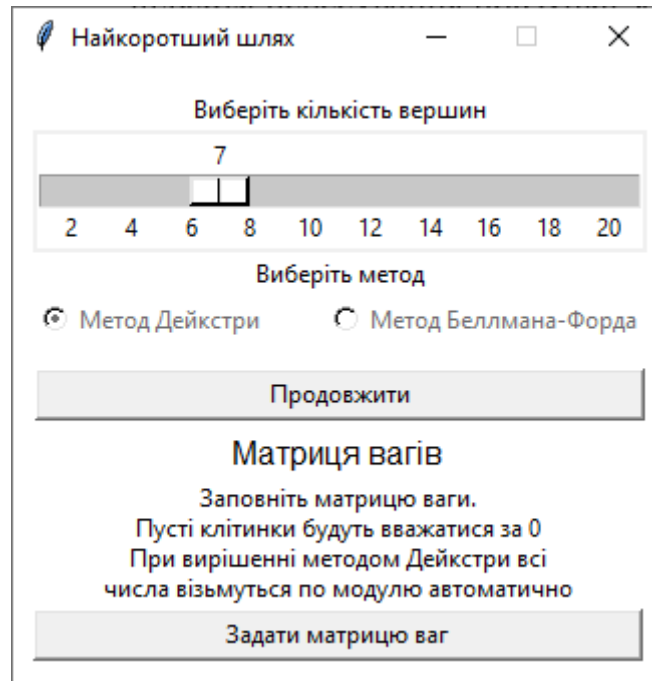


Рисунок 6.4 – Наступна частина вводу, попередні елементи заблоковано, тобто змінити їх вже не можливо

Після цього натискаємо на кнопку «Задати матрицю ваг». Відкриється таблиця для задання матриці ваг, яку потрібно заповнити. Для методу Дейкстри модуль всіх чисел береться автоматично. Всі пусті клітинки автоматично рахуються за 0. Для збереження матриці ваг потрібно закрити таблицю. При введенні некоректних даних таблиця для вводу матриці ваг не закривається і підсвітить некоректні елементи червоним кольором (рисунок 6.5). Їх потрібно буде виправити та закрити таблицю. При коректному вводі всі результати в матриці зберігаються автоматично та кнопка «Задати матрицю ваг» зміниться на кнопку «Продовжити» (рисунок 6.6):

1	2	5	4			
	y	-4	3			
4		9				
		6	7			
4		5				1
	2					
		45			3	

Рисунок 6.5 – Приклад вводу некоректних даних

Рисунок 6.6 – Зміна кнопки «Задати матрицю ваг» на кнопку «Продовжити»

Після натискання кнопки «Продовжити» потрібно вибрати початкову та кінцеву вершину графа за допомогою пересування повзунків з текстами «Виберіть початкову вершину» і «Виберіть кінцеву вершину» відповідно (рисунок 6.7):

Найкоротший шлях

Виберіть кількість вершин

7

2 4 6 8 10 12 14 16 18 20

Виберіть метод

☒ Метод Дейкстри ☐ Метод Беллмана-Форда

Продовжити

Матриця вагів

Заповніть матрицю ваг.
Пусті клітинки будуть вважатися за 0
При вирішенні методом Дейкстри всі
числа візьмуться по модулю автоматично

Продовжити

Виберіть початкову вершину

1

1 3 5 7

Виберіть кінцеву вершину

5

1 3 5 7

Порахувати

Рисунок 6.7 – Вибір початкової і кінцевої вершини

Далі потрібно натисну найбільшу кнопку «Порахувати». З’явиться зображення графу, найкоротший шлях від початкової до кінцевої вершини та його вага (рисунок 6.8):

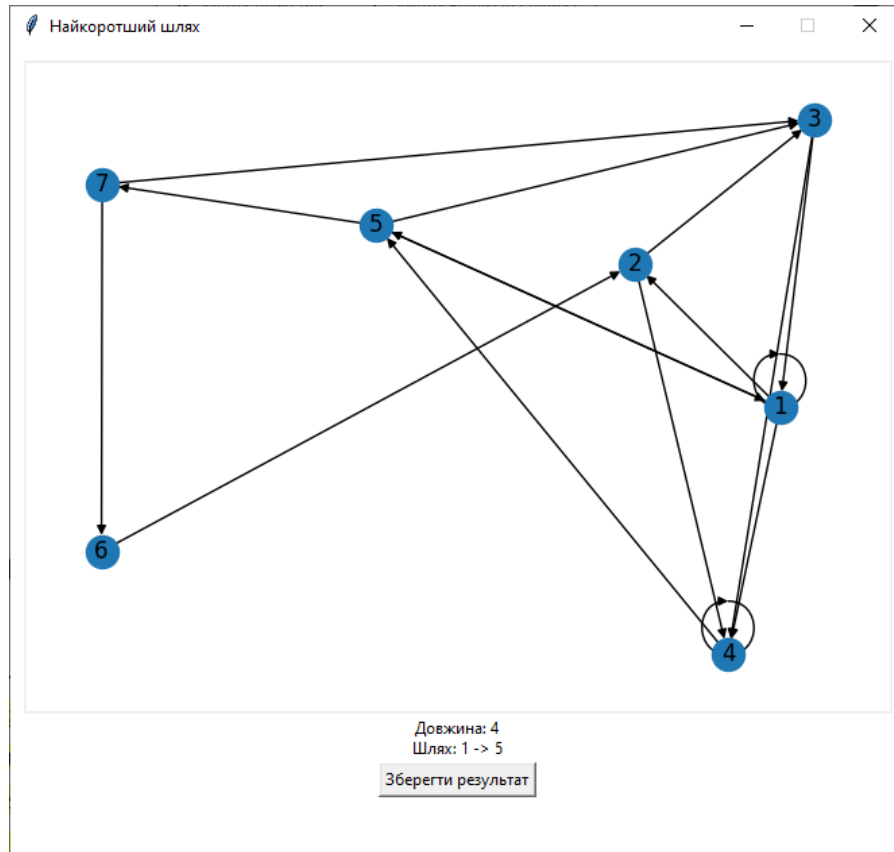


Рисунок 6.8 – Результат

Також є можливість зберегти результат у стандарті json у текстовий файл з іменем «res» за допомогою кнопки з назвою «Зберегти результат» (рисунок 6.9):

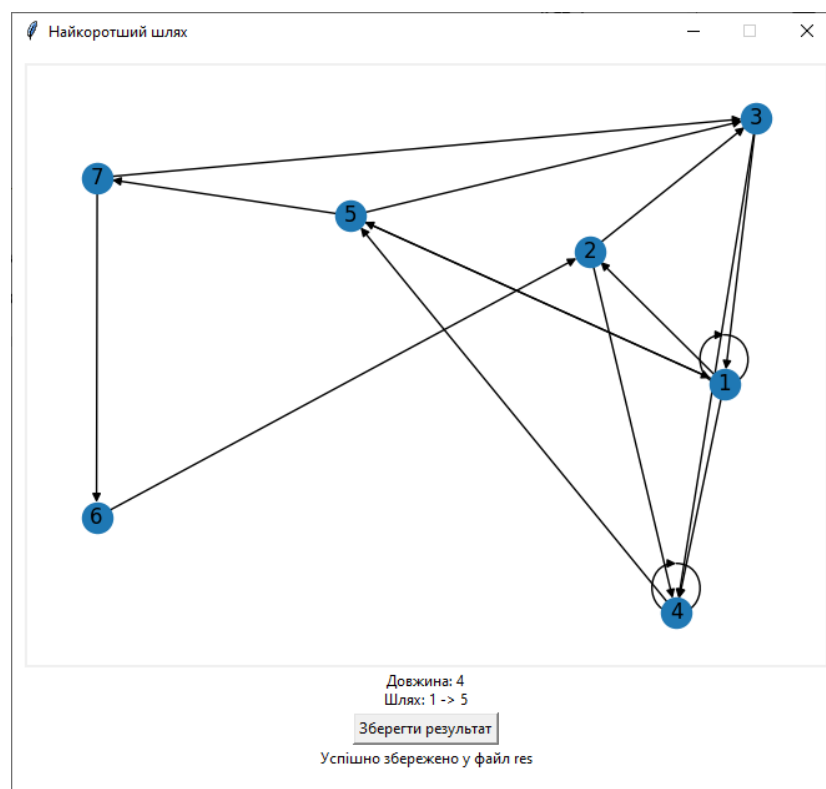
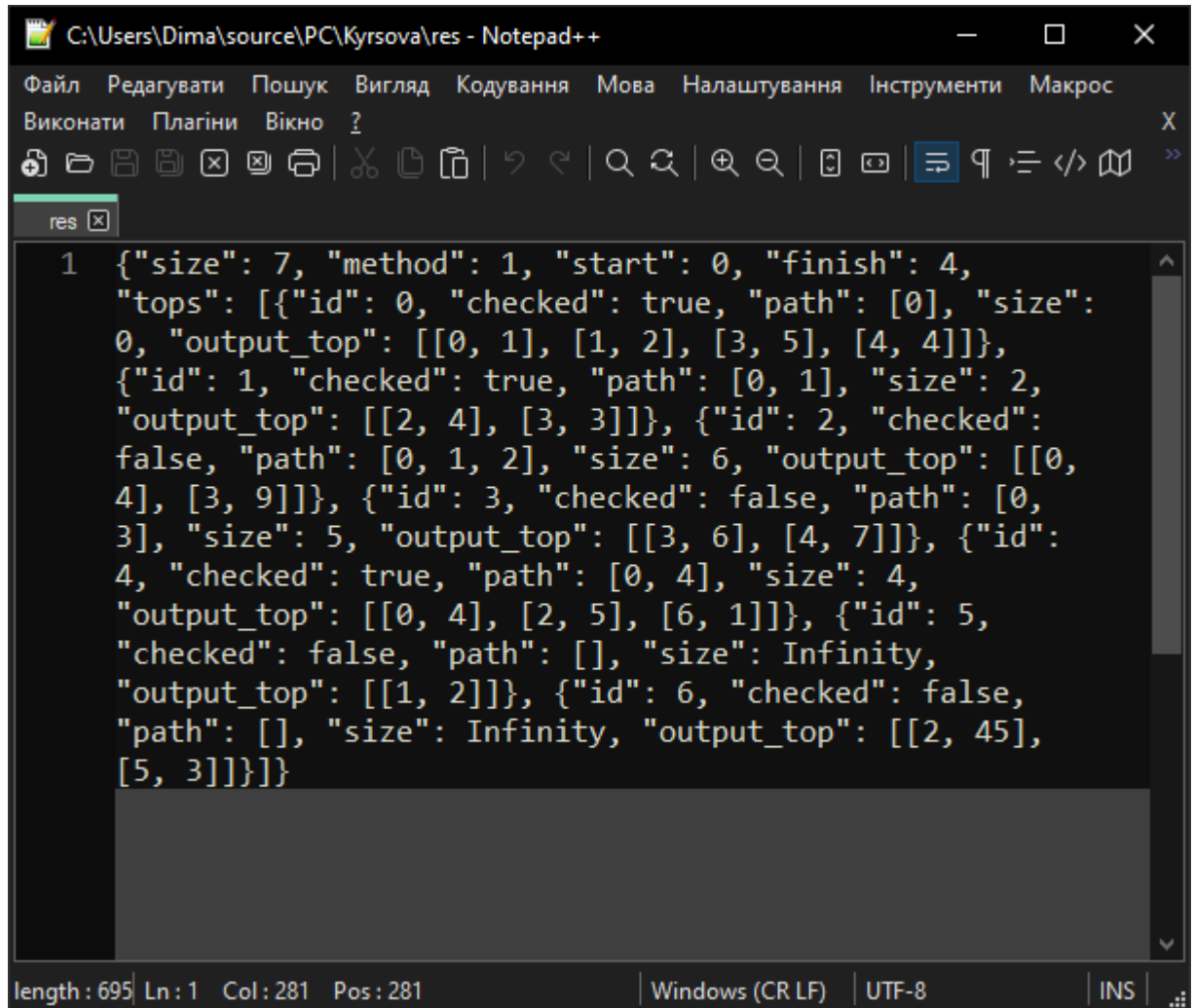


Рисунок 6.9 – Збереження результату у файл



```

1 {"size": 7, "method": 1, "start": 0, "finish": 4,
  "tops": [{"id": 0, "checked": true, "path": [0], "size":
0, "output_top": [[0, 1], [1, 2], [3, 5], [4, 4]]},
{"id": 1, "checked": true, "path": [0, 1], "size": 2,
"output_top": [[2, 4], [3, 3]]}, {"id": 2, "checked":
false, "path": [0, 1, 2], "size": 6, "output_top": [[0,
4], [3, 9]]}, {"id": 3, "checked": false, "path": [0,
3], "size": 5, "output_top": [[3, 6], [4, 7]]}, {"id":
4, "checked": true, "path": [0, 4], "size": 4,
"output_top": [[0, 4], [2, 5], [6, 1]]}, {"id": 5,
"checked": false, "path": [], "size": Infinity,
"output_top": [[1, 2]]}, {"id": 6, "checked": false,
"path": [], "size": Infinity, "output_top": [[2, 45],
[5, 3]]}]]}

```

length: 695 | Ln: 1 | Col: 281 | Pos: 281 | Windows (CR LF) | UTF-8 | INS

Рисунок 6.10 – файл «res»

6.2. Формат вхідних та вихідних даних

Користувачем на вхід програми подається кількість вершин графа, метод вирішення, граф у матричному вигляді, тобто через матрицю ваг, початкову та кінцеву точку для рішення

Результатом виконання програми є знайдений найкоротший шлях обраним методом, та його вагу, граф у вигляді зображення. У випадку наявності від'ємного контуру у графі при вирішенні методом Беллмана-Форда виведеться вага шляху як $-\infty$, а шлях як (-)

6.3. Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Операційна система	Windows 10 (з останніми оновленнями)	Windows 10 (з останніми оновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/ Windows 8/Windows 10)	2 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	660x600	1024x768 або краще
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Python 3.7, бібліотека Networkx версії 2.6.3, бібліотека matplotlib версії 3.5.2, бібліотека Tkinter	

7 АНАЛІЗ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для пошуку найкоротшого шляху в графі методами Дейкстри та Беллмана-Форда.

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, ретельно перевіряються на валідність і лише потім подаються на обробку програмі.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення намалюємо граф на інтернет-ресурсу csacademy.com та вирішимо його «вручну»:

а) Методом Дейкстри (рисунок 7.1)

Намалюємо граф за допомогою інтернет-ресурсу csacademy.com (рисунок 7.2) та порахуємо шлях від 4 до 7 вершини «вручну» (таблиця 7.1):

Результат виконання методу Дейкстри наведено на рисунку 7.1

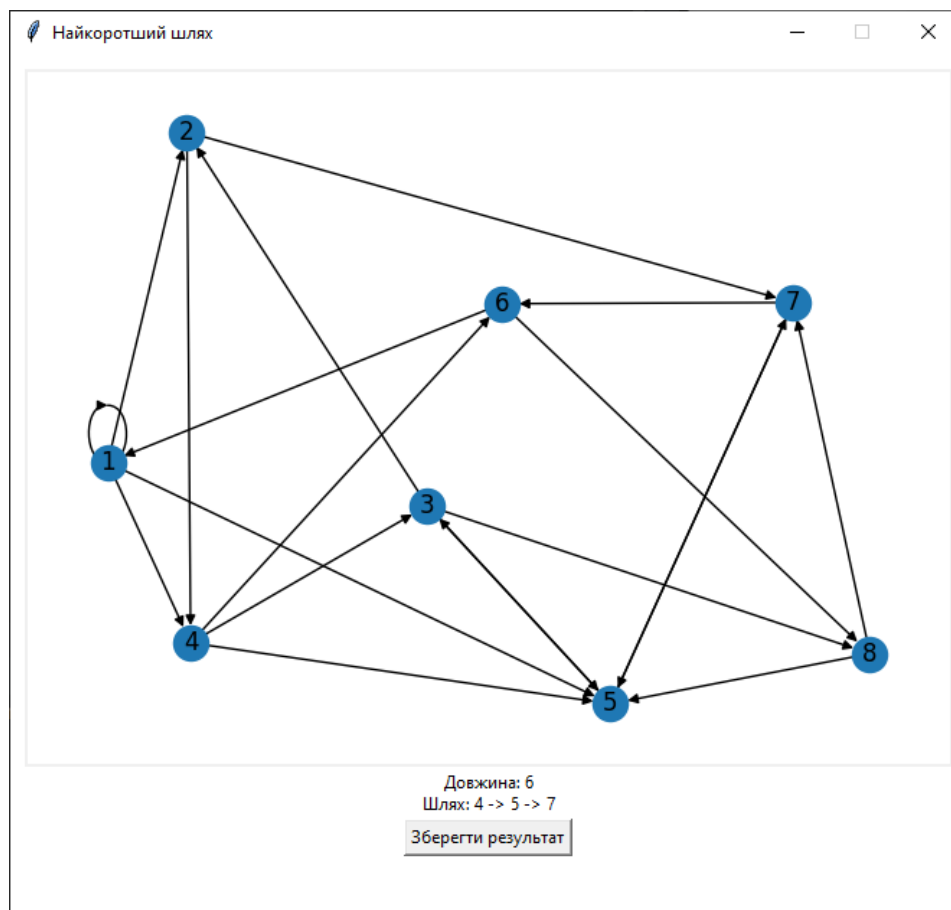


Рисунок 7.1 – Результат виконання методу Дейкстри

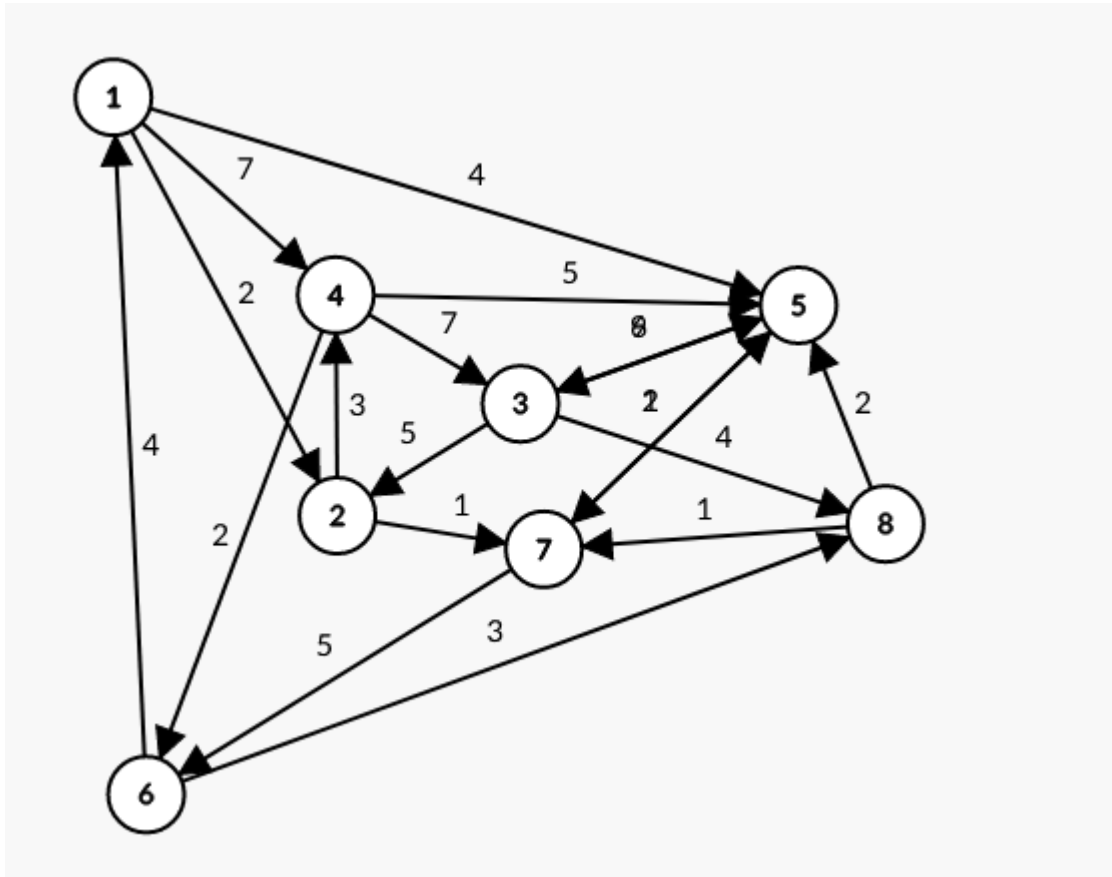


Рисунок 7.2 – Намальований граф на сайті csacademy.com

Таблиця 7.1 – Таблиця розрахунку найкоротшого шляху від 4 вершини

S (Перевірені вершини)	W (поточна вершина)	v ₁	v ₂	v ₃	v ₄	v ₅	v ₆	v ₇	v ₈
---	v ₄	∞	∞	∞	0	∞	∞	∞	∞
4	v ₆	∞	∞	7	-	5	2	∞	∞
4, 6	v ₅	6	∞	7	-	5	-	∞	5
4, 6, 5	v ₈	6	∞	7	-	-	-	6	5
4, 6, 5, 8	v ₁	6	∞	7	-	-	-	6	-
4, 6, 5, 8, 1	v ₇	-	8	7	-	-	-	6	-

З цієї таблиці (7.1) відслідковуємо вагу для 7 вершини – 6. Також відслідковуємо шлях для 7 вершини – 1 варіант: (4 → 5 → 7) та 2 варіант: (4 → 6 → 8 → 7). Метод Дейкстри дозволяє вибрати будь-який з цих шляхів, тому вибираємо 1 варіант. Оскільки результат виконання збігається з результатом, порохованим «вручну», то даний метод працює вірно.

б) Метод Беллмана-Форда (рисунок 7.3)

Намалюємо граф за допомогою інтернет-ресурсу csacademy.com (рисунок 7.4) та порахуємо шлях від 4 до 7 вершини «вручну» (таблиця 7.2):

Результат виконання методу Беллмана-Форда наведено на рисунку 7.3

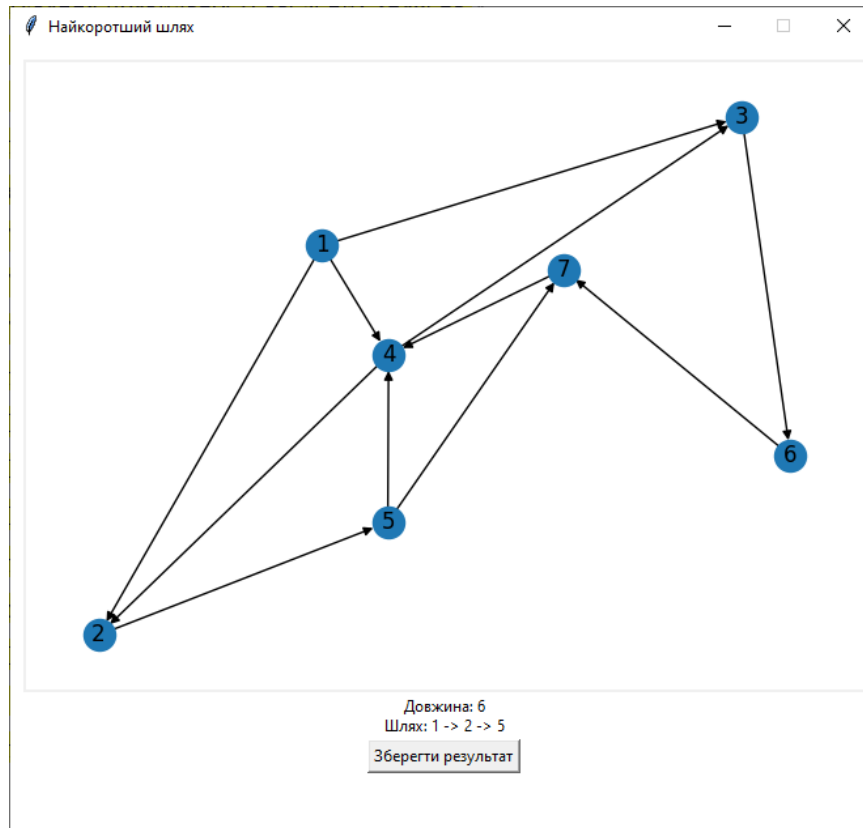


Рисунок 7.3 – Результат виконання методу Беллмана-Форда

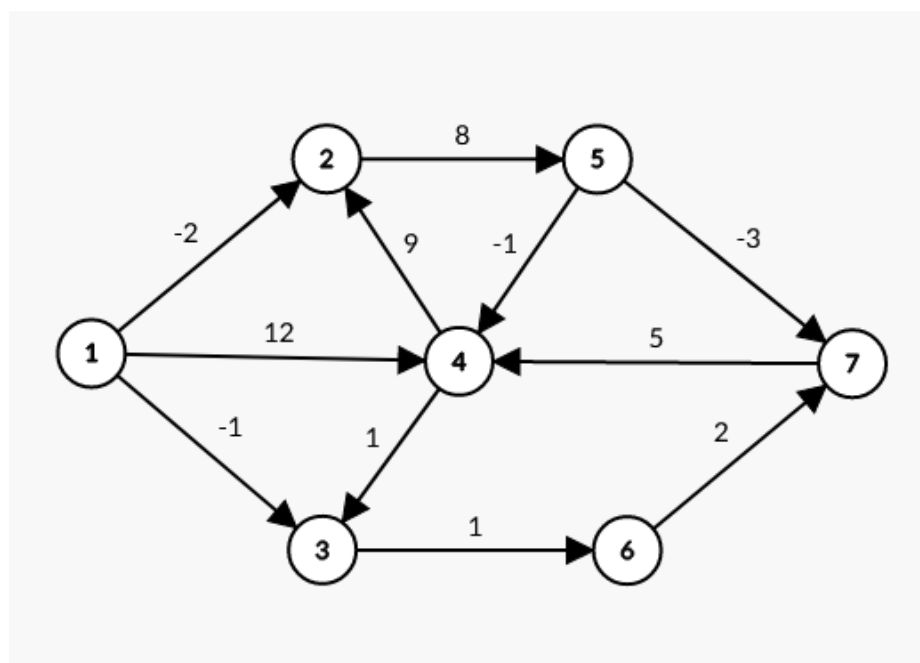


Рисунок 7.4 – Намальований граф на сайті csacademy.com

Таблиця 7.2 – Таблиця розрахунку найкоротшого шляху від 1 вершини

S (Перевірені вершини)	W (поточна вершина)	V ₁	V ₂	V ₃	V ₄	V ₅	V ₆	V ₇
---	1	0	∞	∞	∞	∞	∞	∞
1	2	0	-2	-1	12	∞	∞	∞
1, 2	3	0	-2	-1	12	6	∞	∞
1, 2, 3	6	0	-2	-1	12	6	0	∞
1, 2, 3, 6	7	0	-2	-1	12	6	0	2
1, 2, 3, 6, 7	5	0	-2	-1	7	6	0	2
1, 2, 3, 6, 7, 5	4	0	-2	-1	5	6	0	2
1, 2, 3, 6, 7, 5, 4		0	-2	-1	5	6	0	2

З цієї таблиці (7.2) відслідковуємо вагу для 5 вершини – 6. Також відслідковуємо шлях для 5 вершини – 1 варіант: (1 \rightarrow 2 \rightarrow 5). Оскільки результат виконання збігається з результатом, порахованим «вручну», то даний метод працює вірно.

Для проведення тестування ефективності програми було задано початкову вершину як 1, а кінцеву як – n, де n – розмірність графа. Також створено матриці ваг (таблиця 7.3).

Таблиця 7.3 – матриця ваг

x	v1	v2	v3	v4	...	vn
v1	10	12	13	14	...	10+n
v2	21	0	23	24	...	20+n
v3	31	32	0	34	...	30+n
v4	41	42	43	0	...	40+n
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots	$(n-1)*10+n$
vn	$n*10+1$	$n*10+2$	$n*10+3$	$n*10+4$	$n*10+(n-1)$	0

де n – розмірність системи.

Результати тестування ефективності алгоритмів знаходження найкоротшого шляху розміщено у таблиці 7.4

Таблиця 7.4 – Тестування ефективності методів

Кількість вершин	Параметри	Методи	
		Дейкстри	Беллмана-Форда
2	Кількість ітерацій	2	2
5	Кількість ітерацій	5	81
10	Кількість ітерацій	10	811
20	Кількість ітерацій	20	7221
100	Кількість ітерацій	100	980101
200	Кількість ітерацій	200	7920201

Візуалізація результатів таблиці 7.1 наведено на рисунку 7.1:

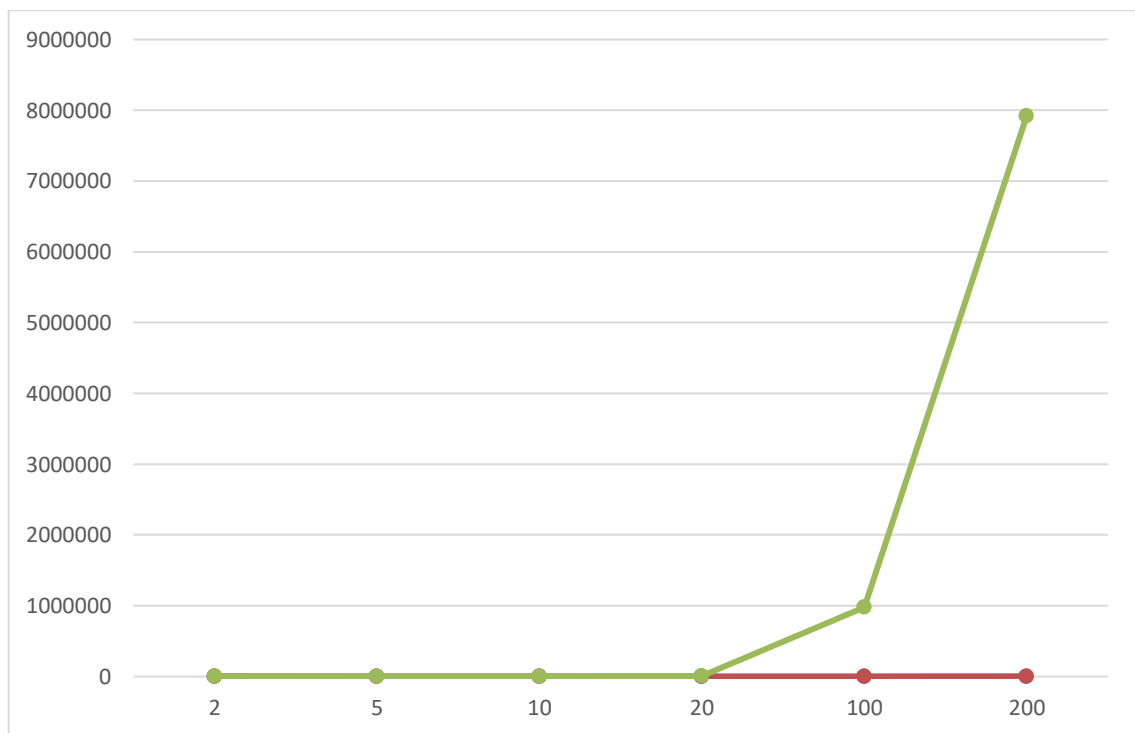


Рисунок 7.1 – Графік залежності кількості ітерацій методу від розміру вхідної системи

За результатами тестування можна зробити такі висновки:

- a) Всі розглянуті методи дозволяють обчислити найкоротший шлях в графі.
- b) Залежність кількості ітерацій від кількості вершин в методі Дейкстри – n , а для методу Беллмана-Форда – $n * (n + 1) + 1$, де n – кількість вершин в графі.
- c) Найоптимальнішим методом для пошуку найкоротшого шляху при додатних значеннях ребер є метод Дейкстри, проте метод Дейкстри працює некоректно при від'ємних значеннях ребер.

ВИСНОВКИ

Під час виконання курсової роботи я пригадав та дослідив різні методи пошуку найкоротшого шляху у зваженому орграфі; розробив програму для знаходження найкоротшого шляху для двох методів: Дейкстри і Беллмана-Форда. Реалізував її на такій мові програмування, як Python, з використанням графічного інтерфейсу бібліотекою tkinter.

Під час тестування переконався, що метод Дейкстри оптимальніший для пошуку, проте не підходить для графів з від'ємною вагою ребер.

Створив докладну інструкцію, щодо користування програмою, яка описує всі етапи з використанням скріншотів, аби у користувача не виникало питань.

Окрім того, провів аналіз результатів, де ще раз впевнився в ефективності та способах використання названих вище методів.

Також переконався у працездатності своєї програми, отримавши очікуваний результат при тестуванні.

ПЕРЕЛІК ПОСИЛАНЬ

1. Метод Дейкстри та Беллмана-Форда [Електронний ресурс] – Режим доступу до ресурсу:
<https://drive.google.com/file/d/1nmMf0mfowrRQvQuuPKioaBq2eXDdPTbT/view?usp=sharing>
2. matplotlib.pyplot [Електронний ресурс]–Режим доступу до ресурсу:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
3. tkinter [Електронний ресурс]–Режим доступу до ресурсу:
<https://docs.python.org/3/library/tkinter.html>
4. networkx [Електронний ресурс]–Режим доступу до ресурсу:
<https://networkx.org/>

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко Максим
Миколайович

« ____ » _____ 201_ р.

Виконавець:

Студент Замковий Дмитро
Володимирович

« ____ » _____ 201_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Розв’язання задач про

найкоротший шлях»

з дисципліни:

«Основи програмування»

1. *Мета:* Метою курсової роботи є розробка програмного забезпечення для пошуку найкоротшого шляху
2. *Дата початку роботи:* «29» квітня 2022 р.
3. *Дата закінчення роботи:* «__»_____ 2022 р.
4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- Можливість задавання розмірності мережі та початковий і кінцевий вузли мережі
- Можливість задавати зв'язки у мережі
- Можливість обирати метод знаходження найкоротшого шляху у мережі
- Можливість знаходження найкоротшого шляху у мережі обраним методом
- Можливість графічного відображення мережі та знайденої для неї найкоротшого шляху
- Можливість збереження результатів роботи алгоритму в текстовий файл
- Можливість відображення статистичних даних роботи алгоритму

2) Нефункціональні вимоги:

- Підтримка Windows 10 та вище
- 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 __. __. 202_ р.)

- 2) Об'єктно-орієнтоване проектування архітектури програмної системи (до __.__.202_p.)
- 3) Розробка програмного забезпечення (до __.__.202_p.)
- 4) Тестування розробленої програми (до __.__.202_p.)
- 5) Розробка пояснювальної записки (до __.__.202_p.).
- 6) Захист курсової роботи (до __.__.202_p.).

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
вирішення задачі пошуку найкоротшого шляху графа*

(Найменування програми (документа))

Онлайн-репозиторій

(Вид носія даних)

28 Кб, 9 аркушів

(Обсяг програми (документа), арк.,

студента групи ІІІ-13 І курсу

Замкового Дмитра Володимировича

main:

```
from view_menu import *
from tkinter import *

# Головний клас
class Main:
    # Виведення графа ініціалізація класу
    def __init__(self):
        root = Tk()
        App(root)
        root.mainloop()

Main()
```

Graph_class:

```
from Top_class import *
from help_modul import *
import networkx as nx
import matplotlib.pyplot as plt

# Клас графа
class TGraph:
    # Ініціалізація
    def __init__(self, size: int, method: int, start: int, finish: int, matrix:
[[]] -> None:

        # __size - Кількість вершин
        # __method - Метод вирішення (1 - Дейкстри, 2 - Беллмана-Форда)
        # __start - Початкова точка для пошуку маршруту
        # __finish - Кінцева точка
        # __tops - Масив об'єктів класу TTop

        if size >= 2:
            self.__size = size
        else:
            self.__size = 10

        if method == 1 or method == 2:
            self.__method = method
        else:
            self.__method = 1

        if start >= 0:
            self.__start = start
        else:
            self.__start = 0

        if finish >= 0:
            self.__finish = finish
        else:
            self.__finish = size - 1
        self.__tops = [(TTop(i)) for i in range(self.__size)]

        if method == 1: # Для Дейкстри беремо по модулю всі числа
            matrix = [(abs(matrix[i][j])) for j in range(self.__size)] for i in
range(self.__size)]

        # Перетворюємо матрицю ваг на масив об'єктів
        for i in range(self.__size):
```

```

        tmp_input = []
        tmp_output = []
        for j in range(self.__size):
            if matrix[j][i] != 0:
                tmp_input.append([j, matrix[j][i]])
            if matrix[i][j] != 0:
                tmp_output.append([j, matrix[i][j]])
        self.__tops[i].output_top = tmp_output

# Задаємо початкову точку і шлях до нього
self.__tops[self.__start].size = 0
self.__tops[self.__start].path = [self.__start]

# Метод Дейкстри
def __dijkstra(self) -> None:

    # w          - поточний елемент
    # for_check  - масив для задання перевірки елементів
    # tops_num   - масив вершин, які виходять з поточної
    # tops_val   - вага шляху від поточної вершини до її сусудів

    w = self.__start
    self.__tops[w].checked = True
    for_check = []
    for i in range(self.__size):
        if w == self.__finish:
            print('Кількість ітерацій ', i)
            return
        for_check.clear()
        tops_num = [(tops[0]) for tops in self.__tops[w].output_top]
        tops_val = [(tops[1]) for tops in self.__tops[w].output_top]
        for j in range(len(tops_num)):
            tmp = self.__tops[tops_num[j]].size
            self.__tops[tops_num[j]].size =
min(self.__tops[tops_num[j]].size, self.__tops[w].size + tops_val[j])
            if tmp != self.__tops[tops_num[j]].size:
                self.__tops[tops_num[j]].path = self.__tops[w].path +
[tops_num[j]]
        for j in self.__tops:
            if not j.checked:
                for_check.append(j.size)
            else:
                for_check.append(float('inf'))
        w = minimum(for_check)
        self.__tops[w].checked = True

# Метод Беллмана-Форда
def __bellman_ford(self) -> None:
    # Масив кортежей вершин для зручності обробки алгоритмом
    edge = []
    for i in range(self.__size):
        edge = edge + [(i, tops[0], tops[1]) for tops in
self.__tops[i].output_top]

    statistics = 1
    # Рахуємо довжину шляху і сам шлях
    for i in range(self.__size - 1):
        for v1, v2, s in edge:
            statistics += 1
            if self.__tops[v1].size != float('inf') and self.__tops[v1].size
+ s < self.__tops[v2].size:
                self.__tops[v2].size = self.__tops[v1].size + s
                self.__tops[v2].path = self.__tops[v1].path + [v2]
    print('Кількість ітерацій ', statistics)

# Перевіряємо на наявність від'ємних циклів

```

```

        for v1, v2, s in edge:
            if self.__tops[v1].size != float('inf') and self.__tops[v1].size + s
< self.__tops[v2].size:
                self.__tops[v2].size = float('-inf')
                self.__tops[v2].path = ['']

# Функція для вирішення графа заданим методом
def solve_graph(self) -> None:
    if self.__method == 1: # Метод Дейкстри
        self.__dijkstra()
    else: # Метод Беллмана-Форда
        self.__bellman_ford()

# Інформація
def info(self) -> {}:
    return {'size': self.__size,
            'method': self.__method,
            'start': self.__start,
            'finish': self.__finish,
            'tops': [(top.info()) for top in self.__tops]}

# Геттери
@property
def size(self) -> int:
    return self.__size

@property
def method(self) -> int:
    return self.__method

@property
def start(self) -> int:
    return self.__start

@property
def finish(self) -> int:
    return self.__finish

@property
def tops(self) -> {}:
    return [(i.info()) for i in self.__tops]

# Клас малюнка графа
class Draw:
    # Клас малюнка графа
    def __init__(self, gr: TGraph) -> None:
        # __size - кількість вершин
        # __tops - вершини графа

        self.__size = gr.size
        self.__tops = gr.tops

    # Намалювати граф
    def draw_gr(self) -> None:
        g = nx.MultiDiGraph()

        [g.add_node(i) for i in range(1, self.__size + 1)]

        for i in range(self.__size):
            [g.add_edge(i + 1, tops[0] + 1, tops[1]) for tops in
self.__tops[i]['output_top']]

        nx.draw(g, with_labels=True)
        plt.savefig('graf.png')

```

Top_class:

```
# Клас вершини графа
class TTop:
    # Ініціалізація
    def __init__(self, id: int, checked: bool = False, path: [] = [], size: int
= float('inf'), output_top: [] = []) -> None:

        # __id          - Ім'я вершини
        # __checked     - Чи перевірина вершина
        # __path        - Шлях до вершини
        # __size        - Вага шляху
        # __output_top  - Список сусідніх елементів, що виходять із вершини

        self.__id = id
        self.__checked = checked
        self.__path = path
        self.__size = size
        self.__output_top = output_top

    # Інформація
    def info(self) -> {}:
        return {'id': self.__id, 'checked': self.__checked, 'path': self.__path,
'size': self.__size, 'output_top': self.__output_top}

    # Геттери
    @property
    def id(self):
        return self.__id

    @property
    def checked(self):
        return self.__checked

    @property
    def path(self):
        return self.__path

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

    @property
    def output_top(self):
        return self.__output_top

    # Сеттери
    @checked.setter
    def checked(self, value: bool):
        self.__checked = value

    @path.setter
    def path(self, value: []):
        self.__path = value

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

    @output_top.setter
    def output_top(self, value: []):
        self.__output_top = value
```


view_menu:

```

from tkinter import *
from help_modul import save_res_in_file
from Graph_class import *

class App:
    def __init__(self, root):
        self.__matrix_val = []

        # Виведення графа
        def gr_output(len, path, data):
            def save_res():
                save_res_in_file('res', data)
                text_save.pack()

            # Видалення рамок 1, 2 та 3
            frame_input_1.destroy()
            frame_input_2.destroy()
            frame_input_3.destroy()
            btn_gen.destroy()

            # Зміна розміру вікна
            width = 660
            height = 600
            root.geometry('%dx%d' % (width, height))

            # Рамка для виведення графа
            frame_output = Frame(root)
            frame_output['bg'] = '#ffffff'
            frame_output.place(x=10, y=10)

            # Зображення графа
            img = PhotoImage(file='graf.png')
            lb_img = Label(frame_output)
            lb_img.image = img
            lb_img['image'] = lb_img.image
            lb_img.pack()

            # Текст про довжину і шлях в графі
            out = Label(frame_output)
            out['text'] = f'Довжина: {len}\nШлях: {path}'
            out['bg'] = '#ffffff'
            out.pack()

            # Кнопка збереження
            btn_save = Button(frame_output)
            btn_save['text'] = 'Зберегти результат'
            btn_save['command'] = save_res
            btn_save.pack()

            # Повідомлення про успішне збереження
            text_save = Label(frame_output)
            text_save['text'] = 'Успішно збережено у файл res'
            text_save['bg'] = '#ffffff'

            # Кінцева функція для закінчення введення даних
            def finish():
                scale_start['state'] = DISABLED
                scale_finish['state'] = DISABLED
                gr = TGraph(scale_size.get(), method.get(), scale_start.get() - 1,
scale_finish.get() - 1,
                        self.__matrix_val)
                gr.solve_graph()

```

```

img = Draw(gr)
img.draw_gr()

path = ''
if not gr.tops[gr.finish]['path']:
    path = 'inf'
elif gr.tops[gr.finish]['path'][0] == ':':
    path = '-'
else:
    for i in range(len(gr.tops[gr.finish]['path'])):
        path = path + (str(int(gr.tops[gr.finish]['path'][i]) + 1) +
' -> ')

    path = path[:len(path) - 5] + str(gr.finish + 1)
gr_output(gr.tops[gr.finish]['size'], path, gr.info())

# Функція, яка виводить повзунки для початкової і кінцевої вершини та
публікує рамку 3
def start_finish_btn():
    frame_input_3.place(y=300, x=10)
    scale_start['to'] = scale_size.get()
    scale_finish['to'] = scale_size.get()

# Перевіряє правильність введення даних у рамці 1
def check_btn_gen():
    if method.get() == 1:
        error_method['fg'] = '#ffffff'
        if_method = True
    elif method.get() == 2:
        error_method['fg'] = '#ffffff'
        if_method = True
    else:
        error_method['fg'] = 'red'
        if_method = False

    if if_method:
        frame_input_2.place(y=180, x=10)
        scale_size['state'] = DISABLED
        rbutton1['state'] = DISABLED
        rbutton2['state'] = DISABLED

# Перевіряє правильність введення даних у рамці 2
def check_btn_set():

    def top_deleted():
        if_check = True
        self.__matrix_val = [(input_val[j][i].get()) for j in
range(scale_size.get())] for i in
                                range(scale_size.get())]
        for i in range(scale_size.get()):
            for j in range(scale_size.get()):
                mx = self.__matrix_val[j][i]
                if mx.isdigit():
                    input_val[i][j]['bg'] = '#ffffff'
                    self.__matrix_val[j][i] =
int(self.__matrix_val[j][i])
                elif mx == ':':
                    self.__matrix_val[j][i] = 0
                    input_val[i][j]['bg'] = '#ffffff'
                elif mx[0] == '-':
                    if mx[1:].isdigit():
                        input_val[i][j]['bg'] = '#ffffff'
                        self.__matrix_val[j][i] =
int(self.__matrix_val[j][i])
                    else:
                        input_val[i][j]['bg'] = '#fca9a9'
                        if_check = False

```

```

        else:
            input_val[i][j]['bg'] = '#fca9a9'
            if_check = False
        if if_check:
            top.destroy()
            btn_set['state'] = NORMAL

        btn_set['state'] = DISABLED
        top = Toplevel()
        input_val = [(Entry(top, bg='#ffffff', width=3)) for i in
range(scale_size.get())] for j in
range(scale_size.get())]
        [(input_val[i][j].grid(column=i, row=j)) for i in
range(scale_size.get())] for j in
range(scale_size.get())]
        top.protocol('WM_DELETE_WINDOW', top_deleted)
        btn_set['command'] = start_finish_btn
        btn_set['text'] = 'Продовжити'

# Налаштування вікна
root.title('Найкоротший шлях')
width = 325
height = 600
screenwidth = root.winfo_screenwidth()
screenheight = root.winfo_screenheight()
root.geometry('%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2,
(screenheight - height) / 2))
root.resizable(width=False, height=False)
root['bg'] = 'white'

# Рамка 1 для кількості вершин і алгоритма вирішення
frame_input_1 = Frame(root)
frame_input_1['bg'] = '#ffffff'
frame_input_1.place(x=10, y=10)

# Підказка щодо кількості вершин
txt_size = Label(frame_input_1)
txt_size['text'] = 'Виберіть кількість вершин'
txt_size['bg'] = '#ffffff'
txt_size.pack()

# Повзунок вибору кількості вершин
scale_size = Scale(frame_input_1)
scale_size['orient'] = HORIZONTAL
scale_size['length'] = 300
scale_size['from'] = 2
scale_size['to'] = 20
scale_size['tickinterval'] = 2
scale_size['resolution'] = 1
scale_size['bg'] = '#ffffff'
scale_size.pack()

# Підказка щодо вибору метода
txt_method = Label(frame_input_1)
txt_method['text'] = 'Виберіть метод'
txt_method['bg'] = '#ffffff'
txt_method.pack()

# Вибір метода
method = IntVar()
rbutton1 = Radiobutton(frame_input_1)
rbutton1['text'] = 'Метод Дейкстри'
rbutton1['variable'] = method
rbutton1['value'] = 1
rbutton1['bg'] = '#ffffff'
rbutton2 = Radiobutton(frame_input_1)

```

```

rbutton2['text'] = 'Метод Беллмана-Форда'
rbutton2['variable'] = method
rbutton2['value'] = 2
rbutton2['bg'] = '#ffffff'
rbutton1.pack(side='left')
rbutton2.pack(side='right')

# Підказка, щодо того, що користувач не вибрав метод
error_method = Label(root)
error_method['text'] = 'Виберіть метод!'
error_method['fg'] = '#ffffff'
error_method['bg'] = '#ffffff'
error_method.place(y=132, x=11)

# Кнопка для підтвердження вибору кількості вершин і методу рішення
btn_gen = Button(root)
btn_gen['text'] = 'Продовжити'
btn_gen['width'] = 42
btn_gen['command'] = check_btn_gen
btn_gen.place(y=150, x=11)

# Рамка 2 для задання графа матрицею ваг
frame_input_2 = Frame(root)
frame_input_2['bg'] = '#ffffff'

# Підказка щодо задання графа
txt_set = Label(frame_input_2)
txt_set['text'] = 'Матриця вагів'
txt_set['bg'] = '#ffffff'
txt_set['font'] = '20'
hint_set = Label(frame_input_2)
hint_set[
    'text'] = 'Заповніть матрицю ваги.\nПусті клітинки будуть вважатися
за 0\nПри вирішенні методом Дейкстри всі\nчисла візьмуться по модулю
автоматично'
hint_set['bg'] = '#ffffff'
txt_set.pack()
hint_set.pack()

# Кнопка підтвердження задання графа
btn_set = Button(frame_input_2)
btn_set['text'] = 'Задати матрицю ваг'
btn_set['width'] = 42
btn_set['command'] = check_btn_set
btn_set.pack()

# Рамка 3 для вибору початкової і кінцевої вершини
frame_input_3 = Frame(root)
frame_input_3['bg'] = '#ffffff'

# Підказка щодо початкової вершини
txt_start = Label(frame_input_3)
txt_start['text'] = 'Виберіть початкову вершину'
txt_start['bg'] = '#ffffff'
txt_start.pack()

# Повзунок вибору початкової вершини
scale_start = Scale(frame_input_3)
scale_start['orient'] = HORIZONTAL
scale_start['length'] = 300
scale_start['from'] = 1
scale_start['tickinterval'] = 2
scale_start['resolution'] = 1
scale_start['bg'] = '#ffffff'
scale_start.pack()

```

```

# Підказка щодо кінцевої вершини
txt_finish = Label(frame_input_3)
txt_finish['text'] = 'Виберіть кінцеву вершину'
txt_finish['bg'] = '#ffffff'
txt_finish.pack()

# Повзунок щодо вибору кінцевої вершини
scale_finish = Scale(frame_input_3)
scale_finish['orient'] = HORIZONTAL
scale_finish['length'] = 300
scale_finish['from'] = 1
scale_finish['tickinterval'] = 2
scale_finish['resolution'] = 1
scale_finish['bg'] = '#ffffff'
scale_finish.pack()

# Кнопка для вирішення графа
btn_count = Button(frame_input_3)
btn_count['text'] = 'Порахувати'
btn_count['width'] = 42
btn_count['height'] = 8
btn_count['command'] = finish
btn_count.pack()

```

help_modul:

```

# Повертає індекс мінімального елемента в масиві
import json

# Пошук індекса масиву з найменшим значенням елемента
def minimum(r1: []) -> int:

    # minim_num - індекс мінімального елемента
    # minim_val - мінімальний елемент

    minim_num = 0
    minim_val = r1[0]
    for i in range(len(r1)):
        if r1[i] < minim_val:
            minim_val = r1[i]
            minim_num = i
    return minim_num

# Записати у файл масив даних
def save_res_in_file(file_name: str, data: []) -> None:
    with open(file_name, 'w') as file:
        file.write(json.dumps(data))

```