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 ІП-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 "ІПЗ"

Курс 1 Група ІП-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 Муха І. П.

(підпис) (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 6](#_Toc105823577)

[1 Постановка задачі 7](#_Toc105823578)

[2 Теоретичні відомості 8](#_Toc105823579)

[2.1. Метод Дейкстри 8](#_Toc105823580)

[2.2. Метод Беллмана-Форда 8](#_Toc105823581)

[3 Опис алгоритмів 9](#_Toc105823582)

[3.1. Загальний алгоритм 9](#_Toc105823583)

[3.2. Алгоритм методу Дейкстри 10](#_Toc105823584)

[3.3. Алгоритм Беллмана-Форда 11](#_Toc105823590)

[4 Опис програмного забезпечення 12](#_Toc105823598)

[4.1. Діаграма класів програмного забезпечення 12](#_Toc105823599)

[4.2. Опис методів частин програмного забезпечення 14](#_Toc105823600)

[4.2.1. Користувацькі методи 14](#_Toc105823601)

[4.2.2. Стандартні методи 20](#_Toc105823602)

[5 Тестування програмного забезпечення 24](#_Toc105823603)

[5.1. План тестування 24](#_Toc105823604)

[5.2. Приклади тестування 25](#_Toc105823605)

[6 Інструкція користувача 34](#_Toc105823606)

[6.1. Робота з програмою 34](#_Toc105823607)

[6.2. Формат вхідних та вихідних даних 40](#_Toc105823611)

[6.3. Системні вимоги 40](#_Toc105823612)

[7 Аналіз результатів 41](#_Toc105823613)

[Висновки 47](#_Toc105823614)

[Перелік посилань 48](#_Toc105823615)

[Додаток А Технічне завдання 49](#_Toc105823616)

[Додаток Б Тексти програмного коду 52](#_Toc105823617)

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 – підходь 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 Белламана-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.

# 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

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.

# Теоретичні відомості

Мережу можна задати таким способом:

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

## 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 , 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

## 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.

# Опис алгоритмів

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

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| *matrix\_val* | Матриця ваг графа |
| tops\_num | В методі Дейкстри масив, в якому лежать номера сусідніх елементів |
| edge | В методі Беллмана-Форда масив списків у форматі (початок\_ребра, кінець\_ребра, вага\_ребра), який показуєвсі ребра у цьому графі |

## Загальний алгоритм

* + 1. ПОЧАТОК
    2. Зчитати кількіть вершин графа
    3. Зчитати метод рішення графа.
    4. Зчитати матрицю ваг:
       1. Зчитати матрицю системи:
       2. Цикл проходу по всіх рядках матриці системи (:
          1. Цикл проходу всіх стовпцях матриці системи   
             (:
          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.
    5. Зчитати початкову вершину
    6. Зчитати кінцеву вершину
    7. ЯКЩО обраний метод Дейкстри, ТО обробити дані згідно алгоритму методу Дейкстри (пункт 1.2)
    8. ЯКЩО обраний метод Беллмана-Форда, ТО обробити дані згідно алгоритму методу Беллмана-Форда (пункт 1.3)
    9. Зчитати початкову точку для пошуку
    10. Зчитати кінцеву точку для пошуку
    11. КІНЕЦЬ

## Алгоритм методу Дейкстри

### ПОЧАТОК

### Задати поточний елемент, як початковим

#### ЦИКЛ проходу по всіх вершинах графа:

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

* + - * 1. Задати масив сусідніх вершин до поточної вершини tops\_num
        2. ЦИКЛ проходу по всіх елементах tops\_num:

Вибираємо мінімальне значення між вагою уже існуючого шляху у первіряємого елемента вершини та сумою ваги поточного елемента та вагою ребра між поточним і перевіряємим елементом

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

* + - * 1. Помітити поточний елемент, як пройдений
        2. Вибрати елемент з мінімальною вагою шляху серед непройдених

### КІНЕЦЬ

## Алгоритм Беллмана-Форда

3.3.1. ПОЧАТОК

### Задати масив (edge) списків ребер графа за принципом (початок\_ребра, кінець\_ребра, вага\_ребра)

### ЦИКЛ проходу по вершинах графа – 1

#### ЦИКЛ проходу по масиву списків ребер графа (edge)

#### 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

#### ЦИКЛ проходу по масиву списків ребер графа (edge)

#### 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 -∞ і видалити шлях до цієї вершини

### КІНЕЦЬ

# Опис програмного забезпечення

## 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

## Опис методів частин програмного забезпечення

### Користувацькі методи

У таблиці 1.2 наведено перелік користувацьких методів використаних в курсовій роботі

– Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | 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 классу | 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 правильності 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 классу | 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 классу | 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 зафарбована | --- | True/False | Top\_class |

Продовження таблиці 1.2

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

Продовження таблиці 1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | 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 мінімального 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 |

### Стандартні методи

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

– Стандартніметоди

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Пакет |
| 1 | Tk | Tk | Створити форму | --- | Об’єкт Tk | tkinter |
| 2 | Tk | mainloop | Головний цикл форми | --- | --- | tkinter |
| 3 | Tk | title | Встановлення заголовок форми | Текст | --- | tkinter |
| 4 | Tk | geometry | Задання розміру і розташування 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 классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Пакет |
| 8 | Tk | resizable | Задати можливість редагувати розміри форми користувачу | Висота та/або ширина дорівнює True/False | --- | tkinter |
| 9 | Toplevel | Toplevel | Створити верхню форму | Форма | Об’єкт Toplevel | tkinter |
| 10 | Toplevel | protocol | Відслідковувати 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 | x, y | --- | tkinter |
| 14 | Frame | destroy | «Зламати» рамку | --- | --- | tkinter |
| 15 | Label | Label | Додати текстове поле | Форма/рамка | Об’єкт Label | tkinter |

Продовження таблиці 1.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | 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 | x, y | --- | 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 воду 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 классу | 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 |

# Тестування програмного забезпечення

## 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-небуть роду, на різні критичні моменти для своєчасного виправлення багів.

Для перевірки коректності роботи програми проведемо тестування за наступними критеріями

1. Тестування правильності введення даних.
   1. Тестування при введені недозволених символів в матрицю ваг
   2. Тестування при введені від’ємних чисел в методі Дейкстри
   3. Тестування при введені від’ємних чисел в методі Беллмана-Форда
   4. Тестування при не коректному вибору методу вирішення
   5. Тестування при незаповнені матриці ваг
   6. Тестування при частковому незаповнені матриці
2. 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
3. Тестування коректності роботи методів Дейкстри та Беллмана-Форда.
   1. Перевірка коректності роботи метода Дейкстри
   2. Перевірка коректності роботи метода Беллмана-Форда
4. Тестування побудови графа

## Приклади тестування

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

Таблиця 5.1 ‑ Приклад роботи програми при введені недозволених символів в матрицю ваг

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно для вводу матриці ваг |
| Вхідні дані |  |
| Схема проведення тесту | Введення невалідних значення в матрицю ваг |
| Очікуваний результат | Повідомлення про помилку невалідності значень (підсвічення елементу з некоректними даними) |
| Стан програми після проведення випробувань | Підсвічено елемент матриці, яки є невалідним червоним кольором |

Таблиця 5.2 ‑ Приклад роботи програми при введені 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 |  |
| 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 |  |
| 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 вибору 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 незаповнені матриці ваг

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість частково не заповнювати матрицю ваг |
| Початковий стан програми | Відкрите вікно матриці ваг |
| Вхідні дані |  |
| 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 |  |
| 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 значеня в матриці ваг по модулю |

Таблиця 5.8 ‑ Приклад роботи програми при методі Беллмана-Фордда 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 |  |
| Схема проведення тесту | Введення графа через матрицю ваг з від’ємним контуром |
| Очікуваний результат | Виведення, як результат, шлях - (-), а довжину (-∞) |
| Стан програми після проведення випробувань | Взяті значеня 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 | початок – 1, кінець – 2 |
| Схема проведення тесту | Задати значеня 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 | початок – 1, кінець – 2 |
| Схема проведення тесту | Задати значеня 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 |
| 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 |
| 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) |

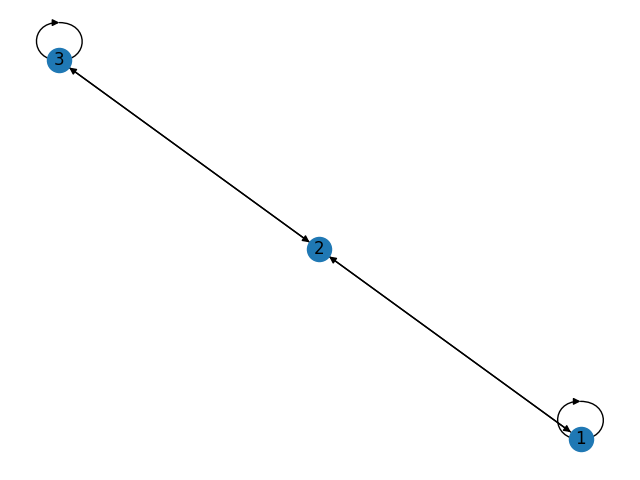
Таблиця 5.15 ‑ Приклад роботи програми введенні некоректних даних на побудову графа

|  |  |
| --- | --- |
| 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 |
| 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.

# Інструкція користувача

## Робота з програмою

### Встановлення

Для запуску програми потрібно установити Python 3.7 з двума додатковими пакетами: mathplotlib, networkx. Їх можна завантажити на офіційних сайтах: [Python](https://www.python.org/downloads/), [mathplotlib](https://github.com/wdecoster/methplotlib), [networkx](https://networkx.org/). Для встановлення потрібно ввести у командний рядок два рядка коду:

pip install mathplotlib

pip install networkx

Потрібно переконатися, що пакети додано до системних змінних.

Після цього завантажте код з [GitHub](https://github.com/DmytroZamkovyi/Course_works/tree/main/KR_2_semestr) на ваш комп’ютер

### Запуск

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

### Початок роботи

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

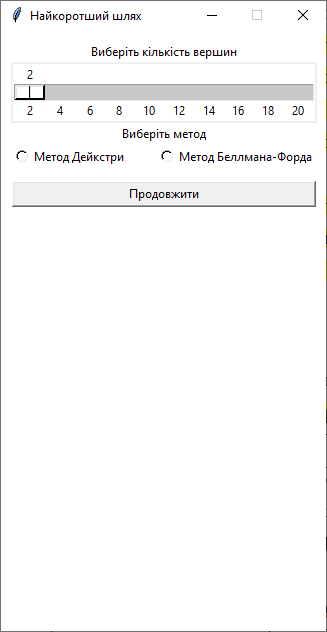


Рисунок 6.1 – Головне вікно програми

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

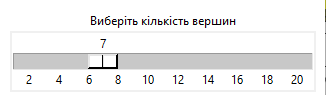


Рисунок 6.2 – Вибір кількості вершин графа

Потім вибираємо метод розв’язку на перемикачі з назвою «Виберіть метод» шляхом відмічення потрібного методо. 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):

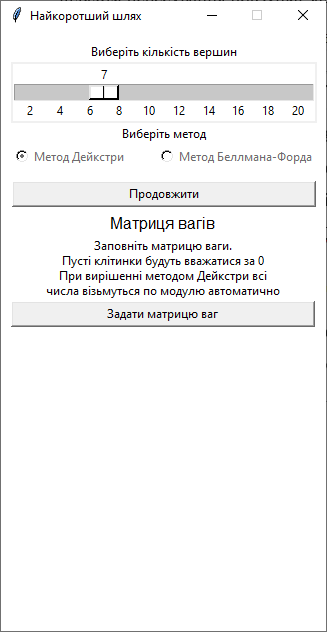


Рисунок 6.4 – Наступна частина вводу, попередні елементи заблоковано, тобто змінити їх вже не можливо

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 введені некоректних даних таблиця для вводу матриці ваг не закриється і підсвітить некоректні елементи червоним кольором (рисунок 6.5). Їх потрібно буде виправити та закрити таблицю. При коректному вводі всі результати в матриці збережуться автоматично та кнопка «Задати матрицю ваг» зміниться на кнопку «Продовжити» (рисунок 6.6):

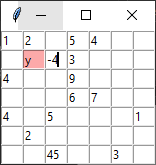


Рисунок 6.5 – Приклад вводу некоректних даних

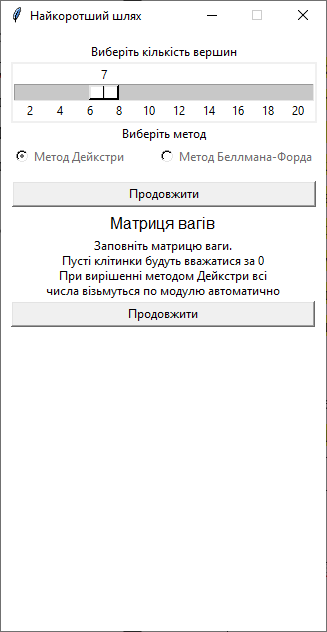


Рисунок 6.6 – Зміна кнопки «Задати матрицю ваг» на кнопку «Продовжити»

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):

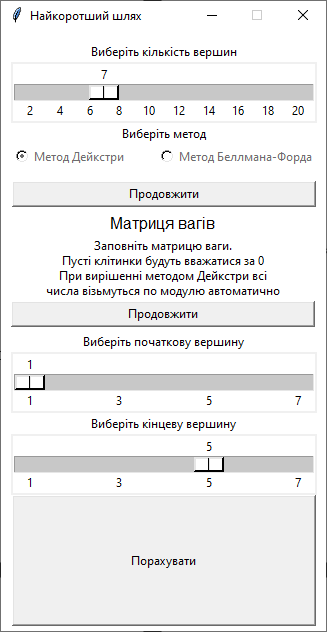


Рисунок 6.7 – Вибір початкової і кінцевої вершини

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

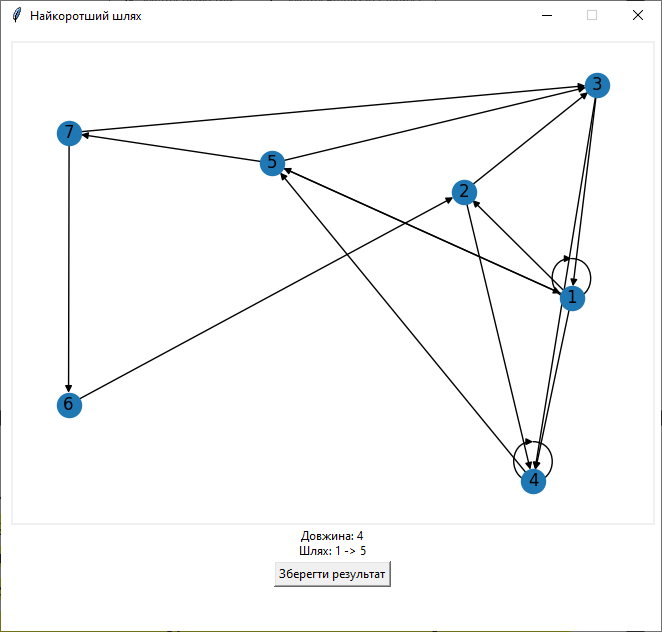


Рисунок 6.8 – Результат

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

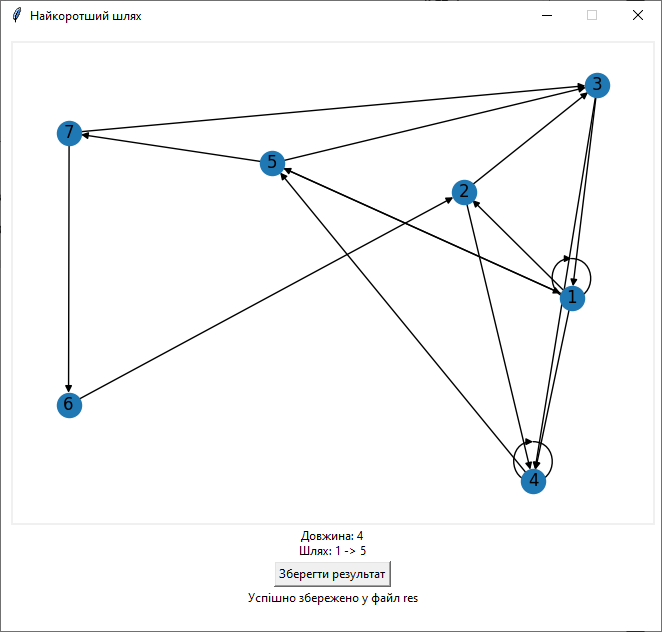


Рисунок 6.9 – Збереження результату у файл

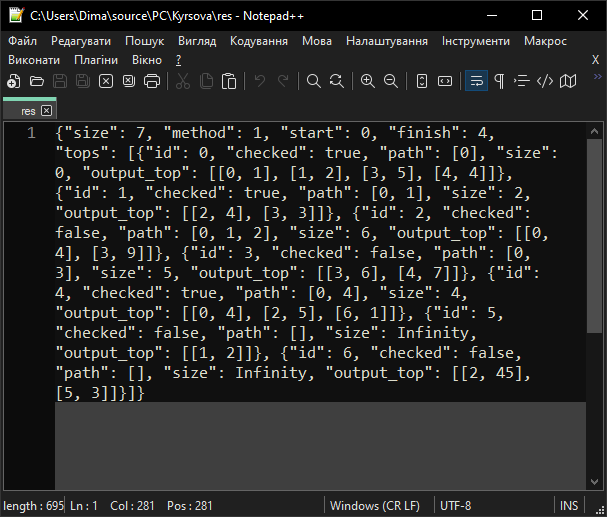


Рисунок 6.10 – файл «res»

## 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 вирішення методом Беллмана-Форда виведеться вага шляху як -∞, а шлях як (-)

## 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 обновленнями) | | Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  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 | 660х600 | | 1024х768 або краще |
| Прилади введення | | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | | Python 3.7, бібліотека Networkx версії 2.6.3, бібліотека mathplotlib версії 3.5.2, бібліотека Tkinter | |

# 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 провіряються 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

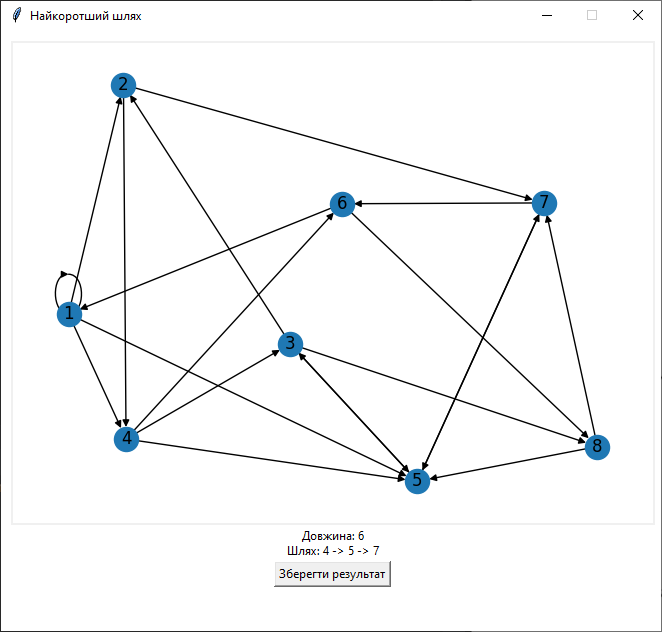


Рисунок 7.1 – Результат виконання методу Дейкстри

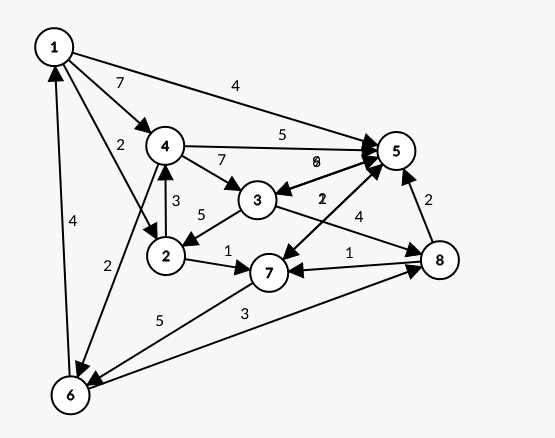


Рисунок 7.2 – Намальований граф на сайті csacademy.com

Таблиця 7.1 – Таблиця розрахунку найкоротшого шляху від 4 вершини

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S (Перевірені вершини) | w (поточна вершина) | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 |
| --- | v4 | ∞ | ∞ | ∞ | 0 | ∞ | ∞ | ∞ | ∞ |
| 4 | v6 | ∞ | ∞ | 7 | - | 5 | 2 | ∞ | ∞ |
| 4, 6 | v5 | 6 | ∞ | 7 | - | 5 | - | ∞ | 5 |
| 4, 6, 5 | v8 | 6 | ∞ | 7 | - | - | - | 6 | 5 |
| 4, 6, 5, 8 | v1 | 6 | ∞ | 7 | - | - | - | 6 | - |
| 4, 6, 5, 8, 1 | v7 | - | 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, порахованим «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):

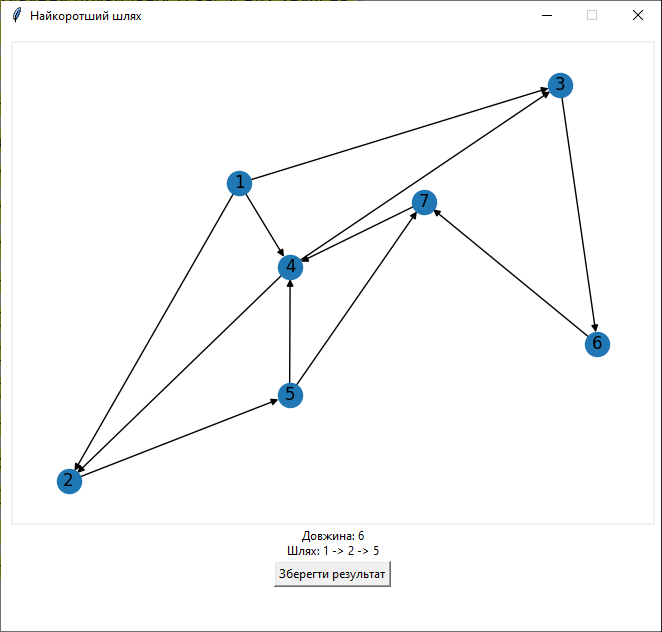
Результат виконання методу Беллмана-Форда наведено на рисунку 7.3

Рисунок 7.3 – Результат виконання методу Беллмана-Форда

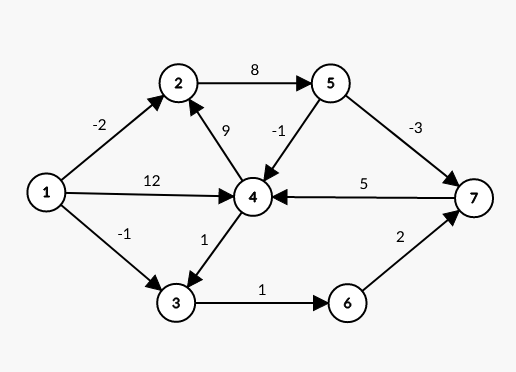


Рисунок 7.4 – Намальований граф на сайті csacademy.com

Таблиця 7.2 – Таблиця розрахунку найкоротшого шляху від 1 вершини

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S (Перевірені вершини) | w (поточна вершина) | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| --- | 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 –> 2 –> 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| х | 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 |
| ⁝ | ⁝ | ⁝ | ⁝ | ⁝ | ⁝⁝⁝ | (n-1)\*10+n |
| vn | n\*10+1 | n\*10+2 | n\*10+3 | n\*10+4 | n\*10+(n-1) | 0 |

де – 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:

1. Всі розглянуті методи дозволяють обчислити найкоротший шлях в графі.
2. Залежність кількості ітерацій від кількості вершин в методі Дейкстри – , а для методу Беллмана-Форда – , де n – кількість вершин в графі.
3. 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»

Київ 2022

* 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 розмірність 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

1. 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.

* 1. *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\_р.)
3. Розробка програмного забезпечення (до \_\_.\_\_.202\_р.)
4. Тестування розробленої програми (до \_\_.\_\_.202\_р.)
5. Розробка пояснювальної записки (до \_\_.\_\_.202\_ р.).
6. Захист курсової роботи (до \_\_.\_\_.202\_ р.).
   1. *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



*студента групи ІП-13 І курсу*

*Замкового Дмитра Володимировича*

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

*28 Кб, 9 аркушів*

(Вид носія даних)

*Онлайн-репозиторій*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі пошуку найкоротшого шляху графа*

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, яки 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))