

Міністерство освіти і науки України  
Національний технічний університет України «Київський політехнічний інститут  
імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №8 з дисципліни  
«Алгоритми та структури даних 2. Структури даних»

« Жадібні алгоритми»  
Варіант 3

Виконав студент ІП-15, Борисик Владислав Тарасович  
(шифр, прізвище, ім'я, по батькові)

Перевірив Соколовський Владислав Володимирович  
( прізвище, ім'я, по батькові)

## Мета лабораторної роботи

Мета роботи – вивчити основні підходи реалізації жадібного алгоритму на прикладі задачі комівояжера.

## Завдання

У даній роботі необхідно запропонувати жадібний алгоритм для задачі комівояжера.

Задача комівояжера формулюється для повного графу. Для зваженого повного графу  $G$  з  $n$  вершинами задані відстані між усіма парами вершин  $(i, j)$ . Необхідно знайти найкоротший маршрут, який проходить через всі вершини графу та заходить в кожную вершину лише один раз.

В даній роботі розглядається симетричний варіант задачі, коли відстань від міста  $i$  до міста  $j$  дорівнює відстані від  $j$  до  $i$  (відстані між  $(i, j)$  та  $(j, i)$  рівні).

Задача комівояжера відноситься до NP-повних задач і для неї не існує оптимального алгоритму, який би працював за поліноміальний час. Тому для її розв'язання часто використовуються евристичні алгоритми, до яких відносяться і жадібні алгоритми.

Вам необхідно запропонувати ідею жадібного алгоритму та перевірити його роботу на декількох екземплярах задач.

## Псевдокод алгоритму

```
def find_min(arr: list[int], except_indexes: list[int]):  
    # мінімальний елемент  
    min_value = -1  
    # індекс мінімального елементу  
    index = -1  
  
    # ініціалізуємо мінімальний елемент і його індекс  
    # для цього ітеруємось по основному масиву  
    Для i від 0 до довжини масиву arr з кроком 1:  
        # ітеруємось по масиву виключень  
        Для j від 0 до довжини масиву except_indexes з кроком 1:  
            # якщо i != елементу в масиві виключені i != 0  
            Якщо i != except_indexes[j] і arr[i] != 0:  
                # ініціалізуємо мінімальний елемент  
                min_value := arr[i]  
                # індекс мінімального елементу  
                index := i  
  
                # виходимо з циклу  
                break  
  
        Все якщо  
  
    Все повторити  
  
    Все повторити  
  
    # ітеруємось по основному масиву  
    Для i від 0 до довжини масиву arr з кроком 1:  
        # ітеруємось по масиву виключень  
        Для j від 0 до довжини масиву except_indexes з кроком 1:  
            # якщо i дорівнює елементу в масиві виключень, то пропускаємо цей хід циклу  
            Якщо i == except_indexes[j]:  
                continue  
  
        Все якщо  
  
    Все повторити  
  
    # якщо елемент масиву != 0 і елемент масиву менший за min value  
    Якщо arr[i] != 0 і arr[i] < min_value:  
        # змінюємо мінімальний елемент  
        min_value =: arr[i]  
        # змінюємо індекс мінімального елементу  
        index := i  
  
    Все якщо  
  
    Все повторити  
  
    Повернути min_value, index
```

```

def find_route(matrix: Matrix):
    # маршрут обходу (починається завжди з 0 вершини)
    route := [0]
    # довжина маршруту
    route_value := 0

    # лічильник
    i = 0
    # масив з індексами виключень
    # зберігає вершини, в яких ми вже були
    except_indexes := [0]

    Поки len(route) < len(matrix):
        # рядок матриці
        row := matrix[i]
        # знаходимо мінімальний елемент і наступну вершину для обходу
        min_value, next_point := find_min(row, except_indexes)
        # змінюємо довжину маршруту
        route_value += min_value
        # додаємо вершину в маршрут
        route.append(next_point)
        # додаємо вершину в масив виключень
        except_indexes.append(next_point)

        # змінюємо лічильник
        i := next_point

    Все повторити

    # додаємо до маршруту першу вершину (так як ми повинні в ній закінчити свій шлях)
    route.append(0)

    # передостанній елемент масиву зі шляхом
    penultimate_element := route[len(route) - 2]
    # останній елемент масиву зі шляхом
    last_element := route[len(route) - 1]

    # додаємо до довжини шляху довжину від останньої вершини до першої
    route_value += matrix[penultimate_element][last_element]

    Повернути route, route_value

```

## Програмна реалізація алгоритму на мові Python

```
def find_min(arr: list[int], except_indexes: list[int]):
    # мінімальний елемент
    min_value = -1
    # індекс мінімального елементу
    index = -1

    # ініціалізуємо мінімальний елемент і його індекс
    # для цього ітеруємось по основному масиву
    for i in range(len(arr)):
        # ітеруємось по масиву виключень
        for j in range(len(except_indexes)):
            # якщо i != елементу в масиві виключені i != 0
            if i != except_indexes[j] and arr[i] != 0:
                # ініціалізуємо мінімальний елемент
                min_value = arr[i]
                # індекс мінімального елементу
                index = i

            # виходимо з циклу
            break

    # ітеруємось по основному масиву
    for i in range(1, len(arr)):
        # ітеруємось по масиву виключень
        for j in range(len(except_indexes)):
            # якщо i дорівнює елементу в масиві виключень, то пропускаємо цей хід циклу
            if i == except_indexes[j]:
                continue

        # якщо елемент масиву != 0 і елемент масиву менший за min value
        if arr[i] != 0 and arr[i] < min_value:
            # змінюємо мінімальний елемент
            min_value = arr[i]
            # змінюємо індекс мінімального елементу
            index = i

    return min_value, index


def find_route(matrix: Matrix):
    # маршрут обходу (починається завжди з 0 вершини)
    route = [0]
    # довжина маршруту
    route_value = 0

    # лічильник
    i = 0
    # масив з індексами виключень
    # зберігає вершини, в яких ми вже були
    except_indexes = [0]

    while len(route) < len(matrix):
        # рядок матриці
```

```
row = matrix[i]
# знаходимо мінімальний елемент і наступну вершину для обходу
min_value, next_point = find_min(row, except_indexes)
# змінюємо довжину маршруту
route_value += min_value
# додаємо вершину в маршрут
route.append(next_point)
# додаємо вершину в масив виключень
except_indexes.append(next_point)

# змінюємо лічильник
i = next_point
# додаємо до маршруту першу вершину (так як ми повинні в ній закінчити свій шлях)
route.append(0)

# передостанній елемент масиву зі шляхом
penultimate_element = route[len(route) - 2]
# останній елемент масиву зі шляхом
last_element = route[len(route) - 1]

# додаємо до довжини шляху довжину від останньої вершини до першої
route_value += matrix[penultimate_element][last_element]

return route, route_value
```

## Вихідний код

**main.py:**

```
from functions import *  
  
file_content = read_input_file("input.txt")  
matrix = parse_matrix(file_content)  
  
route, route_value = find_route(matrix)  
  
fill_output_file("output.txt", route, route_value)
```

**classes.py:**

```
class Matrix:
    def __init__(self, data: list[list[int]]):
        self.__data = data

    def __getitem__(self, index):
        return self.__data[index]

    def __len__(self):
        return len(self.__data)
```



## funtions.py:

```
from classes import *
from math import sqrt

def read_input_file(file_name: str):
    file = open(file_name)

    text = file.read()

    file.close()

    return text

def calculate_distance(x1: int, y1: int, x2: int, y2: int):
    # рахуємо відстань за формулою
    distance = sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

    return distance

def generate_empty_double_list(size: int):
    double_list = []

    for i in range(size):
        row = []
        for j in range(size):
            row.append(0)
        double_list.append(row)

    return double_list

def init_double_list(matrix_size: int, distances: list[str], double_list:
list[list[int]]):
    for i in range(matrix_size):
        for j in range(matrix_size):
            x1 = int(distances[i][0])
            y1 = int(distances[i][1])

            x2 = int(distances[j][0])
            y2 = int(distances[j][1])

            distance = calculate_distance(x1, y1, x2, y2)

            double_list[i][j] = distance
            double_list[j][i] = distance

def split_coordinates(matrix_size: int, coordinates: list[str]):

    for i in range(matrix_size):
```

```

        coordinates[i] = coordinates[i].split(' ')

def parse_matrix(text: str):
    # "розрізаємо" текст по символу нового рядка
    lines = text.split("\n")
    # розмір матриці
    matrix_size = int(lines[0])

    # генеруємо порожній двовимірний масив
    double_list = generate_empty_double_list(matrix_size)

    # отримуємо з тексту координати точок
    coordinates = lines[1:]

    # "розрізаємо" координати по пробілу
    split_coordinates(matrix_size, coordinates)

    # ініціалізуємо двовимірний масив
    # а саме, переводимо координати в матрицю, яка збурігає відстані між вершинами
    init_double_list(matrix_size, coordinates, double_list)

    # генеруємо об'єкт матриці на основі двовимірного масиву
    matrix = Matrix(double_list)

    return matrix

def find_min(arr: list[int], except_indexes: list[int]):
    # мінімальний елемент
    min_value = -1
    # індекс мінімального елементу
    index = -1

    # ініціалізуємо мінімальний елемент і його індекс
    # для цього ітеруємось по основному масиву
    for i in range(len(arr)):
        # ітеруємось по масиву виключень
        for j in range(len(except_indexes)):
            # якщо i != елементу в масиві виключені i != 0
            if i != except_indexes[j] and arr[i] != 0:
                # ініціалізуємо мінімальний елемент
                min_value = arr[i]
                # індекс мінімального елементу
                index = i

        # виходимо з циклу
        break

    # ітеруємось по основному масиву
    for i in range(1, len(arr)):
        # ітеруємось по масиву виключень
        for j in range(len(except_indexes)):
            # якщо i дорівнює елементу в масиві виключень, то пропускаємо цей хід

```

циклу

```
if i == except_indexes[j]:  
    continue
```

```
# якщо елемент масиву != 0 і елемент масиву менший за min value  
if arr[i] != 0 and arr[i] < min_value:  
    # змінюємо мінімальний елемент  
    min_value = arr[i]  
    # змінюємо індекс мінімального елементу  
    index = i
```

```
return min_value, index
```

```
def find_route(matrix: Matrix):
```

```
# маршрут обходу (починається завжди з 0 вершини)  
route = [0]  
# довжина маршруту  
route_value = 0
```

```
# лічильник  
i = 0  
# масив з індексами виключень  
# зберігає вершини, в яких ми вже були  
except_indexes = [0]
```

```
while len(route) < len(matrix):  
    # рядок матриці  
    row = matrix[i]  
    # знаходимо мінімальний елемент і наступну вершину для обходу  
    min_value, next_point = find_min(row, except_indexes)  
    # змінюємо довжину маршруту  
    route_value += min_value  
    # додаємо вершину в маршрут  
    route.append(next_point)  
    # додаємо вершину в масив виключень  
    except_indexes.append(next_point)
```

```
    # змінюємо лічильник  
    i = next_point
```

```
# додемо до маршруту першу вершину (так як ми вонинні в ній закінчити свій шлях)  
route.append(0)
```

```
# передостанній елемент масиву зі шляхом  
penultimate_element = route[len(route) - 2]  
# останній елемент масиву зі шляхом  
last_element = route[len(route) - 1]
```

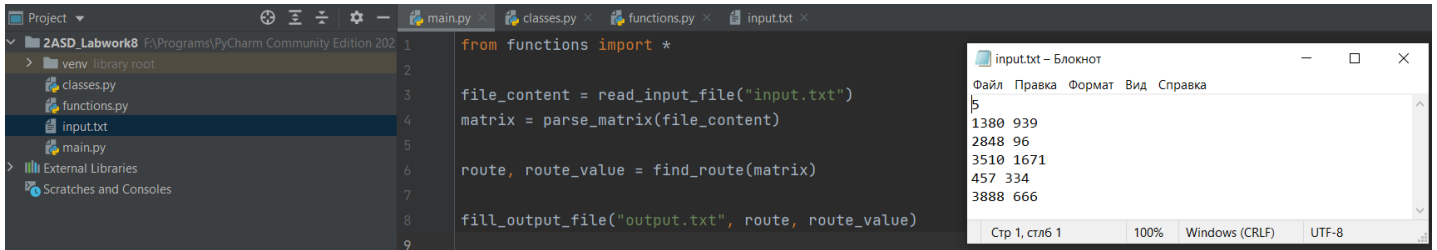
```
# додаємо до довжини шляху довжину від останньої вершини до першої  
route_value += matrix[penultimate_element][last_element]
```

```
return route, route_value
```

```
def fill_output_file(file_name: str, route: list[int], route_value: int):  
    text = f"{int(route_value)}\n"  
  
    for i in range(len(route)):  
        text += f"{route[i]} "  
  
    file = open(file_name, 'wt')  
    file.write(text)  
  
    file.close()
```

## Приклади роботи

Для тестування, оберемо файл, що вказаний в прикладі до задачі. Таким чином, ми зможемо порівняти відповіді і зробити висновок про правильність роботи програми.



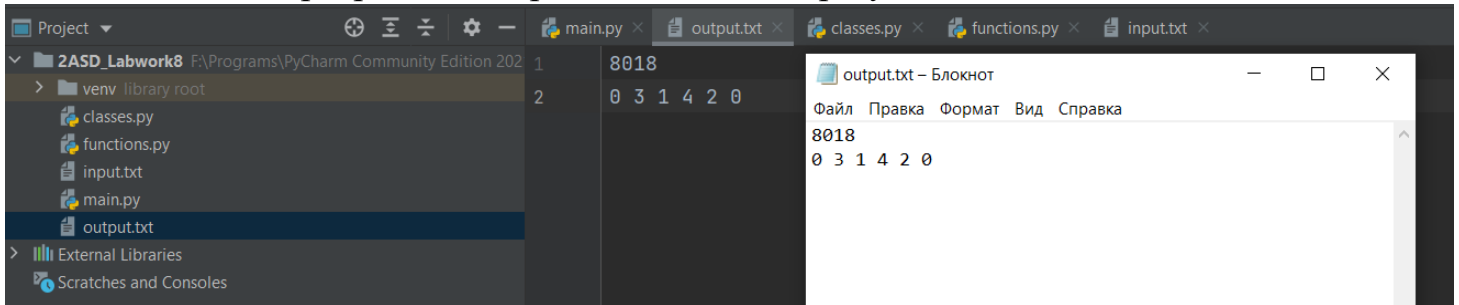
The screenshot shows the PyCharm IDE with the project '2ASD\_Labwork8'. The file explorer on the left shows the project structure. The main editor displays the code in `main.py`:

```
1 from functions import *
2
3 file_content = read_input_file("input.txt")
4 matrix = parse_matrix(file_content)
5
6 route, route_value = find_route(matrix)
7
8 fill_output_file("output.txt", route, route_value)
9
```

A separate window titled 'input.txt - Блокнот' shows the content of the input file:

```
5
1380 939
2848 96
3510 1671
457 334
3888 666
```

Після виконання програми ми отримали ось такий результат:



The screenshot shows the PyCharm IDE with the project '2ASD\_Labwork8'. The file explorer on the left shows the project structure. The main editor displays the code in `main.py` and `output.txt`.

The console output shows the result of the program execution:

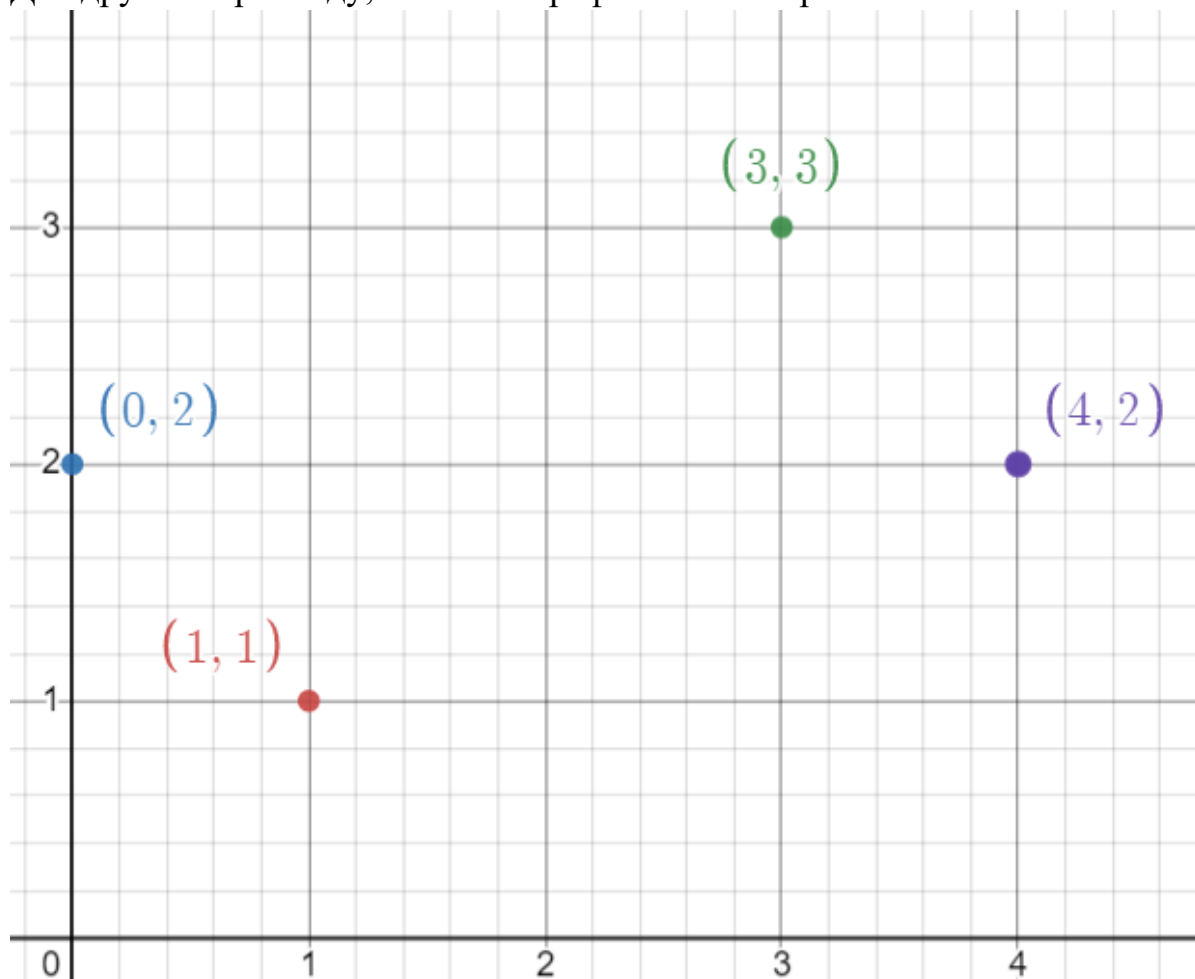
```
8018
0 3 1 4 2 0
```

A separate window titled 'output.txt - Блокнот' shows the content of the output file:

```
8018
0 3 1 4 2 0
```

Бачимо, що на цьому прикладі програма спрацювала правильно, так як відповіді збіглися.

Для другого прикладу, візьмемо граф з такими вершинами:



Подамо ці дані на вхід програмі:

```
Project ▾
2ASD_Labwork8 F:\Programs\PyCharm Community Edition 202
  > venv library root
    classes.py
    functions.py
    input.txt
    main.py
  > External Libraries
  Scratches and Consoles

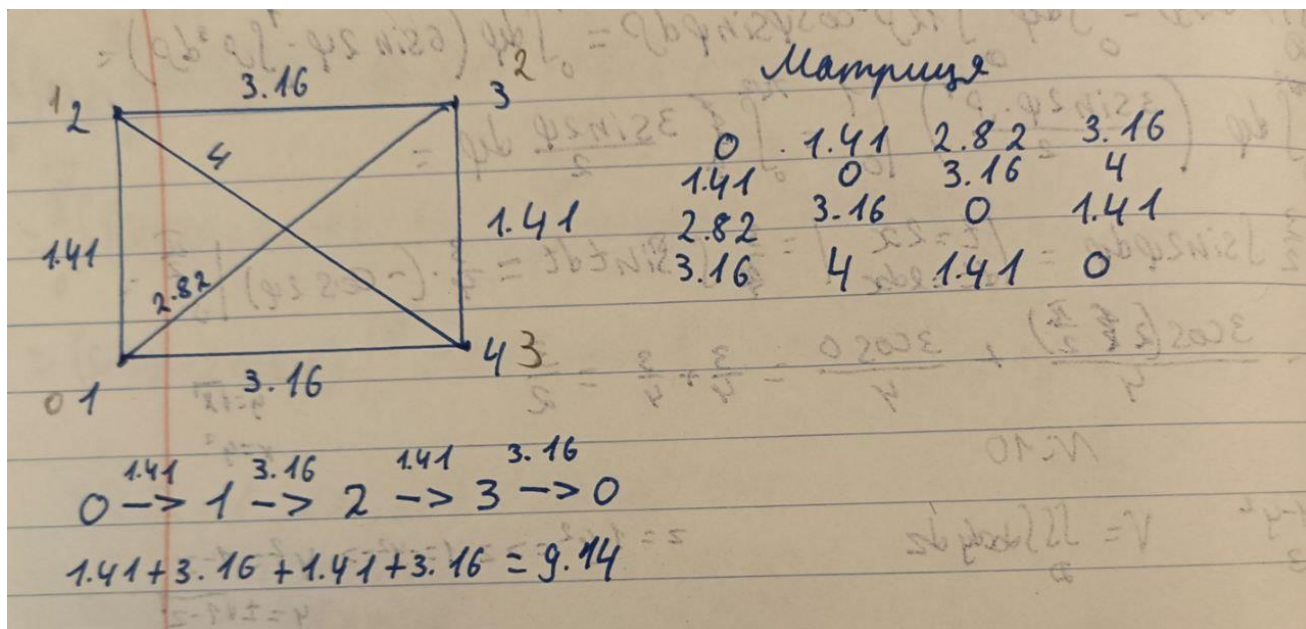
main.py × classes.py × functions.py × input.txt ×
1 4
2 1 1
3 0 2
4 3 3
5 4 2
```

Отримали ось такий результат:

```
Project ▾
2ASD_Labwork8 F:\Programs\PyCharm Community Edition 202
  > venv library root
    classes.py
    functions.py
    input.txt
    main.py
    output.txt

main.py × classes.py × functions.py × input.txt × output.txt ×
1 9
2 0 1 2 3 0
```

Перевіримо його вручну:



Бачимо, що обхід і довжина маршруту збіглись (в програмі довжина маршруту дорівнює 9, тому що ми тип даних float конвертуємо в int).

На двох прикладах програма коректно знайшла маршрут і довжину графу, отже, вона працює правильно.

## **Висновок**

Під час виконання цієї лабораторної роботи я вивчив основні підходи реалізації жадібного алгоритму на прикладі задачі комівояжера. В результаті виконання лабораторної роботи я отримав програму, яка знаходить найкоротший маршрут графу та його довжину за допомогою жадібного алгоритму.