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 №4 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

«Прикладні задачі теорії графів ч.2»
Варіант 3

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

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

Київ 2022

Мета лабораторної роботи

Мета роботи – вивчити додаткові прикладні алгоритми на графах та способи їх імплементації.

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

3	Пошук усіх найкоротших шляхів	Данцига (вбудований спосіб визначення шляхів)	Ортграф, матриця вагів

Код алгоритму

```
def search_all_shortest_ways(self):
    matrix_list = []
    path_matrix = generate_empty_path_matrix(len(self.__data))
    main_matrix_size = len(self.__data)

    for i in range(1, main_matrix_size + 1):
        empty_matrix = generate_empty_matrix(i)
        matrix_list.append(empty_matrix)

    for matrixes in range(1, main_matrix_size):
        m = len(matrix_list[matrixes]) - 1

        for i in range(len(matrix_list[matrixes])):
            for j in range(len(matrix_list[matrixes])):
                if i == j:
                    matrix_list[matrixes][i][j] = 0
                elif m == i:
                    values = []
                    lowest_value = float("inf")
                    lowest_value_index = -1

                    for k in range(m):
                        value = self.__data[i][k] + matrix_list[matrixes - 1][k][j]
                        values.append(value)

                        if value < lowest_value:
                            lowest_value = value
                            lowest_value_index = k

                    min_value = min(values)
                    matrix_list[matrixes][i][j] = min_value

                    if min_value < self.__data[i][j]:
                        path_matrix[i][
                            j] += f"({i + 1}->{lowest_value_index + 1})->({lowest_value_index +
1}->{j + 1})"
                    else:
                        path_matrix[i][j] += f"({i + 1}->{j + 1})"
                elif m == j:
                    values = []
                    lowest_value = float("inf")
                    lowest_value_index = -1

                    for k in range(m):
                        value = matrix_list[matrixes - 1][i][k] + self.__data[k][j]
                        values.append(value)

                        if value < lowest_value:
                            lowest_value = value
                            lowest_value_index = k

                    min_value = min(values)
                    matrix_list[matrixes][i][j] = min_value

                    if min_value < self.__data[i][j]:
```

```
                        path_matrix[i][j] += f"({i + 1}->{lowest_value_index + 1})-
>({lowest_value_index + 1}->{j+1})"
                    else:
                        path_matrix[i][j] += f"({i + 1}->{j+1})"

        for i in range(len(matrix_list[matrixes])):
            for j in range(len(matrix_list[matrixes])):
                if i != j and m != i and m != j:
                    value1 = matrix_list[matrixes - 1][i][j]
                    value2 = matrix_list[matrixes][i][m] + matrix_list[matrixes][m][j]

                    min_value = min(value1, value2)

                    matrix_list[matrixes][i][j] = min_value

                    if value2 < value1:
                        path_matrix[i][j] = f"({i + 1}->{m + 1})->({m + 1}->{j + 1})"


    self.__shortest_ways_matrix = matrix_list[len(matrix_list) - 1]
    self.__path_matrix = path_matrix
```

Програмна реалізація на мові Python

main.py

```
from classes import *

# створюємо матрицю
matrix = Matrix()
# виводимо її в консоль
matrix.show_matrix()

# шукаємо всі найкоротші шляхи
matrix.search_all_shortest_ways()

# виводимо в консоль матрицю усіх найкоротших шляхів
matrix.show_shortest_path_weight_matrix()
# виводимо в консоль усіх найкоротші шляхи
matrix.show_shortest_path_matrix()
```

function.py:

```
import random


def fill_matrix(matrix: list[list[int]]):
    size = int(input("Enter size of your matrix: "))

    mode = input("Would you like to generate matrix randomly or enter it manually?\nR - 
randomly, M - manually:")

    while mode != "r" and mode != "R" and mode != "m" and mode != "M":
        mode = input("Error, enter either M or R\nR - randomly, M - manually:")

    if mode == 'm' or mode == 'M':
        print("Now enter wight matrix of your matrix\n(if there's no path in vertex enter \"-
1\", on diagonal elemets enter \"0\"):")
        for i in range(size):
            row = []

            for j in range(size):
                element = int(input(f"Enter element ({i + 1},{j + 1}): "))
                if element == -1:
                    element = float("inf")
                row.append(element)

            matrix.append(row)

    elif mode == 'r' or mode == 'R':
        for i in range(size):
                row = []

                for j in range(size):
                    if i == j:
                        row.append(0)
                    else:
                        element = random.randint(0, 10)

                        row.append(element)

                matrix.append(row)

def generate_empty_matrix(size: int):
    matrix = []

    for i in range(size):
        row = []

        for j in range(size):
            row.append(0)

        matrix.append(row)

    return matrix

def generate_empty_path_matrix(size: int):
    matrix = []
```

```
for i in range(size):
    row = []

    for j in range(size):
        row.append("")

    matrix.append(row)

return matrix
```

classes.py:

```
from functions import fill_matrix, generate_empty_matrix, generate_empty_path_matrix


class Matrix:
    __data = []
    __shortest_ways_matrix = []
    __path_matrix = []

    def __init__(self):
        # заповнюємо матрицю за допомогою функції
        fill_matrix(self.__data)

    def show_matrix(self):
        print("Matrix:")
        for i in range(len(self.__data)):
            for j in range(len(self.__data)):
                print(f"{self.__data[i][j]} ", end='')
            print("\n", end='')
        print("\n", end='')

    def show_shortest_path_weight_matrix(self):
        print("Shortest path weight matrix:")
        for i in range(len(self.__shortest_ways_matrix)):
            for j in range(len(self.__shortest_ways_matrix)):
                print(f"{self.__shortest_ways_matrix[i][j]} ", end='')
            print("\n", end='')
        print("\n", end='')

    def show_shortest_path_matrix(self):
        print("Shortest path weight matrix:")
        for i in range(len(self.__path_matrix)):
            for j in range(len(self.__path_matrix)):
                if self.__path_matrix[i][j] == "":
                    print('%30s' % ('-'), end='')
                else:
                    print('%30s' % (self.__path_matrix[i][j]), end='')
            print("\n", end='')
        print("\n", end='')


    def search_all_shortest_ways(self):
        # ліст з матрицями розміру 1,2,3,...
        # потрібен для реалізації алгоритму Данцига
        matrix_list = []
        # порожня матриця
        path_matrix = generate_empty_path_matrix(len(self.__data))
        # розмір матриці
        main_matrix_size = len(self.__data)

        # створюємо порожні матриці (готуємо масиви для реалізації алгоритму Данцига)
        for i in range(1, main_matrix_size + 1):
            empty_matrix = generate_empty_matrix(i)
            matrix_list.append(empty_matrix)

        # алгоритм Данцига
```

```
        # лічильник для масиву matrix_list
        for matrices in range(1, main_matrix_size):
            # розмір кожної матриці
            m = len(matrix_list[matrices]) - 1

            # спочатку ітеруємо по крайнім елементам матриць
            for i in range(len(matrix_list[matrices])):
                for j in range(len(matrix_list[matrices])):
                    # якщо i == j, то залишаєм елемент нулем
                    if i == j:
                        matrix_list[matrices][i][j] = 0
                    # m == i, то обчислюємо елемент за відповідною формулою Данцига
                    elif m == i:
                        # значення
                        values = []
                        # найменше значення (для початку дорівнює нескінченності)
                        lowest_value = float("inf")
                        # індекс найменшого значення (для початку дорівнює -1)
                        lowest_value_index = -1

                        # ітеруємось по розміру матриці (потрібно для формули)
                        for k in range(m):
                            # рахуємо значення за відповідною формулою
                            value = self.__data[i][k] + matrix_list[matrices - 1][k][j]
                            # додаємо обчислене значення в масив
                            values.append(value)

                            # якщо значення менше за вже найменше значення
                            if value < lowest_value:
                                lowest_value = value
                                lowest_value_index = k

                        # знаходимо мінімальне значення
                        min_value = min(values)
                        # замінюємо елемент матриці найменшим значенням
                        matrix_list[matrices][i][j] = min_value

                        # заповнюємо матрицю шляху
                        if min_value < self.__data[i][j]:
                            path_matrix[i][j] += f"({i + 1}->{lowest_value_index + 1})-
>({lowest_value_index + 1}->{j + 1})"
                        else:
                            path_matrix[i][j] += f"({i + 1}->{j + 1})"
                    # m == j, то обчислюємо елемент за відповідною формулою Данцига
                    elif m == j:
                        # значення
                        values = []
                        # найменше значення (для початку дорівнює нескінченності)
                        lowest_value = float("inf")
                        # індекс найменшого значення (для початку дорівнює -1)
                        lowest_value_index = -1

                        # ітеруємось по розміру матриці (потрібно для формули)
                        for k in range(m):
                            # рахуємо значення за відповідною формулою
                            value = matrix_list[matrices - 1][i][k] + self.__data[k][j]
                            # додаємо обчислене значення в масив
                            values.append(value)
```

```
                        # якщо значення менше за вже найменше значення
                        if value < lowest_value:
                            lowest_value = value
                            lowest_value_index = k

                    # знаходимо мінімальне значення
                    min_value = min(values)
                    # замінюємо елемент матриці найменшим значенням
                    matrix_list[matrices][i][j] = min_value

                    # заповнюємо матрицю шляху
                    if min_value < self.__data[i][j]:
                        path_matrix[i][j] += f"({i + 1}->{lowest_value_index + 1})-
>({lowest_value_index + 1}->{j+1})"
                    else:
                        path_matrix[i][j] += f"({i + 1}->{j+1})"

        # теперь ітеруємо ще раз
        for i in range(len(matrix_list[matrices])):
            for j in range(len(matrix_list[matrices])):
                # але зараз обчислюємо не крайні елементи
                if i != j and m != i and m != j:
                    # значення 1
                    value1 = matrix_list[matrices - 1][i][j]
                    # значення 2
                    value2 = matrix_list[matrices][i][m] + matrix_list[matrices][m][j]

                    # знаходимо мінімальне значення
                    min_value = min(value1, value2)

                    matrix_list[matrices][i][j] = min_value

                    # заповнюємо матрицю шляху
                    if value2 < value1:
                        path_matrix[i][j] = f"({i + 1}->{m + 1})->({m + 1}->{j + 1})"


    self.__shortest_ways_matrix = matrix_list[len(matrix_list) - 1]
    self.__path_matrix = path_matrix
```

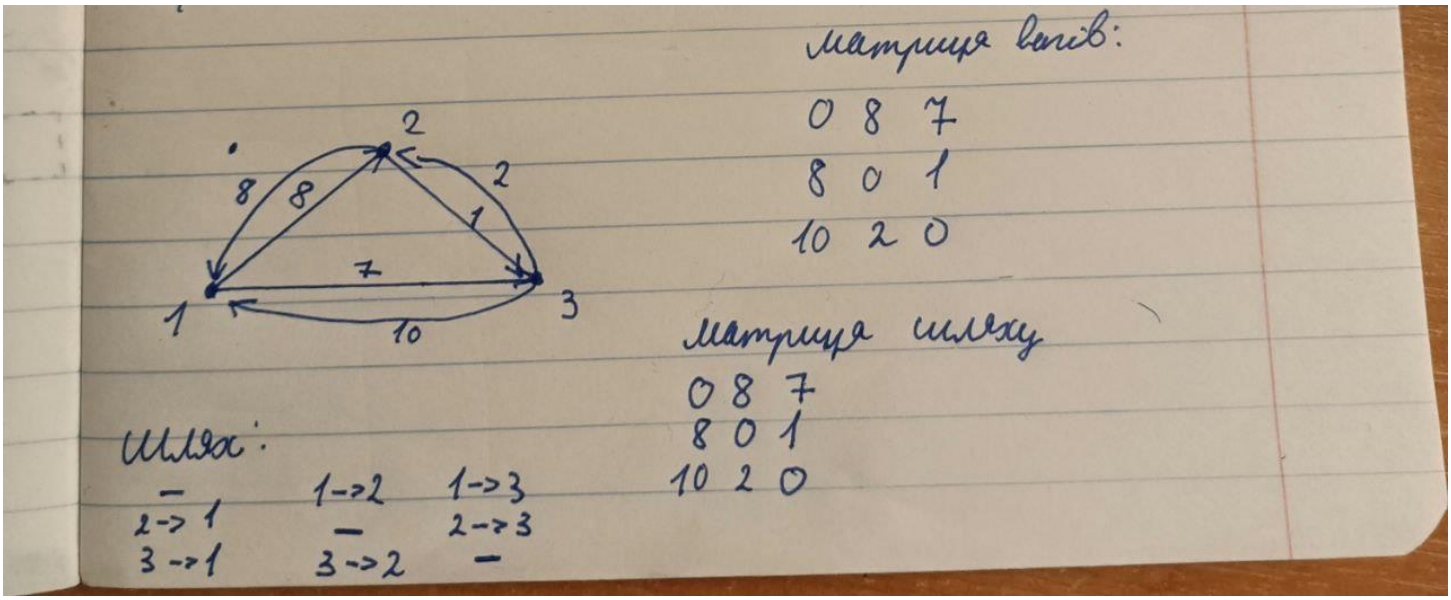
Перевірка алгоритму

Для початку згенеруємо матрицю випадковим чином:

Отримали ось такий результат:

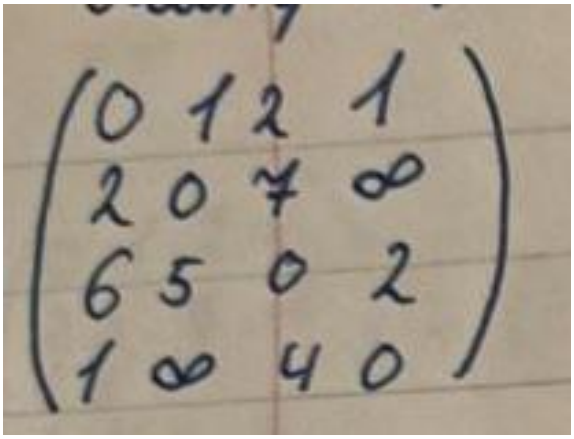
```
Enter size of your matrix: 3
Would you like to generate matrix randomly or enter it manually?
R - randomly, M - manually:r
Matrix:
0 8 7
8 0 1
10 2 0

Shortest path weight matrix:
0 8 7
8 0 1
10 2 0

Shortest path weight matrix:
                          -                    (1->2)                    (1->3)
                    (2->1)                          -                    (2->3)
                    (3->1)                    (3->2)                          -


Process finished with exit code 0
```

Перевіримо результат виконання програми вручну:



Бачимо, що програма правильно знайшла усі найкоротші шляхи.

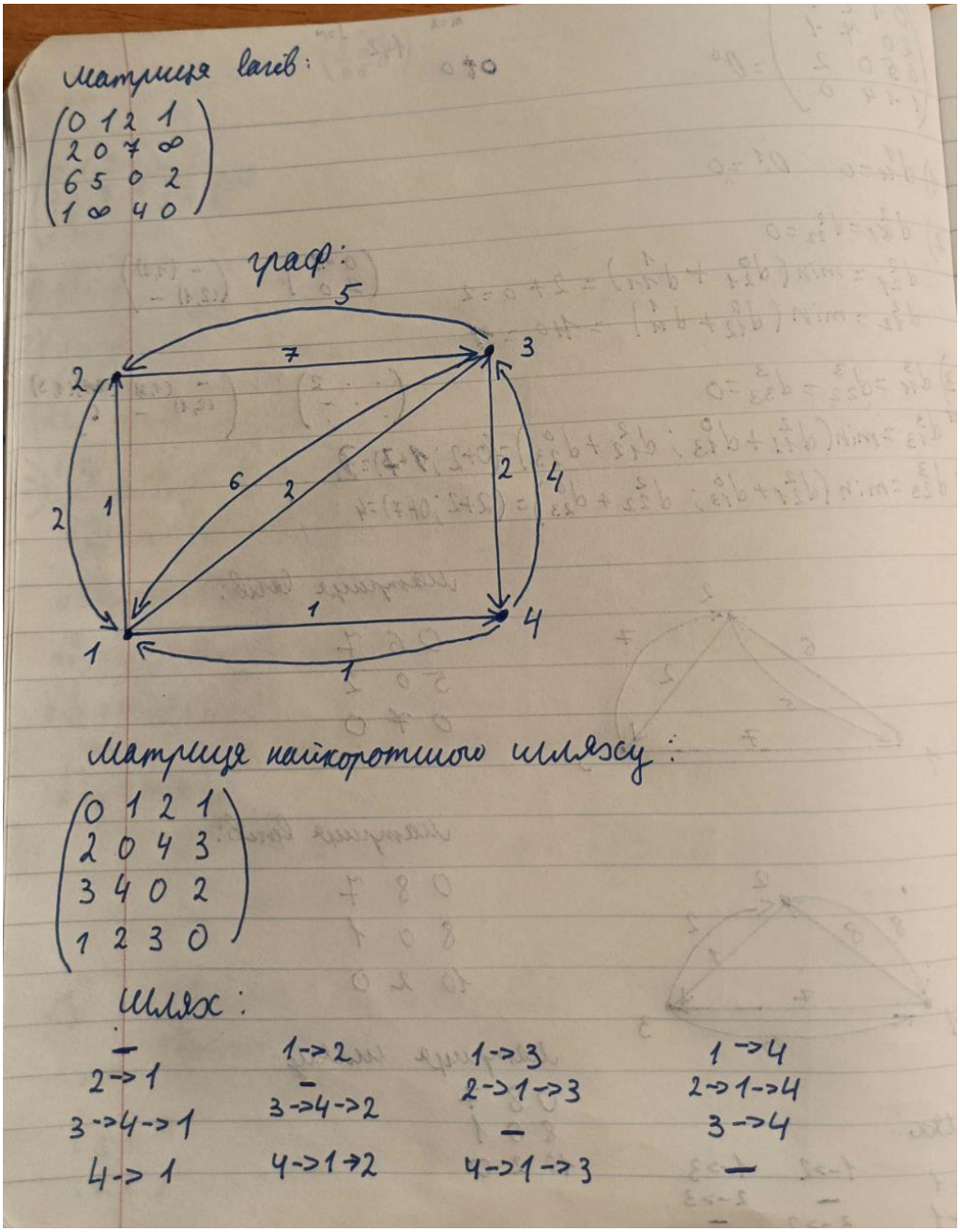
Тепер введемо власну матрицю:



Ми отримали наступний розв'язок усіх найкоротших шляхів:

```
Enter element (3,3): 0
Enter element (3,4): 2
Enter element (4,1): 1
Enter element (4,2): -1
Enter element (4,3): 4
Enter element (4,4): 0
Matrix:
0 1 2 1
2 0 7 inf
6 5 0 2
1 inf 4 0

Shortest path weight matrix:
0 1 2 1
2 0 4 3
3 4 0 2
1 2 3 0

Shortest path weight matrix:
                          -                     (1->2)                (1->3)                (1->4)
                    (2->1)                           -       (2->1)->(1->3)       (2->1)->(1->4)
          (3->4)->(4->1)          (3->4)->(4->2)                           -             (3->4)
                  (4->1)          (4->1)->(1->2)       (4->1)->(1->3)                           -


Process finished with exit code 0
```

Тепер перевіримо результат виконання програми вручну:



Бачимо, що усі розв'язки збіглись і програма правильно знайшла усі найкоротші шляхи. Отже, наша програма працює правильно.

Висновок

Під час виконання цієї лабораторної роботи я вивчив додаткові прикладні алгоритми на графах та способи їх імплементації. В результаті виконання лабораторної роботи я отримав програму, яка може знаходити усі найкоротші маршрути у графі методом Данцига. Під час тестування я переконався, що мої розв'язки задач і розв'язки програми збіглись, отже, вона працює правильно.