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

«Прикладні задачі теорії графів частина 1»

Варіант 5

Виконав студент ІП-15, Буяло Дмитро Олександрович
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

Лабораторна робота 3

Прикладні задачі теорії графів частина 1

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

Індивідуальне завдання

Варіант 5

Завдання

№	Задача	Алгоритм	Тип графу	Спосіб задання
5	Пошук найкоротшого шляху між парою вершин	Дейкстри	Орієнтований	Матриця вагів

1. Псевдокод алгоритму

```
Початок
    Введення start
    Повторити для і від 0 до topCount
        minDist[i] = sum
        vert[i] = 1
    Все повторити
    minDist[start] = 0
    Повторити
        minIndex = sum
        min = minIndex
        Повторити для і від 0 до topCount
            Якщо vert[i] = 1 та minDist[i] < min
             то
                min = minDist[i]
                minIndex = i
            Все якщо
        Якщо minIndex != sum
         то
            Повторити для і від 0 до topCount
                Якщо matrixWeight[minIndex][i] > 0 та
                                        min + matrixWeight[minIndex][i] < minDist[i]
                 то
                    minDist[i] = min + matrixWeight[minIndex][i]
                Все якщо
            Все повторити
            vert[minIndex] = 0
        Все якщо
    Поки minIndex < sum
    Все повторити
Кінець
```

2. Програмна реалізація алгоритму

2.1Вихідний код

```java
package com.company;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws FileNotFoundException {
        Scanner in = new Scanner(System.in);
        System.out.print("Generate automatically? [Y/N]: ");
        String yn = in.nextLine();
        int topCount, random;
        int max = 1, sum = 1; // sum to increase efficiency (not write Integer.MAX_VALUE)
        // max is for a beautiful output))
        int[][] matrixWeight;
        char yes = yn.charAt(0);
        /*--------------------Making matrix--------------------------*/
        // - Generating
        if (yes == 'y' || yes == 'Y' || yes == '+' || yes == '1') {
            System.out.println("You chose automatic generation!");
            System.out.print("Enter the number of graph vertices: ");
            topCount = in.nextInt();
            matrixWeight = new int[topCount][topCount];
            for (int i = 0; i < topCount; i++) {
                for (int j = 0; j < topCount; j++) {
                    if (i < j) {
                        random = (int) (Math.random() * 4);
                        if (random == 0) {
                            matrixWeight[i][j] = (int) (Math.random() * 51 + 1);
                        } else if (random == 1) {
                            matrixWeight[j][i] = (int) (Math.random() * 51 + 1);
                            matrixWeight[i][j] = 0;
                            if (max < matrixWeight[j][i]) max = matrixWeight[j][i];
                        } else if (random == 2) {
                            matrixWeight[i][j] = (int) (Math.random() * 51 + 1);
                            matrixWeight[j][i] = matrixWeight[i][j];
                            sum += matrixWeight[i][j];
                        } else {
                            matrixWeight[i][j] = 0;
                        }
                    }
                    if (max < matrixWeight[i][j]) max = matrixWeight[i][j];
                    sum += matrixWeight[i][j];
                }
            }
        }
        // - Reading
        else {
            System.out.println("You chose manual generation!");
            File file = new File( pathname: "Graph.txt");
            Scanner scan = new Scanner(file);
            topCount = scan.nextInt();
            System.out.println("\nYour graph has " + topCount + " vertices");
```

```
        matrixWeight = new int[topCount][topCount];
        for (int i = 0; i < topCount; i++) {
            for (int j = 0; j < topCount; j++) {
                matrixWeight[i][j] = scan.nextInt();
                if (max < matrixWeight[i][j]) max = matrixWeight[i][j];
                sum += matrixWeight[i][j];
            }
        }
    }
    /*--------------------Matrix done--------------------------*/
    /*--------------------Output matrix------------------------*/
    int len;
    int maxLen = (int) (Math.log10(max) + 1);
    for (int i = 0; i < topCount; i++) {
        for (int j = 0; j < topCount; j++) {
            if (matrixWeight[i][j] == 0) len = 1;
            else len = (int) (Math.log10(matrixWeight[i][j]) + 1);
            System.out.print(matrixWeight[i][j]);
            for (int k = 0; k < (maxLen - len + 3); k++) {
                System.out.print(" ");
            }
        }
        System.out.println();
    }
    /*--------------------Wrote matrix-------------------------*/
    /*--------------------Find the shortest distance to each--------------------------*
    System.out.print("\nEnter start vertex: ");
    int start = in.nextInt() - 1;
    int[] minDist = new int[topCount];
    int[] vert = new int[topCount];
    int minIndex, min;
    for (int i = 0; i < topCount; i++) {
        minDist[i] = sum;
        vert[i] = 1;
    }
    minDist[start] = 0;
    do {
        minIndex = sum;
        min = minIndex;
        for (int i = 0; i < topCount; i++) {
            if ((vert[i] == 1) && (minDist[i] < min)) {
                min = minDist[i];
                minIndex = i;
            }
        }
        if (minIndex != sum) {
            for (int i = 0; i < topCount; i++) {
                if (matrixWeight[minIndex][i] > 0) {
                    if (min + matrixWeight[minIndex][i] < minDist[i]) {
                        minDist[i] = min + matrixWeight[minIndex][i];
                    }
                }
            }
            vert[minIndex] = 0;
```

```
            }
        } while (minIndex < sum);
        /*---------------------Found-------------------------*/
        /*---------------------Output distances--------------------------*/
        System.out.println("\nShortest distance to each vertex:");
        for (int i = 0; i < topCount; i++) {
            System.out.print(minDist[i] + "   ");
        }
        /*---------------------Wrote distances---------------------------*/
        /*---------------------Looking for a way to our top---------------------------*/
        System.out.print("\n\nEnter end vertex: ");
        int end = in.nextInt() - 1;
        int endToOutput = end;
        int[] way = new int[topCount];
        int k = 1; // индекс предыдущей вершины
        int weight = minDist[end]; // вес конечной вершины
        way[0] = end + 1;

        while (end != start)
        {
            for (int i = 0; i < topCount; i++) { // просматриваем все вершины
                if (matrixWeight[i][end] != 0)
                {
                    if (weight - matrixWeight[i][end] == minDist[i])
                    {
                        weight = weight - matrixWeight[i][end]; // сохраняем новый вес
                        end = i;       // сохраняем предыдущую вершину
                        way[k] = i + 1;
                        k++;
                    }
                }
            }
        }
        /*---------------------Found a way--------------------------*/
        /*---------------------Output way---------------------------*/
        System.out.println("\nShortest way " + (start + 1) + " -> " + (endToOutput + 1) + ":");
        for (int i = k - 1; i >= 0; i--) {
            System.out.print(way[i] + "  ");
        }
    }
}
```

2.2Приклад роботи

На рисунках 2.1 і 2.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

```
Generate automatically? [Y/N]: 1
You chose automatic generation!
Enter the number of graph vertices: 7
0    0    11   37   7    11   0
0    0    28   0    0    14   0
0    28   0    0    43   0    0
37   0    0    0    10   0    0
7    0    43   0    0    26   41
11   14   0    40   0    0    0
0    16   0    29   41   29   0

Enter start vertex: 3

Shortest distance to each vertex:
50   28   0   82   43   42   84

Enter end vertex: 4

Shortest way 3 -> 4:
3  2  6  4
Process finished with exit code 0
```

Рисунок 2.1 – Приклад роботи на 7 вершинах

```
Enter the number of graph vertices: 15
0    5    47   0    19   28   3    41   35   0    6    0    29   0    15
0    0    0    45   0    29   0    29   0    16   30   31   0    46   0
47   45   0    45   47   13   0    0    0    50   0    29   16   51   0
25   45   0    0    0    0    0    0    18   44   15   27   0    21   0
0    0    0    0    0    0    0    0    30   0    0    8    36   42   0
0    0    13   4    0    0    50   47   5    0    0    33   0    0    0
3    1    0    33   39   50   0    38   0    31   0    33   44   16   0
41   0    26   0    0    0    0    0    0    0    0    0    0    0    41
0    6    12   18   0    0    0    50   0    0    36   0    0    0    0
0    16   0    44   16   0    0    45   48   0    0    0    0    36   48
0    30   20   0    41   17   0    47   36   0    0    17   0    0    32
41   0    0    27   0    0    0    30   41   0    17   0    0    0    17
29   31   0    40   36   0    44   31   19   35   34   11   0    0    50
8    46   51   21   0    37   0    15   0    0    51   18   0    0    42
15   0    0    17   26   0    41   0    0    0    32   0    50   0    0

Enter start vertex: 6

Shortest distance to each vertex:
29   11   13   4   43   0   32   40   5   27   19   31   29   25   44

Enter end vertex: 10

Shortest way 6 -> 10:
6  9  2  10
```

Рисунок 2.2 – Приклад роботи на 15 вершинах

На рисунках 2.31 і 2.4 виконана перевірка для наведених графів на 7 і 15 вершин відповідно.

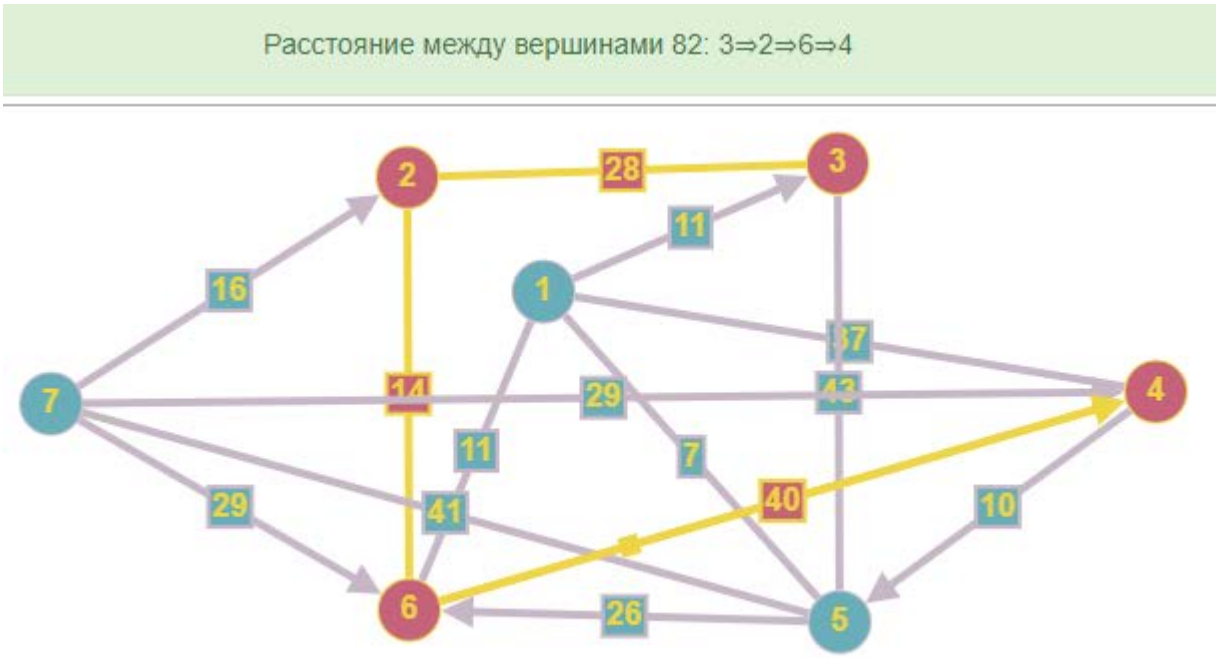

Рисунок 2.3 – Перевірка роботи для графа на 7 вершинах

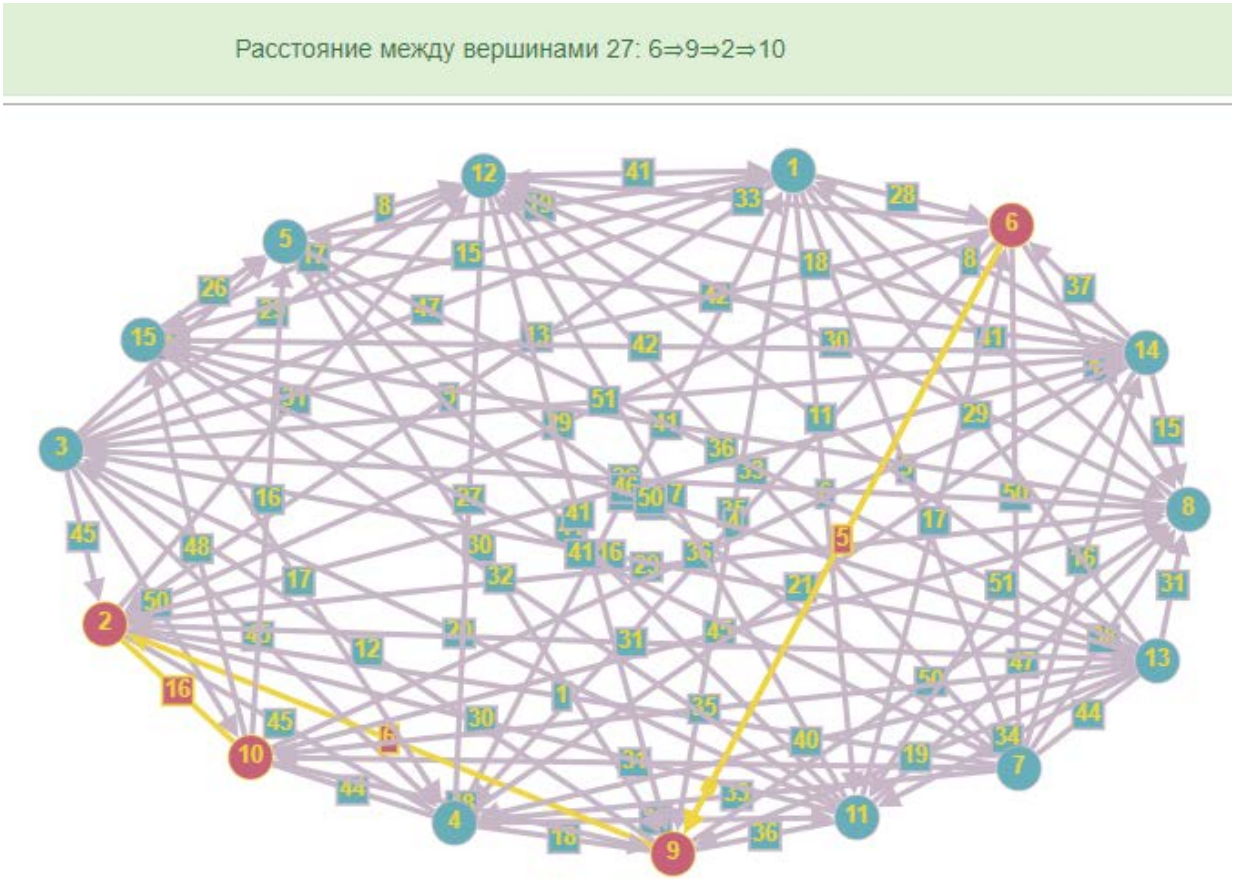

Рисунок 2.4 – Перевірка роботи для графа на 15 вершинах

3. Розв’язання задачі вручну

На рисунку 3.1 наведено ручне розв’язання задачі для графу на 9 вершин.

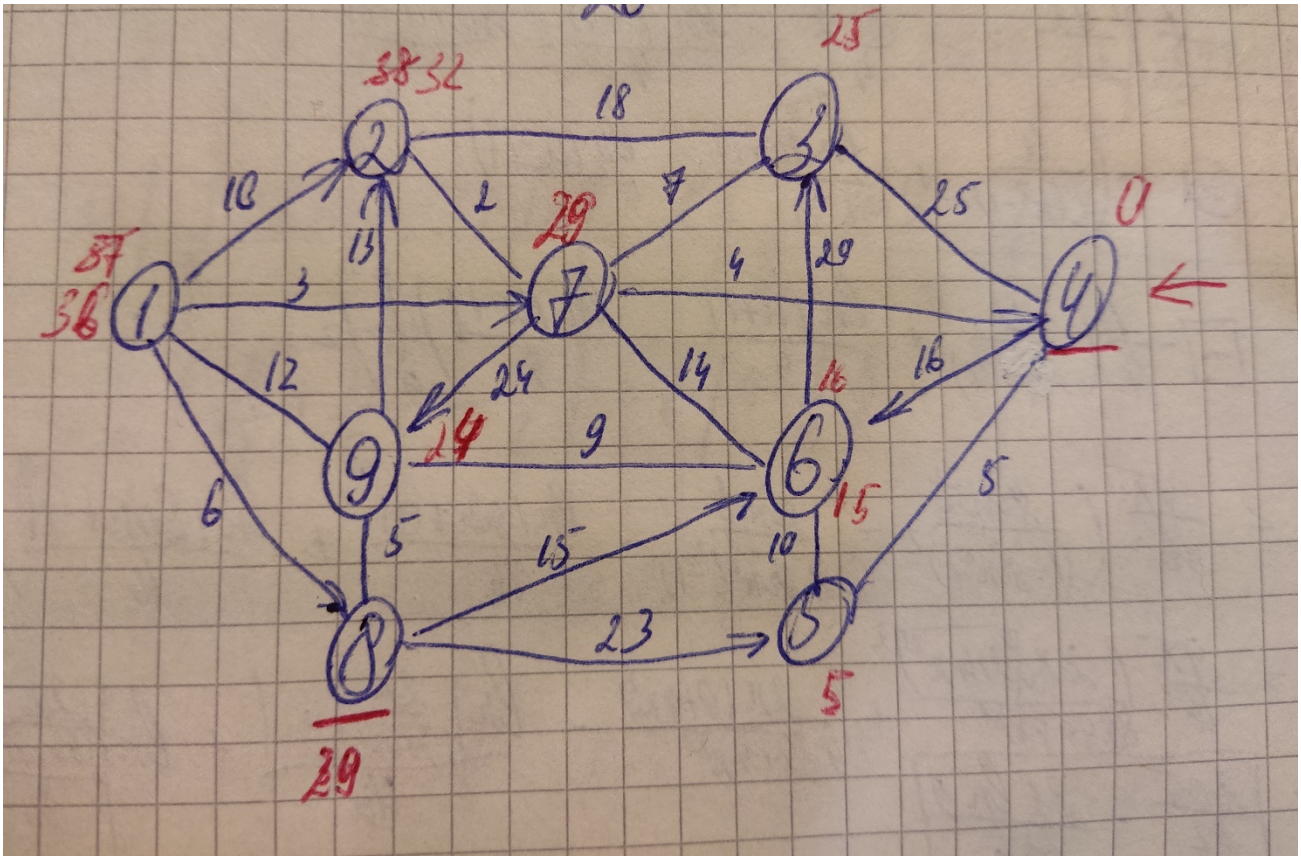

Рисунок 3.1 – Ручне розв’язання

На рисунку 3.2 наведено умову для цієї задачі

Graph.txt – Блокнот

Файл Правка Формат Вид Справка

```
9
0       10      0       0       0       0       3       6       12
0       0       18      0       0       0       2       0       0
0       18      0       25      0       0       0       0       7
0       0       25      0       5       16      0       0       0
0       0       0       5       0       10      0       0       0
0       0       20      0       10      0       14      0       9
0       2       0       4       0       14      0       0       24
0       0       0       0       23      15      0       0       5
12      13      0       0       0       9       24      5       0
```

Рисунок 3.2 – Умова

На рисунку 3.3 наведено розв’язок цієї задачі створеною програмою

```
Generate automatically? [Y/N]: 2
You chose manual generation!

Your graph has 9 vertices
0    10   0    0    0    0    3    6    12
0    0    18   0    0    0    2    0    0
0    18   0    25   0    0    0    0    7
0    0    25   0    5    16   0    0    0
0    0    0    5    0    10   0    0    0
0    0    20   0    10   0    14   0    9
0    2    0    4    0    14   0    0    24
0    0    0    0    23   15   0    0    5
12   13   0    0    0    9    24   5    0

Enter start vertex: 4

Shortest distance to each vertex:
36   31   25   0   5   15   29   29   24

Enter end vertex: 8

Shortest way 4 -> 8:
4  5  6  9  8
Process finished with exit code 0
```

Рисунок 3.3 – Програмне рішення

На рисунку 3.4 виконана перевірка для наведеного графу на 9 вершин.

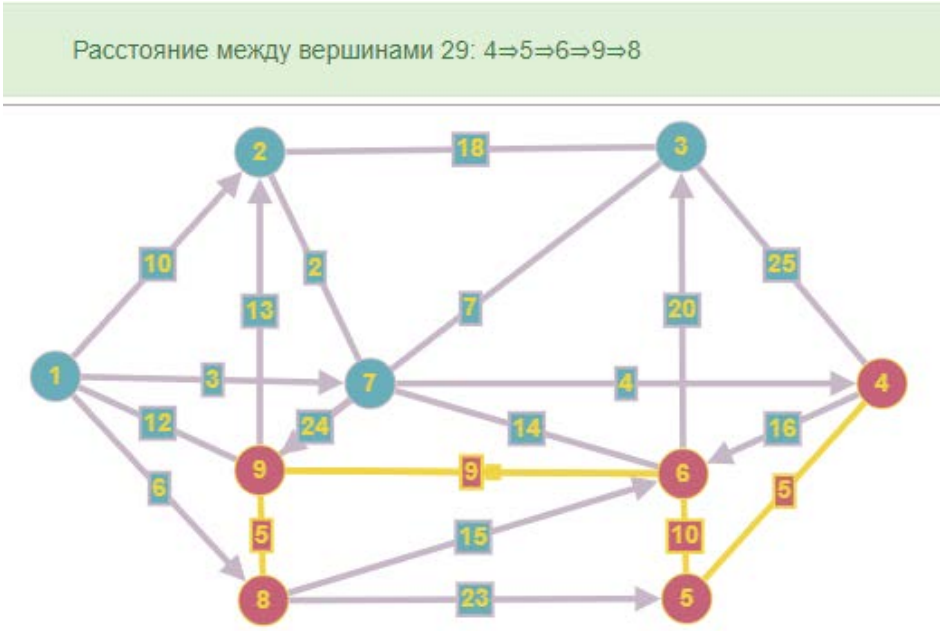

Рисунок 3.4 – Перевірка роботи для графа на 9 вершинах

ВИСНОВОК

При виконанні третьої лабораторної роботи, вивчили алгоритм Дейкстри для графів та спосіб його імплементації, задаючи вершини графу матрицею вагів, яка може генеруватися автоматично чи задаватися вручну. Алгоритм Дейкстри знаходить найкоротший шлях до кожної з вершин.

Програмне розв’язання подібне до ручного, але дозволяє проводити розрахунки набагато швидше.