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 частина 2»

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 4

Прикладні задачі теорії графів частина 2

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

Індивідуальне завдання

Варіант 5

Завдання

№	Задача	Алгоритм	Спосіб задання мережі
5	Задача збільшуючого ланцюга	За означенням	Ортграф, матриця вагів, типи дуг

Виконання

1. Псевдокод алгоритму

```
Початок
    Повторити для i від 1 до n
        p[0] = i
        j0 = 0
        Повторити
        Поки true
            used[j0] = true
            i0 = p[j0]
            Повторити для j від 0 до m
                Якщо used[i]
                 то
                    u[p[j]] += k
                    v[j] -= k
                 Інакше
                    minv[j] -= k
                Все якщо
            Все повторити
        Все повторити
    Все повторити
Кінець
```

2. Програмна реалізація алгоритму

2.1Вихідний код Main.java:

```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; // max is for a beautiful output))
        int[][] matrixWeight;
        int[][] matrixToOutput;
        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];
            matrixToOutput = 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() * 18);
                        if (random == 0) {
                            matrixWeight[i][j] = (int) (Math.random() * 101 + 1);
                            matrixToOutput[i][j] = matrixWeight[i][j];
                        } else if (random == 1) {
                            matrixWeight[j][i] = (int) (Math.random() * 101 + 1);
                            matrixWeight[i][j] = 0;
                            matrixToOutput[i][j] = matrixWeight[i][j];
                            if (max < matrixWeight[j][i]) max = matrixWeight[j][i];
                        } else if (random > 1 && random < 16) {
                            matrixWeight[i][j] = (int) (Math.random() * 101 + 1);
                            matrixWeight[j][i] = matrixWeight[i][j];
                            matrixToOutput[i][j] = matrixWeight[i][j];
                        } else {
                            matrixWeight[i][j] = 0;
                            matrixToOutput[i][j] = 0;
                        }
                    }
                    if (max < matrixWeight[i][j]) max = 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];
            matrixToOutput = new int[topCount][topCount];
            for (int i = 0; i < topCount; i++) {
                for (int j = 0; j < topCount; j++) {
                    matrixWeight[i][j] = scan.nextInt();
                    matrixToOutput[i][j] = matrixWeight[i][j];
                    if (max < matrixWeight[i][j]) max = 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---------------------------*/
        /*---------------------Using algorithm---------------------------*/
        Algorithm a = new Algorithm(matrixWeight);
        int[][] assignment = a.findOptimalAssignment();

        System.out.println();

        if (assignment.length > 0) {
            for (int[] ints : assignment) {
                System.out.print("Col" + ints[0] + " => Row" + ints[1] +
                        " (" + matrixToOutput[ints[1]][ints[0]] + ")");
                System.out.println();
            }
        }
        else {
            System.out.println("no assignment found!");
        }
    }
}
```

Algorithm.java:

```java
package com.company;
import java.util.Arrays;
import java.util.LinkedHashSet;
import java.util.Set;

public class Algorithm {
    int[][] matrix;

    int[] row, col, rowCovered, colCovered, zeroRow;

    public Algorithm(int[][] matrix) {
        this.matrix = matrix;
        row = new int[matrix.length];
        col = new int[matrix[0].length];

        rowCovered = new int[matrix.length];
        colCovered = new int[matrix[0].length];
        zeroRow = new int[matrix.length];
        Arrays.fill(zeroRow, val: -1);
        Arrays.fill(row, val: -1);
        Arrays.fill(col, val: -1);
    }

    public int[][] findOptimalAssignment() {
        stepOne();
        stepTwo();
        stepThree();

        while (!allColumnsAreCovered()) {
            int[] mainZero = stepFour();
            while (mainZero == null) {
                stepSeven();
                mainZero = stepFour();
            }
            if (row[mainZero[0]] == -1) {
                stepSix(mainZero);
                stepThree();
            } else {
                // step five
                rowCovered[mainZero[0]] = 1;  // cover row of mainZero
                colCovered[row[mainZero[0]]] = 0;  // uncover column of mainZero
                stepSeven();
            }
        }

        int[][] optimalAssignment = new int[matrix.length][];
        for (int i = 0; i < col.length; i++) {
            optimalAssignment[i] = new int[]{i, col[i]};
        }
        return optimalAssignment;
    }

    private boolean allColumnsAreCovered() {
        for (int i : colCovered) {
```

```
            if (i == 0) {
                return false;
            }
        }
        return true;
    }

    private void stepOne() {
        for (int i = 0; i < matrix.length; i++) {
            // find the min value of the current row
            int currentRowMin = Integer.MAX_VALUE;
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] < currentRowMin) {
                    currentRowMin = matrix[i][j];
                }
            }
            // subtract min value from each element of the current row
            for (int k = 0; k < matrix[i].length; k++) {
                matrix[i][k] -= currentRowMin;
            }
        }

        for (int i = 0; i < matrix[0].length; i++) {
            // find the min value of the current column
            int currentColMin = Integer.MAX_VALUE;
            for (int[] ints : matrix) {
                if (ints[i] < currentColMin) {
                    currentColMin = ints[i];
                }
            }
            // subtract min value from each element of the current column
            for (int k = 0; k < matrix.length; k++) {
                matrix[k][i] -= currentColMin;
            }
        }
    }

    private void stepTwo() {
        int[] rowHasSquare = new int[matrix.length];
        int[] colHasSquare = new int[matrix[0].length];

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                if (matrix[i][j] == 0 && rowHasSquare[i] == 0 && colHasSquare[j] == 0) {
                    rowHasSquare[i] = 1;
                    colHasSquare[j] = 1;
                    row[i] = j;
                    col[j] = i;
                }
            }
        }
    }

    private void stepThree() {
```

```
109         for (int i = 0; i < col.length; i++) {
110             colCovered[i] = col[i] != -1 ? 1 : 0;
111         }
112     }
113
114     private void stepSeven() {
115         // Find the smallest uncovered value in the matrix
116         int minUncoveredValue = Integer.MAX_VALUE;
117         for (int i = 0; i < matrix.length; i++) {
118             if (rowCovered[i] == 1) {
119                 continue;
120             }
121             for (int j = 0; j < matrix[0].length; j++) {
122                 if (colCovered[j] == 0 && matrix[i][j] < minUncoveredValue) {
123                     minUncoveredValue = matrix[i][j];
124                 }
125             }
126         }
127
128         if (minUncoveredValue > 0) {
129             for (int i = 0; i < matrix.length; i++) {
130                 for (int j = 0; j < matrix[0].length; j++) {
131                     if (rowCovered[i] == 1 && colCovered[j] == 1) {
132                         matrix[i][j] += minUncoveredValue;
133                     } else if (rowCovered[i] == 0 && colCovered[j] == 0) {
134                         matrix[i][j] -= minUncoveredValue;
135                     }
136                 }
137             }
138         }
139     }
140
141 @   private int[] stepFour() {
142         for (int i = 0; i < matrix.length; i++) {
143             if (rowCovered[i] == 0) {
144                 for (int j = 0; j < matrix[i].length; j++) {
145                     if (matrix[i][j] == 0 && colCovered[j] == 0) {
146                         zeroRow[i] = j;
147                         return new int[]{i, j};
148                     }
149                 }
150             }
151         }
152         return null;
153     }
154
155 @   private void stepSix(int[] mainZero) {
156         int i;
157         int j = mainZero[1];
158
159         Set<int[]> K = new LinkedHashSet<>();
160         K.add(mainZero);
161         boolean found = false;
162         do {
```

```
163             if (col[j] != -1) {
164                 K.add(new int[]{col[j], j});
165                 found = true;
166             } else {
167                 found = false;
168             }
169             if (!found) {
170                 break;
171             }
172             i = col[j];
173             j = zeroRow[i];
174             if (j != -1) {
175                 K.add(new int[]{i, j});
176                 found = true;
177             } else {
178                 found = false;
179             }
180
181         } while (found);
182
183         for (int[] zero : K) {
184             if (col[zero[1]] == zero[0]) {
185                 col[zero[1]] = -1;
186                 row[zero[0]] = -1;
187             }
188             if (zeroRow[zero[0]] == zero[1]) {
189                 row[zero[0]] = zero[1];
190                 col[zero[1]] = zero[0];
191             }
192         }
193
194         Arrays.fill(zeroRow,  val: -1);
195         Arrays.fill(rowCovered,  val: 0);
196         Arrays.fill(colCovered,  val: 0);
197     }
198 }
```

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]: yes
You chose automatic generation!
Enter the number of graph vertices: 7
64   26   84   86   70   71   39
89   37   34   66   45   1    71
63   27   59   67   37   13   52
82   2    22   68   19   79   23
6    61   56   17   60   81   54
43   46   24   45   90   94   29
31   41   65   16   76   30   63

Col1 => Row5 (6)
Col2 => Row4 (2)
Col3 => Row6 (24)
Col4 => Row7 (16)
Col5 => Row3 (37)
Col6 => Row2 (1)
Col7 => Row1 (39)

Process finished with exit code 0
```

Рисунок 2.1 – Приклад роботи на 7 вершинах

```
Generate automatically? [Y/N]: y
You chose automatic generation!
Enter the number of graph vertices: 15
65   50   69   37   79   11   86   49   14   11   5    17   64   22   54
45   80   90   69   32   49   92   87   18   45   83   56   17   52   1
11   43   62   89   35   78   27   2    1    31   3    5    74   2    51
58   44   48   80   14   33   49   7    28   62   54   82   6    44   69
43   77   62   72   54   84   34   4    7    45   61   26   12   67   72
101  18   32   74   94   68   65   40   97   3    60   100  59   12   10
62   95   9    21   27   77   72   58   30   26   84   46   39   77   56
96   27   56   66   15   96   58   31   17   35   93   51   28   21   70
13   70   61   6    2    89   38   70   65   51   32   69   32   66   96
101  9    34   47   57   80   66   12   40   37   57   75   90   46   82
18   4    47   69   37   1    76   46   66   16   37   52   31   59   83
87   55   58   60   61   57   73   76   48   70   80   99   22   71   69
25   55   52   39   20   5    84   17   56   19   35   72   24   67   54
39   97   97   19   43   82   2    61   29   89   82   74   11   46   78
64   67   48   72   77   10   76   22   68   13   35   39   45   38   85
```

```
Col1 => Row11 (18)
Col2 => Row10 (9)
Col3 => Row7 (9)
Col4 => Row9 (6)
Col5 => Row8 (15)
Col6 => Row13 (5)
Col7 => Row14 (2)
Col8 => Row4 (7)
Col9 => Row5 (7)
Col10 => Row15 (13)
Col11 => Row1 (5)
Col12 => Row3 (5)
Col13 => Row12 (22)
Col14 => Row6 (12)
Col15 => Row2 (1)

Process finished with exit code 0
```

Рисунок 2.2 – Приклад роботи на 15 вершинах

3. Розв’язання задачі вручну

На рисунку 3.1 наведено розв’язання задачі на 5 вершин вручну.

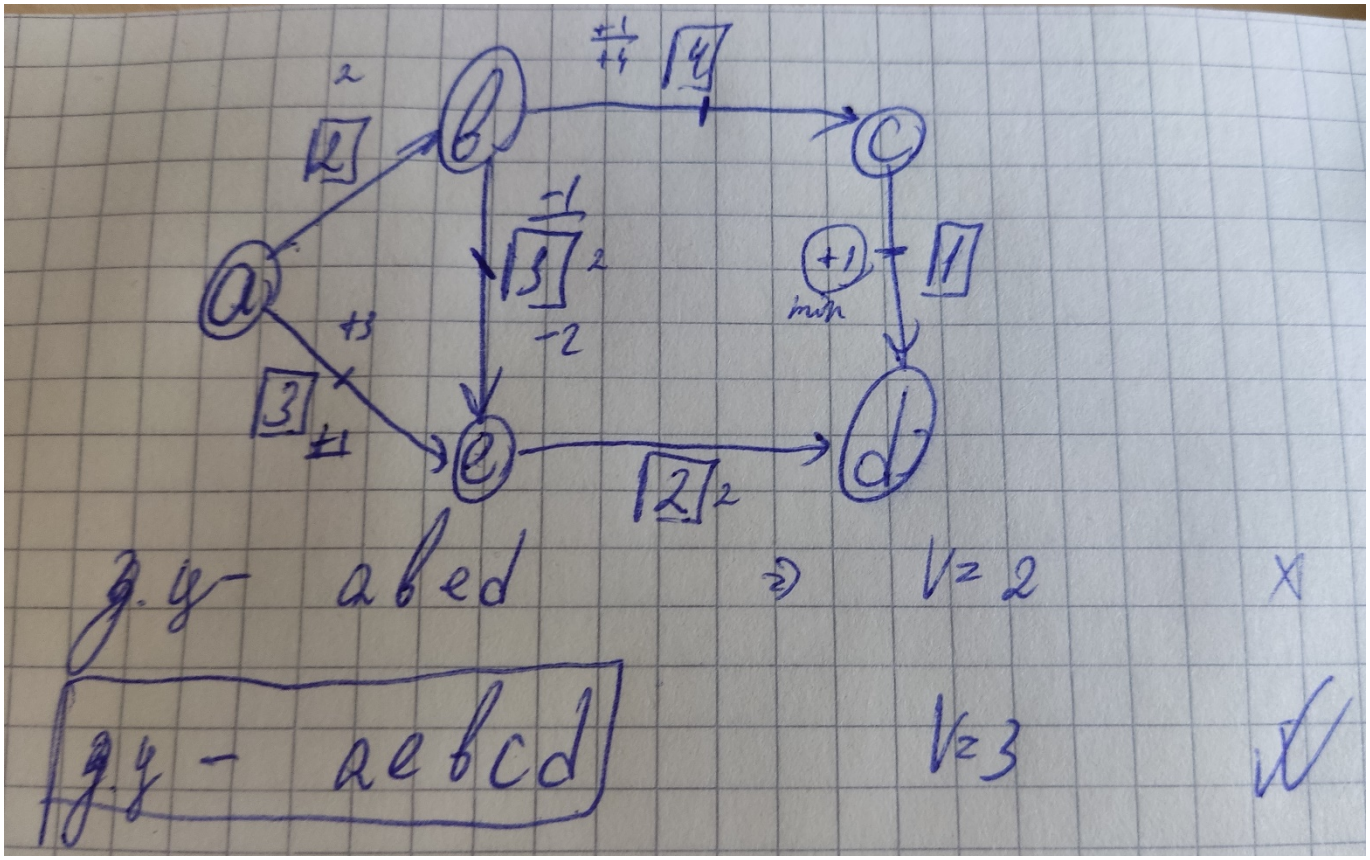

Рисунок 3.1 – Ручне розв’язання

ВИСНОВОК

При виконанні четвертої лабораторної роботи, вивчили алгоритм знаходження збільшуючого потоку для графів та спосіб його імплементації, задаючи вершини графу матрицею вагів, яка може генеруватися автоматично чи задаватися вручну.

Програмне розв’язання діє по чітко заданому алгоритму та може бути використано для графа на велику кількість вершин, а за допомогою ручного розв’язання можливо швидко знайти розв’язок лише для графа з декількома вершинами.