Міністерство освіти і науки України
Національний технічний університет України „КПІ”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки
інформації та управління

ЗВІТ

до лабораторної роботи № 1

з дисципліни “Основи Web-програмування”

Виконав студент *ІП-61 Кушка Михайло Олександрович*

(№ групи, прізвище, ім’я, по батькові)

Прийняв *Ліщук К. І.*

(посада, прізвище, ім’я, по батькові)

Київ 2018

ЗМІСТ

1. ПОСТАНОВКА ЗАДАЧІ

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

a) дозволяти користувачу визначати кількість вершин графа самостійно з консолі;
b) дозволяти користувачу вводити довільні матриці (списки суміжності) різної розмірності самостійно з консолі;
c) мати можливість генерації довільної матриці (списку суміжності) з консолі;
d) виводи на екран результат

В некотором городе есть метро, состоящее из N(1 <N< 1000) станций и M(M <500000) линий, соединяющих их. Каждая линия обеспечивает проезд между какими-то двумя станциями в обе стороны. Между любой парой станций проведено не более одной линии. Сеть метро построена таким образом, чтобы с каждой станции можно было проехать на каждую (возможно, через промежуточные станции). Назовем это свойство связностью метро.

В связи с изобретением принципиально нового вида транспорта метро стало убыточным, и его работу решили прекратить. На заседании мэрии города было постановлено закрывать каждый год по одной станции, но так, чтобы связность метро каждый раз сохранялась. При закрытии какой-либо станции линии, ведущие от этой станции к другим, естественно, тоже перестают функционировать.

Необходимо по введенной информации о сети метро разработать какой-либо порядок закрытия станций, при котором метро всегда будет оставаться связным.

2. РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

```
Choose an action
        1 - Generate random matrix
        2 - Enter adjacency matrix
1
=== Generate random matrix ===
Enter number of stations: 5
Adjacency matrix:
1 0 1 1 0
0 0 0 1 1
1 0 0 1 0
1 1 1 1 0
0 1 0 0 1
Stations closing order:
1 3 2 4 5
```

```
Choose an action
        1 - Generate random matrix
        2 - Enter adjacency matrix
2
=== Enter adjacency matrix ===
Enter number of stations: 3
Enter comma-separated values row by row
Enter one row: 1, 0, 1
Enter one row: 0, 0, 1
Enter one row: a, b, c
All of elements must be 0 or 1. Try again.
1, 1, 1, 1
Number of elements should be equal to 3, but you have 4. Try again.
1,1,1
Adjacency matrix:
1 0 1
0 0 1
1 1 1
Stations closing order:
2 1 3
```

3. КОД ПРОГРАМИ

```
using System;

namespace lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[,] first = {
                {0, 0, 0, 0, 1, 0, 0},
                {0, 0, 1, 0, 1, 0, 0},
                {0, 1, 0, 1, 1, 0, 0},
                {0, 0, 1, 0, 1, 1, 0},
                {1, 1, 1, 1, 0, 0, 0},
                {0, 0, 0, 1, 0, 0, 1},
                {0, 0, 0, 0, 0, 1, 0},
            };

            int[,] second = {
                {0, 0, 1, 1, 0},
                {0, 0, 0, 0, 1},
                {1, 0, 0, 1, 1},
                {1, 0, 1, 0, 1},
                {0, 1, 1, 1, 0},
            };

            int[,] third = {
                {0, 1, 0, 0},
                {1, 0, 1, 0},
                {0, 1, 0, 1},
                {0, 0, 1, 0},
            };

            // Input matrix from the keyboard.
            UserInput input = new UserInput();
            input.start();
            input.printAdjacencyMatrix();
            int[,] arr = input.getAdjacencyMatrix();

            // Correctly close stations.
            SubwayMap subway = new SubwayMap(arr);
            subway.closeStations();
            subway.getClosingOrder();
        }
    }
```

```
    class SubwayMap
    {
        private static int numOfStations; // Number of stations in the subway
        private static int[,] connections; // Number of connection between stations in
the subway
        private const int N = 1000; // Max number of stations
        private const int M = 500000; // Max number of connections between stations
        private int[] removingOrder; // The order of removing the station
        private int removeStationCounter; // Counter of how many stations has been
removed

        /*
        Init number of the subway stations, number of connection between stations
        and stations removing order.
         */
        public SubwayMap(int[,] _connections) {
            int _numOfStations = _connections.GetLength(0);
            if (_numOfStations > 0 && _numOfStations < N  && _connections.GetLength(0)
>= 0 && _connections.GetLength(0) < M)
            {
                numOfStations = _numOfStations;
                connections = _connections;
                removingOrder = new int[numOfStations];
            }
            else
                // If number of stations or number of connections are out of range.
                throw new Exception("Uncorrect input values.");
        }

        /*
        Main function of the program, which closes subway stations in the right order.
        */
        public void closeStations() {
            bool noLeafs = true;
            int iterationRemovedNum = 0;

            while (numOfStations > 0)
            {
                int n = connections.GetLength(0);

                // Remove all leafs from the graph.
                for (int i = 0; i < n; i++)
                {
                    if (isLeaf(i))
                    {
                        removeNode(i);
                        n--;
```

```
                        iterationRemovedNum++;
                        noLeafs = false;
                    }
                }

                // If there is no leafs in the graph than remove any node (first in
this case).
                if (noLeafs)
                    removeNode(0);
                noLeafs = true;
            }

        }

        /*
        Get resulting stations closing ordrer.
        */
        public void getClosingOrder()
        {
            Console.WriteLine("Stations closing order:");
            foreach(int elem in removingOrder)
                Console.Write(elem + " ");
            Console.WriteLine();
        }

        /*
        Check is station has less than 2 connections.
        */
        private bool isLeaf(int nodeNumber)
        {
            int n = connections.GetLength(0);

            // Error handling.
            if (nodeNumber < 0 || nodeNumber >= n)
                throw new Exception("Index of leaf is out of connections range.");

            int[] oneNodeConnections = getLine(nodeNumber, connections);
            int sum = getArraySum(oneNodeConnections);

            // Node is a leaf if it has one or less connections.
            if (sum <= 1)
                return true;

            return false;
        }

        /*
        Remove station from the subway's map.
```

```
        */
        private void removeNode(int nodeIndex)
        {
            // nodeNumber is real number of the node, which we want to remove.
            // Cause we store in removingOrder numbers starting from 1 we add 1
            // and also add number of shifts in the adjacency matrix.
            int nodeNumber = nodeIndex + 1 +
numOfRemovedNodesLessThanCurrent(nodeIndex);
            while (contains(nodeNumber, removingOrder))
                nodeNumber += 1;

            removingOrder[removeStationCounter] = nodeNumber;
            removeStationCounter++;


            // Remove i-th row and i-th column from the adjacency matrix.
            connections = removeRow(nodeIndex, connections);
            connections = removeColumn(nodeIndex, connections);

            numOfStations--;
        }

        /*
        Calculate number of currently removed nodes which are less than current node.
        */
        private int numOfRemovedNodesLessThanCurrent(int index)
        {
            int counter = 0;
            foreach(int node in removingOrder)
                if (node < index+1 && node != 0)
                    counter++;

            return counter;
        }

        /*
        Remove row from two-dimensional array.
        */
        private int[,] removeRow(int index, int[,] arr)
        {
            int n = arr.GetLength(0);
            int m = arr.GetLength(1);
            var newArr = new int[n-1, m];
            int counter = 0;

            // Error handling.
            if (index < 0 || index > m)
                throw new Exception("Index is out of range.");
```

```
        for (int i = 0; i < n; i++)
        {
            if (i == index) continue;
            for (int j = 0; j < m; j++)
            {
                newArr[counter, j] = arr[i, j];
            }
            counter++;
        }

        return newArr;
    }

    /*
    Remove column from two-dimensional array.
    */
    private int[,] removeColumn(int index, int[,] arr)
    {
        int n = arr.GetLength(0);
        int m = arr.GetLength(1);
        var newArr = new int[n, m-1];
        int counter = 0;

        // Error handling.
        if (index < 0 || index > n)
            throw new Exception("Index is out of range.");

        for (int i = 0; i < n; i++)
        {
            counter = 0;
            for (int j = 0; j < m; j++)
            {
                if (j == index) continue;
                newArr[i, counter] = arr[i, j];
                counter++;
            }
        }

        return newArr;
    }

    /*
    Check whether element is in the array.
    */
    private bool contains(int searchElem, int[] arr)
    {
        foreach(int elem in arr)
```

```
                if (elem == searchElem)
                    return true;

            return false;
        }

        /*
        Returns row from two-dimensional array.
        */
        private int[] getLine(int index, int[,] arr)
        {
            int n = arr.GetLength(0);
            int m = arr.GetLength(1);
            int[] oneLine = new int[m];

            // Error handling
            if (index < 0 || index > m)
                throw new Exception("Index is out of range.");

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    if (i != index) continue;
                    oneLine[j] = arr[i, j];
                }
            }

            return oneLine;
        }

        /*
        Get sum of all elements in one-dimensional array.
        */
        private int getArraySum(int[] arr)
        {
            int sum = 0;
            foreach(int elem in arr)
                sum += elem;

            return sum;
        }
    }

    class UserInput
    {
        private int[,] matrix;
```

```
        /*
        Main method of the class, which can generate a matrix depends of the user's
choices.
        */
        public void start()
        {
            string option;
            bool correctInput = false;

            while (!correctInput)
            {
                Console.WriteLine(
                    "Choose an action"
                    + "\n\t1 - Generate random matrix"
                    + "\n\t2 - Enter adjacency matrix"
                );
                option = Console.ReadLine();

                switch(option)
                {
                    case("1"):
                        generateRandomMatrix();
                        correctInput = true;
                        break;
                    case("2"):
                        enterAdjacencyMatrix();
                        correctInput = true;
                        break;
                    default:
                        Console.WriteLine("Incorrect input, try again.");
                        break;
                }
            }
        }

        /*
        Generate random adjacency matrix depending on the it's size.
        */
        public void generateRandomMatrix()
        {
            Console.WriteLine("=== Generate random matrix ===");

            // Get size of the matrix.
            int n = getNumberOfStations();
            matrix = new int[n, n];

            Random rand = new Random();
```

```
            for (int i = 0; i < n; i++)
            {
                for (int j = i; j < n; j++)
                {
                    // Random 0 or 1.
                    int randomNumber = rand.Next(0, 2);
                    matrix[j, i] = matrix[i, j] = randomNumber;
                }
            }
        }

        /*
        Get number of stations from the user's input.
        */
        private int getNumberOfStations()
        {
            string input;
            bool isNumeric = false;
            bool correctInput = false;
            int number = -1;

            while (!correctInput)
            {
                Console.Write("Enter number of stations: ");
                input = Console.ReadLine();
                isNumeric = int.TryParse(input, out number);
                if (!isNumeric)
                {
                    Console.WriteLine("Please enter a number");
                }
                else
                {
                    if (number < 2 || number > 999)
                        Console.WriteLine("Number of stations must be in range (2,
999). Try again.");
                    else
                        correctInput = true;
                }
            }

            return number;
        }

        /*
        Display adjacency matrix.
        */
        public void printAdjacencyMatrix()
        {
```

```
            int n = matrix.GetLength(0);

            Console.WriteLine("Adjacency matrix:");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    Console.Write(matrix[i, j] + " ");
                }
                Console.WriteLine();
            }
        }

        /*
        Get adjacency matrix.
        */
        public int[,] getAdjacencyMatrix()
        {
            return matrix;
        }

        /*
        Enter adjacency marix from keyboard and check is it symmetric or not.
        */
        private void enterAdjacencyMatrix()
        {
            Console.WriteLine("=== Enter adjacency matrix ===");

            int n = getNumberOfStations();
            matrix = new int[n, n];
            bool symmetric = false;

            while (!symmetric)
            {
                fillMaritxFromKeyboard(ref matrix, n);

                if (IsMatrixSymmetric(matrix))
                {
                    symmetric = true;
                }
                else
                {
                    Console.WriteLine("\nYour matrix must be symmetric. Try to enter
another matrix.");
                }
            }
        }
```

```
        /*
        Check is matrix symmetric or not.
        */
        private bool IsMatrixSymmetric(int[,] martix)
        {
            int n = martix.GetLength(0);

            if (n != martix.GetLength(1))
                return false;

            for (int i = 0; i < n; i++)
            {
                for (int j = i; j < n; j++)
                {
                    if (martix[i, j] != martix[j, i])
                        return false;
                }
            }

            return true;
        }

        /*
        Input matrix from keyboard and handle all errors.
        */
        private void fillMaritxFromKeyboard(ref int[,] matrix, int n)
        {
            string inputString;
            string[] tokens;
            bool correctInput;
            bool isNumeric;
            int number;

            Console.WriteLine("Enter comma-separated values row by row");
            for (int i = 0; i < n; i++)
            {
                correctInput = false;
                Console.Write("Enter one row: ");
                while (!correctInput)
                {
                    inputString = Console.ReadLine();
                    tokens = inputString.Split(',');

                    if (tokens.Length != n)
                    {
                        Console.WriteLine("Number of elements should be equal to " + n
+ ", but you have " + tokens.Length + ". Try again.");
                        continue;
```

```
                    }

                    correctInput = true;
                    for (int j = 0; j < tokens.Length; j++)
                    {
                        isNumeric = int.TryParse(tokens[j], out number);

                        if (!isNumeric || (number != 0 && number != 1))
                        {
                            correctInput = false;
                        }
                        else
                        {
                            matrix[i, j] = number;
                        }
                    }

                    if (!correctInput)
                        Console.WriteLine("All of elements must be 0 or 1. Try
again.");
                }
            }
        }

    }
}
```