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

Кафедра автоматизованих систем обробки
інформації та управління

Протокол

з основ Web-програмування № 1

Виконав студент	*ІП-63 Карпа Маркіян Володимирович* (№ групи, прізвище, ім’я, по батькові)
Номер залікової книжки та курс	6314, другий курс

Київ 2018

ЗМІСТ

1 ПОСТАНОВКА ЗАДАЧІ..... 3

2 ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ..... 4

3 ТЕКСТИ ПРОГРАМНОГО КОДУ 6

1 ПОСТАНОВКА ЗАДАЧІ

Постановка задачі до комп'ютерного практикуму № 4

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

Варіант 10

Андрей работает системным администратором и планирует создание новой сети в своей компании. Всего будет N хабов, они будут соединены друг с другом с помощью кабелей. Поскольку каждый сотрудник компании должен иметь доступ ко всей сети, каждый хаб должен быть достижим от любого другого хаба — возможно, через несколько промежуточных хабов. Поскольку имеются кабели различных типов и короткие кабели дешевле, требуется сделать такой план сети (соединения хабов), чтобы максимальная длина одного кабеля была как можно меньшей. Есть еще одна проблема — не каждую пару хабов можно непосредственно соединять по причине проблем совместимости и геометрических ограничений здания. Андрей снабдит вас всей необходимой информацией о возможных соединениях хабов.

Вы должны помочь Андрею найти способ соединения хабов, который удовлетворит всем указанным выше условиям.

Входные данные:

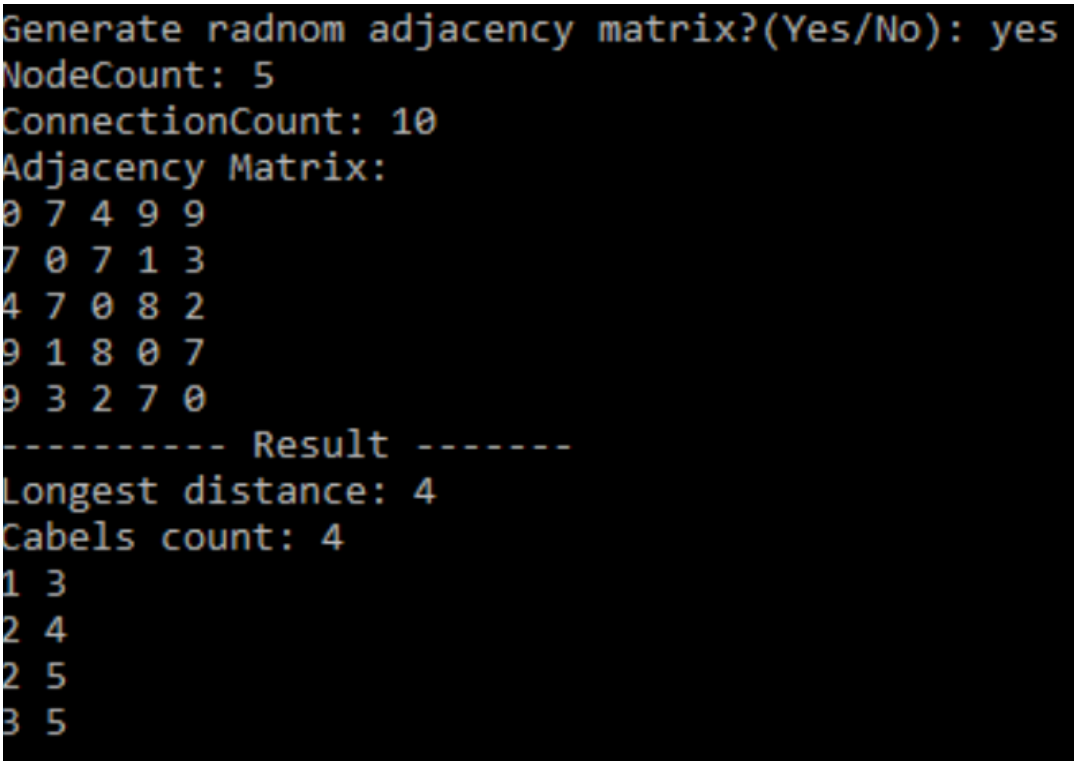
А) два целых числа: N — количество хабов в сети ($1 < N < 1000$) и M — количество возможных соединений хабов ($M < 15\ 000$). Все хабы пронумерованы от 1 до N

Б) Информация о возможных соединениях — номера двух хабов, которые могут быть соединены, и длина кабеля, который требуется, чтобы соединить их. Эта длина — натуральное число. Существует не более одного способа соединить каждую пару хабов. Хаб не может быть присоединен сам к себе. Всегда существует хотя бы один способ соединить все хабы.

Вывод:

Сначала выведите максимальную длину одного кабеля в вашем плане соединений (это величина, которую вы должны минимизировать). Затем выведите свой план: сначала выведите P — количество кабелей, которые вы использовали, затем выведите P пар целых чисел — номера хабов, непосредственно соединенных в вашем плане кабелями..

2 ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



```
Generate radnom adjacency matrix?(Yes/No): no
NodeCount: 7
ConnectionCount: 11
---< AdjacencyMatrix[from, to]= weight >---
AdjacencyMatrix[1,2]= 7
AdjacencyMatrix[1,3]= 8
AdjacencyMatrix[1,4]= 5
AdjacencyMatrix[1,5]= 0
AdjacencyMatrix[1,6]= 0
AdjacencyMatrix[1,7]= 0
AdjacencyMatrix[2,3]= 8
AdjacencyMatrix[2,4]= 9
AdjacencyMatrix[2,5]= 7
AdjacencyMatrix[2,6]= 0
AdjacencyMatrix[2,7]= 0
AdjacencyMatrix[3,4]= 0
AdjacencyMatrix[3,5]= 5
AdjacencyMatrix[3,6]= 0
AdjacencyMatrix[3,7]= 0
AdjacencyMatrix[4,5]= 15
AdjacencyMatrix[4,6]= 6
AdjacencyMatrix[4,7]= 0
AdjacencyMatrix[5,6]= 8
AdjacencyMatrix[5,7]= 9
AdjacencyMatrix[6,7]= 11
Adjacency Matrix:
0 7 8 5 0 0 0
7 0 8 9 7 0 0
8 8 0 0 5 0 0
5 9 0 0 15 6 0
0 7 5 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
---------- Result -------
Longest distance: 9
Cabels count: 6
1 2
1 4
2 5
3 5
4 6
5 7
```

3 ТЕКСТИ ПРОГРАМНОГО КОДУ

Файл Graph.cs

```
using System;

namespace Lab1
{
    class Graph
    {
        public int[,] AdjacencyMatrix { get; set; }
        public int NodeCount { get; set; }
        public int ConnectionCount { get; set; }

        public Graph(int nodeCount, int connectionCount)
        {
            NodeCount = nodeCount;
            ConnectionCount = connectionCount;

            AdjacencyMatrix = new int[nodeCount,nodeCount];
        }

        public void CreateAdjacencyMatrix(bool isRandom)
        {
            if (isRandom)
            {
                // Firstly, we make sure that all nodes will be connected with at least one node
                Random random = new Random();
                int currConnectionCount = 0;

                while (currConnectionCount < ConnectionCount)
                {
                    for (int i = 0; i < NodeCount; i++)
                    {
                        int randomNode = random.Next(i, NodeCount);
                        if (randomNode != i && AdjacencyMatrix[i,randomNode] == 0)
                        {
                            int randomWeight = random.Next(1, 10);
                            AdjacencyMatrix[i, randomNode] = randomWeight;
                            AdjacencyMatrix[randomNode, i] = randomWeight;
                            currConnectionCount++;
                        }

                        if(currConnectionCount >= ConnectionCount)
                        {
                            break;
                        }
                    }
                }
            }
            else
            {
                Console.WriteLine("---< AdjacencyMatrix[from, to]= weight >---");
                for(int i = 0; i < NodeCount; i++)
                {
                    for(int j = i + 1; j < NodeCount; j++)
                    {
                        Console.Write("AdjacencyMatrix[" + (i + 1).ToString() + ',' + (j +
1).ToString() + "]= ");

                        int weight;
                        if (!int.TryParse(Console.ReadLine(), out weight))
```

```
                    {
                        throw new Exception("You have to write a single number!");
                    }

                    AdjacencyMatrix[i, j] = weight;
                    AdjacencyMatrix[j, i] = weight;
                }
            }
        }
    }

    public void PrintAdjacencyMatrix()
    {
        Console.WriteLine("Adjacency Matrix:");
        for(int i = 0; i < NodeCount; i++)
        {
            for(int j = 0; j < NodeCount; j++)
            {
                Console.Write(string.Format("{0} ", AdjacencyMatrix[i, j]));
            }
            Console.WriteLine();
        }
    }

    // Find the minimum spanning tree using Kruskal's algorithm
    public int[,] FindSpanningTree() {
        int[,] result = new int[NodeCount, NodeCount];
        int[] nodes = new int[NodeCount];

        // Initialize result and nodes arrays
        for(int i = 0; i < NodeCount; i++)
        {
            for(int j = 0;j < NodeCount; j++)
            {
                result[i, j] = Int32.MaxValue;
            }

            nodes[i] = i;
        }

        int nodeA = 0, nodeB = 0;
        int currNodeCount = 1;

        while(currNodeCount < NodeCount)
        {
            int min = Int32.MaxValue;
            for(int i = 0; i < NodeCount; i++)
            {
                for(int j = 0; j< NodeCount; j++)
                {
                    if (AdjacencyMatrix[i, j] != 0 && min > AdjacencyMatrix[i,j] && nodes[i]
!= nodes[j])
                    {
                        min = AdjacencyMatrix[i, j];
                        nodeA = i;
                        nodeB = j;
                    }
                }
            }

            if(nodes[nodeA] != nodes[nodeB])
            {
                result[nodeA, nodeB] = min;
                result[nodeB, nodeA] = min;
```

```
            int temp = nodes[nodeB];
            nodes[nodeB] = nodes[nodeA];
            for(int k = 0; k < NodeCount; k++)
            {
                if(nodes[k] == temp)
                {
                    nodes[k] = nodes[nodeA];
                }
            }
            currNodeCount++;
        }
    }
    return result;
}

public int FindSpanningTreeLongestEdge(int[,] spanningTree)
{
    int max = Int32.MinValue;

    for (int i = 0; i < NodeCount; i++)
    {
        for (int j = i; j < NodeCount; j++)
        {
            if (spanningTree[i, j] != Int32.MaxValue)
            {
                if(spanningTree[i,j] > max)
                {
                    max = spanningTree[i, j];
                }
            }
        }
    }
    return max;
}

public int FindSpanningTreeNodeCount(int[,] spanningTree)
{
    int count = 0;

    for (int i = 0; i < NodeCount; i++)
    {
        for (int j = i; j < NodeCount; j++)
        {
            if (spanningTree[i, j] != Int32.MaxValue)
            {
                count++;
            }
        }
    }
    return count;
}

public void PrintSpanningTree(int[,] spanningTree)
{
    for (int i = 0; i < NodeCount; i++)
    {
        for (int j = i; j < NodeCount; j++)
        {
            if (spanningTree[i, j] != Int32.MaxValue)
            {
                Console.Write(string.Format("{0} {1}\n", i + 1, j + 1));
            }
        }
    }
}
```

```
    }
}
```

Файл Program.cs

```
using System;

namespace Lab1
{
    class Program
    {
        static int SumFromOneToValue(int value)
        {
            int result = 0;
            for (int i = 1; i <= value; i++)
            {
                result += i;
            }
            return result;
        }
        static int InitializeIntValue(String message)
        {
            int value;
            Console.Write(message);
            if (!int.TryParse(Console.ReadLine(), out value))
            {
                throw new Exception("You have to write a single number!");
            }
            return value;
        }

        static void Main(string[] args)
        {
            // Find out if matrix should be generate random or not
            bool isRandom = false;
            bool exit = false;
            while (!exit)
            {
                Console.Write("Generate radnom adjacency matrix?(Yes/No): ");
                string decision = Console.ReadLine().ToLower();
                switch (decision)
                {
                    case "yes": isRandom = true; exit = true; break;
                    case "no": isRandom = false; exit = true; break;
                    default: break;
                }
            }

            Graph graph = null;

            // Initialize nodeCount and connectionCount
            int nodeCount = 0, connectionCount = 0;
            try
            {
                nodeCount = InitializeIntValue("NodeCount: ");
                connectionCount = InitializeIntValue("ConnectionCount: ");

                // ConnectionCount can't be greater than 1 + 2 + ... + nodeCount - 1
                if (connectionCount > SumFromOneToValue(nodeCount - 1))
                {
                    throw new Exception("ConnectionCount can't be greater than 1 + 2 + ... +
nodeCount - 1");
                }
```

```
                graph = new Graph(nodeCount, connectionCount);

                graph.CreateAdjacencyMatrix(isRandom);
            }
            catch(Exception ex)
            {
                Console.WriteLine("Error:\n" + ex.Message);
                Console.Read();
                return;
            }

            graph.PrintAdjacencyMatrix();

            int[,] spanningTree = graph.FindSpanningTree();

            // Result
            Console.WriteLine("---------- Result -------");
            Console.WriteLine(String.Format("Longest distance: {0}"
,graph.FindSpanningTreeLongestEdge(spanningTree)));
            Console.WriteLine(String.Format("Cabels count: {0}",
graph.FindSpanningTreeNodeCount(spanningTree)));
            graph.PrintSpanningTree(spanningTree);


            Console.ReadLine();
        }
    }
}
```