

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ОТЧЕТ**

По лабораторной работе №10  
«Поиск расстояний во взвешенном графе»  
По дисциплине «Л и ОА в ИЗ»

Выполнили: ст. гр. 22ВВ4  
Жуков Илья  
Чумаев Сабит

Приняли: Юрова О.В.  
Акифьев И.В.

## Цель работы:

Написать код программы, выполнив следующие задания:

### По заданию 1:

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа  $G$ . Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки C++.
- 3.\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа  $G$ . Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

### По заданию 2:

1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

### По заданию 3:

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже). В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

## Ход работы:

### Описание кода программы по заданию 1.1 - 1.2 + 2.1-2.2:

```
using System;

using System.Collections.Generic;

using System.Security.Policy;

class Program
{
    internal class example2
    {
```

```

static void Main(string[] args)
{
    Console.Write("Введите размер матрицы: ");
    int size2 = Convert.ToInt32(Console.ReadLine());
    int[,] adjacencyMatrix = GenerateAdjacencyMatrix2(size2);
    Console.WriteLine("Матрица смежности для
неориентированного графа:");
    PrintMatrix2(adjacencyMatrix);
    int[,] distances2 = DistanceSearch(adjacencyMatrix);
    Console.WriteLine("Минимальные расстояния между
вершинами:");
    PrintDistances2(distances2);

    Console.WriteLine();

    FindMaxDistance(distances2);

    //массив, в котором хранятся эксцентриситеты каждой
вершины графа.
    int[] eccentricities = new int[size2];

    //переменная, в которой будет храниться текущее значение
диаметра графа.
    int diameter = 0;

    //переменная, в которой будет храниться текущее значение
радиуса графа.
    int radius = int.MaxValue;

    //список, в который будут добавляться вершины с
максимальным эксцентриситетом (периферийные точки).
    List<int> peripheralPoints = new List<int>();

```

```

        //список, в который будут добавляться вершины с
минимальным эксцентриситетом (центральные точки).

        List<int> centralPoints = new List<int>();

        for (int vertex = 0; vertex < size2; vertex++)
        {
            int maxDistance = 0;
            int maxVertex = -1;
            for (int i = 0; i < size2; i++)
            {
                if (i != vertex && distances2[vertex, i] >
maxDistance)
                {
                    //Вычисляется максимальное расстояние
maxDistance от текущей вершины до других вершин графа, исключая саму
вершину.

                    maxDistance = distances2[vertex, i];

                    //Сохраняется индекс вершины maxVertex, до
которой достигается максимальное расстояние.

                    maxVertex = i;
                }
            }

            //Значение maxDistance присваивается соответствующему
элементу массива eccentricities.

            eccentricities[vertex] = maxDistance;

            //Если maxDistance больше текущего значения diameter,
то diameter обновляется.

            if (maxDistance > diameter)
                diameter = maxDistance;
        }
    }
}

```

```
        //Если maxDistance меньше текущего значения radius, то
radius обновляется.
```

```
        if (maxDistance < radius)
            radius = maxDistance;
    }
```

```
    for (int vertex = 0; vertex < size2; vertex++)
    {
```

```
        //Если эксцентриситет текущей вершины равен diameter,
то добавляется индекс вершины + 1 в список peripheralPoints.
```

```
        if (eccentricities[vertex] == diameter)
            peripheralPoints.Add(vertex + 1);
```

```
        //Если эксцентриситет текущей вершины равен radius, то
добавляется индекс вершины + 1 в список centralPoints.
```

```
        if (eccentricities[vertex] == radius)
            centralPoints.Add(vertex + 1);
    }
```

```
    Console.WriteLine("Диаметр графа: " + diameter);
```

```
    Console.WriteLine("Радиус графа: " + radius);
```

```
    Console.WriteLine("Периферийные точки: " + string.Join(",
", peripheralPoints));
```

```
    Console.WriteLine("Центральные точки: " + string.Join(",
", centralPoints));
```

```
    }
```

```
    //Генерация неориентированной матрицы смежности (взвешенной)
```

```
    private static int[,] GenerateAdjacencyMatrix2(int size)
```

```
    {
```

```
        Random r = new Random();
```

```

int[,] matrix = new int[size, size];
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        if (i != j)
        {
            matrix[i, j] = r.Next(0, 10);
            matrix[j, i] = matrix[i, j];
        }
    }
}
return matrix;
}

```

//Вывод матрицы на экран

```

static void PrintMatrix2(int[,] matrix)
{
    int size = matrix.GetLength(0);
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            Console.Write(matrix[i, j] + " ");
        }
        Console.WriteLine();
    }
}

```

//Метод, осуществляющий поиск расстояний в графе

```

static int[,] DistanceSearch(int[,] adjacencyMatrix)

```

```

{
    //размерность матрицы, равная количеству вершин в графе.
    int size = adjacencyMatrix.GetLength(0);

    //двумерный массив, в котором будет храниться информация о
    расстояниях между вершинами.
    int[,] distances = new int[size, size];

    //Сначала инициализируется массив расстояний distances.
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            distances[i, j] = -1;
        }
    }

    for (int vertex = 0; vertex < size; vertex++)
    {
        //Создается массив visited, в котором все элементы
        инициализируются значением false.
        bool[] visited = new bool[size];

        //Создается очередь queue, куда добавляется текущая
        вершина.
        Queue<int> queue = new Queue<int>();

        //Текущей вершине присваивается расстояние 0.
        distances[vertex, vertex] = 0;

        //Помечается текущая вершина как посещенная.

```

```

        visited[vertex] = true;
        queue.Enqueue(vertex);

//Пока очередь не пуста, происходит обход графа в
ширину.

while (queue.Count > 0)
{
    //Извлекается вершина из очереди.

    int currentVertex = queue.Dequeue();

    //Для каждой вершины, смежной с текущей, и которая
еще не была посещена, вычисляется расстояние до нее и добавляется в
очередь и массив расстояний distances.

    for (int i = 0; i < size; i++)
    {
        if (adjacencyMatrix[currentVertex, i] > 0
&& !visited[i])
        {
            distances[vertex, i] = distances[vertex,
currentVertex] + adjacencyMatrix[currentVertex, i];

            visited[i] = true;
            queue.Enqueue(i);
        }
    }
}

int maxDistance = 0;
int maxVertex = -1;

// После обхода графа для каждой вершины вычисляется
максимальное расстояние maxDistance и соответствующая вершина
maxVertex.

for (int i = 0; i < size; i++)

```



```

        {
            if (i != vertex && distances[vertex, i] >
maxDistance)
            {
                maxDistance = distances[vertex, i];
                maxVertex = i;
            }
        }
    }

    return distances;
}

```

```

static void PrintDistances2(int[,] distances)
{
    //получает размер массива с помощью метода GetLength(0)
    int size = distances.GetLength(0);

    //затем использует вложенные циклы for для перебора всех
элементов массива.

    //Если значение элемента больше 0, оно выводится на экран,
в противном случае выводится 0.

    //После каждой строки матрицы происходит переход на новую
строку.

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (distances[i, j] > 0)
            {
                Console.Write(" " + distances[i, j]);
            }
        }
    }
}

```

```

        else
        {
            Console.Write(" " + "0");
        }
    }
    Console.WriteLine();
}

}

//Метод находит максимальное расстояние от каждой вершины до
других вершин в графе.

private static void FindMaxDistance(int[,] distances)
{
    //Сначала определяется размер матрицы distances.
    int size = distances.GetLength(0);

    //Затем происходит цикл по всем вершинам графа.
    for (int vertex = 0; vertex < size; vertex++)
    {
        //Для каждой вершины инициализируется переменная
maxDistance, которая
        //будет содержать максимальное расстояние от текущей
вершины до других вершин.
        int maxDistance = 0;

        //Затем происходит вложенный цикл, в котором
происходит перебор всех элементов в строке матрицы distances для
текущей вершины.
        for (int i = 0; i < size; i++)
        {
            //Если значение элемента больше текущего
maxDistance, то maxDistance обновляется.
            if (distances[vertex, i] > maxDistance)

```

```

        {
            maxDistance = distances[vertex, i];
        }
    }

    //После завершения внутреннего цикла для каждой
    //вершины находится максимальное расстояние до других вершин и выводится
    //в консоль.

    Console.WriteLine("Эксцентриситет " + (vertex + 1) +
        ": " + maxDistance);
    }
}
}
}
}

```

**Описание кода программы по заданию 1.3 + 2.1-2.2:**

```

using System;

using System.Collections.Generic;

using System.Security.Policy;

public class Program
{
    public static void Main(string[] args)
    {
        Console.Write("Введите размер матрицы: ");
        int size = Convert.ToInt32(Console.ReadLine());

        int[,] adjacencyMatrix = GenerateBinaryAdjacencyMatrix(size);
        Console.WriteLine("Ориентированная матрица смежности:");
        PrintMatrix(adjacencyMatrix);

        int[,] weightedMatrix =
        ConvertToWeightedMatrix(adjacencyMatrix);
    }
}

```

```
        Console.WriteLine("Взвешенная матрица смежности для  
ориентированного графа:");  
  
        PrintMatrix(weightedMatrix);  
  
        int[,] distances = DistanceSearch(weightedMatrix);  
        Console.WriteLine("Минимальные расстояния между вершинами:");  
        PrintDistances(distances);  
  
        Console.WriteLine();  
        FindMaxDistance(distances);  
  
        //массив, в котором хранятся эксцентриситеты каждой вершины  
графа.  
        int[] eccentricities = new int[size];  
  
        //переменная, в которой будет храниться текущее значение  
диаметра графа.  
        int diameter = 0;  
  
        //переменная, в которой будет храниться текущее значение  
радиуса графа.  
        int radius = int.MaxValue;  
  
        //список, в который будут добавляться вершины с максимальным  
эксцентриситетом (периферийные точки).  
        List<int> peripheralPoints = new List<int>();  
  
        //список, в который будут добавляться вершины с минимальным  
эксцентриситетом (центральные точки).  
        List<int> centralPoints = new List<int>();  
  
        for (int vertex = 0; vertex < size; vertex++)
```

```

{
    int maxDistance = 0;
    int maxVertex = -1;
    for (int i = 0; i < size; i++)
    {
        if (i != vertex && distances[vertex, i] > maxDistance)
        {
            //Вычисляется максимальное расстояние maxDistance
            от текущей вершины до других вершин графа, исключая саму вершину.

            maxDistance = distances[vertex, i];

            //Сохраняется индекс вершины maxVertex, до которой
            достигается максимальное расстояние.

            maxVertex = i;
        }
    }

    //Значение maxDistance присваивается соответствующему
    элементу массива eccentricities.

    eccentricities[vertex] = maxDistance;

    //Если maxDistance больше текущего значения diameter, то
    diameter обновляется.

    if (maxDistance > diameter)
        diameter = maxDistance;

    //Если maxDistance меньше текущего значения radius, то
    radius обновляется.

    if (maxDistance < radius)
        radius = maxDistance;
}

for (int vertex = 0; vertex < size; vertex++)

```

```

    {
        //Если эксцентриситет текущей вершины равен diameter, то
        добавляется индекс вершины + 1 в список peripheralPoints.

        if (eccentricities[vertex] == diameter)
            peripheralPoints.Add(vertex + 1);

        //Если эксцентриситет текущей вершины равен radius, то
        добавляется индекс вершины + 1 в список centralPoints.

        if (eccentricities[vertex] == radius)
            centralPoints.Add(vertex + 1);
    }

    Console.WriteLine("Диаметр графа: " + diameter);
    Console.WriteLine("Радиус графа: " + radius);
    Console.WriteLine("Периферийные точки: " + string.Join(", ",
peripheralPoints));
    Console.WriteLine("Центральные точки: " + string.Join(", ",
centralPoints));
}

//Генерируется ориентированная матрица состоящая из 0 и 1
private static int[,] GenerateBinaryAdjacencyMatrix(int size)
{
    Random random = new Random();
    int[,] adjacencyMatrix = new int[size, size];
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i != j)
                adjacencyMatrix[i, j] = random.Next(0, 2);
        }
    }
}

```

```

        return adjacencyMatrix;
    }

    //метод ConvertToWeightedMatrix преобразует двоичную матрицу в
    взвешенную, где каждое ребро имеет случайный вес от 1 до 10.

    private static int[,] ConvertToWeightedMatrix(int[,] binaryMatrix)
    {
        Random rand = new Random();

        //Сначала определяется размер матрицы binaryMatrix
        int size = binaryMatrix.GetLength(0);

        //создается новая матрица weightedMatrix того же размера.
        int[,] weightedMatrix = new int[size, size];

        //происходит цикл по всем элементам binaryMatrix.
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                //Если значение элемента равно 1, то в соответствующий
                элемент в weightedMatrix записывается случайное число от 1 до 10.
                if (binaryMatrix[i, j] == 1)
                {
                    weightedMatrix[i, j] = rand.Next(1, 10);
                }
            }
        }

        return weightedMatrix;
    }

    //Вывод матрицы на экран
    private static void PrintMatrix(int[,] matrix)

```

```

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

```

```

private static int[,] DistanceSearch(int[,] adjacencyMatrix)
{
    //размерность матрицы, равная количеству вершин в графе.
    int size = adjacencyMatrix.GetLength(0);

    //двумерный массив, в котором будет храниться информация о
    расстояниях между вершинами.
    int[,] distances = new int[size, size];

    //Сначала инициализируется массив расстояний distances.
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            distances[i, j] = -1;
        }
    }
    for (int vertex = 0; vertex < size; vertex++)

```



```

{
    //Создается массив visited, в котором все элементы
инициализируются значением false.

    bool[] visited = new bool[size];

    //Создается очередь queue, куда добавляется текущая
вершина.

    Queue<int> queue = new Queue<int>();

    //Текущей вершине присваивается расстояние 0.
    distances[vertex, vertex] = 0;

    //Помечается текущая вершина как посещенная.
    visited[vertex] = true;
    queue.Enqueue(vertex);

    //Пока очередь не пуста, происходит обход графа в ширину.
    while (queue.Count > 0)
    {
        //Извлекается вершина из очереди.
        int currentVertex = queue.Dequeue();
        for (int i = 0; i < size; i++)
        {
            //Для каждой вершины, смежной с текущей, и которая
еще не была посещена, вычисляется расстояние до нее и добавляется в
очередь и массив расстояний distances.

            if (adjacencyMatrix[currentVertex, i] > 0
&& !visited[i])
            {
                distances[vertex, i] = distances[vertex,
currentVertex] + adjacencyMatrix[currentVertex, i];
                visited[i] = true;
            }
        }
    }
}

```

```

        queue.Enqueue(i);
    }
}
}
}
return distances;
}
private static void PrintDistances(int[,] distances)
{
    //получает размер массива с помощью метода GetLength(0)
    int size = distances.GetLength(0);

    //затем использует вложенные циклы for для перебора всех
    элементов массива.

    //Если значение элемента больше 0, оно выводится на экран, в
    противном случае выводится 0.

    //После каждой строки матрицы происходит переход на новую
    строку.
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (distances[i, j] > 0)
            {
                Console.Write(" " + distances[i, j]);
            }
            else
            {
                Console.Write(" " + "0");
            }
        }
    }
}

```

```

        Console.WriteLine();
    }
}

//Метод находит максимальное расстояние от каждой вершины до
других вершин в графе.

private static void FindMaxDistance(int[,] distances)
{
    //Сначала определяется размер матрицы distances.
    int size = distances.GetLength(0);

    int diameter = 0;

    //Затем происходит цикл по всем вершинам графа.
    for (int vertex = 0; vertex < size; vertex++)
    {
        //Для каждой вершины инициализируется переменная
maxDistance, которая
        //будет содержать максимальное расстояние от текущей
вершины до других вершин.
        int maxDistance = 0;

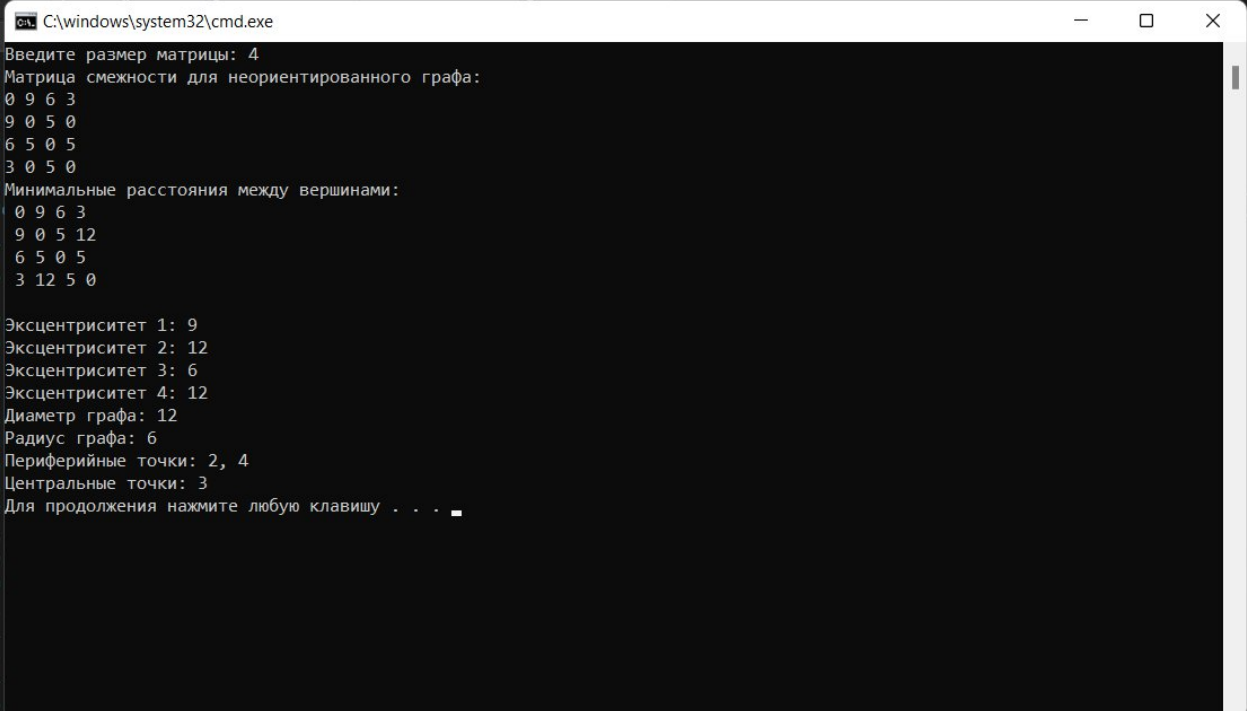
        //Затем происходит вложенный цикл, в котором происходит
перебор всех элементов в строке матрицы distances для текущей вершины.
        for (int i = 0; i < size; i++)
        {
            //Если значение элемента больше текущего maxDistance,
то maxDistance обновляется.
            if (distances[vertex, i] > maxDistance)
            {
                maxDistance = distances[vertex, i];
            }
        }
    }
}

```

//После завершения внутреннего цикла для каждой вершины находится максимальное расстояние до других вершин и выводится в консоль.

```
        Console.WriteLine("Эксцентриситет " + (vertex + 1) + ": " + maxDistance);  
    }  
}  
}
```

## Результат работы программы 1.1-1.2 + 2.1-2.2:



```
C:\windows\system32\cmd.exe  
Введите размер матрицы: 4  
Матрица смежности для неориентированного графа:  
0 9 6 3  
9 0 5 0  
6 5 0 5  
3 0 5 0  
Минимальные расстояния между вершинами:  
0 9 6 3  
9 0 5 12  
6 5 0 5  
3 12 5 0  
Эксцентриситет 1: 9  
Эксцентриситет 2: 12  
Эксцентриситет 3: 6  
Эксцентриситет 4: 12  
Диаметр графа: 12  
Радиус графа: 6  
Периферийные точки: 2, 4  
Центральные точки: 3  
Для продолжения нажмите любую клавишу . . .
```

## Результат работы программы 1.3 + 2.1-2.2:

```
C:\windows\system32\cmd.exe
Введите размер матрицы: 4
Ориентированная матрица смежности:
0 1 1 0
1 0 0 1
0 1 0 0
0 0 0 0
Взвешенная матрица смежности для ориентированного графа:
0 7 6 0
2 0 0 6
0 1 0 0
0 0 0 0
Минимальные расстояния между вершинами:
0 7 6 13
2 0 8 6
3 1 0 7
0 0 0 0
Эксцентриситет 1: 13
Эксцентриситет 2: 8
Эксцентриситет 3: 7
Эксцентриситет 4: 0
Диаметр графа: 13
Радиус графа: 0
Периферийные точки: 1
Центральные точки: 4
Для продолжения нажмите любую клавишу . . .
```

**Вывод:** в ходе лабораторной работы мы научились осуществлять поиск расстояния во взвешенном графе (ориентированном и неориентированном). А также определять радиус, диаметр, подмножества периферийных и центральных вершин графа.