

Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

**ОТЧЕТ**  
по лабораторной работе №10  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Поиск расстояний во взвешенном графе»

Выполнили:

студенты группы 24ВВВ3

Агапов И.А.  
Любченко В.К.

Приняли:

к.т.н., доцент Юрова О.В.  
к.т.н., Деев М.В.

Пенза 2025

**Цель работы** – Освоение методов генерации, визуализации и анализа графов через создание программы, которая реализует генерацию матриц смежности для взвешенных ориентированных и неориентированных графов, выполняет поиск расстояний между вершинами с использованием очереди, вычисляет метрические характеристики (радиус, диаметр, центральные и периферийные вершины), а также обеспечивает обработку параметров командной строки для гибкой настройки типа графа.

## Лабораторное задание:

### Задание 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.

### Задание 2

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

### Задание 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).

## Код программы

C#

```

using System;
using System.Collections.Generic;

class Program
{
    static void Main(string[] args)
    {
        // Параметры по умолчанию
        bool isDirected = false;
        bool isWeighted = true;
        int vertices = 6;
        int maxWeight = 10;
        double density = 0.3;
        int startVertex = 0;

        // Обработка параметров командной строки
        if (args.Length > 0)
        {
            ParseCommandLineArgs(args, ref isDirected, ref isWeighted, ref vertices,
ref maxWeight, ref density, ref startVertex);
        }
        else
        {
            // Если параметры не переданы, запросим у пользователя
            isDirected = SelectGraphType();
        }

        string graphType = isDirected ? "ориентированного" : "неориентированного";
        string weightType = isWeighted ? "взвешенного" : "невзвешенного";

        Console.WriteLine("ЛАБОРАТОРНАЯ РАБОТА №10 – ПОИСК РАССТОЯНИЙ ВО ВЗВЕШЕННОМ
ГРАФЕ");

        Console.WriteLine("=====\\n");
        Console.WriteLine($"Параметры: {graphType} {weightType} граф, вершин:
{vertices}, плотность: {density}");

        // Генерация матрицы смежности
        Console.WriteLine($"\\nМАТРИЦА СМЕЖНОСТИ {graphType.ToUpper()}{weightType.ToUpper()} ГРАФА");
        Console.WriteLine("-----");
        var matrix = GenerateWeightedMatrix(vertices, density, maxWeight,
isDirected, isWeighted);
        PrintWeightedMatrix(matrix);

        // Подробный вывод матрицы смежности
        Console.WriteLine($"\\nПОДРОБНЫЙ ВЫВОД МАТРИЦЫ СМЕЖНОСТИ ({graphType}{weightType} граф)");
        Console.WriteLine("-----");
        PrintDetailedAdjacencyMatrix(matrix, isDirected);

        // Если startVertex не задан в командной строке, запросим у пользователя
        if (startVertex < 0 || startVertex >= vertices)
        {
            startVertex = GetStartVertex(vertices);
        }

        // Поиск расстояний от выбранной вершины
        Console.WriteLine($"\\nПОИСК РАССТОЯНИЙ ОТ ВЕРШИНЫ {startVertex}");
        Console.WriteLine("-----");
        var distances = BFSFindDistancesWeighted(matrix, startVertex);
        PrintDistances(distances, startVertex);

        // Матрица расстояний между всеми вершинами
    }
}

```

```

        Console.WriteLine($"\\nМАТРИЦА РАССТОЯНИЙ МЕЖДУ ВСЕМИ ВЕРШИНАМИ ({graphType}
граф)");
        Console.WriteLine("-----");
        var distanceMatrix = BuildDistanceMatrix(matrix);
        PrintDistanceMatrix(distanceMatrix);

        // Анализ графа (радиус, диаметр, центральные и периферийные вершины)
        Console.WriteLine($"\\nАНАЛИЗ {graphType.ToUpper()} ГРАФА");
        Console.WriteLine("-----");
        AnalyzeGraph(distanceMatrix);

        Console.WriteLine("\\nНажмите любую клавишу для выхода...");
        Console.ReadKey();
    }

    static void ParseCommandLineArgs(string[] args, ref bool isDirected, ref bool
isWeighted,
                                    ref int vertices, ref int maxWeight, ref double
density, ref int startVertex)
    {
        for (int i = 0; i < args.Length; i++)
        {
            switch (args[i].ToLower())
            {
                case "-d":
                case "--directed":
                    isDirected = true;
                    break;
                case "-u":
                case "--undirected":
                    isDirected = false;
                    break;
                case "-w":
                case "--weighted":
                    isWeighted = true;
                    break;
                case "--unweighted":
                    isWeighted = false;
                    break;
                case "-v":
                case "--vertices":

                    if (i + 1 < args.Length && int.TryParse(args[i + 1], out int v))
                        vertices = v;
                    break;
                case "-mw":
                case "--maxweight":
                    if (i + 1 < args.Length && int.TryParse(args[i + 1], out int
mw))
                        maxWeight = mw;
                    break;
                case "-den":
                case "--density":
                    if (i + 1 < args.Length && double.TryParse(args[i + 1], out
double den))
                        density = den;
                    break;
                case "-s":
                case "--start":
                    if (i + 1 < args.Length && int.TryParse(args[i + 1], out int s))
                        startVertex = s;
                    break;
                case "-h":
                case "--help":
                    PrintHelp();
            }
        }
    }
}

```

```

                Environment.Exit(0);
            break;
        }
    }

    static void PrintHelp()
{
    Console.WriteLine("Использование: Lab10.exe [ПАРАМЕТРЫ]");
    Console.WriteLine();
    Console.WriteLine("Параметры:");
    Console.WriteLine(" -d, --directed      Ориентированный граф");
    Console.WriteLine(" -u, --undirected    Неориентированный граф (по
умолчанию)");
    Console.WriteLine(" -w, --weighted      Взвешенный граф (по умолчанию)");
    Console.WriteLine("          --unweighted Невзвешенный граф");
    Console.WriteLine(" -v, --vertices N   Количество вершин (по умолчанию:
6)");
    Console.WriteLine(" -mw, --maxweight N Максимальный вес ребра (по
умолчанию: 10)");
    Console.WriteLine(" -den, --density X  Плотность графа 0.0-1.0 (по
умолчанию: 0.3)");
    Console.WriteLine(" -s, --start N      Начальная вершина для обхода (по
умолчанию: 0)");
    Console.WriteLine(" -h, --help          Показать эту справку");
    Console.WriteLine();
    Console.WriteLine("Примеры:");
    Console.WriteLine(" Lab10.exe -d -w -v 8 -den 0.4 -s 0");
    Console.WriteLine(" Lab10.exe --undirected --unweighted --vertices 5");
    Console.WriteLine(" Lab10.exe -d -mw 15 -den 0.5");
}

static bool SelectGraphType()
{
    while (true)
    {
        Console.WriteLine("ВЫБЕРИТЕ ТИП ГРАФА:");
        Console.WriteLine("1 - Ориентированный граф");
        Console.WriteLine("2 - Неориентированный граф");
        Console.Write("Ваш выбор (1 или 2): ");

        string input = Console.ReadLine();
        if (input == "1") return true;
        if (input == "2") return false;

        Console.WriteLine("Ошибка! Введите 1 или 2");
    }
}

static int[,] GenerateWeightedMatrix(int vertices, double density, int
maxWeight, bool isDirected, bool isWeighted)
{
    Random rand = new Random();
    int[,] matrix = new int[vertices, vertices];

    if (isDirected)
    {
        // Генерация ориентированного графа
        for (int i = 0; i < vertices; i++)
        {
            for (int j = 0; j < vertices; j++)
            {
                if (i != j && rand.NextDouble() < density)
                {
                    int weight = isWeighted ? rand.Next(1, maxWeight + 1) : 1;
                    matrix[i, j] = weight;
                }
            }
        }
    }
}

```

```

                matrix[i, j] = weight;
            }
        }
    }
} else
{
    // Генерация неориентированного графа
    for (int i = 0; i < vertices; i++)
    {
        for (int j = i + 1; j < vertices; j++)
        {
            if (rand.NextDouble() < density)
            {
                int weight = isWeighted ? rand.Next(1, maxWeight + 1) : 1;
                matrix[i, j] = weight;
                matrix[j, i] = weight;
            }
        }
    }
}

return matrix;
}
static int[] BFSDFindDistancesWeighted(int[,] G, int v)
{
    int size_G = G.GetLength(0);
    int[] DIST = new int[size_G];

    // Инициализация: все вершины недостижимы
    for (int i = 0; i < size_G; i++)
    {
        DIST[i] = -1;
    }

    BFSDWeighted(G, v, DIST);
    return DIST;
}

static void BFSDWeighted(int[,] G, int v, int[] DIST)
{
    Queue<int> Q = new Queue<int>();
    Q.Enqueue(v);
    DIST[v] = 0;

    while (Q.Count > 0)
    {
        int currentV = Q.Dequeue();

        // Просматриваем все исходящие ребра из текущей вершины
        for (int i = 0; i < G.GetLength(0); i++)
        {
            // Если есть ребро currentV -> i и вершина i еще не посещена
            if (G[currentV, i] > 0 && DIST[i] == -1)
            {
                Q.Enqueue(i);
                DIST[i] = DIST[currentV] + G[currentV, i];
            }
        }
    }
}

static int[,] BuildDistanceMatrix(int[,] matrix)
{
    int n = matrix.GetLength(0);

```

```

int[,] distanceMatrix = new int[n, n];

for (int i = 0; i < n; i++)
{
    int[] distances = BFSFindDistancesWeighted(matrix, i);
    for (int j = 0; j < n; j++)
    {
        distanceMatrix[i, j] = distances[j];
    }
}

return distanceMatrix;
}

static void AnalyzeGraph(int[,] distanceMatrix)
{
    int n = distanceMatrix.GetLength(0);
    int[] eccentricities = new int[n];

    // Вычисляем эксцентриситет для каждой вершины
    for (int i = 0; i < n; i++)
    {
        int maxDistance = -1;
        for (int j = 0; j < n; j++)
        {
            if (i != j && distanceMatrix[i, j] > maxDistance &&
distanceMatrix[i, j] != -1)
            {
                maxDistance = distanceMatrix[i, j];
            }
        }
        eccentricities[i] = maxDistance;
    }

    // Находим радиус и диаметр
    int radius = int.MaxValue;
    int diameter = -1;

    for (int i = 0; i < n; i++)
    {
        if (eccentricities[i] != -1)
        {
            if (eccentricities[i] < radius) radius = eccentricities[i];
            if (eccentricities[i] > diameter) diameter = eccentricities[i];
        }
    }

    if (radius == int.MaxValue)
    {
        radius = -1;
        diameter = -1;
    }

    Console.WriteLine($"Радиус графа: {(radius == -1 ? "не определен (граф несвязный)" : radius.ToString())}");
    Console.WriteLine($"Диаметр графа: {(diameter == -1 ? "не определен (граф несвязный)" : diameter.ToString())}");

    // Находим центральные вершины
    Console.Write("Центральные вершины: ");
    List<int> centralVertices = new List<int>();
    for (int i = 0; i < n; i++)
    {
        if (eccentricities[i] == radius)
        {

```

```

        centralVertices.Add(i);
    }
}
Console.WriteLine(centralVertices.Count > 0 ? string.Join(", ", centralVertices) : "отсутствуют");

// Находим периферийные вершины
Console.WriteLine("Периферийные вершины: ");
List<int> peripheralVertices = new List<int>();
for (int i = 0; i < n; i++)
{
    if (eccentricities[i] == diameter)
    {
        peripheralVertices.Add(i);
    }
}
Console.WriteLine(peripheralVertices.Count > 0 ? string.Join(", ", peripheralVertices) : "отсутствуют");

// Выводим эксцентриситеты всех вершин
Console.WriteLine("\nЭксцентриситеты вершин:");
for (int i = 0; i < n; i++)
{
    string ecc = eccentricities[i] == -1 ? "∞ (недостижимы некоторые вершины)" : eccentricities[i].ToString();
    Console.WriteLine($"  Вершина {i}: эксцентриситет = {ecc}");
}
}

static int GetStartVertex(int maxVertex)
{
    while (true)
    {
        Console.Write($"\\nВведите начальную вершину (0-{maxVertex - 1}): ");
        if (int.TryParse(Console.ReadLine(), out int vertex) && vertex >= 0 && vertex < maxVertex)
        {
            return vertex;
        }
        Console.WriteLine($"Ошибка! Введите число от 0 до {maxVertex - 1}");
    }
}

static void PrintWeightedMatrix(int[,] matrix)
{
    int n = matrix.GetLength(0);
    Console.Write("   ");
    for (int i = 0; i < n; i++)
        Console.Write($"{i,3} ");
    Console.WriteLine();

    for (int i = 0; i < n; i++)
    {
        Console.Write($"{i,2} ");
        for (int j = 0; j < n; j++)
        {
            if (matrix[i, j] == 0)
                Console.Write(" - ");
            else
                Console.Write($"{matrix[i, j],3} ");
        }
        Console.WriteLine();
    }
}
}
```

```

static void PrintDetailedAdjacencyMatrix(int[,] matrix, bool isDirected)
{
    int n = matrix.GetLength(0);

    string directionInfo = isDirected ?
        "ОРИЕНТИРОВАННЫЙ граф: матрица показывает веса ребер i -> j" :
        "НЕОРИЕНТИРОВАННЫЙ граф: матрица симметрична, ребра i-j двусторонние";

    Console.WriteLine($"Пояснение: {directionInfo}");
    Console.WriteLine(" '-' означает отсутствие ребра");
    Console.WriteLine(" число означает вес ребра\n");

    Console.Write("      ");
    for (int j = 0; j < n; j++)
        Console.Write($"j={j}  ");
    Console.WriteLine();

    for (int i = 0; i < n; i++)
    {
        Console.Write($"i={i} ");
        for (int j = 0; j < n; j++)
        {
            if (matrix[i, j] == 0)
                Console.Write(" - ");
            else
                Console.WriteLine($"{matrix[i, j],3} ");
        }
        Console.WriteLine();
    }

    // Вывод списка всех существующих ребер
    Console.WriteLine($"\\nСПИСОК РЕБЕР ГРАФА ({(isDirected ? "ориентированный" :
    "неориентированный")});");
    string edgeFormat = isDirected ? "  {0} -> {1} (вес: {2})" : "  {0} - {1}
    (вес: {2})";

    bool hasEdges = false;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (matrix[i, j] > 0)
            {
                // Для неориентированного графа выводим только одно направление
                if (!isDirected && i >= j) continue;

                Console.WriteLine(string.Format(edgeFormat, i, j, matrix[i,
                j]));
                hasEdges = true;
            }
        }
    }
    if (!hasEdges)
    {
        Console.WriteLine("  Граф не содержит ребер");
    }
}

static void PrintDistanceMatrix(int[,] distanceMatrix)
{
    int n = distanceMatrix.GetLength(0);
    Console.Write("      ");
    for (int i = 0; i < n; i++)
        Console.Write($"{i,3} ");
    Console.WriteLine();
}

```

```

        for (int i = 0; i < n; i++)
    {
        Console.Write($"{i},2} ");
        for (int j = 0; j < n; j++)
        {
            if (distanceMatrix[i, j] == -1)
                Console.Write(" - ");
            else
                Console.WriteLine($"{distanceMatrix[i, j]},3} ");
        }
        Console.WriteLine();
    }

    // Анализ связности
    Console.WriteLine("\nАНАЛИЗ СВЯЗНОСТИ");
    bool isStronglyConnected = true;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i != j && distanceMatrix[i, j] == -1)
            {
                isStronglyConnected = false;
                break;
            }
        }
        if (!isStronglyConnected) break;
    }

    if (isStronglyConnected)
    {
        Console.WriteLine(" Граф является связным (все вершины достижимы из
любой вершины)");
    }
    else
    {
        Console.WriteLine(" Граф не является связным (не все вершины
достижимы)");
    }
}

static void PrintDistances(int[] distances, int startVertex)
{
    Console.WriteLine($"Расстояния от вершины {startVertex}:");
    for (int i = 0; i < distances.Length; i++)
    {
        string status = distances[i] == -1 ? "недостижима" :
distances[i].ToString();
        Console.WriteLine($" До вершины {i}: {status}");
    }
}

```

## Результаты работы программы

ВЫБЕРИТЕ ТИП ГРАФА:

- 1 - Ориентированный граф
- 2 - Неориентированный граф

Ваш выбор (1 или 2): 1

ЛАБОРАТОРНАЯ РАБОТА №10 - ПОИСК РАССТОЯНИЙ ВО ВЗВЕШЕННОМ ГРАФЕ

=====

Параметры: ориентированного взвешенного графа, вершин: 6, плотность: 0,3

МАТРИЦА СМЕЖНОСТИ ОРИЕНТИРОВАННОГО ВЗВЕШЕННОГО ГРАФА

	0	1	2	3	4	5
0	-	-	-	-	-	-
1	6	-	-	2	-	-
2	7	-	-	-	-	-
3	-	-	-	-	5	-
4	-	-	10	4	-	1
5	3	-	-	-	-	-

ПОДРОБНЫЙ ВЫВОД МАТРИЦЫ СМЕЖНОСТИ (ориентированного взвешенного графа)

Пояснение: ОРИЕНТИРОВАННЫЙ граф: матрица показывает веса ребер  $i \rightarrow j$

'-' означает отсутствие ребра

число означает вес ребра

	j=0	j=1	j=2	j=3	j=4	j=5
i=0	-	-	-	-	-	-
i=1	6	-	-	2	-	-
i=2	7	-	-	-	-	-
i=3	-	-	-	-	5	-
i=4	-	-	10	4	-	1
i=5	3	-	-	-	-	-

СПИСОК РЕБЕР ГРАФА (ориентированный):

- 1 -> 0 (вес: 6)
- 1 -> 3 (вес: 2)
- 2 -> 0 (вес: 7)
- 3 -> 4 (вес: 5)
- 4 -> 2 (вес: 10)
- 4 -> 3 (вес: 4)
- 4 -> 5 (вес: 1)
- 5 -> 0 (вес: 3)

Рисунок 1 - Результат работы программы

ПОИСК РАССТОЯНИЙ ОТ ВЕРШИНЫ 0

-----  
Расстояния от вершины 0:

До вершины 0: 0  
До вершины 1: недостижима  
До вершины 2: недостижима  
До вершины 3: недостижима  
До вершины 4: недостижима  
До вершины 5: недостижима

МАТРИЦА РАССТОЯНИЙ МЕЖДУ ВСЕМИ ВЕРШИНАМИ (ориентированного графа)

	0	1	2	3	4	5
0	0	-	-	-	-	-
1	6	0	17	2	7	8
2	7	-	0	-	-	-
3	22	-	15	0	5	6
4	17	-	10	4	0	1
5	3	-	-	-	-	0

АНАЛИЗ СВЯЗНОСТИ:

Граф не является связным (не все вершины достижимы)

АНАЛИЗ ОРИЕНТИРОВАННОГО ГРАФА

-----  
Радиус графа: 3

Диаметр графа: 22

Центральные вершины: 5

Периферийные вершины: 3

Эксцентрикитеты вершин:

Вершина 0: эксцентрикитет = ? (недостижимы некоторые вершины)  
Вершина 1: эксцентрикитет = 17  
Вершина 2: эксцентрикитет = 7  
Вершина 3: эксцентрикитет = 22  
Вершина 4: эксцентрикитет = 17  
Вершина 5: эксцентрикитет = 3

Рисунок 2 - Результат работы программы

**Вывод:** В ходе лабораторной работы были успешно освоены методы генерации, визуализации и анализа графов, реализованы алгоритмы поиска расстояний с использованием очереди, вычислены метрические характеристики графов.