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

ОТЧЕТ

по лабораторной работе №9

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в 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 – Сравнительное исследование алгоритмов поиска расстояний в графах на основе обхода в ширину (BFS) и обхода в глубину (DFS) с использованием различных структур данных) и оценка их временной эффективности для графов разного порядка.

Лабораторное задание:

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного 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.* Реализуйте процедуру поиска расстояний для графа, представленного списками смежности.

Задание 2*

1. Реализуйте процедуру поиска расстояний на основе обхода в глубину.
2. Реализуйте процедуру поиска расстояний на основе обхода в глубину для графа, представленного списками смежности.
3. Оцените время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и обхода в ширину для графов разных порядков.

Задание 1

Код программы

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
    static void Main()
    {
        Console.WriteLine("ЛАБОРАТОРНАЯ РАБОТА №9 – ЗАДАНИЕ 1");
        Console.WriteLine("==================================\n");

        int vertices = 8;
```

```
        // ЗАДАНИЕ 1.1 - Генерация матрицы смежности
        Console.WriteLine("1.1 - Генерация матрицы смежности для неориентированного
графа");
        Console.WriteLine("-----------------------------------------------------
----");
        var matrix = GenerateAdjacencyMatrix(vertices, 0.3);
        PrintMatrix(matrix);

        // ЗАДАНИЕ 1.2 - Поиск расстояний BFS
        Console.WriteLine("\n1.2 - Поиск расстояний BFS (от вершины 0)");
        Console.WriteLine("----------------------------------------");
        var distances = BFSFindDistances(matrix, 0);
        PrintDistances(distances);

        // ЗАДАНИЕ 1.3* - Работа со списками смежности
        Console.WriteLine("\n1.3* - Работа со списками смежности");
        Console.WriteLine("---------------------------------");
        var adjList = MatrixToAdjacencyList(matrix);
        PrintAdjacencyList(adjList);

        var listDistances = BFSFindDistancesWithAdjacencyList(adjList, 0);
        PrintDistances(listDistances);

        // Проверка совпадения результатов
        bool resultsMatch = Enumerable.SequenceEqual(distances, listDistances);
        Console.WriteLine($"\nРезультаты BFS с матрицей и списками совпадают:
{resultsMatch}");

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

    // 1.1 - Генерация матрицы смежности для неориентированного графа
    static int[,] GenerateAdjacencyMatrix(int vertices, double density = 0.3)
    {
        Random rand = new Random();
        int[,] matrix = new int[vertices, vertices];

        for (int i = 0; i < vertices; i++)
        {
            for (int j = i + 1; j < vertices; j++)
            {
                if (rand.NextDouble() < density)
                {
                    matrix[i, j] = 1;
                    matrix[j, i] = 1;
                }
            }
        }

        return matrix;
    }

    // Вывод матрицы смежности на экран
    static void PrintMatrix(int[,] matrix)
    {
        int n = matrix.GetLength(0);
        Console.Write("   ");
        for (int i = 0; i < n; i++)
            Console.Write($"{i,2} ");
        Console.WriteLine();

        for (int i = 0; i < n; i++)
        {
            Console.Write($"{i,2} ");
```

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

// 1.2 - Поиск расстояний BFS с матрицей смежности
static int[] BFSFindDistances(int[,] G, int v)
{
    int size_G = G.GetLength(0);
    int[] DIST = new int[size_G];

    // П.1.1 - для всех i положим DIST[i] = -1
    for (int i = 0; i < size_G; i++)
    {
        DIST[i] = -1;
    }

    // П.1.2 - ВЫПОЛНЯТЬ BFSD(v)
    BFSD(G, v, DIST);

    return DIST;
}

// Алгоритм BFSD(v) из методички
static void BFSD(int[,] G, int v, int[] DIST)
{
    // П.2.1 - Создать пустую очередь
    Queue<int> Q = new Queue<int>();

    // П.2.2 - Поместить v в очередь
    Q.Enqueue(v);

    // П.2.3 - Обновить вектор расстояний DIST[x] = 0
    DIST[v] = 0;

    // П.2.4 - ПОКА Q != ∅ ВЫПОЛНЯТЬ
    while (Q.Count > 0)
    {
        // П.2.5 - v = Q.front()
        int currentV = Q.Dequeue();

        // П.2.8 - ДЛЯ i = 1 ДО size_G ВЫПОЛНЯТЬ
        for (int i = 0; i < G.GetLength(0); i++)
        {
            // П.2.9 - ЕСЛИ G(v,i) == 1 И DIST == -1
            if (G[currentV, i] == 1 && DIST[i] == -1)
            {
                // П.2.11 - Поместить i в очередь
                Q.Enqueue(i);
                // П.2.12 - Обновить вектор расстояний DIST[i] = DIST[v] + 1
                DIST[i] = DIST[currentV] + 1;
            }
        }
    }
}

// 1.3* - Преобразование матрицы смежности в списки смежности
static Dictionary<int, List<int>> MatrixToAdjacencyList(int[,] matrix)
{
    int n = matrix.GetLength(0);
    Dictionary<int, List<int>> adjList = new Dictionary<int, List<int>>();
```

```
        for (int i = 0; i < n; i++)
        {
            adjList[i] = new List<int>();
            for (int j = 0; j < n; j++)
            {
                if (matrix[i, j] == 1)
                {
                    adjList[i].Add(j);
                }
            }
        }

        return adjList;
    }

    // 1.3* - Поиск расстояний BFS со списками смежности
    static int[] BFSFindDistancesWithAdjacencyList(Dictionary<int, List<int>>
adjList, int v)
    {
        int size_G = adjList.Count;
        int[] DIST = new int[size_G];

        for (int i = 0; i < size_G; i++)
        {
            DIST[i] = -1;
        }

        Queue<int> Q = new Queue<int>();
        Q.Enqueue(v);
        DIST[v] = 0;

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

            foreach (int neighbor in adjList[currentV])
            {
                if (DIST[neighbor] == -1)
                {
                    Q.Enqueue(neighbor);
                    DIST[neighbor] = DIST[currentV] + 1;
                }
            }
        }

        return DIST;
    }

    // Вспомогательные функции
    static void PrintDistances(int[] distances)
    {
        Console.WriteLine("Вектор расстояний:");
        for (int i = 0; i < distances.Length; i++)
        {
            string status = distances[i] == -1 ? "недостижима" :
distances[i].ToString();
            Console.WriteLine($"  Вершина {i}: расстояние = {status}");
        }
    }

    static void PrintAdjacencyList(Dictionary<int, List<int>> adjList)
    {
        Console.WriteLine("Списки смежности:");
        foreach (var vertex in adjList.Keys.OrderBy(k => k))
        {
```

```
            Console.WriteLine($"  Вершина {vertex}: [{string.Join(", ",
adjList[vertex])}]");
        }
    }
}
```

Результаты работы программы

```
1.1 - Генерация матрицы смежности для неориентированного графа
----------------------------------------------------------------
    0  1  2  3  4  5  6  7
 0  0  0  0  0  0  0  0  0
 1  0  0  1  0  0  0  0  1
 2  0  1  0  0  0  0  1  0
 3  0  0  0  0  1  0  0  0
 4  0  0  0  1  0  1  0  1
 5  0  0  0  0  1  0  0  1
 6  0  0  1  0  0  0  0  0
 7  0  1  0  0  1  1  0  0

1.2 - Поиск расстояний BFS (от вершины 0)
----------------------------------------
Вектор расстояний:
  Вершина 0: расстояние = 0
  Вершина 1: расстояние = недостижима
  Вершина 2: расстояние = недостижима
  Вершина 3: расстояние = недостижима
  Вершина 4: расстояние = недостижима
  Вершина 5: расстояние = недостижима
  Вершина 6: расстояние = недостижима
  Вершина 7: расстояние = недостижима

1.3* - Работа со списками смежности
-----------------------------------
Списки смежности:
  Вершина 0: []
  Вершина 1: [2, 7]
  Вершина 2: [1, 6]
  Вершина 3: [4]
  Вершина 4: [3, 5, 7]
  Вершина 5: [4, 7]
  Вершина 6: [2]
  Вершина 7: [1, 4, 5]
Вектор расстояний:
  Вершина 0: расстояние = 0
  Вершина 1: расстояние = недостижима
  Вершина 2: расстояние = недостижима
  Вершина 3: расстояние = недостижима
  Вершина 4: расстояние = недостижима
  Вершина 5: расстояние = недостижима
  Вершина 6: расстояние = недостижима
  Вершина 7: расстояние = недостижима

Результаты BFS с матрицей и списками совпадают: True
```

Рисунок 1 - Результат работы программы 1

Задание 2

Код программы

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        Console.WriteLine("ЛАБОРАТОРНАЯ РАБОТА №9 - ЗАДАНИЕ 2*");
        Console.WriteLine("==================================\n");

        int vertices = 8;

        // Генерация тестового графа
        var matrix = GenerateAdjacencyMatrix(vertices, 0.3);
        var adjList = MatrixToAdjacencyList(matrix);

        Console.WriteLine("Тестовый граф:");
        PrintMatrix(matrix);

        // ЗАДАНИЕ 2.1* - Поиск расстояний DFS с матрицей
        Console.WriteLine("\n2.1* - Поиск расстояний DFS с матрицей смежности (от
вершины 0)");
        Console.WriteLine("-----------------------------------------------------------
----");
        var dfsMatrixDistances = DFSFindDistances(matrix, 0);
        PrintDistances(dfsMatrixDistances);

        // ЗАДАНИЕ 2.2* - Поиск расстояний DFS со списками
        Console.WriteLine("\n2.2* - Поиск расстояний DFS со списками смежности (от
вершины 0)");
        Console.WriteLine("-----------------------------------------------------------
------");
        var dfsListDistances = DFSFindDistancesWithAdjacencyList(adjList, 0);
        PrintDistances(dfsListDistances);

        // Сравнение с BFS для проверки корректности
        Console.WriteLine("\nСравнение с BFS для проверки:");
        Console.WriteLine("-----------------------------");
        var bfsDistances = BFSFindDistances(matrix, 0);
        Console.WriteLine("BFS расстояния:");
        PrintDistances(bfsDistances);

        bool dfsResultsMatch = Enumerable.SequenceEqual(dfsMatrixDistances,
dfsListDistances);
        Console.WriteLine($"Результаты DFS с матрицей и списками совпадают:
{dfsResultsMatch}");

        // ЗАДАНИЕ 2.3* - Сравнение производительности
        Console.WriteLine("\n2.3* - Сравнение производительности алгоритмов");
        Console.WriteLine("-----------------------------------------------");
        CompareAlgorithms();

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

```
    // Генерация матрицы смежности
    static int[,] GenerateAdjacencyMatrix(int vertices, double density = 0.3)
    {
        Random rand = new Random();
        int[,] matrix = new int[vertices, vertices];

        for (int i = 0; i < vertices; i++)
        {
            for (int j = i + 1; j < vertices; j++)
            {
                if (rand.NextDouble() < density)
                {
                    matrix[i, j] = 1;
                    matrix[j, i] = 1;
                }
            }
        }

        return matrix;
    }

    // 2.1* - Поиск расстояний на основе обхода в глубину (матрица)
    static int[] DFSFindDistances(int[,] G, int v)
    {
        int size_G = G.GetLength(0);
        int[] DIST = new int[size_G];
        bool[] visited = new bool[size_G];

        for (int i = 0; i < size_G; i++)
        {
            DIST[i] = -1;
            visited[i] = false;
        }

        DFSMatrix(G, v, 0, DIST, visited);

        return DIST;
    }

    static void DFSMatrix(int[,] G, int current, int depth, int[] DIST, bool[]
visited)
    {
        visited[current] = true;
        DIST[current] = depth;

        for (int i = 0; i < G.GetLength(0); i++)
        {
            if (G[current, i] == 1 && !visited[i])
            {
                DFSMatrix(G, i, depth + 1, DIST, visited);
            }
        }
    }

    // 2.2* - Поиск расстояний DFS со списками смежности
    static int[] DFSFindDistancesWithAdjacencyList(Dictionary<int, List<int>>
adjList, int v)
    {
        int size_G = adjList.Count;
        int[] DIST = new int[size_G];
        bool[] visited = new bool[size_G];

        for (int i = 0; i < size_G; i++)
        {
            DIST[i] = -1;
```

```
            visited[i] = false;
        }

        DFSList(adjList, v, 0, DIST, visited);

        return DIST;
    }

    static void
DFSList(
Dictionary<int, List<int>> adjList, int current, int depth, int[] DIST, bool[]
visited)
    {
        visited[current] = true;
        DIST[current] = depth;

        foreach (int neighbor in adjList[current])
        {
            if (!visited[neighbor])
            {
                DFSList(adjList, neighbor, depth + 1, DIST, visited);
            }
        }
    }

    // 2.3* - Сравнение производительности алгоритмов
    static void CompareAlgorithms()
    {
        int[] graphSizes = { 100, 500, 1000 };
        double[] densities = { 0.1, 0.3, 0.5 };

        Console.WriteLine("Сравнение времени выполнения (в тиках):");
        Console.WriteLine("=====================================");

        foreach (int size in graphSizes)
        {
            foreach (double density in densities)
            {
                Console.WriteLine($"\nГраф: {size} вершин, плотность {density:P0}");

                // Генерация тестовых данных
                var matrix = GenerateAdjacencyMatrix(size, density);
                var adjList = MatrixToAdjacencyList(matrix);

                // Измерение времени выполнения
                long bfsMatrixTime = MeasureTime(() => BFSFindDistances(matrix, 0));
                long bfsListTime = MeasureTime(() =>
BFSFindDistancesWithAdjacencyList(adjList, 0));
                long dfsMatrixTime = MeasureTime(() => DFSFindDistances(matrix, 0));
                long dfsListTime = MeasureTime(() =>
DFSFindDistancesWithAdjacencyList(adjList, 0));

                Console.WriteLine($"BFS с матрицей:    {bfsMatrixTime,8} тиков");
                Console.WriteLine($"BFS со списками:   {bfsListTime,8} тиков");
                Console.WriteLine($"DFS с матрицей:    {dfsMatrixTime,8} тиков");
                Console.WriteLine($"DFS со списками:   {dfsListTime,8} тиков");

                // Анализ эффективности
                Console.WriteLine($"Эффективность списков над матрицей:");
                Console.WriteLine($"  BFS: {((double)bfsMatrixTime /
bfsListTime):F2}x");
                Console.WriteLine($"  DFS: {((double)dfsMatrixTime /
dfsListTime):F2}x");
            }
        }
```

```
    }

    // Вспомогательные функции

    static Dictionary<int, List<int>> MatrixToAdjacencyList(int[,] matrix)
    {
        int n = matrix.GetLength(0);
        Dictionary<int, List<int>> adjList = new Dictionary<int, List<int>>();

        for (int i = 0; i < n; i++)
        {
            adjList[i] = new List<int>();
            for (int j = 0; j < n; j++)
            {
                if (matrix[i, j] == 1)
                {
                    adjList[i].Add(j);
                }
            }
        }

        return adjList;
    }

    static int[] BFSFindDistances(int[,] G, int v)
    {
        int size_G = G.GetLength(0);
        int[] DIST = new int[size_G];

        for (int i = 0; i < size_G; i++)
            DIST[i] = -1;

        Queue<int> Q = new Queue<int>();
        Q.Enqueue(v);
        DIST[v] = 0;

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

            for (int i = 0; i < G.GetLength(0); i++)
            {
                if (G[currentV, i] == 1 && DIST[i] == -1)
                {
                    Q.Enqueue(i);
                    DIST[i] = DIST[currentV] + 1;
                }
            }
        }

        return DIST;
    }

    static int[] BFSFindDistancesWithAdjacencyList(Dictionary<int, List<int>>
adjList, int v)
    {
        int size_G = adjList.Count;
        int[] DIST = new int[size_G];

        for (int i = 0; i < size_G; i++)
            DIST[i] = -1;

        Queue<int> Q = new Queue<int>();
        Q.Enqueue(v);
        DIST[v] = 0;
```

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

            foreach (int neighbor in adjList[currentV])
            {
                if (DIST[neighbor] == -1)
                {
                    Q.Enqueue(neighbor);
                    DIST[neighbor] = DIST[currentV] + 1;
                }
            }
        }

        return DIST;
    }

    static long MeasureTime(Action algorithm)
    {
        var watch = Stopwatch.StartNew();
        algorithm();
        watch.Stop();
        return watch.ElapsedTicks;
    }

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

    static void PrintDistances(int[] distances)
    {
        for (int i = 0; i < distances.Length; i++)
        {
            string status = distances[i] == -1 ? "недостижима" :
distances[i].ToString();
            Console.WriteLine($"  Вершина {i}: расстояние = {status}");
        }
    }
}
```

Результаты работы программы

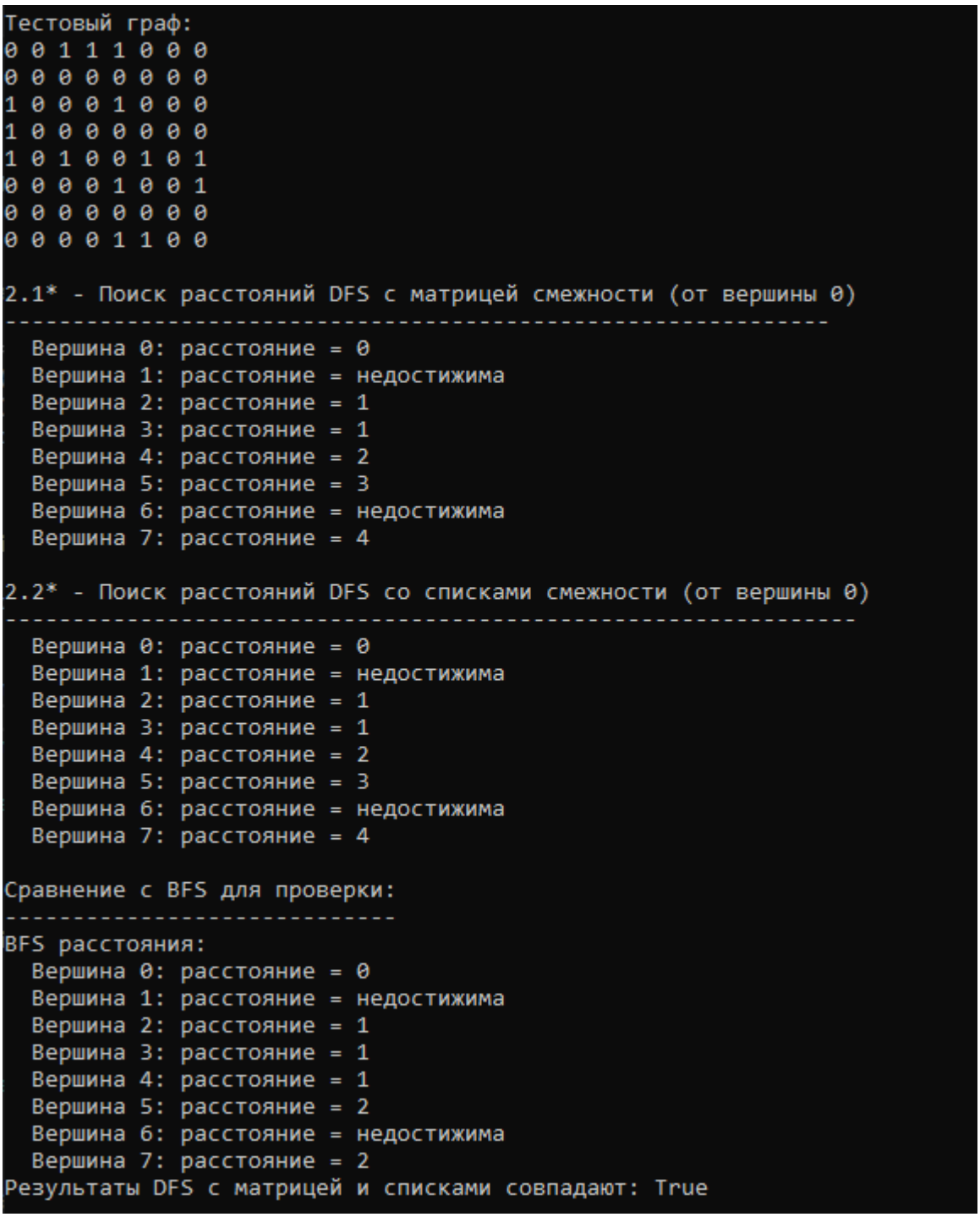

Рисунок 2 - Результат работы программы 2

```
2.3* - Сравнение производительности алгоритмов
------------------------------------------------
Сравнение времени выполнения (в тиках):
========================================

Граф: 100 вершин, плотность 10 %
BFS с матрицей:        3759 тиков
BFS со списками:       2749 тиков
DFS с матрицей:        1191 тиков
DFS со списками:        488 тиков
Эффективность списков над матрицей:
  BFS: 1,37x
  DFS: 2,44x

Граф: 100 вершин, плотность 30 %
BFS с матрицей:         860 тиков
BFS со списками:        365 тиков
DFS с матрицей:         857 тиков
DFS со списками:        179 тиков
Эффективность списков над матрицей:
  BFS: 2,36x
  DFS: 4,79x

Граф: 100 вершин, плотность 50 %
BFS с матрицей:        1026 тиков
BFS со списками:        249 тиков
DFS с матрицей:        1018 тиков
DFS со списками:        214 тиков
Эффективность списков над матрицей:
  BFS: 4,12x
  DFS: 4,76x

Граф: 500 вершин, плотность 10 %
BFS с матрицей:       15057 тиков
BFS со списками:       1618 тиков
DFS с матрицей:       14180 тиков
DFS со списками:       1832 тиков
Эффективность списков над матрицей:
  BFS: 9,31x
  DFS: 7,74x

Граф: 500 вершин, плотность 30 %
BFS с матрицей:       20210 тиков
BFS со списками:       4065 тиков
DFS с матрицей:       20916 тиков
DFS со списками:       5394 тиков
Эффективность списков над матрицей:
  BFS: 4,97x
  DFS: 3,88x

Граф: 500 вершин, плотность 50 %
BFS с матрицей:       24384 тиков
BFS со списками:       5506 тиков
DFS с матрицей:       24708 тиков
DFS со списками:       4690 тиков
Эффективность списков над матрицей:
  BFS: 4,43x
  DFS: 5,27x
```

Рисунок 3 - Результат работы программы 2

Вывод: В ходе лабораторной работы были успешно реализованы и исследованы алгоритмы поиска расстояний в графах на основе обхода в ширину (BFS) и обхода в глубину (DFS) с использованием двух различных структур данных - матрицы смежности и списков смежности.