

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №9
«Поиск расстояний в 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 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.

По заданию 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.

Ход работы:

Описание кода программы по заданию 1.1 - 1.2:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR9
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите размер матрицы: ");

            int size = Convert.ToInt32(Console.ReadLine());

            int[,] adjacencyMatrix = GenerateAdjacencyMatrix(size);

            Console.WriteLine("Матрица смежности для графа G1:");
            PrintMatrix(adjacencyMatrix);

            Console.Write("Введите вершину, с которой хотите начать обход:");

            int start = Convert.ToInt32(Console.ReadLine());

            BFS(adjacencyMatrix, size, start);
        }
        //Метод GenerateAdjacencyMatrix генерирует случайную матрицу
        //смежности для графа.
        private static int[,] GenerateAdjacencyMatrix(int size)
        {
            int[,] matrix = new int[size, size];
            Random random = new Random();
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    matrix[i, j] = random.Next(0, 2);
                }
            }
            return matrix;
        }
    }
}
```

```

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)
        {
            //для каждой пары вершин (i, j) генерируется случайное
число 0 или 1, которое указывает наличие или отсутствие ребра между
вершинами.

            matrix[i, j] = r.Next(2);
            matrix[j, i] = matrix[i, j];
        }
    }
}

return matrix;
}

//Метод PrintMatrix выводит матрицу смежности на экран.
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();
    }
}

```

```

    }
    Console.WriteLine();
}

//Метод BFS осуществляет поиск в ширину. Он использует очередь
для сохранения вершин, которые нужно посетить.

private static void BFS(int[,] adjacencyMatrix, int size, int
source)
{
    //Создается очередь queue
    Queue<int> queue = new Queue<int>();

    //массив visited для отслеживания посещенных вершин
    bool[] visited = new bool[size];

    //массив distance для хранения расстояния от исходной вершины
до каждой вершины графа.
    int[] distance = new int[size];

    //Исходная вершина помечается как посещенная и добавляется
в очередь.
    visited[source] = true;
    queue.Enqueue(source);

    //Пока очередь не пуста, извлекаем элемент из очереди и
перебираем его соседей в матрице смежности.
    while (queue.Count > 0)
    {
        int currentVertex = queue.Dequeue();

        //Перебираем соседей текущей вершины
        for (int i = 0; i < size; i++)
        {

```

```

        //Проверяем, является ли вершина с индексом i соседом
текущей вершины
        if (adjacencyMatrix[currentVertex, i] == 1
&& !visited[i])
        {
            //Помечаем соседнюю вершину как посещенную
            visited[i] = true;

            //Устанавливаем расстояние до соседней вершины
            distance[i] = distance[currentVertex] + 1;

            //Добавляем соседнюю вершину в очередь
            queue.Enqueue(i);
        }
    }

    //Выводим расстояния до всех вершин графа
    Console.WriteLine("Расстояния до вершин графа:");
    for (int i = 0; i < size; i++)
    {
        Console.WriteLine("Расстояние до вершины {0}:{1}", i,
distance[i]);
    }
}
}
}

```

Описание кода программы по заданию 1.3:

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

```

```
namespace _1._3
{
    class Graph
    {
        List<List<int>> adjacencyList;

        //Конструктор класса принимает размер графа и инициализирует
        //пустой список adjacencyList, который будет хранить смежные вершины для
        //каждой вершины графа.
        public Graph(int size)
        {
            adjacencyList = new List<List<int>>();
            for (int i = 0; i < size; i++)
            {
                adjacencyList.Add(new List<int>());
            }
        }

        //Метод AddEdge принимает две вершины - from и to, и добавляет
        //их в список смежности друг друга.
        //Это позволяет установить связь между вершинами графа.
        public void AddEdge(int from, int to)
        {
            adjacencyList[from].Add(to);
            adjacencyList[to].Add(from);
        }

        //Метод GetNeighbors принимает вершину графа и возвращает список
        //ее смежных вершин.
        public List<int> GetNeighbors(int vertex)
        {
            return adjacencyList[vertex];
        }
    }
}
```

//Метод PrintGraph выводит на экран информацию о графе. Он перебирает все вершины графа и для каждой вершины выводит ее номер //, а затем перебирает все смежные вершины и выводит их номера.

```
public void PrintGraph()
{
    for (int i = 0; i < adjacencyList.Count; i++)
    {
        Console.WriteLine($"Вершина {i + 1}: ");
        foreach (var vertex in adjacencyList[i])
        {
            Console.WriteLine($"{{vertex + 1}} ");
        }
        Console.WriteLine();
    }
}
```

//метод FindDistance, который используется для нахождения минимального расстояния между двумя вершинами в неориентированном графе.

```
public int FindDistance(int from, int to)
{
```

//метод проверяет, являются ли вершины from и to одинаковыми. Если да, то возвращается 0, так как расстояние от вершины до самой себя равно 0.

```
    if (from == to)
    {
        return 0;
    }
```

//Создается очередь queue

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

//массив visited для отслеживания посещенных вершин

```
bool[] visited = new bool[adjacencyList.Count];
```


//массив distance для хранения расстояния от вершины from до каждой вершины в графе.

```
int[] distance = new int[adjacencyList.Count];
```

//Исходная вершина from помечается как посещенная и добавляется в очередь.

```
queue.Enqueue(from);
```

```
visited[from] = true;
```

//В цикле, пока очередь не пуста, извлекается вершина current из очереди.

```
while (queue.Count > 0)
```

```
{
```

```
    int current = queue.Dequeue();
```

//Для каждого соседа neighbor данной вершины current, проверяется, является ли он непосещенным.

```
    foreach (int neighbor in adjacencyList[current])
```

```
    {
```

//Если соседняя вершина не посещена, то она добавляется в очередь, помечается как посещенная,

//вычисляется расстояние от исходной вершины from до данной соседней вершины neighbor как расстояние от текущей вершины current + 1.

```
        if (!visited[neighbor])
```

```
        {
```

```
            queue.Enqueue(neighbor);
```

```
            visited[neighbor] = true;
```

```
            distance[neighbor] = distance[current] + 1;
```

//Если соседняя вершина равна вершине to, то найдено минимальное расстояние от вершины from до вершины to и это расстояние возвращается.

```
            if (neighbor == to)
            {
                return distance[to];
            }
        }
    }
}
```

 //Если после прохода по всем вершинам в графе не удалось достичь вершины to, то возвращается -1, что означает, что пути между этими вершинами не существует.

```
        return -1;
```

```
    }
```

```
}
```

```
internal class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        Console.Write("Введите размер графа: ");
```

```
        int size = Convert.ToInt32(Console.ReadLine());
```

```
        Graph graph = GenerateAdjacencyList(size);
```

```
        Console.WriteLine("Список смежности для графа G1:");
```

```
        graph.PrintGraph();
```

```
        Console.Write("Введите вершину от: ");
```

```
        int from = Convert.ToInt32(Console.ReadLine()) - 1;
```

```

        Console.WriteLine("Введите вершину до: ");
        int to = Convert.ToInt32(Console.ReadLine()) - 1;

        int distance = graph.FindDistance(from, to);

        if (distance == -1)
        {
            Console.WriteLine("Пути между указанными вершинами нет.");
        }
        else
        {
            Console.WriteLine($"Расстояние между вершинами {from +
1} и {to + 1}: {distance}");
        }
    }

    //Функция GenerateAdjacencyList генерирует случайный граф
заданного размера size в виде списка смежности.
    private static Graph GenerateAdjacencyList(int size)
    {
        Random r = new Random();
        Graph graph = new Graph(size);

        //двойной цикл, который перебирает все возможные комбинации
вершин графа.
        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++)
            {
                //Если сгенерированное число равно 1, то вызывается
метод AddEdge объекта graph для добавления ребра между вершинами i и j.
                if (r.Next(2) == 1)
                {

```

```

graph.AddEdge(i, j);
    }
}
}

//После завершения циклов возвращается объект graph, содержащий
случайно сгенерированный граф в виде списка смежности.

return graph;
}
}
}

```

Описание кода программы по заданию 2.2:

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace _2._2
{
    class Graph
    {
        //поле adjacencyList типа List<List<int>>, которое
        представляет список списков для представления смежности графа.

        List<List<int>> adjacencyList;

        //Конструктор класса Graph принимает размер графа и
        инициализирует adjacencyList пустыми списками для каждой вершины
        графа.

        public Graph(int size)
        {
            adjacencyList = new List<List<int>>();

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

```

```

        {
            adjacencyList.Add(new List<int>());
        }
    }

```

//Метод AddEdge служит для добавления ребра между двумя вершинами графа. Он принимает два параметра: вершину from и вершину to,

//и добавляет to в список смежности from, а также добавляет from в список смежности to.

```

public void AddEdge(int from, int to)
{
    adjacencyList[from].Add(to);
    adjacencyList[to].Add(from);
}

```

//Метод GetNeighbors принимает вершину графа и возвращает список смежных с ней вершин.

```

public List<int> GetNeighbors(int vertex)
{
    return adjacencyList[vertex];
}

```

//Метод PrintGraph выводит на консоль информацию о графе. Он перебирает все вершины графа и для каждой вершины выводит ее номер и список смежных вершин.

```

public void PrintGraph()
{
    for (int i = 0; i < adjacencyList.Count; i++)
    {
        Console.WriteLine($"Вершина {i + 1}: ");
        foreach (var vertex in adjacencyList[i])
        {
            Console.WriteLine($"{vertex + 1} ");
        }
    }
}

```

```
    }  
    Console.WriteLine();  
}  
}
```

//Метод FindDistance используется для нахождения минимального расстояния между двумя вершинами графа from и to.

```
public int FindDistance(int from, int to)  
{  
    //Сначала метод проверяет, являются ли вершины from и to  
    //одинаковыми. Если да, то возвращается 0, так как расстояние от вершины  
    //до самой себя равно 0.  
    if (from == to)  
    {  
        return 0;  
    }  
  
    //создается стек stack  
    Stack<int> stack = new Stack<int>();  
  
    //массив visited для отслеживания посещенных вершин  
    bool[] visited = new bool[adjacencyList.Count];  
  
    //массив distance для хранения расстояния от вершины from  
    //до каждой вершины в графе.  
    int[] distance = new int[adjacencyList.Count];  
  
    //Исходная вершина from помечается как посещенная и  
    //добавляется в стек.  
    stack.Push(from);  
    visited[from] = true;
```

```

        //В цикле, пока стек не пуст, извлекается вершина current
из стека.

        while (stack.Count > 0)
        {
            int current = stack.Pop();

            //Для каждого соседа neighbor данной вершины current,
            проверяется, является ли он непосещенным.

            foreach (int neighbor in adjacencyList[current])
            {
                //Если соседняя вершина не посещена, то она
                добавляется в стек, помечается как посещенная, вычисляется расстояние
                от

                //исходной вершины from до данной соседней вершины
                neighbor как расстояние от текущей вершины current + 1.

                if (!visited[neighbor])
                {
                    stack.Push(neighbor);

                    visited[neighbor] = true;

                    distance[neighbor] = distance[current] + 1;

                    //Если соседняя вершина равна вершине to, то
                    найдено минимальное расстояние от вершины from до вершины to и это
                    расстояние возвращается.

                    if (neighbor == to)
                    {
                        return distance[to];
                    }
                }
            }
        }

        //Если после прохода по всем вершинам в графе не удалось
        достичь вершины to, то возвращается -1, что означает, что пути между
        этими вершинами не существует.

```

```

        return -1;
    }
}

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

        Graph graph = GenerateAdjacencyList(size);

        Console.WriteLine("Список смежности для графа G1:");
        graph.PrintGraph();

        Console.Write("Введите вершину от: ");
        int from = Convert.ToInt32(Console.ReadLine()) - 1;

        Console.Write("Введите вершину до: ");
        int to = Convert.ToInt32(Console.ReadLine()) - 1;

        int distance = graph.FindDistance(from, to);

        if (distance == -1)
        {
            Console.WriteLine("Нет пути между указанными
вершинами.");
        }
        else
    }
}

```



```

        {
            Console.WriteLine($"Расстояние между вершинами {from +
1} и {to + 1}: {distance}");
        }
    }

    //Функция GenerateAdjacencyList генерирует случайный граф
заданного размера size в виде списка смежности.

    private static Graph GenerateAdjacencyList(int size)
    {
        Random r = new Random();

        Graph graph = new Graph(size);

        //двойной цикл, который перебирает все возможные
комбинации вершин графа.
        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++)
            {
                //Если сгенерированное число равно 1, то
вызывается метод AddEdge объекта graph для добавления ребра между
вершинами i и j.

                if (r.Next(2) == 1)
                {
                    graph.AddEdge(i, j);
                }
            }
        }

        //После завершения циклов возвращается объект graph,
содержащий случайно сгенерированный граф в виде списка смежности.

        return graph;
    }
}

```

```
}
```

Описание кода программы по заданию 2.1 и 2.3:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace _2._1

{

    internal class Program

    {

        static void Main(string[] args)

        {

            Console.Write("Введите размер матрицы: ");

            int size = Convert.ToInt32(Console.ReadLine());

            int[,] adacencyMatrix = GenerateAdjacencyMatrix(size);

            Console.WriteLine("Матрица смежности для графа G1:");

            PrintMatrix(adacencyMatrix);

            Console.Write("Введите вершину, с которой хотите начать
обход: ");

            int start = Convert.ToInt32(Console.ReadLine());

            DateTime startTime = DateTime.Now;

            Console.WriteLine("Результат обхода в ширину:");

            BFS(adacencyMatrix, size, start);
```

```

        DateTime endTime = DateTime.Now;

        TimeSpan listTime = endTime - startTime;

        Console.WriteLine("Время работы обхода в ширину: " +
listTime.TotalMilliseconds + " миллисекунд" + "\n");

        DateTime startTime2 = DateTime.Now;

        Console.WriteLine("Результат обхода в глубину:");
        DFS(adacencyMatrix, size, start);

        DateTime endTime2 = DateTime.Now;
        TimeSpan listTime2 = endTime2 - startTime2;

        Console.WriteLine("Время работы обхода в глубину: " +
listTime2.TotalMilliseconds + " миллисекунд");
    }

    //Метод GenerateAdjacencyMatrix генерирует случайную матрицу
смежности для графа
    private static int[,] GenerateAdjacencyMatrix(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)
        {
            //для каждой пары вершин (i,j) генерируется
случайное число
            //0 или 1, которое указывает наличие или
отсутствие ребра между вершинами
            matrix[i, j] = r.Next(2);
            matrix[j, i] = matrix[i, j];
        }
    }
}
return matrix;
}

```

//Метод PrintMatrix выводит матрицу смежности на экран.

```

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();
    }
    Console.WriteLine();
}

```

//Метод BFS осуществляет поиск в ширину. Он использует очередь для сохранения вершин, которые нужно посетить.

```

private static void BFS(int[,] adjacencyMatrix, int size, int
source)

```

```

{
    //Создается очередь queue
    Queue<int> queue = new Queue<int>();

    //массив visited для отслеживания посещенных вершин
    bool[] visited = new bool[size];

    //массив distance для хранения расстояния от исходной
    //вершины до каждой вершины графа.
    int[] distance = new int[size];

    //Исходная вершина помечается как посещенная и добавляется
    //в очередь.
    visited[source] = true;
    queue.Enqueue(source);

    //Пока очередь не пуста, извлекаем элемент из очереди и
    //перебираем его соседей в матрице смежности.
    while (queue.Count > 0)
    {
        int currentVertex = queue.Dequeue();

        //Перебираем соседей текущей вершины
        for (int i = 0; i < size; i++)
        {
            //Проверяем, является ли вершина с индексом i
            //соседом текущей вершины
            if (adjacencyMatrix[currentVertex, i] == 1
            && !visited[i])
            {
                //Помечаем соседнюю вершину как посещенную
                visited[i] = true;
            }
        }
    }
}

```

```

        //Устанавливаем расстояние до соседней вершины
        distance[i] = distance[currentVertex] + 1;

        //Добавляем соседнюю вершину в очередь
        queue.Enqueue(i);
    }
}

// Выводим расстояния до всех вершин графа
for (int i = 0; i < size; i++)
{
    Console.WriteLine("Расстояние до вершины {0}:{1}", i,
distance[i]);
}
}

```

//Метод DFS осуществляет поиск в глубину. Он использует стек для сохранения вершин, которые нужно посетить.

```

private static void DFS(int[,] adjacencyMatrix, int size, int
source)

```

```

{
    //Создается стек stack
    Stack<int> stack = new Stack<int>();

    //массив visited для отслеживания посещенных вершин
    bool[] visited = new bool[size];

    //массив distance для хранения расстояния от исходной
вершины до каждой вершины графа.
    int[] distance = new int[size];

```

```

//Помечаем вершину source как посещенную и добавляем ее в
стек
visited[source] = true;
stack.Push(source);

//Пока стек не пуст, извлекаем вершину из стека и
перебираем ее соседей в матрице смежности.
while (stack.Count > 0)
{
    int currentVertex = stack.Pop();

    // Перебираем соседей текущей вершины и добавляем их в
стек
    for (int i = 0; i < size; i++)
    {
        // Если соседняя вершина еще не посещена и
существует ребро между текущей вершиной и соседней вершиной,
        // то мы помечаем ее как посещенную, добавляем в
стек и вычисляем расстояние от исходной вершины до данной вершины.
        if (!visited[i] && adjacencyMatrix[currentVertex,
i] != 0)
        {
            stack.Push(i);
            visited[i] = true;

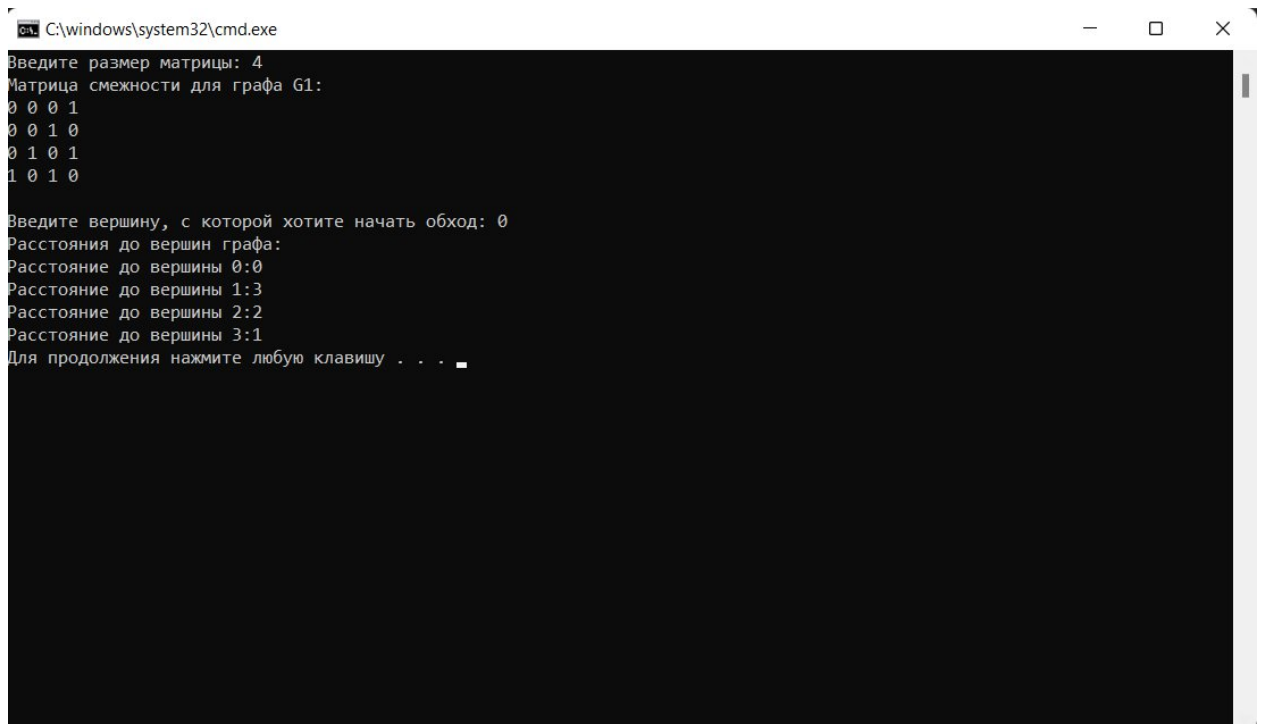
            //Вычисляем расстояние от исходной вершины до
данной вершины
            distance[i] = distance[currentVertex] + 1;
        }
    }
}

// Выводим расстояние от исходной вершины до каждой
вершины

```

```
        for (int i = 0; i < size; i++)
        {
            Console.WriteLine($"Расстояние до вершины
{i}:{distance[i]}");
        }
    }
}
```

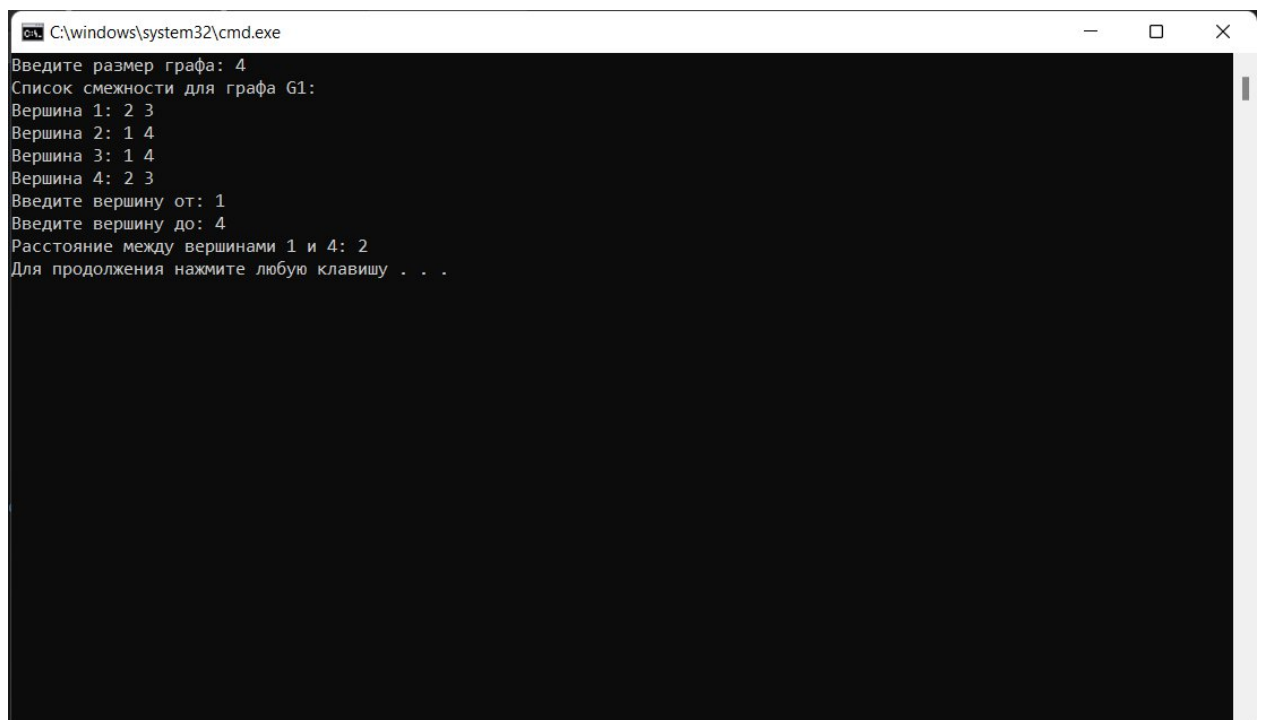

Результат работы программы 1.1-1.2:



```
C:\windows\system32\cmd.exe
Введите размер матрицы: 4
Матрица смежности для графа G1:
0 0 0 1
0 0 1 0
0 1 0 1
1 0 1 0

Введите вершину, с которой хотите начать обход: 0
Расстояния до вершин графа:
Расстояние до вершины 0:0
Расстояние до вершины 1:3
Расстояние до вершины 2:2
Расстояние до вершины 3:1
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 1.3:



```
C:\windows\system32\cmd.exe
Введите размер графа: 4
Список смежности для графа G1:
Вершина 1: 2 3
Вершина 2: 1 4
Вершина 3: 1 4
Вершина 4: 2 3
Введите вершину от: 1
Введите вершину до: 4
Расстояние между вершинами 1 и 4: 2
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 2.2:

```
C:\windows\system32\cmd.exe
Введите размер графа: 4
Список смежности для графа G1:
Вершина 1: 3 4
Вершина 2: 3
Вершина 3: 1 2 4
Вершина 4: 1 3
Введите вершину от: 1
Введите вершину до: 2
Расстояние между вершинами 1 и 2: 2
Для продолжения нажмите любую клавишу . . .
```

Результат работы программ 2.1 и 2.3:

```
C:\windows\system32\cmd.exe
Введите размер матрицы: 4
Матрица смежности для графа G1:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Введите вершину, с которой хотите начать обход: 0
Результат обхода в ширину:
Расстояние до вершины 0:0
Расстояние до вершины 1:1
Расстояние до вершины 2:1
Расстояние до вершины 3:1
Время работы обхода в ширину: 7,8625 миллисекунд

Результат обхода в глубину:
Расстояние до вершины 0:0
Расстояние до вершины 1:1
Расстояние до вершины 2:1
Расстояние до вершины 3:1
Время работы обхода в глубину: 1,0028 миллисекунд
Для продолжения нажмите любую клавишу . . .
```

Вывод: в ходе лабораторной работы были получены навыки работы с поиском расстояний в графе. Научились осуществлять процедуру поиска расстояний, реализованную в соответствии с приведенным описанием. При реализации алгоритма в качестве очереди использовался класс **queue** из стандартной библиотеки C#. Также научились реализовывать процедуру

поиска расстояний на основе обхода в глубину и ширину для графа, представленного в виде матрицы смежности и списка смежности. А также оценили время работы реализаций алгоритмов поиска расстояний на основе обхода в глубину и ширину для графов разных порядков. И в ходе данного «эксперимента» было выявлено, что алгоритм обхода в глубину выполнил свою задачу быстрее, чем алгоритм обхода в ширину.