МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №7

«Обход графа в глубину»

По дисциплине «Л и ОА в ИЗ»

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

Приняли: Юрова О.В.
Акифьев И.В.

2023

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 обхода в глубину, реализованную в соответствии с приведенным выше описанием.
3.* Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

По заданию 2:

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

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 _7laba

{

    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);


            bool[] visited = new bool[size];


            Console.WriteLine("Обход в глубину:");


            //происходит обход графа в глубину. Для каждой вершины графа, если она еще не посещена, проверяется, является ли она изолированной (не имеет ребер с другими вершинами).

            //Если вершина изолированная, вызывается метод DepthFirstSearchIsolatedVertex для ее обхода. В противном случае вызывается метод DepthFirstSearchNonRecursive для обхода графа.
```

```
            for (int startVertexIndex = 0; startVertexIndex < size;
startVertexIndex++)
            {
                if (!visited[startVertexIndex])
                {
                    if (IsIsolatedVertex(startVertexIndex,
adjacencyMatrix))
                    {
                        Console.WriteLine("Вершина №" +
(startVertexIndex + 1) + " не имеет ребро с другими вершинами");

DepthFirstSearchIsolatedVertex(startVertexIndex);
                    }
                    else
                    {
                        DepthFirstSearch(startVertexIndex,
adjacencyMatrix, visited);
                    }
                }
            }
        }
        //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();
        }
        //Данный код реализует алгоритм обхода графа в глубину
        static void DepthFirstSearch(int startVertexIndex, int[,] adjacencyMatrix, bool[] visited)
```

```
        {
            //Вначале функция выводит на экран информацию о посещении стартовой вершины.
            Console.WriteLine("Посещена вершина №" + (startVertexIndex + 1));

            //Затем она помечает данную вершину как посещенную, устанавливая соответствующий элемент в массиве visited в значение true.
            visited[startVertexIndex] = true;

            //Затем происходит цикл, который перебирает все вершины, смежные со стартовой вершиной.
            //Если в матрице смежности значение равно 1 (т.е. вершины связаны) и данная вершина еще не была посещена (значение в массиве visited равно false),
            //то вызывается рекурсивно функция DepthFirstSearch для данной вершины.
            for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)
            {
                if (adjacencyMatrix[startVertexIndex,i] == 1 && !visited[i])
                {
                    DepthFirstSearch(i, adjacencyMatrix, visited);
                }
            }
        }
        //Метод DepthFirstSearchIsolatedVertex выводит сообщение о посещении изолированной вершины.
        static void DepthFirstSearchIsolatedVertex(int vertex)
        {
            Console.WriteLine("Посещена изолированная вершина №" + (vertex + 1));
        }
```

```
        //Метод IsIsolatedVertex проверяет, является ли заданная вершина изолированной, путем проверки наличия ребер с другими вершинами в матрице смежности.

        static bool IsIsolatedVertex(int vertex, int[,] adjacencyMatrix)

        {

            for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)

            {

                if (adjacencyMatrix[vertex, i] == 1)

                   {

                return false;

                   }

            }

              return true;

         }

    }

}
```

<u>Описание кода программы по заданию 1.3:</u>

```
using System;

using System.Collections.Generic;


class Graph

{

    private List<List<int>> adjacencyList;


    public Graph()

    {

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

    }
```

```
    public void AddEdge(int from, int to)
    {
        while (adjacencyList.Count <= from || adjacencyList.Count <=
to)
        {
            adjacencyList.Add(new List<int>());
        }
        adjacencyList[from].Add(to);
        adjacencyList[to].Add(from);
    }

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

    //Вывод графа на экран
    public void PrintGraph()
    {
        for (int i = 0; i < adjacencyList.Count; i++)
        {
            Console.Write("Вершина " + (i + 1) + ": ");
            foreach (int neighbor in adjacencyList[i])
            {
                Console.Write((neighbor + 1) + " ");
            }
            Console.WriteLine();
        }
    }
}
```

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

        Graph graph = GenerateAdjacencyList(size);

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

        bool[] visited = new bool[size];

        for (int startVertexIndex = 0; startVertexIndex < size;
startVertexIndex++)
        {
            if (!visited[startVertexIndex])
            {
                Console.WriteLine("Обход в глубину, начиная с вершины
" + (startVertexIndex + 1) + ":");
                DepthFirstSearch(startVertexIndex, graph, visited);
            }
        }
    }

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

```
        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++)
            {
                int randomEdge = r.Next(2);
                if (randomEdge == 1)
                {
                    graph.AddEdge(i, j);
                }
            }
        }

        return graph;
    }

    //алгоритм обхода графа в глубину
    static void DepthFirstSearch(int startVertexIndex, Graph graph, bool[] visited)
    {
        //Вначале функция помечает текущую вершину как посещенную
        visited[startVertexIndex] = true;

        //Затем она получает список смежных вершин для текущей вершины с помощью метода GetNeighbors
        List<int> neighbors = graph.GetNeighbors(startVertexIndex);

        //Затем функция проходит по всем смежным вершинам и для каждой смежной вершины проверяет, была ли она уже посещена.
        //Если смежная вершина не была посещена, то функция рекурсивно вызывает себя для этой смежной вершины, передавая ее в качестве новой стартовой вершины.
```

```
        foreach (int neighbor in neighbors)
        {
            if (!visited[neighbor])
            {
                //При каждом рекурсивном вызове функция выводит на экран ребро, которое соединяет текущую вершину и смежную вершину.
                Console.WriteLine((startVertexIndex + 1) + " -> " + (neighbor + 1));
                DepthFirstSearch(neighbor, graph, visited);
            }
        }
    }
}
```

<u>Описание кода программы по заданию 2:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _7laba
{
    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);


            //создается массив visited, который будет использоваться для отслеживания посещенных вершин

            bool[] visited = new bool[size];


            Console.WriteLine("Обход в глубину:");


            //происходит обход графа в глубину. Для каждой вершины графа, если она еще не посещена, проверяется, является ли она изолированной (не имеет ребер с другими вершинами).

            //Если вершина изолированная, вызывается метод DepthFirstSearchIsolatedVertex для ее обхода. В противном случае вызывается метод DepthFirstSearchNonRecursive для обхода графа.

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

            {

                if (!visited[startVertexIndex])

                {

                    if (IsIsolatedVertex(startVertexIndex, adjacencyMatrix))

                    {

                        Console.WriteLine("Вершина №" + (startVertexIndex + 1) + " не имеет ребро с другими вершинами");


DepthFirstSearchIsolatedVertex(startVertexIndex);

                    }

                    else

                    {

                        DepthFirstSearchNonRecursive(startVertexIndex, adjacencyMatrix, visited);

                    }
```

```
                }

            }

        }

        //Метод DepthFirstSearchNonRecursive осуществляет обход графа в глубину с использованием стека.

        private static void DepthFirstSearchNonRecursive(int startVertexIndex, int[,] adjacencyMatrix, bool[] visited)

        {

            //Начиная с заданной вершины, он помещает ее в стек

            Stack<int> stack = new Stack<int>();

            stack.Push(startVertexIndex);


            //затем до тех пор, пока стек не пуст, извлекает текущую вершину из стека.

            while (stack.Count > 0)

            {

                int currentVertexIndex = stack.Pop();

                //Если эта вершина еще не посещена, она отмечается как посещенная и выводится на экран. Затем происходит проверка всех смежных вершин текущей вершины и,

                //если они еще не посещены, они добавляются в стек.

                if (!visited[currentVertexIndex])

                {

                    Console.WriteLine("Посещена вершина №" + (currentVertexIndex + 1));

                    visited[currentVertexIndex] = true;


                    for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)

                    {

                        if (adjacencyMatrix[currentVertexIndex, i] == 1 && !visited[i])

                        {
```

```
                        stack.Push(i);
                    }
                }
            }
        }
    }
    //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 DepthFirstSearchIsolatedVertex выводит сообщение о посещении изолированной вершины.
        static void DepthFirstSearchIsolatedVertex(int vertex)
        {
            Console.WriteLine("Посещена изолированная вершина №" + (vertex + 1));
        }

        //Метод IsIsolatedVertex проверяет, является ли заданная вершина изолированной, путем проверки наличия ребер с другими вершинами в матрице смежности.
        static bool IsIsolatedVertex(int vertex, int[,] adjacencyMatrix)
        {
            for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)
            {
                if (adjacencyMatrix[vertex, i] == 1)
                {
```

```
                    return false;
                }
            }
            return true;
        }
    }
}
```

Дополнительное задание:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _7laba
{
    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);

            bool[] visited = new bool[size];
```

```
            Console.Write("Введите номер вершины, с которой хотите
начать обход: ");

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



            Console.WriteLine("Обход в глубину рекурсивно:");



            //происходит обход графа в глубину. Для каждой вершины
графа, если она еще не посещена, проверяется, является ли она
изолированной (не имеет ребер с другими вершинами).

            //Если вершина изолированная, вызывается метод
DepthFirstSearchIsolatedVertex для ее обхода. В противном случае
вызывается метод DepthFirstSearchNonRecursive для обхода графа.

            if (IsIsolatedVertex(startVertex, adjacencyMatrix))

            {

                Console.WriteLine("Вершина №" + (startVertex + 1) + "
не имеет ребро с другими вершинами");

                DepthFirstSearchIsolatedVertex(startVertex);

            }

            else

            {

                DepthFirstSearch(startVertex, adjacencyMatrix,
visited);

            }

            Console.WriteLine("\n" + "Обход в глубину не
рекурсивно:");



            Array.Clear(visited, 0, visited.Length); // Сброс массива
посещений перед использованием для не рекурсивного обхода



            //происходит обход графа в глубину. Для каждой вершины
графа, если она еще не посещена, проверяется, является ли она
изолированной (не имеет ребер с другими вершинами).

            //Если вершина изолированная, вызывается метод
DepthFirstSearchIsolatedVertex для ее обхода. В противном случае
вызывается метод DepthFirstSearchNonRecursive для обхода графа.
```

```
            if (IsIsolatedVertex(startVertex, adjacencyMatrix))
            {
                Console.WriteLine("Вершина №" + (startVertex + 1) + " не имеет ребро с другими вершинами");
                DepthFirstSearchIsolatedVertex(startVertex);
            }
            else
            {
                 DepthFirstSearchNonRecursive(startVertex, adjacencyMatrix, visited);
            }
        }


        //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.
```

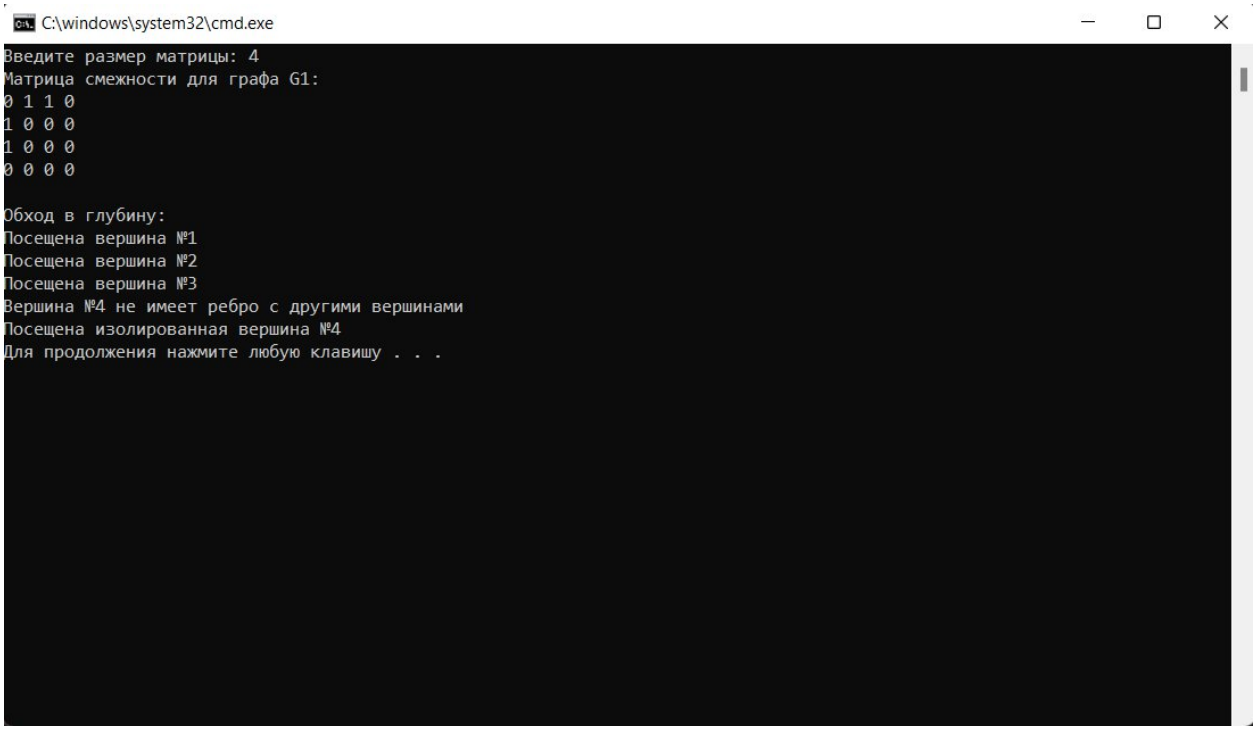
```
                    matrix[i, j] = r.Next(2);
                    matrix[j, i] = matrix[i, j];
                }
            }
        }
        return matrix;
    }
    //Метод PrintMatrix выводит матрицу смежности на экран.
    static void PrintMatrix(int[,] matrix)
    {
        int size = matrix.GetLength(0);

        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                Console.Write(matrix[i, j] + " ");
            }
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    //Данный код реализует алгоритм обхода графа в глубину
    static void DepthFirstSearch(int startVertexIndex, int[,]
adjacencyMatrix, bool[] visited)
    {
        //Вначале функция выводит на экран информацию о посещении
стартовой вершины.
        Console.WriteLine("Посещена вершина №" + (startVertexIndex
+ 1));
```

```
            //Затем она помечает данную вершину как посещенную,
устанавливая соответствующий элемент в массиве visited в значение
true.

            visited[startVertexIndex] = true;



            //Затем происходит цикл, который перебирает все вершины,
смежные со стартовой вершиной.

            //Если в матрице смежности значение равно 1 (т.е. вершины
связаны) и данная вершина еще не была посещена (значение в массиве
visited равно false),

            //то вызывается рекурсивно функция DepthFirstSearch для
данной вершины.

            for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)

            {

                if (adjacencyMatrix[startVertexIndex, i] == 1
&& !visited[i])

                {

                    DepthFirstSearch(i, adjacencyMatrix, visited);

                }

            }

        }

        //Метод DepthFirstSearchIsolatedVertex выводит сообщение о
посещении изолированной вершины.

        static void DepthFirstSearchIsolatedVertex(int vertex)

        {

            Console.WriteLine("Посещена изолированная вершина №" +
(vertex + 1));

        }

        //Метод IsIsolatedVertex проверяет, является ли заданная
вершина изолированной, путем проверки наличия ребер с другими
вершинами в матрице смежности.

        static bool IsIsolatedVertex(int vertex, int[,]
adjacencyMatrix)

        {

            for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)
```

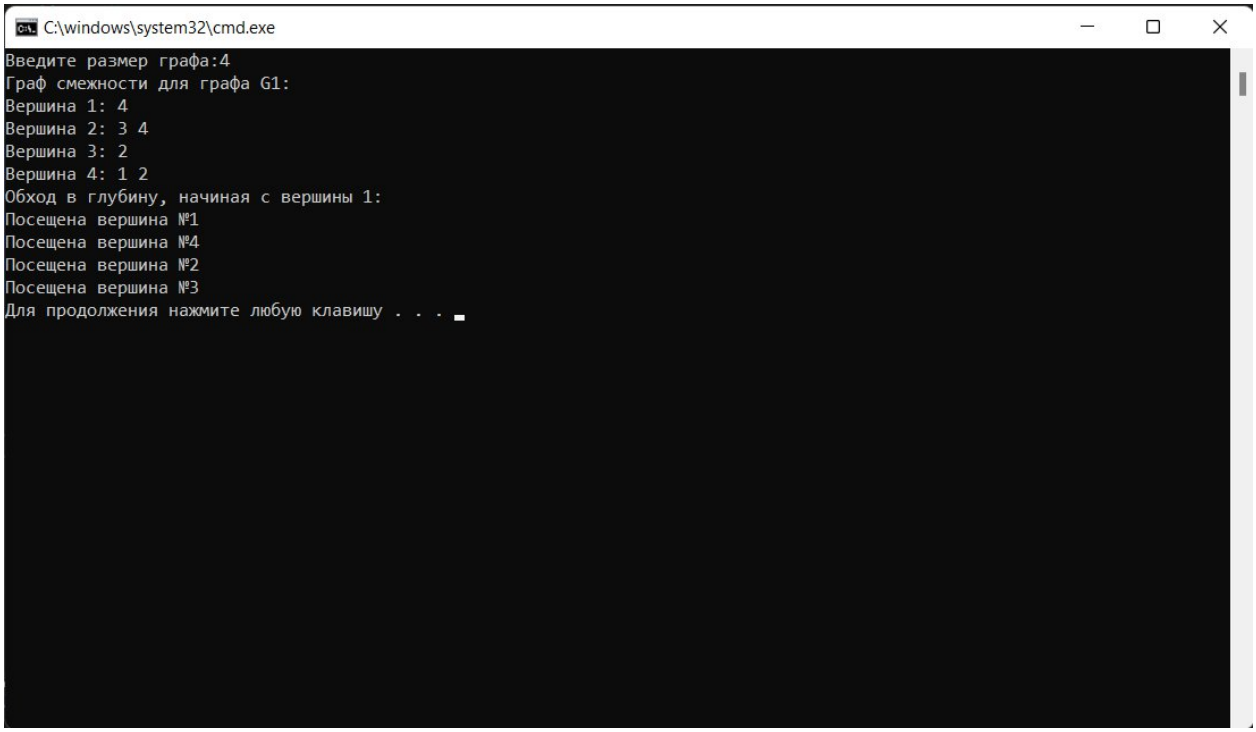
```
            {
                if (adjacencyMatrix[vertex, i] == 1)
                {
                    return false;
                }
            }
            return true;
        }
        private static void DepthFirstSearchNonRecursive(int
startVertexIndex, int[,] adjacencyMatrix, bool[] visited)
        {
            //Начиная с заданной вершины, он помещает ее в стек
            Stack<int> stack = new Stack<int>();
            stack.Push(startVertexIndex);

            //затем до тех пор, пока стек не пуст, извлекает текущую
вершину из стека.
            while (stack.Count > 0)
            {
                int currentVertexIndex = stack.Pop();
                //Если эта вершина еще не посещена, она отмечается как
посещенная и выводится на экран. Затем происходит проверка всех
смежных вершин текущей вершины и,
                //если они еще не посещены, они добавляются в стек.
                if (!visited[currentVertexIndex])
                {
                    Console.WriteLine("Посещена вершина №" +
(currentVertexIndex + 1));
                    visited[currentVertexIndex] = true;

                    for (int i = 0; i < adjacencyMatrix.GetLength(1);
i++)
```

```
                    {
                        if (adjacencyMatrix[currentVertexIndex, i] ==
1 && !visited[i])
                        {
                            stack.Push(i);
                        }
                    }
                }
            }
        }
    }
}
```

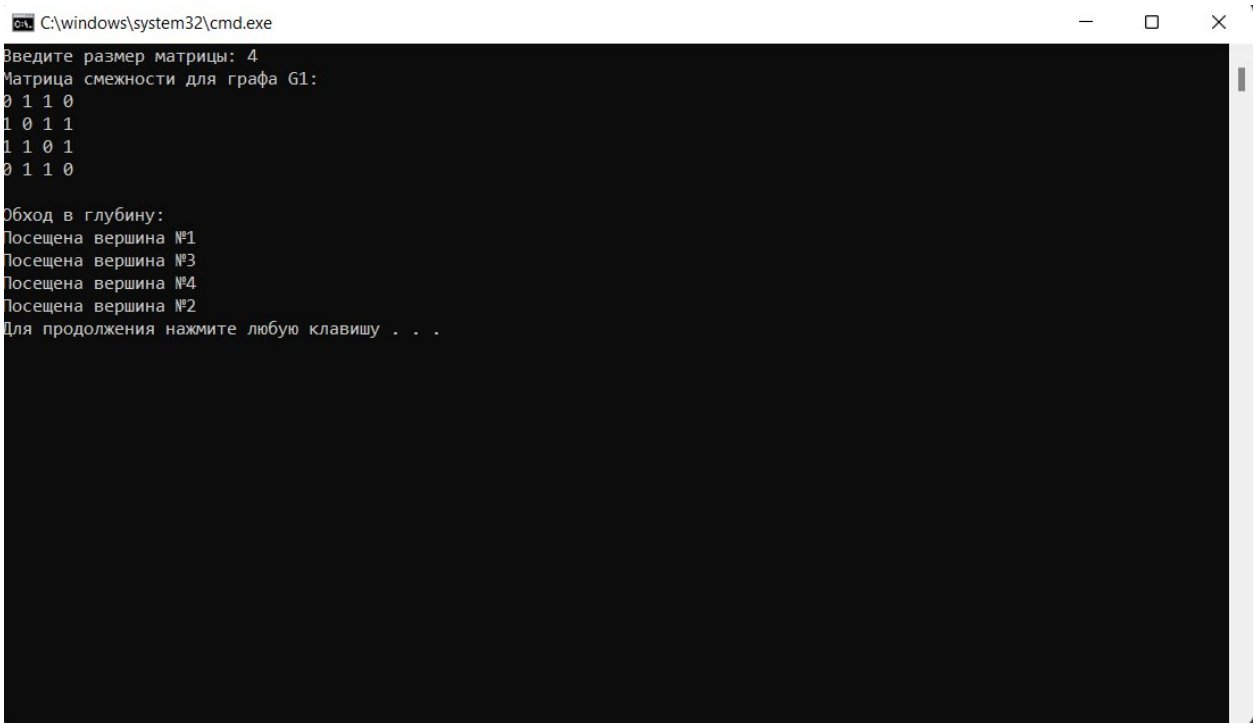
Результат работы программы 1.1-1.2:



Результат работы программы 1.3:



Результат работы программы 2:



Вывод: в ходе лабораторной работы мы научились осуществлять процедуру обхода в глубину для графа, представленного в виде матрицы смежности и списка смежности. А также преобразовывать рекурсивную реализацию обхода графа к не рекурсивной.