

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.

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;  
}  
}
```

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

номера. //, а затем перебирает все смежные вершины и выводит их номера.

```
public void PrintGraph()
{
    for (int i = 0; i < adjacencyList.Count; i++)
```



```

        {
            Console.Write($"Вершина {i + 1}: ");
            foreach (var vertex in adjacencyList[i])
            {
                Console.Write($"{vertex + 1} ");
            }
            Console.WriteLine();
        }
    }
}

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("Граф смежности для графа G1:");
        graph.PrintGraph();

        //массив visited, который используется для отслеживания
        //посещенных вершин.
        bool[] visited = new bool[size];

        //цикл, который проходит по всем вершинам графа.
        for (int startVertexIndex = 0; startVertexIndex < size;
startVertexIndex++)
        {

```

```

        //Если текущая вершина не была посещена, то
        выполняется обход в глубину с использованием функции DepthFirstSearch

        if (!visited[startVertexIndex])
        {
            Console.WriteLine("Обход в глубину, начиная с
            вершины " + (startVertexIndex + 1) + ":");

            DepthFirstSearch(startVertexIndex, graph,
            visited);
        }
    }

    //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;
```

```
    }
```

```
    //Функция DepthFirstSearch осуществляет поиск в глубину,  
    начиная с заданной стартовой вершины startVertexIndex.
```

```
    //Входные параметры функции - граф graph и массив visited,  
    отслеживающий посещенные вершины.
```

```
    static void DepthFirstSearch(int startVertexIndex, Graph  
    graph, bool[] visited)
```

```
    {
```

```
        //Алгоритм использует стек для хранения вершин, которые  
        нужно посетить.
```

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

```
        //Начальная вершина помещается в стек
```

```
        stack.Push(startVertexIndex);
```

```
        visited[startVertexIndex] = true;
```

```
        //а затем пока стек не пустой, извлекается текущая вершина  
        из стека.
```

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

```
        {
```

```
            int currentVertexIndex = stack.Pop();
```

```
            //Затем выводится информация о посещенной вершине (в  
            данном случае просто ее номер)
```

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

```
            //и для каждого соседа текущей вершины, который еще не  
            был посещен, он добавляется в стек и отмечается как посещенный.
```

```
            List<int> neighbors =  
            graph.GetNeighbors(currentVertexIndex);
```



```

        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.

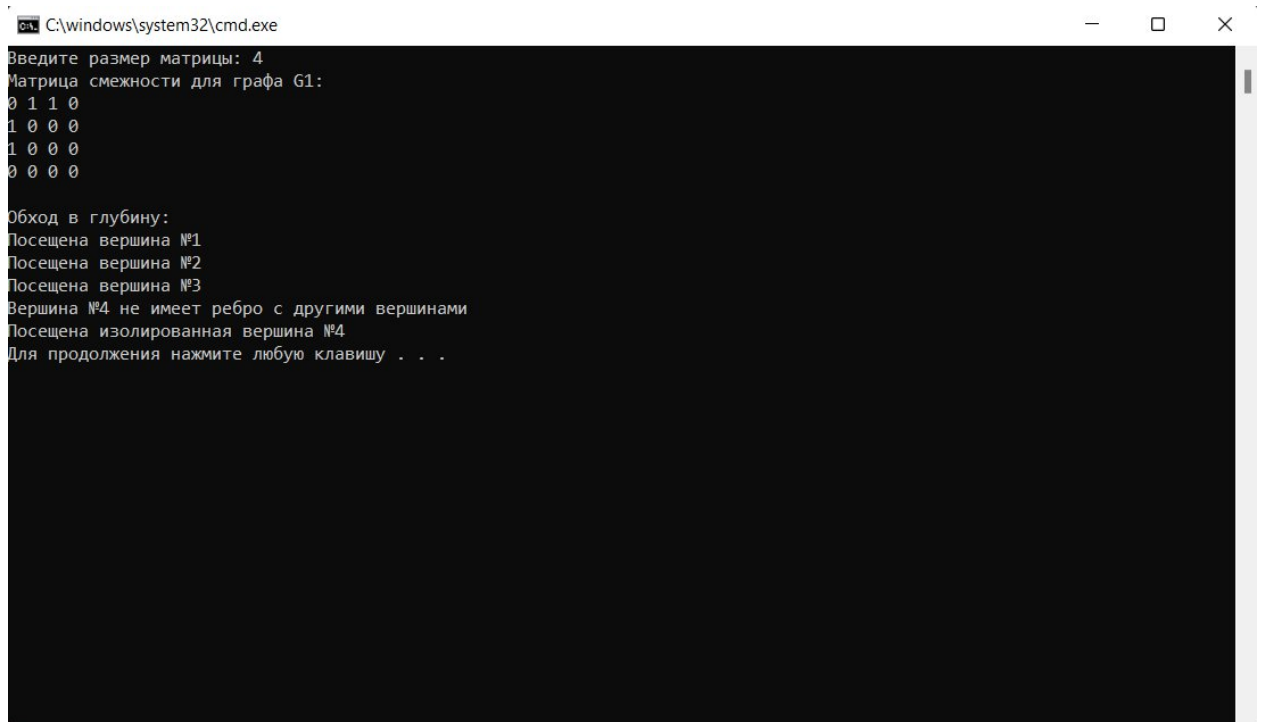
static bool IsIsolatedVertex(int vertex, int[,]
adjacencyMatrix)
{
    for (int i = 0; i < adjacencyMatrix.GetLength(1); i++)
    {
        if (adjacencyMatrix[vertex, i] == 1)
        {

```



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

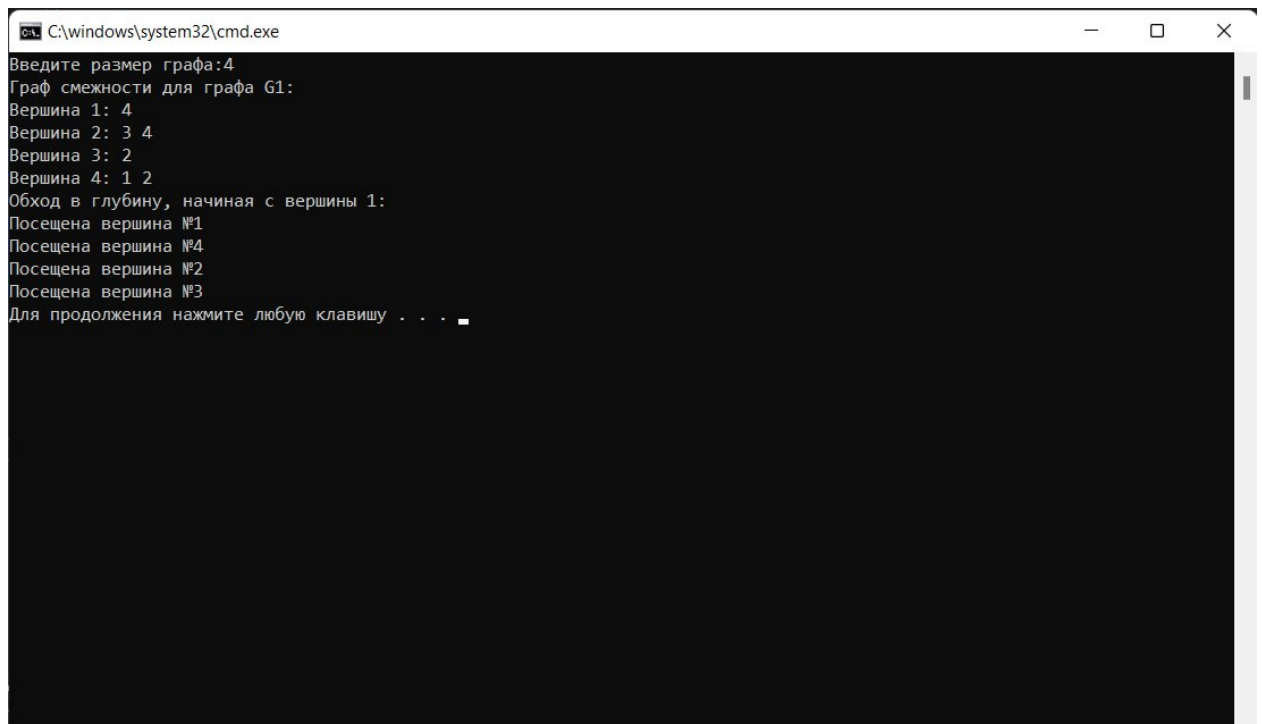
Результат работы программы 1.1-1.2:



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

Обход в глубину:
Посещена вершина №1
Посещена вершина №2
Посещена вершина №3
Вершина №4 не имеет ребро с другими вершинами
Посещена изолированная вершина №4
Для продолжения нажмите любую клавишу . . .
```

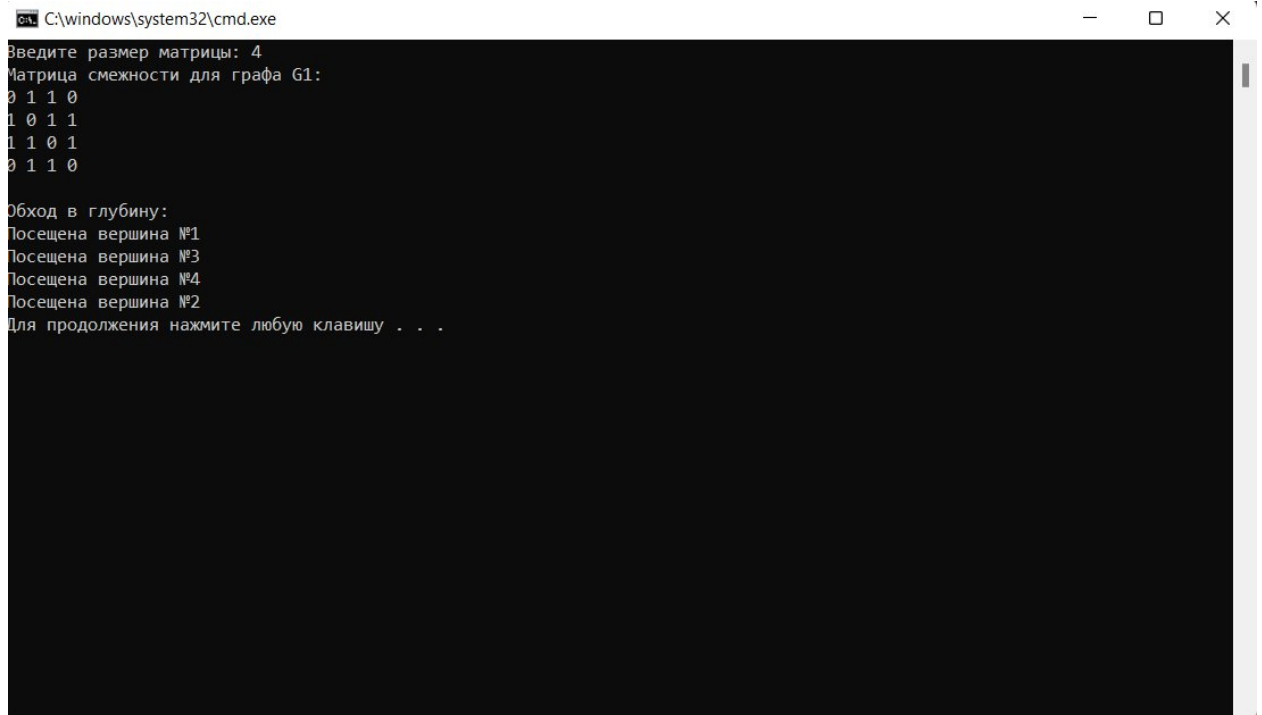
Результат работы программы 1.3:



```
C:\windows\system32\cmd.exe
Введите размер графа:4
Граф смежности для графа G1:
Вершина 1: 4
Вершина 2: 3 4
Вершина 3: 2
Вершина 4: 1 2

Обход в глубину, начиная с вершины 1:
Посещена вершина №1
Посещена вершина №4
Посещена вершина №2
Посещена вершина №3
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 2:



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

Обход в глубину:
Посещена вершина №1
Посещена вершина №3
Посещена вершина №4
Посещена вершина №2
Для продолжения нажмите любую клавишу . . .
```

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.