

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №5
«Определение характеристик графов»
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. Сгенерировать (используя генератор случайных чисел) матрицу смежности для неориентированного графа G . Вывести матрицу на экран.
2. Определить размер графа G , используя матрицу смежности графа.
3. Найти изолированные, концевые и доминирующие вершины.

По заданию 2*:

1. Построить для графа G матрицу инцидентности.
2. Определить размер графа G , используя матрицу инцидентности графа.
3. Найти изолированные, концевые и доминирующие вершины.

Ход работы:

Описание кода программы по заданию 1:

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

namespace example2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите количество вершин графа: ");
            int numNodes = Convert.ToInt16(Console.ReadLine());
            //Создает двумерный массив `adjacencyMatrix` с помощью метода
            `GenerateAdjacencyMatrix`, который принимает `numNodes` в качестве
            аргумента.
            int[,] adjacencyMatrix = GenerateAdjacencyMatrix(numNodes);

            Console.WriteLine("Матрица смежности: ");
            //Вызывает метод `PrintMatrix`, который принимает
            `adjacencyMatrix` в качестве аргумента и печатает его содержимое в
            консоль.
            PrintMatrix(adjacencyMatrix);

            //Получает количество вершин графа из размера массива
            `adjacencyMatrix`.
```

```

        numNodes = adjacencyMatrix.GetLength(0);
        //Создает одномерный массив `isolatedNodes` с помощью метода
`FindIsolatedNodes`, который принимает `adjacencyMatrix` в качестве
аргумента и возвращает изолированные вершины графа
        int[] isolatedNodes = FindIsolatedNodes(adjacencyMatrix);
        //Создает одномерный массив `terminalNodes` с помощью метода
`FindTerminalNodes`, который принимает `adjacencyMatrix` в качестве
аргумента и возвращает концевые вершины графа
        int[] leafyNodes = FindLeafyNodes(adjacencyMatrix);
        //Создает одномерный массив `dominatingNodes` с помощью метода
`FindDomingatingNodes`, который принимает `adjacencyMatrix` в качестве
аргумента и возвращает доминирующие вершины графа
        int[] dominatingNodes = FindDomingatingNodes(adjacencyMatrix);

        Console.WriteLine("Размер графа: " + numNodes);
        Console.WriteLine("Изолированные вершины: " + string.Join(",
", isolatedNodes));
        Console.WriteLine("Концевые вершины: " + string.Join(", ",
leafyNodes));
        Console.WriteLine("Доминирующие вершины: " + string.Join(",
", dominatingNodes));
    }

    //генерирует матрицу смежности для заданного количества узлов.
    //Каждый элемент матрицы может принимать значения 0 или 1, которые
случайным образом генерируются с помощью объекта класса Random.
    static int[,] GenerateAdjacencyMatrix(int numNodes)
    {
        Random rand = new Random();
        int[,] matrix = new int[numNodes, numNodes];

        for (int i = 0; i < numNodes; i++)
        {
            for (int j = i + 1; j < numNodes; j++)
            {
                int value = rand.Next(2);
                //Значение элемента[i, j] определяет наличие или
отсутствие ребра между узлами i и j
                matrix[i, j] = value;
                //Значение элемента [j, i] также устанавливается для
обеспечения симметрии матрицы
                matrix[j, i] = value;
            }
        }

        return matrix;
    }

    //Метод PrintMatrix печатает матрицу смежности в консоль.
    //Он проходит по каждому элементу матрицы и выводит его значение,
а также пробел.
    static void PrintMatrix(int[,] matrix)

```

```

    {
        int numNodes = matrix.GetLength(0);

        for (int i = 0; i < numNodes; i++)
        {
            for (int j = 0; j < numNodes; j++)
            {
                Console.Write(matrix[i, j] + " ");
            }
            Console.WriteLine();
        }
    }
    //Метод FindIsolatedNodes находит изолированные узлы в матрице
    смежности.
    //Метод перебирает каждый узел в матрице и проверяет, есть ли
    у него смежные узлы (ребра).
    //Если нет смежных узлов, то текущий узел считается изолированным
    и добавляется в список isolatedNodes.
    //Возвращается массив изолированных узлов.
    static int[] FindIsolatedNodes(int[,] matrix)
    {
        int numNodes = matrix.GetLength(0);
        List<int> isolatedNodes = new List<int>();

        for (int i = 0; i < numNodes; i++)
        {
            bool isolated = true;

            for (int j = 0; j < numNodes; j++)
            {
                if (matrix[i, j] == 1 || matrix[j, i] == 1)
                {
                    isolated = false;
                    break;
                }
            }

            if (isolated)
            {
                isolatedNodes.Add(i + 1);
            }
        }

        return isolatedNodes.ToArray();
    }

    //Метод FindTerminalNodes находит концевые узлы в матрице
    смежности.
    //Метод перебирает каждый узел в матрице и проверяет, если кол-
    во ребер больше 1, то ничего не выводит
    //Возвращается массив концевых узлов.

```

```

static int[] FindLeafyNodes(int[,] matrix)
{
    int numNodes = matrix.GetLength(0);
    List<int> leafyNodes = new List<int>();

    for (int i = 0; i < numNodes; i++)
    {
        bool terminal = true;
        int count = 0;

        for (int j = 0; j < numNodes; j++)
        {
            if (matrix[i, j] == 1 || matrix[j, i] == 1)
            {
                count++;
            }
            if (count == 1)
            {
                terminal = false;
            }
            else
            {
                terminal = true;
            }
        }

        if (!terminal)
        {
            leafyNodes.Add(i + 1);
        }
    }

    return leafyNodes.ToArray();
}

```

//Метод FindDominatingNodes находит доминирующие узлы в матрице смежности.

//Метод перебирает каждый узел в матрице и проверяет, есть ли у него ребра с каждым другим узлом, кроме самого себя.

//Если есть, то текущий узел считается доминирующим и добавляется в список dominatingNodes.

//Возвращается массив доминирующих узлов.

```

static int[] FindDominatingNodes(int[,] matrix)
{
    int numNodes = matrix.GetLength(0);
    List<int> dominatingNodes = new List<int>();

    for (int i = 0; i < numNodes; i++)
    {
        bool dominating = true;

```

```

        for (int j = 0; j < numNodes; j++)
        {
            if (i != j && matrix[i, j] == 0)
            {
                dominating = false;
                break;
            }
        }

        if (dominating)
        {
            dominatingNodes.Add(i + 1);
        }
    }

    return dominatingNodes.ToArray();
}

}
}

```

Описание кода программы по заданию 2*:

```

using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        Console.Write("Введите количество вершин графа: ");
        int numNodes = Convert.ToInt16(Console.ReadLine());
        Console.Write("Введите количество ребер графа: ");
        int numEdges = Convert.ToInt16(Console.ReadLine());
        int[,] incidenceMatrix = GenerateIncidenceMatrix(numNodes,
numEdges);

        //Вывод матрицы инцидентности
        PrintIncidenceMatrix(incidenceMatrix);

        int graphSize = GetGraphSize(incidenceMatrix);
        Console.WriteLine("Размер графа G: " + graphSize);

        //Вызывается метод FindIsolatedNodes, который находит
        //изолированные вершины графа, то есть вершины, не связанные ни с одним
        //ребром.
        //Результат сохраняется в списке isolatedNodes, который выводится
        //на консоль
        List<int> isolatedNodes = FindIsolatedNodes(incidenceMatrix);
        Console.WriteLine("Изолированные вершины: " + string.Join(", ",
        isolatedNodes));
    }
}

```

//Вызывается метод FindLeafNodes, который находит концевые вершины графа, то есть вершины, связанные только с одним ребром.
//Результат сохраняется в списке leafNodes, который выводится на консоль.

```
List<int> leafNodes = FindLeafNodes(incidenceMatrix);  
Console.WriteLine("Концевые вершины: " + string.Join(", ",  
leafNodes));
```

//Вызывается метод FindDominatingNodes, который находит доминирующие вершины графа, то есть вершины, которые связаны со всеми ребрами.

//Результат сохраняется в списке dominatingNodes, который выводится на консоль.

```
List<int> dominatingNodes = FindDominatingNodes(incidenceMatrix);  
Console.WriteLine("Доминирующие вершины: " + string.Join(", ",  
dominatingNodes));  
}
```

//Метод GenerateIncidenceMatrix генерирует случайную матрицу инцидентности для заданного количества вершин и ребер.

```
static int[,] GenerateIncidenceMatrix(int numNodes, int numEdges)  
{  
    //Он создает двумерный массив размером numNodes на numEdges и  
    //заполняет его случайными значениями 0 и 1.  
    Random rand = new Random();  
    int[,] matrix = new int[numNodes, numEdges];  
  
    for (int i = 0; i < numEdges; i++)  
    {  
        int node1 = rand.Next(numNodes);  
        int node2 = rand.Next(numNodes);  
  
        //Каждая строка матрицы соответствует вершине, а каждый столбец  
        //соответствует ребру.  
        //Если в ячейке матрицы стоит 1, это означает, что вершина  
        //связана с соответствующим ребром.  
        matrix[node1, i] = 1;  
        matrix[node2, i] = 1;  
    }  
  
    return matrix;  
}
```

//Метод PrintIncidenceMatrix выводит матрицу инцидентности на консоль.

//Он получает размеры матрицы из ее параметров и использует два вложенных цикла для печати значений каждой ячейки.

```
static void PrintIncidenceMatrix(int[,] matrix)  
{
```

```

        int numNodes = matrix.GetLength(0);
        int numEdges = matrix.GetLength(1);

        Console.WriteLine("Матрица инцидентности:");

        for (int i = 0; i < numNodes; i++)
        {
            for (int j = 0; j < numEdges; j++)
            {
                Console.Write(matrix[i, j] + " ");
            }

            Console.WriteLine();
        }
    }
}

```

//Метод GetGraphSize вычисляет количество ребер в графе, представленном матрицей инцидентности.

//Он проходит по всем ячейкам матрицы и увеличивает счетчик, если в ячейке стоит 1.

```

static int GetGraphSize(int[,] matrix)
{
    int numNodes = matrix.GetLength(0);
    int numEdges = matrix.GetLength(1);

    int graphSize = 0;

    for (int i = 0; i < numNodes; i++)
    {
        for (int j = 0; j < numEdges; j++)
        {
            if (matrix[i, j] == 1)
            {
                graphSize++;
                break;
            }
        }
    }

    return graphSize;
}

```

//Метод FindIsolatedNodes находит изолированные вершины в графе. Он проходит по всем вершинам и проверяет, есть ли связанные с ними ребра.

//Если ни одно ребро не связано с вершиной, она считается изолированной и добавляется в список изолированных вершин.

```

static List<int> FindIsolatedNodes(int[,] matrix)
{
    List<int> isolatedNodes = new List<int>();
    int numNodes = matrix.GetLength(0);
    int numEdges = matrix.GetLength(1);
}

```



```

for (int i = 0; i < numNodes; i++)
{
    bool isolated = true;

    for (int j = 0; j < numEdges; j++)
    {
        if (matrix[i, j] == 1)
        {
            isolated = false;
            break;
        }
    }

    if (isolated)
    {
        isolatedNodes.Add(i + 1);
    }
}

```

```

return isolatedNodes;

```

```

}

```

//Метод FindLeafNodes находит концевые вершины в графе. Он проходит по всем вершинам и подсчитывает количество связанных с ними ребер.

//Если количество ребер равно 1, вершина считается концевой и добавляется в список концевых вершин.

```

static List<int> FindLeafNodes(int[,] matrix)

```

```

{
    List<int> leafNodes = new List<int>();
    int numNodes = matrix.GetLength(0);
    int numEdges = matrix.GetLength(1);

```

```

for (int i = 0; i < numNodes; i++)
{
    bool isLeafNode = true;
    int count = 0;

    for (int j = 0; j < numEdges; j++)
    {
        if (matrix[i, j] == 1)
        {
            count++;
        }
        if (count == 1)
        {
            isLeafNode = false;
        }
        else
        {
            isLeafNode = true;

```

```

        }
    }

    if (!isLeafNode)
    {
        leafNodes.Add(i + 1);
    }
}

return leafNodes;
}

//Метод FindDominatingNodes находит доминирующие вершины в графе.
Он проходит по всем вершинам и проверяет, связаны ли все ребра с данной
вершиной.
//Если все ребра связаны с вершиной, она считается доминирующей и
добавляется в список доминирующих вершин.
static List<int> FindDominatingNodes(int[,] matrix)
{
    List<int> dominatingNodes = new List<int>();
    int numNodes = matrix.GetLength(0);
    int numEdges = matrix.GetLength(1);

    for (int i = 0; i < numNodes; i++)
    {
        bool isDominatingNode = true;

        for (int j = 0; j < numEdges; j++)
        {
            if (matrix[i, j] == 0)
            {
                isDominatingNode = false;
                break;
            }
        }
        if (isDominatingNode)
        {
            dominatingNodes.Add(i+1);
        }
    }

    return dominatingNodes;
}
}

```

Результат работы программы 1:

```
C:\windows\system32\cmd.exe
Введите количество вершин графа: 5
Матрица смежности:
0 0 0 1 0
0 0 0 0 0
0 0 0 0 0
1 0 0 0 1
0 0 0 1 0
Размер графа: 5
Изолированные вершины: 2, 3
Концевые вершины: 1, 5
Доминирующие вершины:
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы 2*:

```
C:\windows\system32\cmd.exe
Введите количество вершин графа: 3
Введите количество ребер графа: 4
Матрица инцидентности:
1 1 1 1
0 1 1 0
1 0 0 1
Размер графа G: 3
Изолированные вершины:
Концевые вершины:
Доминирующие вершины: 1
Для продолжения нажмите любую клавишу . . .
```

Вывод: в данной лабораторной работе мы научились работать с графами, использовать матрицы смежности и инцидентности, а также находить изолированные, концевые и доминирующие вершины.