**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ ИНСТИТУТ**»

**ИНСТИТУТ ГЕОЛОГИИ И НЕФТЕГАЗОДОБЫЧИ**

Кафедра Кибернетических систем

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Математические основы программирования»

на тему «Потоки в сетях и родственные задачи»

**РУКОВОДИТЕЛЬ:**

*преподаватель*

*Гапанович И.В*

**РАЗРАБОТЧИК:**

*студент группы АСОиУ-21-1*

*Астахов С.О*

*Быльцев С.А*

Тюмень, 2022г.

**Оглавление**

[Постановка задачи 2](#_Toc115318918)

[Описание алгоритма решения задачи 3](#_Toc115318919)

[Ручной просчет 4](#_Toc115318920)

[Описание программы 6](#_Toc115318921)

[Тестирования программы 7](#_Toc115318922)

[*Приложение 1 – код программы.* 8](#_Toc115318923)

# **Постановка задачи**

Решить задачу нахождения наибольшего паросочетания в  
двудольном графе, сведя ее к задаче построения максимального потока в  
транспортной сети и используя первый алгоритм построения максимального  
потока.

Паросочетание - это набор попарно несмежных рёбер.

Граф называется двудольным, если существует разбиение множества его вершин на две части такое, что концы каждого ребра принадлежат разным долям.

Поток ϕ называется максимальным, если его величина принимает максимальное значение по сравнению с другими допустимыми потоками в  
транспортной сети D. Максимальный поток всегда является полным.

К множеству долей графа добавим 2 вершины, которые являются: стоком и истоком. Соединяем исток, со всеми вершинами первой доли, а сток соединяем со всеми вершинами второй доли. Таким образом, из двудольного графа мы получаем транспортную сеть.

Для заданной транспортной сети D и допустимого потока ϕ в  
этой сети орграф приращений I(D,ϕ), имеющий те же вершины, что и сеть D.  
Каждой дуге x = (v,w) ∈ X транспортной сети D в орграфе приращений I(D,  
ϕ) соответствуют две дуги: x и x\* = (w,v) – дуга, противоположная по  
направлению дуге x.

0, если ϕ(x) < c(x),

l(x) =

∞, если ϕ(x) = c(x),

0, если ϕ(x) > 0,

l(x\*) =

∞, если ϕ(x) = 0.

Поиск максимального потока через орграф приращения напрямую связан с максимальным паросочетанием, ведь поток через вершины долей показывает наличие паросочетания(есть поток через ребро равен 1, то данное ребро входит в множество, содержащие максимальное число паросочетаний, если поток равен 0, то ребро не входит).

# **Описание алгоритма решения задачи**

**Алгоритм построения максимального потока:**

Данные: транспортная сеть D, заданная матрицей пропускных способностей дуг.

Результат: максимальный поток в сети.

1. Полагаем, что в поток (ϕ) в сети изначально равен 0, а максимальный пропускная способность равна 1.
2. По сети D и потоку ϕстроим орграф приращений I(D, ϕ).
3. Находим простую цепь μi, являющуюся минимальным путем из v1 вvn в нагруженном орграфе I(D, ϕ). Если длина этой цепи равна ∞, то поток ϕ максимален, и работа алгоритма заканчивается. В противном случае увеличиваем поток вдоль цепи μ на максимально допустимую величину ai > 0, где ai∈Z (прибавляя ее для каждой дуги x∈X, через которую проходит цепь μi, к уже имеющейся величине потока по дуге x, если дуга x входит в μi, и вычитая, если дуга x\* входит в μi), такую, что при этом сохраняется условие допустимого потока. В результате меняется поток в транспортной сети D, т.е. от потока ϕпереходим к потоку ϕ+1, который является допустимым, и при этом величина его увеличивается наai. Переходим к шагу 2.

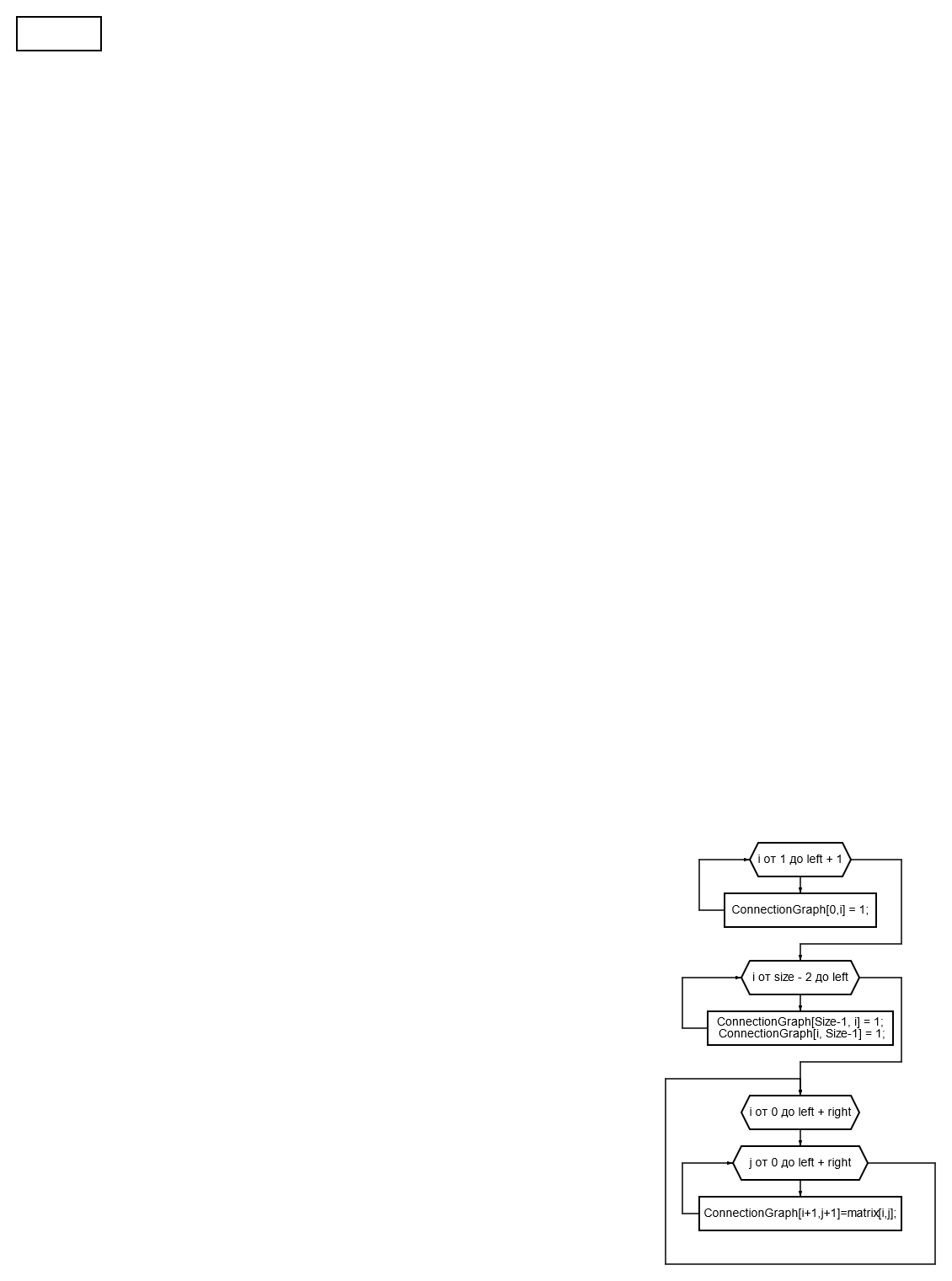


Рисунок 1 – Блок – схема проверки соединения вершин

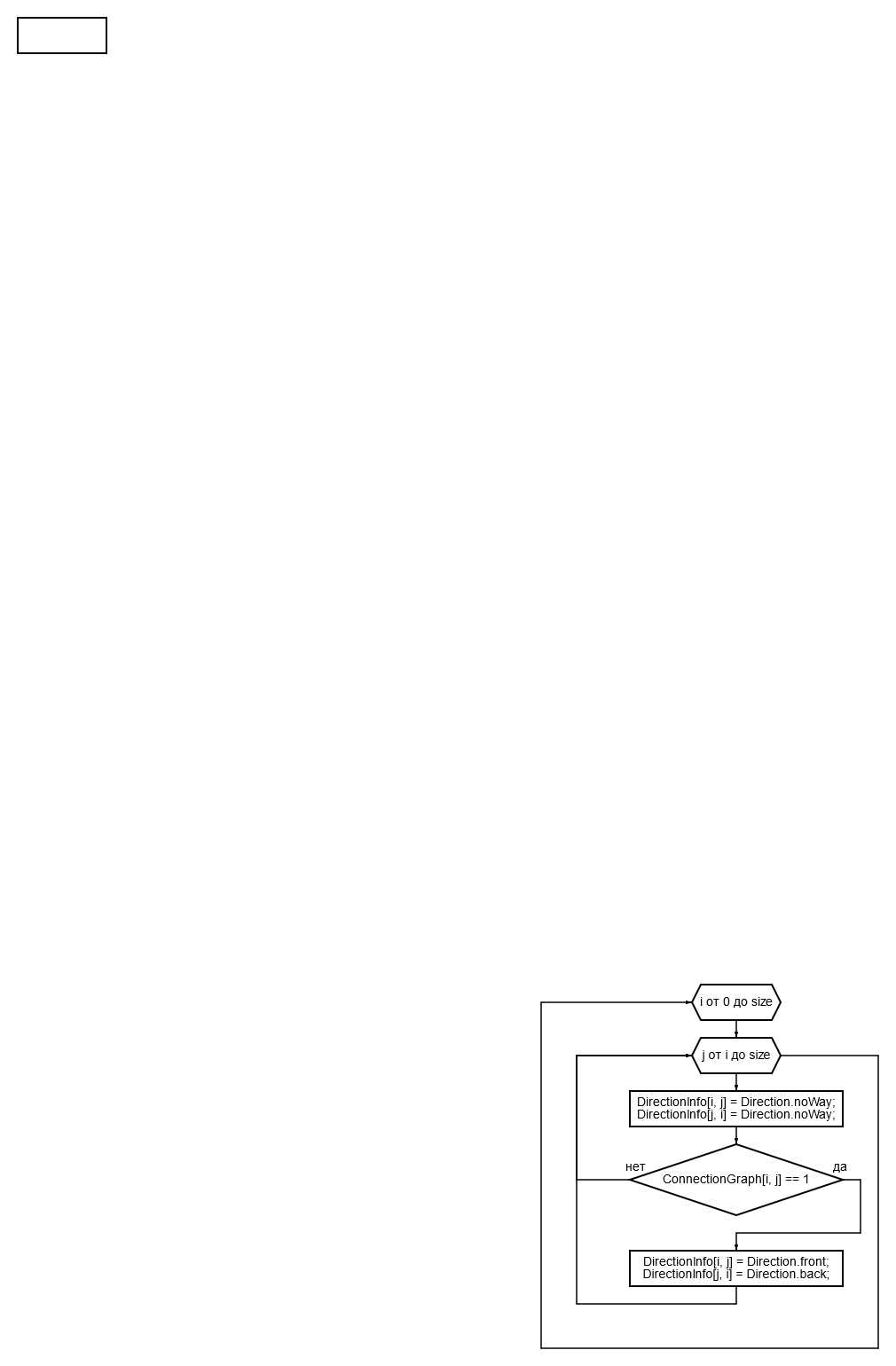


Рисунок 2 – Блок – схема проверки пути из вершины

# **Ручной просчет**

Пусть кол-во вершин первой доли – 2

Пусть кол-во вершин второй доли – 3

Номер первой вершины – 1

Номер второй вершины – 3

Номер первой вершины – 2

Номер второй вершины – 4

Номер первой вершины – 2

Номер второй вершины – 3

Задаем матрицу смежности:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| **1** | 0 | 0 | 1 | 1 |
| **2** | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 |

Строим транспортную сеть с источником S и стоком T след. образом:

* Источник S соединяем с вершинами первой доли;
* Вершины второй доли соединяем со стоком T;
* Пропускная способность всех дуг = 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | 1 | 1 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 | 1 | 0 |
| **2** | 0 | 0 | 0 | 1 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 1 |
| **4** | 0 | 0 | 0 | 0 | 0 | 1 |
| **5** | 0 | 0 | 0 | 1 | 1 | 0 |

Строим орграф приращения:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **0** | -1 | 0 | 0 | -1 | -1 | -1 | -1 |
| **1** | 1 | -1 | -1 | 0 | -1 | -1 | -1 |
| **2** | 1 | -1 | -1 | 0 | 0 | -1 | -1 |
| **3** | -1 | 1 | 1 | -1 | -1 | -1 | 0 |
| **4** | -1 | -1 | 1 | -1 | -1 | -1 | 0 |
| **5** | -1 | -1 | -1 | -1 | -1 | -1 | 0 |
| **6** | -1 | -1 | -1 | 1 | 1 | 1 | -1 |

С помощью орграфа приращения ищем минимальный путь из S в T. Если длина минимального пути равна нулю, то изменяем поток цепи на +- единицу в зависимости от направления дуги. Продолжаем искать минимальные пути, пока мин путь не будет равен бесконечности. После завершение алгоритма поиска, начинаем перебор нагруженных ребер. Если ребро между долями имеет вел равный единице, то добавляем его к множеству образующих максимальное число паросочетаний.

# **Описание программы**

Данная программа, предназначенная для нахождения наибольшего паросочетания в двудольном графе. Программа написана на языке C# с использованием компилятора CIL.

Классы: Algorithm, Matrix, Graph, Program.

Функции: SetMatrixEdges, SetConnectionGraph, SetWeightGraph, SetDirectionInfo, SetIncrementGraph, ChangeWeightGraph

В программе используются динамические массивы, классы и функции. Ответ представлен в виде номеров вершин, образующих ребро (паросочетание).

# **Тестирования программы**

Укажите кол-во вершин первой доли -> 2

Укажите кол-во вершин второй доли -> 3

Укажите ребра соединяющие вершины каждой доли, чтобы выйди из редактирования введите "-1"

Номер первой вершины -> 1

Номер второй вершины -> 3

Номер первой вершины -> 2

Номер второй вершины -> 3

Номер первой вершины -> 2

Номер второй вершины -> 4

Номер первой вершины -> -1

0 0 1 0 0

0 0 1 1 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

Ответ представлен в виде номеров вершин, образующих ребро (паросочетание)

Паросочетания:

1 3

2 4

# *Приложение 1 – код программы.*

**Class Program**

using System;

namespace GraphMop

{

class Program

{

static void Main()

{

Console.Write("Укажите кол-во вершин первой доли -> ");

var leftPart = int.Parse(Console.ReadLine());

Console.Write("Укажите кол-во вершин второй доли -> ");

var rightPart = int.Parse(Console.ReadLine());

var matrix = new Matrix(leftPart + rightPart);

Console.WriteLine("Укажите ребра соединяющие вершины каждой доли, чтобы выйди из редактирование введите \"-1\"");

matrix.SetMatrixEdges();

Console.WriteLine();

var graph = new Graph(leftPart, rightPart);

graph.SetConnectionGraph(matrix.AdjacencyMatrix);

graph.SetWeightGraph();

graph.SetDirectionInfo();

graph.SetIncrementGraph();// либо прошлое работает правильно а это нет

//Console.WriteLine();

//matrix.PrintAdjacencyMatrix(); матрица смежности 2 долей

//Console.WriteLine();

//Graph.PringGraph(graph.ConnectionGraph); матрица смежности графа

//Console.WriteLine();

//Graph.PringGraph(graph.WeightedGraph); матрица нагруженного графа

//Console.WriteLine();

//Graph.PringGraph(graph.IncrementGraph); матрица орграфа приращения

//Console.WriteLine();

var alg = new Algorithm();

do

{

alg.ClearLists();

alg.MinWay(graph.IncrementGraph, 0);

if (alg.Roads.Count == 0)

break;

graph.ChangeWeightGraph(alg.ChooseBestWay());

graph.SetIncrementGraph();

}

while (alg.Roads.Count > 0);

graph.PringAnswer();

}

}

}

**Class Graph.**

using System;

using System.Collections.Generic;

namespace GraphMop

{

class Graph

{

public readonly int Left;

public readonly int Right;

public readonly int Size;

public int[,] ConnectionGraph; //матрица связности

public Reach[,] WeightedGraph; // нагруженный граф

public Weight[,] IncrementGraph; // граф приращения

public Direction[,] DirectionInfo; // показывает направление дуги

public enum Reach

{

zero=0,

one=1,

noWay=-1

};

public enum Direction

{

noWay=-1,

back = 0,

front =1,

};

public enum Weight

{

inf=1,

zero=0,

noWay=-1

};

public Graph(int left, int right)

{

Size = left+right+2;

Left=left;

Right=right;

ConnectionGraph = new int[Size, Size];

DirectionInfo = new Direction[Size, Size];

WeightedGraph = new Reach[Size, Size];

IncrementGraph = new Weight[Size, Size];

}

public static void PringGraph(int[,] graph)

{

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

{

for (int j = 0; j < graph.GetLength(0); j++)

{

Console.Write(graph[i, j] + "\t");

}

Console.WriteLine();

}

}

public static void PringGraph(Direction[,] graph)

{

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

{

for (int j = 0; j < graph.GetLength(0); j++)

{

Console.Write(graph[i, j] + "\t");

}

Console.WriteLine();

}

}

public static void PringGraph(Reach[,] graph)

{

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

{

for (int j = 0; j < graph.GetLength(0); j++)

{

Console.Write(graph[i, j] + "\t");

}

Console.WriteLine();

}

}

public static void PringGraph(Weight[,] graph)

{

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

{

for (int j = 0; j < graph.GetLength(0); j++)

{

Console.Write((int)graph[i, j] + "\t");

}

Console.WriteLine();

}

}

public void SetConnectionGraph(int[,] matrix)

{

for (int i = 1; i < Left+1 ; i++)

{

ConnectionGraph[0,i] = 1;

}

for (int i = Size-2; i > Left ; i--)

{

ConnectionGraph[Size-1, i] = 1;

ConnectionGraph[i, Size-1] = 1;

}

for (int i = 0; i < Left+Right; i++)

{

for (int j = 0; j < Left + Right; j++)

{

ConnectionGraph[i+1,j+1]=matrix[i,j];

}

}

}

public void SetWeightGraph()

{

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

for (int j = 0; j < Size; j++)

WeightedGraph[i, j] = Reach.noWay;

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

{

for (int j = 0; j < Size; j++)

{

if (ConnectionGraph[i, j] == 1)

{

WeightedGraph[i, j] = Reach.zero;

WeightedGraph[j, i] = Reach.zero;

}

}

}

}

public void SetDirectionInfo()

{

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

{

for (int j = i; j < Size; j++)

{

DirectionInfo[i, j] = Direction.noWay;

DirectionInfo[j, i] = Direction.noWay;

if (ConnectionGraph[i, j] == 1)

{

DirectionInfo[i, j] = Direction.front;

DirectionInfo[j, i] = Direction.back;

}

}

}

}

public void SetIncrementGraph()

{

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

{

for (int j = i; j < Size; j++)

{

if (WeightedGraph[i, j] == Reach.one)

{

IncrementGraph[i, j] = Weight.inf;

IncrementGraph[j, i] = Weight.zero;

}

else if (WeightedGraph[i, j] == Reach.zero)

{

IncrementGraph[i, j] = Weight.zero;

IncrementGraph[j, i] = Weight.inf;

}

else

{

IncrementGraph[i, j] = Weight.noWay;

IncrementGraph[j, i] = Weight.noWay;

}

}

}

}

public void ChangeWeightGraph(List <int> way)

{

for (int i = 0; i < way.Count-1; i++)

{

if (DirectionInfo[way[i], way[i + 1]] == Direction.front)

{

WeightedGraph[way[i], way[i + 1]] = Reach.one;

WeightedGraph[way[i + 1], way[i]] = Reach.one;

}

else

{

WeightedGraph[way[i], way[i + 1]] = Reach.zero;

WeightedGraph[way[i + 1], way[i]] = Reach.zero;

}

}

}

public void PringAnswer()

{

Console.WriteLine("Ответ представлен в виде номеров вершин образующих ребро(паросочетание)");

Console.WriteLine("Паросочетания:");

for (int i = 1; i < Left + 1; i++)

{

for (int j = Left + 1; j < WeightedGraph.GetLength(0) - 1; j++)

{

if (ConnectionGraph[i, j] == 1 && WeightedGraph[i, j] == Graph.Reach.one)

Console.WriteLine(i + " " + j);

}

}

}

}

}

**Class Matrix.**

using System;

namespace GraphMop

{

class Matrix

{

public readonly int Size;

public readonly int[,] AdjacencyMatrix; // матрица смежности для двух долей

public Matrix(int size)

{

Size = size;

AdjacencyMatrix = new int[size, size];

}

public void SetMatrixEdges()

{

while (true)

{

Console.Write("Номер первой вершины -> ");

int FirstPoint = int.Parse(Console.ReadLine());

if (FirstPoint == -1)

break;

Console.Write("Номер второй вершины -> ");

int SecondPoint = int.Parse(Console.ReadLine());

if (FirstPoint != SecondPoint)

AdjacencyMatrix[FirstPoint - 1, SecondPoint - 1] = 1;

}

}

public void PrintAdjacencyMatrix()

{

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

{

for (int j = 0; j < Size; j++)

{

Console.Write(AdjacencyMatrix[i, j] + " ");

}

Console.WriteLine();

}

}

}

}

**Class Algorithm.**

using System;

using System.Collections.Generic;

using System.Linq;

namespace GraphMop

{

class Algorithm

{

public List<int> CelectedPeaks;

public List<List<int>> Roads;

public Algorithm()

{

CelectedPeaks = new List<int>();

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

}

public void MinWay(Graph.Weight[,] graph, int index)

{

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

{

if (index == graph.GetLength(0) - 1)

{

Roads.Add(CelectedPeaks.GetRange(0,CelectedPeaks.Count).ToList());

CelectedPeaks.RemoveAt(CelectedPeaks.Count - 1);

break;

}

if (graph[index, i] == Graph.Weight.zero && i != 0)

{

CelectedPeaks.Add(i);

MinWay(graph, i);

}

if (i == graph.GetLength(0) - 1 && CelectedPeaks.Count>0)

CelectedPeaks.RemoveAt(CelectedPeaks.Count - 1);

}

}

public List<int> ChooseBestWay()

{

var result = Roads[0];

foreach (var item in Roads)

{

if (item.Count < result.Count)

result=item;

}

return result;

}

public void ClearLists()

{

Roads.Clear();

CelectedPeaks.Clear();

CelectedPeaks.Add(0);

}

}

}