Московский авиационный институт

(Национальный исследовательский университет)

Факультет №3

Кафедра 304

Отчёт по лабораторной работе

По учебной дисциплине

Структуры и алгоритмы обработки данных

На тему:  
“Работа с графами”

Выполнил:

Плотников Кирилл

Крещук Николай

Группа: М30-224БК-17

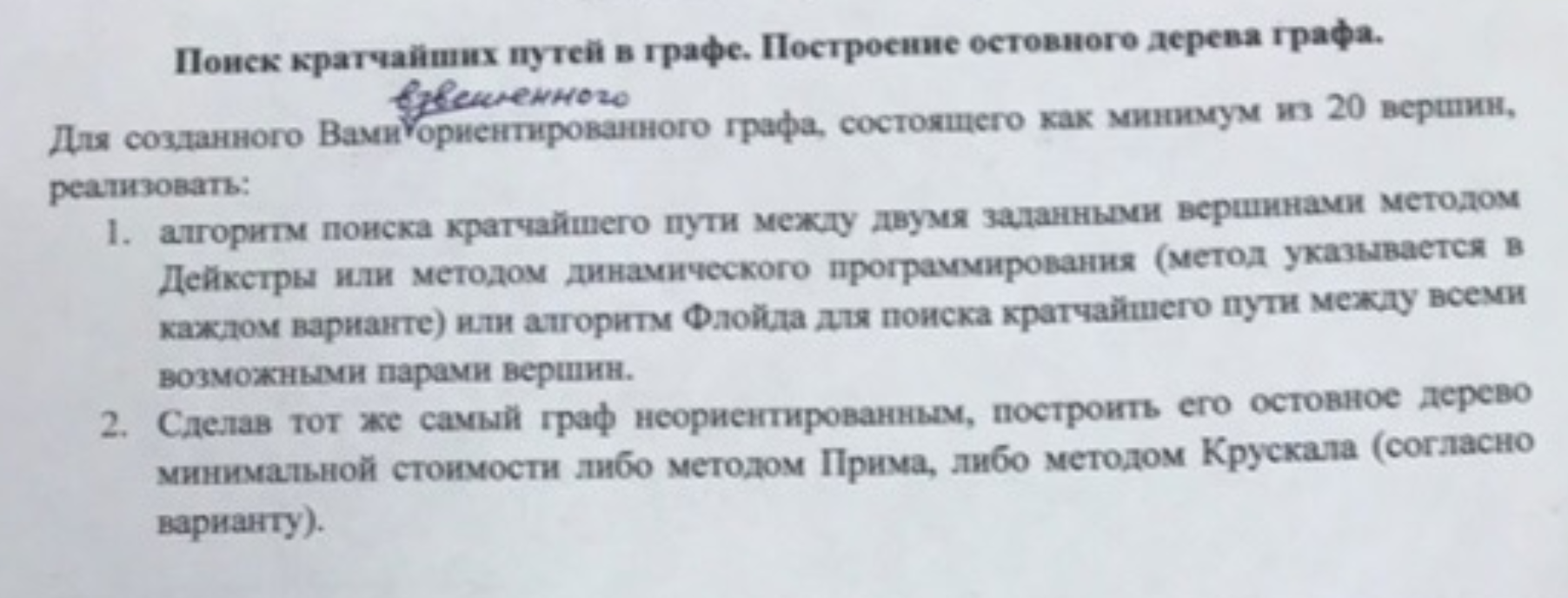
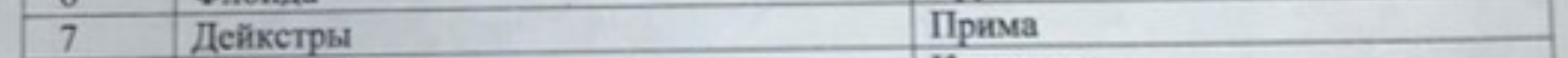
Бригада №7

Принял:

Силаев А.В.

Москва, 2018

**Задание.**

**** 

**Класс Задания**

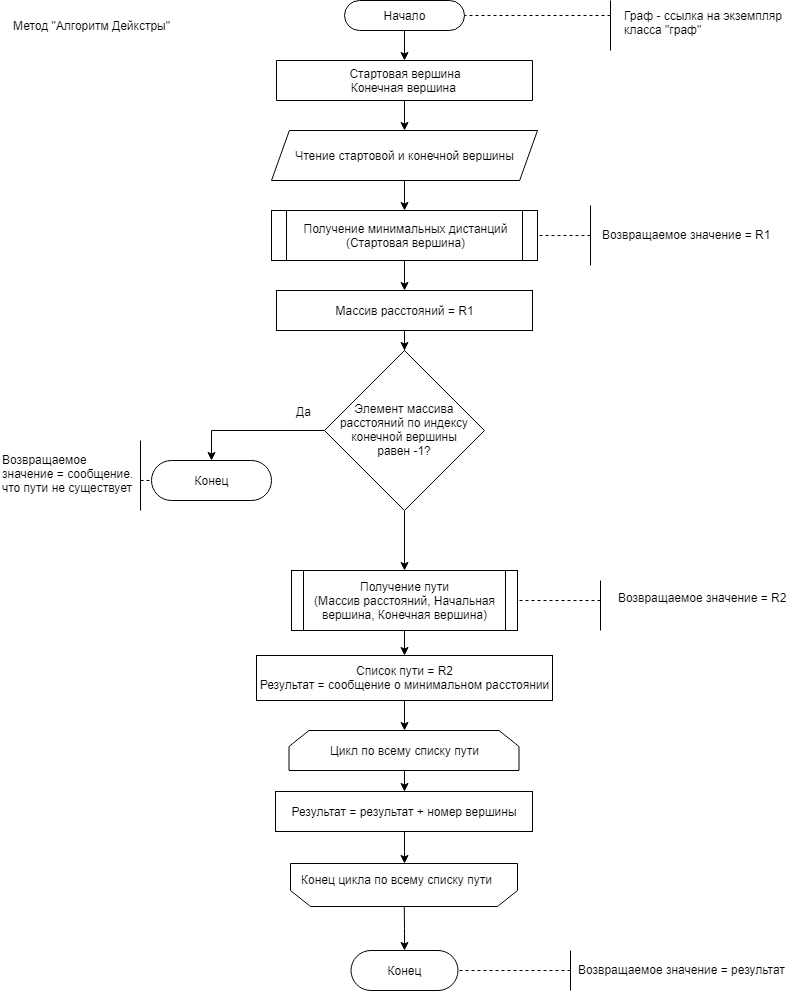
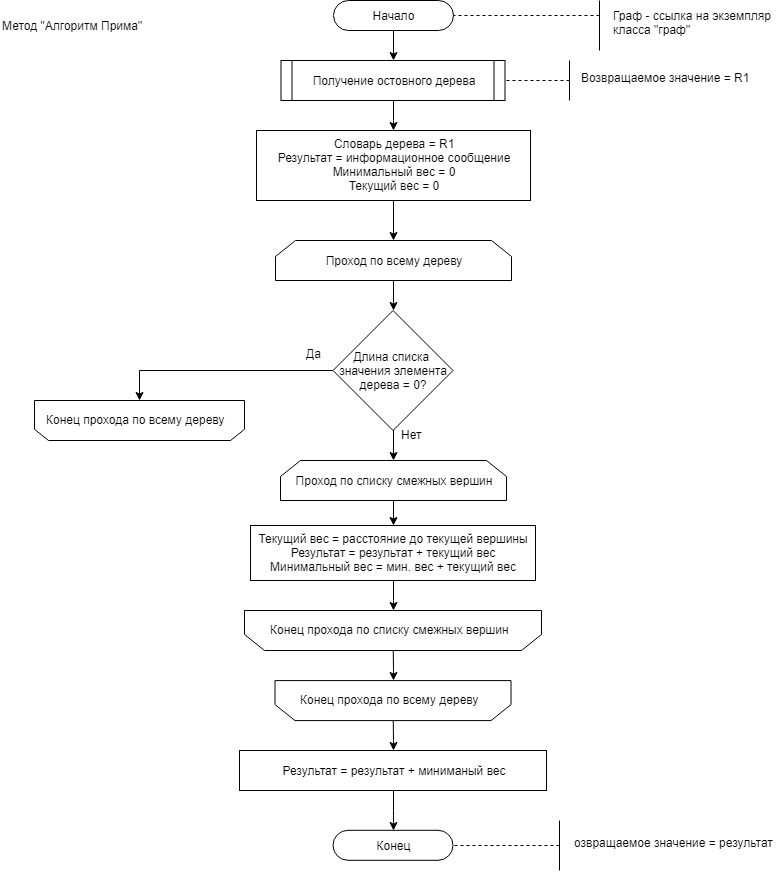
**Имя:** Topic.

**Поля:**

1. Количество вершин (целочисленное значение).
2. Матрица смежности (целочисленный массив).

**Методы:**

1. Конструктор – инициализация полей данных.
2. Чтение матрицы из файла – считывает матрицы из файла и записывает данные в поле «матрица смежности».
3. Инициализация графа – возвращает объект «граф», созданный по подобии матрицы смежности.
4. Получение неориентированного графа – возвращает объект «граф», в котором все рёбра являются неориентированными.
5. Алгоритм Дейкстры – выполняет алгоритм Дейкстры и возвращает полученный результат.
6. Алгоритм Прима – выполняет алгорити Прима и возвращает полученный результат.



**Класс Графа.**

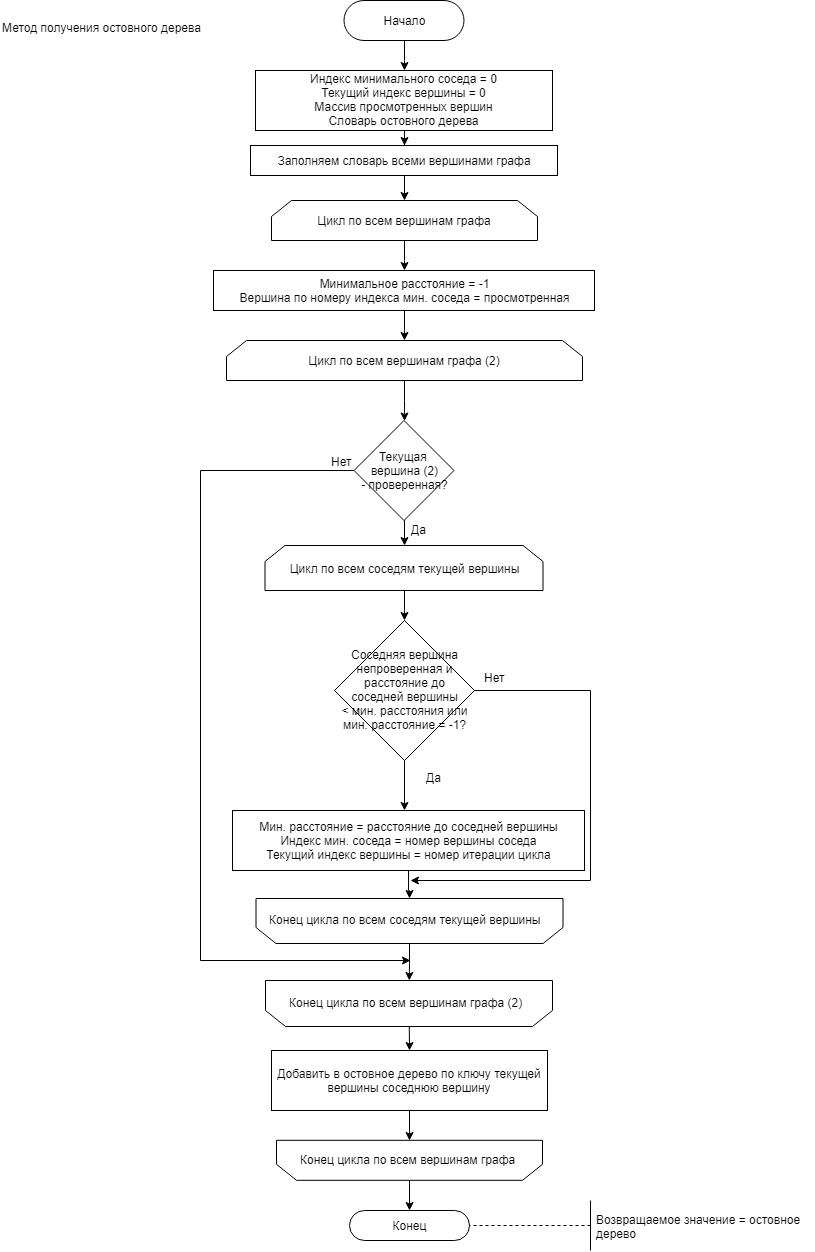
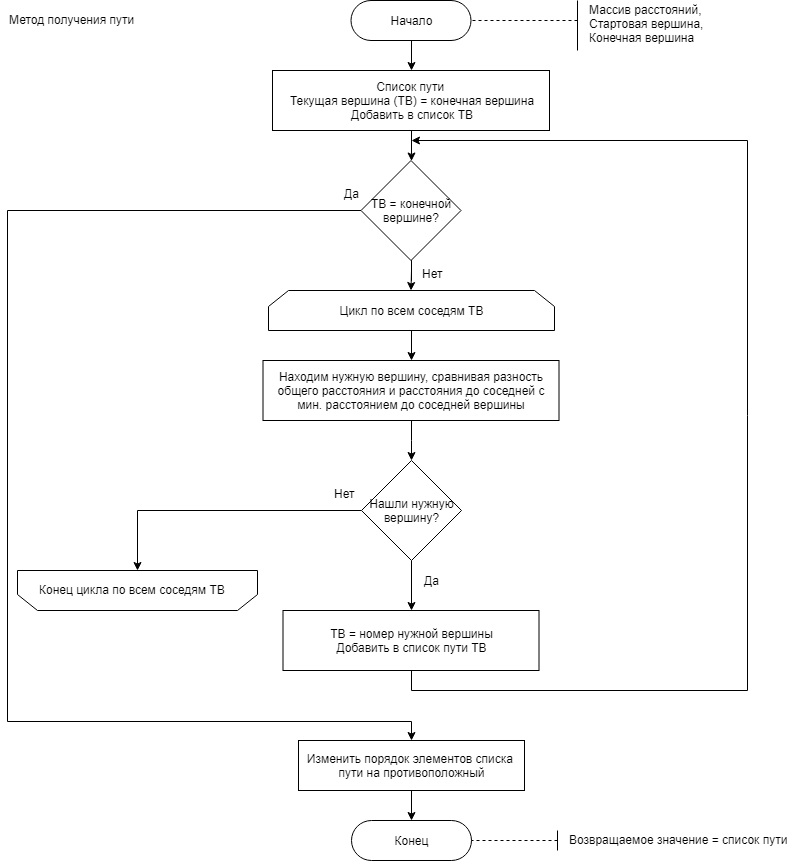
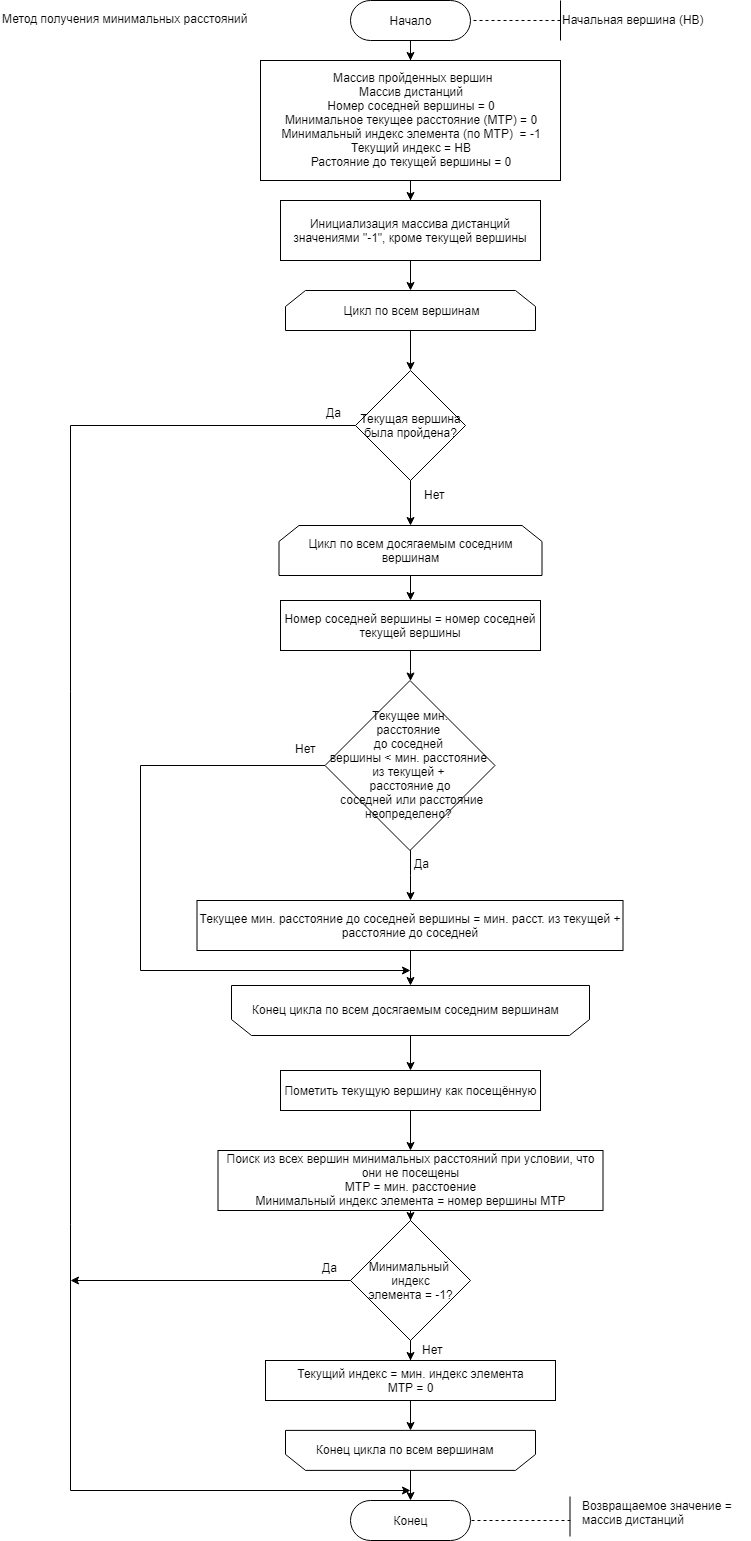
**Имя:** Graph.

**Поля:**

1. Словарь вершин (ключ – номер вершины, значение – экземпляр класса вершина).

**Методы:**

1. Конструктор – инициализация полей данных.
2. Добавление вершины – добавляет вершину в словарь.
3. Получение вершины – получение вершины по индексу вершины.
4. Получение минимальной дистанции – возвращает целочисленный массив минимальных дистанций до всех остальных.
5. Получение пути – возвращает список вершин, в которые входит минимальный путь.
6. Получение минимального остовного дерева – возврщает словарь остовного дерева.



**Класс вершины.**

**Имя:** Node.

**Поля:**

1. Номер вершины (целочисленное).
2. Словарь соседей (ключ – экземпляр вершины, значение – номер вершины).

**Методы:**

1. Конструктор – инициализация полей данных.

**Класс основной программы.**

**Имя:** Program.

**Методы:**

1. Main – создаёт экземпляр класса задания, с помощью него читает данные из файла.

С помощью экземпляра получаем объект «граф» и выполняем алгоритм Дейкстры, затем преобразуем граф в неориентированный и выполняем алгоритм Прима.

**Код программы.**

class Program

{

static void Main(string[] args)

{

// Количество вершин графа.

int n = 4;

string fileName = @"D:\GitHub\Labs\_Struct\_Prog\Labs\_Structures\_And\_Algorithms\Lab\_4\Test.txt";

Topic t = new Topic(n);

t.ReadMatrixFromFile(fileName);

Graph graph = t.InitializeGraph();

Console.WriteLine(t.DoDijkstra(graph));

Console.WriteLine();

Graph nonOrientedGraph = t.GetNonOrientedGraph();

Console.WriteLine(t.DoPrima(nonOrientedGraph));

Console.ReadLine();

}

}

class Node

{

/// <summary>

/// Номер вершины.

/// </summary>

public int Number { get; }

/// <summary>

/// Соседние вершины и расстояние до них.

/// </summary>

public Dictionary<Node, int> Neighbourhood { get; set; }

/// <summary>

/// Конструктор класса вершины.

/// </summary>

/// <param name="numb">Номер вершины.</param>

public Node(int numb)

{

Number = numb;

Neighbourhood = new Dictionary<Node, int>();

}

}

class Graph

{

private Dictionary<int, Node> \_nodes { get; set; }

public Graph()

{

\_nodes = new Dictionary<int, Node>();

}

public void AddNode(int number)

{

if (!\_nodes.ContainsKey(number))

\_nodes.Add(number, new Node(number));

}

public Node GetNode(int number)

{

return \_nodes[number];

}

public int[] GetMinDistancesToOther(int start)

{

bool[] isCheck = new bool[\_nodes.Count];

int[] distances = new int[\_nodes.Count];

int NumberNeighbour = 0;

int minCurDistance = 0;

int minIndexElem = -1;

int iCur = start;

distances[iCur] = 0;

for (int i = 0; i < \_nodes.Count; i++)

{

if (i == iCur)

continue;

// -1 означает, что путь из начальной вершины не найден/не существует.

distances[i] = -1;

}

for (int i = 0; i < \_nodes.Count; i++)

{

if (isCheck[iCur])

break;

var reachableNeighbors = \_nodes[iCur].Neighbourhood.Where((n) => n.Value != -1).Select((n) => n);

foreach (var neighb in reachableNeighbors)

{

NumberNeighbour = neighb.Key.Number;

if (distances[NumberNeighbour] > distances[iCur] + neighb.Value || distances[NumberNeighbour] == -1)

distances[NumberNeighbour] = distances[iCur] + neighb.Value;

}

isCheck[iCur] = true;

for (int j = 0; j < \_nodes.Count; j++)

{

if (distances[j] != -1 && (distances[j] < minCurDistance || minCurDistance == 0) && !isCheck[j])

{

minCurDistance = distances[j];

minIndexElem = j;

}

}

if (minIndexElem == -1)

break;

iCur = minIndexElem;

minCurDistance = 0;

}

return distances;

}

public List<int> GetWay(int[] distances, int start, int finish)

{

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

int iCur = finish;

ascendingWay.Add(iCur);

while (iCur != start)

{

foreach (KeyValuePair<Node, int> node in \_nodes[iCur].Neighbourhood)

{

Node neighbourNode = node.Key;

Node curNode = GetNode(iCur);

if (neighbourNode.Neighbourhood.ContainsKey(curNode) && neighbourNode.Neighbourhood[curNode] != -1 &&

distances[neighbourNode.Number] == distances[iCur] - neighbourNode.Neighbourhood[curNode])

{

iCur = neighbourNode.Number;

ascendingWay.Add(iCur);

break;

}

}

}

ascendingWay.Reverse();

return ascendingWay;

}

public Dictionary<Node, List<Node>> GetMinSpanningTree()

{

int minDistance;

int iMinNeibour = 0;

int iCur = 0;

bool[] isChecked = new bool[\_nodes.Count];

Dictionary<Node, List<Node>> spanningTree = new Dictionary<Node, List<Node>>();

for (int i = 0; i < \_nodes.Count; i++)

spanningTree.Add(\_nodes[i], new List<Node>());

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

{

minDistance = -1;

isChecked[iMinNeibour] = true;

for (int j = 0; j < \_nodes.Count; j++)

{

if (isChecked[j])

{

foreach (KeyValuePair<Node, int> neighbour in \_nodes[j].Neighbourhood)

{

if (!isChecked[neighbour.Key.Number] && (neighbour.Value < minDistance || minDistance == -1))

{

minDistance = neighbour.Value;

iMinNeibour = neighbour.Key.Number;

iCur = j;

}

}

}

}

spanningTree[GetNode(iCur)].Add(GetNode(iMinNeibour));

}

return spanningTree;

}

}

class Topic

{

public int NumberOfTops { get; set; }

public int[,] MatrixAdjacency { get; set; }

public Topic(int numb)

{

NumberOfTops = numb;

MatrixAdjacency = new int[numb, numb];

}

public void ReadMatrixFromFile(string filename)

{

string[] rowsOfNodes = new string[NumberOfTops];

string[] columns = new string[NumberOfTops];

rowsOfNodes = File.ReadAllLines(filename);

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

{

columns = rowsOfNodes[i].Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries);

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

MatrixAdjacency[i, j] = Convert.ToInt32(columns[j]);

}

}

public Graph InitializeGraph()

{

Graph g = new Graph();

int distance = 0;

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

{

g.AddNode(i);

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

{

distance = MatrixAdjacency[i, j];

if (distance != 0)

{

g.AddNode(j);

Node iNode = g.GetNode(i);

Node jNode = g.GetNode(j);

// Признак того, что граф ориентированный.

if (distance != MatrixAdjacency[j, i])

jNode.Neighbourhood.Add(iNode, -1);

iNode.Neighbourhood.Add(jNode, distance);

}

}

}

return g;

}

public Graph GetNonOrientedGraph()

{

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

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

if (MatrixAdjacency[i, j] != 0)

MatrixAdjacency[j, i] = MatrixAdjacency[i, j];

else

MatrixAdjacency[i, j] = MatrixAdjacency[j, i];

return InitializeGraph();

}

public string DoDijkstra(Graph g)

{

Console.WriteLine("Нумерация вершин начинается с \"0\".");

Console.Write("Введите начальную вершину: ");

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

Console.Write("Введите конечную вершину: ");

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

int[] distances = g.GetMinDistancesToOther(start);

if (distances[finish] == -1)

return $"Пути между вершинами {start} и {finish} не существует.";

List<int> way = g.GetWay(distances, start, finish);

StringBuilder result = new StringBuilder($"Крайчайший путь по алгоритму Дейкстры между вершинами" +

$" {start} и {finish}: {distances[finish]}.\n");

result.Append("Путь: ");

foreach (int node in way)

result.Append($"{node} ");

return result.ToString();

}

public string DoPrima(Graph g)

{

Dictionary<Node, List<Node>> spanningTree = g.GetMinSpanningTree();

StringBuilder result = new StringBuilder($"Связи между вершинами, в \"[]\" указано расстояние до соседних вершин.\n");

int minWeight = 0;

int curWeight = 0;

foreach (KeyValuePair<Node, List<Node>> node in spanningTree)

{

if (node.Value.Count == 0)

continue;

result.Append($"Вершина {node.Key.Number}: ");

foreach (Node neighbour in node.Value)

{

curWeight = neighbour.Neighbourhood[node.Key];

result.Append($"{neighbour.Number}[{curWeight}] ");

minWeight += curWeight;

}

result.AppendLine();

}

result.Append($"Мининимальный вес основного дерева по алгоритму Прима = {minWeight}.");

return result.ToString();

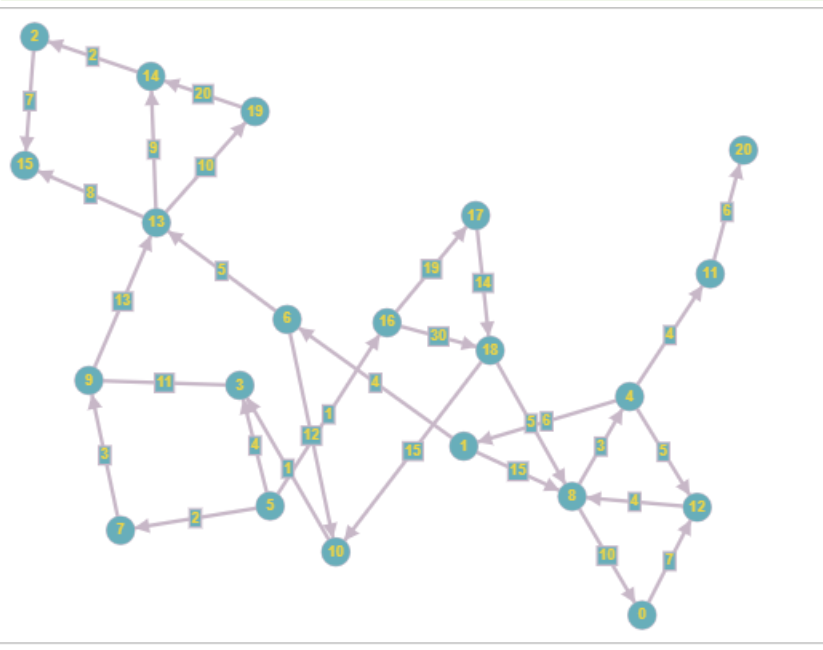
}

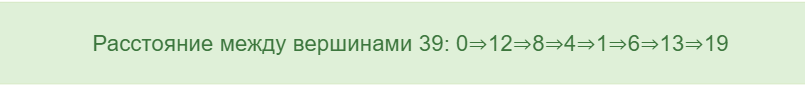
}

**Результаты работы программы.**

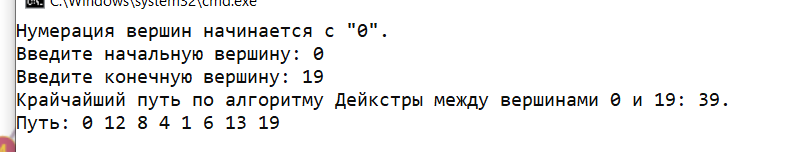
**Задание №1.**

1. **Входные данные:**



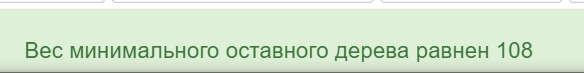
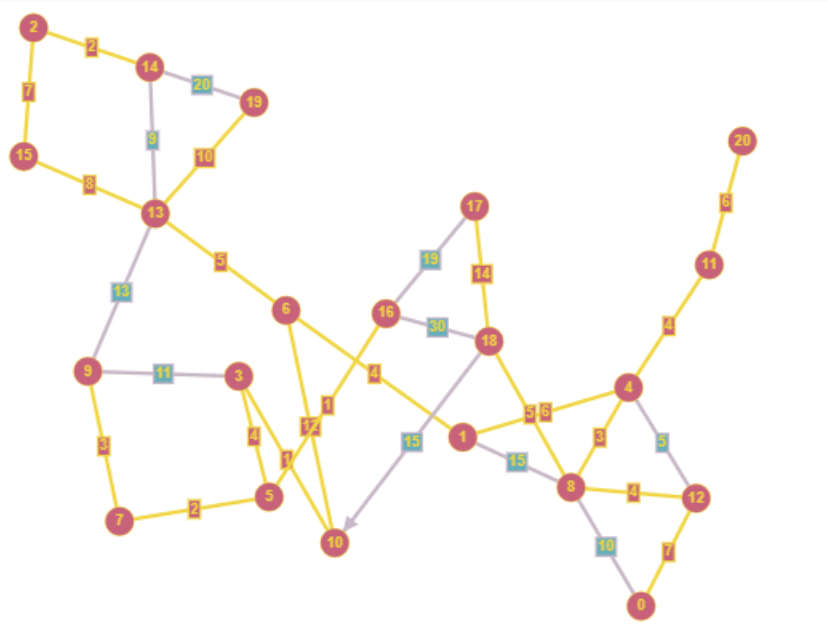
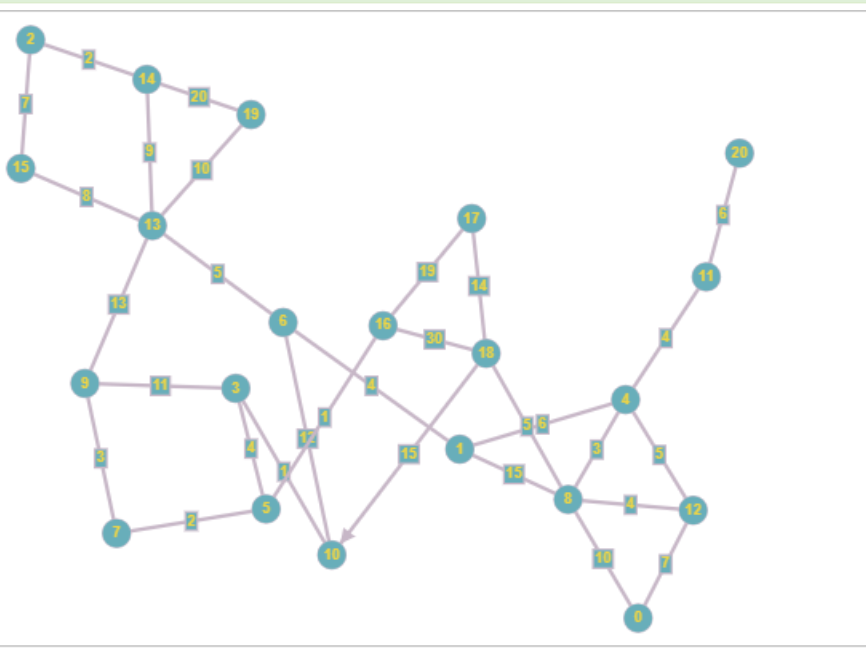


1. **Выходные данные:**



**Задание №2.**

1. **Входные данные:**



1. **Выходные данные:**

