УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №7.2

по предмету «Основы алгоритмизации и программирования»

Вариант 11

Выполнил:

Егоров А.С.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

В графе, заданном списками инцидентности, найти кратчайшие

расстояния между всеми парами городов (алгоритм Форда-

Фалкерсона). Граф визуализировать.

**Код программы Delphi:**

Unit GraphUnit;

Interface

Uses

Winapi.Windows,

System.SysUtils, System.Classes,

Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls,

LinkedList, EdgeUnit, Vcl.StdCtrls;

Type

TuVCLGraph = Class(TForm)

PaintBox: TPaintBox;

Procedure CreateParams(Var Params: TCreateParams); Override;

Procedure FormShow(Sender: TObject);

Procedure FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure PaintBoxPaint(Sender: TObject);

Private

FEdges: TList<TEdge>;

FNodes: TArray<TList<Integer>>;

Procedure DrawVerteces(Const CenterX, CenterY, Distance: Integer;

Const RotateAngle: Real);

Procedure DrawVertex(Const X, Y: Integer; Const Text: String);

Procedure DrawEdges(Const CenterX, CenterY, Distance: Integer;

Const RotateAngle: Real; IndexV: Integer);

Procedure DrawEdge(Const X1, Y1, X2, Y2: Integer; Const Text: String);

Public

Property Edges: TList<TEdge> Write FEdges;

Property Nodes: TArray < TList < Integer >> Write FNodes;

End;

Var

UVCLGraph: TuVCLGraph;

Implementation

{$R \*.dfm}

Uses

Math;

Const

VertexSize = 30;

Procedure TuVCLGraph.CreateParams(Var Params: TCreateParams);

Begin

Inherited;

Params.ExStyle := Params.ExStyle Or WS\_EX\_APPWINDOW;

End;

Procedure TuVCLGraph.FormShow(Sender: TObject);

Begin

PaintBoxPaint(PaintBox);

End;

Procedure TuVCLGraph.DrawEdge(Const X1, Y1, X2, Y2: Integer;

Const Text: String);

Begin

With PaintBox.Canvas Do

Begin

MoveTo(X1 + VertexSize Div 2, Y1 + VertexSize Div 2);

LineTo(X2 + VertexSize Div 2, Y2 + VertexSize Div 2);

TextOut((X1 + X2) Div 2, (Y1 + Y2) Div 2, Text);

End;

End;

Procedure TuVCLGraph.DrawEdges(Const CenterX, CenterY, Distance: Integer;

Const RotateAngle: Real; IndexV: Integer);

Var

PairNodes: TArray<Integer>;

CurEdge: TEdge;

CurEdgesArr: TArray<TEdge>;

I, J: Integer;

Begin

PairNodes := FNodes[IndexV].ToArray();

For I := 0 To High(PairNodes) Do

Begin

CurEdge.A := IndexV;

CurEdge.B := PairNodes[I];

CurEdgesArr := FEdges.FindAllByValue(CurEdge);

For J := 0 To High(CurEdgesArr) Do

// добавить вес

DrawEdge(Round(CenterX + Distance \* Sin(IndexV \* RotateAngle)),

Round(CenterY - Distance \* Cos(IndexV \* RotateAngle)),

Round(CenterX + Distance \* Sin((PairNodes[I]) \* RotateAngle)),

Round(CenterY - Distance \* Cos((PairNodes[I]) \* RotateAngle)),

IntToStr(CurEdgesArr[J].Weight));

End;

End;

Procedure TuVCLGraph.DrawVerteces(Const CenterX, CenterY, Distance: Integer;

Const RotateAngle: Real);

Var

I: Integer;

Angle: Real;

Begin

Angle := 0;

For I := 0 To High(FNodes) Do

Begin

DrawEdges(CenterX, CenterY, Distance, RotateAngle, I);

DrawVertex(Round(CenterX + Distance \* Sin(Angle)),

Round(CenterY - Distance \* Cos(Angle)), IntToStr(I + 1));

Angle := Angle + RotateAngle;

End;

End;

Procedure TuVCLGraph.DrawVertex(Const X, Y: Integer; Const Text: String);

Begin

With PaintBox.Canvas Do

Begin

Ellipse(X, Y, X + VertexSize, Y + VertexSize);

TextOut(X + (VertexSize - TextWidth(Text)) Div 2,

Y + (VertexSize - TextHeight(Text)) Div 2, Text);

End;

End;

Procedure TuVCLGraph.FormKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If Key = VK\_ESCAPE Then

Close;

End;

Procedure TuVCLGraph.PaintBoxPaint(Sender: TObject);

Var

CenterX, CenterY, Distance: Integer;

RotateAngle: Real;

Count: Integer;

Begin

Count := Length(FNodes);

If Count <> 0 Then

Begin

PaintBox.Canvas.Pen.Width := 1;

PaintBox.Canvas.Brush.Color := ClWhite;

PaintBox.Canvas.Pen.Color := ClWhite;

CenterX := PaintBox.Width Div 2;

CenterY := PaintBox.Height Div 2;

Distance := Round(LogN(2, Count) \* VertexSize);

RotateAngle := 2 \* Pi / Count;

DrawVerteces(CenterX, CenterY, Distance, RotateAngle);

End;

End;

Function TuVCLFindPath.FindMinPath(StartPoint, FinshPoint: Integer)

: TArray<Integer>;

Const

INF: Integer = 100;

Var

I, J: Integer;

From: TArray<Integer>;

Dist: TArray<Integer>;

Path: TList<Integer>;

Count: Integer;

Begin

SetLength(Dist, FCountNodes);

For I := 0 To High(Dist) Do

Dist[I] := INF;

SetLength(From, FCountNodes);

For I := 0 To High(From) Do

From[I] := -1;

Dist[StartPoint] := 0;

Count := FCountNodes - 2;

For I := 0 To Count Do

For J := 0 To High(FEdges) Do

If (Dist[FEdges[J].A] <> INF) And

(Dist[FEdges[J].B] > Dist[FEdges[J].A] + FEdges[J].Weight) Then

Begin

Dist[FEdges[J].B] := Dist[FEdges[J].A] + FEdges[J].Weight;

From[FEdges[J].B] := FEdges[J].A;

End;

Path := TList<Integer>.Create();

I := FinshPoint;

While I <> -1 Do

Begin

Path.Add(I);

I := From[I];

End;

If (Path.Count = 1) And (Path[0] = FinshPoint) Then

Path.Clear

Else

Path.Reverse;

FindMinPath := Path.ToArray;

End;

End.

**Код программы CSharp:**

**Solver.cs**

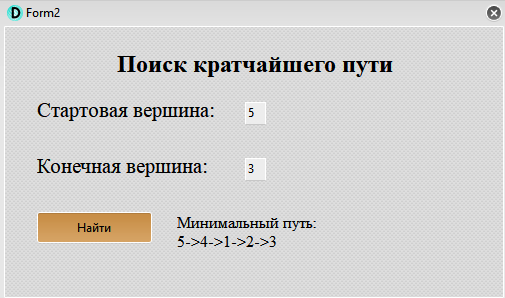
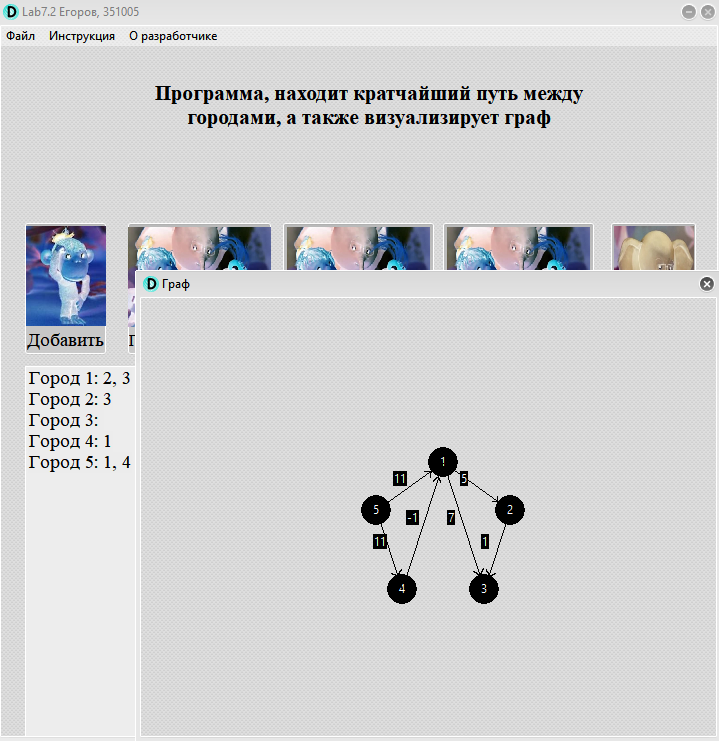
namespace Lab7\_2;  
  
public static class Solver  
{  
 public static int[] FindPath(int countOfNodes, Edge[] edges,int start,int finish)  
 {  
 const int INF = Int32.MaxValue;  
 int[] from = new int[countOfNodes];  
 Array.Fill(from,-1);  
 int[] dist = new int[countOfNodes];  
 Array.Fill(dist,INF);  
 dist[start] = 0;  
 int count = countOfNodes - 1;  
 for (int i = 0; i < count; i++)  
 {  
 foreach (var (a,b,weight) in edges)  
 {  
 if (dist[a] != INF && dist[b] > dist[a] + weight)  
 {  
 dist[b] = dist[a] + weight;  
 from[b] = a;  
 }  
 }   
 }  
  
 List<int> path = new List<int>();   
 for (int v = finish; v != -1; v = from[v])  
 path.Add(v);  
 path.Reverse();  
 return path.ToArray();  
 }  
   
}

**Edge.cs**

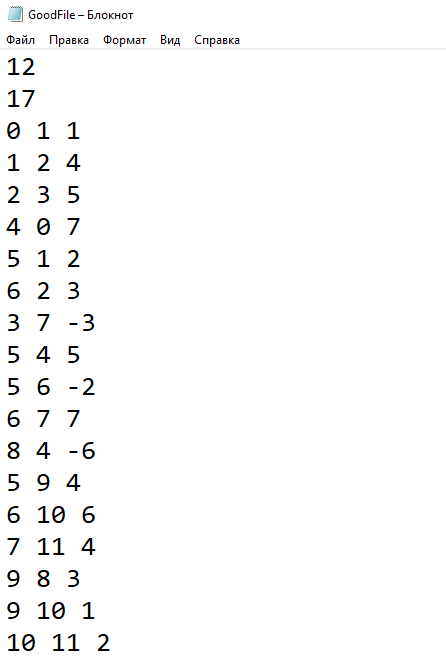
public struct Edge  
{  
 public int a;  
 public int b;  
 public int weight;  
  
 public void Deconstruct(out int a, out int b, out int weight)  
 {  
 a = this.a;  
 b = this.b;  
 weight = this.weight;  
 }  
}

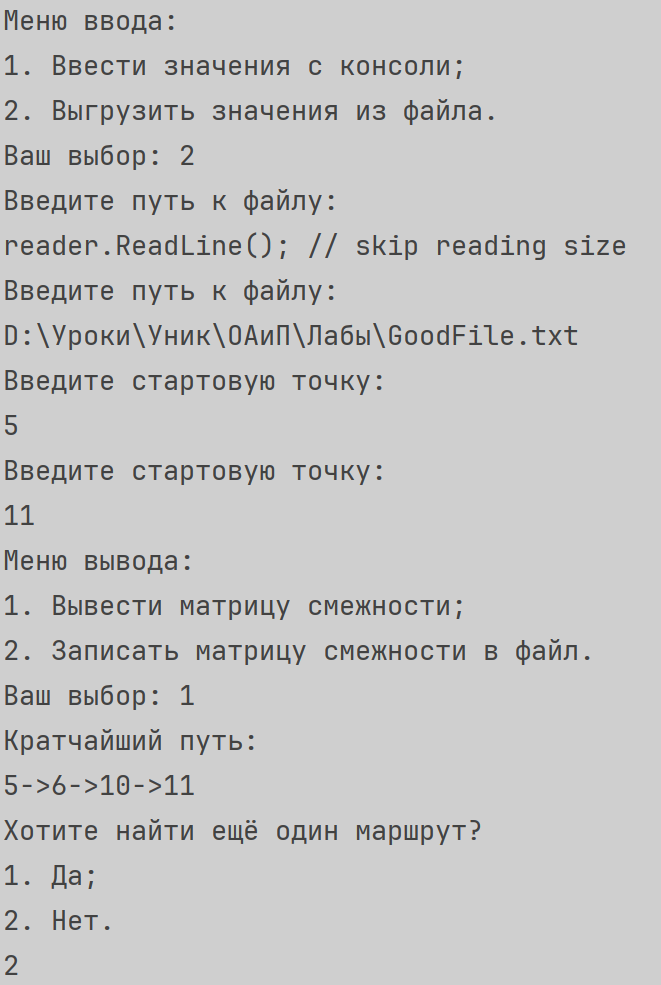
**Скриншоты:**

**Delphi:**

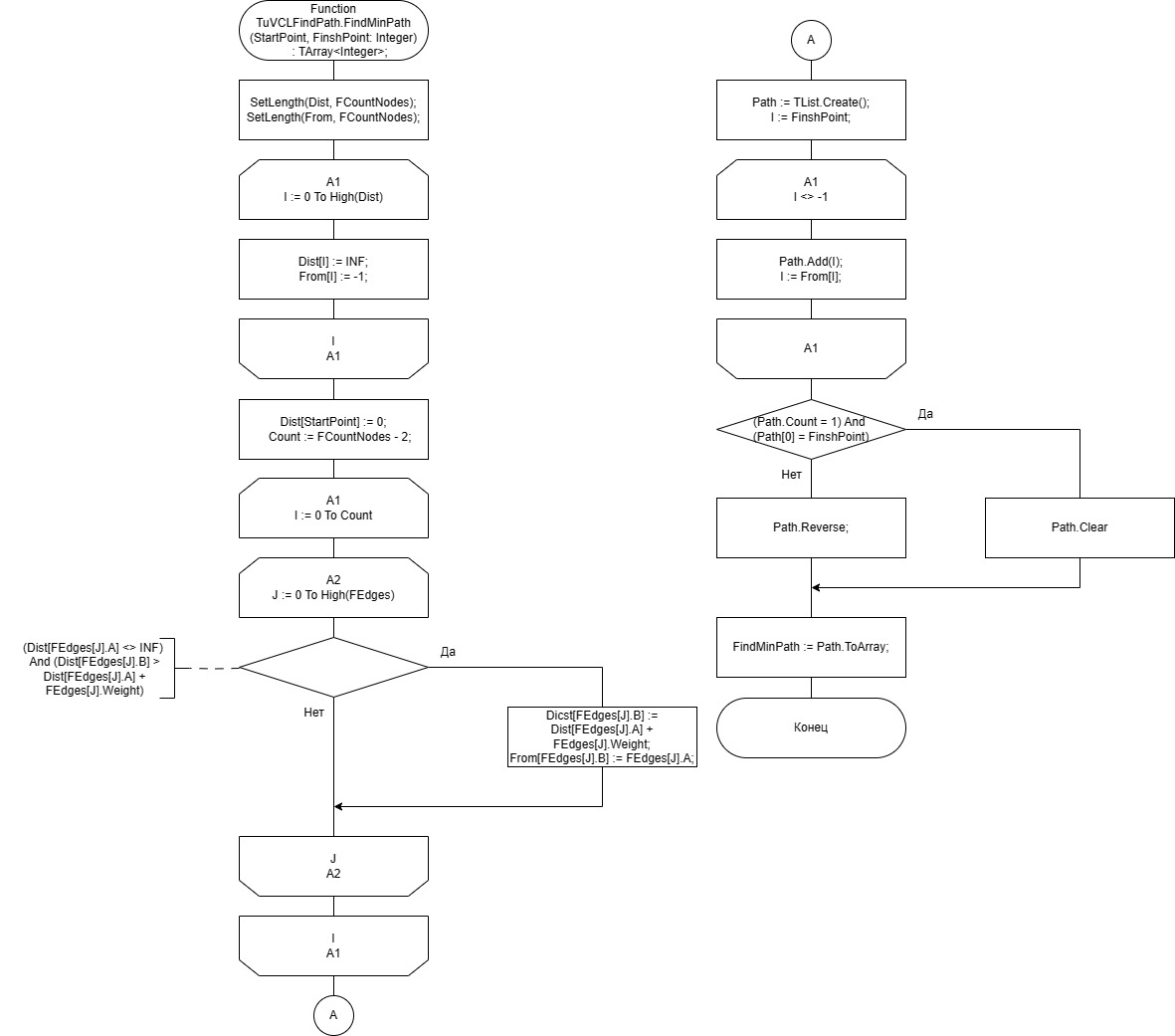
****

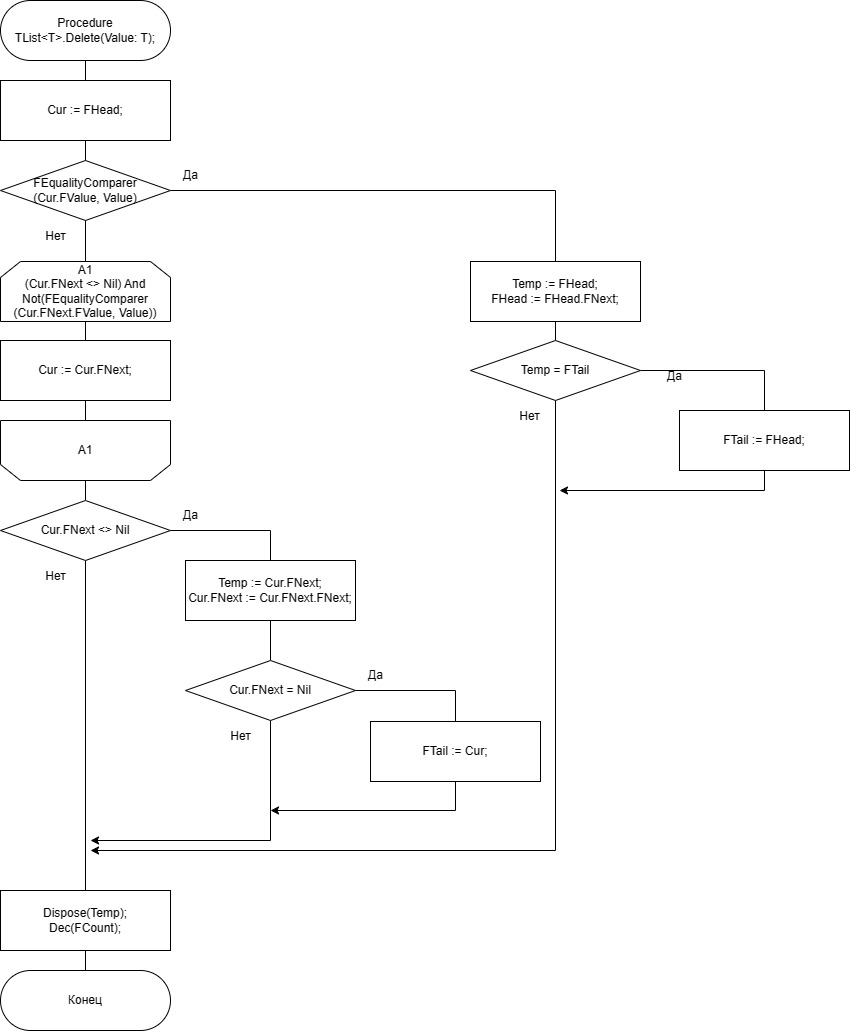
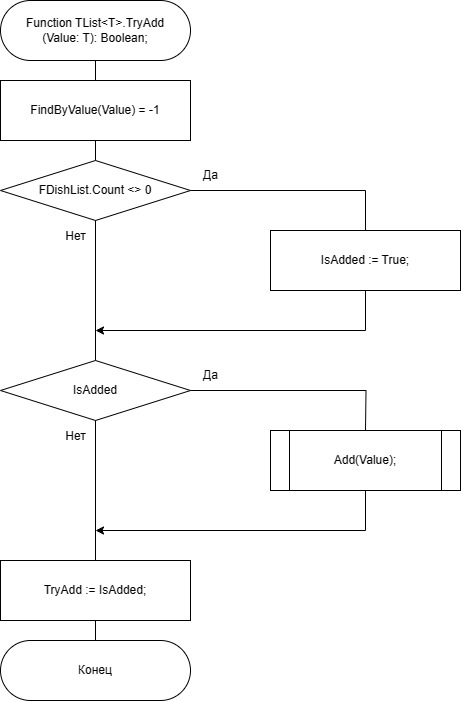
**CSharp:**

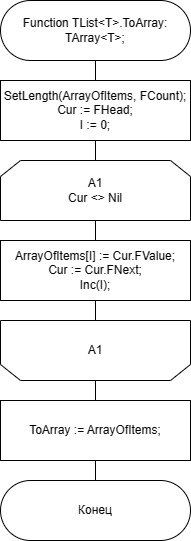
****

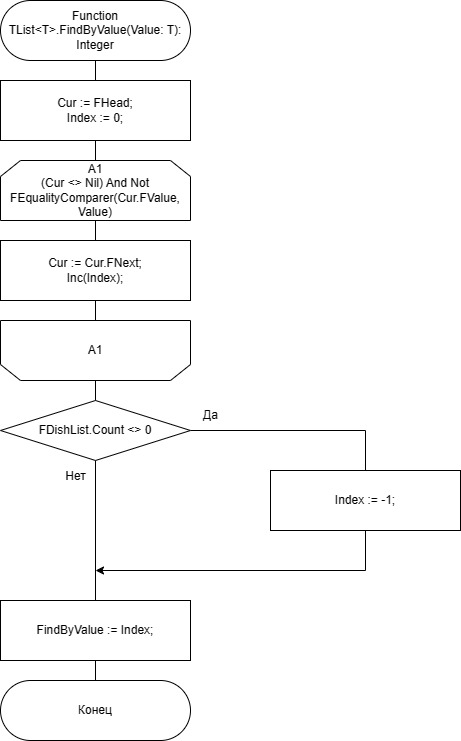


**Блок Схема:**

****



****

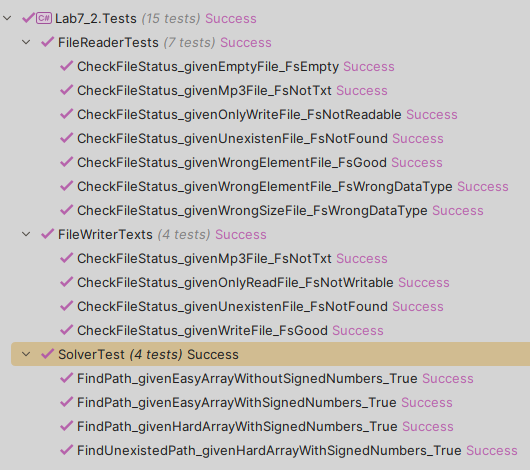


**Юнит Тесты**

**Код**

using System;  
using System.Collections.Generic;  
using JetBrains.Annotations;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using Lab7\_2;  
using Microsoft.VisualStudio.TestPlatform.TestHost;  
  
  
[TestClass]  
[TestSubject(typeof(FileReader))]  
public class FileReaderTests  
{  
 private static FileReader fileReader = new FileReader();   
  
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyWriteFile\_FsNotReadable()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyWriteFile.txt";  
 fileReader.InputCountOfEdges();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotReadable);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongSizeFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongSizeFile.txt";  
 fileReader.InputCountOfNodes();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongTypeFile.txt";  
 fileReader.InputEdges(fileReader.InputCountOfEdges());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenEmptyFile\_FsEmpty()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\EmptyFile.txt";  
 fileReader.InputCountOfNodes();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsEmpty);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsGood()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFile.txt";  
 fileReader.InputEdges(fileReader.InputCountOfEdges());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(FileWriter))]  
public class FileWriterTexts  
{  
 private static FileWriter fileWriter = new FileWriter();  
   
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyReadFile\_FsNotWritable()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyReadFile.txt";  
 int[] temp = { 1, 5, 42, 4, 1 };  
 fileWriter.Output(temp);  
 Assert.AreEqual( fileWriter.FileStatus,FileStatus.FsNotWritable);  
 }   
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWriteFile\_FsGood()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\OnlyWriteFile.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(Solver))]  
public class SolverTest  
{  
 private static FileReader fileReader = new FileReader();  
   
 [TestMethod]  
 public void FindPath\_givenEasyArrayWithoutSignedNumbers\_True()  
 {  
 const int START = 4;  
 const int FINISH = 2;  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFileEasyUnSigned.txt";  
 int[] answer = { 4,0,1,2 };  
 CollectionAssert.AreEquivalent(Solver.FindPath(fileReader.InputCountOfNodes(),fileReader.InputEdges(fileReader.InputCountOfEdges()),START,FINISH),answer);  
 }  
   
 [TestMethod]  
 public void FindPath\_givenEasyArrayWithSignedNumbers\_True()  
 {  
 const int START = 4;  
 const int FINISH = 2;  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFileEasySigned.txt";  
 int[] answer = { 4,3,0,1,2 };  
 CollectionAssert.AreEquivalent(Solver.FindPath(fileReader.InputCountOfNodes(),fileReader.InputEdges(fileReader.InputCountOfEdges()),START,FINISH),answer);  
 }  
   
 [TestMethod]  
 public void FindPath\_givenHardArrayWithSignedNumbers\_True()  
 {  
 const int START = 5;  
 const int FINISH = 11;  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFileHard.txt";  
 int[] answer = { 5,6,10,11 };  
 CollectionAssert.AreEquivalent(Solver.FindPath(fileReader.InputCountOfNodes(),fileReader.InputEdges(fileReader.InputCountOfEdges()),START,FINISH),answer);  
 }  
   
 [TestMethod]  
 public void FindUnexistedPath\_givenHardArrayWithSignedNumbers\_True()  
 {  
 const int START = 2;  
 const int FINISH = 4;  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFileHard.txt";  
 int[] answer = {4};  
 CollectionAssert.AreEquivalent(Solver.FindPath(fileReader.InputCountOfNodes(),fileReader.InputEdges(fileReader.InputCountOfEdges()),START,FINISH),answer);  
 }  
}

**Скриншоты:**

****