

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра программных систем

**Задание №3**

**Алгоритмы на графах**

Выполнил: Мороз

Илья Олегович

Группа № K3220

Преподаватель: Иванов Сергей Евгеньевич

Санкт-Петербург

2020

**Задание:** реализовать способ задачи графов с помощью матриц смежности и инцидентности. Для графов реализовать алгоритмы поиска в глубину, в ширину, алгоритм Дейкстры.

**Исходный код:**

**Program.cs**

using System;  
  
namespace task3  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 */\*  
 \* Adjacency matrix  
 \*/* GraphByAdjacencyMatrix graphByAdjacencyMatrix = new GraphByAdjacencyMatrix("adjacencyMatrix.txt");  
 Console.WriteLine(graphByAdjacencyMatrix);  
  
 */\*  
 \* Depth first search for graph by adjacency matrix  
 \*/* int[] depthFirstSearchResult0 = DepthFirstSearch.BypassPath(graphByAdjacencyMatrix, 1);  
 Console.WriteLine("Result of DFS:");  
 Console.WriteLine(string.Join(" -> ", depthFirstSearchResult0));  
  
 */\*  
 \* Breadth first search for graph by adjacency matrix  
 \*/* int[] breadthFirstSearch0 = BreadthFirstSearch.BypassPath(graphByAdjacencyMatrix, 0, 2);  
 Console.WriteLine("Result of BFS:");  
 if (breadthFirstSearch0.Length == 0)  
 {  
 Console.WriteLine("BFS is impossible!");  
 }  
 else  
 {  
 Console.WriteLine(string.Join(" -> ", breadthFirstSearch0));  
 }  
  
 */\*  
 \* Incidence matrix  
 \*/* GraphByIncidenceMatrix graphByIncidenceMatrix = new GraphByIncidenceMatrix("incidenceMatrix.txt");  
 Console.WriteLine(graphByIncidenceMatrix);  
  
 */\*  
 \* Depth first search for graph by adjacency matrix  
 \*/* int[] depthFirstSearchResult = DepthFirstSearch.BypassPath(graphByIncidenceMatrix, 1);  
 Console.WriteLine("Result of DFS:");  
 Console.WriteLine(string.Join(" -> ", depthFirstSearchResult));  
  
 */\*  
 \* Breadth first search for graph by adjacency matrix  
 \*/* int[] breadthFirstSearch = BreadthFirstSearch.BypassPath(graphByIncidenceMatrix, 0, 3);  
 Console.WriteLine("Result of BFS:");  
 if (breadthFirstSearch.Length == 0)  
 {  
 Console.WriteLine("BFS is impossible!");  
 }  
 else  
 {  
 Console.WriteLine(string.Join(" -> ", breadthFirstSearch));  
 }  
  
 */\*  
 \* Dijkstra algorithm  
 \*/* int startVortex = 0;  
 int[] distanceCost = DijkstraAlgorithm.BypassPath(graphByIncidenceMatrix, startVortex);  
 Console.WriteLine($"The cost of the path from vortex {startVortex} to the remaining vortexes");  
 for (int i = 1; i < distanceCost.Length; i++)  
 {  
 if (distanceCost[i] == Int32.**MaxValue**)  
 {  
 Console.WriteLine($"Can't find way from {startVortex} to {i}");  
 continue;  
 }  
 Console.WriteLine($"{startVortex} -> {i} = {distanceCost[i]}");  
 }  
 }  
 }  
}

**GraphByAdjacencyMatrix.cs**

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
  
namespace task3  
{  
 */\*  
 \* Class represents graph that was created by adjacency matrix  
 \* Adjacency matrix is reading from file  
 \* At the first line in file you need to write count of vortexes  
 \* Next you write your adjacency matrix  
 \* Attention! Your file should be near compiled program  
 \*/* public class GraphByAdjacencyMatrix : Graph  
 {  
 */\*  
 \* Variable with read adjacency matrix  
 \*/* private int[][] adjacencyMatrix;  
  
 */\*  
 \* Constructor for graph object  
 \*/* public GraphByAdjacencyMatrix(string inputFileName)  
 {  
 ReadFromFile(inputFileName);  
 }  
  
 */\*  
 \* Read adjacency matrix from file  
 \*/* private void ReadFromFile(string inputFileName)  
 {  
 try  
 {  
 StreamReader inputFile = new StreamReader(inputFileName);  
  
 */\*  
 \* Get count of vortexes from file  
 \*/* string lineWithVortexCount = inputFile.ReadLine();  
 int vortexCount = Int32.Parse(lineWithVortexCount);  
 adjacencyMatrix = new int[vortexCount][];  
  
 */\*  
 \* Read lines with information of adjacents  
 \*/* for (int i = 0; i < vortexCount; i++)  
 {  
 string inputLine = inputFile.ReadLine();  
  
 */\*  
 \* Throw exception if line is null  
 \*/* string[] splittedInputLine = (inputLine != null) ? inputLine.Split(" ") : throw new IOException();  
 adjacencyMatrix[i] = splittedInputLine.Select(vortexInfo => Int32.Parse(vortexInfo)).ToArray();  
 }  
  
 inputFile.Close();  
 }  
 catch (Exception e)  
 {  
 ErrorHandler.WriteErrorInConsole(e, e.Message);  
 }  
 }  
  
 */\*  
 \* Return count of vortexes in graph  
 \*/* public override int VortexCount()  
 {  
 return adjacencyMatrix.GetLength(0);  
 }  
  
 */\*  
 \* Return array of vortex neighbours  
 \*/* public override int[] GetVortexNeighbours(int vortex)  
 {  
 List<int> result = new List<int>();  
 for (int i = 0; i < VortexCount(); i++)  
 {  
 if (adjacencyMatrix[vortex][i] != 0 && vortex != i)  
 {  
 result.Add(i);  
 }  
 }  
  
 return result.ToArray();  
 }  
  
 */\*  
 \* Override ToString method for matrix objects  
 \*/* public override string ToString()  
 {  
 string result = "\n" +  
 "Adjacency matrix for graph:\n";  
 for (int i = 0; i < VortexCount(); i++)  
 {  
 for (int j = 0; j < VortexCount(); j++)  
 {  
 result += $"{adjacencyMatrix[i][j]}\t";  
 }  
  
 result += "\n";  
 }  
  
 return result;  
 }  
  
 */\*  
 \* Adjacency matrix does not support getting distance  
 \*/* public override int GetDistance(int vortex1, int vortex2)  
 {  
 throw new NotImplementedException();  
 }  
 }  
}

**GraphByIncidenceMatrix.cs**

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
  
namespace task3  
{  
 */\*  
 \* Class represents graph that was created by incidence matrix  
 \* Incidence matrix is reading from file  
 \* At the first line in file you need to write count of vortexes and count of edges  
 \* Next you write your incidence matrix  
 \* Attention! Your file should be near compiled program  
 \*/* public class GraphByIncidenceMatrix : Graph  
 {  
 */\*  
 \* Variable with read adjacency matrix  
 \*/* private int[][] incidenceMatrix;  
  
 */\*  
 \* Constructor for graph object  
 \*/* public GraphByIncidenceMatrix(string inputFileName)  
 {  
 ReadFromFile(inputFileName);  
 }  
  
 */\*  
 \* Read adjacency matrix from file  
 \*/* private void ReadFromFile(string inputFileName)  
 {  
 try  
 {  
 StreamReader inputFile = new StreamReader(inputFileName);  
  
 */\*  
 \* Get count of vortexes and edges from file  
 \*/* string lineWithVortexAndEdgeCount = inputFile.ReadLine();  
 int vortexCount = Int32.Parse(lineWithVortexAndEdgeCount.Split(" ")[0]);  
 int edgeCount = Int32.Parse(lineWithVortexAndEdgeCount.Split(" ")[1]);  
 incidenceMatrix = new int[vortexCount][];  
  
 */\*  
 \* Read lines with information of incidences  
 \*/* for (int i = 0; i < vortexCount; i++)  
 {  
 string inputLine = inputFile.ReadLine();  
  
 */\*  
 \* Throw exception if line is null  
 \*/* string[] splittedInputLine = (inputLine != null) ? inputLine.Split(" ") : throw new IOException();  
 if (splittedInputLine.Length != edgeCount) throw new IOException();  
 incidenceMatrix[i] = splittedInputLine.Select(vortexInfo => Int32.Parse(vortexInfo)).ToArray();  
 }  
  
 inputFile.Close();  
 }  
 catch (Exception e)  
 {  
 ErrorHandler.WriteErrorInConsole(e, e.Message);  
 }  
 }  
  
 */\*  
 \* Return count of vortexes in graph  
 \*/* public override int VortexCount()  
 {  
 return incidenceMatrix.GetLength(0);  
 }  
  
 */\*  
 \* Return count of edges in graph  
 \*/* public int EdgeCount()  
 {  
 return incidenceMatrix[0].GetLength(0);  
 }  
  
 */\*  
 \* Return array of vortex neighbours  
 \*/* public override int[] GetVortexNeighbours(int vortex)  
 {  
 List<int> result = new List<int>();  
 for (int i = 0; i < EdgeCount(); i++)  
 {  
 if (incidenceMatrix[vortex][i] > 0)  
 {  
 for (int j = 0; j < VortexCount(); j++)  
 {  
 if (j != vortex && incidenceMatrix[j][i] < 0)  
 {  
 result.Add(j);  
 }  
 }  
 }  
 }  
  
 return result.ToArray();  
 }  
  
 */\*  
 \* Override ToString method for matrix objects  
 \*/* public override string ToString()  
 {  
 string result = "\n" +  
 "Incidence matrix for graph:\n";  
 for (int i = 0; i < VortexCount(); i++)  
 {  
 for (int j = 0; j < EdgeCount(); j++)  
 {  
 result += $"{incidenceMatrix[i][j]}\t";  
 }  
  
 result += "\n";  
 }  
  
 return result;  
 }  
  
 */\*  
 \* Return distance from the first vortex to the second  
 \*/* public override int GetDistance(int vortex1, int vortex2)  
 {  
 for (int i = 0; i < EdgeCount(); i++)  
 {  
 if (incidenceMatrix[vortex1][i] != 0 && incidenceMatrix[vortex2][i] != 0 &&  
 incidenceMatrix[vortex1][i] / incidenceMatrix[vortex2][i] == -1)  
 {  
 return Math.Abs(incidenceMatrix[vortex1][i]);  
 }  
 }  
  
 return -1;  
 }  
 }  
}

**ErrorHandler.cs**

using System;  
  
namespace task2  
{  
 public class ErrorHandler  
 {  
 public static void WriteErrorInConsole(Exception exception, string message = "Something went wrong!")  
 {  
 Console.ForegroundColor = ConsoleColor.**Red**;  
 Console.WriteLine(message);  
 Console.WriteLine("Stack trace:");  
 Console.WriteLine(exception.StackTrace);  
 Console.ResetColor();  
 System.Environment.Exit(1);  
 }  
 }  
}

**BreadthFirstSearch.cs**

using System.Collections.Generic;  
using System.Linq;  
  
namespace task3  
{  
 public class BreadthFirstSearch  
 {  
 */\*  
 \* Graph for search  
 \*/* private static Graph graphForSearch;  
  
 private static Queue<int> queue;  
  
 private static int[] colors;  
  
 */\*  
 \* Return array of vortexes in the bypass path  
 \*/* public static int[] BypassPath(Graph graph, int source, int destination)  
 {  
 graphForSearch = graph;  
 colors = new int[graph.VortexCount()];  
 queue = new Queue<int>();  
 Step[] result = Search(source, destination).ToArray();  
 Stack<int> shortestWay = new Stack<int>();  
 shortestWay.Push(result.Last().To);  
 Step currentStep = result.Last();  
 while (currentStep.From != source)  
 {  
 for (int i = 0; i < result.Length; i++)  
 {  
 if (result[i].To == currentStep.From)  
 {  
 shortestWay.Push(result[i].To);  
 currentStep = result[i];  
 break;  
 }  
 }  
 }  
 shortestWay.Push(currentStep.From);  
 return shortestWay.ToArray();  
 }  
  
 */\*  
 \* Recursive function for depth first search  
 \*/* private static List<Step> Search(int source, int destination)  
 {  
 List<Step> result = new List<Step>();  
 colors[source] = 1;  
 if (source == destination)  
 {  
 result.Add(new Step(source, destination));  
 return result;  
 }  
  
 int[] neighbourVortexes = graphForSearch.GetVortexNeighbours(source);  
 foreach (int neighbourVortex in neighbourVortexes)  
 {  
 if (colors[neighbourVortex] == 0)  
 {  
 Step step = new Step(source, neighbourVortex);  
 result.Add(step);  
 if (neighbourVortex == destination)  
 {  
 return result;  
 }  
  
 queue.Enqueue(neighbourVortex);  
 colors[neighbourVortex] = 1;  
 }  
 }  
  
 if (queue.Count != 0)  
 {  
 result.AddRange(Search(queue.Dequeue(), destination));  
 }  
  
 colors[source] = 2;  
  
 return result;  
 }  
 }  
}

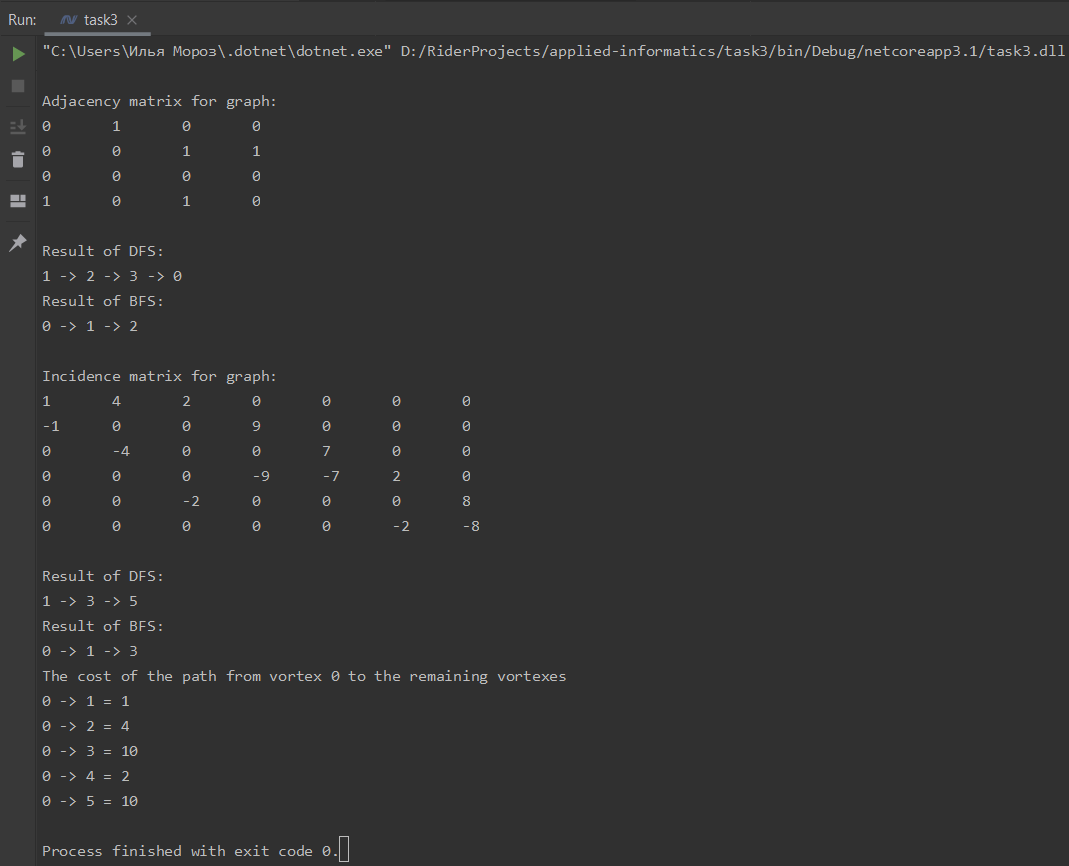
**DepthFirstSearch.cs**

using System.Collections.Generic;  
  
namespace task3  
{  
 */\*  
 \* Class implements depth first search  
 \*/* public class DepthFirstSearch  
 {  
 */\*  
 \* Graph for search  
 \*/* private static Graph graphForSearch;  
  
 */\*  
 \* Information about visited vortexes  
 \*/* private static bool[] isVisited;  
  
 */\*  
 \* Return array of vortexes in the bypass path  
 \*/* public static int[] BypassPath(Graph graph, int vortex)  
 {  
 graphForSearch = graph;  
 isVisited = new bool[graph.VortexCount()];  
 return Search(vortex).ToArray();  
 }  
  
 */\*  
 \* Recursive function for depth first search  
 \*/* private static List<int> Search(int vortex)  
 {  
 List<int> result = new List<int>();  
 result.Add(vortex);  
 isVisited[vortex] = true;  
 int[] neighbourVortexes = graphForSearch.GetVortexNeighbours(vortex);  
 foreach (int neighbour in neighbourVortexes)  
 {  
 if (!isVisited[neighbour])  
 {  
 result.AddRange(Search(neighbour));  
 }  
 }  
  
 return result;  
 }  
 }  
}

**DijkstraAlgorithm.cs**

using System;  
  
namespace task3  
{  
 public class DijkstraAlgorithm  
 {  
 */\*  
 \* Graph for search  
 \*/* private static Graph graphForSearch;  
  
 private static bool[] isVisited;  
  
 private static int[] distance;  
  
 */\*  
 \* Return array of vortexes in the bypass path  
 \*/* public static int[] BypassPath(Graph graph, int vortex)  
 {  
 graphForSearch = graph;  
 isVisited = new bool[graph.VortexCount()];  
 distance = new int[graph.VortexCount()];  
 for (int i = 0; i < graphForSearch.VortexCount(); i++)  
 {  
 distance[i] = Int32.**MaxValue**;  
 }  
  
 return Search(vortex);  
 }  
  
 */\*  
 \* Function for dijkstra algorithm  
 \*/* private static int[] Search(int vortex)  
 {  
 int index = 0, u = 0;  
 distance[vortex] = 0;  
 for (int count = 0; count < graphForSearch.VortexCount() - 1; count++)  
 {  
 int min = Int32.**MaxValue**;  
 for (int i = 0; i < graphForSearch.VortexCount(); i++)  
 {  
 if (!isVisited[i] && distance[i] <= min)  
 {  
 min = distance[i];  
 index = i;  
 }  
 }  
  
 u = index;  
 isVisited[u] = true;  
 int[] neighbourVortexes = graphForSearch.GetVortexNeighbours(u);  
 foreach (int neighbourVortex in neighbourVortexes)  
 {  
 int distanceTemp = graphForSearch.GetDistance(u, neighbourVortex);  
 if (!isVisited[neighbourVortex] &&  
 distance[u] + distanceTemp < distance[neighbourVortex] &&  
 distance[u] != Int32.**MaxValue**)  
 {  
 distance[neighbourVortex] = distance[u] + distanceTemp;  
 }  
 }  
 }  
  
 return distance;  
 }  
 }  
}

**Результат работы:**

****

**Вывод:** были реализованы алгоритмы работы с графами: поиск в глубину, поиск в ширину, алгоритм Дейкстры. Графы задавались с помощью матриц смежности и инцидентности.

Исходный код также доступен на GitHub: [ilyamore88/applied-informatics/task3](https://github.com/ilyamore88/applied-informatics/tree/master/task3).