Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

ЗВІТ

про виконання лабораторного практикуму №4(2)

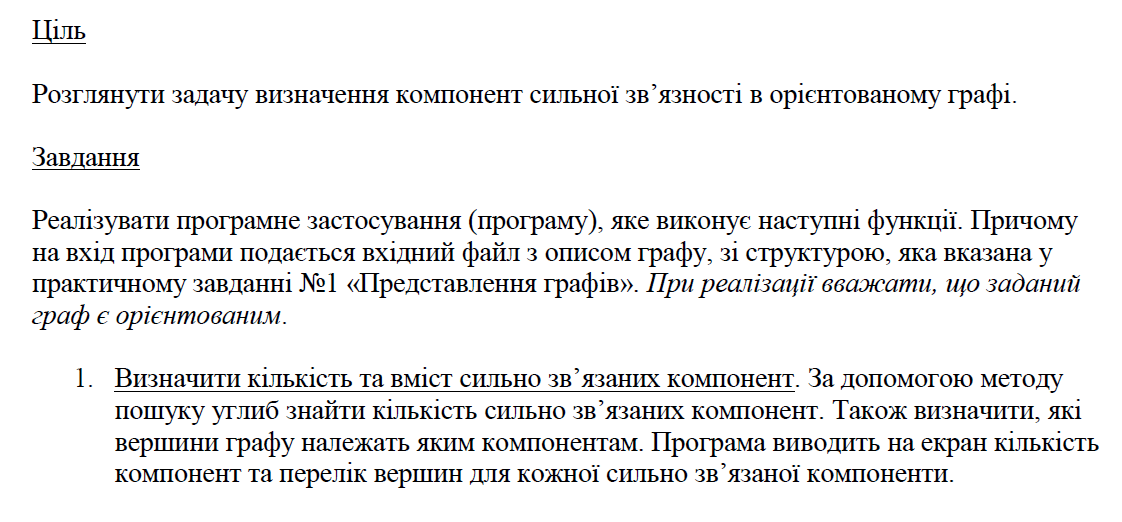
Виконав:

студент 1- го курсу ФІОТ

групи *ІП-91*

*Кінчур Вадим Вікторович*

Київ 2020

2. Умова лабораторної роботи

3. Програмний код(C#):

using System.Collections.Generic;  
using System;  
using System.IO;  
  
namespace LAB\_4\_2\_  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 Parser parser = new Parser("input.txt");  
 (int n, int m, List<(int, int)> edgesList) = parser.GetInput();  
 DirectedGraph graph = new DirectedGraph(n, m, edgesList);  
  
 System.Console.Write("Do you want to work with file or console?(console/file) : ");  
 string writeType = System.Console.ReadLine();  
  
 if (writeType == "console")  
 {  
 GraphAlgorithm graphAlg = new GraphAlgorithm(graph, "console");  
 graphAlg.PrintStrongConnectedComponents();  
 }  
 else  
 {  
 System.Console.Write("Type the name of output file : ");  
 string fileName = System.Console.ReadLine();  
 GraphAlgorithm graphAlg = new GraphAlgorithm(graph, "file", fileName);  
 graphAlg.PrintStrongConnectedComponents();  
 }  
 }  
 }  
  
 class Parser  
 {  
 private readonly StreamReader file;  
 public Parser(string fileName)  
 {  
 string path = "../../../" + fileName;  
 file = new StreamReader(path);  
 }  
  
 private (int, int) ParseRow(string row)  
 {  
 string[] characters = row.Split(" ");  
 int[] numbers = Array.ConvertAll(characters, s => int.Parse(s));  
 return (numbers[0], numbers[1]);  
  
 }  
  
 public (int, int, List<(int, int)>) GetInput()  
 {  
 string line = file.ReadLine();  
 (int n, int m) = ParseRow(line);  
 List<(int, int)> edgesList = new List<(int, int)>();  
  
 for (int i = 0; i < m; ++i)  
 {  
 line = file.ReadLine();  
 if (line != null)  
 edgesList.Add(ParseRow(line));  
 }  
 return (n, m, edgesList);  
 }  
 }  
  
 class Writer  
 {  
 private readonly string writeMode;  
 private readonly string fileName;  
 private readonly System.IO.StreamWriter cw;  
  
 public Writer(string writeMode, string fileName = "output.txt")  
 {  
 this.writeMode = writeMode;  
 if (this.writeMode == "file")  
 {  
 this.fileName = fileName;  
 this.cw = new System.IO.StreamWriter("../../../" + fileName);  
 cw.Close();  
 this.cw = new System.IO.StreamWriter("../../../" + fileName, true);  
 }  
 }  
  
 public void Write(string line = "")  
 {  
 if (writeMode == "console")  
 {  
 System.Console.Write(line);  
 }  
 else  
 {  
 cw.Write(line);  
 }  
 }  
  
 public void WriteLine(string line = "")  
 {  
 Write(line + "\n");  
 }  
  
 public void Close()  
 {  
 if (writeMode != "console")  
 {  
 cw.Close();  
 }  
 }  
 }  
  
 class DirectedGraph  
 {  
 public int verticesNum { get; }  
 private readonly int edgesNum;  
 private List<(int, int)> edgesList;  
  
 public DirectedGraph(int n, int m, List<(int, int)> edgesArray)  
 {  
 verticesNum = (n >= 0) ? n : 0;  
 edgesNum = (m >= 0) ? m : 0;  
 edgesList = edgesArray;  
 }  
  
 public DirectedGraph(DirectedGraph graph)  
 {  
 verticesNum = graph.verticesNum;  
 edgesNum = graph.edgesNum;  
 edgesList = new List<(int, int)>(graph.edgesList);  
 }  
  
 public Dictionary<int, List<int>> GetAdjList()  
 {  
 Dictionary<int, List<int>> adjList = new Dictionary<int, List<int>>();  
  
 for (int i = 0; i < verticesNum; ++i)  
 {  
 adjList.Add(i, new List<int>());  
 }  
  
 foreach (var edge in edgesList)  
 {  
 (int start, int finish) = (edge.Item1 - 1, edge.Item2 - 1);  
 if (!adjList[start].Contains(finish))  
 {  
 adjList[start].Add(finish);  
 }  
 }  
 foreach (var key in adjList.Keys)  
 {  
 adjList[key].Sort((a, b) => a.CompareTo(b));  
 }  
 return adjList;  
 }  
  
 public void Transpose()  
 {  
 List<(int, int)> transposedList = new List<(int, int)>();  
  
 foreach (var edge in edgesList)  
 {  
 transposedList.Add((edge.Item2, edge.Item1));  
 }  
  
 edgesList = new List<(int, int)>(transposedList);  
 }  
 }  
  
 class GraphAlgorithm  
 {  
 private readonly Writer writer;  
 private readonly DirectedGraph graph;  
  
 public GraphAlgorithm(DirectedGraph graph, string writeMode, string fileName = null)  
 {  
 this.graph = new DirectedGraph(graph);  
 writer = new Writer(writeMode, fileName);  
 }  
  
 private int[] TopologicalSort()  
 {  
 HashSet<int> visited = new HashSet<int>();  
 int[] answer = new int[graph.verticesNum];  
 Dictionary<int, List<int>> adjList = graph.GetAdjList();  
 int currentPlace = graph.verticesNum;  
  
  
 for (int vertex = 0; vertex < graph.verticesNum; ++vertex)  
 {  
 if (!visited.Contains(vertex))  
 {  
 DFS(vertex);  
 }  
 }  
  
 void DFS(int start)  
 {  
 visited.Add(start);  
  
 foreach (var vertex in adjList[start])  
 {  
 if (!visited.Contains(vertex))  
 {  
 DFS(vertex);  
 }  
 }  
 answer[--currentPlace] = start + 1;  
 }  
 return answer;  
 }  
  
  
 private List<List<int>> GetConnectComponents()  
 {  
 int[] sortedVertices = TopologicalSort();  
 graph.Transpose();  
 Dictionary<int, List<int>> adjList = graph.GetAdjList();  
 HashSet<int> visited = new HashSet<int>();  
 List<int> component = new List<int>();  
 List<List<int>> componentsList = new List<List<int>>();  
 int humanIndex = 1;  
  
 foreach (var vertex in sortedVertices)  
 {  
 if (!visited.Contains(vertex - humanIndex))  
 {  
 DFS(vertex - humanIndex);  
 }  
 if (component.Count != 0)  
 {  
 componentsList.Add(new List<int>(component));  
 component.Clear();  
 }  
 }  
  
 void DFS(int start)  
 {  
 visited.Add(start);  
 component.Add(start + humanIndex);  
  
 foreach (var vertex in adjList[start])  
 {  
 if (!visited.Contains(vertex))  
 {  
 DFS(vertex);  
 }  
 }  
 }  
  
 return componentsList;  
 }  
  
 public void PrintStrongConnectedComponents()  
 {  
 List<List<int>> componetsList = GetConnectComponents();  
  
 writer.WriteLine($"The number of strongly connected components is equal to {componetsList.Count}");  
 writer.WriteLine("\nComponets : ");  
  
 foreach (var component in componetsList)  
 {  
 component.Sort();  
 foreach (var vertex in component)  
 {  
 writer.Write($"{vertex} ");  
 }  
 writer.WriteLine();  
 }  
  
 writer.Close();  
 }  
 }  
}

4. Результати виконання:

**Вхідні дані:**

5 5

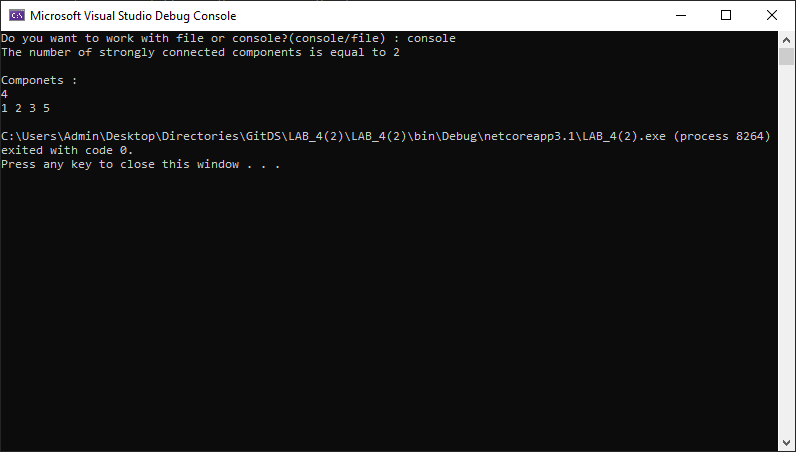
1 2

2 3

3 1

1 5

5 1



**Вхідні дані :**

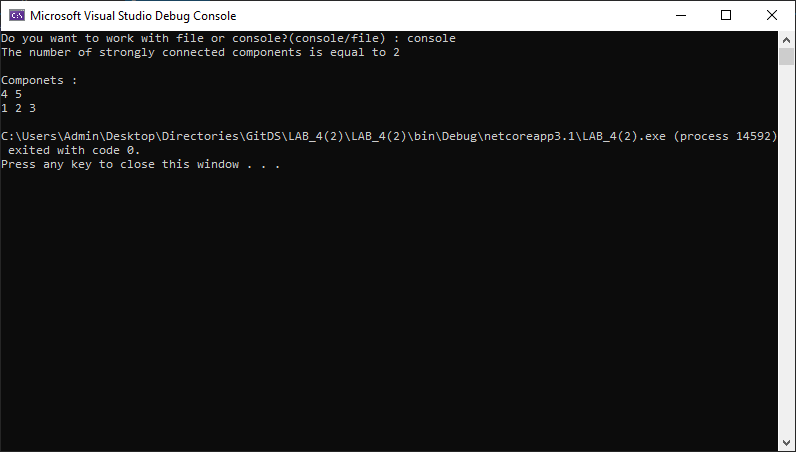
5 5

1 2

2 3

3 1

4 5

5 4