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 №3

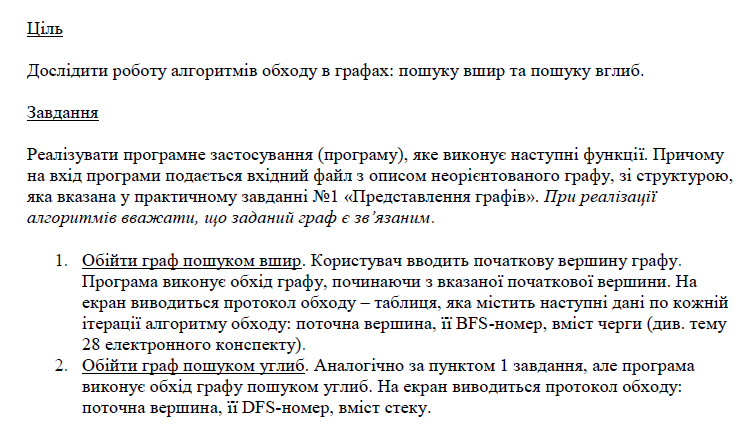
Виконав:

студент 1- го курсу ФІОТ

групи *ІП-91*

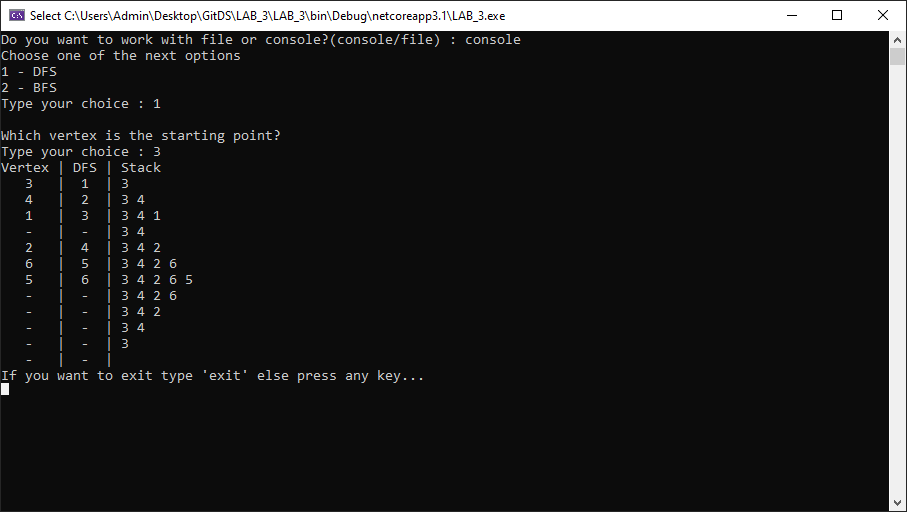
*Кінчур Вадим Вікторович*

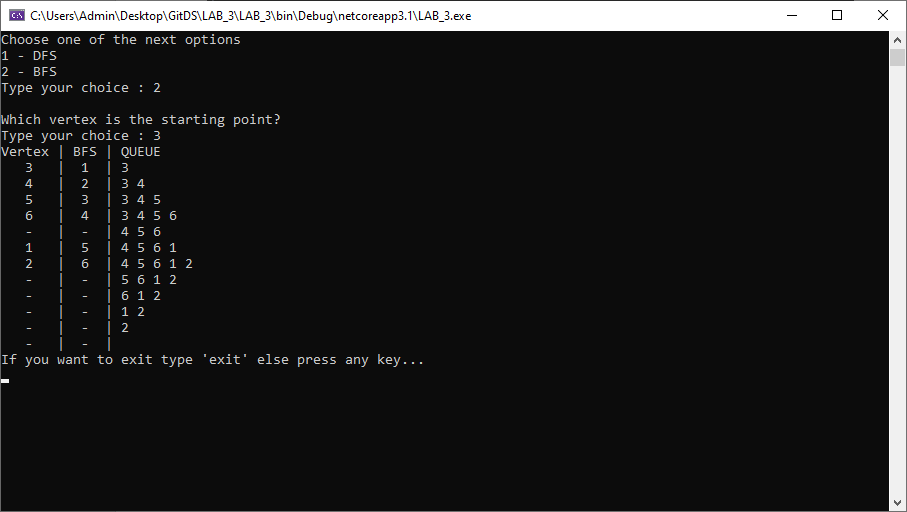
Київ 2020

2. Умова лабораторної роботи

3. Програмний код(C#) :

using System.Collections.Generic;  
using System.IO;  
  
namespace LAB\_3  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 Parser parser = new Parser("input.txt");  
 (int n, int m, List<(int, int)> edgesList) = parser.GetInput();  
 NotDirectedGraph graph = new NotDirectedGraph(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.StartMenu();  
 }  
 else  
 {  
 System.Console.Write("Type the name of output file : ");  
 string fileName = System.Console.ReadLine();  
 GraphAlgorithm graphAlg = new GraphAlgorithm(graph, "file", fileName);  
 graphAlg.StartMenu();  
 }  
 }  
 }  
  
 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 = System.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 Stack  
 {  
 private List<int> stack;  
  
 public Stack() => stack = new List<int>();  
  
 public int Pop()  
 {  
 int toReturn = stack[stack.Count - 1];  
 stack.RemoveAt(stack.Count - 1);  
 return toReturn;  
 }  
  
 public void Push(int item) => stack.Add(item);  
  
 public int Peek() => stack[stack.Count - 1];  
  
 public int Count() => stack.Count;  
  
 public List<int> ToList() => stack;  
 }  
  
 class Queue  
 {  
 private readonly List<int> queue;  
  
 public Queue() => queue = new List<int>();  
  
 public void Enqueue(int item)  
 {  
 queue.Add(item);  
 }  
  
 public int Dequeue()  
 {  
 int toReturn = queue[0];  
 queue.RemoveAt(0);  
 return toReturn;  
 }  
  
 public int Peek() => queue[0];  
  
 public int Count() => queue.Count;  
  
 public List<int> ToList() => queue;  
 }  
  
 class NotDirectedGraph  
 {  
 public int verticesNum { get; }  
 private readonly int edgesNum;  
 private readonly List<(int, int)> edgesList;  
  
 public NotDirectedGraph(int n, int m, List<(int, int)> edgesArray)  
 {  
 verticesNum = (n >= 0) ? n : 0;  
 edgesNum = (m >= 0) ? m : 0;  
 edgesList = edgesArray;  
 }  
  
 public NotDirectedGraph(NotDirectedGraph 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);  
 }  
 if (!adjList[finish].Contains(start))  
 {  
 adjList[finish].Add(start);  
 }  
 }  
 foreach (var key in adjList.Keys)  
 {  
 adjList[key].Sort((a, b) => a.CompareTo(b));  
 }  
 return adjList;  
 }  
 }  
  
 class GraphAlgorithm  
 {  
 private readonly Writer writer;  
 private readonly NotDirectedGraph graph;  
  
 public GraphAlgorithm(NotDirectedGraph graph, string writeMode, string fileName = null)  
 {  
 this.graph = new NotDirectedGraph(graph);  
 writer = new Writer(writeMode, fileName);  
 }  
  
 private void ShowMenu()  
 {  
 System.Console.WriteLine("Choose one of the next options\n" +  
 "1 - DFS\n" +  
 "2 - BFS");  
 }  
  
 private int EnterNumber(int up)  
 {  
 int number = 0;  
  
 while (number > up || number < 1)  
 {  
 System.Console.Write("Type your choice : ");  
 \_ = int.TryParse(System.Console.ReadLine(), out number);  
 if (number > up || number < 1)  
 System.Console.WriteLine("Invalid input!");  
 }  
 return number;  
 }  
  
 private void WorkWithNumber(int number)  
 {  
 switch (number)  
 {  
 case 1:  
 System.Console.WriteLine("\nWhich vertex is the starting point?");  
 DFS(EnterNumber(graph.verticesNum));  
 break;  
 case 2:  
 System.Console.WriteLine("\nWhich vertex is the starting point?");  
 BFS(EnterNumber(graph.verticesNum));  
 break;  
 }  
 }  
  
 public void StartMenu()  
 {  
 string answer = "";  
 while (answer != "exit")  
 {  
 ShowMenu();  
 WorkWithNumber(EnterNumber(2));  
 System.Console.WriteLine("If you want to exit type 'exit' else press any key...");  
 answer = System.Console.ReadLine();  
 writer.WriteLine();  
 }  
 writer.Close();  
 }  
  
 private void ShowTable(int vertex, int bfsNum)  
 {  
 writer.Write($" {vertex + 1} | {bfsNum} | ");  
 }  
  
 private void PrintList(List<int> toPrint)  
 {  
 foreach (var value in toPrint)  
 {  
 writer.Write($"{value + 1} ");  
 }  
 writer.WriteLine();  
 }  
  
 private void BFS(int start)  
 {  
 Dictionary<int, List<int>> adjList = graph.GetAdjList();  
 Queue queue = new Queue();  
 HashSet<int> visited = new HashSet<int>();  
 int k = 0;  
 int humanIndex = 1;  
 start -= humanIndex;  
  
 writer.WriteLine("Vertex | BFS | QUEUE");  
  
 queue.Enqueue(start);  
 visited.Add(start);  
 ShowTable(start, ++k);  
 PrintList(queue.ToList());  
  
 while (queue.Count() != 0)  
 {  
 start = queue.Peek();  
  
 foreach (var vertex in adjList[start])  
 {  
 if (!visited.Contains(vertex))  
 {  
 queue.Enqueue(vertex);  
 visited.Add(vertex);  
 ShowTable(vertex, ++k);  
 PrintList(queue.ToList());  
 }  
 }  
 start = queue.Dequeue();  
 writer.Write($" - | - | ");  
 PrintList(queue.ToList());  
  
 }  
 }  
  
 private void DFS(int start)  
 {  
 Dictionary<int, List<int>> adjList = graph.GetAdjList();  
 Stack stack = new Stack();  
 HashSet<int> visited = new HashSet<int>();  
 int k = 0;  
 int humanIndex = 1;  
 start -= humanIndex;  
  
 writer.WriteLine("Vertex | DFS | Stack");  
  
 stack.Push(start);  
 visited.Add(start);  
 ShowTable(start, ++k);  
 PrintList(stack.ToList());  
  
 DoWork(start);  
  
 writer.WriteLine(" - | - | ");  
  
 void DoWork(int start)  
 {  
 foreach (var vertex in adjList[start])  
 {  
 if (!visited.Contains(vertex))  
 {  
 stack.Push(vertex);  
 visited.Add(vertex);  
 ShowTable(vertex, ++k);  
 PrintList(stack.ToList());  
 DoWork(vertex);  
  
 if (stack.Count() == 0) return;  
 int stackHead = stack.Pop();  
 writer.Write(" - | - | ");  
 PrintList(stack.ToList());  
 DoWork(stackHead);  
 }  
 }  
  
 }  
 }  
 }  
}

4. Результати виконання :



**Вхідні дані :**

**6** **10**

1 4

2 4

2 6

3 4

3 5

3 6

4 1

4 2

4 3

5 6