Отчет по 4 лабораторной студента группы ПЗПИ-17-2 Вирченко Марка

using System;

using System.Collections.Generic;

namespace ConsoleApplication3

{

public class Node

{

public int Data;

public List<GraphNode> neighbors = null;

public Node(int Data)

{

this.Data = Data;

}

public Node(int Data, List<GraphNode> neighbors)

{

this.Data = Data;

this.neighbors = neighbors;

}

public Node() { }

}

public class GraphNode : Node

{

public List<int> costs;

public GraphNode() : base() { }

public GraphNode(int Data) : base(Data) { }

public GraphNode(int Data, List<GraphNode> neighbors) : base(Data, neighbors) { }

public List<GraphNode> \_neighbors;

new public List<GraphNode> neighbors

{

get

{

if (\_neighbors != null)

return \_neighbors;

else

\_neighbors = new List<GraphNode>();

return \_neighbors;

}

}

public List<int> Costs

{

get

{

if (costs == null)

costs = new List<int>();

return costs;

}

set

{

costs = value;

}

}

}

class Graph

{

private List<GraphNode> nodeSet = new List<GraphNode>();

public Graph() { }

public Graph(List<GraphNode> nodeSet)

{

if (nodeSet == null)

this.nodeSet = new List<GraphNode>();

else

this.nodeSet = nodeSet;

}

public void AddNode(int value)

{

nodeSet.Add(new GraphNode(value));

}

public void AddNode(GraphNode node)

{

nodeSet.Add(node);

}

public void AddDirectedEdge(GraphNode from, GraphNode to, int cost = 100000)

{

from.neighbors.Add(to);

from.Costs.Add(cost);

}

public void AddUndirectedEdge(GraphNode from, GraphNode to, int cost = 10000)

{

from.neighbors.Add(to);

to.neighbors.Add(from);

from.Costs.Add(cost);

to.Costs.Add(cost);

}

public bool Remove(int value)

{

GraphNode A = new GraphNode(value);

foreach (var i in nodeSet)

{

if (i.Equals(A))

{

nodeSet.Remove(i);

return true;

}

}

return false;

}

public List<GraphNode> DFS(GraphNode start)

{

List<GraphNode> visited = new List<GraphNode>();

if (!this.nodeSet.Contains(start))

return visited;

var stack = new Stack<GraphNode>();

stack.Push(start);

while (stack.Count > 0)

{

var vertex = stack.Pop();

if (visited.Contains(vertex))

continue;

visited.Add(vertex);

foreach (var neighbor in vertex.neighbors)

{

if (!visited.Contains(neighbor))

{

stack.Push(neighbor);

}

}

}

return visited;

}

public GraphNode BFS(GraphNode root, GraphNode end)

{

Queue<GraphNode> Q = new Queue<GraphNode>();

HashSet<GraphNode> S = new HashSet<GraphNode>();

Q.Enqueue(root);

S.Add(root);

while (Q.Count > 0)

{

GraphNode e = Q.Dequeue();

if (e.Equals(end))

return e;

foreach (GraphNode node in e.neighbors)

{

if (!S.Contains(node))

{

Q.Enqueue(node);

S.Add(node);

}

}

}

return null;

}

}

class Program

{

static void Main(string[] args)

{

Graph graph = new ConsoleApplication3.Graph();

GraphNode five = new GraphNode(5);

GraphNode six = new GraphNode(6);

graph.AddNode(five);

graph.AddNode(six);

graph.AddDirectedEdge(five, six);

List<GraphNode> list = graph.DFS(five);

foreach (var i in list)

Console.Write(i.Data + " ");

Console.Write("\n" + graph.BFS(five, six).Data);

Console.ReadKey();

}

}

}