МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра автоматизированных систем управления

Лабораторная работа № 7

**«Графы»**

Выполнил:

студент гр.935М

Копцев Г.А.

Принял:

доцент, к.т.н

Челебаев С.В.

Рязань 2019

**Цель работы.** Изучение графовых алгоритмов по нахождению путей.

**Выполнение работы**

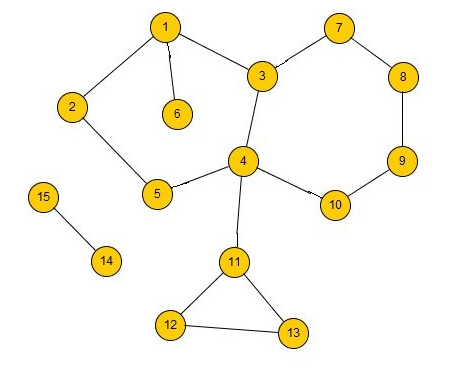
****

Рисунок 1 – Граф

Соответствие вершин городам:

01 - "Анк\_Морпорк"

02 - "Арглтон"

03 - "Арканар"

04 - "Аттилан"

05 - "Восторг"

06 - "Глупов"

07 - "Город\_грехов"

08 - "Город\_под\_куполом"

09 - "Каркоза"

10 - "Касл\_Рок"

11 - "Лаленбург"

12 - "Лапута"

13 - "Либерти\_Сити"

14 - "Вайс\_Сити"

15 - "Сайлент\_Хилл"

**Исходный код:**

using System;

using System.Collections.Generic;

using System.Linq;

namespace ISAT6

{

class Program

{

static Node n01 = new Node("Анк\_Морпорк");

static Node n02 = new Node("Арглтон");

static Node n03 = new Node("Арканар");

static Node n04 = new Node("Аттилан");

static Node n05 = new Node("Восторг");

static Node n06 = new Node("Глупов");

static Node n07 = new Node("Город\_грехов");

static Node n08 = new Node("Город\_под\_куполом");

static Node n09 = new Node("Каркоза");

static Node n10 = new Node("Касл\_Рок");

static Node n11 = new Node("Лаленбург");

static Node n12 = new Node("Лапута");

static Node n13 = new Node("Либерти\_Сити");

static Node n14 = new Node("Вайс\_Сити");

static Node n15 = new Node("Сайлент\_Хилл");

static List<Node> NodeList = new List<Node>

{

n01, n02, n03, n04, n05, n06, n07, n08, n09, n10, n11, n12, n13, n14, n15

};

static List<Buffer> Nodebuffer = new List<Buffer>();

static void Main()

{

Console.Clear();

n01.AddChildren(n02, 150).AddChildren(n03, 100);

n02.AddChildren(n05, 160);

n03.AddChildren(n04, 120);

n04.AddChildren(n05, 90).AddChildren(n10, 95).AddChildren(n11, 100);

n06.AddChildren(n01, 50);

n07.AddChildren(n03, 75).AddChildren(n08, 70);

n09.AddChildren(n08, 56).AddChildren(n10, 23);

n11.AddChildren(n12, 234).AddChildren(n13, 435);

n12.AddChildren(n13, 45);

n14.AddChildren(n15, 34);

PrintAllShortestPathes(1, NodeList);

Console.WriteLine("-----------------");

FindShortestPath(n07, n11);

Console.WriteLine("-----------------");

AllShortestPathForOneNode(n06);

Console.WriteLine("-----------------");

AllUnattainabletPathForOneNode(n06);

Console.WriteLine("-----------------");

PrintAllShortestPathes(1, NodeList);

Console.WriteLine("-----------------");

Console.Read();

}

private static void FindShortestPath(Node start, Node target)

{

PrintPath(0, start, target);

}

private static void AllShortestPathForOneNode(Node start)

{

Console.WriteLine($"Все достижимые вершины для города {start.Name}");

var nodeList = new List<Node> { start };

PrintAllShortestPathes(1, nodeList, true);

}

private static void AllUnattainabletPathForOneNode(Node start)

{

Console.WriteLine($"Все недостижимые вершины для города {start.Name}");

var nodeList = new List<Node> { start };

PrintAllShortestPathes(2, nodeList, true, true);

}

private static void PrintAllShortestPathes(int flag, List<Node> nodeList, bool onlyResult = false, bool onlyResultForOneNode = false)

{

var buffer = new List<string>();

foreach (var start in nodeList)

{

foreach (var target in NodeList)

{

if (start.Name == target.Name || buffer.Contains(target.Name)) continue;

var nodeBuff = Nodebuffer.FirstOrDefault(n => n.StartCity == start.Name && n.EndCity == target.Name);

if (nodeBuff != null)

{

if (!onlyResultForOneNode) Console.Write($"{start.Name}-{target.Name}: ");

if (!onlyResult && !onlyResultForOneNode) Console.Write(string.Join(" -> ", nodeBuff.Path.Select(x => x.Name)) + " Расстояние: " + nodeBuff.Distance);

else if (onlyResult && !onlyResultForOneNode) Console.Write("Путь присутствует");

if (!onlyResultForOneNode) Console.WriteLine();

continue;

}

PrintPath(flag, start, target, onlyResult);

}

buffer.Add(start.Name);

}

}

private static void PrintPath(int flag, Node start, Node target, bool onlyResult = false)

{

var path = new DepthFirstSearch().IDDFS(start, target);

if (path.Count == 0 && (flag == 2 || flag == 0))

{

Console.Write($"{start.Name} - {target.Name}: ");

Console.Write("Путь не найден!");

Console.WriteLine();

}

else

{

int dist = 0;

foreach (var p in path)

{

dist += p.Distance;

}

if ((flag == 0 || flag == 1) && dist != 0)

{

Console.Write($"{start.Name}-{target.Name}: ");

if (!onlyResult) Console.Write(string.Join(" -> ", path.Select(x => x.Name)) + " Расстояние: " + dist);

else Console.Write("Путь присутствует");

Console.WriteLine();

Nodebuffer.Add(new Buffer(start.Name, target.Name, path, dist));

}

}

}

}

class Node

{

/// Имя вершины.

public string Name { get; }

/// Список соседних вершин.

public List<Node> Children { get; }

public int Distance { get; set; }

public Node(string name)

{

Name = name;

Children = new List<Node>();

}

/// Добавляет новую соседнюю вершину.

public Node AddChildren(Node node, int distance, bool bidirect = true)

{

node.Distance = distance;

Children.Add(node);

if (bidirect)

{

node.Children.Add(this);

}

return this;

}

}

class Buffer

{

public string StartCity { get; set; }

public string EndCity { get; set; }

public LinkedList<Node> Path { get; set; }

public int Distance { get; set; }

public Buffer(string \_startCity, string \_endCity, LinkedList<Node> \_path, int \_distance)

{

StartCity = \_startCity;

EndCity = \_endCity;

Path = \_path;

Distance = \_distance;

}

}

class DepthFirstSearch

{

// Список посещенных вершин

private HashSet<Node> visited;

// Путь из начальной вершины в целевую.

private LinkedList<Node> path;

private Node goal;

// Был ли поиск завершен из-за того, что достиг предела глубины.

private bool limitWasReached;

/// <summary>

/// Первый попавшийся путь поиском в глубину

/// </summary>

/// <param name="start"></param>

/// <param name="\_goal"></param>

/// <returns></returns>

public LinkedList<Node> DFS(Node start, Node \_goal)

{

visited = new HashSet<Node>();

path = new LinkedList<Node>();

goal = \_goal;

DFS(start);

if (path.Count > 0)

{

path.AddFirst(start);

}

return path;

}

private bool DFS(Node node)

{

if (node == goal)

{

return true;

}

visited.Add(node);

foreach (var child in node.Children.Where(x => !visited.Contains(x)))

{

if (DFS(child))

{

path.AddFirst(child);

return true;

}

}

return false;

}

/// <summary>

/// Первый попавшийся путь поиском в глубину, имеющий не более заданного количества пересадок

/// </summary>

/// <param name="start"></param>

/// <param name="goal"></param>

/// <param name="limit"></param>

/// <returns></returns>

public LinkedList<Node> DLS(Node start, Node goal, int limit)

{

visited = new HashSet<Node>();

path = new LinkedList<Node>();

limitWasReached = true;

this.goal = goal;

DLS(start, limit);

if (path.Count > 0)

{

path.AddFirst(start);

}

return path;

}

private bool DLS(Node node, int limit)

{

if (node == goal)

{

return true;

}

if (limit == 0)

{

limitWasReached = false;

return false;

}

visited.Add(node);

foreach (var child in node.Children.Where(x => !visited.Contains(x)))

{

if (DLS(child, limit - 1))

{

path.AddFirst(child);

return true;

}

}

return false;

}

/// <summary>

/// Кратчайший путь поиском в глубину

/// </summary>

/// <param name="start"></param>

/// <param name="goal"></param>

/// <returns></returns>

public LinkedList<Node> IDDFS(Node start, Node goal)

{

for (int limit = 1; ; limit++)

{

var result = DLS(start, goal, limit);

if (result.Count > 0 || limitWasReached)

{

return result;

}

}

}

}

}

**Задача 4.** Реализовать отношение достижимости на графе.

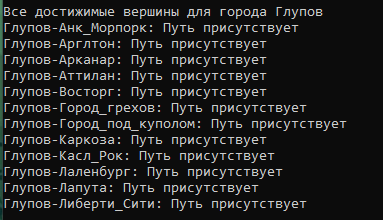
****

Рисунок 2 – Все достижимые вершина города Глупов

**Задача 5.** Найти кратчайшие пути при помощи поиска в глубину.

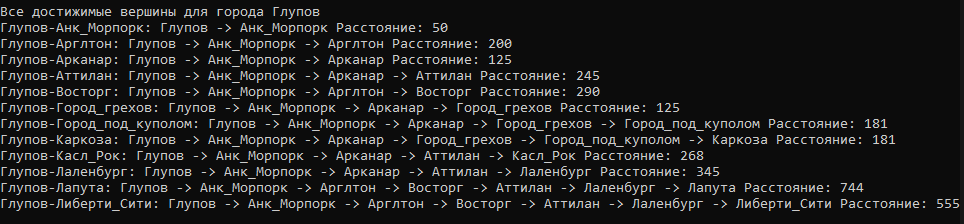


Рисунок 3 – Все достижимые вершина города Глупов подробно

**Задача 6.** Найти вершины, недостижимые из заданной вершины.

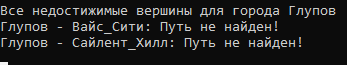


Рисунок 4 – Все недостижимые вершина города Глупов

**Задача 7.** Выполнить эвристический поиск кратчайших путей

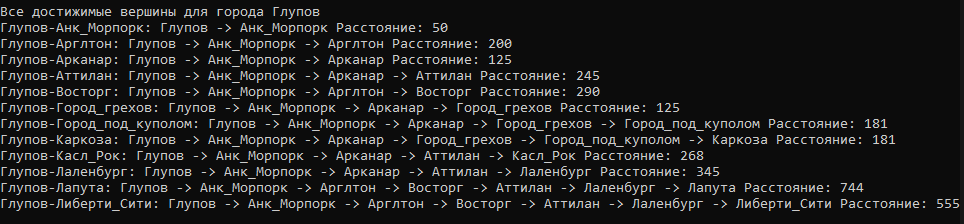


Рисунок 5 – Результат эвристического поиска

Ссылка на удаленный репозиторий:

<https://github.com/ImVeriBed/ISAT6>