**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра информатики и прикладной математики**

Алгоритмы и структуры данных

Лабораторная работа №5

Вариант 3

Выполнил: Гхази Даниэль

Группа P3218

Преподаватель: Зинчик А.А.

2017 г.

**Текст задания**

Имеется N городов, которые пронумерованы от 1 до N (где N - натуральное, 1<N<=100). Некоторые из них соединены двухсторонними дорогами, которые пересекаются только в городах. Имеется два типа дорог - шоссейные и железные. Для каждой дороги известна базовая стоимость проезда по ней.   
Необходимо проехать из города А в город В, уплатив минимальную сумму за проезд. Стоимость проезда зависит от набора проезжаемых дорог и от способа проезда. Так, если вы подъехали к городу С по шоссейной (железной) дороге X->C и хотите ехать дальше по дороге C->Y того же типа, то вы должны уплатить только базовую стоимость проезда по дороге C->Y. Если тип дороги C->Y отличен от типа дороги Х->C, то вы должны уплатить базовую стоимость проезда по дороге C->Y плюс 10% от базовой стоимости проезда по этой дороге (страховой взнос). При выезде из города А страховой взнос платится всегда. Написать программу, которая находит самый дешевый маршрут проезда в виде последовательности городов и вычисляет стоимость проезда по этому маршруту.   
Спецификация входных данных   
Входные данные находятся в текстовом файле с именем TOUR.IN и имеют следующий формат:   
- в первой строке находятся число N   
- во второй строке - число M (количество дорог, натуральное M<=1000);   
- в каждой из следующих M строк находятся 4 числа x, y, t, p, разделенные пробелом, где x и y - номера городов, которые соединяет дорога, t - тип дороги (0 - шоссейная, 1 - железная), p - базовая стоимость проезда по ней (p - вещественное, 0<p<=1000).   
- в последней строке задается номера начального и конечного городов A и B.   
Спецификация выходных данных   
Выходные данные должны быть записаны в текстовый файл с именем TOUR.OUT и иметь следующий формат:   
- в первой строке находится число S - стоимость проезда по самому дешевому маршруту, с точностью 2 знака после запятой;   
- в каждой из последующих строк, кроме последней, находится два числа - номер очередного города в маршруте (начиная с города А) и тип дороги, по которой он выезжает из этого города (0 или 1), разделенные пробелом.   
- в последней строке находится единственное число - номер города B.

**Текст программы**

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System;

namespace CheapestPath

{

class ShortestPathFinder

{

/// <summary>

/// Cost of travel from start city to another city (index = city name).

/// </summary>

private float[] cost;

/// <summary>

/// Priority queue to extract shortest road to adjacent city.

/// </summary>

private List<ElementOfPriorityQueue> priorityQueue;

/// <summary>

/// List of visited cities. Adds a city if min cost of travel to it was already found.

/// </summary>

private List<ElementOfPath> parentCities;

private class ElementOfPriorityQueue : IComparable<ElementOfPriorityQueue>

{

internal int VertexName { get; set; }

internal float Cost { get; set; }

internal ElementOfPriorityQueue(int vertexName, float cost)

{

VertexName = vertexName;

Cost = cost;

}

public int CompareTo(ElementOfPriorityQueue other)

{

if (other == null) throw new NullReferenceException();

return other.Cost.CompareTo(this.Cost);

}

}

public ShortestPathFinder(int numberOfCities)

{

cost = new float[numberOfCities + 1];

priorityQueue = new List<ElementOfPriorityQueue>(numberOfCities + 1);

// Adding element to unused zero-indexed slot.

priorityQueue.Add(new ElementOfPriorityQueue(0, float.MaxValue));

parentCities = new List<ElementOfPath>(numberOfCities + 1);

// Adding element to unused zero-indexed slot.

parentCities.Add(new ElementOfPath());

}

/// <summary>

/// Finds shortest path using Dijkstra's algorithm.

/// </summary>

/// <param name="input"> User-inputed data. </param>

/// <returns> List of cities in shortest path. </returns>

public void FindShortestPath(Input input, Output output)

{

ReadOnlyCollection< List<Road> > roads = input.ReadOnlyRoads;

Initialization(input);

DijkstraAlgorithm(roads, input.StartCity);

output.cheapestTraveCost = cost[cost.Length - 1];

output.SetShortestPath(FindPath());

}

/// <summary>

/// Initializing costs array and queue.

/// </summary>

private void Initialization(Input input)

{

cost[input.StartCity] = 0;

for (int city = 1; city <= input.NumberOfCities; city++)

{

if (city != input.StartCity) cost[city] = float.MaxValue;

priorityQueue.Add(new ElementOfPriorityQueue(city, cost[city]));

parentCities.Add(new ElementOfPath());

}

}

private int FindIndexOfRoadToElement(int roadTo)

{

for (int i = 1; i < priorityQueue.Count; i++)

{

if (priorityQueue[i].VertexName == roadTo) return i;

}

return 0;

}

/// <summary>

/// Finds shortest path in graph using Dijkstra's algorithm

/// </summary>

/// <param name="roads"> Edges of graph. </param>

private void DijkstraAlgorithm(ReadOnlyCollection<List<Road>> roads, int startCity)

{

while (priorityQueue.Count > 0)

{

priorityQueue.Sort();

int fromCity = priorityQueue[priorityQueue.Count - 1].VertexName;

priorityQueue.RemoveAt(priorityQueue.Count - 1);

foreach (Road road in roads[fromCity])

{

float costOfTravelWithInsurance = road.CostOfTravel;

bool roadChanged = RoadTypeChanged(road, startCity);

if (roadChanged) costOfTravelWithInsurance \*= Constants.insuranceCoefficient;

if (cost[fromCity] + costOfTravelWithInsurance < cost[road.To])

{

int indexOfRoadToElement = FindIndexOfRoadToElement(road.To);

cost[road.To] = cost[fromCity] + costOfTravelWithInsurance;

priorityQueue[indexOfRoadToElement].Cost = cost[road.To];

ElementOfPath newCity = new ElementOfPath { ParentCity = fromCity, CameOnSuchRoad = road.RoadType };

parentCities[road.To] = newCity;

}

}

}

}

private bool RoadTypeChanged(Road road, int startCity)

{

if (road.From == startCity) return true;

if (road.RoadType != parentCities[road.From].CameOnSuchRoad) return true;

else return false;

}

/// <summary>

/// Returns cities of shortest path.

/// </summary>

private List<ElementOfPath> FindPath()

{

int parentCity = parentCities[parentCities.Count - 1].ParentCity;

List<ElementOfPath> citiesInPath = new List<ElementOfPath>();

// Adding last city, which is the same as last index of parentCities.

citiesInPath.Add(parentCities[parentCities.Count - 1]);

while (parentCity != 1)

{

citiesInPath.Add(parentCities[parentCity]);

parentCity = parentCities[parentCity].ParentCity;

}

citiesInPath.Reverse();

return citiesInPath;

}

}

}

**Вывод**

В результате выполнения лабораторной работы была решена задача о поиске наиболее дешевого пути между двумя заданными городами

Для решения данной задачи был использован алгоритм Дейкстры. Основную часть программы (С#) составляют три класса: Input, FindShortestPath, Output. Названия говорящие.

Было интересно использовать алгоритм Дейстры не для обычного поиска кратчайшего пути. Пришлось поразмыслить над необходимыми модификациями, чтобы был возможен учет страховке при смене типа дороги.