19l-1316

dsa project code:

#include <iostream>

#include <string>

#include <cstring>

#include <fstream>

#include <stdio.h>

#include <limits.h>

#include <stack>

using namespace std;

class route

{

private:

double distance;

double cost;

public:

route()

{

distance = 0;

cost = 0;

}

void setdistance(double d)

{

distance = d;

}

void setcost(double c)

{

cost = c;

}

double getdistance()

{

return distance;

}

double getcost()

{

return cost;

}

};

class Graph

{

private:

route\*\* adj\_matrix;

int numVertices;

double\* MinDist;

public:

//part1: constructor initializes adjacency matrix

Graph(int numVertex)

{

numVertices = numVertex;

adj\_matrix = new route\*[numVertices];

for (int i = 0; i<numVertices; i++)

{

adj\_matrix[i] = new route[numVertices];

}

}

//part2: returns the number of vertices in the graph

int GetNumVertices() { return numVertices; }

void insertEdge(int frmVertex, int toVertex, double d, double c)

{

adj\_matrix[frmVertex][toVertex].setdistance(d);

adj\_matrix[frmVertex][toVertex].setcost(d);

}

void printGraph()

{

cout<<"Distance Adjacency Matrix Representation \n";

for(int i=0;i<numVertices;i++)

{

for(int j=0;j<numVertices;j++)

{

cout<<adj\_matrix[i][j].getdistance()<<"\t";

}

cout<<endl;

}

cout<<endl<<endl;

cout<<"Cost Adjacency Matrix Representation \n";

for(int i=0;i<numVertices;i++)

{

for(int j=0;j<numVertices;j++)

{

cout<<adj\_matrix[i][j].getcost()<<"\t";

}

cout<<endl;

}

}

int minDistance(int dist[], bool minSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

for (int v = 0; v < numVertices; v++)

if (minSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void Dijkstra(int loc, int opt, city\* cit)

{

int\* dist;

dist = new int[numVertices];

bool\* minSet;

minSet = new bool[numVertices];

for (int i = 0; i < numVertices; i++)

dist[i] = INT\_MAX, minSet[i] = false;

dist[loc] = 0;

for (int count = 0; count < numVertices - 1; count++)

{

int u = minDistance(dist, minSet);

minSet[u] = true;

switch (opt)

{

case 0: //For Distance

for (int v = 0; v < numVertices; v++)

if (!minSet[v] && adj\_matrix[u][v].getdistance() && dist[u] != INT\_MAX && dist[u] + adj\_matrix[u][v].getdistance() < dist[v])

dist[v] = dist[u] + adj\_matrix[u][v].getdistance();

cout << "City \t Distance from" << cit[loc].getcname() << endl;

break;

case 1: //For MinimumHops

for (int v = 0; v < numVertices; v++)

if (!minSet[v] && bool(adj\_matrix[u][v].getdistance) && dist[u] != INT\_MAX && dist[u] + bool(adj\_matrix[u][v].getdistance) < dist[v])

dist[v] = dist[u] + bool(adj\_matrix[u][v].getdistance());

cout << "City \t Hops from" << cit[loc].getcname() << endl;

for(int k=0;k<numVertices;k++)

MinDist[k]=dist[k];

break;

default:

cout << "Error: Invalid Option entry." << endl;

return;

}

for (int i = 0; i < numVertices; i++)

cout << cit[i].getcname() << "\t" << dist[i] << endl;

}

}

double\* getMinDist(){return MinDist;}

};

int main()

{

ifstream fin;

fin.open("city data.txt");

ofstream fout;

fout.open("city data.txt");

int nv, dist, cost, f, t;

string rname,ra,sname,sa,;

fin >> nv;

GraphMap(nv);

city\* cit;

cit = new city[nv];

int i = 0;

while (!EOF)

{

fin >> rname;

fin >> ra;

fin >> sname;

fin >> sa;

fin >> dist;

fin >> cost;

}

fin.close();

cout << "Shortest Path Tree:";

for (int i = 0; i < nv; i++)

{

Map.Dijkstra(i, 0, cit);

}

cout << "Minimum Hops From every city to the other";

{

Map.Dijkstra(i, 1, cit);

}

system("pause");

fin.open("booking courier.txt");

int lsize, UID,index=0;

fin >> lsize;

request \*arr=new request[lsize];

string Prem, trash;

while (!EOF)

{

fin >> UID;

fin >> CF;

fin >> CT;

fin >> Prem;

fin >> trash;

arr[index].setUserID(UID);

arr[index].setCFrom(CF);

arr[index].setCTo(CT);

arr[index].setPUser(Prem);

index++;

}

fin.close();

return 0;

}