#include <iostream>

#include <queue>

#include <stack>

#include <fstream>

#include <string>

#include <cstring>

using namespace std;

#ifndef GRAPH\_H

#define GRAPH\_H

class graph

{

private:

double \*\*distance\_matrix;

double \*\*cost\_matrix;

int \*\*link\_matrix;

string \*cities;

double \*cars;

string \*\*path;

double \*\*path\_num;

int size;

int inc = 0;

int hashsize = 0;

public:

graph()

{

}

graph(int s, string x)

{

size = s;

////////////////////////////////////////// DISTANCE MATRIX //////////////////////////////////////////////

distance\_matrix = new double\*[size];

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

distance\_matrix[i] = new double[size];

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

{

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

{

distance\_matrix[i][j] = 0;

}

}

////////////////////////////////////////// COST MATRIX //////////////////////////////////////////////

cost\_matrix = new double\*[size];

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

cost\_matrix[i] = new double[size];

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

{

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

{

cost\_matrix[i][j] = 0;

}

}

/////////////////////////////////////////// LINK MATRIX //////////////////////////////////////////////

link\_matrix = new int\*[size];

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

link\_matrix[i] = new int[size];

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

{

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

{

link\_matrix[i][j] = 0;

}

}

////////////////////////////////////////// CITY MATRIX //////////////////////////////////////////////

cities = new string[size];

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

cities[i] = " ";

///////////////////////////////////////// CARS MATRIX //////////////////////////////////////////////

cars = new double[size];

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

{

cars[i] = 0;

}

Take\_input(x);

path\_num = new double\*[hashsize];

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

path\_num[i] = new double[hashsize];

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

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

path\_num[i][j] = 0;

path = new string\*[hashsize];

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

path[i] = new string[hashsize];

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

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

path[i][j] = "";

Initializer();

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool check(int count, string x)

{

bool check = false;

count = count + 1;

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

{

if (cities[i] == x)

{

return false;

break;

}

else

check = true;

}

return check;

}

void assigning\_numbers(int &count, int size, string a, string b)

{

bool check\_a = false;

bool check\_b = false;

check\_a = check(count, a);

if (check\_a == true)

{

cities[count] = a;

count++;

}

check\_b = check(count, b);

if (check\_b == true)

{

cities[count] = b;

count++;

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void print\_cities()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CITIES HAVE BEEN ASSIGNED NUMBERS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

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

cout << cities[i] << "= City " << i << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void Take\_input(string filename)

{

ifstream inn;

inn.open(filename);

ofstream out;

out.open("tempfile2.txt");

string first, second;

double distance, cost;

inn >> size;

while (!inn.eof())

{

hashsize++;

inn >> first >> second >> distance >> cost;

assigning\_numbers(inc, size, first, second);

out << first << " " << second << " " << distance << " " << cost << endl;

}

cout << endl;

inn.close();

out.close();

assigning\_values();

}

void Display\_file\_data(string filename)

{

ifstream in;

in.open(filename);

string first, second;

double distance, cost;

in >> size;

while (!in.eof())

{

in >> first >> second >> distance >> cost;

cout << "(" << first << "," << second << ") " << "distance " << distance << " km and cost/passenger is " << cost << " for bus service" << endl;

}

cout << endl;

in.close();

}

//////////////////////////////////////////////////////// DATA ASSIGNING /////////////////////////////////////////////////

void addvertex(int a, int b, double distance, double cost)

{

cost\_matrix[a][b] = cost;

cost\_matrix[b][a] = cost;

distance\_matrix[a][b] = distance;

distance\_matrix[b][a] = distance;

link\_matrix[a][b] = 1;

link\_matrix[b][a] = 1;

}

int set\_matrix\_enteries(string x, int &a, int size)

{

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

{

if (cities[i] == x)

{

a = i;

break;

}

}

return 0;

}

void assigning\_values()

{

int a = 0; int b = 0;

ifstream inn;

inn.open("tempfile2.txt");

string first, second;

double distance, cost;

while (!inn.eof())

{

inn >> first >> second >> distance >> cost;

set\_matrix\_enteries(first, a, size);

set\_matrix\_enteries(second, b, size);

addvertex(a, b, distance, cost);

}

}

//////////////////////////////////////////////// DISPLAY MATRICES ////////////////////////////////////////////////////////

template<class DT>

void display\_2p\_matrix(DT \*arr[], int size)

{

cout << " ";

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

cout << i << "|" << " ";

cout << endl;

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

{

cout << i << "|" << " ";

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

{

cout << arr[i][j] << " ";

}

cout << endl;

}

}

int give\_index(string city, int &x)

{

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

{

if (cities[i] == city)

x = i;

}

return x;

}

////////////////////////////////////////// SHORTEST DISTANCE BETWEEN POINTS //////////////////////////////////////////

int find\_city( string x, int &num)

{

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

{

if (cities[i] == x)

{

num = i;

break;

}

}

return num;

}

int findminvertex(double \*distance, bool \*visited, int n)

{

int minvertex = -1;

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

{

if (!visited[i] && (minvertex == -1 || distance[i] < distance[minvertex]))

minvertex = i;

}

return minvertex;

}

bool check\_cars(string x)

{

int num=0;

find\_city(x,num);

cars[num]++;

if (cars[num] > 5)

return false;

return true;

}

int printpath(int parent[], int j,double dist[], int x,string &spath)

{

if (parent[j] == -1)

return 0;

printpath(parent, parent[j],dist, x,spath);

spath = spath + cities[j]+"->";

return j;

}

int printsolution(double dist[], int parent[],int x)

{

string temp = cities[x] + "->";

string spath;

spath = temp;

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

{

printpath(parent, i,dist,x,spath);

path[x][i] = spath;

cout << spath;

path\_num[x][i] = dist[i];

cout << path\_num[x][i] << "km" << endl;

spath = temp;

}

return 0;

}

void Initializer()

{

clear\_memory();

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

ShortestPath(distance\_matrix, i, "0");

}

void User\_choice(string choice)

{

int runtimep = 0;

int runtimem = 0;

cout << endl;

if (choice == "3")

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Shortest Paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

if (runtimep < 1)

{

clear\_memory();

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

ShortestPath(distance\_matrix, i, choice);

runtimep++;

}

else

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ALL SHORTEST PATHS ALREADY DISPLAYED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

else if (choice == "4")

{

if (runtimem < 1)

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Minimum Number of Hops \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

clear\_memory();

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

ShortestPath(link\_matrix, i, choice);

runtimem++;

}

else

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ALL MINIMUM HOPS ALREADY DISPLAYED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

else if (choice == "6")

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Shortest Path With Respect To Cost \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

clear\_memory();

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

ShortestPath(cost\_matrix, i, choice);

runtimem++;

}

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

template<class DT>

void ShortestPath(DT \*\*arr,int x,string choice)

{

double \*distance = new double[size];

bool \*visited = new bool[size];

int \*parent = new int[size];

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

{

distance[i] = INT\_MAX;

visited[i] = false;

}

distance[x] = 0;

parent[x] = -1;

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

{

int minvertex = findminvertex(distance, visited, size);

visited[minvertex] = true;

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

{

if (arr[minvertex][j] != 0 && !visited[j])

{

double dist = distance[minvertex] + arr[minvertex][j];

if (dist < distance[j])

{

distance[j] = dist;

parent[j] = minvertex;

}

}

}

}

if (choice == "3")

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DISPLAYING SHORTEST PATHS OF(" << cities[x] << ") WITH THE OTHER CITIES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

else if (choice == "4")

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DISPLAYING MINIMUM NUMBER OF HOPS OF(" << cities[x] << ") WITH THE OTHER CITIES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

printsolution(distance, parent,x);

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SHORTEST PATHS DISPLAYED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n" << endl;

}

void clear\_memory()

{

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

{

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

{

path[i][j] = " ";

path\_num[i][j] = 0;

}

}

}

void display\_matrix(int a, int b)

{

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

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

{

if (path\_num[i][j]!=0)

cout << cities[i] << "->" << cities[j] << "= " << path[i][j] << path\_num[i][j] << endl;

}

}

double fare\_calculation(int a,int b,double basic)

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CALCULATING FARE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

double fare = 0;

cout << "Shortest Path= " << path[a][b] <<" "<<path\_num[a][b]<< endl;

fare = basic\*path\_num[a][b];

cout << "FARE For this ride= " << fare << endl;

return fare;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FARE CALCULATION COMPLETE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

};

#endif