**CODE**

/\*#############################

MET-ASSISTANT

###############################\*/

#include<iostream>

#include<bits/stdc++.h>

#include<windows.h>

#include<mysql.h>

#include<sstream>

#include<map>

#include<queue>

#include<string>

#include<graphics.h>

using namespace std;

class Graph\_M

{

public:

class Vertex

{

public:

map<string,int> nbrs;

Vertex()

{

this->nbrs={};

}

};

map<string, Vertex> vtces;

Graph\_M()

{

this->vtces = {};

}

int numVertex()

{

return vtces.size();

}

bool containsVertex(string vname)

{

if(vtces.find(vname)!=vtces.end())

return true;

else

return false;

}

void addVertex(string vname)

{

Vertex vtx; //= new Vertex();//doubt/

vtces.insert({vname, vtx});

}

void removeVertex(string vname)

{

Vertex &vtx = vtces[vname];

vector<string> keys ;

//

for (auto it = vtx.nbrs.begin(); it != vtx.nbrs .end();it++)

{

keys.push\_back(it->first);

}

for (auto key : keys)

{

Vertex &nbrVtx = vtces[key];

nbrVtx.nbrs.erase(vname);

}

vtces.erase(vname);

}

int numEdges()//whole map

{

vector<string> keys;

for (auto it = vtces.begin(); it != vtces.end(); it++)

{

keys.push\_back(it->first);

}

int count = 0;

// for (auto key : keys)//traversing the vector

for (auto key: keys)

{

auto it = vtces.find(key);

Vertex vtx = it->second;

count = count + vtx.nbrs.size();

}

return count / 2;

}

bool containsEdge(string vname1, string vname2)

{

Vertex vtx1 = vtces[vname1];

Vertex vtx2 = vtces[vname2];

if(vtces.find(vname1)==vtces.end() || vtces.find(vname2)==vtces.end() || vtx1.nbrs.find(vname2)==vtx1.nbrs.end())

{

return false;

}

return true;

}

void addEdge(string vname1, string vname2, int value)

{

Vertex &vtx1 = vtces[vname1];

Vertex &vtx2 = vtces[vname2];

if (vtces.find(vname1)==vtces.end() || vtces.find(vname2)==vtces.end() || vtx1.nbrs.find(vname2) != vtx1.nbrs.end()) {

return;

}

vtx1.nbrs.insert({vname2, value});

vtx2.nbrs.insert({vname1, value});

}

void removeEdge(string vname1, string vname2)

{

Vertex &vtx1 = vtces[vname1];

Vertex &vtx2 = vtces[vname2];

//check if the vertices given or the edge between these vertices exist or not

if ( vtces.find(vname1)==vtces.end() || vtces.find(vname2)==vtces.end()|| vtx1.nbrs.find(vname2) == vtx1.nbrs.end()) {

return;

}

vtx1.nbrs.erase(vname2);

vtx2.nbrs.erase(vname1);

}

vector<string> returnkeys()

{

vector<string>keys;

for(auto i: vtces)

{

keys.push\_back(i.first);

}

return keys;

}

void display\_Map()

{

cout<<"\t------------------";

cout<<"------------------------------";

cout<<"\t Delhi Metro Map";

cout<<"\t------------------";

cout<<"-------------------------------\n";

vector<string> keys;

for (auto it : vtces)

{

keys.push\_back(it.first);

}

for (int key=0; key<keys.size();key++)

{

string str = keys[key] + " =>\n";

Vertex vtx = vtces[keys[key]];

vector<string> vtxnbrs;

for (auto it: vtx.nbrs)

{

vtxnbrs.push\_back(it.first);

}

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

{

str = str + "\t" + vtxnbrs[i] + "\t";

if (vtxnbrs[i].length()<16)

str = str + "\t";

if (vtxnbrs[i].length()<8)

str = str + "\t";

str = str + to\_string(vtx.nbrs[vtxnbrs[i]]) + "\n";

}

cout<<str<<endl;

}

cout<<"\t------------------";

cout<<"---------------------------------------------------\n";

}

void display\_Stations()

{

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

vector<string> keys;

for (auto it : vtces)

{

keys.push\_back(it.first);

}

int i=1;

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

{

cout<<i<<" "<<keys[j]<<endl;

i++;

}

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

}

bool hasPath(string vname1, string vname2, map<string, bool> &processed)

{

// DIR EDGE

if (containsEdge(vname1, vname2)) {

return true;

}

//MARK AS DONE

processed.insert({vname1, true});

cout<<endl;

Vertex vtx = vtces[vname1];

vector<string> keys;

for (auto it = vtx.nbrs.begin(); it != vtx.nbrs.end(); it++)

{

keys.push\_back(it->first);

}

//TRAVERSE THE NBRS OF THE VERTEX

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

{

if (processed.find(keys[i])== processed.end())

{

if (hasPath(keys[i], vname2, processed))

{

return true;

}

}

}

return false;

}

struct pair

{

string vname;

string psf;

int min\_dis;

int min\_time;

};

string get\_Minimum\_Distance(string src, string dst)

{

int min = INT\_MAX;

//int time = 0;

string ans = "";

map<string, bool> processed;

vector<pair>stock;

// create a new pair

struct pair sp;

sp.vname = src;

sp.psf = src + ",";

sp.min\_dis = 0;

sp.min\_time = 0;

stock.insert(stock.begin(),sp);

// while stack is not empty keep on doing the work

while (!stock.empty())

{

// remove a pair from stack

pair rp = stock[0];

stock.erase(stock.begin());

if (processed.find(rp.vname) != processed.end())

{

continue;

}

// processed put

processed.insert({rp.vname, true});

//if there exists a direct edge b/w removed pair and destination vertex

if (rp.vname == dst)

{

int temp = rp.min\_dis;

if(temp<min) {

ans = rp.psf;

min = temp;

}

continue;

}

Vertex rpvtx = vtces[rp.vname];

vector<string> keys;

for(auto i: rpvtx.nbrs)

{

keys.push\_back(i.first);

}

for(auto nbr : keys)

{

// process only unprocessed nbrs

if (processed.find(nbr) == processed.end()) // != ya == doubt

{

// make a new pair of nbr and put in queue

pair np;

np.vname = nbr;

np.psf = rp.psf + nbr + ",";

np.min\_dis = rp.min\_dis + rpvtx.nbrs[nbr];

stock.insert(stock.begin(),np);

}

}

}

ans = ans + to\_string(min);

return ans;

}

string get\_Minimum\_Time(string src, string dst)

{

int min = INT\_MAX;

string ans = "";

map<string, bool> processed;

vector<pair> stock;

// create a new pair

pair sp;

sp.vname = src;

sp.psf = src + ",";

sp.min\_dis = 0;

sp.min\_time = 0;

// put the new pair in queue

stock.insert(stock.begin(), sp); //replacing addFirst

while (!stock.empty()) {

pair rp = stock[0];

stock.erase(stock.begin());

if (processed.find(rp.vname) != processed.end())

{

continue;

}

// processed put

processed.insert({rp.vname, true});

//if there exists a direct edge b/w removed pair and destination vertex

if (rp.vname == dst)

{

int temp = rp.min\_time;

if(temp<min) {

ans = rp.psf;

min = temp;

}

continue;

}

Vertex rpvtx = vtces[rp.vname];

vector<string> keys;

for(auto i: rpvtx.nbrs)

{

keys.push\_back(i.first);

}

for (string nbr : keys)

{

// process only unprocessed nbrs

if (processed.find(nbr) == processed.end()) {

struct pair np;

np.vname = nbr;

np.psf = rp.psf + nbr + ",";

//np.min\_dis = rp.min\_dis + rpvtx.nbrs.get(nbr);

np.min\_time = rp.min\_time + (rpvtx.nbrs[nbr]\*60)/40;

stock.insert(stock.begin(), np); //replacing addFirst

}

}

}

//double minutes = ceil((double)min / 60);

ans = ans + to\_string(min);

return ans;

}

vector<string> printCodeList()

{

cout<<"List of station along with their codes:\n";

vector<string> keys;

for (auto it: vtces)

{

keys.push\_back(it.first);

}

vector<string> codes;

for (string key: keys)

{

//string key =keys[3];

cout<<key<<endl;

vector<string> words;

string word = "";

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

{

if (key[i]==' '||key[i]=='\0')

{

words.push\_back(word);

word="";

}

else

{

word=word+key[i];

}

}

if (words.size()==1)

{

string code = words[0].substr(0,2);

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

{

if (code[i]>=97 && code[i]<=122)

{

code[i] = (code[i]-32);

}

}

codes.push\_back(code);

//cout<<code<<endl;

}

else if (words.size()>=2)

{

string code;

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

{

code += words[i].substr(0,1);

}

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

{

if (code[i]>=97 && code[i]<=122)

{

code[i] = (code[i]-32);

}

}

codes.push\_back(code);

//cout<<code<<endl;

}

}

int m=0;

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

{

cout<<i<<". "<<keys[i];

if (keys[i].size()<(22-m))

cout<<"\t";

if (keys[i].size()<(14-m))

cout<<"\t";

if (keys[i].size()<(6-m))

cout<<"\t";

cout<<codes[i]<<endl;

if (i==pow(10,m))

m++;

}

return codes;

}

class dijkstrapair

{

public:

string vname;

string psf;

int cost;

};

int dijkstra(string src, string des)

{

int val =0;

vector<string>ans;

map<string, dijkstrapair> m;

for(auto i: vtces)

{

string key = i.first;

dijkstrapair np;

np.vname = key;

np.cost = INT\_MAX;

if(key == src)

{

np.cost = 0;

np.psf = key;

}

m.insert({key,np});

}

while(true)

{

int mi = INT\_MAX;

dijkstrapair rp;

for(auto i :m)

{

if(i.second.cost<mi)

{

mi = i.second.cost;

rp = i.second;

}

}

if(rp.vname==des)

{

val = rp.cost;

break;

}

m.erase(rp.vname);

ans.push\_back(rp.vname);

Vertex v = vtces[rp.vname];

for(auto nbr: v.nbrs)

{

if(m.find(nbr.first)!=m.end())

{

int oc = m[nbr.first].cost;

Vertex k = vtces[rp.vname];

int nc;

nc = rp.cost + k.nbrs[nbr.first];

if(nc < oc)

{

dijkstrapair &gp = m[nbr.first];

gp.psf = rp.psf + nbr.first;

gp.cost = nc;

m[nbr.first] = gp;

}

}

}

}

return val;

}

};

vector<string> get\_Interchanges(string str)

{

vector<string>arr;

vector<string>res;

int f=0,l=0;

for(int i=0; i<str.length();i++)

{

if(str[i] == ',')

{

res.push\_back(str.substr(l,i-l));

l=i+1;

}

}

res.push\_back(str.substr(l,str.length()-l));

cout<<endl;

arr.push\_back(res[0]);

int count =0;

for(int i=1;i<res.size()-1;i++)

{

int index = res[i].find\_first\_of("-");

string s = res[i].substr(index+1);

//cout<<s<<endl;

if(s.length()==2)

{

string prev = res[i-1].substr((res[i-1].find\_first\_of("-"))+1);

string next = res[i+1].substr((res[i+1].find\_first\_of("-"))+1);

if(prev == next)

{

arr.push\_back(res[i]);

}

else

{

arr.push\_back(res[i] + "==>" + res[i+1]);

i++;

count++;

}

}

else

{

arr.push\_back(res[i]);

}

}

arr.push\_back(to\_string(count));

arr.push\_back(res[res.size()-1]);

return arr;

}

void create\_metro\_map(Graph\_M \*g)

{

MYSQL\* con;

MYSQL\_ROW row;

MYSQL\_RES\* res;

con = mysql\_init(0);

con = mysql\_real\_connect(con,"localhost","root",NULL,"miniproject",0,NULL,0);

if(con)

{

int qstate = mysql\_query(con, "SELECT station\_name FROM station");

if(!qstate)

{

res = mysql\_store\_result(con);

while(row = mysql\_fetch\_row(res))

{

g->addVertex(row[0]);

}

}

}

mysql\_close(con);

MYSQL\* con1;

MYSQL\_ROW row1;

MYSQL\_RES\* res1;

con1 = mysql\_init(0);

con1 = mysql\_real\_connect(con1,"localhost","root",NULL,"miniproject",0,NULL,0);

if(con1)

{

int qstate = mysql\_query(con1, "SELECT station\_one, station\_two, weight FROM route");

if(!qstate)

{

res1 = mysql\_store\_result(con1);

while(row1 = mysql\_fetch\_row(res1))

{

int x;

sscanf(row1[2], "%d", &x);

g->addEdge(row1[0],row1[1],x);

}

}

}

mysql\_close(con1);

}

int main()

{

Graph\_M g;

create\_metro\_map(&g);

initwindow(700,600,"Image Window");

readimagefile("map.jpg", 0,0,700,600);

while(true)

{

cout<<"\t\t\t\t~~LIST OF ACTIONS~~\n\n"<<endl;

cout<<"1. LIST ALL THE STATIONS IN THE MAP"<<endl;

cout<<"2. SHOW THE METRO MAP"<<endl;

cout<<"3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION"<<endl;

cout<<"4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION"<<endl;

cout<<"5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION"<<endl;

cout<<"6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION"<<endl;

cout<<"7. GET FARE FOR JOURNEY FROM 'SOURCE' STATION TO 'DESTINATION' STATION"<<endl;

cout<<"8. EXIT THE MENU"<<endl;

cout<<"\nENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) : ";

int choice = -1;

cin>>choice;

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

if(choice == 8)

{

exit(0);

}

switch(choice)

{

case 1:

g.display\_Stations();

break;

case 2:

g.display\_Map();

break;

case 3:

{

vector<string>keys = g.returnkeys();

vector<string> codes = g.printCodeList();

cout<<"\n1. TO ENTER SERIAL NO. OF STATIONS\n2. TO ENTER CODE OF STATIONS\n3. TO ENTER NAME OF STATIONS\n"<<endl;

cout<<"ENTER YOUR CHOICE:"<<endl;

int ch;

cin>>ch;

int j;

string st1 = "", st2 = "";

if(ch == 1)

{

cout<<"ENTER SERIAL NO. 1:";

int in1;

cin>>in1;

cout<<"ENTER SERIAL NO. 2:";

int in2;

cin>>in2;

st1 = keys[in1-1];

st2 = keys[in2-1];

}

else if(ch == 2)

{

string a,b;

cout<<"ENTER CODE 1:";

cin>>a;

cout<<"ENTER CODE 2:";

cin>>b;

transform(a.begin(), a.end(), a.begin(), ::toupper);

transform(b.begin(), b.end(), b.begin(), ::toupper);

for(j=0;j<codes.size();j++)

{

if(a==codes[j])

{

break;

}

}

st1 = keys[j];

for(j=0;j<codes.size();j++)

{

if(b==codes[j])

{

break;

}

}

st2 = keys[j];

}

else if(ch == 3)

{

cout<<"ENTER THE SOURCE STATION: "<<endl;

fflush(stdin);

getline(cin,st1);

cout<<"ENTER THE DESTINATION STATION: "<<endl;

fflush(stdin);

getline(cin,st2);

}

else

{

cout<<"INVALID CHOICE"<<endl;

exit(0);

}

map<string, bool>processed;

if(!g.containsVertex(st1) || !g.containsVertex(st2) || !g.hasPath(st1, st2, processed))

{

cout<<"THE INPUTS ARE INVALID"<<endl;

}

else

{

cout<<"SHORTEST DISTANCE FROM "<<st1<<" TO "<<st2<<" IS "<<g.dijkstra(st1, st2)<<"KM\n"<<endl;

}

break;

}

case 4:

{

string st1="",st2="";

cout<<"ENTER THE SOURCE STATION: "<<endl;

fflush(stdin);

getline(cin,st1);

cout<<"ENTER THE DESTINATION STATION: "<<endl;

fflush(stdin);

getline(cin,st2);

map<string, bool>processed;

if(!g.containsVertex(st1) || !g.containsVertex(st2) || !g.hasPath(st1, st2, processed))

{

cout<<"THE INPUTS ARE INVALID"<<endl;

}

else

{

cout<<"SHORTEST TIME FROM "<<st1<<" TO "<<st2<<" IS "<<((g.dijkstra(st1, st2)\*60)/40)+" MINUTES\n\n"<<endl;

}

break;

}

case 5:

{

cout<<"ENTER THE SOURCE STATION: "<<endl;

fflush(stdin);

string s1;

getline(cin,s1);

cout<<"ENTER THE DESTINATION STATION: "<<endl;

fflush(stdin);

string s2;

getline(cin,s2);

map<string, bool> processed2;

if(!g.containsVertex(s1) || !g.containsVertex(s2) || !g.hasPath(s1, s2, processed2))

{

cout<<"THE INPUTS ARE INVALID"<<endl;

}

else

{

vector<string> str = get\_Interchanges(g.get\_Minimum\_Distance(s1, s2));

int len = str.size();

cout<<"SOURCE STATION : "<<s1<<endl;

cout<<"DESTINATION STATION : "<<s2<<endl;

cout<<"DISTANCE : "<<str[len - 1]<<endl;

cout<<"NUMBER OF INTERCHANGES : "<<str[len - 2]<<endl;

cout<<"~~~~~~~~~~~~~"<<endl;

cout<<"START ==> "<<str[0]<<endl;

for(int i=1; i<len-3; i++)

{

cout<<str[i]<<endl;

}

cout<<str[len - 3]<<" ==> END"<<endl;

cout<<"~~~~~~~~~~~~~"<<endl;

}

break;

}

case 6:

{

cout<<"ENTER THE SOURCE STATION: "<<endl;

fflush(stdin);

string ss1;

getline(cin,ss1);

cout<<"ENTER THE DESTINATION STATION: "<<endl;

fflush(stdin);

string ss2;

getline(cin,ss2);

map<string, bool> processed3;

if(!g.containsVertex(ss1) || !g.containsVertex(ss2) || !g.hasPath(ss1, ss2, processed3))

{

cout<<"THE INPUTS ARE INVALID"<<endl;

}

else

{

vector<string> str = get\_Interchanges(g.get\_Minimum\_Time(ss1, ss2));

int len = str.size();

cout<<"SOURCE STATION : "<<ss1<<endl;

cout<<"DESTINATION STATION : "<<ss2<<endl;

cout<<"TIME : "<<str[len - 1]<<" MINUTES"<<endl;

cout<<"NUMBER OF INTERCHANGES : "<<str[len - 2]<<endl;

cout<<"~~~~~~~~~~~~~"<<endl;

cout<<"START ==> "<<str[0]<<endl;

for(int i=1; i<len-3; i++)

{

cout<<str[i]<<endl;

}

cout<<str[len - 3]<<" ==> END"<<endl;

cout<<"~~~~~~~~~~~~~"<<endl;

}

break;

}

case 7:

{

cout<<"ENTER THE SOURCE STATION: "<<endl;

fflush(stdin);

string ss1;

getline(cin,ss1);

cout<<"ENTER THE DESTINATION STATION: "<<endl;

fflush(stdin);

string ss2;

getline(cin,ss2);

map<string, bool> processed3;

if(!g.containsVertex(ss1) || !g.containsVertex(ss2) || !g.hasPath(ss1, ss2, processed3))

{

cout<<"THE INPUTS ARE INVALID"<<endl;

}

else

{

//Assuming that per km the cost is fixed

//Assuming it to be 3 rupees per km

string str = g.get\_Minimum\_Distance(ss1, ss2);

vector<string>res;

int l=0;

for(int i=0; i<str.length();i++)

{

if(str[i] == ',')

{

res.push\_back(str.substr(l,i-l));

l=i+1;

}

}

res.push\_back(str.substr(l,str.length()-l));

int dis = stoi(res[res.size() - 1]);

int fare = dis \* 3;

if(fare < 20)

{

fare = 20;

}

else if(fare > 70)

{

fare = 70;

}

cout<<"SOURCE STATION : "<<ss1<<endl;

cout<<"DESTINATION STATION : "<<ss2<<endl;

cout<<"FARE : "<<fare<<" RUPEES"<<endl;

}

break;

}

default:

cout<<"Please enter a valid option! "<<endl;

cout<<"The options you can choose are from 1 to 6. "<<endl;

}

}

}