#include <string>

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <vector>

#include <math.h>

using namespace std;

struct Connect

{

int start, end;

double distance;

string name, kind;

Connect(string n, string k, int s, int e, double d)

{

start = s;

end = e;

name = n;

kind = k;

distance = d;

};

};

struct Location

{

int index;

string state, name;

double longitude, latitude, distance;

bool flag;

vector<Connect> connects;

Location(int i, double lo, double la, double d, string s, string n)

{

index = i;

longitude = lo;

latitude = la;

distance = d;

state = s;

name = n;

flag = false;

};

void add(Connect c)

{

connects.push\_back(c);

};

};

string calculat\_direction(double latitude1, double longitude1, double latitude2, double longitude2)

{

const double PI = atan(1.0)\*4;

double a = latitude1 \* PI / 180;

double b = longitude1 \* PI / 180;

double c = latitude2 \* PI / 180;

double d = longitude2 \* PI / 180;

double angle = 0;

string str = "";

if (cos(c) \* sin(d - b) == 0)

{

if (c > a)

angle = 0;

else

angle = 180;

}

else

{

double angle = atan2( cos(c) \* sin(d - b) , sin(c) \* cos(a) - sin(a) \* cos(c) \* cos(d - b) );

angle = fmod( angle \* 180.0 / PI + 360 , 360 );

}

if( angle >= 360-15 || angle <= 0+15 ) { str="N";}

if( angle >= 90-15 && angle <= 90+15) { str="E";}

if( angle >= 180-15 && angle <= 180+15) { str="S";}

if( angle >= 270-15 && angle <= 270+15) { str="W";}

if( angle > 0+15 && angle < 90-15) { str="NE";}

if( angle > 90+15 && angle < 180-15) { str="SE";}

if( angle > 180+15 && angle < 270-15) { str="SW";}

if( angle > 270+15 && angle < 360-15) { str="NW";}

return str;

}

double totalDistance(vector<Connect> connect)

{

double total = 0.0;

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

total += connect[i].distance;

return total;

}

vector<Connect> Beens;

void navigate(vector<Location> &locations, vector<Connect> &been, Location &current\_location, int end)

{

locations[current\_location.index].flag = true;

if( current\_location.connects.size() == 0 )

{

been.pop\_back(); return;

}

if( Beens.size()!=0 && totalDistance(been) >= totalDistance(Beens) )

{

been.pop\_back(); return;

}

for (int i=0; i<current\_location.connects.size(); i++)

{

if( been.size() != 0 && been[been.size()-1].end == end)

continue;

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

{

if( current\_location.connects[i].start == been[j].start && current\_location.connects[i].end == been[j].end ) continue;

}

if( locations[current\_location.connects[i].end].flag == true )

{

if (i == current\_location.connects.size()-1 ) break;

continue;

}

if ( current\_location.connects[i].end == end )

{

been.push\_back(current\_location.connects[i]);

if( Beens.size() == 0 )

{

Beens = been; cout << "> A short path has been found, number of roads: " << Beens.size() << " total distance: " << totalDistance(Beens) << " miles.\n";

}

else if( totalDistance(Beens) > totalDistance(been) )

{

Beens = been; cout << "> A shorter path has been found, number of roads: " << Beens.size() << " total distance: " << totalDistance(Beens) << " miles.\n";

}

for(int j=0; j<locations.size(); j++) locations[j].flag = false;

for(int j=0; j<been.size()-1; j++) locations[ been[j].end ].flag = true;

been.pop\_back();

been.pop\_back();

return;

}

if( locations[current\_location.connects[i].end].flag == false )

{

been.push\_back(current\_location.connects[i]);

navigate(locations, been, locations[current\_location.connects[i].end], end);

}

else

{

if(been.size()>1) been.pop\_back();

return;

}

}

if(been.size()>1) been.pop\_back();

return;

}

int main()

{

vector<Location> locations;

vector<Connect> connect;

double longitude, latitude, distance;

string state, name, kind, ignored;

int start, end, index = 0;

ifstream myfile("/home/www/class/een318/intersections.txt");

while (myfile >> longitude >> latitude >> distance >> state)

{

getline(myfile, name);

Location location(index, longitude, latitude, distance, state, name);

locations.push\_back(location);

index++;

}

myfile.close();

ifstream myfile1("/home/www/class/een318/connections.txt");

while (myfile1 >> name >> kind >> start >> end >> distance)

{

Connect connect1(name, kind, start, end, distance);

Connect connect2(name, kind, end, start, distance);

locations[start].add(connect1);

locations[end].add(connect2);

}

myfile1.close();

cout << "open file success" << endl;

cout << "Location to start: ";

cin >> start;

if (start < 0 || start >= locations.size())

{

cout << "input num between 0 and " << locations.size() << endl;

return 0;

}

cout << endl;

cout << "Location to end: ";

cin >> end;

if (end < 0 || end >= locations.size())

{

cout << "input num between 0 and " << locations.size() << endl;

return 0;

}

cout << endl;

Location current\_location = locations[start];

locations[start].flag = true;

cout << "> Calculating..." << endl;

navigate(locations, connect, current\_location, end);

if(Beens.size()==0)

{

cout << "It is impossible to get from " << start << " to " << end << endl;

return 0;

}

cout << endl << "To get from [" << start << "] to [" << end << "] use the following directions:" << endl;

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

{

string direct = calculat\_direction(locations[Beens[i].start].latitude,locations[Beens[i].start].longitude, locations[Beens[i].end].latitude,locations[Beens[i].end].longitude);

cout << "from intersection " << Beens[i].start << " take " << Beens[i].name << " " << Beens[i].distance << " miles " << direct << " to intersection " << Beens[i].end << endl;

}

return 0;

}







