Nikeem Dunkelly-Allen, ECE318, Assignment 4

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <vector>

using namespace std;

int not\_visited = 999999999;

struct connection

{

string name;

string type;

int a, b;

double length;

connection(string n, string t, int x, int y, double l)

{

name = n;

type = t;

a=x;

b=y;

length = l;

}

void print()

{ cout<<name<<" "<<type<<" "<<a<<" "<<b<<" "<<length<<"\n";}

};

struct location

{

double longi;

double lati;

double dist;

string state, place;

vector <connection \*> roads;

double total\_dist;

bool visited;

int connect;

int num;

string direction;

location(double lo,double la, double d, string s, string p, int n)

{

longi = lo;

lati = la;

dist = d;

state = s;

place = p;

total\_dist = not\_visited;

visited = false;

num = n;

connect = 0;

direction = "";

}

void print()

{

cout<<longi<<" "<<lati<<" "<<dist<<" "<<state<<" "<<place<<" total distance is: "<<total\_dist<<"\n";

}

void print\_roads()

{

int x=roads.size();

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

roads[i]->print();

}

};

vector <location \*> gps;

struct minheap{

vector <int> data;

int size;

minheap(int i)

{

size = i;

data.push\_back(0);

}

bool isempty()

{

if(data.size()==1)

return true;

return false;

}

void add(int i, double d)

{

size += 1;

data.push\_back(i);

gps[i]->total\_dist = d;

work\_up(size);

}

void work\_up(int pos)

{

if(pos==1)

{}

else if(gps[data[pos]]->total\_dist<gps[data[pos/2]]->total\_dist)

{swap(pos,pos/2);

work\_up(pos/2);}

}

void swap(int x, int y)

{

int temp = data[x];

data[x]=data[y];

data[y]=temp;

}

location \* get(int x)

{

return gps[x];

}

void change\_priority(location \* d, double x)

{

int i=0;

for(i; i<size; i++)

if(gps[data[i]]==d)

break;

gps[data[i]]->total\_dist = x;

work\_up(i);

}

void pop\_front()

{

data[1]=data[size];

data.pop\_back();

size -=1;

work\_down(1);

}

void work\_down(int x)

{

if(x\*2 >= size)

{}

else if(gps[data[x]]->total\_dist > gps[data[x\*2]]->total\_dist)

if(gps[data[x]]->total\_dist > gps[data[x\*2+1]]->total\_dist)

{

if (gps[data[x\*2]]->total\_dist > gps[data[x\*2+1]]->total\_dist)

{

swap(x,x\*2+1);

work\_down(x\*2+1);

}

else

{

swap(x,x\*2);

work\_down(x\*2);

}

}

else

{

swap(x,x\*2);

work\_down(x\*2);

}

else if(gps[data[x]]->total\_dist > gps[data[x\*2+1]]->total\_dist)

{

swap(x,x\*2+1);

work\_down(x\*2+1);

}

}

void show\_size()

{

cout<<"the size is: "<<size<<"\n";

}

void print\_all()

{

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

gps[data[i+1]]->print();

}

};

bool isNumeric(string s)

{

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

{

if(isdigit(s[i]) == false)

return false;

}

return true;

}

bool isequal(double x, double y)

{

if(x-y<.0001&&x-y>-.0001)

return true;

return false;

}

string get\_direction(location \* a, location \* b)

{

if(a->lati - b->lati < -.01)//a is more west

if(a->longi - b->longi <-.01) //a is more south

return "sw";

else if(a->longi - b->longi >.01)//more north

return "nw";

else

return "w";

else if(a->lati - b->lati >.01)//more east

if(a->longi - b->longi <-.01) //a is more south

return "se";

else if(a->longi - b->longi >.01)//more north

return "ne";

else

return "e";

else

if(a->longi - b->longi < 0) //a is more south

return "s";

else

return "n";

}

int main()

{

location \* bingus = new location(0,0,0, " ", " ",0);

connection \* bongus = new connection(" "," ",0,0,0);

vector<connection \*> routes;

ifstream fin;

fin.open("/home/www/class/een318/intersections.txt");

if (fin.fail())

{

cout<<"error opening file \n";

exit(1);

}

double lo, lat, d;

string s, p, temp, phldr;

int counter = 0;

while(true)

{

getline(fin,temp);

if(fin.fail())

break;

phldr.assign(temp,0,9);

lo=stod(phldr);

phldr.assign(temp,9,10);

lat=stod(phldr);

phldr.assign(temp,19,8);

d=stod(phldr);

s.assign(temp,28,2);

p.assign(temp,31);

bingus = new location(lo,lat,d,s,p,counter);

gps.push\_back(bingus);

counter ++;

}

fin.close();

fin.open("/home/www/class/een318/connections.txt");

if(fin.fail())

{cout<<"error opening file \n";

exit(1);}

string n,t;

int a,b;

double len;

while(true)

{

fin>>n>>t>>a>>b>>len;

if(fin.fail())

break;

bongus = new connection(n,t,a,b,len);

gps[a]->roads.push\_back(bongus);

gps[b]->roads.push\_back(bongus);

}

delete bingus;

delete bongus;

minheap m(0);

int start, dest, here;

cout<<"Enter the location where you would like to start.. ";

cin>> start;

cout<<"Enter the location of your destination.. ";

cin>> dest;

m.add(start,0.0);

while(!m.isempty())

{

here = m.data[1];

if(m.data[1]==dest)

{break;

}

for(int i =0; i<gps[here]->roads.size();i++)

{

a=gps[here]->roads[i]->a;

if(gps[a]->total\_dist==not\_visited)

m.add(a,gps[here]->roads[i]->length+gps[here]->total\_dist);

else

{

if(gps[here]->total\_dist+gps[here]->roads[i]->length < gps[a]->total\_dist)

{ m.change\_priority(gps[a],gps[here]->total\_dist+gps[here]->roads[i]->length);

}

b=gps[here]->roads[i]->b;

if(gps[b]->total\_dist==not\_visited)

m.add(b,gps[here]->roads[i]->length+gps[here]->total\_dist);

else

{

if(gps[here]->total\_dist+gps[here]->roads[i]->length < gps[b]->total\_dist)

m.change\_priority(gps[b],gps[here]->total\_dist+gps[here]->roads[i]->length);

}

}

}

m.pop\_front();

// m.print\_all();

}

if(here!=dest)

{cout<<"error place couldnt be reached \n";

exit(1);}

vector <location \*> answer;

answer.push\_back(gps[here]);

cout << "Calculating fastest route... \n";

while(true)

{

int h = here;

if(here==start)

break;

if(gps[here]->total\_dist == not\_visited)

{cout<<"error\n";

break;}

for(int i = 0; i < gps[here]->roads.size();i++)

{

double total =gps[here]->total\_dist - gps[here]->roads[i]->length;

//cout<<gps[here]->roads[i]->length<<"\n";

if(here == gps[here]->roads[i]->a)

{

b=gps[here]->roads[i]->b;

if(isequal(total, gps[b]->total\_dist))

{

answer.push\_back(gps[here]);

gps[here]->connect = i;

gps[here]->direction = get\_direction(gps[here],gps[b]);

here = b;

break;

}

}

else

{

a=gps[here]->roads[i]->a;

if(isequal(total, gps[a]->total\_dist))

{

answer.push\_back(gps[here]);

gps[here]->connect = i;

gps[here]->direction = get\_direction(gps[here],gps[b]);

here = a;

break;

}

}

if(here!=h)

break;

}

if(h==here)

{

cout<<"error no path found\n";

exit(1);

}

}

for(int i = answer.size()-1;i>0;i--){

int x=answer[i]->connect;

cout<<"From intersection ";

if(answer[i]->roads[x]->a==answer[i]->num)

cout<<answer[i]->roads[x]->b;

else

cout<<answer[i]->roads[x]->a;

cout<<", take "<<answer[i]->roads[x]->name<<" "<<answer[i]->roads[x]->length<<" miles "<<answer[i]->direction;

cout<<" to intersection " << answer[i]->num << "\n";;

}

}