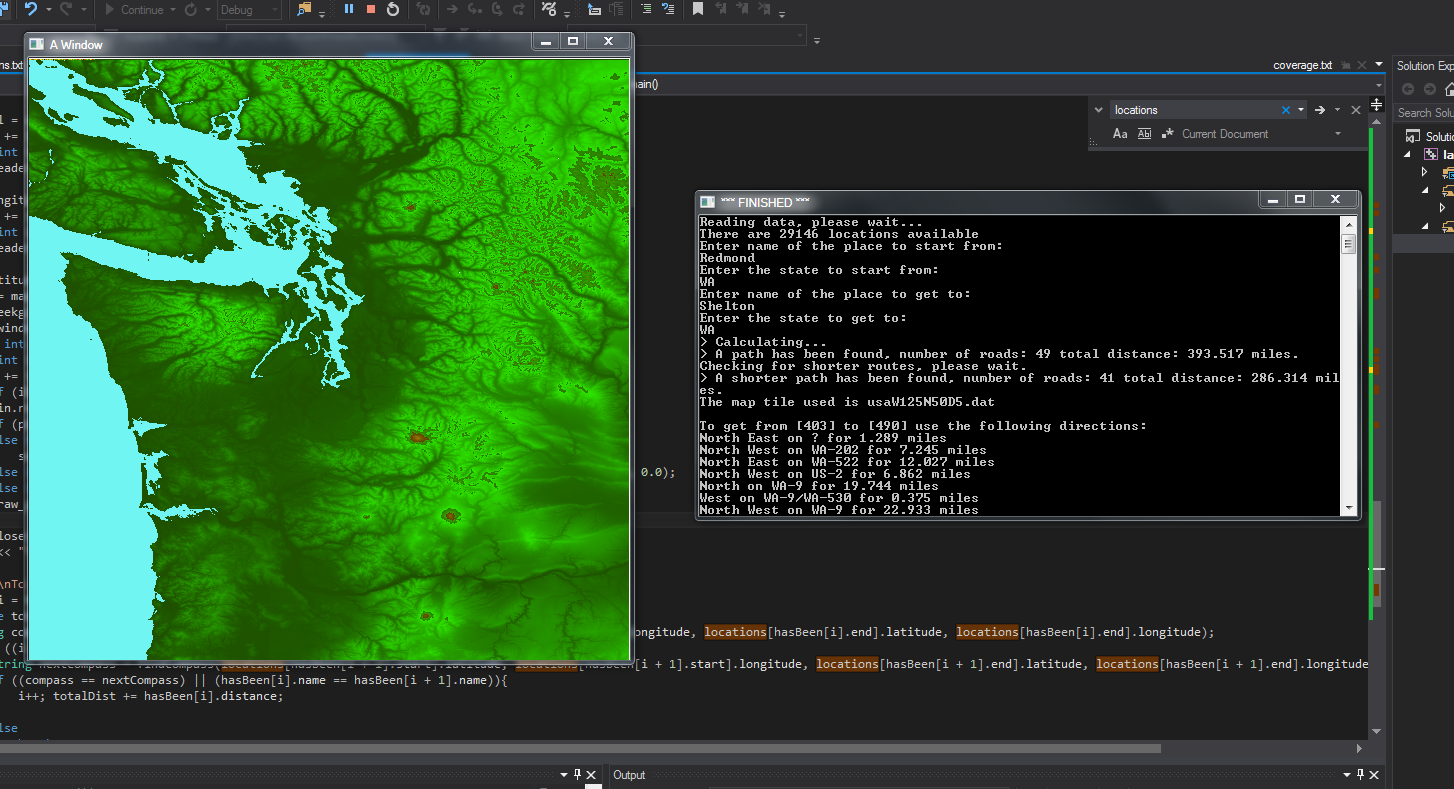
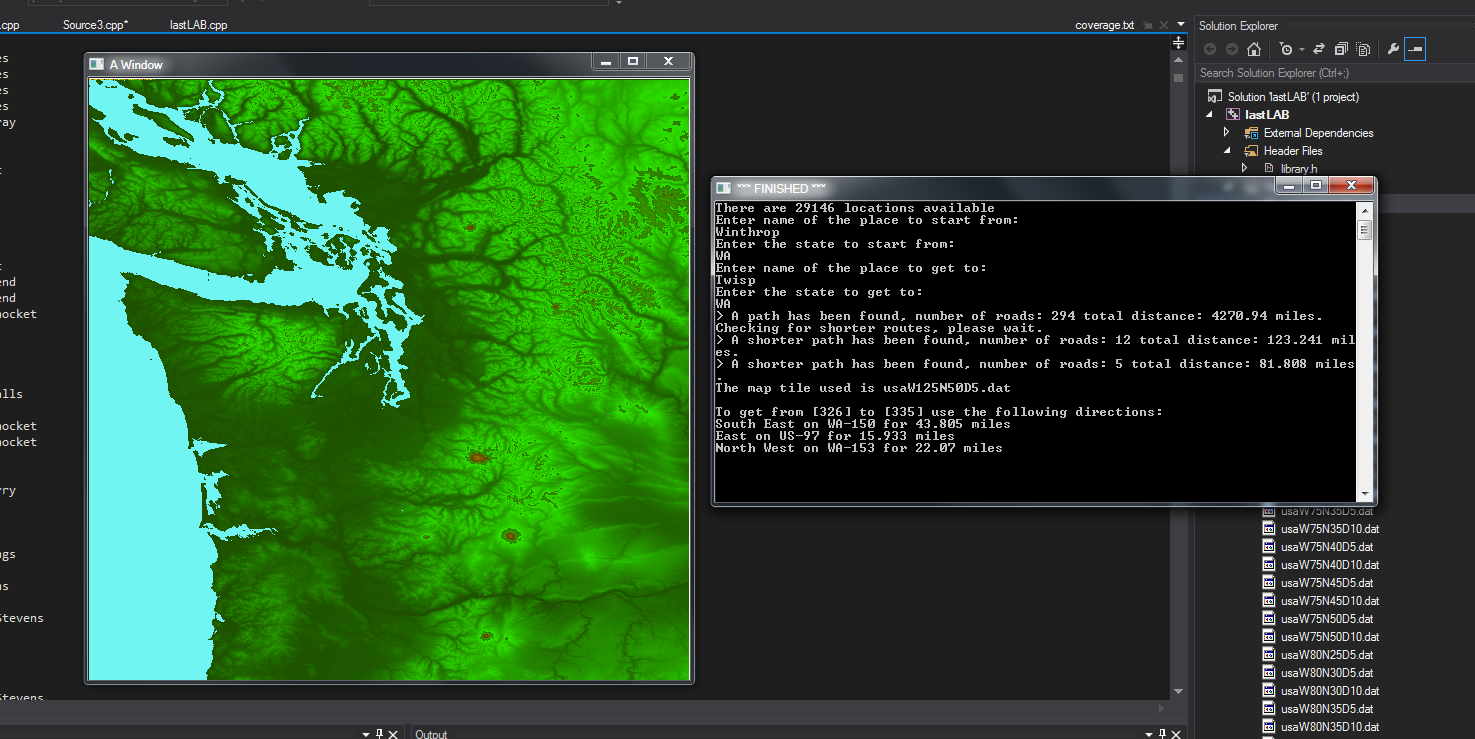
Nikeem Dunkelly-Allen, ECE318, Assignment 5





#include "library.h"

#include <fstream>

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

int HASH(string s){

int C = 42354;

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

C = 739 \* C + s[i];

}

if (C < 0)

C = -C;

return C;

}

struct Road{

string name;

int start;

int end;

double distance;

string type;

Road(string title, string roadtype, int begin, int last, double d){

name = title;

type = roadtype;

start = begin;

end = last;

distance = d;

}

};

struct Location{

int index;

string place;

string state;

double longitude, latitude, distance;

vector<Road> roads;

bool visited;

string key; //state + name

Location \* next;

Location(int pin, double plon, double plat,

double pdis, string psta, string ppla){

index = pin;

longitude = plon;

latitude = plat;

distance = pdis;

state = psta;

place = ppla;

visited = false;

next = NULL;

key = state + place;

}

void add\_connection(Road c)

{

roads.push\_back(c);

};

};

struct Wordentry : public Location

{

int count;

Wordentry(int pin, double plon, double plat, double pdis, string psta, string ppla) : Location( pin, plon, plat, pdis, psta, ppla)

{

count = 1;

}

};

class Hashtable

{

protected:

int size;

Location \* \* table;

public:

Hashtable(int sz)

{

size = sz;

table = new Location \*[size];

for (int i = 0; i < size; i += 1)

table[i] = NULL;

}

Location \* find(string s)

{

int pos = HASH(s) % size;

Location \* h = table[pos];

while (h != NULL)

{

if (h->key == s){

return h;

}

h = h->next;

}

return NULL;

}

int search(string s){

int pos = HASH(s) % size;

Location \* h = table[pos];

while (h != NULL)

{

if (h->key == s){

return h->index;

}

h = h->next;

}

return NULL;

}

void insert(Location \* h)

{

int pos = HASH(h->key) % size;

h->next = table[pos];

table[pos] = h;

}

~Hashtable()

{

for (int p = 0; p < size; p += 1)

{

Location \* h = table[p];

while (h != NULL)

{

Location \* n = h->next;

delete h;

h = n;

}

}

delete[] table;

}

void list()

{

for (int p = 0; p < size; p += 1)

{

cout << p << ":";

Location \* h = table[p];

while (h != NULL)

{

Wordentry \* w = (Wordentry \*)h;

cout << " " << w->key << "(" << w->count << ")";

h = h->next;

}

cout << "\n";

}

}

};

double cHead(double latty1, double longi1, double latty2, double longi2){

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

double a = latty1 \* PI / 180;

double b = longi1 \* PI / 180;

double c = latty2 \* PI / 180;

double d = longi2 \* PI / 180;

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

if (c > a)

return 0;

else

return 180;

else{

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

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

}

}

string findCompass(double latty, double longi, double lattin, double longin){

string compass;

double angle = cHead(latty, longi, lattin, longin);

if (angle >= 360 - 15 || angle <= 0 + 15)

compass = "North";

if (angle >= 90 - 15 && angle <= 90 + 15)

compass = "East";

if (angle >= 180 - 15 && angle <= 180 + 15)

compass = "South";

if (angle >= 270 - 15 && angle <= 270 + 15)

compass = "West";

if (angle > 0 + 15 && angle < 90 - 15)

compass = "North East";

if (angle > 90 + 15 && angle < 180 - 15)

compass = "South East";

if (angle > 180 + 15 && angle < 270 - 15)

compass = "South West";

if (angle > 270 + 15 && angle < 360 - 15)

compass = "North West";

return compass;

}

int from, to;

vector<Road> hasBeen;

double totalDistance(vector<Road> r){

double total = 0.0;

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

total += r[i].distance;

return total;

}

void FindSpath(vector<Location> &locations, vector<Road> &Been, Location &c\_location){

locations[c\_location.index].visited = true;

if (c\_location.roads.size() == 0) {

Been.pop\_back(); return;

}

if (hasBeen.size() != 0 && totalDistance(Been) >= totalDistance(hasBeen)) {

Been.pop\_back(); return;

}

for (int i = 0; i<c\_location.roads.size(); i++) {

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

continue;

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

if (c\_location.roads[i].start == Been[j].start && c\_location.roads[i].end == Been[j].end)

continue;

if (locations[c\_location.roads[i].end].visited == true){

if (i == c\_location.roads.size() - 1) break;

else continue;

}

if (c\_location.roads[i].end == to){

Been.push\_back(c\_location.roads[i]);

if (hasBeen.size() == 0){

hasBeen = Been; cout << "> A path has been found, number of roads: " << hasBeen.size() << " total distance: " << totalDistance(hasBeen) << " miles.\nChecking for shorter routes, please wait.\n";

if (hasBeen.size() == 1) return;

}

else if (totalDistance(hasBeen) > totalDistance(Been)){

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

}

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

locations[j].visited = false;

for (int j = 0; j<Been.size() - 1; j++)

locations[Been[j].end].visited = true;

Been.pop\_back(); Been.pop\_back();

return;

}

if (locations[c\_location.roads[i].end].visited == false){

Been.push\_back(c\_location.roads[i]);

FindSpath(locations, Been, locations[c\_location.roads[i].end]);

}

}

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

return;

}

void main(){

vector<Location> locations;

int siz = 15;

Hashtable hasht(siz);

double longitude, latitude, distance, lat1, lat2, lon1, lon2, plat1, plat2, plon1, plon2;

string state, place, type, map, current\_map = "", ignored;

int index = 0, start, end;

from = 0;

to = 0;

ifstream infile("intersections.txt");

if (!infile.is\_open()){

cout << "Unable to read intersections.txt\n"; exit(1);

}

while (infile >> longitude >> latitude >> distance >> state) {

getline(infile, place);

while (place[0] == ' ')

place.erase(0, 1);

Location loc(index, longitude, latitude, distance, state, place);

locations.push\_back(loc);

Location \*LOC = new Location(index, longitude, latitude, distance, state, place);

hasht.insert(LOC);

index += 1;

}

cout << "There are " << locations.size() << " locations available";

infile.close();

ifstream infile2("connections.txt");

if (!infile2.is\_open()){

cout << "Unable to read connections.txt\n"; exit(1);

}

while (infile2 >> place >> type >> start >> end >> distance) {

Road r1(place, type, start, end, distance);

locations[start].add\_connection(r1);

Road r2(place, type, end, start, distance);

locations[end].add\_connection(r2);

}

infile2.close();

string S1, S2, P1, P2;

cout << endl << "Enter name of the place to start from: " << endl;

cin >> P1;

cout << "Enter the state to start from: " << endl;

cin >> S1;

cout << "Enter name of the place to get to: " << endl;

cin >> P2;

cout << "Enter the state to get to: " << endl;

cin >> S2;

from = hasht.search(S1 + P1);

to = hasht.search(S2 + P2);

/\* while (true) {

cout << "\nLocation to start: ";

cin >> from;

cout << "Destination: ";

cin >> to;

if (from > 0 && from < locations.size() && to > 0 && to < locations.size())

break;

else

cout << locations[0].place << ". Choose between 0 and " << locations.size() << ".\n";

}\*/

vector<Road> Been;

Location current\_location = locations[from];

locations[from].visited = true;

FindSpath(locations, Been, current\_location);

ifstream infile3("coverage.txt");

if (!infile3.is\_open()){

cout << "Unable to read coverage.txt\n"; exit(1);

}

while (infile3 >> lat1 >> lat2 >> lon1 >> lon2 >> map){

if (((locations[from].latitude < lat1) && (locations[from].latitude >= lat2) && (locations[from].longitude >= lon1) && (locations[from].longitude < lon2))

&& ((locations[to].latitude < lat1) && (locations[to].latitude >= lat2) && (locations[to].longitude >= lon1) && (locations[to].longitude < lon2))){

if (current\_map == ""){

current\_map = map;

}

else if (((lat1 - lat2) + (lon2 - lon1)) < ((plat1 - plat2) + (plon2 - plon1))){

current\_map = map;

}

}

plat1 = lat1; plat2 = lat2; plon1 = lon1; plon2 = lon2;

}

infile3.close();

bool map\_exists = true;

if (current\_map == ""){

cout << "Unable to find a map tile for these two locations.\n"; exit(1); map\_exists = false;

}

if (hasBeen.size() == 0){

cout << "It is impossible to get from " << from << " to " << to << endl; exit(1);

}

double pPixel, mapLongitude, mapLatitude;

int size, width, height, xpos, ypos;

if (map\_exists){

int x = -1, y = 0;

string path = current\_map;

ifstream fin;

fin.open(path, ios::in | ios::binary);

if (!fin.is\_open()){

cout << "Unable to read map." << endl; exit(1);

}

fin.seekg(0, ios::end);

size = fin.tellg();

fin.seekg(0, ios::beg);

char num[4];

char bob[142];

fin.read((char\*)&bob, 141); bob[141] = '\0';

num[0] = bob[5]; num[1] = bob[6]; num[2] = bob[5]; num[3] = '\0';

height = atoi(num);

num[0] = bob[17]; num[1] = bob[18]; num[2] = bob[19]; num[3] = '\0';

width = atoi(num);

char reader[20];

int count = 53;

for (int i = 0; true; i++){

reader[i] = bob[count]; if (reader[i] == ' ') { reader[i] = '\0'; break; } count++;

}

pPixel = atof(reader);

count += 17;

for (int i = 0; true; i++){

reader[i] = bob[count]; if (reader[i] == ' ') { reader[i] = '\0'; break; } count++;

}

mapLongitude = atof(reader);

count += 15;

for (int i = 0; true; i++){

reader[i] = bob[count]; if (reader[i] == ' ') { reader[i] = '\0'; break; } count++;

}

mapLatitude = atof(reader);

xpos = mapLongitude, ypos = mapLatitude;

fin.seekg(0, ios::beg);

make\_window(width, height - 5);

short int pixel;

for (int i = 0; i<size; i += 1){

x += 1;

if (i%width == 0) { y += 1; x = 0; }

fin.read((char\*)&pixel, 2);

if (pixel == -500) set\_pen\_color\_int(112, 245, 242);

else if (pixel >= 00 && pixel<1800)

set\_pen\_color\_int(30 + pixel / 120.0, 80 + pixel \* 10 / 120.0, 0.0);

else if (pixel >= 1800 && pixel<2400) set\_pen\_color\_int(3 + pixel / 45.0, 10 + pixel / 16.0, 0.0);

else if (pixel >= 2400) set\_pen\_color\_int(10 + pixel / 30.0, 8 + pixel / 40.0, 0.0);

draw\_point(x, y);

}

fin.close();

cout << "The map tile used is " << current\_map << endl;

}

cout << "\nTo get from [" << from << "] to [" << to << "] use the following directions:\n";

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

double totalDist = hasBeen[i].distance;

string compass = findCompass(locations[hasBeen[i].start].latitude, locations[hasBeen[i].start].longitude, locations[hasBeen[i].end].latitude, locations[hasBeen[i].end].longitude);

while ((i + 1) < hasBeen.size()){

string nextCompass = findCompass(locations[hasBeen[i + 1].start].latitude, locations[hasBeen[i + 1].start].longitude, locations[hasBeen[i + 1].end].latitude, locations[hasBeen[i + 1].end].longitude);

if ((compass == nextCompass) || (hasBeen[i].name == hasBeen[i + 1].name)){

i++; totalDist += hasBeen[i].distance;

}

else

break;

}

cout << compass << " on " << hasBeen[i].name << " for " << totalDist << " miles " << endl;

}