```
#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <queue>

#include <limits>

#include <cmath>

#include <assert.h>

#include <utility>


using namespace std;


/*

Road Network Assignment 4, finding the shortest path.

This program builds upon RoadNetworkA3.cpp, refer to that file for more details

*/


int LOCATION_TOTAL = 0;

struct connection {

	string name, code;

	int isecA, isecB; //intersection A and B

	double length; //in miles

	connection(string n, string c, int a, int b, double l) {

		name = n;

		code = c;

		isecA = a;

		isecB = b;

		length = l;

	}
```

```
};
struct intersection {
	string state, name; //nearest named place name and state
	double lon, lat, dist; // distance from nearest named place
	intersection(string st, string na, double lo, double la, double dis) {
		state = st;
		name = na;
		lon = lo;
		lat = la;
		dist = dis;
	}
};
struct node {//equivalent to location
	vector<connection*> v_conn_data;
	intersection* inter_data;
	int number;

	/// used for dijkstras algorithm
	double spe = numeric_limits<double>::infinity();
	node* predecessor = NULL;
	bool visited = false;
	bool mpqAdded = false;
	///

	node(intersection* inter, int num) {
		inter_data = inter;
		number = num;
	}
```

```
};
struct graph {
	vector<node*> v_nodes;
};
struct compare {//used for priority queue
	bool operator()(node* a, node* b) {
		if (a->spe > b->spe) { // < creates a max priority queue (top() returns largest), > creates a min priority queue (top() returns smallest)
			return true;
		}
		return false;
	}
};

//<helperFunctions>
node* returnNodePointer(graph *g, int locationCode) {
	if (locationCode >= 0 && locationCode < g->v_nodes.size()) {
		return g->v_nodes.at(locationCode);
	}
	return NULL;
}
bool checkNumber(string str) {
	for (int i = 0; i < str.length(); i++) {
		if (!isdigit(str[i])) {
			return false;
		}
	}
	return true;
}
```

```
connection* connectionBetween(node* a, node* b) {
	if (a == NULL || b == NULL) {
		return NULL;
	}
	for (int i = 0; i < a->v_conn_data.size(); i++) {
		if (a->v_conn_data.at(i)->isecB == b->number) {
			return a->v_conn_data.at(i);
		}
	}
	return NULL;
}
string CardinalAndOrdinalDirectionsAtoB(node* a, node* b) {

	double pi = 3.14159265;
	double x = b->inter_data->lon - a->inter_data->lon;
	double y = b->inter_data->lat - a->inter_data->lat;
	double r = sqrt(x * x + y * y);
	double theta;

	if (y >= 0 && r != 0) {
		theta = acos(x / r);
	}
	if (y < 0) {
		theta = -acos(x / r);
	}
	if (r == 0) {
		return "ERROR";
	}
```

```
if (theta >= 0 && theta <= pi / 8) {
	return "E";
}
else if (theta > pi / 8 && theta < 3 * pi / 8) {
	return "NE";
}
else if (theta >= 3 * pi / 8 && theta <= 5 * pi / 8) {
	return "N";
}
else if (theta > 5 * pi / 8 && theta < 7 * pi / 8) {
	return "NW";
}
else if (theta >= 7 * pi / 8 && theta <= pi) {
	return "W";
}

else if (theta <= 0 && theta >= -pi / 8) {
	return "E";
}
else if (theta < -pi / 8 && theta > -3 * pi / 8) {
	return "SE";
}
else if (theta <= -3 * pi / 8 && theta >= -5 * pi / 8) {
	return "S";
}
else if (theta < -5 * pi / 8 && theta > -7 * pi / 8) {
	return "SW";
}
```

```
		else if (theta <= -7 * pi / 8 && theta >= -pi) {
			return "W";
		}

		return "DNE ERROR";
}
void coutTop10InPriorityQueue(priority_queue<node*, vector<node*>, compare> pq) {
		for (int i = 0; i < 10; i++) {
			node* nptr = pq.top();
			pq.pop();
			cout << nptr->number << "-" << nptr->inter_data->name << " priority: " << nptr->spe << 
endl;
		}
}
//</helperFunction>

//<graphModifiers>
void createGraphIntersections(graph &g) {
		ifstream interIN;
		interIN.open("intersections.txt");
		if (interIN.fail()) {
			cout << "unable to open intersections.txt" << endl;
			exit(1);
		}

		string s, n; double lo, la, d;
		string name; node* nptr; intersection* iptr;

		while (!interIN.eof()) {
```

```
		interIN >> lo >> la >> d >> s;
		getline(interIN, name);
		name.erase(0, 1); //removes space from beginning of name

		iptr = new intersection(s, name, lo, la, d);

		nptr = new node(iptr, LOCATION_TOTAL);//location 0 is "No Such Place Exists"

		g.v_nodes.push_back(nptr);

		LOCATION_TOTAL++;
	}
	assert(!interIN.fail());

	interIN.close();
}
void connectGraph(graph &g) {
	ifstream conIN;
	conIN.open("connections.txt");
	if (conIN.fail()) {
		cout << "unable to open connections.txt" << endl;
		exit(1);
	}

	string N, C; int A, B; double L; connection* cptrA, *cptrB;

	while (!conIN.eof()) {
		conIN >> N >> C >> A >> B >> L;
		cptrA = new connection(N, C, A, B, L);
```

```
        g.v_nodes.at(A)->v_conn_data.push_back(cptrA);

        cptrB = new connection(N, C, B, A, L);
        g.v_nodes.at(B)->v_conn_data.push_back(cptrB);
    }
    assert(!conIN.fail());

    conIN.close();

}
//</graphModifiers>

void dijkstrasAlgorithm(graph *g, int SourceNodeCode) {
    /// This function implements Dijkstra's shortest path algorithm
    /// See Introduction to Algorithms 4th edition pg 620 for reference
    /// All page references shown in this program will refer to this book

    //NOTE: pirority queues assume their content will remain constant
    //priority_queue<node*, vector<node*>, compare> mpq;//adds complexities and longer
runtime when implemented
    vector<node*> mpq; //minimum priority queue, we use a vector as the minimum priority
queue's container

    /// "INITIALIZE-SINGLE-SOURCE"
    node* source = returnNodePointer(g, SourceNodeCode);
    if (source != NULL) {
        source->spe = 0;
        source->visited = true;
        source->mpqAdded = true;
```

```
        mpq.push_back(source);
    }
    else {
        cout << "\nError dijkstrasAlgorithm: Source Uninitialized\n" << endl;
        return;
    }
    ///

    /// "INSERT"
    for (int i = 0; i < g->v_nodes.size(); i++) {
        if (g->v_nodes.at(i) != source) {//source already inserted, we dont want a duplicate
            g->v_nodes.at(i)->visited = false;
            g->v_nodes.at(i)->mpqAdded = false;
            g->v_nodes.at(i)->spe = numeric_limits<double>::infinity();; //reinitialize every
node except source to infinity, needed when dijkstras algorithm used more than once
            //mpq.push_back(g->v_nodes.at(i)); //insert all other nodes
        }
    }
    ///

    string sfoo = "foo"; double dfoo = -1;
    intersection* ifoo = new intersection(sfoo, sfoo, dfoo, dfoo, dfoo);
    node* temp = new node(ifoo, dfoo);
    temp->spe = numeric_limits<double>::infinity();
    while (!mpq.empty()) {
        //cout << mpq.size() << endl;
        //cout << mpqTotalUnvisited << endl;
        /// "EXTRACT-MIN"
        int selected = -1;
```

```
node* minimum = temp;
for (int i = 0; i < mpq.size(); i++) {
	if ((minimum->spe > mpq.at(i)->spe)) {
		minimum = mpq.at(i);
		selected = i;
	}
}

//in the while loop the invariant Q = V - S must be kept, this is done by keeping track which nodes have been visited and used as minimum
if (minimum != temp) {
	minimum->visited = true;
}
else {
	cout << "temp error" << endl;
	continue;
}
if (selected == -1) {
	cout << "selected error" << endl;
	continue;
}
else {
	mpq.erase(mpq.begin() + selected);
}
///

for (int i = 0; i < minimum->v_conn_data.size(); i++) { //for each node adjacent to minimum that has not already been visited
	/// preliminary work for "RELAX" function
```

```
			node* nodeA = minimum;

			node* nodeB = returnNodePointer(g, minimum->v_conn_data.at(i)->isecB);

			connection* connData = connectionBetween(nodeA, nodeB);

			double weight = connData->length; // the "weight" of a connection is just its
length

			///


			if (nodeB != NULL) {

				/// "RELAX"

				if (nodeB->spe > (nodeA->spe + weight)) { // change in spe is recoreded
in mpqcpy automatically (since its a vector), however they're not changed in mpq (since its a priority
queue)

					nodeB->spe = (nodeA->spe + weight);

					nodeB->predecessor = nodeA;

					//cout << nodeB->number << "-" << nodeB->inter_data->name
<< " predecessor = " << nodeA->number << "-" << nodeA->inter_data->name << endl;

				}

				///


				if (!nodeB->visited && !nodeB->mpqAdded) {

					nodeB->mpqAdded = true;

					mpq.push_back(nodeB);

				}

			}


			/// "DECREASE-KEY" pg 173

			//this is done automatically in the "RELAX" function

			//theres no need to write anything here only because we are using a vector,
where the contents can be dynamic (at least what they point to)

			//we would have to implement an actual "DECREASE-KEY" function if we used a
priority queue implemented from a c++ library
```

```
                ///
            }
        }
        delete ifoo;
        delete temp;

}
vector<node*> shortestPath(graph *g, int nodeAcode, int nodeBcode) {
        dijkstrasAlgorithm(g, nodeAcode);
        node* nodeA = returnNodePointer(g, nodeAcode);
        node* nodeB = returnNodePointer(g, nodeBcode);
        vector<node*> empty;
        vector<node*> path;

        if (nodeA == NULL || nodeB == NULL) {
                return empty;
        }

        node* temp = nodeB;
        while (true) {
                if (temp == NULL) {
                        return empty;
                }
                if (temp == nodeA) {
                        path.push_back(temp);
                        return path;
                }
                path.push_back(temp);
                temp = temp->predecessor;
```

```
	}
}
void coutShortestPath(graph *g,  int pointANodeCode, int pointBNodeCode) {
	vector<node*> v = shortestPath(g, pointANodeCode, pointBNodeCode);


	if (v.size() == 0) {
		cout << "No path Exists" << endl;
	}
	for (int i = v.size(); i > 1; i--) {
		node* tempA = v.at(i - 1);
		node* tempB = v.at(i - 2);
		connection* connData = connectionBetween(tempA, tempB);
		cout << "from intersection " << tempA->number;
		cout << " take " << connData->name << " " << connData->length << " miles " <<
CardinalAndOrdinalDirectionsAtoB(tempA, tempB) << " to intersection " << tempB->number << endl;
	}


}
void titlePage() {


	cout << "**************************************************" << endl;
	cout << "|                                                |" << endl;
	cout << "| A Graph of Interconnected Nodes (Assignement 4) |" << endl;
	cout << "| By: Brandon Rubio, ECE 318, University of Miami |" << endl;
	cout << "|                                                |" << endl;
	cout << "**************************************************" << endl;
	cout << endl;
}
int main() {
```

```
titlePage();
cout << "Loading.." << endl;
graph G;
createGraphIntersections(G);
connectGraph(G);
string inputA, inputB;
while (true) {
	bool inputAisInt = false;
	bool inputBisInt = false;
	cout << "Enter two intersection numbers" << endl;
	cout << "Intersection A: ";
	cin >> inputA;
	if (inputA == "exit") {
		cout << "exiting program" << endl;
		break;
	}
	else if (!checkNumber(inputA)) {
		cout << "Please enter a positive integer" << endl;
	}
	else {
		inputAisInt = true;
		cout << "Intersection B: ";
		cin >> inputB;
		if (inputB == "exit") {
			cout << "exiting program" << endl;
			break;
		}
		else if (!checkNumber(inputB)) {
			cout << "Please enter a positive integer" << endl;
```

```
			}
			else {
				inputBisInt = true;
			}
		}
		if (inputAisInt && inputBisInt) {
			int stoiA = stoi(inputA);
			int stoiB = stoi(inputB);
			cout << "Searching.." << endl;
			coutShortestPath(&G, stoiA, stoiB);
		}
	}

	return 0;
}
```

Microsoft Visual Studio Debu

```
***************************************************
|                                                 |
| A Graph of Interconnected Nodes (Assignement 4) |
| By: Brandon Rubio, ECE 318, University of Miami |
|                                                 |
***************************************************

Loading..
Enter two intersection numbers
Intersection A: 5
Intersection B: 6
Searching..
from intersection 5 take ? 14.385 miles SW to intersection 6
Enter two intersection numbers
Intersection A: 6
Intersection B: 5
Searching..
from intersection 6 take ? 14.385 miles NE to intersection 5
Enter two intersection numbers
Intersection A: 18039
Intersection B: 19999
Searching..
from intersection 18039 take ? 12.684 miles W to intersection 18212
from intersection 18212 take NC-18 15.51 miles SE to intersection 18531
from intersection 18531 take NC-18 1.388 miles S to intersection 18554
from intersection 18554 take NC-18 1.729 miles SW to intersection 18584
from intersection 18584 take NC-16/NC-18 3.054 miles SW to intersection 18652
from intersection 18652 take NC-18 21.448 miles SW to intersection 18956
from intersection 18956 take US-64/NC-18 1.462 miles SW to intersection 18983
from intersection 18983 take US-64/NC-18 0.86 miles W to intersection 18992
from intersection 18992 take US-64/NC-18 14.498 miles SW to intersection 19262
from intersection 19262 take NC-18 1.546 miles SE to intersection 19275
from intersection 19275 take NC-18 30.446 miles S to intersection 19977
from intersection 19977 take NC-18 1.005 miles SW to intersection 19999
```

```
Searching..
from intersection 6 take ? 14.385 miles NE to intersection 5
Enter two intersection numbers
Intersection A: 18039
Intersection B: 19999
Searching..
from intersection 18039 take ? 12.684 miles W to intersection 18212
from intersection 18212 take NC-18 15.51 miles SE to intersection 18531
from intersection 18531 take NC-18 1.388 miles S to intersection 18554
from intersection 18554 take NC-18 1.729 miles SW to intersection 18584
from intersection 18584 take NC-16/NC-18 3.054 miles SW to intersection 18652
from intersection 18652 take NC-18 21.448 miles SW to intersection 18956
from intersection 18956 take US-64/NC-18 1.462 miles SW to intersection 18983
from intersection 18983 take US-64/NC-18 0.86 miles W to intersection 18992
from intersection 18992 take US-64/NC-18 14.498 miles SW to intersection 19262
from intersection 19262 take NC-18 1.546 miles SE to intersection 19275
from intersection 19275 take NC-18 30.446 miles S to intersection 19977
from intersection 19977 take NC-18 1.005 miles SW to intersection 19999
Enter two intersection numbers
Intersection A: 19999
Intersection B: 18039
Searching..
from intersection 19999 take NC-18 1.005 miles NE to intersection 19977
from intersection 19977 take NC-18 30.446 miles N to intersection 19275
from intersection 19275 take NC-18 1.546 miles NW to intersection 19262
from intersection 19262 take US-64/NC-18 14.498 miles NE to intersection 18992
from intersection 18992 take US-64/NC-18 0.86 miles E to intersection 18983
from intersection 18983 take US-64/NC-18 1.462 miles NE to intersection 18956
from intersection 18956 take NC-18 21.448 miles NE to intersection 18652
from intersection 18652 take NC-16/NC-18 3.054 miles NE to intersection 18584
from intersection 18584 take NC-18 1.729 miles NE to intersection 18554
from intersection 18554 take NC-18 1.388 miles N to intersection 18531
from intersection 18531 take NC-18 15.51 miles NW to intersection 18212
from intersection 18212 take ? 12.684 miles E to intersection 18039
Enter two intersection numbers
Intersection A: 5
Intersection B: 18039
Searching..

C:\Users\brand\source\repos\ECE318Algorithms\Debug\ECE318Algorithms.exe (process 13924) exited with code -1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```