#include <fstream> //file input output

#include <iostream> //console in/output

#include <list> //make a list of neighbors

#include <map> //link indexes with names of cities (i.e. 1 = LA)

#include <queue> //for making the priority queue

#include <stack> //no idea honestly

#include <string> //to use functions when for reading the file

#include <utility> //no idea here either

#include <vector> //for linking nodes and edges (making a graph)

using namespace std;

#include <cstdlib>

struct Node

{

string name;

bool isVisited;

list<pair<int, double> > neighbors;

int prev;

double cost;

};

struct Dijkstra //find the shortest path between nodes and the cost

{

int index; //index of node within database array

int prev; //previous nodes index in database array

double cost; //cost increment of getting from one node to the next

};

//function to compare two path with overloading operator

bool operator<(const Dijkstra& a, const Dijkstra& b)

{

//reverse logic to sort cost from low to high

return b.cost < a.cost;

}

//use dijkstra's algorithm to find the cheapest route

pair<stack<int>, double> getCheapestRoute(int iStart, int iEnd, vector<Node>& database)

{

pair<stack<int>, double> result;

//initialize the database

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

{

database[i].cost = 0;

database[i].prev = -1;

database[i].isVisited = false;

}

//make a to-do list for the program filled with dijkstra type objects

priority\_queue<Dijkstra> toDo;

//create a Dijkstra object for the start node with prev=-1, cost=0

//push the start node's Dijkstra object onto the priority queue to-do list

Dijkstra startNode = {iStart, -1, 0};

toDo.push(startNode);

//while the priority queue to-do list still has objects

while(!toDo.empty())

{

Dijkstra popT = toDo.top();

toDo.pop();

if(database[popT.index].isVisited) continue;

database[popT.index].isVisited = true;

database[popT.index].cost = popT.cost;

database[popT.index].prev = popT.prev;

//leave the while loop once we hit the destination node

if(popT.index == iEnd) break;

//loop for every unvisited neighbor of the object's node

for (pair<int, double> &neb : database[popT.index].neighbors){

//create a new object based on the neighbor

//the cose of this node is the edge's cost

//assign the prev value as the node's index

//push the Dijkstra object into the priority queue

Dijkstra temp = {neb.first, popT.index, database[popT.index].cost + neb.second};

toDo.push(temp);

}

}

result.second = database[iEnd].cost;

int nextIndex = iEnd;

while(nextIndex != iStart)

{

result.first.push(nextIndex);

nextIndex = database[nextIndex].prev;

}

result.first.push(iStart);

return result;

}

int main()

{

map <int, string> locations; //location names and indexes

string fName; //file name

cout << "Enter file name with locations and paths: "; //prompt

getline(cin, fName); //get file name from user

ifstream fin;

fin.open(fName);

if (!fin.good()) throw "I/O error";

//read file up till the first -1

string line, word1, word2;

word1 = word2 = "";

while(getline(fin, line) && line != "-1")

{

int itr = 0;

word1 = "";

for (auto x : line)

{

//first word (the numer) is assigned to word2, i.e. 3

if (x == ' ' && itr == 0)

{

word2 = word1;

word1 = "";

itr++;

}

//the rest of the line is put in string word1

else

{

word1 = word1 + x;

}

}

//insert pair with the number (word2 as int) and the location (word1)

locations.insert(pair<int, string>(stoi(word2), word1));

}

//output location options to the user

cout << "Locations";

for (auto x: locations) cout << " : " << x.second;

cout << endl;

//read the input file and process it

vector<Node> database;

while (getline(fin, line) && line != "-1") //read until -1

{

string fromCity, toCity, cost;

int iter = 0;

string word = "";

for (auto x : line)

{

if (x == ' ')

{

if (iter == 0) //from city is first word

{

fromCity = locations.at(stoi(word));

iter++;

}

else if (iter == 1 || iter == 2) //dest is second word

{

toCity = locations.at(stoi(word));

iter++;

}

word = "";

}

else word = word + x;

}

cost = word; //cost is last word

//add nodes for new cities included in the edge

int iToNode = -1, iFromNode = -1, i;

for (i = 0; i < database.size(); i++) //go to "to" city index

{

if (database[i].name == fromCity) break;

}

if (i == database.size()) //node isn't logged yet

{

//add the node to the database

Node fromNode = {fromCity};

database.push\_back(fromNode);

}

iFromNode = i;

for (i = 0; i < database.size(); i++) //go to "from" city index

{

if (database[i].name == toCity) break;

}

if (i == database.size()) //node isn't logged yet

{

//add the node to the database

Node toNode = {toCity};

database.push\_back(toNode);

}

iToNode = i;

//add the edges to the database for the node

double edgeCost = atof(cost.c\_str());

database[iFromNode].neighbors.push\_back(pair<int, double>(iToNode, edgeCost));

database[iToNode].neighbors.push\_back(pair<int, double>(iFromNode, edgeCost));

}

fin.close();

while (true)

{

string fromCity, toCity;

cout << endl << "Enter the starting city [enter to exit]: ";

getline(cin, fromCity);

if (fromCity.length() == 0) break;

//locate the starting city

int iFrom;

for (iFrom = 0; iFrom < database.size(); iFrom++)

{

if (database[iFrom].name == fromCity) break;

}

cout << "Enter the destination city [enter to exit]: ";

getline(cin, toCity);

if (toCity.length() == 0) break;

//locate the destination city

int iTo;

for (iTo = 0; iTo < database.size(); iTo++)

{

if (database[iTo].name == toCity) break;

}

//Output the shortest route and the path taken

cout << "Route: Start";

pair<stack<int>, double> result = getCheapestRoute(iFrom, iTo, database);

for (; !result.first.empty(); result.first.pop())

{

cout << " -> " << database[result.first.top()].name;

}

cout << endl << "Total cost: $" << result.second << endl;

}

}



