CSE 232 Spring 2017

Programming Project #8

Assignment Overview

This project focuses on the use of structs. It is worth 50 points (5% of your overall grade). It is due Monday 3/27 before midnight.

The Problem

Imagine that you are one of a number of people in a large shopping mall when all the wireless access points go out. Is it still possible to at least get a message via your phone to someone still in the mall?

The answer would be yes if you could set up what is called an "ad hoc" network. Such a network establishes a route between phones (or more generally, nodes) by passing a message from the source phone, through a series of other phones, to the destination phone. We are going to simulate (to some extent) creating an ad hoc network and creating routes through existing phones/nodes.

Basic Premise

The way this will work is as follows. The "network" knows (via GPS) the x,y location of all the other phones in the mall[1]. When the source phone sends a message, the networ establishes a route from the source phont to the destination phone using what is typically called a ***greedy*** method. A greedy method uses a simple rubric to solve a problem. In this case the rubric is that each step in the route is from the current phone (whatever the current phone is in the route being established) to the next phone that is ***closest*** in x,y coordinates. Note this is not a guarantee to create the shortest route to the destination, but it is a reasonable way to establish "which phone/node comes next" in the route.

Two `structs`

```
struct Node{
  int x;
  int y;
  string label;

  Node()=default;
  Node(int i, int j, string l) :  x(i), y(j), label(l) { } ;

  string to_string () const;
  bool equal_nodes(const Node&);
  double distance(const Node &)const;
};
```

```
struct Network{
  string label;
  map<string, Node> nodes;
  vector<string> route;

  Network()=default;

  Network(ifstream &);
  string to_string () const;
  Node get_node(string);
  bool in_route(const Node&);
  Node closest(Node &);
  string calculate_route(const Node&, const Node&);
  string route_string();
};
```

A `Node` has an `int x` and `int y` 2D coordinate, as well as a `string label`.

[1] This is an unreasonable assumption, but for now it makes our job easier so we go with it.

A `Network` has a `string label` and `map<string, Node> nodes`, mapping the label of each Node in the network to the actual Node. Thus the Network knows all the nodes in the network. Finally, a Network has a `vector<string> route` which is a sequence of node labels for a requested route.

Your Tasks
Your job is to complete the underlined methods above (all methods, no functions) in the two structs.

Node functions
- `string to_string () const;` Converts a `Node` to a `string`. See the test cases for the format. The `const` at the end means the method is guaranteed not to change the variable pointed to by `this`.
- `double distance(const Node &)const` Returns the Euclidean distance between two Nodes (look it up).
- `bool equal_nodes(const Node&)` You cannot use the standard comparison operator == (and by extension `!=`) to check whether two nodes are equal. You can assume that two Nodes are equal if their labels are the same. This will come in handy.

Network functions
- `Network(ifstream &)` constructor.
    - reads in a text description of the network from the provided (open) file stream
    - for each line, creates a new `Node` and initializes it according to the provided Node constructor (see test cases for format)
    - adds the new node to the `map<string, Node> nodes`, where the string is the `Node` label.
- `string to_string () const` Prints all the nodes in `nodes` of `net`. Uses `Node::to_string`. See test cases for format.
- `Node get_node(string)` and `void put_node(Node)`. Either return a Node (based on a string, the label of a `node` in the map `nodes`) or adds a Node to the map `nodes` in a network.
    - If `get_node` cannot find the indicated label in the `nodes` map, it throws an `out_of_range` error
- `bool in_route(const Node &node)` Useful to know if `node` is already in a network's `route` vector. If so, then don't use that `node` in the route (it will create a cycle if you do).
- `Node closest(Node &n)` For a given node `n`, return the closest node in the network (excluding n itself) in terms of Euclidean distance.
- `string calculate_route(const Node &start, const Node &finish)` Calculates a route from `start` to `finish` in the network. Returns a string with the total distance covered and the sequence of node labels in the route (see the test cases for format).

Test Cases
Test cases are provided in the subdirectory `test` of the project directory (because it is getting confusing), at least one for each function . ***However!*** It is time you start writing your own tests. The tests I provide ***are not complete***, and we are free to write extra tests for grading purposes. Do a good job and test your code. Just because you pass the one (probably simple) test does not necessarily mean your code is fine. Think about your own testing!!!!

Assignment Notes
1. You are given the following files:

a. `main.cpp` – This file includes the main function where the test cases will be run. **Do not modify** this file.
b. `functions.h` – This file is the header file for your functions.cpp file. **Do not modify** this file.
c. input#.txt – These are text files that will be used to run the test cases.
d. correct_output_#.txt – These text files will be used to grade your output based on the corresponding input files. Be sure that your output matches these text files exactly to get credit for the test cases.

2. You will write only `functions.cpp` and compile that file using `functions.h` and `main.cpp` in the same directory (as you have done in the lab). You are only turning in `functions.cpp` for the project.
3. Comparing outputs: You can use redirected output to compare the output of your program to the correct output. Use redirected output and the "diff" command in the Unix terminal to accomplish this:
    a. Run your executable on the desired input file. Let's assume you're testing input1.txt, the first test case. Redirect the output using the following line when in your proj06 directory: `./a.out < input1.txt > output1.txt`
    b. Now your output is in the file output1.txt. Compare output1.txt to correct_output1.txt using the following line: `diff output1.txt correct_output1.txt`
    c. If the two files match, nothing will be output to the terminal. Otherwise, "diff "will print the two outputs.

Deliverables

`functions.cpp` -- your completion of the functions based on `main.cpp` and `functions.h` (both provided).

1. Remember to include your section, the date, project number and comments.
2. Please be sure to use the specified file name, i.e. "functions.cpp"
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient