CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2014
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MWF 9:05-9:55
Handout 17: Lab #11 — Vickery 100

1 Breadth-First Search: Finding the Center of the USA

In this lab exercise, we will use practice using breadth-first search to compute shortest paths. Specifically, we will be trying to find the most "central" states in the continental USA according to their *eccentricities*. The eccentricity of a node x in a graph is the shortest path distance to the node y farthest away from x. A node with minimum eccentricity is sometimes known as a *center* of a graph, and its eccentricity is known as the *radius* of the graph (this makes sense if you mentally picture the graph as a circle).

Our input for this lab comes from the following file, provided by Don Knuth as part of the Stanford Graphbase:

`/group/course/cpsc212/f14/lab11/usa_48_state_graph.txt`

in which each line tells you an adjacent pair of states (and Washington DC) in the continental USA. Here is a picture of the resulting graph:

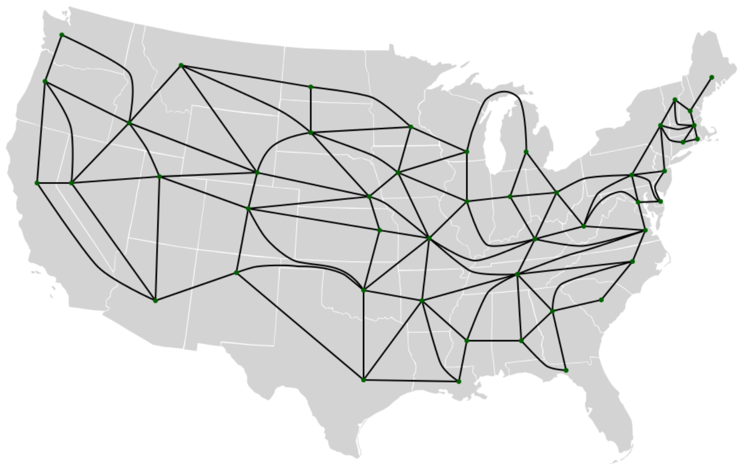

Figure 1: Graph indicating adjacency between pairs of states (image courtesy of Randy Bryant).

Your goal is to use breadth-first search to compute the eccentricity of each node (state), and then to print a list of all states along with their eccentricities (sorted by eccentricity).

2 Use of the STL

To represent our graph in memory, we will make use of an STL "map", since each state is identified by a two-character string. Specifically, you should define your graph in a structure such as the following:

```
struct State {
  string name;
  vector<string> neighbors;
};

map<string, State *> Nodes;
```

You can now conveniently access information on any node in your graph using array notation; for example, `Nodes["SC"]` is a pointer to the structure holding information for the state of South Carolina. To test whether a state is present in your structure (e.g., as you read in the input), you can see if `Nodes.count("SC")` is positive. Don't forget that when you add a state to the structure for the first time, you will need to allocate memory for it; for example:

```
Nodes["SC"] = new State;
Nodes["SC"]->name = "SC";
Nodes["SC"]->neighbors.push_back("NC");
...
```

Also do not forget that when you add an edge from state x to state y, you will need to add the edge to the neighbor lists of both x and y.

When you run breadth-first search starting from every node in the graph, you will need to use an iterator:

```
map<string, State *>::iterator it;
for (it = Nodes.begin(); it != Nodes.end(); it++) {
  // compute shortest paths starting from a designated source state
  // it->first is the name of the state, and it->second is a pointer
  // to the struct for that state
}
```

Breadth-first search needs to keep track of a "distance label" for each node; you can either store these inside your State struct or store them in a separate map (e.g., `map<string,int> dists`). Don't forget to re-initialize these distances every time you start running your breadth-first search.

As you compute eccentricities, you probably want to store all states along with their associated eccentricities in a vector of pairs, to facilitate sorting using the STL sort function. As you want to sort on eccentricity, the first element of the pair should be eccentricity, and the second element should be the state name.

If you finish early, there are many other fun (optional) exercises you might want to try. For example, consider finding the two states that are farthest apart (the distance between these is known as the *diameter* of the graph) and printing the sequence of states along a shortest path between them.

3 Submission and Grading

Please name your file `main.cpp`. For this lab, you will receive 8 points for correctness and 2 points for having well-organized, readable code. Zero points will be awarded for code that does not compile, so make sure your code compiles on the lab machines before submitting!

Final submissions are due by 11:59pm on the evening of Friday, November 21. No late submissions will be accepted.