# CpSc 2120: Algorithms and Data Structures

**Instructor:** Dr. Brian Dean
**Webpage:** `http://www.cs.clemson.edu/~bcdean/`
**Handout 15:** Lab #10

Fall 2017
TTh 12:30-1:45
McAdams 119

## 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/f17/lab10/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:

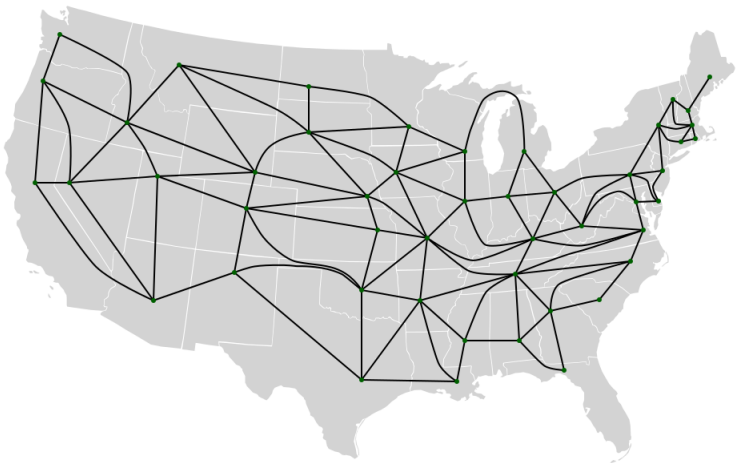

Figure 1: Graph indicating adjacency between pairs of states (image courtesy of Randy Bryant).

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. The most straightforward approach for doing this is probably to make a map from the name of a state to a vector containing its neighbors:

```
map<string, vector<string>> Neighbors;
```

After doing this, for example, you could write `Neighbors["SC"].push_back("NC")` to add North Carolina to the list of neighbors of South Carolina. 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 $y$.

Breadth-first search needs to keep track of a "distance label" for each node; it is probably best to store these in a separate map (e.g., `map<string,int> dists`). Don't forget to re-initialize or clear these distances every time you start running your breadth-first search, so distances from the prior search don't cause problems with the next 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 Sunday, November 26. No late submissions will be accepted.