**Phase Four**

**Due: Thursday, February 27th**

The big one! Our last phase! Time for *the* algorithm!

You will turn in **3 or 4** classes for phase four:

1. GraphTraverser

2. Edge

3. EdgeList

4. A tester (or a main method in GraphTraverser)

**Part 1: The algorithm!**

**CLASS: GraphTraverser**

This class will do the actual logic of the four algorithms. It will do this using ONE method. This method is traverse(). It takes in three parameters:

* –  DataStructure<Path> ds - the data structure we will use to maintain records. This parameter is the one that ultimately determines which of the four algorithms you will use.
* –  AdjacencyList graph – the graph we will run on
* –  String startName – all the algorithms in this homework are single-source. This  parameter is the name of the start vertex for our traversal.
* The method will produce an ArrayList<Path> of results (essentially the Code version of our “visited” list). These are all the valid paths in the graph produced by the algorithm. If the algorithm was running Dijkstra's, for example, it will be all the shortest paths. If it were running BFS, it will be the paths that use the least amount of edges.
* Whether this method will return this ArrayList<Path> or not and whether it will be static or not is your choice.
* You should also implement a printPaths() method that will print all the paths in a given ArrayList<Path> on separate lines. You can to decide take theArrayList in as a parameter or not.
* You need to take advantage of polymorphism in the traverse() method. YOU MAY NOT USE INSTANCEOF OR ANYTHING LIKE THIS TO DO IT. It must be ONE algorithm that ends up being able to do FOUR. There must be no switch or if statements that try to determine which DataStructure<Path> you are using. The method needs to be agnostic.

To verify results, there are solution files on the google drive. For graph2.txt, just do DFS and BFS on “A”. For graph3.txt, do Dijkstra's and Prim's on “W”.

-To check DFS, you probably need to go through by hand since the results can be very different depending on how the adjacency list was implemented.

-To check BFS, first see if the number of vertexes in each path is the same as the solution's. If one of them is different, there is an error. For the ones that have the same number of vertexes, but different vertexes, check the path's validity (i.e. the edges in the path exist in the graph).

-To check Dijkstra's, first see if the path distances are the same as the solution's. Then, for anyone that has the same distance but different vertexes, check path's validity.

**Part 2: Readable Results**

Not done yet. For BFS, DFS, and Dijkstra's, the path list is a nice solution. For Prim's, which is trying to make a minimum spanning tree, it's a little awkward. We are going to convert our path list into an edge list.

**CLASS: Edge**

The edge class is simple. It has two vertexes and a weight. All you need is getters/setters and a toString. The toString format:

vertex<->vertex weight.

**CLASS: EdgeList**

Just a list of edges. No need to read in a file, but do have a way to add an edge. The toString merely goes over each edge.

**CLASS: GraphTraverser (from part 1)**

Create a printEdges() method. This method converts the ArrayList<Record> into an edge list. The easiest way to do this is to make an edge out of the last two vertexes of each path except the starting one. As you print edges, take a tally of the edge weights and print the sum right after.

-To check Prim's, make sure the total edge weights are the same and see if the edges themselves make sense.

**Part 3: Pretty Runner**

Finally, implement a small driver class (or main method in GraphTraverser) for testing. The driver should accomplish the following steps:

1. Ask the user for a graph file

2. Print the adjacency list

3. Ask the user for the type of graph algorithm

4. Ask the user starting vertex

5. Ask the user if they would like a path list or an edge list

6. Print the respective path list or edge list

7. Ask the user if he wants to input a different graph, perform a different algorithm, or stop. Go to step 1, 3, or quit depending on the answer.

You can assume the user will only type appropriate responses.