**Phase One**

**Due: Wednesday February 3rd, *before class***

This assignment is the first of 4 phases to code graph theory! Our end game is to actually perform BFS, DFS, Djikstra’s, and Prim’s on different graphs. However, we have to do some other things first. Our first step is actually storing the graph in memory. We did that once with the adjacency matrix, but for this assignment, you’ll be making adjacency lists.

You will turn in three classes for phase one.

1. Vertex

2. AdjacencyList

3. A tester

You have been provided with:

for (int i = 1; i<= 3; i++){

graphi.txt

graphi.png

graphiAdjList.txt

}

You can use these files to test your code! Please please please test as you go. Seriously. If you wait until the end to test, I will not be happy. Unhappy teachers do not like to help students.

Let’s get into it.

**CLASS: Vertex**

The Vertex class needs to keep track of three things:

– A name for the vertex

– A boolean flag that indicates if this vertex has been visited or not

– All the vertices this particular vertex is adjacent to as well as all the edge weights to

those vertices (all adjacent neighbors and the cost to get to them)

You can choose whatever data structure you want for this. I recommend using a hash table, but you are more than welcome to pick your own! Java has it’s own hash structure (they call it a HashMap). I choose this one because you can do key/value as vertex/edge weight.

HashMap: <http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>

The Vertex class should also provide at least the following functionality:

– Add a vertex and weight to that vertex to the adjacency structure

– Allow the user to get the weight to some vertex if that vertex exists in the adjacency structure. Else, return something like null.

– Getters/Setters for the name and boolean

– Return all the adjacent vertexes in some structure (Hint: A simple graph means a vertex shouldn't appear twice in this structure)

– A toString that uses the following format:

Name : <adjacent vertex, weight> <adjacent vertex, weight> <adjacent vertex, weight>

See the graph1 files for examples

**CLASS: Tester/Runner/Whatever**

Before you move on to the adjacency list, you must *thoroughly* test your code.

Start testing the basic functionality first, and don’t worry about the tricky parts. Eventually though, try thinking of ways to break it, because I will try to break it.

Try to think of weird situations like what happens if…

I try to mark something as visited and it’s already visited?

I try to add a vertex to your adjacency structure twice?

I try to add a vertex to its own adjacency structure? (A.add(A))

Once you’ve finished *and tested* your vertex code, you’ll get started on making an adjacency list. Your adjacency list is actually how you’re going to store the whole graph. So. I recommend reading through the description below and looking at the provided files (template, text, and picture) first. **Then, write down what you’re supposed to do in your own words, on paper, in detail before you think about the coding part.**

**CLASS: AdjacencyList**

The AdjacencyList simply needs to maintain a master list of all vertexes in the graph.

The class needs to be able to do the following:

– Read in a file in edge list format and create an adjacency list out of it. You may do this in the constructor.

Edge list format: Vertex1 Vertex2 Weight

So vertex1 is connected to vertex2 with the given edge weight

– Return a vertex object given the name of that vertex

– Reset the boolean flag of all the vertexes in the list

– A toString that simply returns a String of each vertex's toString in separate lines.

A common error in the creation of the adjacency list is the duplication of variables. Avoid this by always searching for an already existing vertex with whatever name you just read in. If no vertex exists, then you may make a new one.