**Description of Algorithms**

1. TheGraph
   1. add – adds a new node to the HashMap list of names and ArrayList of vertexes
   2. size – returns the current size of the graph by returning the size of the list of vertexes
   3. addAdjac – adds a new adjacency to the specified node
   4. getVert – gets a node by finding the int hashed to the string parameter, and getting the node at that index from the array of nodes
   5. Getters and Setters
2. GVertex (inner class of TheGraph)
   1. addAdj – adds a new adjacency with the specified destination node and weight
   2. Getters and Setters
3. Adjac (inner class ofGVertex)
   1. Getters and Setters
4. PriQueue
   1. deq – returns the current head of the queue, decrements the size
   2. enq – ctreates a new queItem from the vertex and priority parameters, adds it to the queue, and sorts the que, increments the size
   3. sort - starts at the tail (the new item added to the queue), and using a while loop, adjusts pointers until the item previous the added item has a priority lower or equal than that of the added item
   4. nullify – nulls the prev and next fields of the queItem
   5. size – returns the size of the queue
5. QueItem (inner class of PriQue)
   1. getNode – returns the node of the QueItem
6. DijAlg – Dijkstras Algorithm class
   1. dijkstras – runs Dijkstra’s algoritm on the graph and start node parameters
   2. tracePaths – runs Dijkstra’s algoritm on the graph and start node parameters and creates and ArrayList of the paths while printing them out as well
7. DijObj – object for dijkstras algorithm to run on
   1. Getters
8. Driver
   1. readFile – reads in a file and adds to graph based on 3-tuples, finds the stop point, and sets the next line as the start node
   2. main – reads in a file and runs dikstra’s algorithm and DijAlg.tracePath function on the resulting graph and start node

Layout and Structure

TheGraph is the main graph structure. It includes inner classes for the GVertex structure and Adjac is the adjacencies between nodes, assuming the node containing the adjacency is the source node. There data members are a HashMap containing the String names of nodes as the keys and integers as the values and an ArrayList containing nodes. The integer values in the HashMap are indexes of the corresponding nodes in the ArrayList.

GVertex contains a String element, a Dijkstra’s Algorithm object, and a list of adjacencies as well as the Adjacency class. It also has a method for adding adjacencies. This adjacency class contains a node and a weight

Whenever a node is added to the graph, its name is hashed to the index of the array of nodes it will be added to, a new GVertex is created and added to the array of nodes (the size function will get the size of this array.) Nodes can be found by passing in a string and getting the int hashed to that String and getting that index in the array of nodes. When a new adjacency is added to the graph, the add adjacency method is called in the source node parameter using the destination node and weight parameters. A new adjacency object is created with the destination node and weight, and is added to the adjacency list of the source node and sorted based on weight.

The PriorityQue class is the queue used in dijkstra’s algorithm. The items added to the queue are of the queitem class, containing a previous and next pointer, the node itself, and it’s priority. There’s only a node getter because the other data members are only used in the queue class itself. Items added to the queue are created using the Gvertex and integer priority. The que is then sorted to keep priority order. Items can be dequed and returned from the front of the queue.

The Dijkstra’s pulls the other classes together. It contains the inner class of Dijkstra’s objects. It contains the data members Dijktra’s algorithm needs: the node itself, the Boolean “known,” the shortest found distance, and the node this distance came from. The Dijkstra’s algorithm itself iterates on the graph and makes Dijlksra’s objects from the nodes and adds them to a list. It then runs dijkstra’s on the list of dijkstra’s objects. The paths can then be printed and recorded using the tracePaths method.

The driver class reads in a file and creates a graph and start node. Dijkstra’s algorithm is run and the paths printed out.

**Testing**

I created a JUnit test class. I made methods for add and checked the sizes after adding and whether the specified key was in the HashMap. The addAdjac method adds nodes and adjacencies to the graph. It then goes though each node and finds the size of the adjacency list to make sure the correct numbers of adjacencies exist. It goes back through every node and checks the total weight of its adjacency list to make sure the weights are right. The priority queueTest method adds several nodes with different priorities out of order to the queue and checks the size before and after. The nodes are then dequed and checked for correct priority order. The dijkstra’s algorithm test runs the test we did by hand in class and prints out the paths found. They are visually checked against a comment containing the correct path.

These tests check the basic functionality of each major class alone using assert functions and then put together in the main Dijkstra’s algorithm class. Since the base classes themselves work independently, it will make the Dijkstra’s class itself work correctly.