How I Obtained the Solutions

I started off reading the assignment 4 pdf which gave me an understanding of how to work my way through the code since we just add methods to ClosenessCentrality.java. My code was based off reading and analyzing the test which helped gave me fundamental ideas on how this assignment is to be expected. From analyzing closeness centrality it takes the number of nodes minus one divide by all the shortest path. I was able to get my hands on a book called "Data Structures and Other Objects Using Java" 4th edition. This book had fundamentals of getting minimum distances using Dijkstra's Algorithm. I was trying out codes in order to make my code work in the process , but it took a lot of researching from online resources. I noticed there were mistakes in the table for the assignment but I was able to figure it out. The major idea of the assignment was to get the shortest distance of every node and the correct centrality. We were given 2 files for graphs and 1 file of interfaces. I did a lot of research online on Dijkstra's algorithm, usually geek to geek codes in order to find an idea to help me solve this project. CSI2110 Lab 9 helped me a lot too. It gave me numerous concepts to approach the code. Since the lab contained graphs where we had to implement a Dijkstra's algorithm in order for the source code to work. I watched a lot of YouTube videos in order to fully understand ways in order to identity the most influential person in a social network using the Closeness Centrality algorithm but the code ran for a long time with the right results. I learned closeness centrality measures the mean distance from a vertex to other vertices through a shared network. Closeness centrality keeps track of the mean distance. It takes number of nodes -1 over the sum of all of the nodes where only the sum varies over time while the node -1 usually is constant in term of closeness centrality. In the end I tried to run the email-dnc.edges input file, but the code was giving the right results in the debugging system but it took a really long run time. My major mistake was using Dijkstra's algorithm as Kruskal's algorithm and Prim's algorithm would have depth first search and breadth first search making the run time shorter and take up less memory. As my Dijkstra's algorithm was a never ending run time for the main source input but in the end it was the correct results.

A brief description of the classes and methods

The closeness centrality class calculates the more central the node is. The Closeness is a measurement of the node's capacity to effect all other elements in the network. The score is based on their closeness to all other nodes in the network. As the higher closeness value mean that the node is more central because it takes less steps in order to go from one node to another. Through graph analysis the closeness centrality detect nodes that are able to spread information efficiently. Unfortunately my code started off with zero instead of one but I was able to grasp an ideal code that was able to give the results that would satisfy the assignment. The code skips the first line of code which was 1,1,1 but at first it was confusing why the first value was 1,1,1 in the input but it all made sense while trying to debug the TestClosenessCentrality.java file.

Dijkstra's algorithm takes the list of adjacent nodes and it passes the Dijkstra's algorithm in all the list of the nodes. Adjacent is the edges in our case. Edges have a node and all the nodes are connected to the edges. We got a distance array which gave the minimum distance from one node to another node. Using this calculation I initialized the distance then I designed a HashSet which contains Integer and see if it was there in the HashSet or not. As HashSet makes no duplicates at all. At the beginning we initialize all the nodes to infinity (maximum value). After initializing it I create a node and add it to the priority

queue. An inner class was created in order to calculate the paths of every node. Sometimes the node comes back to the same queue and the priority queue checks if it's the same. Every time a node is process it is added to the queue and once it is process it is removed from the queue. Settled has the list of all the nodes that has been processed and has distance array which is all calculated. Array distance has all the elements in the array and gives off information meaning this node is settled and is done with entirely. The code grabs the file and checks the size of the graph which has all the distances of the current node. Started off with an empty priority queue. A CentralityMap (which is a new HashMap of type Integer and Double) was created to measure adjacent nodes of each nodes. Each node is a key of an array list and the calculation between the distance of every node. The code grabs the edges and the central nodes which every graph is pointing to. Loops through the all nodes from i=0 to all other nodes in the network. The result was satisfying for a graph that contains a short number of nodes.