1. adjacentVertices and edgeCost method both have a worst case run time of O(|E|); both only contains a for-each loop which runs over the collection of edges. shortestPath method has a worst case run time of O(|V2|) as it has a while loop with a nested for-each loop that run over the vertices.
2. Before moving on to part 2, we’ve created a test file to make sure the method we’ve written in part one are working properly. The test.java file includes adding a bunch of vertexes, and then adding a bunch of edges to connect the vertexes. Then we print out the graph to make sure it’s being constructed according how we’ve built it. We’ve also tested the adjacentVertices and edgeCost functions to make sure they are written properly. After finish testing part 1, we’ve start testing part 2 when it’s done. We’ve tested the graph by creating our own edge.txt and vertex.txt to make sure our program is able to read in different types of graphs. We’ve tested our method with random graphs, no path, and other types of graphs. We’ve also tested edge cases such as illegal arguments being passed in or non-existent vertexes being passed in as a parameter.
3. We have worked on part 1 of assignment individually and then we got together to compare the code each one has written. After we’ve decided which person’s code to go with for part 2, Stephen has taken the charge of finishing the ShortestPath method while Daniel helped with the write-up and the FindPath.jave file. We’ve spent around 2 hours planning, 5 - 7 hours coding, and then around 2 hours testing the code. One good thing about working together is that you can have another pair of eyes to look over each other’s code and give opinions. One bad thing is that for an assignment like this, it’s hard to balance the work load evenly between the two. There’s a tradeoff between collaboration, efficiency, and work distribution.