**Algorithms & Data Structures II Design Document**

**Shortest Path:**

For the shortest path I decided to use the Dijkstra algorithm to find the path from the source to the destination. Since we are only interested in the cheapest path from a single source, it is a suitable algorithm compared to Floyd-Warshall which looks for every path from every source which is unnecessary calculation for us. The time it would take to find the path is at a constant *E* log(*V*) for a typical case and the worst case as well and its space complexity takes up *V* space, so the wait time for the user will be similar regards the path the user will search for. Bellman-Ford was another choice I thought of, its space complexity is the same as Dijkstra but the typical and worst case time is *E V*. When implementing Dijkstra, I thought it would be cleaner to implement it by creating the Edge and Vertex as their own classes compared to my previous first time implementation of Dijkstra. I used an ArrayList to save and sort different values and inputs for this algorithm since it is easy to use in java and has good features that are useful for different operations. Its sort function uses either the Mergesort or the Timsort, both of which have the worst-case time complexity of nlog(n) which is pretty good considering the input files we are dealing with are very large. The ShortestPath class has 2 methods besides the Dijkstra calculation which is required for the project, one of them calculates the distance, which will be the cost, and the other one records and then prints the path it takes. That way it’s easier to calculate separate results from the ShortestPath class. For stop ID’s I decided to try out using one of the java features which is a HashMap function to store and retrieve bus stop ID’s since it has a constant time of O(1) to store the stop ID’s and worst case O(n) to get the stop ID’s.

**Bus stop name searching:**

**Printing the arrival times:**