Diarmuid McGonagle

19310779

Algorithms & Data Structures II

Algos Final Assessment Design Document

When mapping out this project, I had to make a lot of decisions about codebase structure, classes, & application flow, as well as the particular algorithm & data structure I would use to implement the core functionality. In structuring the codebase, I decided to have a Main.java file that runs the program. This Main.java file accepts user input which decides which function of the core functionality will be executed. The core functionality: ShortestPathBetweenTwoStops.java, SearchStops.java, & SearchStopTimes.java all have their own files & classes. These files also have helper functions for user input & input validation along with a ‘main’ method that executes the expected behavior.

When implementing ShortestPathBetweenTwoStops.java, I had to decide which shortest path algorithm to implement. I decided to implement Dijkstra’s Shortest Path. Because I was finding the shortest path to a given point, I believed that Dijkstra’s would be more efficient (& less complex) than A\*. Dijkstra’s Algorithm is also more memory efficient.   
To store the stop data, I used a Edge Weighted Digraph. Initializing this data had a runtime of N^2 & a memory usage of N+E. I used this data structure because it made the most sense to store the data in pairs, one for the value of the current node & the other for the distance from the origin point. The Function ‘Relax’ was used to calculate the ‘cost’ of each node along the way to the final destination.

When implementing SearchStops.java, I heavily followed Sedgewich & Waynes implementation of a TST. I created a TST class (TST.java) & divided the file between the core functionality (TST() & populating the tree) & the helper methods that are called when populating the tree.   
  
When implementing SearchStopTimes.java, I used the existing Edge Weighted Digraph I had already initialized to find stops that matched the time I was searching for. Because it was already initialized, I saved significantly on space complexity since I could use the same object for two methods.

For the user interface, I created the application in the console. I used the Scanner Java implementation for accepting user input & sanitized it on each input. If the input did not meet the proper runtimeType or was not in the correct bounds, I required the user to regenerate it.