Design Document

Name: Kevin O'Donnell
Student Number: 20334001

- A document explaining the design decisions (a choice of data structures and algorithms used to implement each of the 3 main features), justifying them based on specific space/time trade-offs between alternatives you have considered, considered in the context of the type and size of data you are dealing with.

Shortest Path Between Two Bus Stops

When beginning this part of project I had to decide to use either Floyd-Warshall, Djikstra or A* shortest path algorithm. I quickly dismissed using Floy-Warshall and so was left with the decision to choose between A* or Djikstra, a decision in which Djkstra came out on top.

Dijkstra's shortest path algorithm was chosen within the mindset of simplicity as well as effectiveness. My previous experience implementing Djikstra, deeper understanding of Dijkstra, the exclusion of negative weights form the data set, A* being more memory inefficient and requires more operations per node as well as the difficulty of finding a suitable heuristic to use for A* were all major factors in this decision.

Dijkstra has a worst case asymptotic running time of Dijkstra is O(ElogV) with |E| and |V| be the number of edges and the number of vertices in the graph respectively. Adding all vertices of the graph will take O(|V|) time, priority value updates will take O(|E| log |V|) time, calculating updated priority priority values takes O(|E|) time, each vertex is removed from the fringe once so O(|V|) times and for each run it takes O(|LogV|) time for a total of O(|V|Log|V|) time and iterating over all vertices neighbours takes O(|E|) time. Adding all of these running times brings us to our total of O(ELogV)time. This run time proves Djikstra to be superior to other shortest path algorithms like Floyd-Warshall and Bellman-Ford.

Djiksta also has a solid space complexity of O(V), meaning it is efficient in both space and time complexities, further backing up my reasoning for using this algorithm. This space complexity fairs well against other shortest path algorithms as it is the same as Bellman-Fords while Floyd-Warshall's space complexity is O(n^2).

I also made use of a hashMap as an array was not possible due to the large volumes of data. The extra space used for the hashMap was O(E).

Bus Stop Search

In the case of searching via bus stop I had little choice in the data structure I used. It was specified that we must use a ternary search tree (TST). I implemented the TST.java class as found from the princeton library while adding a hashMap. I added the hasMap solely for reasons of simplicity so I could easily grab any information needed through the .get() function. The extra space used for the hasMap was O(N), N being the number of stops from stops.txt. The TST has a worst case time complexity of O(N) when inserting or deleting a node and an average case of O(logN). I also created a LinkedList of all stops with a matching name to that of the input, this added a cost of space O(N), where N is the number of stops matching the input.

Arrival Time Search

To tackle this problem I created an arrayList of type stopDetails where I could store all the stops with a matching arrival time to that of the inputs information. After taking in the input I then compare all arrival times in stop_times.txt to the input, and if they match I add the stop information to my arrayList. This arrayList has a space complexity of O(N), where N is the number of matching stops. The time complexity to add said stops is O(N-1) also, where N is the number of lines in stop_times.txt.

After this I failed to implement Collections.sort (quicksort). I attempted to implement this to sort our stops by their trip ID. At first I thought merge sort was most suited as it has a guaranteed worst case runtime of O(NLogN) compared to quicksorts O(N^2). Merge sort was optimum to use for sorting an array of objects. It also has a linear space complexity of O(N). However, quicksort is the most efficient in its average case and requires little space, for this reason it was best to try and implement.

User Interface

I had planned to create a graphical user interface but due to the workload of the project as well as workload outside the module I did not have time to implement it. Hence, I have implemented a simple console/command line interface.