Final Project Design Document

Miguel Arrieta

Overview

I chose to make the following decisions for implementing the Data Structures and Algorithms and functionality required.

Graph class

I needed a Graph class to store the bus network of Vancouver. This graph contains weighted, directed edges as required in the specification. I decided to make this graph generic, as I could worry about the implementation of the actual graph first and then I could worry about using it specifically for the bus network.
A connection record implementing the Comparator interface was used to represent edges in the Graph. It implements the Comparator interface so that it can then be used with a PriorityQueue when finding the shortest path with dijkstra.
I used an adjacency map/list instead of an adjacency matrix to conserve memory. I implemented this using a Map, specifically a HashMap, since bus stops aren't just increasing from 0 up to a certain number, they seem random, so a HashMap still provides O(1) lookup times, like an array.

Shortest Path Algorithm

The project required the user to be able to receive the shortest path between any two stops in the bus network. I chose Dijkstra's shortest path algorithm, since I didn't have enough information to create a heuristic function for A*.
Floyd-Washall would have been slower and also doesn't record the path taken, so it wasn't an option.
I decided to implement Dijkstra with a priority queue, returning a Path object which contains the path taken and the total distance covered. The method is **Graph::getShortestPath**. The Bus Network class then uses this method in **BusNetwork::getShortestPath**.

Ternary Search Tree Class

A ternary search tree class was needed to store the names of all stops and all other stops associated with that given order of letters.
TernarySearchTree::insert takes in the stop name as a String and the stop ID as an int.

This then adds this stop ID to all the matching Nodes' lists (Node is an auxiliary class), meaning that when the user searches for this same name or partial name, this stop ID will be present in the return list by **TernarySearchTree::search** which takes in a string.
The Bus Network class then uses this method in **BusNetwork::getMatchingStops** and converts the List of stop ID's to a List of Bus Stops, by using a Map mapping Stop ID to Bus Stop.

Searching for Arrival Times

I decided to have two classes for this, the actual Trips, which just stores a map mapping Trip ID to a list of TripSegment's, a class which models each individual Trip segment. I used a TreeMap so that the trips would be sorted by Trip ID.
I needed to sort each list in this map by arrival time, so I made a Comparator object for this in the Trips class, which I then used for using **List::sort**, a stable, adaptive, iterative mergesort to provide fast, stable sorting, just in case something wasn't sorted .
I then also used this object to use **Collections::binarySearch** for fast searching through each list, searching by arrival time, providing O(log(n)) search time as opposed to linear search's O(n) time.

Reading in files

I chose to use a BufferedReader wrapping an InputStreamReader as this tends to be much faster than using the Scanner class for reading in information. All files are read in a very similar way, feeding in the inputs given by the file for each line as Strings into the respective constructor (TripSegment or BusStop) and letting the constructor deal with most of the error handling and validation.
For reading in the transfers it is very similar except I just extract the information from each line of transfers.txt, parse it accordingly and add it as an edge to the Bus Network's graph with **Graph::addEdge**.

User Interface

For the UI, I decided to use a command line interface, with ANSI colouring, so that key information stands out. ANSI colouring however isn't supported on many consoles on Windows so there is error checking for checking if the current console supports ANSI, there's also a program argument that can be passed to turn off colours or force them if the user thinks that their console supports ANSI but the program isn't detecting it.

The user is asked to wait as the program loads and is then met by the program automatically running the help command, telling them what commands are available and how to use them.
All the functionality outlined in the assignment specification is available.