Translink Journey Planner Implementation

Rían Errity

April 10, 2022

Abstract

This project concerns the implementation of a "management" system for Vancouver's Translink transportation network. In reality, this is a simple Journey Planner in which a user can:

1. Look up information pertaining to a stop by Name or ID.
2. Find a route, including transfers between any two connected stops.
3. Search for all stops which have an arrival at a specified time.

Chapter 1

Tooling, Implementation and Execution

1.1 Tooling

This project was implemented in Java with the use of Gradle, a dependency management and build tool. This necessitated a different project structure than one would be used to if they were not already familiar with the larger Java ecosystem, or without prior exposure to build tools such as Apache Maven or Ant. This particular Gradle project is making use of the Kotlin DSL as opposed to the Groovy DSL.

Development was performed in Intellij IDEA Ultimate, hence the presence of `.idea/` and `*.iml` within the `.gitignore` file. These ignore Intellij-specific workspace configuration files, which are unnecessary to commit. Likewise, I have ignored the Gradle wrapper, to maintain version independence, and the results of Gradle builds, usually contained within the `build/` sub-directory.

1.2 Implementation

This project has been implemented as one cohesive project with two disjoint interfaces, one with a graphical user interface implemented in JavaFX, which is incomplete at the time of writing as well as a command-line REPL [1]

This project required some slight modifications to vanilla Gradle, namely the need to attach STDIO [2] to the `gradle run` task as well as the addition of the JavaFX Gradle plugin which includes some of the runtime artifacts which JavaFX requires but which are no longer included in the JVM following Oracle's open sourcing of the project post-Java-1.8.

1.2.1 Parsing

A major aspect of this project was simply the parsing of the unsanitised and inconsistent comma-separated value (CSV) files provided as input to this project. As it was such an important task, this is what I decided to tackle first — I created a standalone class `CSVParser.java` for this purpose which now contains methods to

1. Read in a file as a list of Strings, and split them by comma to create a 2D Array containing all fields.
2. Parse the individual fields into their appropriate fields into their appropriate POJO [3]

There ended up being a little more work to be done on this file after the fact, particularly due to the sanitation requirements which I neglected when I initially wrote the class, including the need to trim additional white-space around the time parser, and some fields which aren't always present. In the case of a field which isn't always present I simply reverted them to a default, invalid value (such as -1.)

1.3 Execution

This project can be executed in one of two ways, both of which are outlined in `README.md`. The former being to use the aforementioned `./gradlew run` command or to build a portable Java Archive (JAR) file, which can then be ran anywhere through the use of `java -jar MyJar.jar` provided there is a JVM on `$PATH`.

The program respects the presence of a command-line flag (`-nogui`) which reverts the program into using the REPL-environment mentioned earlier, alternatively this can be enabled through the use of the `nogui=true` environment variable. The REPL is the first-class experience at the time of writing due to a lack of time to implement the graphical user interface, as I alluded to earlier.

[1] Read-Evaluate-Print Loop is an interactive command line application in which a user is prompted for input which is then parsed and a result is printed, much akin to a shell environment.

[2] Often referred to as simply `System.in` in Java

[3] Plain Ol' Java Object, a term used to refer to a dummy object whose sole purpose is to hold structured data.

Chapter 2

Design Decisions

2.1 Feature 1: Shortest Path

2.1.1 Description

The shortest path feature is accessible in the REPL-envoirnment through the use of the `journey` command which accepts two stop ID numbers [1].

This command was implemented using Dijkstra's shortest-path algorithm. An algorithm which we both covered in class, and previously implemented for a past assignment, meaning I was both familiar with it and aware of its costs and limitations. I suspect there was enough information to make use of the A* shortest-path algorithm, with both directions (given as prefixes in stop names) as well as the geographical co-ordinates of the stops, however I found it was more useful to have a working implementation rather than wasting what time I had left on a small chance of an alternative algorithm being implementable.

The algorithm operates on a HashMap of Integers mapping to a list of StopNodes, whereby Integer represents the ID of a given stop and the list of StopNodes represents a simple to-from-cost structure containing all of the edges from the stop represented by the map key

The algorithm has been modified somewhat to provide both a queue of stops it has visited, as well as the total cost it took to reach a given point. The cost is stored as a field within the class due to a limitation in which Java cannot natively return multiple values.

2.1.2 Complexity

The Time Complexity of Dijkstra's Algorithm is $O(V^2)$ where V is the number of vertices / stops.

2.1.3 Example

```
journey-planner $ journey 3053 12235
+---------+-----------+---------------------------------------+--------------------------------
| stop_id | stop_code | stop_name                             | stop_desc
+---------+-----------+---------------------------------------+--------------------------------
| 3053    | 53021     | COQUITLAM CENTRAL STN BAY 3           | COQUITLAM CENTRAL STN @ BAY 3
| 8032    | -1        | COQUITLAM CENTRAL STN WESTBOUND       | WEST COAST EXPRESS @ COQUITLAM
| 12235   | -1        | COQUITLAM CENTRAL STATION PLATFORM 2  | SKYTRAIN @ COQUITLAM CENTRAL ST
+---------+-----------+---------------------------------------+--------------------------------
This had a total cost of: 3.2
```

The full table was too wide for this document. You can view a full raw version HERE

2.2 Feature 2: Stop Search by Name

2.2.1 Description

Feature 2 requires a user to be able to search for stop information by only providing a partial of the name of the desired stop, which the program should then provide the stop information for all stops which may be the desired stop. This is implemented using a Ternary Search Tree as suggested in the specification. As stops are parsed their names are added to a Ternary Search Tree as well as a HashMap which maps names to the rest of the Stop meta data, this allows constant time lookup of the remainder of the stop metadata which is presented in AsciiTable form.

2.2.2 Complexity

The complexity of a TST Search is considered to be $O(\log n)$

[1] Stop IDs are among the information which can be obtained through the use of Feature 2 as described in §2.2

2.2.3 Example

```
journey-planner $ lookup POWELL ST FS
+---------+-----------+--------------------------------+----------------------------+-----------
| stop_id | stop_code | stop_name                      | stop_desc                  | stop_lat
+---------+-----------+--------------------------------+----------------------------+-----------
| 435     | 50431     | POWELL ST FS CLARK DR EB       | POWELL ST @ CLARK DR       | 49.283029
| 38      | 50038     | POWELL ST FS COLUMBIA ST WB    | POWELL ST @ COLUMBIA ST    | 49.283285
| 437     | 50433     | POWELL ST FS COMMERCIAL DR EB  | POWELL ST @ COMMERCIAL DR  | 49.283885
| 512     | 50508     | POWELL ST FS COMMERCIAL DR WB  | POWELL ST @ COMMERCIAL DR  | 49.283837
| 516     | 50512     | POWELL ST FS GLEN DR WB        | POWELL ST @ GLEN DR        | 49.282792
| 520     | 50516     | POWELL ST FS GORE AVE WB       | POWELL ST @ GORE AVE       | 49.283223
| 517     | 50513     | POWELL ST FS HAWKS AVE WB      | POWELL ST @ HAWKS AVE      | 49.28303
| 518     | 50514     | POWELL ST FS HEATLEY AVE WB    | POWELL ST @ HEATLEY AVE    | 49.283076
| 519     | 50515     | POWELL ST FS JACKSON AVE WB    | POWELL ST @ JACKSON AVE    | 49.283146
| 436     | 50432     | POWELL ST FS MCLEAN DR EB      | POWELL ST @ MCLEAN DR      | 49.283237
| 439     | 50435     | POWELL ST FS VICTORIA DR EB    | POWELL ST @ VICTORIA DR    | 49.284771
| 500     | 50496     | POWELL ST FS VICTORIA DR WB    | POWELL ST @ VICTORIA DR    | 49.2848
| 513     | 50509     | POWELL ST FS WOODLAND DR WB    | POWELL ST @ WOODLAND DR    | 49.283481
+---------+-----------+--------------------------------+----------------------------+-----------
```

The full table was too wide for this document. You can view a full raw version HERE

2.3 Feature 3: Stop Search by Time

2.3.1 Description

This feature is based on *stop_times.txt* which is parsed by CSVParser.txt before being loaded into an ArrayList in BusNetwork. This is then probed by looking for the equality of LocalTime objects. This equality is an integer comparison which can be significantly faster than string equivalence, it is also a semantic search, as we're searching for the actual values rather than a string representation of them (which would require separate handling of 05:12:05 and 5:12:5 for example)

This feature is accessible in the REPL-environment through the use of the `timesearch` command. It takes a single parameter which is an HH:MM:SS time representation.

2.3.2 Complexity

This requires a search of the entire List<StopTime> registry, which contains 1.3m records with the given example data. The registry is sorted by trip id, though so it does not require further sorting. $O(N)$

2.3.3 Example

```
journey-planner $ timesearch 9:17:21
+-------+----------+-----------+
| ID    | Arrival  | Departure |
+-------+----------+-----------+
| 30    | 09:17:21 | 09:17:21  |
| 1281  | 09:17:21 | 09:17:21  |
| 11087 | 09:17:21 | 09:17:21  |
| 1356  | 09:17:21 | 09:17:21  |
| 11049 | 09:17:21 | 09:17:21  |
| 291   | 09:17:21 | 09:17:21  |
| 1999  | 09:17:21 | 09:17:21  |
| 1865  | 09:17:21 | 09:17:21  |
| 2215  | 09:17:21 | 09:17:21  |
| 10479 | 09:17:21 | 09:17:21  |
| 12257 | 09:17:21 | 09:17:21  |
| 3168  | 09:17:21 | 09:17:21  |
| 3168  | 09:17:21 | 09:17:21  |
| 4251  | 09:17:21 | 09:17:21  |
| 5516  | 09:17:21 | 09:17:21  |
| 5844  | 09:17:21 | 09:17:21  |
| 5311  | 09:17:21 | 09:17:21  |
| 5525  | 09:17:21 | 09:17:21  |
```

```
| 10757 | 09:17:21 | 09:17:21  |
| 6854  | 09:17:21 | 09:17:21  |
| 6953  | 09:17:21 | 09:17:21  |
| 6354  | 09:17:21 | 09:17:21  |
| 5570  | 09:17:21 | 09:17:21  |
| 11828 | 09:17:21 | 09:17:21  |
| 4170  | 09:17:21 | 09:17:21  |
| 948   | 09:17:21 | 09:17:21  |
| 4675  | 09:17:21 | 09:17:21  |
+-------+----------+-----------+
```