Vancouver Bus Application

Ailbhe McEvoy - 19334654
Conn Breathnach - 19333813
Dónal Heelan - 19333466
Melissa Mazura - 19335692

Introduction

The purpose of this application is to provide the user with functionality that will allow them to quickly and easily determine key characteristics of the Vancouver bus system. This functionality includes:

1. The ability to find the shortest path between 2 bus stops
2. The ability to search for a bus stop by name (or by the first few characters of the name)
3. The ability to search for all trips with given arrival time
4. A command-line menu interface that allows for easy access to modes

Design Considerations

The end-goal is to create a product that satisfies the aforementioned requirements before the allocated deadline (30th April 2021 at 23:59pm UTC). In order to complete the task on time, the following division of labour has been allocated:

Roles and Responsibilities

- Conn Breathnach: Implementation of task 1 entailing the finding of shortest paths between 2 given bus stops, dealing with user input as appropriate and returning the list of stops along the resulting route along with the associated cost.
- Melissa Mazura: Implementation of task 2 by creating a bus stop search system which utilises ternary search trees to efficiently return the stop information of each stop which matches the user's input, handling edge-case user input where appropriate.
- Ailbhe McEvoy: Implementation of task 3 by creating a trip search mechanism which will allow the user to search for specific trips on the basis of their arrival time. The resulting trips should be returned sorted by ID and user input should be allowed only in hh:mm:ss format only.
- Dónal Heelan: Design front interface which allows the user to select and run one of the aforementioned programmes as well as allowing the user to quit the application should they wish. Creation of design document satisfying project requirements.

System Assumptions

The successful running of the application presumes the following:

- System has either Java Runtime Environment (JRE) or Java Development Kit (JDK)
- GPU requirements: Minimal due to command-line implementation
- RAM requirements: Low minimum required but will impact algorithm runtime

Architectural Strategy and Development Methods

In each of the 3 main sections of the application, specific data structures and algorithms were employed to ensure optimality in both time and space:

1. Shortest Path: Dijkstra's algorithm was implemented to find the shortest path between the source node and every other node, resulting in a shortest-path tree. A minimum-priority queue was used to implement the algorithm which results in a space complexity of O(E log V) and a time complexity of O(V+E log V), where E is the number of edges in the graph and V the number of vertices
2. Bus stop search by name: TST was implemented to search for bus stops. This tree is more space efficient than other Tries as it does not store 26 pointers for each letter. It is also well suited to having a lot of entries / a big height. A recursive algorithm was used to find the bus stops and to initialise it. The time complexity is O(n) for the search itself and the initialisation of the tree.
3. Search for trips by arrival time: An ArrayList was used to store the Trip objects that matched the arrival time given by the user. This allowed less restriction in the size of the ArrayList as the size depends upon the user input arrival time. This arrayList was sorted using timsort. A comparator interface was created which allowed for custom ordering of the Trip objects by trip ID. The overall time complexity to implement this is O(NlogN) and the space complexity is O(N).

Challenges and Solutions

1.

- **Task**: Trying to create a graphical user interface (GUI) as the application
- **Challenge**: Java's GUI frameworks e.g. AWT & Swing, were found to be slow and awkward to program with. What was supposed to streamline the user experience ended up being cumbersome to use and implement.
- **Solution**: A command-line interface was instead implemented which, when combined with appropriate error-checking to prevent the user from inputting illegal values, allowed for a far more straightforward and pleasant UX.

2.

- Task: Read in files and create graph based on inputs.
- Challenge: Linux and Windows have different file endings, so using \n as a delimiter caused problems on Windows machines
- Solution: Used \r for return character as delimiter when reading in txt files.

3.

- **Task:** Search for a bus stop in TST.
- **Challenge:** Finding the right algorithm to search for a bus stop with only part of the name in the TST.
- **Solution:** Dividing the bus stop search into two parts: first one: looking if the given prefix even exists in the TST. If it exists but the input is longer, call another function that returns all words from that node.

4.
- **Task:** Search for all trips with a given arrival time and return all details of all of the trips.
- **Challenge:** Parsing the time from a string and formatting inconsistencies within the stop_times.txt.
- **Solution:**

The input arrival time is compared as a string with the arrival times from the text file. The input arrival is parsed to Date in order to make sure that is a valid time (i.e. lower than the maximum time 23:59:59).
The arrival/departure times in the stop_times.txt file are separated by spaces before the previous commas while the other trip details in the line are separated by only commas. A space was added to the user input string in order for it to be compared accurately with the arrival time in the file.

Appendix

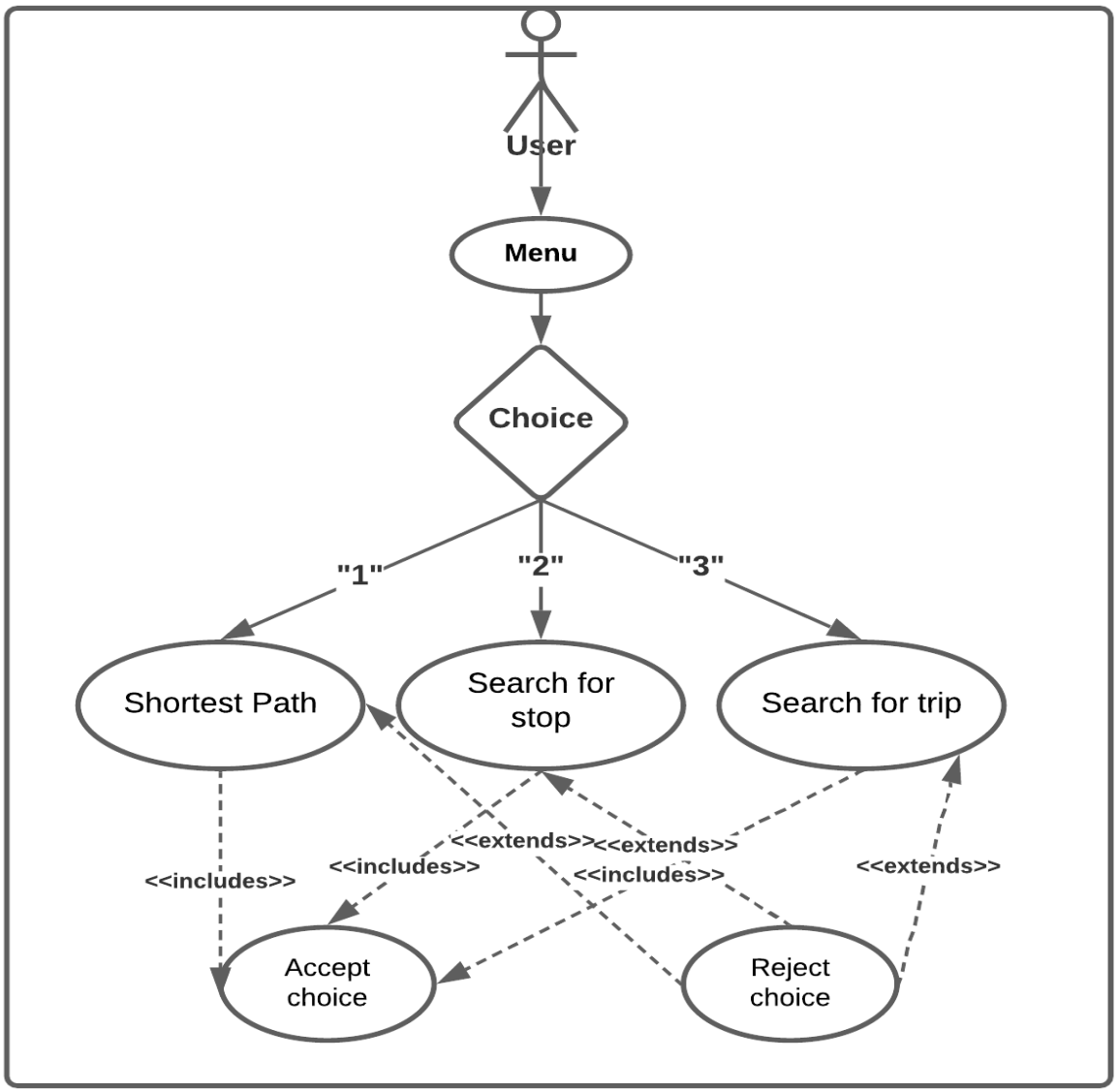

Fig 1. UML Use Case Diagram

-Code acquired from Princeton book website here