Group7

COMP 1828 - Designing, developing and testing a journey planner model for the London Underground system

Team members:

Name: Jubril-Awwal Shomoye - 001066247

Name: Arbaaz Sheikh - 001086906

Name: Kevin Luu - 001063342

Name: Phakonekham Phichit - 001041931

1. “HOW TO guide” for a customer and any assumptions made

1) Prerequisites before running program (INSTALLATION)

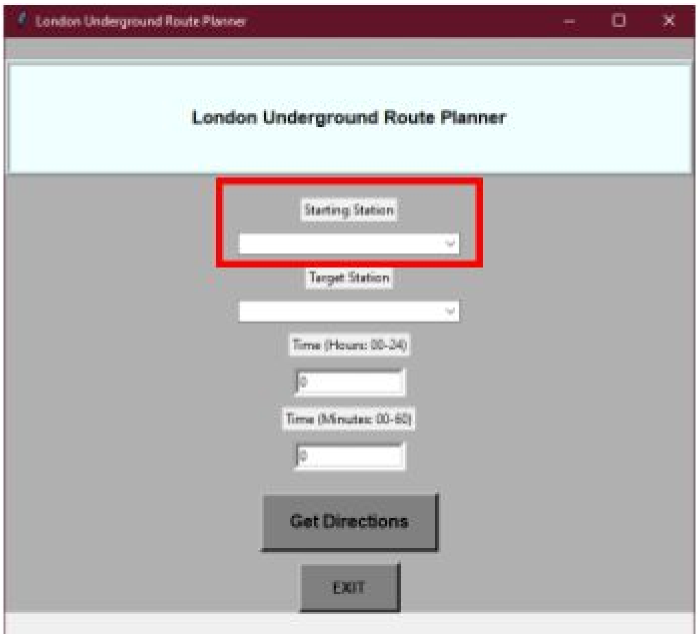
Ensure that the python version installed on your device is 3.8.x

If you do not currently have the latest version of Python you can install the updated version by visiting the website https://www.python.org/downloads/

Once the program has been successfully updated you will then be able to proceed with the running of the program.

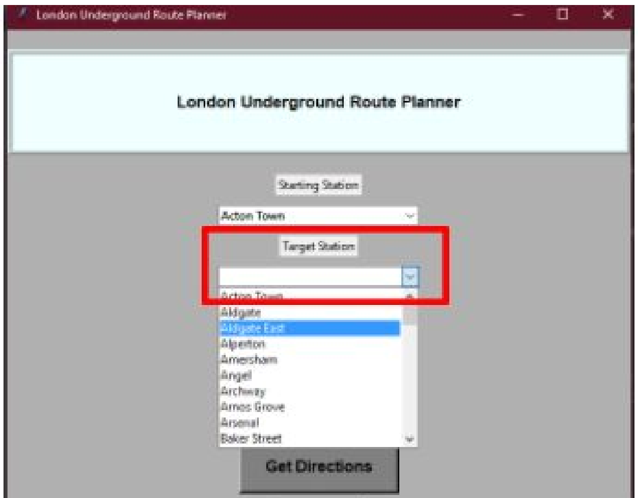
2) Select the starting station

The journey planner program has been preloaded with all the London Underground Train Stations including the different lines such as: Bakerloo, Central, Northern, Circle, District, Hammersmith & City, Jubilee, Metropolitan, Piccadilly, Victoria and Waterloo & City Line. This is to help you find the quickest route to your destination.



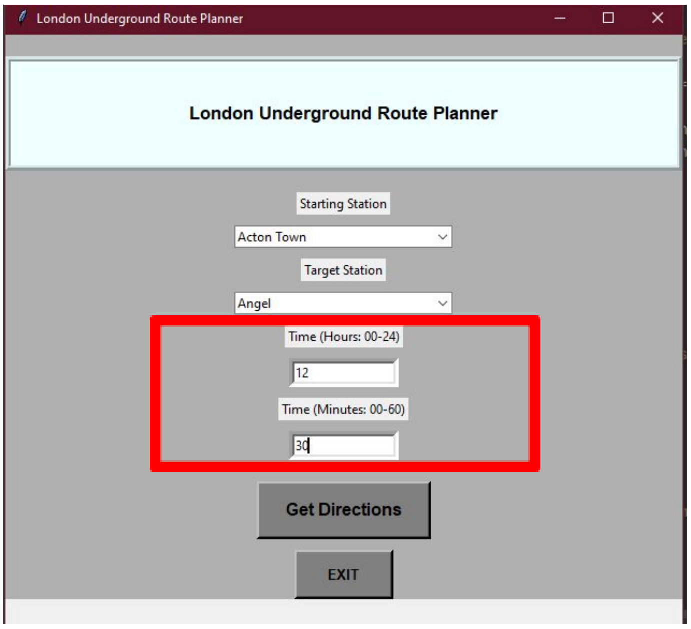
Click the starting station box with the arrow and a dropdown menu will appear with a list of all the stations sorted in alphabetical order. Sorting the stations in alphabetical order allows you to sort through the list quickly so you don't need to spend time filtering through to find the correct station.

3) Select the target station



Once the starting station has been selected, the target station should be selected and the same dropdown menu will appear with a list of all the stations, where you can then select the destination station you'd like to travel to.

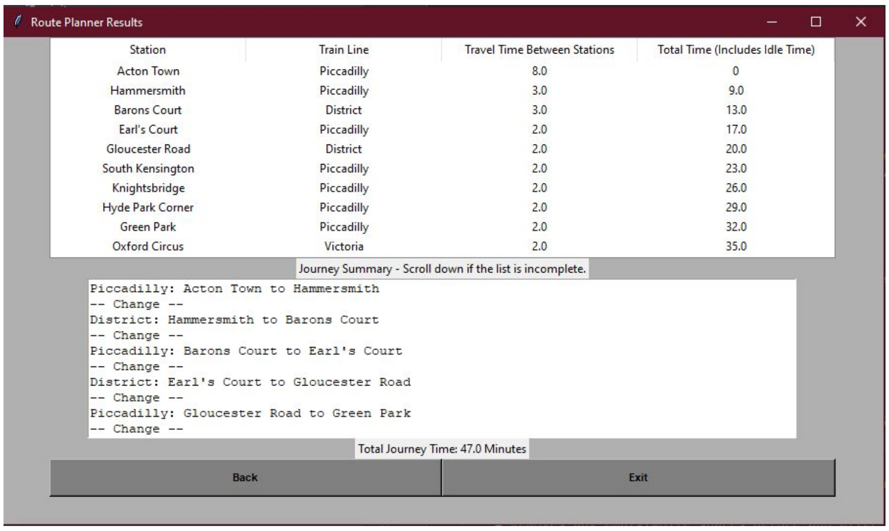
4) Enter the departure time


London Underground Route Planner
London Underground Route Planner
Get Directions
EXIT

Starting Station: Acton Town
Target Station: Angel
Time (Hours: 00-24): 12
Time (Minutes: 00-60): 30

The departure time has been split into a separate 'hour' box and a 'minute' box. The hours are from 00-24 and minutes are from 00-60 so if you would like to travel at 13:30 you'd enter 13 in the hour box and 30 in the minute box. Train lines have different timings throughout the day, entering the time will allow for the journey planner to find the next available train and will inform you on how long it takes to reach the destination.

5) Get directions



Station	Train Line	Travel Time Between Stations	Total Time (Includes Idle Time)
Acton Town	Piccadilly	8.0	0
Hammersmith	Piccadilly	3.0	9.0
Barons Court	District	3.0	13.0
Earl's Court	Piccadilly	2.0	17.0
Gloucester Road	District	2.0	20.0
South Kensington	Piccadilly	2.0	23.0
Knightsbridge	Piccadilly	2.0	26.0
Hyde Park Corner	Piccadilly	2.0	29.0
Green Park	Piccadilly	2.0	32.0
Oxford Circus	Victoria	2.0	35.0

Journey Summary - Scroll down if the list is incomplete.

```
Piccadilly: Acton Town to Hammersmith
-- Change --
District: Hammersmith to Barons Court
-- Change --
Piccadilly: Barons Court to Earl's Court
-- Change --
District: Earl's Court to Gloucester Road
-- Change --
Piccadilly: Gloucester Road to Green Park
-- Change --
```

Total Journey Time: 47.0 Minutes

After you have selected the time, you click on the 'Get Directions' button and the 'Route Planner Results' window opens up. This shows you how to get to the target station from the start station and what line to take, the travel time between the stations and the total time taken from each station which includes idle time. The journey summary is located at the bottom of the screen and provides a list of what stations to take and in what order to take them in. The 'Route Planner Result' window also displays the total time required to travel from the start station to the target station in minutes.

Assumptions

Along with the current assumptions taken into account when using dijkstra's algorithm, we have added a few more assumptions in our journey planner program.

One assumption we have made is that ‘if your journey requires a train swap, the next train will already be ready for you when you reach the platform’. Another assumption that has been made was ‘after reaching each station, the trains will be idle for one minute before departure for passengers to embark and disembark the train before moving off’. Lastly, the app is optimised based on time, so there will be instances where you may need to switch several times. As a result of this, our final assumption is that ‘there is no time required to switch between trains for the passengers’. This is to ensure that the trains are running as time effectively as possible and there are little to no delays in the train line service, which means that passengers that are using the train will be able to reach their destination station at their earliest time possible.

2. Critical evaluation of the performance of the data structure and algorithms used

The finalised version of the journey planner system mainly consists of using the Dijkstra algorithm and doubly linked list as the data structure. Furthermore, in this analysis of the program, will be discussed the comparisons of data structure to the chosen implementation, Big O notation performance analysis of the algorithm and data structure, and the relative overall performance of the system.

Data Structure: Doubly Linked List

The data structure of the program, the doubly linked list, is used to create nodes using each station to travel to the designated stations whether it is the next station or the previous one in a linear manner . In addition, the implementing the doubly linked list also has its benefits for the dynamic and flexible expansion whether is the ability to add and remove elements from a specific location due to the nature of the data structure. This includes from the front, end and the middle of the list robustly. In contrast, traversing to the previous node is incapable if done by a normal array. In addition, in order to insert or delete elements, it has to traverse the entire list for it to find the required element. This can be explained through the comparison of the complexity of each data structure.

The complexity analysis of Big O notation for the running time complexity on an array when an added or removed element is O(n) on its average and worst cases. While for doubly linked lists, it is O(1) since it can be added in certain places without the need to traverse. However, in accessing or searching for an element, a normal array has a better running of O(1). In contrast, a doubly linked list has to traverse in a linear manner of O(n). As for now, using a doubly linked list is more preferable for the data structure. Moreover, in order to prevent the worst case of traversing within loops, the doubly linked list is satnised and sorted. Overall, the running complexity performance for a doubly linked list is O(n) when accessing and finding stations.

Algorithm: Dijkstra Algorithm

The core algorithm of finding the shortest path from chosen station to designated station is done through using the Dijkstra algorithm. The algorithm is used to check all neighboring stations of the chosen station and update their values until the shortest path is found. The implementation of the algorithm uses mainly doubly linked lists for the data structure, in order to achieve the required output. The doubly linked lists are used to store the previous information of the stations which are the distance, line and previous stations that are up to date. In addition, the doubly linked list is implemented for the stations to traverse to the stations and containing all the information of the stations, line and time. The running time complexity of the algorithm is O(n^2) due to the stations being traversed within the two loops in order to find the shortest route. Therefore, it may not be optimised for larger size but it is able to output the requirement results. Overall, the running time of the doubly linked list is O(n) + O(n^2) of the dijkstra algorithm, concluding the complexity of O(n^2).

3. Discussion for the choice of test data you provide and a table detailing the tests performed

Before ensuring that we have finished the software, we performed a series of tests. These tests will show us if the software has reached its requirements or not. The software must be able to:

- Verify and prompt the user to input a start and a final station
- Output a path from station to station
- Output a path from one side of the underground map to the furthest side
- Validate if the time inputted is an integer and not a string or float
- Verify that the starting station and final station are not the same
- Cutting travel time by half based on the time for the Bakerloo line

The software must be able to perform all these tests to reach the requirements needed for the software to function correctly. The test table below will present whether the software has passed the tests.

<u>TEST TABLE</u>

Test Name	Test Type	Expected Outcome	Actual Outcome	Pass/Fail?
Testing each train line from start station to final station (central line)	Unit Test	Program will output route from West Ruislip to Epping and will list out all stations in between	Program has outputted complete route and listed all the stations in order from the start of the central line at West Ruislip to the end at Epping	PASS
Testing station from one side of the underground map to the other	Unit Test	Program will output a route from one side of the underground map to the other e.g. Amersham-Upminister.	Program outputted the complete route from Amersham to Upminister going through the Metropolitan line, Circle Line, Central Line, Jubilee, Hammersmith & City line and the district line in order	PASS
Testing user input handling e.g. if string is entered instead for time	Unit Test	An error message pops up alerting the user that the 'Time' must contain a positive numeric value ONLY to be able to continue	When a string was entered into the time box(s) a message has popped up altering the user that only positive numeric values can be entered into the time box	PASS
Testing error handling when start station and destination station are The same	Unit Test	An error pops up alerting the user that the entries for the start and destination station are both the same value. Will not process a journey planner until rectified	Error pops up altering the user that the start station and the destination station are the same and will not proceed to generate a journey planner for this trip until the stations are not identical	PASS

Testing what happens to the travel time between stations on the bakerloo line between times 9-4pm and 7pm-midnight	Unit Test	During any time between and including 9-4pm and 7pm to midnight the travel time on the bakerloo line should be twice the speed which means the travel time should be cut in half ONLY on the bakerloo line at these specified times.	Travel time on the bakerloo line was cut in half during the specified times 9-4pm and 7pm to midnight and anytime around that the specified times the speed is normal and is not twice the times of a bakerloo line journey	PASS
No input entered for starting station, ending or times	Unit Test	A message occurs when no values are entered in the entry boxes and prompts the user to input a starting and ending station	A message occurs alerting the user that a starting station and final station has not been entered	PASS

As shown from the test table above, the software was able to pass every test. This means that our software has met all the requirements and is able to function correctly.

4. Individual contribution by each team member and reflection

Name	**Allocation of marks agreed by team (0-100)**
Jubril-Awwal Shomoye	100
Arbaaz Sheikh	100
Kevin Luu	100
Phakonekham Phichit	100

Reflections:

Jubril-Awwal Shomoye

My contribution to this project is creating the input page for the gui. I used entry boxes, dropdown boxes and buttons to allow the user to enter in their starting and ending station and the time they started the journey. I created a test plan for the software, to ensure that it had met the requirements. Arbaaz had assisted me in completing the test table and ensuring the software had passed all the requirements. If there was any problem with the Dijkstra's algorithm, we would quickly figure out a solution.

Arbaaz Sheikh

Main contribution to the project was creating the double linked list using all the stations and the corresponding station next to them, both previous station and next station on the line. With the assistance of the lecture slides provided from session 2-4 linked lists I was able to complete this task. Assisted with team members to clean data on the underground spreadsheet as there had been errors in timing and names of the stations. In addition to the double linked list, I had constructed a test table in order to thoroughly test the program to ensure that it met the project's criteria. The how to guide had been created by me with the help of the entire team and was tested to ensure that any user would be able to follow the instructions and receive a complete route planner for their journey. If I had approached the task differently, I would have added a tail index pointer, as the program would not have to start at the beginning of the list and move from element to element until it reaches the end, but you would be able to go directly to the tail pointer and add nodes from there.

Kevin Luu

My main contributions to this projects are:

- Dijkstra's Algorithm (with the use of Doubly Linked Lists)
- The importation of the data from the excel document.
- (Partial help with) Doubly Linked List
- The journey summary output on the results page.
- The updated Bakerloo train times depending on the time given by the user.
- Putting everyone's parts/code together and commenting on the project.

Dijkstra's Algorithm posed the biggest difficulty I had throughout this project because of the requirements of this task. There was little to no information regarding the implementation of Dijkstra's using Doubly Linked Lists so I had to refer to Cos (our module leader) to get some guidance on how to attempt this. If I had more time to make improvements, I would've preferred to optimise the algorithm further since it's not the most efficient even though it works.

Phakonekham Phichit

Main contribution on the output page of the graphic interface,writing the evaluate the performance analysis on the data structure and algorithm and sanitised the spreadsheet data. In this coursework, I have shown the ability to implement the resultant outcome of the underground route planner. This is by creating tables and buttons to display an user friendly interface of the journey including the stations, line and travelled time. As for the report, it consists of a deep critical analysis of the Big O analysis and run-time complexity relating to data structure and algorithm used by the system.I rechecked and sanitised the spreadsheet data through the latest tfl underground working timetables. As for my reflection, the summary was not implemented as well as predicted due to the summary being unable to display a shorter journey distance of the line and stations.

References

- COMP1828 Module - Data Structures I (Linked Lists)
- COMP1828 Module - Searching (Dijkstra's Algorithm)

Big-O Cheat Sheet

Eric, R., 2020. *Big-O Cheat Sheet.* [Online]

Available at: https://www.bigocheatsheet.com/ [Accessed 11 11 2020].

TFL underground timetable

https://tfl.gov.uk/corporate/publications-and-reports/working-timetables [Accessed 9 11 2020]

Dijkstra's Algorithm References:

- https://rosettacode.org/wiki/Dijkstra%27s_algorithm#Python
- https://startupnextdoor.com/dijkstras-algorithm-in-python-3/
- https://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/