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

Team members:

Name: Lakshman Thillainathan

Name: Mahdi Rahman

Name: Zarin Tasnim

Name: Sreeraj Nair

1. “HOWTO guide” for a customer and any assumptions made (max 3 pages) [10 marks]

The increasing demand for the public transport makes it difficult to plan a journey which is when our new Journey planner comes in hand as It makes planning your journey simple and easy. This app provides the user with tube times in a user-friendly manner, making travel easy with all the information the Londoners need which is our top priority for this Journey planner.

The transport network is central to life in our city which is why we are helping the public travel safe and sound as we equip the public with correct information they require to make their safe journey choices a head of time , whether it might be for going to work on time or leisure. Moreover, planning your journey ahead helps to reduce the traffic and time of you need to wait for a train as will be aware of the information in advance.

Furthermore, our journey planner determines the quickest journey time for a planned route to travel in London Underground for the customers. It provides the Underground lines, the travelling distance, interchanges between stations and total time taken to reach the destination station. The assumptions made are as follows:

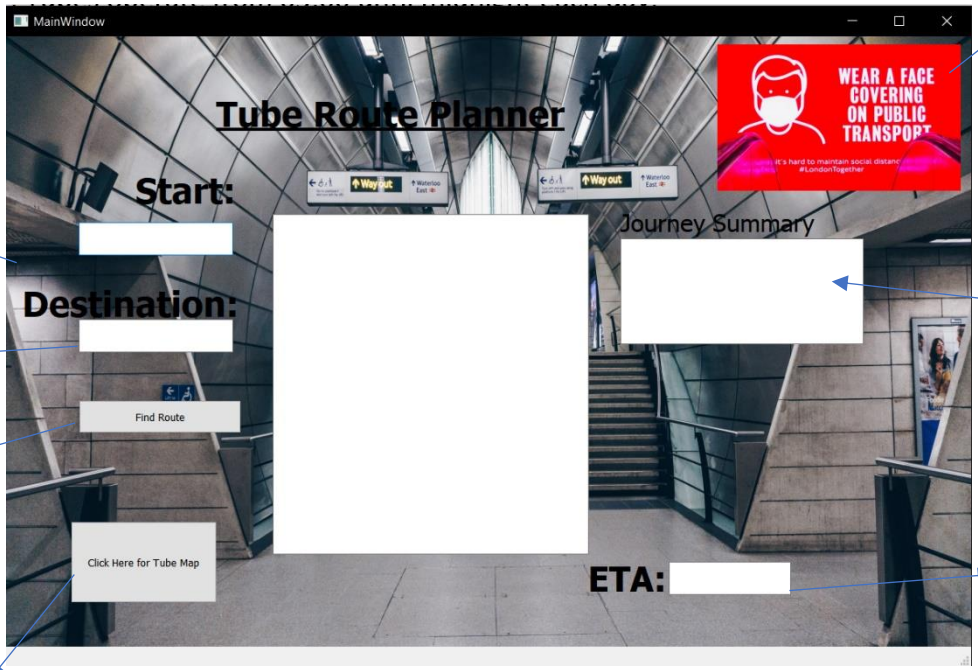
1. Tubes operate from 05:00 until midnight each day.
2. There is no difference between peak times and off-peak times and that the tubes run every 5 minutes with a 1-minute stop at each station to allow passengers to (dis)embark.

We have created this simple to use route planner for your ease, you simply type in your start and end destinations on the left and it will provide you the route in the central box.
With a Journey Summary to summarise when you should get off the tube and switch, will also include your total journey time. And for those people who love to know an estimate as to when they will arrive at the destination. We have implemented an ETA button that will calculate your arrival based on your systems time.

Furthermore we have currently replaced our means of advertisement with a reminder that you wear your face masks during public transport for the current pandemic’s safe.

For those who are interested in viewing the tube map, you may do so by pushing on the Tube Map button. This will pop up an image of the tube map for you to view.

The Main Window:



Advertisements and important notices will be displayed here

Enter your starting Station Here

Once Route is calculated, will provide a summary of the journey here

Enter your ending Station Here

Click here once you have made your choice

Your estimated arrival time will appear here

View the official Tube Map here

The GUI was built around being simple and the ease of information. Having all the information on one page.

The Tube Map Window:

The Official Tube Map

You can open the Tube map without it disrupting the Main Window.

2. Critical evaluation of the performance of the data structures and algorithms used (max 1 page) [15 marks]

We utilised Dijkstra's algorithm for the design and creation of London underground journey planner for the public use. Dijkstra's algorithm provided us the shortest route from node A to node B which in this case is from station A to Station B and it calculates the distance through every station and checks to see the shortest path to the destination the user has chosen. We thought about using the A* algorithm, however we did not need the extra capability of using a best-first algorithm. We created a double linked list data structure to connect all the nodes together. Therefore, we added extra variable to the Node class, such as "next_node" which holds just the name of the station, compared to "next_node_reference" which is essentially the main link between nodes.

The doubly linked list was challenging to build. Initially, our Doubly Linked list read the excel file as a whole. We figured this problem occurred because we used panda module. Therefore, used xlrd module instead to read the excel file and format the file in correspond to the structure of the double linked list. The second problem encountered was the linked list did not split the underground lines, resulting in a long connection. To overcome this, we modified a while loop to an indefinite iteration to split all the nodes into different double linked list depending on the line. Another difficulty faced was linking the list to the Dijkstra's algorithm and connecting the supplied excel data of the underground stations. To solve this, we had to change the format of our double linked list to be accepted by the Dijkstra's, so we created a nest dictionary to hold the start and endNodes for each node.

A better approach to linking with Dijkstra's would be with Fibonacci heap. This would allow priority queue to be implemented in Dijkstra's. In Fibonacci heap, each node has zero or more child nodes and each node has a sorting value known as key. Fibonacci heap is a list of self-contained sub-heaps, which we could use to represent each station like in the sub- graph. The station and the key would be stored in a node as collection of heap-ordered tree. Fibonacci makes runtime of Dijkstra's algorithm faster than with doubly linked list. It takes O (n) time to make a heap, O (1) time for key, and O (log n) time to de-queue, hence resulting in a fast runtime of O (m + n log n) for Dijkstra's algorithm.

Furthermore, we used a weighted graph where the weight on each edge represents the time it takes to travel between the stations at either ways. Each station has been represented in a sub-graph which is the nested dictionaries. We used nested dictionary and to create a graph. Dictionaries key forms the node for the graph where each key value corresponds to a list containing the nodes. Moreover, hast table was applied to organise data so that you can immediately look up values for a given unique key, thus in this case underground stations. The hast table is excellent for fast lookups as it has an average lookup of O(1) but the bigger the dataset, more time it will take as it must loop all the way through the entire London underground tube station dataset to find the correct key which can lower the lookup time to O(n) at worst case scenario.

We used a 2d array data structure as it allowed us to iterate through all the lists that we used. Arrays maintains several variable names utilizing a single name which is beneficial in this project has we have a large London underground tube map with roughly 300 stations and 11 tube lines covering 402km. In addition, we were able to use arrays to maintain each tube line with multiple stations under one single name but there were drawbacks to using arrays because insertion and deletion are

complicated to achieve in an array since the elements remain stored in a consecutive memory location.

Discussion for the choice of test data you provide and a table detailing the tests performed (max 2 pages) [10 marks]

Test Number	Test Input/procedure	Test output	Expected?	Reason for error (if exists)
1	Start: Kenton, Destination: Maida Vale	[‘Kenton’, ‘South Kenton', ' North Wembley', 'Wembley Central', 'Stonebridge Park', 'Harlesden', 'Willesden Junction', 'Kensal Green', "Queen's Park", 'Kilburn Park', 'Maida Vale'] 21 MINS	Yes	-
2	Start: Queen’s Park, Destination: Mile End	["Queen's Park", 'Kilburn Park', 'Maida Vale', 'Warwick Avenue', 'Paddington', 'Edgware Road', 'Marylebone', 'Baker Street', 'Bond Street', 'Green Park', 'Westminster', 'Waterloo', 'Southwark', 'London Bridge', 'Bermondsey', 'Canada Water', 'Canary Wharf', 'North Greenwich', 'Canning Town', 'West Ham', 'Stratford', 'Mile End'] 47 MINS	Yes	-
3	Start: Leyton, Destination: Bank	['Leyton', 'Stratford', 'Mile End', 'Bethnal Green', 'Liverpool Street', 'Bank'] 14 MINS	Yes	-
4	Start: Kenton, Destination: Brixton	Path-is-not.reachable	No, should be a transfer at Oxford Circus.	Error in Dijkstra’s reading backwards

5	Start: London Bridge, Destination: Clapham South	['London Bridge', 'Borough', 'Elephant & Castle', 'Kennington', 'Oval', 'Stockwell', 'Clapham North', 'Clapham Common', 'Clapham South'] 17 MINS	Yes	-
6	Start: Woodside Park, Destination: Waterloo	['Woodside Park', 'West Finchley', 'Finchley Central', 'East Finchley', 'Highgate', 'Archway', 'Tufnell Park', 'Kentish Town', 'Camden Town', 'Mornington Crescent', 'Euston', 'Warren Street', 'Oxford Circus', 'Green Park', 'Westminster', 'Waterloo'] 33 MINS	Yes	-
7	Start: LLLLLLLLL, Destination: Baker Street	Path-is-not.reachable	Yes and no	The path is not reachable as it does not exist, however we could have implemented some if statements to provide a different error message. E.g. “Station does not exist.”
8	Start: Kenton, Destination: RRRRR	Path-is-not.reachable	Yes and no	Same as above
9	Start: Kentor, Destination, Waterloo	Path-is-not.reachable	Expected but not desired	We could implement a function so that instead of reading it as “Kentor” it reads it as 5/6 of “Kenton” and accepts it as Kenton.
10/From here we tested the ETA function	Start: Kenton, Destination: Maida Vale Time: 13:37	13:58	Yes	-
11	Start: Woodside Park, Destination: Waterloo Time: 05:00	05:33	Yes	- (All values for time will be correct unless there was an error with the path finding)
12	Start: Woodside Park, Destination: Waterloo Time: 02:40	03:13	Expected but not desired	Tube Lines close at midnight. This should then print either “Tubes closed” or a route from 5 am.

We picked data for our test data by looking at the excel file and picking 2 random stations to compare. We thought this would provide a fair show of the capabilities of the route finder. We could have provided further tests, in regards to the GUI specifically but there was no time, or point.

3. Individual contribution by each team member and reflection (max 2 pages) [10 marks]

Mahdi Rahman

What I contributed to this project was to code the Dijkstra's algorithm and specifying it for the scenario we were given which was the train line of London. This part required me to adapt the Dijkstra code so that the distance was actually time between nodes and use the train line as a graph instead of your standardised [a, b, c ,d e..] graph. I also implemented Depth First Search algorithm to try find all the possible routes that can be taken to a specific destination. At the beginning what went well while working on this project was the actual coding of the Dijkstra as it worked very easily on its own as it had little errors such as maybe indentation errors. Furthermore what went well was the communication between group members as I did find some issues further deep in the code when I couldn't link the double linked list as a graph for the Dijkstra's, I was given suggestions by Lakshman for example to maybe try specifying nodes etc. what I could have done better is work better with time, although I wasn't behind on the work I was doing, the extra time I had could have used to further improve the program by maybe adding features such as estimated time arrival or try rework the algorithm to overcome situations like: if there was an obstacle or if a train lien was shut down. Also, I think I could have given more suggestions and ideas to my peers for their part of the project. what I have learned in is the different ways algorithms can be implemented into different situations and how to overcome coding issues where a certain data type is allowed. I have also learned the importance of teamwork and communication as that is what helped us create a fully functioning program in a more efficient manner.

Lakshman Thillainathan

I was lucky that I had studied Dijkstra's before and had focused my entire year 13 project on Dijkstra's. Learning about the double linked list was interesting, as I first thought that the double linked list would be unnecessary until I mapped the graph out on a piece of paper. Reading from the excel file and converting each row into an object was a difficult part. There were several errors in the line, also considering the repetition of certain firstNodes. It was difficult to split them up. Furthermore, if you read the whole excel file, it would link different lines to each other, unless you split for each line. I also took into consideration that maybe someone wants to add a new entry to the end of the file. It would correctly insert it into the specified double linked list depending on the line name. I feel as though we could have gotten more features implemented and had a fully working code with the GUI if my team had better communication and contribution. Overall, I enjoyed learning new ways of implementing Dijkstra's algorithm with a double linked list data structure, as it proved to be

a challenge. Looking back on the project, I think if we changed the format the Dijkstra's takes data from a nested dictionary to the actual double linked lists, we would have had less issues to deal with. During the creation of the GUI I had issues with the IDLE continuously restarting and crashing, and therefore did not manage to complete the GUI to my expectations. Therefore, provided the outputs in the idle as a backup.

Zarin Tasnim:

I worked on creating a Doubly Linked List. Every node in a double linked list has three parts meaning each data has a pointer to the next node and to the previous node. The data used represents each station in London underground. We can iterate the excel sheet in either direction by reading every rows and columns and then store them in individual node. As the code iterates through the excel sheet given, a new node is created for each station which allows us to go back and forth from every station. First, we import and create the node object. The class node is defined have two attributes- a value to hold the actual value of the node and a next, which holds the pointer to its next node. The _init_() function is called the constructor and it initialises the attribute of the created node. This function is automatically called when any object of class node is created. Whenever any node is created, it is independent until we add it to a list, that's why we are initialising next to null. The self-parameter in the function works as a pointer of the object. All the nodes in the linked list can be accessed if the head node is known. The insert() function is for adding new node at the end of the list; this can only be done if added to an empty node or if new node is added at the end of a non-empty list. The traverse() function access the next pointer of each node, starting from head node and continuously loops till a node pointes to null. Here, the traverse function iterates through nodes (stations) to another station until final station is reached. In the end we import the excel data sheet containing the data set for our London ground system.

Sreeraj Nair:

The contributions I made to this project was assisting in the report process and the implication for the GUI. Working in a group requires great communication which I believe we have achieved has we had regular updates, helped each other to learn from one another and complete the task at hand. With our project, we were able to complete almost everything except in the extra task where we had to show the list of train stations from start to finish. Although we did have some problems at such as implementing the doubly linked list as we have not used this function before, we were able to conquer the issue at the end through a great deal of research. Furthermore, the GUI I made at first was good but it wasn't appropriate for this journey planner project as we were required to show the interchanges between each station so therefore, it meant that we had to redesign the GUI again to meet the requirements. In addition, this project has taught me about Dijkstra algorithm, doubly linked list and how to utilize them which would not have been possible without my group. Lastly, I believe that I could have taken more part in the additional

feature implications, manage my time and involve myself in the discussions a bit more to be an efficient team player which I will work on for my future projects.

Name	**Allocation of marks agreed by team (0-100)**
Lakshman Thillainathan	
Mahdi Rahman	
Zarin Tasnim	
Sreeraj Nair	

REFERENCES:

[1] Kazimierski, W., et al. (2015). *Analysis of Graph Searching Algorithms for Route Planning in Inland Navigation*. TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation, 9(2), pp.281–286.

[2] Logan, B. (n.d.). *G52APT AI Programming Techniques Lecture 8: Depth first search in Prolog*. [online] Available at: http://www.cs.nott.ac.uk/~pszbsl/G52APT/slides/08-Depth-first-search.pdf [Accessed 19 Nov. 2020].

[3] Stout, W. Bryan (1998) *Smart Moves: Intelligent Path* [online] Available at: https://www.semanticscholar.org/paper/Smart-Moves%3A-Intelligent-Pathfinding-Stout/49bd015e47044e1cea3fc65ec02fac962303f575#citing-papers [Accessed 21 Nov. 2020].

[4] Mprostak (2020) Stack Overflow. (n.d.). *depth first search - Python get all paths from graph*. [online] Available at: https://stackoverflow.com/questions/62656477/python-get-all-paths-from-graph. [Accessed 20 Nov. 2020].

[5] Amitabha Dey (2019). Dijkstra's Algorithm in Python Explained. YouTube. Available at: https://www.youtube.com/watch?v=Ub4-nG09PFw [Accessed 20 Nov. 2020].