(Group 38)

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

Team members:

Name: shahzeb syed

Name: Mohammed Abdill

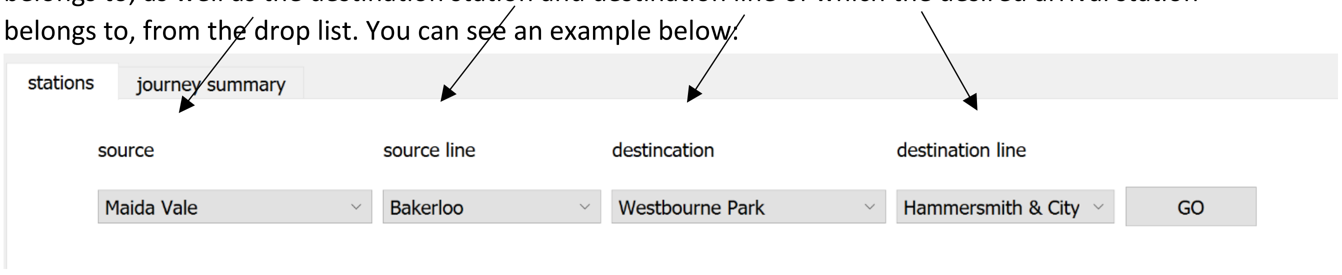
Name: Tejal Lama

Name: Siddharta Kanchala

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

To run the application, you should open the zip file, and run the Dikstra.py file, this will run the GUI of the application, and shortly after a desktop GUI will pop up on the device you are using.

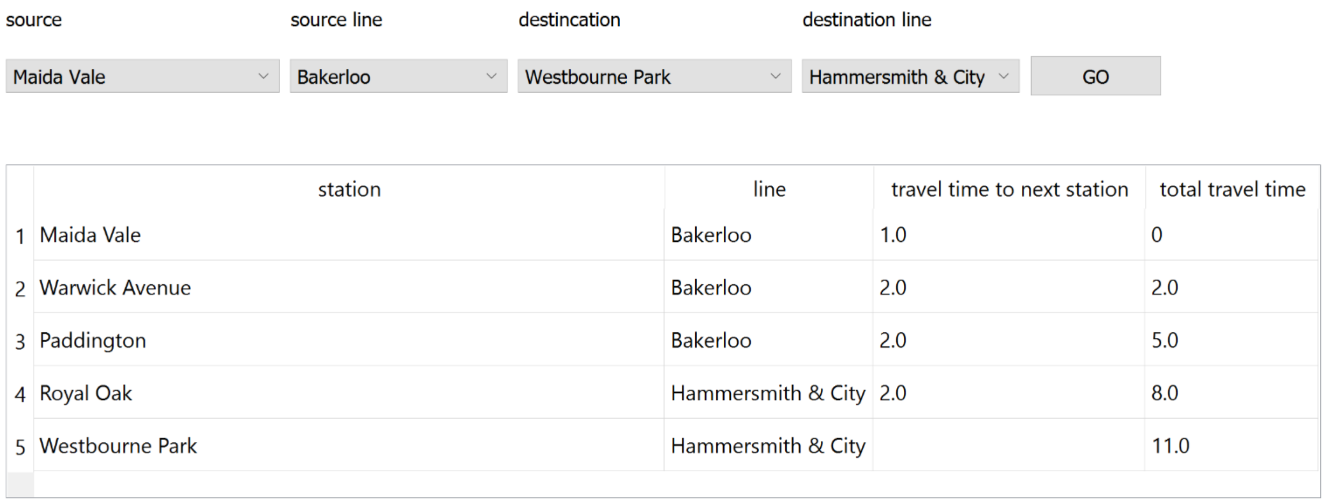
Once this is done, using the application itself is a fairly simple process with only a few steps required. The user is firstly required to input the source station alongside the source line of which the station belongs to, as well as the destination station and destination line of which the desired arrival station belongs to, from the drop list. You can see an example below:



Press the go button once all input fields are filled, a results table will then appear, showing the user the following:

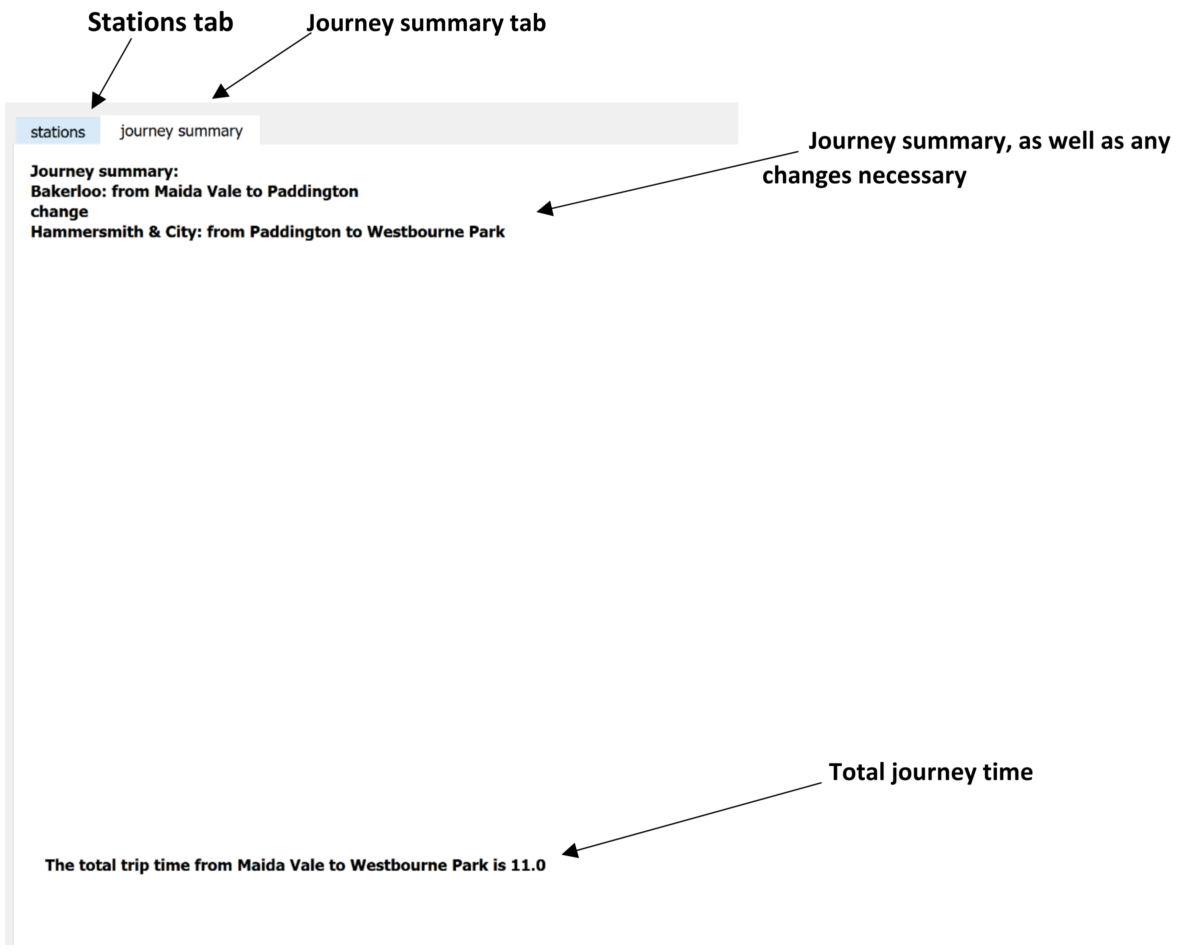
- The list of stations you will travel through during the journey, shown on the first column
- the line you will travel on for each station, shown on the 2nd column
- the travel time to the next station, shown on the third column
- Shown on column 4 is the accumulated travel time after each corresponding station (note: the time includes an additional minute for waiting time after each station).

The results table is shown below:



	station	line	travel time to next station	total travel time
1	Maida Vale	Bakerloo	1.0	0
2	Warwick Avenue	Bakerloo	2.0	2.0
3	Paddington	Bakerloo	2.0	5.0
4	Royal Oak	Hammersmith & City	2.0	8.0
5	Westbourne Park	Hammersmith & City		11.0

Once you have pressed the button and the results appear, you can also click on the summary tab on to the top left of the interface, which provides a summary that includes the total journey time, as well as any required line changes you will have to make during your journey, an example can be found below, to go back to the initial results table, simply go click on the stations tab:



Journey summary:
Bakerloo: from Maida Vale to Paddington
change
Hammersmith & City: from Paddington to Westbourne Park

The total trip time from Maida Vale to Westbourne Park is 11.0

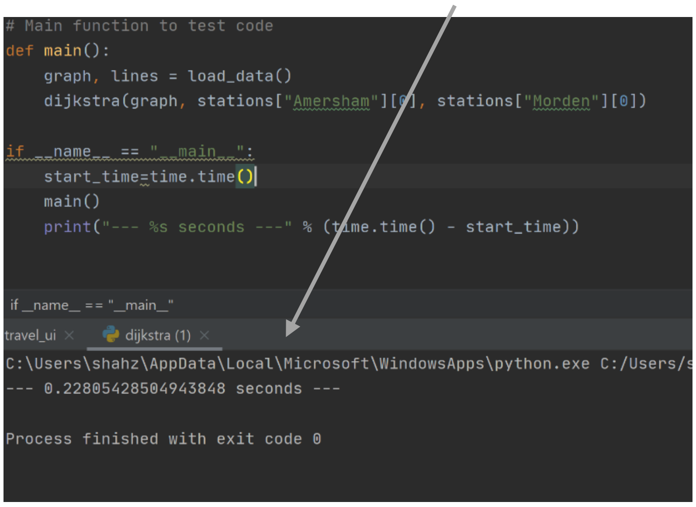
If you would like to query a different journey, simply just re-enter the required input fields and press the go button, the results will automatically be updated.

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

The required algorithm to be used for finding the shortest time between two given nodes was Dijkstra's algorithm, whereby each station and its connections was required to be stored in a doubly linked list.

The way the algorithm works, is the data is first extracted from the excel sheet, and then the connection is stored in a doubly linked list and represented in a graph. To find the shortest path between two given stations, the algorithm initialises the costs of each node in the graph to infinity, which marks each node as unvisited. Each node directly connected from the starting node (source station) are then visited, and assigned the given time taken to reach that node. The node with the shortest path is then assigned the current node, and the process is repeated until the destination node (destination station) is reached, and the shortest path is then calculated.

As you can see on the screenshots below to the right, the average running time of the algorithm is 0.203 seconds, which is very quick. This holds true regardless of the journey you query, weather it Is a small path, or a longer path you have to calculate, this is highlighted in the screenshot below to the left which shows the running time of the code when querying a long journey from Amersham to Morden, which outputs a running time of 0.22 seconds.

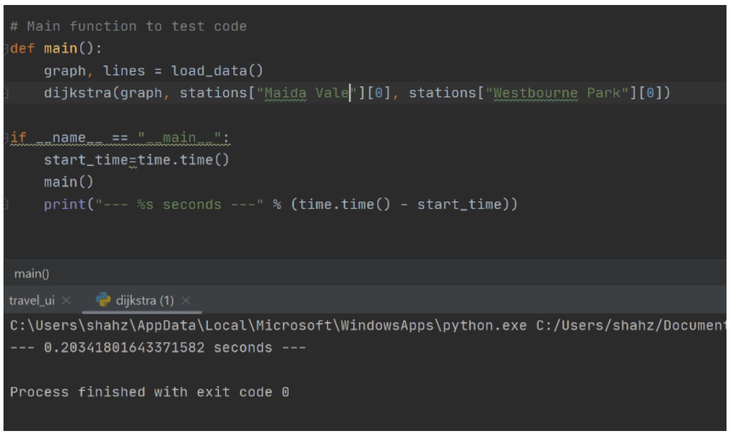


```
# Main function to test code
def main():
    graph, lines = load_data()
    dijkstra(graph, stations["Amersham"][0], stations["Morden"][0])

if __name__ == "__main__":
    start_time=time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))
```

```
C:\Users\shahz\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/s
--- 0.22805428504943848 seconds ---

Process finished with exit code 0
```



```
# Main function to test code
def main():
    graph, lines = load_data()
    dijkstra(graph, stations["Maida Vale"][0], stations["Westbourne Park"][0])

if __name__ == "__main__":
    start_time=time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))
```

```
C:\Users\shahz\AppData\Local\Microsoft\WindowsApps\python.exe C:/Users/shahz/Documen
--- 0.20341801643371582 seconds ---

Process finished with exit code 0
```

The time complexity of the Dijkstra is usually O(V log E), whereby V is the number of vertices and E is the number of edges. However, as we are not using a priority queue, the time complexity is O(V^2), meaning the performance is directly proportional to the square of the vertices inputted.

Implementing this was quite a challenge, because this was our first-time using Dijkstra's algorithm as well as a double linked list, however after researching and looking through the lecture codes, which were very helpful, we managed to present a working solution for the problem.

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

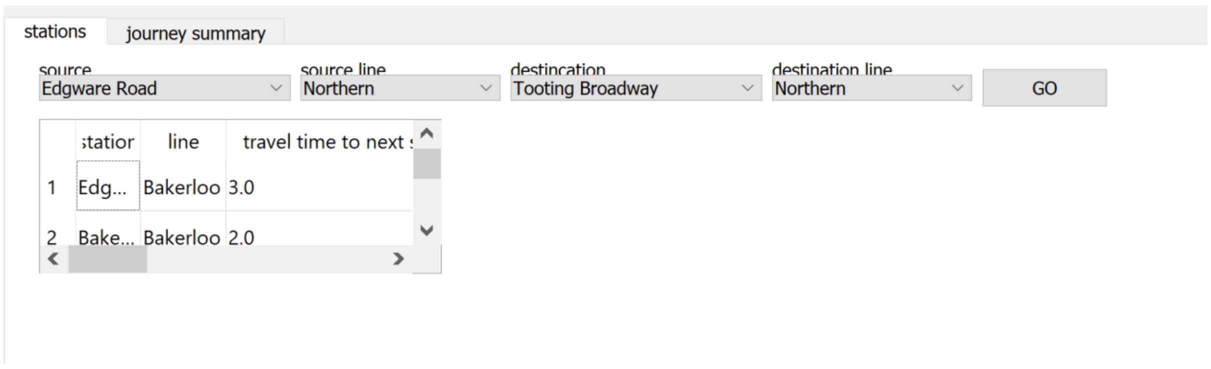
To test the application, we went through a variety of tests ranging from:

- Functional testing
- UI testing
- Performance testing

The functional testing was performed to ensure the full functionality of the software runs as required by the specification. To do this we came up with a range of test cases, ranging from small journeys to long journeys, as well as direct journeys and journeys which require changes. After each test case was performed, we double checked it with the actual TfL journey planner, found here; https://tfl.gov.uk/, and if the journey time was similar to those found on the website, we would pass the test as a success. Otherwise, we would refactor the code until the test case is run successfully. You can see a table of the tests we performed, as well as the result, to assess the functionality below:

Test case	**Result**	**Additional comments**
• Maida Vale to Westbourne park	• failed first attempt, but passed second	• on stations with multiple lines, we were having hard time with calculating which line the user will travel on, and therefore had to refactor the GUI so it asks the user to input the source and destination line
• Tooting Broadway to London Bridge	• Success	• This was performed to analyse the success of direct journeys which require no change.
• South Wimbledon to Amersham	• Success	• This was performed to analyse the success of long journeys which require multiple changes.

The second group of tests were performed to analyse the UI, to make sure the GUI performs as expected. There was some problem with this in the beginning as the, paddings and margins were off, resulting the text not displaying properly, and the results table not outputting all the results, However after some changes with the padding and margins, we fixed this problem an example of this can be seen below:



	statior	line	travel time to next s
1	Edg...	Bakerloo	3.0
2	Bake...	Bakerloo	2.0

The whole range of tests performed for UI can be found below:

Test case	**Result**	**additional comments**
Button interactivity	Passed	this test was performed to assess the interactivity of the application. Once the button was pressed, all results were updated in displayed correctly.
typography	failed at first, passed after refactoring	The text was not readable at first, so had to adjust the padding and margins until it was
Functional Validation: Maida vale to Westbourne park	Passed	This test was performed to ensure the outputted results display correctly as shown in the spec.
Adaptability	Passed	This test was performed to ensure the application works across a range of devices and screen sizes. The application was tested on both OS and windows, as well as on monitors with different dimension, all tests were passed

The final group of tests we performed were to assess the actual performance of the algorithm. This was done by using a variety of test cases and calculating the running time of each case. The results highlighted a clear and consistent running time, across a range of inputs. The results can be shown below:

Test case	running time
Maida vale to Westbourne park	0.203 seconds
Amersham to Morden	0.228 seconds
Wembley park to Whitechapel	0.194 seconds
West Ruislip to Epping	0.24 seconds

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

Name	Allocation of marks agreed by team (0-100)
shahzeb syed	100
Tejal lama	100
Mohammad abdill	100
Siddharta Kanchala	100

Shahzeb syed

although this was a team effort, Me and Abdill were responsible for implementing Dijkstra's algorithm and using the Doubly linked list to ensure that the connections are stored in the list, and the shortest path is found, While Tejal and Siddharta were responsible for the GUI. This was particularly hard, as usually Dijkstra does not use A Doubly linked list to store connections between nodes, so this required a lot of research and practice to get right. I did have a lot of difficulties in implementing this, and I believe in the future I could improve certain aspects of the algorithm.

For instance, the algorithm requires the user the input the departure line, as well as the arrival line for the journey, this is because some stations have more than one line, and I was not able to figure out how to fix this problem, because in reality it would not be practical to ask the user to input the line they will be travelling on, as the user may not be aware of which line they will be travelling on. To conclude, I believe this coursework was a very good learning experience, and has improved my communication skills, as well as my ability to code, and I look forward to improving this application in the future.

Tejal Lama

Siddhartha and I were responsible for creating the GUI and we were using the algorithm written by shahzeb and Abdill to find the correct output for the shortest path calculated and we also used the inputs provided by the user. We were responsible for coding the user input fields and loading the stations and lines onto the input drop downs, which were provided from the data sheet. We also had to make sure that the results outputted are cleared after every query.

Furthermore, I also conducted the tests on the GUI, which tested the interactivity, user input fields, and of course the results outputted. The GUI took quite some time to write, as I was not familiar with python GUI, so I had to get familiar with the pyqt5 library, and do extensive research on it, however we managed to come up with a working solution.

Mohammad abdill

My focus of this assignment was the implementation of the linked list. I had my group mates help me out whenever I was stuck in a section, but I had many challenges ahead of me that I had to figure out to implement the linked list correctly. As required, we used a linked list this data structure is also better than an array, as arrays are much slower in this case since an array would need to shift each node after deletion, wasting memory and time. I also had to consider the disadvantage of using linked lists which is that it requires more memory since they contain pointers that point to the next node. This problem isn't much of an issue in this case because the number of stations on the London underground is fixed. Secondly, a linked list requires each note to be visited beforehand, which increases the time a particular node is accessed compared to arrays. This however will not affect performance as diskstra's algorithm requires us to visit every node. Furthermore me and shahzeb also wrote Dijkstra's algorithm to calculate the shortest path between the nodes stored in the doubly linked list. These considerations made me understand the work I was doing and towards the end of the assignment I was very confident in my understanding of the work.

Siddharta Kanchala

As mentioned by tejal, me and him were responsible for the GUI portion of the application. We both wrote the code for loading the data onto the input fields, and implementing Dijkstra's algorithm on the given inputs to find the shortest path from the given nodes. We then conducted the tests on the GUI, to test the results outputs are accurate and precise, and that the dynamic aspects of the GUI, i.e the go button, works as expected and the data is loaded correctly.

References

Bogotobogo.com. 2020. *Python Tutorial: Dijkstra's Shortest Path Algorithm - 2020*. [online] Available at: <https://www.bogotobogo.com/python/python_Dijkstras_Shortest_Path_Algorithm.php> [Accessed 22 November 2020].

GeeksforGeeks. 2020. *Doubly Linked List | Set 1 (Introduction And Insertion) - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/doubly-linked-list/> [Accessed 22 November 2020].

Teachyourselfpython.com. 2020. *Howto: Searching Algorithms - Dijkstra's Algorithm*. [online] Available at: <https://www.teachyourselfpython.com/challenges.php?a=01_Solve_and_Learn&t=7-Sorting_Searching_Algorithms&s=02c_Dijkstras_Algorithm> [Accessed 22 November 2020].

Gist. 2020. *Dijkstra Algorithm Using Doubly Linked List On Clojure*. [online] Available at: <https://gist.github.com/dhk465/7b3529c8aac6378d932e520f553bb823> [Accessed 22 November 2020].

En.wikipedia.org. 2020. *Dijkstra's Algorithm*. [online] Available at: <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm> [Accessed 22 November 2020].

Lecture 2-Data structures I – Arrays, ADTs and Linked lists

Lecture 4- Searching – BST, AVL, Krushkal and Dijkstra algorithms