

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

### **Team members:**

Name: Han Gia Ha	ID: 001052068
Name: Barbara Slomba	ID: 001075526
Name: George Robert Robescu	ID: 001064882

## Table of Contents

<b>1. "HOWTO guide" for a customer and any assumptions made .....</b>	<b>3</b>
"HOWTO" guide.....	3
Assumptions .....	4
<b>2. Critical evaluation of the performance of the DS&amp;Algo used.....</b>	<b>5</b>
<b>3. Discussion for the choice of test data you provide and a table detailing the tests performed .....</b>	<b>6</b>
Discussion and table of test cases .....	6
Table of excel errors and modification.....	8
<b>4. Individual contribution by each team member and reflection .....</b>	<b>9</b>
Team members' reflection .....	9
Table of contribution .....	10
<b>5. References .....</b>	<b>10</b>
<b>6. Appendix .....</b>	<b>11</b>
<a href="#"><i>Code for TFL coursework (london_tube_final_v4.py).....</i></a>	<a href="#"><i>11</i></a>
Code files that are <u>not</u> used for our final coursework .....	15
Other solution for Dijkstra Algorithm implementation ( <i>test2.py</i> ) + explanation.....	15
Code for Doubly Linked List ( <i>doublelinklist.py</i> ) + explanation .....	18
Code for Doubly Linked List ( <i>linesdoubly.py</i> ) + explanation.....	20
Code for Doubly Linked List ( <i>linesdoubly_v2.py</i> ) + explanation .....	22

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

## 1.1. “HOWTO” guide

TFL Journey Planner app allows its users to find the fastest route between two tube stations in the London underground network. Using it is not complicated at all. Here is a simple guide:

1. This the graphic user interface to be seen when open the app:
2. Users can enter the name of departure and arrival stations where they wish their journey to start and end at.

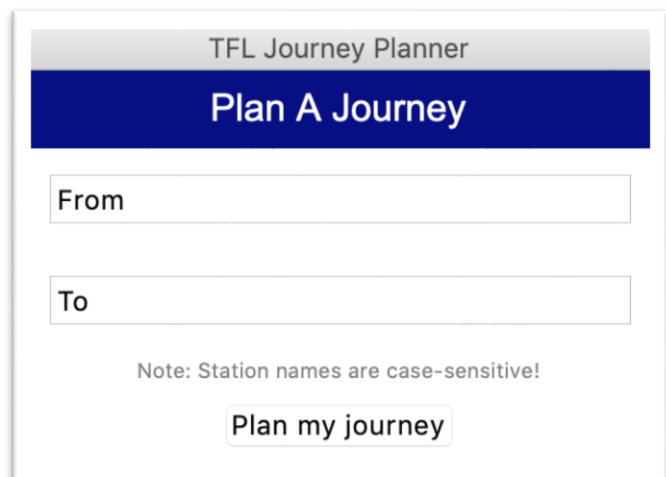
Tips: Users can either delete the words “from” and “to” or double-click to make those words disappear before enter station names.

Note: Stations name are case-sensitive and the app will only take station names with each word’s first letter being capitalized, which users need to be aware to get their desired results. For example:

Acceptable input: Maida Vale

Error input(s): MAIDA VALE, maida vale, MaiDa vAlE, etc.

3. The program will return a suggestive fastest route to users’ destination when they click “Plan my journey” button. Moreover, the button will be disabled after a click and return a message for users: “Your journey planned!”.



TFL Journey Planner

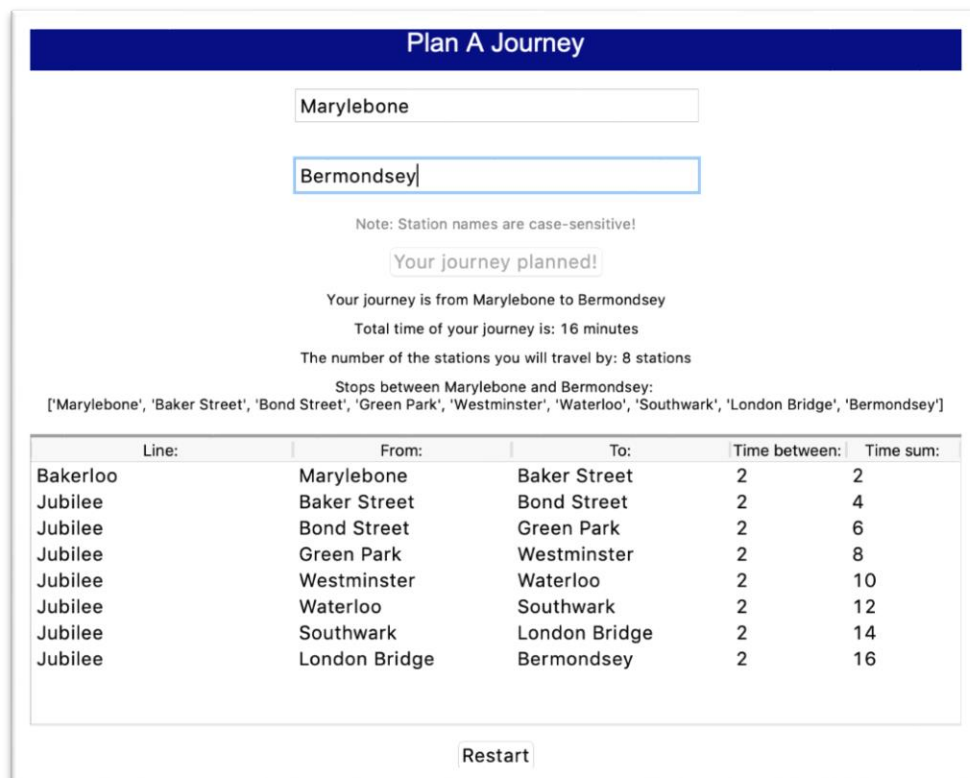
### Plan A Journey

From

To

Note: Station names are case-sensitive!

Plan my journey



### Plan A Journey

Marylebone

Bermondsey

Note: Station names are case-sensitive!

Your journey planned!

Your journey is from Marylebone to Bermondsey

Total time of your journey is: 16 minutes

The number of the stations you will travel by: 8 stations

Stops between Marylebone and Bermondsey:  
['Marylebone', 'Baker Street', 'Bond Street', 'Green Park', 'Westminster', 'Waterloo', 'Southwark', 'London Bridge', 'Bermondsey']

Line:	From:	To:	Time between:	Time sum:
Bakerloo	Marylebone	Baker Street	2	2
Jubilee	Baker Street	Bond Street	2	4
Jubilee	Bond Street	Green Park	2	6
Jubilee	Green Park	Westminster	2	8
Jubilee	Westminster	Waterloo	2	10
Jubilee	Waterloo	Southwark	2	12
Jubilee	Southwark	London Bridge	2	14
Jubilee	London Bridge	Bermondsey	2	16

Restart

In the above figure, if the program is successful, the journey summary will be displayed including:

1. Total time of the journey
2. Number of the stations that journey will take
3. The names of stations (stops) between journey's departure and arrival stations
4. A table<sup>1</sup> to show journey specific details:
  - 🚆 Name of line(s)
  - 🚆 From – to consecutive stations
  - 🚆 Time taken between two consecutive stations
  - 🚆 Total time after each portion of travel
5. Restart button to go back to the welcome interface for users to check another journey

## 1.2. Assumptions

Assumptions for our group's project:

- 🚆 No rush hours
- 🚆 No such problems like signal problems
- 🚆 The underground system operates 24 hours per day (As our program does not check for the time)
- 🚆 Our group neither assume a 1-minute stop at each station to allow passengers to (dis)embark.

---

<sup>1</sup> Please rather refer to the list of the stops for your journey than to the travel details table, because the table is in beta version and in some cases it doesn't show the right direction. Also it shows all the lines you can possibly take between your departure and arrival station, not only those that will make your journey the quickest according to the "Stops between" list.

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

We are using heap implementation of Dijkstra's algorithm.

Heap is a binary tree data structure where the value in a node is always smaller than both of its children. Heap is mainly used to implement priority queues (abstract data types). The heap implementation of the priority queue guarantees that both pushing (adding) and popping (removing) elements have logarithmic time complexity  $O(\log n)$ . Heap is a binary tree but heaps can be implemented as a **list**. This is what the Python **heapq** module does. **Heapq** module uses the convention of the smallest element having the highest priority.

Dijkstra's algorithm requires the input graph to have no negative-weight edges.

Our implementation of Dijkstra's algorithm:

Creates empty priority queue (pairs of weight and vertex)

```
1. g = defaultdict(list)
```

For loop: Add a vertex to the Queue (list)  $\rightarrow O(\log n)$

```
1. for l,r,c in stationlist: #l-from_station, c-cost/time, r-to_station
2.     g[l].append((c,r))
3. q, seen, mins = [(0,fromstation,())], set(),
```

While loop: Heappop, removes and returns elements of Q with the smallest weight, loop ends when empty  $\rightarrow O(\log n)$

```
1. while q:
2.     (cost,v1,path) = heappop(q)
3.     if v1 not in seen:
4.         seen.add(v1)
5.         path = (v1, path)
6.         if v1 == tostation: return (cost, path)
```

Nested for loop: heappush, adding elements, for each vertex updates the value in a priority queue to a lower value  $\rightarrow O(\log n)$

```
1. for c, v2 in g.get(v1, ()):
2.     if v2 in seen: continue
3.     prev = mins.get(v2, None)
4.     next = cost + c
5.     if prev is None or next < prev:
6.         mins[v2] = next
7.         heappush(q, (next, v2, path))
8.     stations_list = list(mins.items())
9. return float("inf"), stations_list[-1]
```

There are **m** edges to examine, and each one causes a heap operation of time  **$O(\log n)$**  so the running time is:

<b><math>O(m \log n)</math> or <math>O(E \log V)</math></b>
---

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

We have split the test data into three groups:

- ☞ Case 1 -7: Entering stations' names
- ☞ Case 8 – 15: Checking accuracy of the suggested journey
- ☞ Case: 16 & 17: User interface

All the tests were performed manually using the black box method.

When checking the accuracy of the journey times and shortest path we compared actual outputs to real world situations and as a reference we use the *Citymapper* app.

The table contains the description of each test case. After checking whether program functionality produces the expected outcome, the last column of the table provides problem identification, proposed solution and a comment if needed.

- ☞ Tests were performed for Dijkstra algorithm output: Front-end: "Stops between...", Back-end: "Stations between...", not for journey details table.
- ☞ Because the test cases table exceeds the 2-page limit of this chapter, it will be moved to the Appendix and only the first 8 test cases will be available on the next page (complete table with all the test cases in Appendix)

#### Summary of findings:

1. We had fixed some errors in the excel file before we started to work on the code but there were unnoticed errors which we only could detect when performing tests: Morden, Epping and Richmond stations were not recognized by the program due to space after their names in the excel file. That was fixed during testing. Also during testing it turned out that Metropolitan line needed fix: removing 2 rows that were making shortcut between Moor Park and Finchley Road and shortcut between Wembley Park and Harrow-On-The-Hill (all the other changes we did in excel file are enumerated in the Appendix);
2. Dijkstra's algorithm works well, it outputs the right number of the stops which are in the right sequence;
3. Time does not always reflect real travel times, because in our code we haven't included 1 extra minute after each station;
4. Serious limitation of the program is that it doesn't tell the user where he should change lines, the user has to figure it out himself;
5. A table with journey details does not meet expectations; in most cases stations shown in the table are in wrong order, there are also additional lines that shouldn't be there (all the other lines that also connect desired stations).

**Conclusion:** The program does not fully meet specifications.

Case#	Test case	Inputs	Expected output	Actual output	Result	Corrective action
1.	Checking if station exists	"Finsbury Victoria"	"This station does not exist"	Error, program terminates	FAIL	Add exception
2.	Checking if station exists	"Morden"	Journey summary	Error, program terminates	FAIL	Fix error in excel (space after name) → fixed & works

3.	Checking if station exists	"Epping"	Journey summary	Error, program terminates	FAIL	Fix error in excel (space after name) → fixed & works
4.	Typo in station's name	"Waterlo"	"This station does not exist"	Error, program terminates	FAIL	Add exception
5.	No space in station's name	"Highbury&Islington"	"This station does not exist"	Error, program terminates	FAIL	Add exception
6.	Copying and pasting station's name	"King's Cross St. Pancras"	Journey summary	Error, program terminates	FAIL	Must be entered manually because of apostrophe (') changing shape
7.	No capital letter in station's name	"holborn"	Journey summary	Error, program terminates	FAIL	We have tried <b>.title()</b> but letter after apostrophe (') was capitalized (eg. "Queen'S Park")
8.	Dijkstra algorithm check with stations from the same line	"Stratford" → "Greenford"	22 stops - 48 mins	22 stops - 49 mins	PASS	
9.	Dijkstra check: Same line	"Morden" → "Archway"	24 stops - 47 mins - 1 line	20 stops - 42 mins - 2 changes	PASS	Suggests to change at Stockwell to Victoria line and again at Euston to Northern line
10.	Dijkstra check: Different lines	"Holborn" → "North Greenwich"	8 stops - 25 mins	8 stops - 17 mins	PASS	
11.	Dijkstra check: Different lines	"Warwick Avenue" → "Aldgate"	12 stops - 26 mins	11 stops - 23 mins	PASS	Skipped "Marylebone" because changed at Paddington
12.	Checking direction: Route from station A to B and route from B to A	"Bayswater" → "Alperton" and back	x stops - x min	Same time and stops	PASS	
13.	A → B , B → A	"Eastcote → "High Barnet" and back	y stops - y min	Same time and stops	PASS	

14.	Checking interchange stations	"Baker Street" → "Watford"	12 stops - 40 min	5 stops - 43 mins	FAIL	Skips all stations between Finchley Road & Moor Park, except Harrow-on-the-Hill (Solution: delete excel row 14 (Moor Park & Harrow-on-the-hill), row 16 (Harrow-on-the-hill & Finchley Road ). After fixing, show 10 stops instead of 12 as expectation → Delete row 9 (Wembley Park & Harrow-on-the-hill) → solved
15.	Checking interchange station (District line)	"Victoria" → "Richmond"	13 stops - 30 min	Error, program terminates	FAIL	Remove space after Richmond in excel. After fixing, still miss 3 stations
16.	GUI check: stops between list	"Greenford" → "Epping"	all the stops between those two stations	first stops and last stops are not visible	FAIL	solution: maximize program window
17.	GUI check: table with more than 10 stops	"Greenford" → "Epping"	full table	10 rows of the table	PASS	click on the table and scroll down

**Table of excel errors and modification:**

Error was:	Changed to:	Error was:	Changed to:
Tottenham station (Central line)	Tottenham Court Road	row: Metropolitan - Moor Park - Harrow-on-the-hill - 14, giving unrealistic journey results (journey too quick)	deleted
Finsbury Park Victoria	Finsbury Park		
Holborn Central	Holborn	row: Metropolitan - Harrow-on-the-hill - Finchley Road - 16, giving false travel results (skipping stations)	
unnecessary space after Morden ("Morden ")	"Morden"		
unnecessary space after Epping ("Epping ")	"Epping"	row: Metropolitan - Wembley Park - Harrow-on-the-hill - 9, giving wrong results (same problem above)	
unnecessary space after Richmond ("Richmond ")	"Richmond"		



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

### Team members' reflection:

*Han: The end result is not just the completion of a coursework*

Regard skills and knowledge, not only earning my material selecting skills as we went through a few similar libraries and GUI toolkits before deciding on *openpyxl* and *tkinter*, but I also improved my source finding skills (mostly for Dijkstra implementation) and equipped myself with better programming skills (we experienced dictionary and list (DS), Dijkstra (Algo) and Tkinter (GUI)), which I find the most practical result from this module and can be useful for further studies and even future career. Although our team could not fulfil all requirements, I am satisfied with our work because for me, the improvement in coding ability and the flexible adaptability in the process of creating it is the most valuable outcome of this module.

Regarding experiences, I consider this coursework as an interesting chance to apply lessons into a real-world based project that I am familiar with and able to use real-world applications like Google Maps, Citymapper for output checking and comparison. We, therefore, can dig deeper into the testing phase and gain trouble shooting skills through trial and errors. In addition, this is the first remote project that all team members have experienced. Although not as expected in the work assignment and project management, especially the timing as I am the one to be half the hemisphere away from the rest of the team, we made it through is what deserves the most reflect.

*Barbara: I found the project very interesting and very challenging*

It was an opportunity for me to acquire new skills: I have learnt how to work with excel and how to use excel data in Python projects. I have also discovered that I very much enjoyed working with real world data and this is what made this project so special to me. It was more stimulating to try and design a program for London tube travels - something that I do every day and know very well - than working on some distant issue that I cannot relate to.

However, even though I find the coursework topic very pleasant and even though we had more than enough time to finalize the project, the implementation caused some difficulties for us. The problem wasn't trivial and we were not able to fully develop our solutions. We have created doubly linked list but weren't successful in applying it to Dijkstra's algorithm, this is why we chose heap implementation instead. We have attached the DLL code in the zip folder.

Finally, it was my first group project done fully remotely. That was a new experience for me and I have to admit that I enjoyed it and it went well, especially that I collaborated with a student living on another continent and there was a time difference.

*George: I feel that this project opened my eyes to real world problems*

For me this was the project with the most real-life applications that I have ever done. It made me aware of company requirements, and it stimulated me to try and surpass myself due to the fact that I was able to test the performance of our code against the likes of *Citymapper* and *Google Maps*.

However, even though the topic encouraged me to do better work than usual some of the requirements were a bit tough to implement and did not make too much sense as to why we should use them.

In the end, I feel like I have improved myself by completing this group project. It was a unique experience having to do everything remotely and to take into consideration time differences, connection problems and other technical difficulties that may emerge.

**Table of contribution:**

Name	Allocation of marks agreed by team (0-100)
Han Gia Ha	100%
Barbara Slomba	100%
George Robert Robescu	100%

## 5. References

Johansson, D., 2008. An Evaluation of Shortest Path Algorithms. *Final Thesis*, 18 December, 1(1), pp. 1-80.

Kachaiev, O., 2020. *Dijkstra shortest path algorithm based on python heapq heap implementation*. [Online] Available at: <https://gist.github.com/kachayev/5990802> [Accessed 22 November 2020].

Montanaro, A., 2013. *Priority queues and Dijkstra's algorithm*, Bristol : Department of Computer Science, University of Bristol.

Tutorialspoint, 2020. *Python - Advanced Linked list*. [Online] Available at: [https://www.tutorialspoint.com/python\\_data\\_structure/python\\_advanced\\_linked\\_list.htm](https://www.tutorialspoint.com/python_data_structure/python_advanced_linked_list.htm) [Accessed 22 November 2020].

Zadka, M., 2020. *The Python heapq Module: Using Heaps and Priority Queues*. [Online] Available at: <https://realpython.com/python-heapq-module/> [Accessed 20 November 2020].

## 6. Appendix

*London\_tube\_final\_v4.py: For this TFL coursework*

```
1. # Import libraries
2. from collections import defaultdict, namedtuple
3. from heapq import *
4. from openpyxl import Workbook
5. from openpyxl import load_workbook
6. import openpyxl
7. from astropy.table import Table # important to install astropy!!
8. from tkinter import *
9. from tkinter import ttk
10. import sys
11. import os
12.
13.
14.
15. # Read data from excel file
16. wb = Workbook()
17. wb = load_workbook("London Underground data modified4.xlsx")
18. ws = wb.active
19.
20. book = openpyxl.load_workbook('London Underground data modified4.xlsx')
21. sheet = book.active
22.
23.
24.
25. # BE inputs
26. # from_station = input("From: ").title()
27. # to_station = input("To: ").title()
28.
29.
30.
31. # Create nested list of all stations
32. stations = []
33.
34. for row in sheet.iter_rows(min_row=1, min_col=2, max_row=754, max_col=4): # 757 last one, 49 bakerloo
35.     stations_row = []
36.     for cell in row:
37.         stations_row.append(cell.value)
38.
39.     if stations_row[2] is not None:
40.         stations.append(stations_row)
41.
42.     reversed_stations_row = stations_row[:]
43.     element0 = reversed_stations_row[0]
44.     reversed_stations_row[0] = reversed_stations_row[1]
45.     reversed_stations_row[1] = element0
46.
47.     stations.append(reversed_stations_row)
48. #print("print stations", stations)
49.
50.
51.
52. # Dijkstra
53. def dijkstra(stationlist, fromstation, tostation):
54.     g = defaultdict(list)
55.     for l,r,c in stationlist: #l-from_station, c-cost/time, r-to_station
56.         g[l].append((c,r))
57.     q, seen, mins = [(0,fromstation,())], set(), {fromstation: 0}
58.     while q:
```

```

59.     (cost,v1,path) = heappop(q)
60.     if v1 not in seen:
61.         seen.add(v1)
62.         path = (v1, path)
63.         if v1 == tostation: return (cost, path)
64.
65.         for c, v2 in g.get(v1, ()):
66.             if v2 in seen: continue
67.             prev = mins.get(v2, None)
68.             next = cost + c
69.             if prev is None or next < prev:
70.                 mins[v2] = next
71.                 heappush(q, (next, v2, path))
72.         stations_list = list(mins.items())
73.         return float("inf"), stations_list[-1]
74.
75.
76.
77. # Flatten nested list of journey stations
78. def flatten(object):
79.     for item in object:
80.         if isinstance(item, (list, tuple, set)):
81.             yield from flatten(item)
82.         else:
83.             yield item
84.
85.
86.
87. # Create list of stations resulted from dijkstra
88. biglist = []
89. for element in stations:
90.     biglist.append(element)
91.
92.
93.
94. # Tkinter inputs
95. root = Tk()
96. root.title('TFL Journey Planner')
97. root.geometry('800x600')
98.
99. mainframe=Frame(root)
100. mainframe.pack(fill="both") #expand=True
101.
102. label=Label(mainframe,text="Plan A Journey",bg="navy",fg="white",padx=5,pady=5)
103. label.config(font=("Arial",18))
104. label.pack(fill="x")
105.
106. # Create entry
107. frStn_var = StringVar()
108. frStn_entry = Entry(mainframe, width=30, textvariable=frStn_var)
109. frStn_entry.pack(padx=10, pady=10)
110. frStn_var.set("From")
111.
112. toStn_var = StringVar()
113. toStn_entry = Entry(mainframe, width=30, textvariable=toStn_var)
114. toStn_entry.pack(padx=10, pady=10)
115. toStn_var.set("To")
116.
117. # Friendly disclaimer
118. reminder = Label(mainframe, width=30, text='Note: Station names are case-sensitive!', font='arial, 10', fg="gray")
119. reminder.pack()
120.
121.

```

```

122.
123. # Create restart button
124. def restart_program():
125.     python = sys.executable
126.     os.execl(python, python, * sys.argv)
127.
128.
129.
130. def findStn():
131.     # Get entry values
132.     from_station = frStn_var.get()    #.title()
133.     to_station = toStn_var.get()      #.title()
134.     stnLabel = Label(mainframe, text='Your journey is from ' + from_station + ' to ' + to_station, font='arial, 10')
135.     stnLabel.pack(padx=10)
136.
137.     # Disable button
138.     jrneBtn.config(text='Your journey planned!', state=DISABLED)
139.
140.     # Get journey result
141.     result = list(dijkstra(biglist, from_station, to_station))
142.     a = list(flatten(result))
143.
144.     # Journey summary
145.     # result_stations
146.     result_stations = []
147.     for row in sheet.iter_rows(min_row=1, min_col=1, max_row=754, max_col=4):
148.         result_stations_row = []
149.         for cell in row:
150.             result_stations_row.append(cell.value)
151.             if (result_stations_row[1] in a and result_stations_row[2] in a and result_stations_row[3] is not None):
152.                 result_stations.append(result_stations_row)
153.         print("result_stations:", result_stations)
154.
155.     # Total time
156.     time_in_total = a[0]
157.     print("\nTotal time of your journey is:", time_in_total, "minutes")
158.     timeLabel = Label(mainframe, text='Total time of your journey is: ' + str(time_in_total) + ' minutes', font='arial, 10')
159.     timeLabel.pack(padx=10)
160.
161.     # Number of stations
162.     stat_between = a[1:]
163.     print("\nThe number of the stations you will travel by:", len(stat_between)-1, "stations")
164.     numberStnLabel = Label(mainframe, text='The number of the stations you will travel by: ' + str(len(stat_between)-
165. 1) + ' stations', font='arial, 10')
166.     numberStnLabel.pack(padx=10)
167.
168.     # BE departure - arrival stations
169.     stat_between_order = reversed(stat_between)
170.     print("\nStations between", from_station, "and", to_station, ":")
171.     fromtoLabel = Label(mainframe, text='Stops between ' + str(from_station) + ' and ' + str(to_station) + ': ' + '\n' + str(stat_
172. between[::-1]), font='arial, 10')
173.     fromtoLabel.pack(padx=10)
174.
175.     # BE station list
176.     for item in stat_between_order:
177.         print(item)
178.
179.     # BE - create @time column for table
180.     c = result_stations
181.     minutes = []
182.     for i in c:
183.         minutes.append(i[3])

```

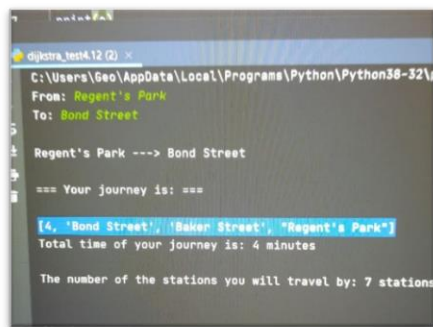
```

183.
184. # BE - create @totaltime column for table
185. def sum_minutes():
186.     total = 0
187.     sum_min = []
188.     for val in l:
189.         total = total + val
190.         # print(total)
191.         sum_min.append(total)
192.     return sum_min
193. list_min = sum_minutes(minutes)
194. for i in range(len(c)):
195.     c[i].append(list_min[i])
196.
197. # BE - @table
198. data_rows_new = c
199. tt = Table(rows=data_rows_new, names=('Line:', 'From:', 'To:', 'Time between:', 'Time sum:'))
200. print(tt)
201.
202. # FE - @table: Treeview
203. jrneTable = ttk.Treeview(mainframe)
204. jrneTable['column'] = ('Line', 'From', 'To', 'Time between', 'Time sum')
205.
206. jrneTable.column('#0', width=0, stretch=NO)
207. jrneTable.column('Line', anchor=W, width=180)
208. jrneTable.column('From', anchor=W, width=150)
209. jrneTable.column('To', anchor=W, width=150)
210. jrneTable.column('Time between', anchor=W, width=80)
211. jrneTable.column('Time sum', anchor=W, width=80)
212.
213. jrneTable.heading('#0', text="", anchor=W)
214. jrneTable.heading('Line', text='Line:', anchor=W)
215. jrneTable.heading('From', text='From:', anchor=W)
216. jrneTable.heading('To', text='To:', anchor=W)
217. jrneTable.heading('Time between', text='Time between:', anchor=W)
218. jrneTable.heading('Time sum', text='Time sum:', anchor=W)
219.
220. count = 0
221. for record in c:
222.     jrneTable.insert(parent="", index='end', iid=count, text="", values=(record[0], record[1], record[2], record[3], record[4]))
223.     count += 1
224.
225. jrneTable.pack(pady=10)
226.
227. restartBtn = Button(mainframe, text="Restart", command=restart_program)
228. restartBtn.pack()
229.
230.
231.
232. # Create journey button
233. jrneBtn = Button(mainframe, text='Plan my journey', command=findStn)
234. jrneBtn.pack(padx=10, pady=5)
235.
236.
237.
238. # Display window
239. root.mainloop()

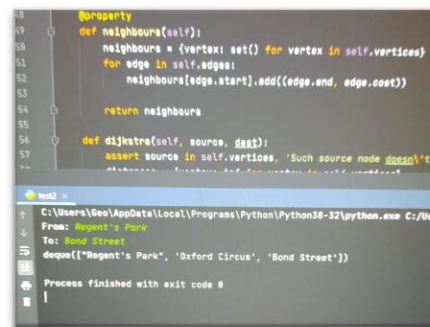
```

### Test2.py: Another solution for Dijkstra implementation that was not used for coursework

This approach was implemented in order to eliminate some of the errors that we first had by using a list.



London\_tube\_final\_v3.py



Test2.py

The outputs of stations have difference between *london\_tube\_final\_v3.py* and *test2.py* as can be seen from the pictures above:

<i>London_tube_final_v3.py</i> :	Regent's Park → Baker Street → Bond Street
<i>Test2.py</i> :	Regent's Park → Oxford Circus → Bond Street

In this case I opted to transform the list into a weighted graph to get rid of reversing a list or iterating it in reverse. I stored the elements in the graph as “neighbours”, each of them containing the information required in order for the algorithm to know where it can go next from a station (node) and the time it will take (cost).

We embed up not using this method as we could not get the travel times between each station and create a table with the results.

```
1. from collections import deque, namedtuple
2. from openpyxl import Workbook
3. from openpyxl import load_workbook
4. import openpyxl
5. from astropy.table import Table
6.
7. inf = float('inf')
8. Edge = namedtuple('Edge', 'start, end, cost')
9.
10. def make_edge(start, end, cost=1):
11.     return Edge(start, end, cost)
12.
13. class Graph:
14.     def __init__(self, edges):
15.         wrong_edges = [i for i in edges if len(i) not in [2, 3]]
16.         if wrong_edges:
17.             raise ValueError('Wrong edges data: {}'.format(wrong_edges))
18.
19.         self.edges = [make_edge(*edge) for edge in edges]
20.
21.     @property
22.     def vertices(self):
23.         return set(
24.             sum(
25.                 ([edge.start, edge.end] for edge in self.edges), []
26.             )
27.         )
28.
29.     def get_node_pairs(self, n1, n2, both_ends=True):
30.         if both_ends:
31.             node_pairs = [[n1, n2], [n2, n1]]
32.         else:
33.             node_pairs = [[n1, n2]]
34.         return node_pairs
```

```

35.
36.     def add_edge(self, n1, n2, cost=0, both_ends=True):
37.         node_pairs = self.get_node_pairs(n1, n2, both_ends)
38.         for edge in self.edges:
39.             if [edge.start, edge.end] in node_pairs:
40.                 return ValueError('Edge {} {} already exists'.format(n1, n2))
41.
42.         self.edges.append(Edge(start=n1, end=n2, cost=cost))
43.         if both_ends:
44.             self.edges.append(Edge(start=n2, end=n1, cost=cost))
45.
46.     @property
47.     def neighbours(self):
48.         neighbours = {vertex: set() for vertex in self.vertices}
49.         for edge in self.edges:
50.             neighbours[edge.start].add((edge.end, edge.cost))
51.
52.         return neighbours
53.
54.     def dijkstra(self, source, dest):
55.         distances = {vertex: inf for vertex in self.vertices}
56.         previous_vertices = {
57.             vertex: None for vertex in self.vertices
58.         }
59.         distances[source] = 0
60.         vertices = self.vertices.copy()
61.
62.         while vertices:
63.             current_vertex = min(vertices, key=lambda vertex: distances[vertex])
64.             vertices.remove(current_vertex)
65.             if distances[current_vertex] == inf:
66.                 break
67.             for neighbour, cost in self.neighbours[current_vertex]:
68.                 alternative_route = distances[current_vertex] + cost
69.                 if alternative_route < distances[neighbour]:
70.                     distances[neighbour] = alternative_route
71.                     previous_vertices[neighbour] = current_vertex
72.
73.             path, current_vertex = deque(), dest
74.             while previous_vertices[current_vertex] is not None:
75.                 path.appendleft(current_vertex)
76.                 current_vertex = previous_vertices[current_vertex]
77.             if path:
78.                 path.appendleft(current_vertex)
79.
80.         return distances[dest], path
81.
82. wb = Workbook()
83. wb = load_workbook("London Underground data.xlsx")
84. ws = wb.active
85.
86. book = openpyxl.load_workbook('London Underground data.xlsx')
87. sheet = book.active
88.
89. stations = []
90.
91. for row in sheet.iter_rows(min_row=1, min_col=2, max_row=754, max_col=4):
92.     stations_row = []
93.
94.     for cell in row:
95.         stations_row.append(cell.value)
96.
97.     if stations_row[2] is not None:
98.         stations.append(stations_row)
99.
100.         reversed_stations_row = stations_row[:]

```



```

101.         element0 = reversed_stations_row[0]
102.         reversed_stations_row[0] = reversed_stations_row[1]
103.         reversed_stations_row[1] = element0
104.
105.         stations.append(reversed_stations_row)
106.
107.     stations_final = Graph(stations)
108.
109.     from_station = input("From: ")
110.     to_station = input("To: ")
111.
112.     final = stations_final.dijkstra(from_station,to_station)
113.     print("Total time of your journey is:", final[0], "minutes\n")
114.     print("The number of the stations you will travel by is", len(final[1]),':')
115.     #print(final)
116.     final_list = list(final[1])
117.     for i in final_list:
118.         print(i)

```

## *doublelinklist.py*

Even though we couldn't implement DLL code in our final project, we have worked on it. We have started by creating a class DoublyLinkedList (code inspired from:

[https://www.tutorialspoint.com/python\\_data\\_structure/python\\_advanced\\_linked\\_list.htm](https://www.tutorialspoint.com/python_data_structure/python_advanced_linked_list.htm))

```
1. class DoublyLinkedList:
2.
3.     class Node:
4.         def __init__(self, datavalue):
5.             self.datavalue = datavalue # data element
6.             self.nextval = None
7.             self.prevval = None
8.
9.
10.    def __init__(self):
11.        self.headval = self.Node(None)
12.        self.tail = self.headval
13.        self.current = None
14.
15.    def __iter__(self):
16.        self.current = None
17.        return self
18.
19.    def __next__(self):
20.        if self.is_empty() or self.current == self.tail:
21.            raise StopIteration()
22.        elif self.current is None:
23.            self.current = self.headval
24.            self.current = self.current.get_next()
25.        return self.current
26.
27.    def get_next(self, node):
28.        if node == self.tail:
29.            raise Exception("Cannot get the element after the trailer of this list")
30.        else:
31.            return node.get_next()
32.
33.    def size(self):
34.        return self.size
35.
36.    def is_empty(self):
37.        return self.size == 0
38.
39.    # Adding data elements:
40.    def push(self, newVal):
41.        newNode = self.Node(newVal)
42.        newNode.nextval = self.headval
43.        if self.headval is not None:
44.            self.headval.prevval = newNode
45.        self.headval = newNode
46.
47.    def insert(self, prev_node, newVal):
48.        if prev_node is None:
49.            return
50.        newNode = self.Node(newVal)
51.        newNode.nextval = prev_node.nextval
52.        prev_node.nextval = newNode
53.        newNode.prevval = prev_node
54.        if newNode.nextval is not None:
55.            newNode.nextval.prevval = newNode
56.
57.    # Define the append method to add elements at the end:
58.    def append(self, newVal):
59.        newNode = self.Node(newVal)
```

```

60.         newNode.nextval = None
61.         if self.headval is None:
62.             newNode.prevval = None
63.             self.headval = newNode
64.             return
65.         last = self.headval
66.         while (last.nextval is not None):
67.             last = last.nextval
68.         last.nextval = newNode
69.         newNode.prevval = last
70.         return
71.
72.     def listprint(self, node):
73.         while (node is not None):
74.             print(node.datavalue)
75.             last = node
76.             node = node.nextval
77.
78. #dbllist = DoublyLinkList()
79. #dbllist.push(12)
80. #dbllist.push(8)
81. #dbllist.push(62)
82. #dbllist.insert(dbllist.headval.nextval, 13)
83. #dbllist.append(45)
84. #dbllist.listprint(dbllist.headval)

```

### *linesdoubly.py*

Here we imported code from doublelinklist.py file and we created Line class with nested \_line class. We believe it was the correct way of making doubly linked list using then openpyxl on it to extract stations' data. Unfortunately, when trying to apply it to different variations of Dijkstra's algorithm it was giving an error ("not iterable")

```
1. import doublelinklist
2. from openpyxl import Workbook
3. from openpyxl import load_workbook
4.
5. class Line:
6.
7.     class _line:
8.         def __init__(self, from_station, to_station, time):
9.             self.from_station = from_station
10.            self.to_station = to_station
11.            self.time = time
12.
13.        def __str__(self):
14.            return str(self.from_station) + " - " + str(self.to_station) + " - " + str(self.time) + " min"
15.
16.    def __init__(self):
17.        self.dblist = doublelinklist.DoublyLinkedList()
18.
19.    def add(self, from_station, to_station, time):
20.        if self.dblist.headval.datavalue == None:
21.            self.dblist.headval.datavalue = self._line(from_station, to_station, time)
22.        else:
23.            self.dblist.append(self._line(from_station, to_station, time))
24.
25.    def show_all(self):
26.        self.dblist.listprint(self.dblist.headval)
27.
28. wb = Workbook()
29. ws = wb.active
30. tube = load_workbook("London Underground data.xlsx")
31. sheet = tube.active
32.
33. print()
34. print("VICTORIA LINE:")
35. stations_v=[]
36. def vict_info():
37.     for i in sheet.iter_rows(min_row=740, max_row=754, min_col=2, max_col=4, values_only=True):
38.         stations_v.append(i)
39.     return
40.
41. vict_info()
42. # print(stations_v)
43. vict_full=Line()
44.
45. for el in stations_v:
46.     vict_full.add(el[0], el[1], el[2])
47.
48. vict_full.show_all()
49.
50. print()
51. print("JUBILEE LINE:")
52. stations_j=[]
53. def jub_info():
54.     for i in sheet.iter_rows(min_row=421, max_row=446, min_col=2, max_col=4, values_only=True):
55.         stations_j.append(i)
56.     return
```

```

55.         stations_j.append(i)
56.     return
57.
58. jub_info()
59. jub_full=Line()
60.
61. for el in stations_j:
62.     jub_full.add(el[0], el[1], el[2])
63.
64. jub_full.show_all()
65.
66. print()
67. print("HAMMERSMITH & CITY LINE:")
68. stations_h=[]
69. def hamm_info():
70.     for i in sheet.iter_rows(min_row=366, max_row=393, min_col=2, max_col=4, values_only=True):
71.         stations_h.append(i)
72.     return
73.
74. hamm_info()
75. hamm_full=Line()
76.
77. for el in stations_h:
78.     hamm_full.add(el[0], el[1], el[2])
79.
80. hamm_full.show_all()

```

## linesdoubly\_v2.py

This is an improved and developed version of previous code.

```
1. import doublelinklist
2. from tkinter import *
3. from openpyxl import Workbook
4. from openpyxl import load_workbook
5.
6. root = Tk()
7. root.title('Journey Planner')
8. root.geometry('500x800')
9.
10. wb = Workbook()
11. wb = load_workbook("London Underground data.xlsx")
12. ws = wb.active
13. #tube = load_workbook("London Underground data.xlsx")
14. #sheet = tube.active
15. column_a = ws['A']  #(Han) Get departure line_name
16. column_b = ws['B']  #(Han) Get arrival station
17. bakerloo = ws['A1']
18. central = ws['A50']
19. circle = ws['A148']
20. district = ws['A218']
21. hammersmith = ws['A337']
22. jubilee = ws['A394']
23. metropolitan = ws['A447']
24. northern = ws['A517']
25. piccadilly = ws['A618']
26. victoria = ws['A724']
27. waterloo = ws['A755']
28.
29. class Line:
30.
31.     class _line:
32.         def __init__(self, from_station, to_station, time):
33.             self.from_station = from_station
34.             self.to_station = to_station
35.             self.time = time
36.
37.         def __str__(self):
38.             return str(self.from_station) + " - " + str(self.to_station)+" - " + str(self.t
ime) + " min"
39.
40.     def __init__(self):
41.         self.dblist = doublelinklist.DoublyLinkedList()
42.
43.     def add(self, from_station, to_station, time):
44.         if self.dblist.headval.datavalue == None:
45.             self.dblist.headval.datavalue = self._line(from_station, to_station, time)
46.         else:
47.             self.dblist.append(self._line(from_station, to_station, time))
48.
49.     def show_all(self):
50.         self.dblist.listprint(self.dblist.headval)
51.
52. def find_station(): #(Han) should be 'get_departure' function
53.     line_list = ''
54.     station = []
55.     station_list = Line()
56.     for cell in column_a:  #(Han) 'cell' currently is A1 - Barkerloo, I'll try drop-
down menu later so users can choose another line
57.         #print(cell.value)
58.         if cell.value == bakerloo.value:
59.             line_list = f'{line_list + str(cell.value)}'
```

```

60.         label_a.config(text=line_list)
61.         print('\nThis is', line_list, 'line')
62.         for i in ws.iter_rows(min_row=26, max_row=49, min_col=2, max_col=4, values_only
= True):
63.             station.append(i)
64.             if cell.value == central.value:
65.                 line_list = f'{line_list + str(cell.value)}'
66.                 print('\nThis is', line_list, 'line')
67.                 for i in ws.iter_rows(min_row=99, max_row=147, min_col=2, max_col=4, values_onl
y=True):
68.                     station.append(i)
69.             else:
70.                 print('Please check your line\' name again!')
71.                 for el in station:
72.                     station_list.add(el[0], el[1], el[2])
73.                 station_list.show_all()
74.                 break
75.
76. '''vict_info()          #Hey Barbara, I'm so sorry for cut your Hammersmith and Victoria
77. # print(stations_v)    #Your idea was great and helpful for us
78. vict_full=Line()      #But I think instead of using 10x2 functions for 10 lines for both
    'from' and 'to'
79.                        #We can use only get_departure and get_arrival functions
80. for el in stations_v:  #I'll text to the group, just in case, also leave comments here! lo
    ve chu <3
81.     vict_full.add(el[0], el[1], el[2])
82.
83. vict_full.show_all()'''
84.
85. def get_departure():
86.     pass
87.
88. ba = Button(root, text="From", command = find_station) #(Han) Show time to travel between
    each 2 stops when you click the button
89. ba.pack(pady=20)
90. label_a = Label(root, text="")
91. label_a.pack(pady=20)
92.
93. def get_arrival():
94.     line_list = ''
95.     for cell in column_b:
96.         #line_list = list(dict.fromkeys(line_list)) #Remove duplicates
97.         line_list = f'{line_list + str(cell.value)}\n'
98.         label_b.config(text=line_list)
99. ba = Button(root, text="To", command = get_arrival)
100.     ba.pack(pady=20)
101.     label_b = Label(root, text="")
102.     label_b.pack(pady=20)
103.
104.     find_station()
105.
106.     root.mainloop()

```