COMP 1828 - Designing, developing and testing solutions for the London Underground system

Team members:

- Tristan Read (ID: 001151378)

Justification of the choice of the data structures and algorithms

(max 1 page) [10 marks]

For my construction of a graph, I have decided to use an OOP based approach, this is for a few reasons.
A graph in my program is a collection of nodes and edges, each as their own separate object. My nodes contain an adjacency dictionary who's keys are the neighboring node IDs and values are another dictionary of edge IDs and edges. The adjacency dict is multi-dimensional because I needed to have support for multigraphs, where one node can have many edges to the same neighbor.
The graph object contains a dictionary of nodes and another dictionary of edges. The nodes dictionary is used to store the nodes in the graph, and the edges dictionary is used to store the edges in the graph with their source and destination nodes.

Critical evaluation of the performance of the data structures and algorithms used

(max 2 pages) [20 marks]

The reason behind me using dictionaries for everything is that while they may take a bit longer to index and use a bit more memory is, overall they are faster and more reliable because if I were to add a new node, I would not have to resize the entire array. The story is similar for removing an item, but you also don't have to update all of your array indexers, by giving each element their own ID, you can always index that same elements info no matter what modifications are made to the dictionary. This is as opposed to if you had an array where you would also have to keep track of all of the elements if you needed to index them again.
My choice of algorithm for the primary task was to use Dijkstra's algorithm. The reason I picked this algorithm is because it was the one that I understood the best by myself (without looking at external code resources). While this is a "greedy" algorithm, meaning it

will not follow the shortest path for every search it makes, it is still fast and reliable enough for this application.
Before an attempt is made to search the graph for a path between the specified nodes, a depth first search is run to ensure that the graph is connected because if it is not, then the search would have a chance of failing, and in the case of the algorithm I used in task 3, for Bellman Ford's dynamic programming algorithm, it would fail when checking for negative cycles.
For task 3, I decided implement the dynamic programming variant of the Bellman Ford algorithm, this algorithm is not greedy and follows the shortest path every iteration. In my implementation of Dijkstra's algorithm, I had made an optimization where I would return the result once the end node was boxed. I was unable to implement this into the Bellman Ford algorithm because it would end up throwing off the negative cycle detection.

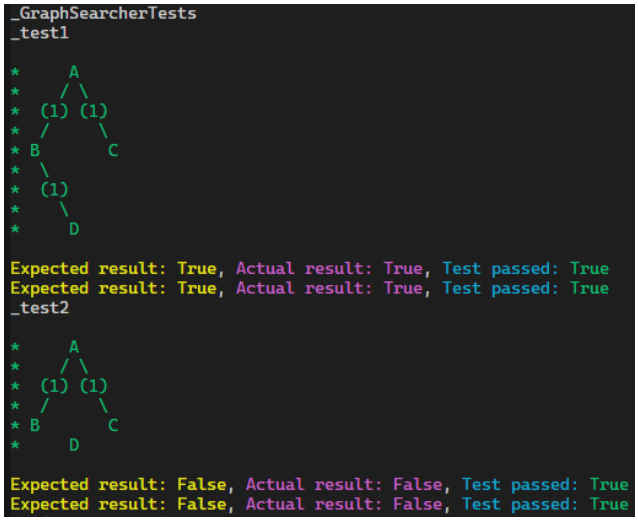
Discussion for the choice of test data you provide and a table detailing the tests performed

(max 2 pages) [10 marks]

`Summary` Discuss my JSON data structure and why I choose it.
Before writing this Python program and after I had settled on a data structure for the internal program, I decided to write a csx (C#) script that would convert the provided csv tubemap data file to a JSON file. The json structure that I had created is similar to what is used in the internal program however it utilizes more lists than dictionaries, as if I were to make a 1:1 copy of the internal program's data structure, it would result in an infinite recursion loop because of how I store values by reference on the nodes.
For example in the internal program I store the node ID and node reference in the dictionary, but the resulting json file would only store the node ID, which when loaded back into the program would be used to build a new node object to be added to a graph.
For my test data I created some simple graphs that I could figure the result of in my head with ease as well as one example from the lecture slides, I then compared this against the programs result using the various algorithms I had implemented.

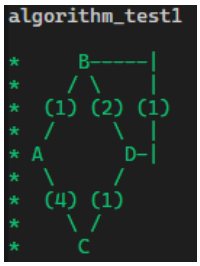
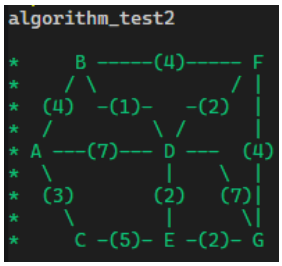
Graph searcher tests:

In this test I check to see if a graph is connected or not.



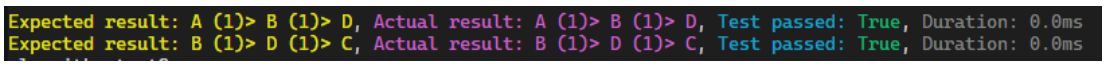
Pathfinder algorithm tests:

In this test I check to see if the pathfinder algorithms are working correctly against two known graphs.
Below are the two graphs used for testing:

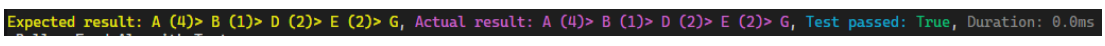
Graph 1	Graph 2
	

Dijkstra's algorithm:

Graph 1

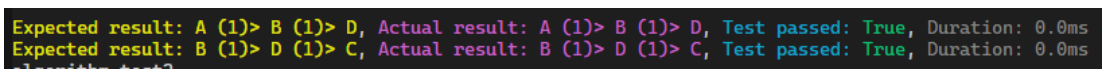


Graph 2



Bellman Ford algorithm:

Graph 1



Graph 2

```
Expected result: A (4)> B (1)> D (2)> E (2)> G, Actual result: A (4)> B (1)> D (2)> E (2)> G, Test passed: True, Duration: 1.1234283447265625ms
```

Screen-captured demonstration of your working source code

(max 3 pages) [20 marks]

Task 1a:

Task 1a asked us to calculate and display the shortest path between two nodes.

```
> start paddington
The start station is now 'Paddington'.
> end woodford
The end station is now 'Woodford'.
> go
The route from 'Paddington' to 'Woodford' has a duration of 39 minutes.
Start at 'Paddington'.
Ride 1 stop to 'Edgware Road' via the 'Bakerloo' line.
Ride 4 stops to 'King's Cross St. Pancras' via the 'Circle' line.
Ride 1 stop to 'Farringdon' via the 'Metropolitan' line.
Ride 3 stops to 'Liverpool Street' via the 'Circle' line.
Ride 8 stops to 'Woodford' via the 'Central' line, where you will arrive at your destination.
```

Task 1b:

Task 1b asked us to show a histogram of the time between each station in the path.

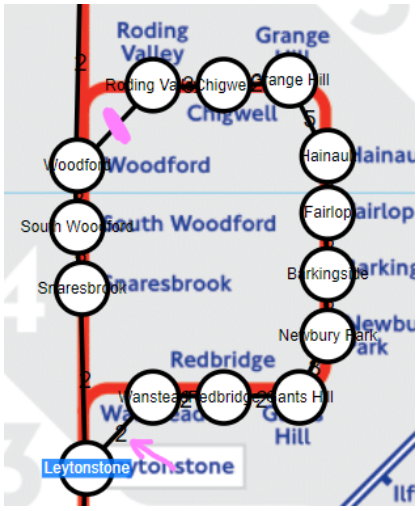
```
Ride 8 stops to 'Woodford' via the 'Central' line, where you will arrive at your destination.
Histogram of times between each previous station:
       Station        |       Time between previous station (minutes)
                      | 0           1           2           3           4
          Paddington  |
        Edgware Road  | ==========
        Baker Street  | ========================================
Great Portland St...  | ==========
       Euston Square  | ==========
  King's Cross St...  | ==========
         Farringdon   | ==========
           Barbican   | =========
           Moorgate   | ==========
    Liverpool Street  | ==========
      Bethnal Green   | ========================================
           Mile End   | ==========
          Stratford   | ==================================================
             Leyton   | ========================================
         Leytonstone  | ==========
         Snaresbrook  | ==========
      South Woodford  | ==========
            Woodford  | ========================================
>
```

Task 2a:

Task 2a tasked us with being able to close a line, and denying closure if it is not feasible.
Here I am closing the line between `Woodford` and `Roding Valley` via the `Central` line.
This closure is accepted because it does not disconnect the graph.
However when I try to close the line between `Leystone` and `Wanstead` via the `Central` line, the closure is denied because it would disconnect the graph because all stations between `Roding Valley` and `Wanstead` would be unreachable.

```
> line info Woodford "Roding Valley"
Histogram of times between Woodford and Roding Valley:
  Line  |                    Time (minutes)
        | 0              1              2              3              4
Central | ==========================================================
> line close Woodford "Roding Valley" central
The Line between 'Woodford' and 'Roding Valley' via 'Central' is now closed.
> line info Leytonstone Wanstead
Histogram of times between Leytonstone and Wanstead:
  Line  |                      Time (minutes)
        | 0                              1                              2
Central | ==============================================================
> line cclose Leytonstone Wanstead central
Invalid syntax.
> line close Leytonstone Wanstead central
The Line between 'Leytonstone' and 'Wanstead' via 'Central' cannot be closed as it would cause one of the stations to be unreachable.
>
```

Below is a visual representation of what I was explaining above.



Task 2b:

Task 2b asked us to show the impact of a closure one or multiple lines would have on the quickest journey between two stations.

```
> line close Woodford "Roding Valley" central
The Line between 'Woodford' and 'Roding Valley' via 'Central' is now closed.
> start woodford
The start station is now 'Woodford'.
> end "Roding Valley"
The end station is now 'Roding Valley'.
> go
The route from 'Woodford' to 'Roding Valley' has a duration of 32 minutes.
Due to some line closures, the journey will take 28 minutes longer than the most optimal route (4 minutes).
Start at 'Woodford'.
```

Task 3:

Task 3 asked us to either come up with another algorithm to solve the pathfinding problem or create a visual representation of the graph. I had decided to do both, however the visual representation was not written in Python.

Part 1, implementing a different algorithm:

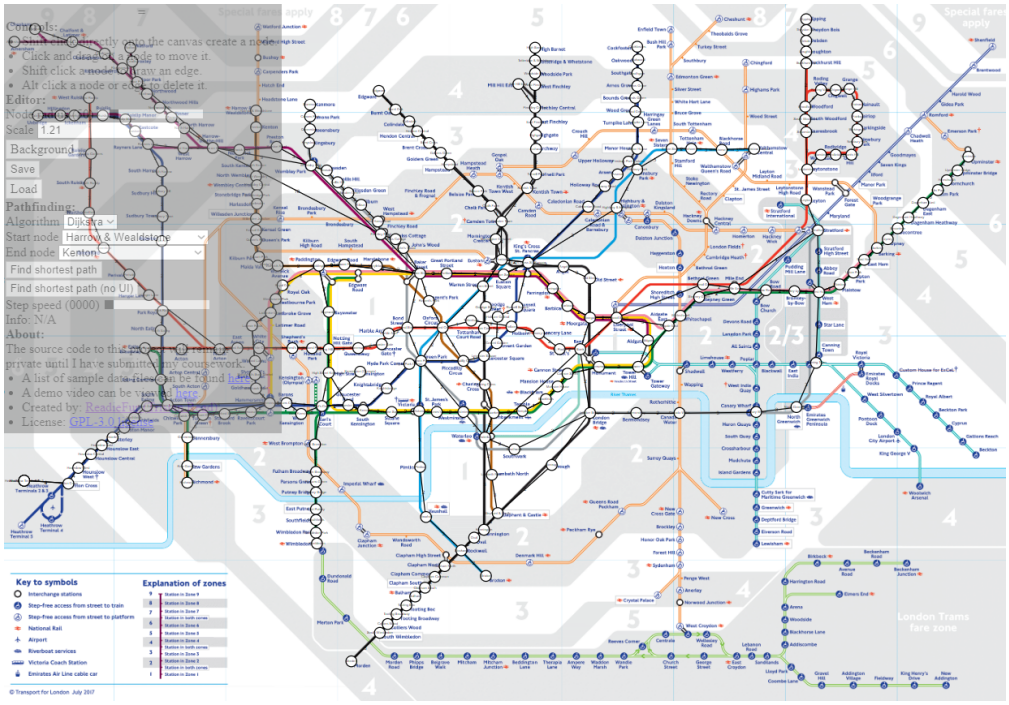
```
> start paddington
The start station is now 'Paddington'.
> end woodford
The end station is now 'Woodford'.
> algorithm list
Available algorithms:
- Dijkstra
- Bellman Ford DP
```

```
> algorithm dijkstra
The algorithm is now 'Dijkstra'.
> go debug
Calculation took 87.66ms, using Dijkstra's algorithm.
The route from 'Paddington' to 'Woodford' has a duration of 39 minutes.
Start at 'Paddington'.
```

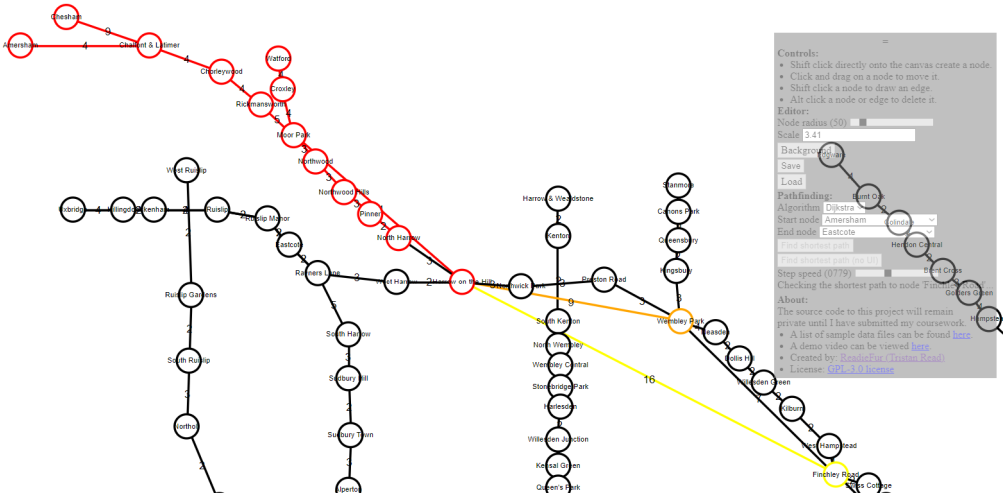
```
> algorithm "bellman ford dp"
The algorithm is now 'Bellman Ford DP'.
> go debug
Calculation took 751.62ms, using Bellman Ford DP's algorithm.
The route from 'Paddington' to 'Woodford' has a duration of 39 minutes.
Start at 'Paddington'.
```

Part 2, visual representation of the graph:

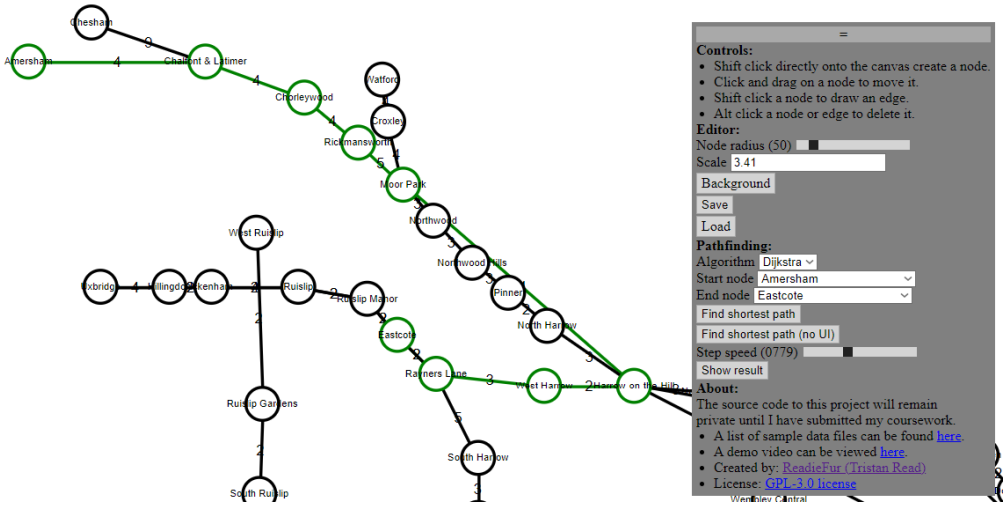
My web-app is what I had used to tidy up the data after my OCR pass in a different C# program. This web-app is able to add, remove and edit nodes and edges.



This image shows the visualized process of the pathfinding algorithm.



This image shows the final result from the pathfinding algorithm.



Outcomes of Task 1, 2 and 3

(max 2 pages) [20 marks]

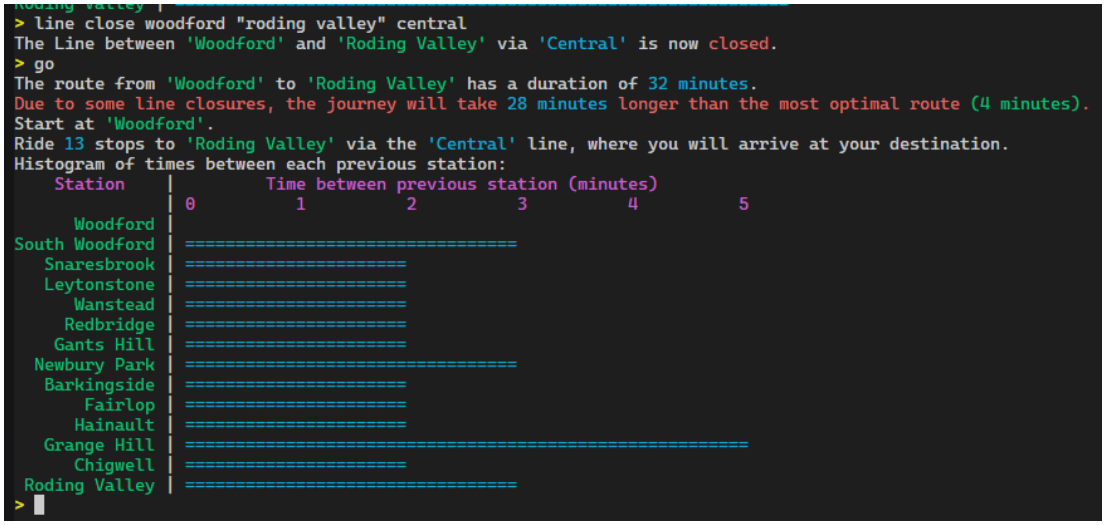
Comparison of the histogram results between task 1b and 2b.

Task 1b

```
The route from 'Woodford' to 'Roding Valley' has a duration of 4 minutes.
Start at 'Woodford'.
Ride 1 stop to 'Roding Valley' via the 'Central' line, where you will arrive at your destination.
Histogram of times between each previous station:
   Station     |           Time between previous station (minutes)
               | 0             1             2             3             4
     Woodford |
Roding Valley | ============================================================
```

Task 2b

Task 2b



Conclusion and critical discussion on the limitation of the work done

(max 1 page) [20 marks]

During my creation of this program, I had initially created the program in TypeScript building the web-app seen in task 3 part 2. I had written things this way because I am more familiar with type strict languages and I find them much easier to debug. By building the web-app I also had the benefits of visualizing the data with more ease. I had converted the csv file using a C# script and Google's Tesseract OCR library to line them up with the London underground map where I then used my web-app to tidy up the data and export it for use in this program.
In terms of the algorithms used in this project, I had read that there are much more efficient algorithms for solving the shortest path problem, however I had decided to use the algorithms that were taught in the lectures as they gave me the strongest lead on how to solve the problem.

Weekly log of progress, individual contribution toward the final outcome by each team member

(max 2 pages) [10 marks]

Week	Progress
23rd October	Started experimenting with data structures.

Week	Progress
30th October	Started building Dijkstra's algorithm.
6th November	Completed building TS core web-app (source, private as of submission, demo).
13th November	Converted TS web-app core to a Python CLI program and added task 2 specific properties.
20th November	**1.** Completed task 1B to display a histogram timeline. **2.** Completed task 2A to allow line closures. **3.** Added Bellman Ford's dynamic programming algorithm.

Name	Allocation of marks agreed by team (0 - 100%)
Tristan Read	100%