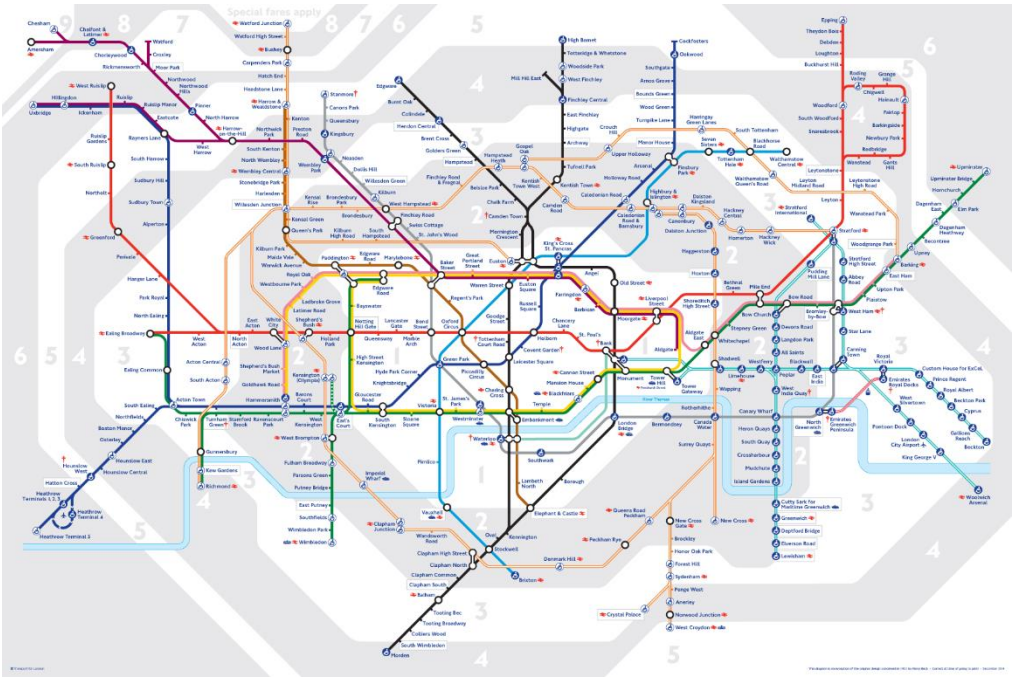
Data Structures and Algorithms 2

CA Exercise 2 – London Underground Route Finder

"Create a route finder for the London underground."

The objective of this team CA exercise is to create a JavaFX application that travellers can use to search for and retrieve routes between stations on the London underground given a starting station and a destination station. A sample map of the London underground:



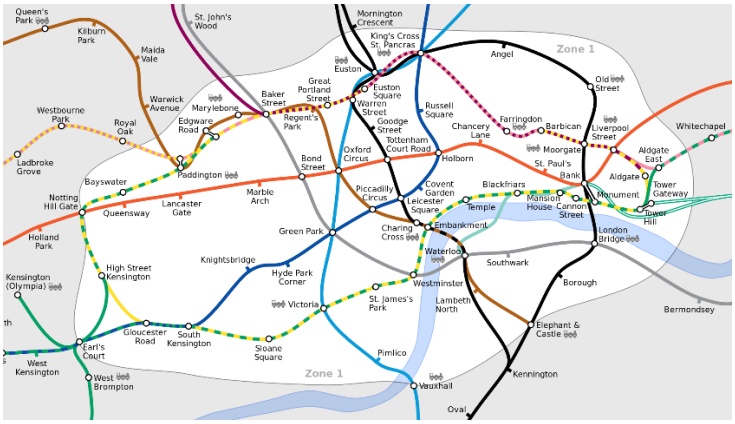
The application should create/generate the following on request:

- Multiple route permutations between starting and destination stations.
    - This functionality should use depth-first search (DFS).
- Route with the fewest stations/stops between the starting and destination stations.
    - This functionality should use breadth-first search (BFS).
- Shortest route (in terms of distance travelled) between the starting and destination stations.
    - This functionality should use Dijkstra's algorithm.
- Shortest route (in terms of distance travelled) between the given starting and destination stations, but allowing a user-specified cost penalty to be applied for every time a traveller has to change lines/trains at a station.
    - Using Dijkstra's algorithm again, but with a variable cost penalty incorporated for every line/train change.

The application should also allow travellers to specify stations to avoid on the route, and to specify stations that must be visited on the route to the final destination station. It should also show a geographical map representation of routes that are entirely within Zone 1.

Implementation Notes

- This is a team CA exercise. Teams should consist of 2 students only. Students may be permitted to work by themselves in very limited circumstances (this requires the prior approval of your lab lecturer).
- This CA exercise is worth 35% of your overall module mark. You must submit it on Moodle and demonstrate it (as a team) to your assigned lab lecturer for it to be assessed and included in your overall marks. The demo/interview is mandatory!
- The key point of this CA exercise is in the use of graphs (i.e. nodes/vertices connected by links/edges), along with some algorithms to search/traverse graphs.
    - Think of stations as nodes/vertices and connecting lines as links/edges.
    - You should develop your own custom graph-based data structure in the first instance as the main data structure for representing stations and their interconnecting lines. However, beyond the required custom graph, you are free to also use any other JCF collections/classes you wish to implement the required features and algorithms, etc.
- The system should be comprehensive and ideally use a full dataset for the full London underground.
    - CSV data for the full London underground can be found, for instance, here.
    - You may decide to augment any downloaded data with additional data if it helps with your functionality.
- The application should provide a suitable user-friendly JavaFX graphical user interface.
    - Routes should be shown in some easily readable and user-friendly fashion.
        - For instance, you could also use a TreeView control to show the route(s) as a collapsible hierarchy.
        - You could also use some suitable container/view (e.g. a scrolling list) to show stations (and the lines connecting them) encountered along a route.
        - Note that routes should clearly show both the stations and the lines used (the line name and/or the line colour should be used as part of the route information e.g. the Central line is “red”, the Circle line is “yellow”, etc.).
    - Other JavaFX controls might also be used as appropriate to provide for a nice, intuitive, and coherent user experience.
- A geographical map should also be used to show an on-screen representation of routes in Zone 1 only (for simplicity). A sample geographical map for Zone 1 (link):



This map (or similar) could be edited to make it "blank" by removing station names, and greying-out all line colours. Then, to show a Zone 1 route on the blank greyscale map, coloured lines (appropriate to the tube lines travelled) could be drawn/superimposed to show stations on the identified routes, along with station names. Again, this functionality only applies to routes that are entirely within Zone 1.

- You should use Dijkstra's algorithm (or a variant such as uniform cost) to identify the shortest route between two stations based on Euclidean distance.
    - You can easily calculate Euclidean (or straight-line) distance between stations using longitude and latitude (Google to see how to do this).
- You should also use Dijkstra's algorithm (or a variant such as uniform cost) to identify the shortest route between two stations based on Euclidean distance, but where any line/train changes on route incur an additional cost penalty.
    - How severe this penalty is should be user-specified. The more severe the penalty, the more likely that the algorithm will choose a route that minimises or avoids line/train changes.
- A route may have specified waypoints (one or more stations) between the starting station and destination that the route(s) must go through. Waypoints should ideally be supported in some way in all route-finding operations (single route (BFS), multiple routes (DFS), shortest route (DA), etc.)
- Similarly, it should also be possible to specify stations to avoid on the route.

Indicative Marking Scheme

- Custom graph data structure/classes = 10%
- Generate any single valid route between two stations = 10%
- Generate multiple valid route permutations using DFS = 10%
- Shortest route using Dijkstra's algorithm = 10%
- Shortest route with line/train change penalties using Dijkstra's algorithm = 10%
- Illustrating Zone 1 routes on geographical map = 15%
- Waypoint support = 5%
- Avoiding specified stattions = 5%
- JavaFX GUI = 10%
- JUnit testing = 5%
- JMH benchmarking of key methods = 5%
- General (overall completeness, structure, commenting, logic, etc.) = 5%

Note that it is not expected that all students will attempt all aspects of this assignment. Use the above marking scheme to guide your efforts as this is what you will be marked against.