**C Coursework – Tube Navigation**

Open the terminal and type the command “gcc -o execute tube.c” press <return> then “./execute” and press <return> to execute the program. A menu will appear. Type the number of the option you want to select (1 – calculate the time needed to go between the stations, 2 – calculate the fastest way between the stations, 3 – calculate the route with the least number of stations between the stations, 4 – quits the program) and press <return>.

A screen shot of a social media post

Description automatically generated

You will be asked to give the name of the start station and the destination station (the program will remove any characters that are not letters and will convert any letter in lowercase, so you can write the name of the station in any way). Once the program receives the stations, it prints the route, telling what line to take and where to change. The menu appears again until the user closes the program.

A screenshot of a cell phone

Description automatically generated

I have chosen this question because I have always been impressed by Google Maps, Google Earth and other map kind apps and I thought that this is a great chance to implement my own navigation.

I’ve started by doing some research about the London Underground and I found out that it is possible to go between two any stations by changing the line twice (Jubilee line having common stations with any other line). My first version of the algorithm was able to find the common line and use that in order to go to the destination. Unfortunately, that was not a good idea at all because it takes too much time. In my second version I used **Dijkstra's algorithm** which I learned in high school. I also found all the stations on a website, the lines and the connections between two stations (which I needed in order to build the graph) and the time between two stations. In my code, I use 3 matrices, the first one that stores in node conn\_time[x][y] the time that takes to go between station x and station y, the second one every node conn\_stat[x][y] is 1 if station x is connected with station y. The third one stores in the node conn\_line[x][y] the line which connects station x and station y. I implemented 2 functions, one for the start station and one for the final station (startStation and finalStation), which ask for a valid input (if the program doesn’t recognize the input as a valid one, the user will be asked to write the name of the station again). Each function is called if the user chooses one of the options 1, 2 or 3. After the user gives the input, 2 functions (toLowerCase, justLetters) remove from the input every nonletter char and every letter in converted to lowercase (In this way errors such as forgetting the “.” from “King's Cross St. Pancras” or writing with “k” instead of “K” will be avoided). After the conversion is done, the program searches for the name of station through the station lists and returns its id when it finds it in the list( findID). The function printPath it where the route and the station where a change is needed are printed and it is called in path function, where, depending on the selector (the option chose from the menu), the program displays the route or the time. The path function is called in the calculatePath function, where **Dijkstra's algorithm** is implemented. The last function is all, where takes a parameter which is the selector and using a switch...case choose what the program must do.

**Options**

**For option 1:** The program calculates the approximate time between 2 stations using Dijkstra's algorithm and the conn\_time graph. The algorithm calculates the route which takes the least time, choosing the stations between which the tube takes the least time and displays the result.

**For option 2:** The program calculates the approximate time between 2 stations and, also prints the route, using **Dijkstra's algorithm** and the conn\_time graph. Moreover, the program indicates the stations where the user needs to change the line and what line to take by using an array containing the id of the lines on the route. Where the id changes, means the user needs to change the line.

**For option 3:** The program calculates shortest the distance between 2 station (with the least number of station), using **Dijkstra's algorithm** and the conn\_stat graph. As before, the program indicates the stations where the user needs to change the line and what line to take by using an array containing the id of the lines on the route. Where the id changes, means the user needs to change the line.

For option 4: Closes the program (in function run).

Websites used:

<http://markdunne.github.io/2016/04/10/The-London-Tube-as-a-Graph/>

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

<https://en.wikipedia.org/wiki/List_of_London_Underground_stations>