2.1 Implement DFS, BFS and UCS

In this question, I have implemented BFS (Breadth-First Search) for finding the shortest path b/w any given two stations, DFS(Depth-First Search) for node exploration b/w any two given stations and finally, UCS(Uniform-Cost Search) using a priority queue to explore paths be/w stations in order of increasing costs. We have also created the main functions for the same (bfs,dfs_path,ucs_path) which take the station graph, Starting station and destination station as parameters. Moreover, I have displayed the total cost(time) b/w the starting and goal stations as well as the number of nodes expanded. I'll cover more on them in the next part of the question.

2.2 Compare DFS, BFS and UCS

Paths Tested	BFS	DFS	UCS
Tottenham Court Road to Stratford	Average time : 18 Number of nodes expanded: 8	Average time: 94 Number of nodes expanded: 45	Average time: 18 Number of nodes expanded: 8
Baker Street to Wembley Park	Average time : 13 Number of nodes expanded: 2	Average time: 61 Number of nodes expanded: 25	Average time: 13 Number of nodes expanded: 2
Canada Water to Stratford	Average time : 15 Number of nodes expanded: 5	Average time: 24 Number of nodes expanded: 11	Average time: 14 Number of nodes expanded: 7
Ealing Broadway to South Kensington	Average time : 20 Number of nodes expanded: 8	Average time: 89 Number of nodes expanded: 43	Average time: 19 Number of nodes expanded: 8

Personally, I do not believe that any method is better than the latter since there is often a trade-off b/w the three when it comes to measuring the key significant factors namely: optimality, memory utilization and efficiency. BFS is designed to find the shortest path from the start to the goal and also handles unweighted graphs quite efficiently, UCS happens to find the most optimal solution in terms of cost and is quite proficient when it comes to handling graphs with varying costs. however, DFS on the other hand doesn't guarantee the same; it's aimed more at node exploration and handling problems with deep paths. However, amongst the three, UCS seems to be the most memory-exhausting followed by BFS and finally, DFS using up the least amount of memory, since they do not include the whole search tree in memory. A set for the nodes visited has been put in place to ensure that the algorithms do not revisit the nodes during the exploration of a specific path, tackling the subject of getting stuck in infinite loops.

2.3 Extending the cost function

```
UCS path from Tottenham Court Road to Stratford: ['Tottenham Court Road', 'Holborn', 'Russell Square', 'Stratford']
Average time: 8
```

After thorough observations throughout, I believe the path selection for BFS and DFS remains unchanged after the line change because they happen to take equal-cost paths simultaneously but the line change might affect the actual travel time though. However, the path selection for UCS might be altered where it might

favor paths with fewer line changes if the travel time is longer in terms of station-to-station distances since the end goal is look for the most optimal solution.

2.4 Heuristic Search

My idea is based on the presumption that the stations in the same or nearby zones might be closer in terms of distance or time taken to travel. The formula is based on the number of zones between the current station and the end goal. The central idea is to take paths traversing fewer zones.

Path tested	UCS	BFS with Heuristic
Tottenham Court Road to Stratford	Average time: 18 Number of nodes expanded: 8	Average time: 30

As per my observation, The BFS with heuristic follows a more guided thorough search, efficiently exploring the search space while the UCS keeps the optimal interests only with total costs.

3.1 Genetic Algorithm

```
True Password: 4PNPF4PW91
Found Password: 4PNPF4PW91
```

3.2 Algorithm Components

The state is represented as a list of strings, where each string happens to be a candidate password. In this implementation of the selection method, the process involves sorting the population based on fitness scores and selecting the top 20% as parents for reproduction. Whereas, in the crossover method, a random crossover point is chosen, and two parents bestow parts of their genetic material to form a new child password. After Crossover, there is a mutation stage where a random mutation point is chosen and a random character is inserted at that position.

3.3 Number of reproductions

```
Average Generations: 112.8
Standard Deviation of Generations: 52.65890238126883
```

3.4 Hyperparameters

With a mutation rate of 0.1 :

```
Average Generations: 112.8
Standard Deviation of Generations: 52.65890238126883
```

Mutation rate is 0.2

```
Average Generations: 53.8
Standard Deviation of Generations: 20.178206064960282
```

As we can see, the mutation rate of 0.2, resulted in a significantly lower number of generations and a smaller standard deviation, implying a more consistent implementation across multiple runs.