Paris Metro: Exploring the Paris Subway Network

Implementation and Experiments Report

Data Structures:

The Paris metro network is represented as a graph, where each station is a vertex, and the connections between each station are edges. My chosen data structure is an adjacency list, implemented using ArrayLists, which efficiently stores the connections between vertices and is simple to implement.

Two classes, Node.java and Edge.java, were also created to encapsulate the individual vertices and edges of the graph. Node.java stores information about a station, and Edge.java represents the connection between 2 stations, storing the destination station and the weight of the connection.

ArrayLists are extensively used in my program to store various data structures, such as the list of adjacent vertices, the distance and previous node information in Dijkstra's algorithm, and the reconstructed path. ArrayLists provide dynamic resizing and convenient methods for manipulating lists, making them essential for the implementation.

A priority queue pq is also used in the implementation, specifically for Disjktra's algorithm. It is utilized to maintain a sorted order of nodes based on their distances, which allows for efficient retrieval of the node with the shortest known distance at each step of the algorithm.

Algorithms:

Depth-First Search (DFS)
The DFS algorithm is used to identify stations on a line, given a starting station. The algorithm starts at the chosen station and explores as far as possible along each branch before going back again and reconstructing. This process continues until all stations on the same line are discovered. The algorithm leverages a recursive approach to navigate through the network, marking visited stations to prevent revisiting and infinite loops.

Dijkstra's shortest path algorithm
Dijkstra's algorithm is used to find the shortest path between 2 stations in the Paris metro network. It considers the weight of each edge, which represents the travel time between stations. The main idea is to maintain a priority queue that efficiently selects the station with the shortest known distance at each step. The algorithm proceeds by iteratively selecting the station with the smallest edge weight, updating distances to adjacent stations, and continuing until the end station is reached. This guarantees the discovery of the shortest path.

Handling out of commission stations
To handle the out of commission stations for Question 2.iii) the Dijkstra's algorithm is adapted. When an out of commission station is specified, the algorithm avoids paths that include this station. This is implemented through a boolean array, where each index

corresponds to a station, and the array marks the stations that should be avoided during the pathfinding process. By excluding out of commission stations, the algorithm ensures a reliable representation of the available routes in the Paris metro network.

Experimentation:

Below are screenshots of the outputs when running my program:

Q 2.i) Using only N1:

```
C:\Users\mbelk\Desktop\CSI2110\P2_300301354>javac ParisMetro.java

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 0
line: [0, 238, 322, 217, 353, 325, 175, 78, 9, 338, 308, 347, 294, 218, 206, 106, 231, 368, 360, 80, 274, 81, 178, 159,
147, 191, 194, 276]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 1
line: [1, 12, 213, 235, 284, 211, 86, 21, 75, 142, 339, 151, 13, 5, 239, 27, 246, 302, 366, 204, 85, 351, 56, 362, 256]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 4
line: [4, 171, 264, 286, 370, 189, 367, 103, 327, 132, 223, 289, 38, 336, 309, 8, 350, 312, 229, 305, 285, 115, 263, 114
, 251]
```

Q 2.ii) Using N1 and N2:

```
C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 0 42
Time = 996
Path: [0, 238, 239, 5, 13, 151, 339, 142, 75, 21, 86, 211, 284, 235, 1, 12, 213, 215, 42]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 56 221
Time = 760
Path: [56, 55, 127, 109, 50, 77, 78, 9, 338, 308, 347, 346, 174, 221]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 7 172
Time = 951
Path: [7, 8, 309, 336, 38, 289, 223, 225, 177, 79, 138, 158, 371, 156, 76, 111, 35, 172]
```

Q 2.iii) Using N1, N2 and N3

```
C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 0 42 1
Time = 1021
Path: [0, 159, 147, 191, 192, 64, 14, 124, 121, 65, 342, 344, 315, 220, 316, 369, 58, 307, 215, 42]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 56 221 0
Time = 763
Path: [56, 57, 150, 30, 354, 230, 26, 98, 155, 154, 349, 99, 358, 346, 174, 221]

C:\Users\mbelk\Desktop\CSI2110\P2_300301354>java ParisMetro 7 172 4
Time = 960
Path: [7, 290, 136, 68, 67, 173, 227, 356, 77, 79, 138, 158, 371, 156, 76, 111, 35, 172]
```

Overall, the results of the experiments demonstrate the effectiveness of the implemented algorithms in providing accurate and efficient solutions to the given tasks.