Project: Paris Metro: Exploring the Paris Subway Network
Michael Massaad

1. Construction of Graph class and Implementation

In this assignment, we are required to build a directed graph for the Paris subway network provided in the metro.txt file. In my implementation, I am representing the graph using an adjacency map as seen in my lab. I am using 2 linked lists to store the vertices, which in this situation are the stations, and the edges, which are the connections between stations in the graph. Since we are implementing a directed graph, I created a nested vertex class which will store the information on a station, whether it being the name (as a string), number (as an integer), outgoing and incoming edges (stored as a concurrenthashmap which had the opposite station as the key and the weight as the value, the usage of concurrenthashmap was advised by ChatGPT along with a fellow classmate, Jayson, to resolve a concurrentmodificationexception that I was receiving, https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html). I had also created a nested edge class which will store the weight of the edge (as an integer) and the endpoints (stored as an array of size 2). I also implemented all the basic graph methods, using the guidance of the provided lab 8 implementations. In this assignment, we also had to implement a ParisMetro class where the operations for the Paris metro network is delt. In that class, I keep a static reference to the graph of the metro system, and I implemented the readMetro method to read the metro.txt file. In that class, I also use the priority queue, array list, hash map and hash table data structures imported from java.util in the implementations of the operations for the Paris metro network.

2. Algorithms Used to:

a) Identify all stations belonging to the same line of a given station.

To solve this question theoretically, we start with a given station from which we would traverse through all its connecting stations. These connecting stations are found by observing the endpoints of the weighted edges. As we have seen in class, the DFS algorithm allows us to "visit[s] all the vertices and edges in the connected component of v" for a given vertex v. Therefore, the DFS algorithm was used in my implementation to solve this part of task 2. However, as it is mentioned in the instructions, an edge with a weight of -1 indicates that the opposite vertex is not a connected station, so we are not supposed to consider that edge as a connection to a station belonging to the same line. In my implementation, I ensured to not include those stations using an if-statement to check the weight of each edge. Since I am using an adjacency map/list structure for implementing my graph, the time complexity of this algorithm would be O(m), where m is the number of edges.

b) Find the Shortest Path Between any Two Stations, i.e. The Path Taking the Minimum Total Time. Print all the Stations of the Path in Order, and Print the Total Travel Time.

Dijkstra's algorithm computes the distance of all the vertices from a given start vertex s. Therefore, to find the shortest path between a station to all the other stations, I implemented this algorithm using the guidance of the implementation from lab 8. However, since this part wants us to find the shortest path between a starting station and a destination station, during the implementation of Dijkstra's algorithm, I keep track of the order of which the stations are traversed and the resulting weight (time) it took to arrive at each station. I used this so that when the algorithm had completed its job of finding the shortest path to all the stations, I backtrack from the destination to the starting station, and I stored the stations that were traveled in a list. Since I am using the java.util priority queue, which is implemented using a minimum-heap, the overall worst time complexity for this part, as we have seen in class, would be O(m*log(n)), where m is the number of edges.

c) Find the shortest path between two stations when we have the information that one given line is not functioning (the line is identified by one of its endpoints). The same type of information will be printed as in ii).

Since this part also want us to find the shortest path between two stations, my implementation for the Dijkstra's algorithm used in section b is also used for this part. However, since we are given a third station which indicates the line that is no longer functional, when finding the shortest path, it would modify the time and the stations that were traversed along the path since there would be some stations that cannot be accessed. Therefore, to find the non-functional stations/line, we use the DFS algorithm used in part a to find all the stations that belong to the same line as the third station. We then disconnect the connections (edges) of those stations in the graph, then we run our Dijkstra algorithm with our updated system, which would return a new shortest path that takes an alternate route than the original graph, depending on which stations are no longer functional. This part has a worst time complexity of O(m + m*log(n)) = O(m*log(n)).

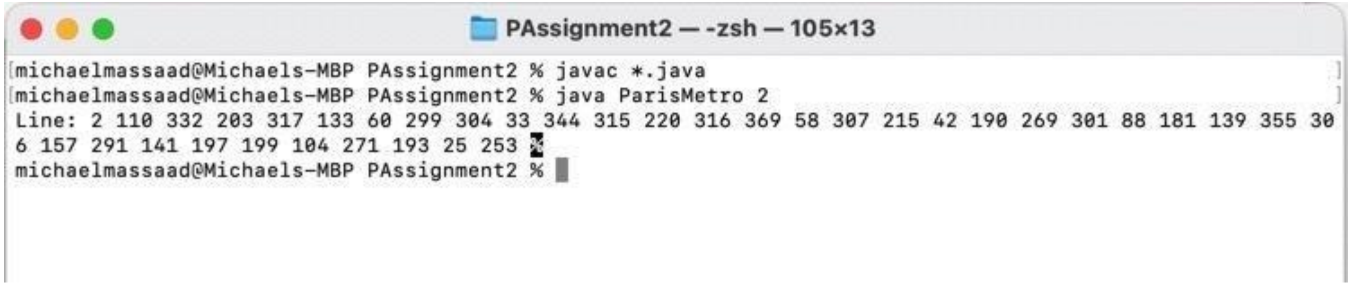
3. Example Outputs for Questions from Part 2

Question a)

Q1.Ex. 1

Inputs: N1 = 2

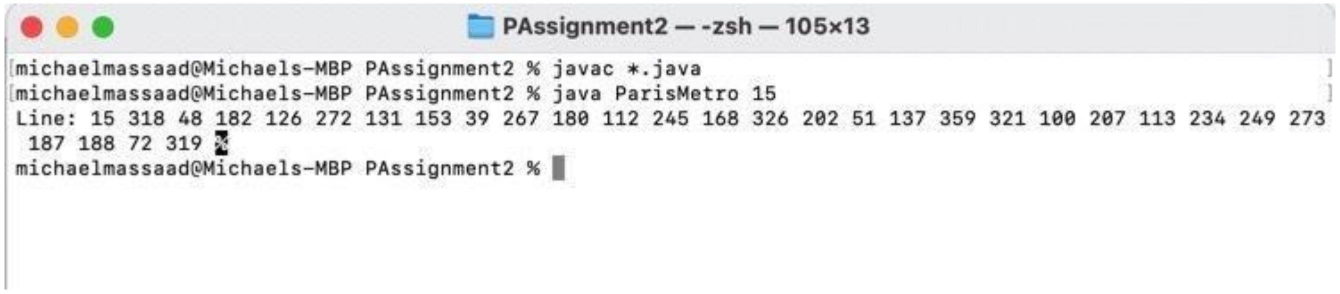
Output:



```
PAssignment2 — -zsh — 105×13
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 2
Line: 2 110 332 203 317 133 60 299 304 33 344 315 220 316 369 58 307 215 42 190 269 301 88 181 139 355 30
6 157 291 141 197 199 104 271 193 25 253 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

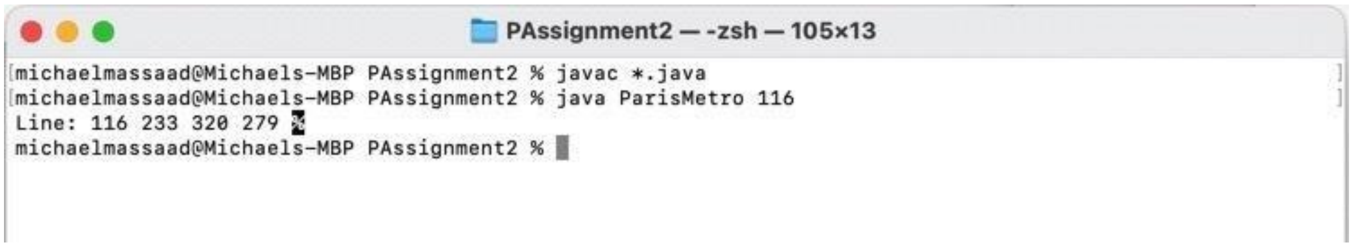
**Just to clarify, the vertex number between 355 and 157 is 306

Q1.Ex. 2
Inputs: N1 = 15
Output:



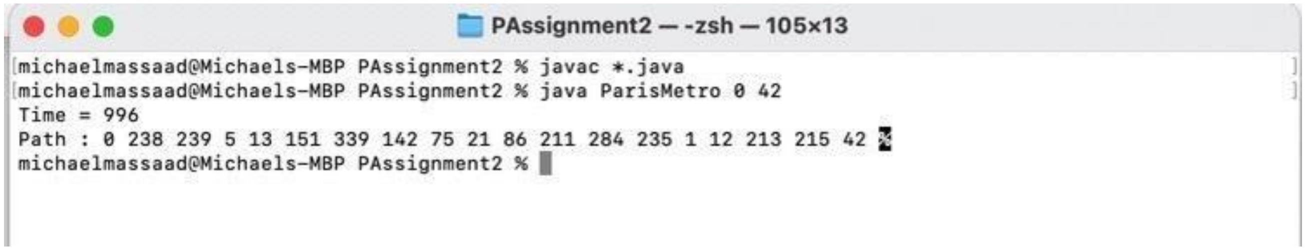
```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 15
Line: 15 318 48 182 126 272 131 153 39 267 180 112 245 168 326 202 51 137 359 321 100 207 113 234 249 273
 187 188 72 319 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

Q1.Ex. 3
Inputs: N1 = 116
Output:



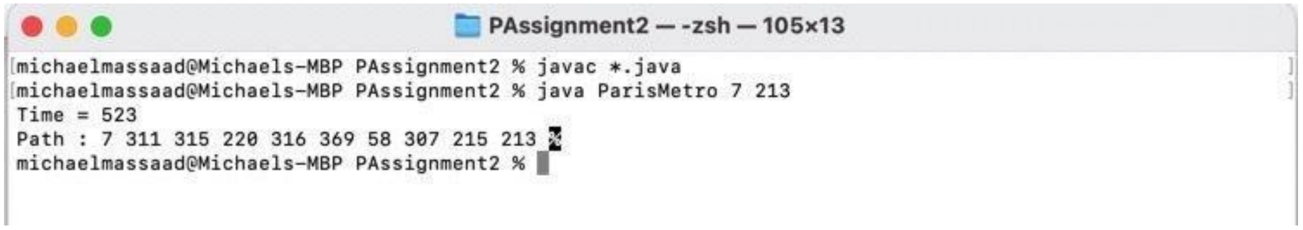
```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 116
Line: 116 233 320 279 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

<u>Question 2</u>
Q2.Ex. 1
Inputs: N1 = 0   N2 = 42 Output:



```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % 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 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

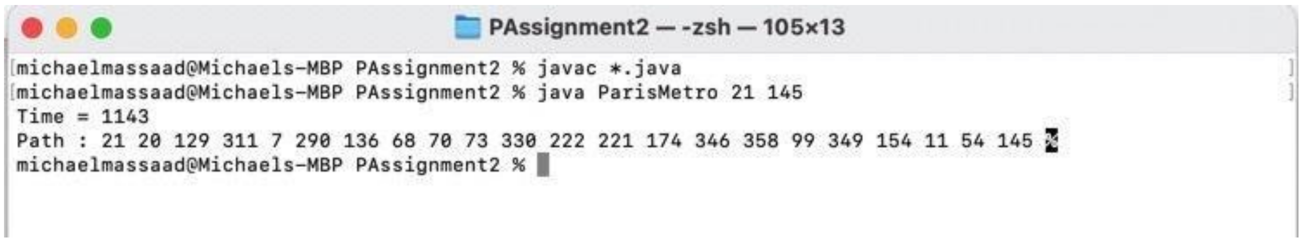
Q2.Ex. 2
Inputs: N1 = 7   N2 = 213 Output:



```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 7 213
Time = 523
Path : 7 311 315 220 316 369 58 307 215 213 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

Q2.Ex. 3

Inputs: N1 = 21 N2 = 145 Output:



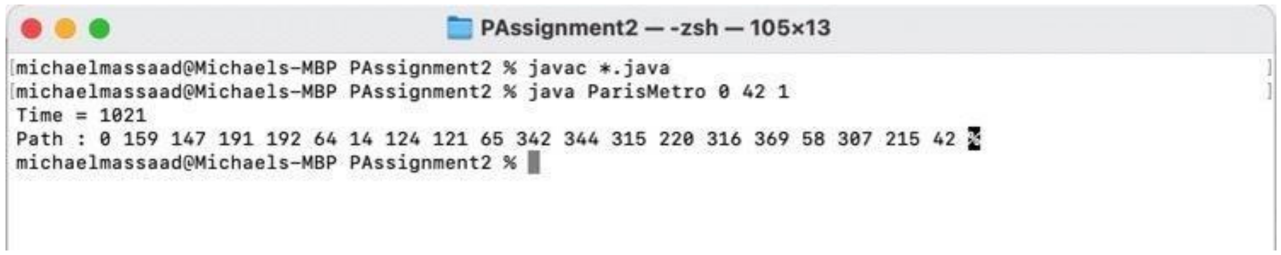
PAssignment2 — -zsh — 105×13

```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 21 145
Time = 1143
Path : 21 20 129 311 7 290 136 68 70 73 330 222 221 174 346 358 99 349 154 11 54 145 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

<u>Question 3</u>

Q3.Ex. 1

Inputs: N1 = 0 N2 = 42 N3 = 1 Output:

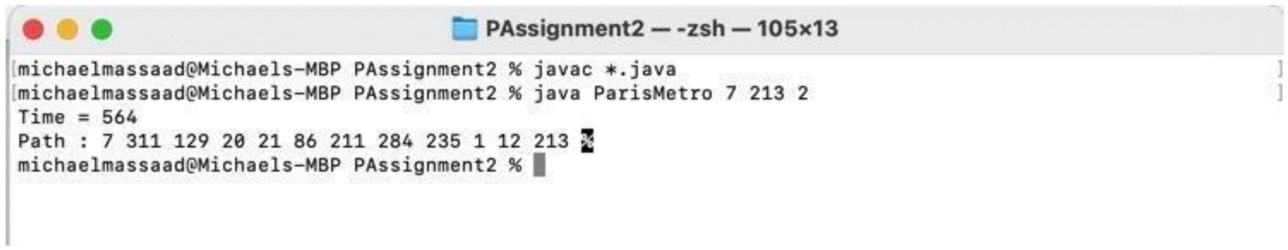


PAssignment2 — -zsh — 105×13

```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % 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 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

Q3.Ex. 2

Inputs: N1 = 7 N2 = 213 N3 = 2 Output:



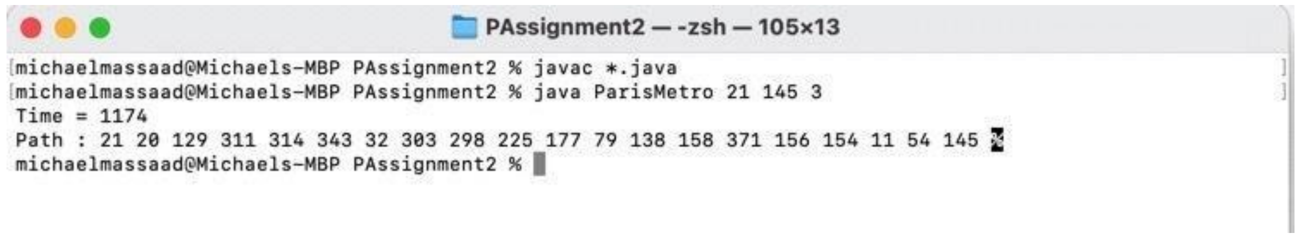
PAssignment2 — -zsh — 105×13

```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 7 213 2
Time = 564
Path : 7 311 129 20 21 86 211 284 235 1 12 213 %
michaelmassaad@Michaels-MBP PAssignment2 %
```

Q3.Ex. 3

Inputs: N1 = 21 N2 = 145 N3 = 3

Output:



PAssignment2 — -zsh — 105×13

```
michaelmassaad@Michaels-MBP PAssignment2 % javac *.java
michaelmassaad@Michaels-MBP PAssignment2 % java ParisMetro 21 145 3
Time = 1174
Path : 21 20 129 311 314 343 32 303 298 225 177 79 138 158 371 156 154 11 54 145 %
michaelmassaad@Michaels-MBP PAssignment2 %
```