P2 Report

ID: 300283854

Name: Matin Mobini

i) Description of Chosen Data Structure:

My implementation of the graph that represents the subway network is an AdjacencyMapGraph which is based upon the implementation of AdjacencyMapGraph given in the lab. This graph includes two data structures InnerVertex and InnerEdge, these subclasses of AdjacencyMapGraph allow the ParisMetro class to handle and compute several methods related to a station's vertex and its incoming and outgoing edges. The AdjacencyMapGraph manages the InnerVertex using a ConcurrentHashMap which allows for concurrent modification which is required in the case of removing vertexes which we must do in task 2iii). The InnerEdge subclass is untouched from the lab. To store the vertices and edges of the AdjacencyMapGraph I store them each in a LinkedList as that is what I prefer for this situation, although the vertices and edges could be stored as different data structures. Within my ParisMetro class, I store a variable vistedStations which I use to track the visited stations when completing the DFS method, I store vistedStations as an ArrayList of vertexes.

ii) Description of Algorithms:

Depth-First Search(DFS): I used DFS to identify all stations that belong to the same line as a given station. The DFS method recursively traverses adjacent vertexes while also marking them as visited within the vistedStations variable which will be used in other methods.

Dijkstra’s Algorithm: I referred to the lab's implementation of Dijkstra’s Algorithm and altered it to be used for task 2ii/2iii), the method finds the shortest path between two reachable stations within the subway. It completes this task while tracking the time it would take to travel to the desired station.

iii) Examples of Outputs:

2i) which is for a given station, identify all stations on the same subway line:

- **Output Example 1:** Input: 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
- **Output Example 2:** Input : 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
- **Output Example 3:** Input: 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 , 306 , 157 , 291 , 141 , 197 , 199 , 104 , 271 , 193 , 25 , 253

2ii) which is calculating the shortest path between two reachable stations:

- **Output Example 1:** Input: 0 42 Time = 996
  Path: 0 , 238 , 239 , 5 , 13 , 151 , 339 , 142 , 75 , 21 , 86 , 211 , 284 , 235 , 1 , 12 , 213 , 215 , 42
- **Output Example 2:** Input: 0 247 Time = 766
  Path: 0 , 238 , 239 , 5 , 13 , 151 , 339 , 142 , 144 , 31 , 41 , 34 , 248 , 247
- **Output Example 3:** Input: 0 288 Time = 957
  Path: 0 , 159 , 147 , 191 , 192 , 64 , 14 , 124 , 125 , 122 , 140 , 313 , 219 , 297 , 40 , 17 , 288

2iii) which is for a certain non-working line(found using DFS from a given station), calculate the shortest path between two reachable stations:

- **Output Example 1:** Input: 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
- **Output Example 2:** Input: 0 247 1 Time = 923
  Path: 0 , 159 , 147 , 191 , 192 , 64 , 14 , 124 , 125 , 340 , 143 , 144 , 31 , 41 , 34 , 248 , 247
- **Output Example 3:** Input: 0 288 3 Time = 1051
- Path: 0 , 238 , 322 , 217 , 353 , 325 , 175 , 78 , 77 , 356 , 227 , 173 , 67 , 135 , 331 , 16 , 17 , 288