

Assignment 4 Report

For this assignment, I am asked to read a given file containing subway stations and lines, then traverse through the graph to identify the line starting from the station entered by user, compute the shortest path between two stations user entered, and the shortest path if one station malfunctions.

I have used net.datastructures implemented by Goodrich et al. (specifically, AdjacencyMapGraph to store the graph, HeapPriorityQueue to store the distances for computing shortest paths, and ProbeHashMap to store all the vertices in the cloud) and some lab solutions to solve this assignment. I have also used an array list to store all vertices on the same line as the malfunction station.

To identify the line, one should do the Depth-First traversal to the graph. The idea is to start from the source and visit the next vertex connected to the source vertex, and mark this vertex as visited and set this vertex as the new starting point. We keep doing this traversal until we have visited all vertices on the line. I used a hash table to store all visited vertices so that if we encounter a cycle we will exit the traversal.

For computing the shortest path, one can use Dijkstra's algorithm. The algorithm simply calculates the shortest distances(time in this assignment) to every other vertices the source vertex can reach. We have a cloud of visited vertices(vertices that already have the shortest distance computed). Initially our cloud has only one vertex, which is the source vertex, it has a distance 0 since traveling from the source vertex to the source vertex has a distance 0 and it is indeed the shortest distance. Then we set distances to all other vertices to ∞ since we have not visited them yet and we have not calculated the distances. Once we start visiting vertices, we will perform edge relaxation step, where we update the shortest distance by comparing the original shortest distance with the new distance with the weight of edges added to it. If the new distance is shorter, then the new shortest distance will be that distance, and the new vertices will be added to the cloud. In the case where a station malfunctions, we simply skip the vertices that are on the same line as our malfunction station while performing edge relaxation so that these vertices will not be in the cloud.

To show the shortest path from one vertex to another, I have used an array storing integers(since our vertices are represented as integers). Each index represents the vertex, and the content of each index is the previous vertex along in the shortest path. Whenever an edge relaxation step happens, I will add the previous vertex(current source vertex) to this array at the vertex index(current destination vertex). Then I print it by reversely tracing the indexes starting with the destination vertex.

Here are some examples of the outputs of my program:

```
C:\Users\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 0
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\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 0
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\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 1
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\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 2
2 139 355 306 157 291 141 197 199 104 271 193 25 253 110 332 203 317 133 60 299 304 33 344 315 220 316 369 58 307 215 42 190 269 301 88 181
C:\Users\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 3
3 210 95 292 361 208 333 334 323 222 330 73 70 165 375 310 342 65 121 124 14 64 192 337 268 262
C:\Users\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 0 42
0 238 239 5 13 151 339 142 75 21 86 211 284 235 1 12 213 215 42
Time: 996

C:\Users\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 0 42 1
0 159 147 191 192 64 14 124 121 65 342 344 315 220 316 369 58 307 215 42
Time: 1021

C:\Users\wmq98\Desktop\Data Structure Assignments\4>java ParisMetro 0 247
0 238 239 5 13 151 339 142 144 31 41 34 248 247
Time: 766
```

```
C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 0 247 1
0 159 147 191 192 64 14 124 125 340 143 144 31 41 34 248 247
Time: 923

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 0 247 31
0 238 239 5 13 151 339 142 75 21 20 283 146 247
Time: 784

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 0 288
0 159 147 191 192 64 14 124 125 122 140 313 219 297 40 17 288
Time: 957

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 0 288 3
0 238 322 217 353 325 175 78 77 356 227 173 67 135 331 16 17 288
Time: 1051

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 14 15
14 13 5 239 27 246 245 153 131 272 126 182 48 318 15
Time: 865

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 14 15 1
14 64 192 191 147 159 0 238 322 217 353 325 326 168 245 153 131 272 126 182 48 318 15
Time: 1125

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 14 97
14 124 121 65 342 344 315 220 316 369 58 307 215 214 236 19 93 97
Time: 913

C:\Users\wmq98\Desktop\Data Structure Assignments\A4>java ParisMetro 14 97 2
14 124 121 65 342 343 314 107 335 61 18 163 105 296 205 94 93 97
Time: 914
```

(References concerning the methods I used from other sources are available in the comments of my program).