CSI2110
Assignment 4
Fall 2017
Due Date: 2017-12-06

Nevin Wong Syum Ganesan - 8831598

Mabroor Kamal - 8669942

A4 GROUP 101

Questions:

You must submit your Java code plus a document/report (pdf file) describing:
• Your chosen data structure (classes, attributes and methods).
• High level description of the algorithms you use to answer the questions in part 2.
• Examples of outputs for answering the questions of part 2.

Data Structures Used

- Graph (sGraphStored) to store and build the graph
- Arraylist (to store the list of IDs and Station Name from line 1 to line 377)
- AdjacencyMapGraph. (to initialize a directed Graph for sGraphStored)
- Hashset (to store data in same line/path for question 2.1 & 2.3)
- Hashtable (to store vertices while finding shortest path)
- LinkedList (to store vertices/edges before printing)
- Maps from Lab and java.util.Map (to store cloud and use java map methods)
- MyEntry (to store ID and Station name in String <String,String>)
- Set (to use DFSHelper for recursion)
- Edge (to use DFSHelper for recursion)

Instance Variables/Attributes

(explanation in comment headers of the code in ParisMetro.java)

```
Graph<String, Integer> sGraphStored;
private ArrayList<MyEntry<String,String>> listEntry;
private int RecordedNumVertices; private int RecordedNumEdges;
private String vertStored;
private final static Integer weightTimeConstant=90;
private boolean question1; private boolean question2; private boolean question3;
private boolean done;
private String vertStoredDes;
private String vertDestroyed;
private HashSet<String> hashSet;
```

Explanation for Constructors

1)public **ParisMetro**(String fileName) throws Exception, IOException {}

This is our default constructor (for testing one point shortest path), if no arguments are entered in our main method then this would be the first constructor to be executed. What this simply does is that we create a directed adjacency map graph called sGraphStored which takes in 2 generic parameters which is String and Integer. The integer will represent the weight of an edge (time taken from one vertex to another in seconds) , and the string will represent the vertices ID (with non - leading zeroes). This is similar to lab 10 code structure.

2)public **ParisMetro**(String fileName, String vertStored) throws Exception, IOException {}

To handle Question 2.1 of the assignment:

This is our constructor that is called once a parameter in the main method is given such that we are given the a String called vertStored. This parameter will represent N1. Simply again we will also create sGraphStored which again will take the 2 generic parameters. This time we will also set some flags, in particular boolean question1=true, so the other methods of the class will know how to execute the algorithm differently. For example, here we will need to build the graph without considering the -1 edges in our metro.txt file, so we can use DFS to find our same line given N1. Since Boolean question1is set to true, it will break out from the loop in read metro when it comes to reading the line of weight -1.

3)public **ParisMetro**(String fileName,String vertStored,String vertStoredDes) throws Exception, IOException {}

To handle Question 2.2 of the assignment:

This is our constructor which is used when we are given not only N1 but also the parameter N2. We will again create sGraphStored which again will take the 2 generic parameters. And again we will set our Boolean question2 to true and other flags to false. Essentially is the same as our previous constructor given above, this time we are just given another parameter which will represent N2. This value of N2 will be stored in a private String called vertStoredDes which represents our destination.

4)public **ParisMetro**(String fileName,String vertStored,String vertStoredDes,String vertDestroyed) throws Exception, IOException {}

To handle Question 2.3 of the assignment:

Lastly our most viable constructor would be this one where it takes in 3 extra parameters. vertStored(N1), vertStoredDes(N2), and vertDestroyed(N3). We will create another private String which will essentially store vertDestroyed. We also created the HashSet. This HashSet will contain the String of vertices of all the destroyed vertices. If N1, N2 and N3 are specified the program must answer question 2-iii) where N1 is the departure station, N2 is the arrival station and N3 is the endpoint of a broken line (line that is not functioning). We will add this to the HashSet which stores these vertices.

Similar to handle Question 2.1 to build the line and in order to add it to the HashSet, we reused the method called printSameLine() which essentially call printDFS(), and then we it will call the DFSComplete method which then will complete and call DFSHelper recursively. This is basically the same code from the lab but just renamed and modified. But the printSameLine() and printDFS does not print anything for question 2.3 since boolean question3=true , boolean question1=false , question2=false.

5)protected void **readMetro**(String fileName) throws Exception,IOException{}

When readMetro is called, we will build a graphFile, which will take in our fileName. We will also create a HashTable which will contain all of the vertices. We will create a string called Line and a counter for our loops. We need a counter to read the lines differently. So we added 3 if statements while the lines if not finished reading.

Firstly, the method will record the number of vertices and number of edges, which is useful for checking if we have the correct graph. Secondly, the station ID and station name will be stored in the ArrayList. Thirdly, similar to the lab code, the method will build a graph by adding vertices and edges if it's not present in the graph.

There are 2 exceptional cases will break out of this loop. First exceptional case to break is when question 1 is true in order to build the same line/path. Second case, is to build the broken line for N3. I have added to if statements for clarity purpose. Note that if the weight is -1, we will substitute it with constant time of 90 seconds.

6)protected Vertex<String> **getVertex**(String vert) throws Exception {}

This method will just gets the vertex from the graph stored (sGraphStored). Throws an exception if the Vertex is not found. The Vertex stores the ID of the station in String.

7)public void **print**() {}

This method will print all the vertices in the list, followed by printing all the edges. We used this for debugging.

8)public static String **readVertex**() throws IOException {

When default constructor is called, we used this for debugging the shortest of one path to all the points. To check if we have the correct method of finding the shortest path before proceeding to Question 2.2 & 2.3.

//--------To Solve Question 2.1, we used printSameLine(),printDFS(),DFSComplete and DFSHelper

9)public void **printSameLine**() throws Exception{}

This method begins with us creating our variables. We will create a String called vertStoredID which will contain our stored Vertex. A Vertex List that will hold our stored vertices. A MyEntry mineEntry will be created. Here we will use our flags that we created in our constructors. This will test all of our question(s) 1, and 3. If question1 is true then we will print all the stations in the same line using the list we created which stored all the vertices in the same line. If question3 is true then will also essentially do the same algorithm but we will not print anything, because we want to reuse this method to add N3 broken line into our HashSet.

10)public void **printDFS**(String vert) throws Exception{}

Print DFS is a helper method for printSameLine() that will basically print the DFS of a given vertex in the argument. For question 1 it prints, and for question 3 it will add it to the HashSet which was created. For this we will create an endLocationID which is given the calue from finding the key for the ent variable in the for loop. Once we obtain the endID. We created a mineEndEntry list which will store the endLocations in the entries.

11)public <String,Integer> Map<Vertex<String>,Edge<Integer>>
DFSComplete(Graph<String,Integer> g) throws Exception {}

DFS complete basically creates a known set which will store the known vertices. We also create a forest which will be a new ProbeHashMap. We then run a for loop which takes all of the edges and vertices and if we already have a known vertex we will run the DFSHelper method.

12)private <V,E> void **DFSHelper**(Graph<V,E> g, Vertex<V> u, Set<Vertex<V>> known, Map<Vertex<V>,Edge<E>> forest) {}

The DFS helper method takes 3 arguments. From this DFS Helper we consider we found u. Then for every outgoing edge from u (source). We mark them as visited and we keep going down the tree. Once we reach the end, we will go again to the top and go down the forest until we found an unfound vertex.

//-------- end of question 2.1

13)void **printAllShortestDistances**(String vert) throws Exception { }

This handles the default method of the shortest path .We also created a sorted priority queue called sPQ. This will sort the printed output. First we will find the name of the station id. This will be the origin. We will retrieve the first 4 digits of the id. Then we will get the station name associated with the id. For example 0000 -> Abbesses. We will then get the Digit id (For example, 1 Destination). We then proceed with a while loop which will put a 0 in front of the endLocationId so we can parse the data correctly to access the name of the station stored in our ArrayList using MyEntry. Once this loop is done we will then fetch the edge distance, then we proceed to call MyEntry class to determine the end name. (Stored in 4 digits). From this we then insert the sPQ to sort, then we print unsorted and sortedly.

//Question 2.2 Modified from lab code to find the total time taken from Point N1 to N2
14) public void **printAllShortestDistances**(String vert, String vert2) throws Exception {}

We will also call the graph algorithm class and to use that to get our cloud of shortest distance. This method is taken from the lab and modified to print the shortest distance from N1 to N2, all we added is the start station ID, start station name, EndID , end station name and one if statement to selectively print one output. A special case if start point and end point is the same then we print Time taken is 0 seconds because same location.

15)public void **shortestPathLengthsStations**(Graph<String,Integer> g, Vertex<String> src, Vertex<String> des) {}

Modified from the lab code to get the shortestPath, here we changed the return type to void, we stop the cloud once we reached our destination. We also used the map called connectionMap and connectionEdges to trace/ track back our vertices and edges so we can print the shortest path in between.

// Used for Question 2.2 & 2.3
16) public void **printAllStations**(String vertSource,String vertDes) throws Exception{}
If question 2, calls **shortestPathLengthsStations** method and prints the shortest path from N1 to N2. If question 3 calls **shortestPathLengthsStations3** method and prints the shortest path from N1 to N2 provided N3 is broken line.

//question 2.3
17) public void **shortestPathLengthsStations3**(Graph<String,Integer> g, Vertex<String> src, Vertex<String> des) throws Exception {} similar to answer question 2.2 , we used this to answer question 2.3, all we added was an if statement to check if the station is present in our broken line

then we do not add to our cloud, means we skip the adding. For example if our code, if(hashSet.contains(v.getElement())){continue;}

// default if user enters no arguments
18)void **printAllShortestDistances**(String vert) throws Exception {}

Our definition of broken line for N3. If n3 is located in that line, means the whole line (same colour/path) connected to n3 will not function. For example 151 0 151 no path , or 0 151 151 no path

19) Our main method, public static void **main**(String[] argv) {}

If argv.length is <-1, shows the user how to input the data by printing a message. If argv.length=1, do Question 2.1 or default debugging by reading metro.txt . If argv.length=2, do Question 2.2, if argv.length=3, do Question 2.3. Else, shows error message shows the user how to input the data.

Example of Outputs:

Question 2.1 (Note that our output is unsorted and matches the expected output)

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 0

 Printing all same line, given station ID is 0

 ID: 0000 : Abbesses
 ID: 0276 : Porte de la Chapelle
 ID: 0368 : Volontaires
 ID: 0159 : Lamarck Caulaincourt
 ID: 0206 : Montparnasse Bienvenue
 ID: 0009 : Assemblée Nationale
 ID: 0081 : Corentin Celton
 ID: 0274 : Porte de Versailles
 ID: 0080 : Convention
 ID: 0147 : Jules Joffrin
 ID: 0191 : Marcadet Poissonniers
 ID: 0178 : Mairie d'Issy
 ID: 0238 : Pigalle
 ID: 0106 : Falguière
 ID: 0231 : Pasteur
 ID: 0217 : Notre Dame de Lorette
 ID: 0325 : Saint-Lazare
 ID: 0308 : Rue du Bac
 ID: 0360 : Vaugirard
 ID: 0294 : Rennes
 ID: 0347 : Sèvres Babylone
 ID: 0175 : Madeleine
 ID: 0218 : Notre-Dame-des-Champs
 ID: 0078 : Concorde
 ID: 0353 : Trinité d'Estienne d'Orves
 ID: 0322 : Saint-Georges
 ID: 0194 : Marx Dormoy
 ID: 0338 : Solférino
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 1

 Printing all same line, given station ID is 1

 ID: 0001 : Alexandre Dumas
 ID: 0239 : Pigalle
 ID: 0339 : Stalingrad
 ID: 0246 : Place de Clichy
 ID: 0085 : Courcelles
 ID: 0366 : Villiers
 ID: 0056 : Charles de Gaulle, Étoile
 ID: 0211 : Ménilmontant
 ID: 0235 : Philippe-Auguste
 ID: 0362 : Victor Hugo
 ID: 0204 : Monceau
 ID: 0027 : Blanche
 ID: 0256 : Porte Dauphine
 ID: 0013 : Barbès Rochechouart
 ID: 0151 : La Chapelle
 ID: 0005 : Anvers
 ID: 0021 : Belleville
 ID: 0302 : Rome
 ID: 0142 : Jaurès
 ID: 0075 : Colonel Fabien
 ID: 0012 : Avron
 ID: 0086 : Couronnes
 ID: 0351 : Ternes
 ID: 0284 : Père Lachaise
 ID: 0213 : Nation
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 31

 Printing all same line, given station ID is 31

 ID: 0031 : Bolivar
 ID: 0034 : Botzaris
 ID: 0144 : Jaurès
 ID: 0280 : Pré-Saint-Gervais
 ID: 0248 : Place des Fêtes
 ID: 0041 : Buttes Chaumont
 ID: 0170 : Louis Blanc
 ID: 0092 : Danube
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 116

 Printing all same line, given station ID is 116

 ID: 0116 : Gambetta
 ID: 0320 : Saint-Fargeau
 ID: 0233 : Pelleport
 ID: 0279 : Porte des Lilas
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 7

 Printing all same line, given station ID is 7

 ID: 0007 : Arts et Métiers
 ID: 0068 : Châtelet
 ID: 0283 : Pyrénées
 ID: 0136 : Hôtel de Ville
 ID: 0290 : Rambuteau
 ID: 0183 : Mairie des Lilas
 ID: 0278 : Porte des Lilas
 ID: 0129 : Goncourt
 ID: 0311 : République
 ID: 0146 : Jourdain
 ID: 0357 : Télégraphe
 ID: 0020 : Belleville
 ID: 0247 : Place des Fêtes
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 0022

 Printing all same line, given station ID is 22

 ID: 0022 : Bercy
 ID: 0281 : Pyramides
 ID: 0069 : Châtelet
 ID: 0120 : Gare de Lyon
 ID: 0024 : Bibliothèque François Mitterand
 ID: 0176 : Madeleine
 ID: 0084 : Cour Saint-Émilion
```

Question 2.2

```
------------ PRINTING THE SPECIFIED SHORTEST DISTANCE QUESTION 2.2  -----------------

 StartID: 0000 InitialName: Abbesses TO EndID: 0042 Destination: Buzenval
 TOTAL TIME TAKEN : 996 SECONDS

[0, 238, 239, 5, 13, 151, 339, 142, 75, 21, 86, 211, 284, 235, 1, 12, 213, 215, 42]

PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN

 Time Taken :   0 seconds ID: 0000 : Abbesses
 Time Taken :  41 seconds ID: 0238 : Pigalle
 Time Taken : 131 seconds ID: 0239 : Pigalle
 Time Taken : 185 seconds ID: 0005 : Anvers
 Time Taken : 252 seconds ID: 0013 : Barbès Rochechouart
 Time Taken : 309 seconds ID: 0151 : La Chapelle
 Time Taken : 402 seconds ID: 0339 : Stalingrad
 Time Taken : 429 seconds ID: 0142 : Jaurès
 Time Taken : 522 seconds ID: 0075 : Colonel Fabien
 Time Taken : 573 seconds ID: 0021 : Belleville
 Time Taken : 619 seconds ID: 0086 : Couronnes
 Time Taken : 656 seconds ID: 0211 : Ménilmontant
 Time Taken : 705 seconds ID: 0284 : Père Lachaise
 Time Taken : 749 seconds ID: 0235 : Philippe-Auguste
 Time Taken : 793 seconds ID: 0001 : Alexandre Dumas
 Time Taken : 829 seconds ID: 0012 : Avron
 Time Taken : 871 seconds ID: 0213 : Nation
 Time Taken : 961 seconds ID: 0215 : Nation
 Time Taken : 996 seconds ID: 0042 : Buzenval

 TOTAL TIME TAKEN : 996 seconds
```

```
------------ PRINTING THE SPECIFIED SHORTEST DISTANCE QUESTION 2.2  -----------------

 StartID: 0000 InitialName: Abbesses TO EndID: 0247 Destination: Place des Fêtes
 TOTAL TIME TAKEN : 766 SECONDS

[0, 238, 239, 5, 13, 151, 339, 142, 144, 31, 41, 34, 248, 247]

PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN

 Time Taken :   0 seconds ID: 0000 : Abbesses
 Time Taken :  41 seconds ID: 0238 : Pigalle
 Time Taken : 131 seconds ID: 0239 : Pigalle
 Time Taken : 185 seconds ID: 0005 : Anvers
 Time Taken : 252 seconds ID: 0013 : Barbès Rochechouart
 Time Taken : 309 seconds ID: 0151 : La Chapelle
 Time Taken : 402 seconds ID: 0339 : Stalingrad
 Time Taken : 429 seconds ID: 0142 : Jaurès
 Time Taken : 519 seconds ID: 0144 : Jaurès
 Time Taken : 555 seconds ID: 0031 : Bolivar
 Time Taken : 583 seconds ID: 0041 : Buttes Chaumont
 Time Taken : 640 seconds ID: 0034 : Botzaris
 Time Taken : 676 seconds ID: 0248 : Place des Fêtes
 Time Taken : 766 seconds ID: 0247 : Place des Fêtes

 TOTAL TIME TAKEN : 766 seconds
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 000 288

------------ PRINTING THE SPECIFIED SHORTEST DISTANCE QUESTION 2.2  ----------------

 StartID: 0000 InitialName: Abbesses TO EndID: 0288 Destination: Quai de la Rapée
 TOTAL TIME TAKEN : 957 SECONDS

[0, 159, 147, 191, 192, 64, 14, 124, 125, 122, 140, 313, 219, 297, 40, 17, 288]

PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN
 Time Taken :   0 seconds ID: 0000 : Abbesses
 Time Taken :  46 seconds ID: 0159 : Lamarck Caulaincourt
 Time Taken :  90 seconds ID: 0147 : Jules Joffrin
 Time Taken : 139 seconds ID: 0191 : Marcadet Poissonniers
 Time Taken : 229 seconds ID: 0192 : Marcadet Poissonniers
 Time Taken : 286 seconds ID: 0064 : Château Rouge
 Time Taken : 314 seconds ID: 0014 : Barbès Rochechouart
 Time Taken : 360 seconds ID: 0124 : Gare du Nord
 Time Taken : 450 seconds ID: 0125 : Gare du Nord
 Time Taken : 509 seconds ID: 0122 : Gare de l'Est
 Time Taken : 612 seconds ID: 0140 : Jacques Bonsergent
 Time Taken : 659 seconds ID: 0313 : République
 Time Taken : 740 seconds ID: 0219 : Oberkampf
 Time Taken : 790 seconds ID: 0297 : Richard Lenoir
 Time Taken : 827 seconds ID: 0040 : Bréguet-Sabin
 Time Taken : 868 seconds ID: 0017 : Bastille
 Time Taken : 957 seconds ID: 0288 : Quai de la Rapée

 TOTAL TIME TAKEN : 957 seconds
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 21 104


------------ PRINTING THE SPECIFIED SHORTEST DISTANCE QUESTION 2.2  ----------------

 StartID: 0021 InitialName: Belleville TO EndID: 0104 Destination: Exelmans
 TOTAL TIME TAKEN : 1271 SECONDS

[21, 20, 129, 311, 315, 344, 33, 304, 299, 60, 133, 317, 203, 332, 110, 2, 139, 355, 306, 157, 291, 141, 197, 199, 104]

PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN
 Time Taken :   0 seconds ID: 0021 : Belleville
 Time Taken :  90 seconds ID: 0020 : Belleville
 Time Taken : 138 seconds ID: 0129 : Goncourt
 Time Taken : 187 seconds ID: 0311 : République
 Time Taken : 277 seconds ID: 0315 : République
 Time Taken : 384 seconds ID: 0344 : Strasbourg Saint-Denis
 Time Taken : 436 seconds ID: 0033 : Bonne Nouvelle
 Time Taken : 477 seconds ID: 0304 : Rue Montmartre, Grands Boulevards
 Time Taken : 525 seconds ID: 0299 : Richelieu Drouot
 Time Taken : 588 seconds ID: 0060 : Chaussée d'Antin, La Fayette
 Time Taken : 633 seconds ID: 0133 : Havre Caumartin
 Time Taken : 682 seconds ID: 0317 : Saint-Augustin
 Time Taken : 735 seconds ID: 0203 : Miromesnil
 Time Taken : 775 seconds ID: 0332 : Saint-Philippe du Roule
 Time Taken : 812 seconds ID: 0110 : Franklin D. Roosevelt
 Time Taken : 881 seconds ID: 0002 : Alma Marceau
 Time Taken : 931 seconds ID: 0139 : Iéna
 Time Taken : 977 seconds ID: 0355 : Trocadéro
 Time Taken : 1010 seconds ID: 0306 : Rue de la Pompe
 Time Taken : 1046 seconds ID: 0157 : La Muette
 Time Taken : 1095 seconds ID: 0291 : Ranelagh
 Time Taken : 1132 seconds ID: 0141 : Jasmin
 Time Taken : 1181 seconds ID: 0197 : Michel Ange Auteuil
 Time Taken : 1209 seconds ID: 0199 : Michel Ange Molitor
 Time Taken : 1271 seconds ID: 0104 : Exelmans

 TOTAL TIME TAKEN : 1271 seconds
```

Question 2.3 (we also included some extra test cases)

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 35 200 19


--------PRINTING THE SHORTEST PATH FOR QUESTION 2.3-------


[35, 111, 76, 156, 371, 158, 138, 79, 77, 356, 227, 173, 67, 135, 331, 16, 18, 163, 105, 296, 205, 94, 200]

 Time taken : 1162
PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN

 Time Taken :   0 seconds ID: 0035 : Boucicaut
 Time Taken :  34 seconds ID: 0111 : Félix Faure
 Time Taken :  64 seconds ID: 0076 : Commerce
 Time Taken : 123 seconds ID: 0156 : La Motte Picquet, Grenelle
 Time Taken : 169 seconds ID: 0371 : École Militaire
 Time Taken : 235 seconds ID: 0158 : La Tour-Maubourg
 Time Taken : 292 seconds ID: 0138 : Invalides
 Time Taken : 363 seconds ID: 0079 : Concorde
 Time Taken : 453 seconds ID: 0077 : Concorde
 Time Taken : 510 seconds ID: 0356 : Tuileries
 Time Taken : 548 seconds ID: 0227 : Palais Royal, Musée du Louvre
 Time Taken : 582 seconds ID: 0173 : Louvre, Rivoli
 Time Taken : 633 seconds ID: 0067 : Châtelet
 Time Taken : 720 seconds ID: 0135 : Hôtel de Ville
 Time Taken : 785 seconds ID: 0331 : Saint-Paul, Le Marais
 Time Taken : 847 seconds ID: 0016 : Bastille
 Time Taken : 937 seconds ID: 0018 : Bastille
 Time Taken : 969 seconds ID: 0163 : Ledru Rollin
 Time Taken : 1016 seconds ID: 0105 : Faidherbe-Chaligny
 Time Taken : 1058 seconds ID: 0296 : Reuilly Diderot
 Time Taken : 1096 seconds ID: 0205 : Montgallet
 Time Taken : 1135 seconds ID: 0094 : Daumesnil
 Time Taken : 1162 seconds ID: 0200 : Michel Bizot

 TOTAL TIME TAKEN : 1162 seconds
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 35 241 11


--------PRINTING THE SHORTEST PATH FOR QUESTION 2.3-------


[35, 111, 76, 156, 155, 45, 348, 232, 209, 101, 293, 96, 324, 128, 83, 243, 244, 164, 49, 241]

 Time taken : 923
PRINTING ALL THE PATH ID'S & NAMES IN BETWEEN

 Time Taken :   0 seconds ID: 0035 : Boucicaut
 Time Taken :  34 seconds ID: 0111 : Félix Faure
 Time Taken :  64 seconds ID: 0076 : Commerce
 Time Taken : 123 seconds ID: 0156 : La Motte Picquet, Grenelle
 Time Taken : 213 seconds ID: 0155 : La Motte Picquet, Grenelle
 Time Taken : 248 seconds ID: 0045 : Cambronne
 Time Taken : 291 seconds ID: 0348 : Sèvres Lecourbe
 Time Taken : 337 seconds ID: 0232 : Pasteur
 Time Taken : 391 seconds ID: 0209 : Montparnasse Bienvenue
 Time Taken : 427 seconds ID: 0101 : Edgar Quinet
 Time Taken : 470 seconds ID: 0293 : Raspail
 Time Taken : 512 seconds ID: 0096 : Denfert Rochereau
 Time Taken : 564 seconds ID: 0324 : Saint-Jacques
 Time Taken : 606 seconds ID: 0128 : Glacière
 Time Taken : 655 seconds ID: 0083 : Corvisart
 Time Taken : 691 seconds ID: 0243 : Place d'Italie
 Time Taken : 781 seconds ID: 0244 : Place d'Italie
 Time Taken : 841 seconds ID: 0164 : Les Gobelins
 Time Taken : 883 seconds ID: 0049 : Censier Daubenton
 Time Taken : 923 seconds ID: 0241 : Place Monge

 TOTAL TIME TAKEN : 923 seconds
```

Test Special Cases for Question 2.3

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 154 154 154

 NOT POSSIBLE ORIGIN, DESTINATION, BROKEN LINE ENTERED IS THE SAME
 TIME TAKEN : 0 , BECAUSE ALL 3 POINTS IS THE SAME
```

All same input case

```
DarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 154 0 154

NO POSSIBLE PATH
BECAUSE LINE NOT FUNCTIONING IS IN YOUR ORIGIN
```

N1= N3 case

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 0 154 154

 NO POSSIBLE PATH
 BECAUSE LINE NOT FUNCTIONING IS IN YOUR DESTINATION
```

N2= N3 case

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 145 112 11

 NO POSSIBLE PATH
 BECAUSE LINE NOT FUNCTIONING IS IN YOUR ORIGIN
```

When N1 and N3 is in the same line, because 145 and 11 is in the same line, 145 is origin

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 145
Printing all same line, given station ID is 145
ID: 0145 : Javel
ID: 0196 : Michel Ange Auteuil
ID: 0148 : Jussieu
ID: 0373 : Église d'Auteuil
ID: 0154 : La Motte Picquet, Grenelle
ID: 0195 : Maubert Mutualité
ID: 0054 : Charles Michels
ID: 0036 : Boulogne, Jean Jaurès
ID: 0037 : Boulogne, Pont de Saint-Cloud, Rond Point Rhin et Danube
ID: 0099 : Duroc
ID: 0052 : Chardon Lagâche
ID: 0047 : Cardinal Lemoine
ID: 0221 : Odéon
ID: 0174 : Mabillon
ID: 0349 : Ségur
ID: 0346 : Sèvres Babylone
ID: 0259 : Porte d'Auteuil
ID: 0358 : Vaneau
ID: 0117 : Gare d'Austerlitz
ID: 0074 : Cluny, La Sorbonne
ID: 0198 : Michel Ange Molitor
ID: 0011 : Avenue Émile Zola
ID: 0201 : Mirabeau
```

```
iDarkDuck-MacBookPro:a4 iDarkDuck$ java ParisMetro 112 154 11

NO POSSIBLE PATH
BECAUSE LINE NOT FUNCTIONING IS IN YOUR DESTINATION
```

Programming and design

(comments, headers, clarity and code readability, conciseness of methods, did you design/use a good algorithm? did you define the right methods? did your algorithms solve the right problem? Is your solution unnecessarily complex? etc)

We reused the lab code from lab 10 and modified the methods in between, our algorithm was complex because it has many lines of code (lengthy) but it managed to solve the problem efficiently and easily when we use the lab code. We used the methods from the lab in order to retrieve our desired values as described in the comment headers and methods. In most cases, our methods are using O(n). We defined the right methods and solved the right problem, tested the exceptional cases and our answers matched the expected output. The methods we have defined in our case are complex algorithms in which one calls upon another and so on and so forth. The data which is being used within each algorithm is called within the method itself. Our algorithms are not directly solving the problem, instead we are using helper methods and methods which all upon each other to help ease the problem given at hand. This allows one to have much simpler code with more efficient running times.

Since we are using a multi-method approach to our solution we are able to have much simpler algorithms and cleaner code which is easier to read and/or edit if need be. Though this approach may not be considered the absolute correct solution. We are considering an approach where it has much more reusability and reliability.

Our solution is based off the Lab 10 (stated above). This solution was initially complex to read and write but as we create more and more helper variables and methods. This vastly simplifies the solution. The solution not necessarily solves the problem given in the question but is also a viable solution for many more applications. Such as train stations, plane's landing and taking off. Etc. Essentially, we are creating a basic design in which one is able to modify and create new helper methods to increase expandability with our software.

REFERENCES:

1)WeightGraph.java CSI2110 Lab 10 Code

```
// $Id: WeightGraph.java,v 1.1 2006/11/18 01:20:12 jlang Exp $
// CSI2110 Fall 2006 Laboratory 9: Adjacency List and DFS
//
================================================================================
// (C)opyright:
//
//   Jochen Lang
//   SITE, University of Ottawa
//   800 King Edward Ave.
//   Ottawa, On., K1N 6N5
//   Canada.
//   http://www.site.uottawa.ca
//
// Creator: jlang (Jochen Lang)
// Email:   jlang@site.uottawa.ca
//
================================================================================
// $Log: WeightGraph.java,v $
// Revision 1.1  2006/11/18 01:20:12  jlang
// Added lab10
//
// Revision 1.1  2006/11/11 03:15:52  jlang
// Added Lab9
// Modified by Thais Bardini on November 19th, 2017 (tbard069@uottawa.ca)
================================================================================
// Modified by CSI 2110 FALL 2017 ASSIGNMENT 4 GROUP 101 on DECEMBER 5th, 2017
(ngane103@uottawa.ca)
================================================================================
```

2) MyEntry Class

//https://stackoverflow.com/questions/3110547/java-how-to-create-new-entry-key-value

3) String Tokenizer Class

//https://stackoverflow.com/questions/10263298/check-to-see-if-next-string-tokenizer-is-empty