

ALGORITHM PROJECT

Professor: Patricia Conde Céspedes

Prepared by: Linfeng WANG 9858

Zheng WU 10362

Jiawei DAFIT 9570

5 juin 2017

TABLE OF CONTENTS

Project cover.....	1
Table of contents.....	2
1.Introduction.....	3
2.Data Collection and Graph construction	
2.1 Analyses the source data.....	4
2.2 Read data.....	4
2.3 Build graph.....	8
3.Calculation of shortest paths	
3.1 BFS.....	12
3.2 Dijkstra.....	12
3.2 Longest Paths.....	13
4.Shortest paths for Graph clustering.....	15
5.Code.....	18
6.Bibliography.....	18

1. INTRODUCTION

The main objective is to use the algorithmic and programming tools treated in the lectures and tutorial courses in a real graph in order to calculate the diameter (the longest of the shortest paths) of the Parisian subway graph.

The specific objectives of the project are:

1. Manipulate graph data (Parisian Metro Map), get the data collection and then build the graph.
2. Implement algorithms studied in the lectures and tutorial courses for the calculation of shortest paths in real data.

In this project, we will focus on graphs and the algorithms with which we get the result we want.

More precisely, we download the Parisian metro map, analysis and import them in our program, using a BFS and Dijkstra algorithm to calculate the shortest paths. Finally, Based on the shortest path, we get the graph clustering.

For the graph, in our case, we assume that each metro stop is a vertex, and the edge is the connection between the two metro stops. As for weight, we will calculate the distance between the two stops as their weight in the graph.

2. DATA COLLECTION AND GRAPH CONSTRUCTION

2.1 ANALYSES THE SOURCE DATA

First of all, download the dataset from the RATP site web: <https://data.ratp.fr/explore/>.

We choose the RATP_GTFS_LINES(all metro lines are separated, here is what we downloaded as input pic 01).

routes.txt: Each line have route for different direction.they are distinguished by route_id.

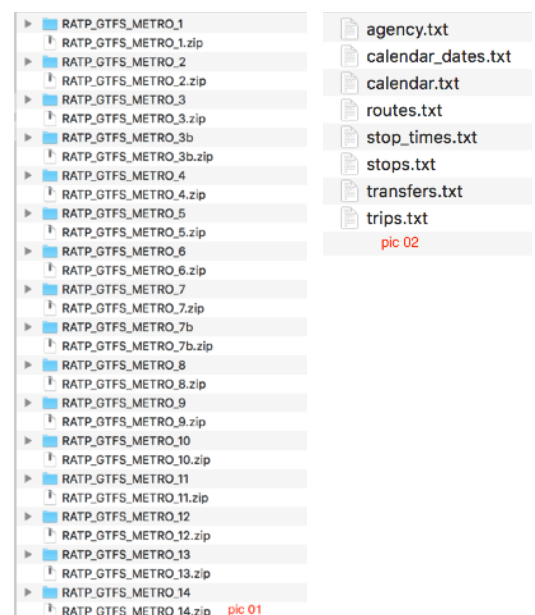
stops.txt:Each stops in one line has

stop_id, stop_name, stop_lat, stop_lon ; these are informations we care most.

stop_times.txt: in this file, we can get the sequence of the stops.

For each line, we have find that all the informations is separated by comma which

facilitate us to read data.



2.2 READ DATA

Import CSVUtils utility.

Import org.apache.commons.lang3, use this Utility methods for dealing with metro line files.The code detail is in CSVUtils.java, check the chapter code for detail.

In this class, we write function with different parameter for parsing the string input.

```
public static List<String> parseLine() {}
```

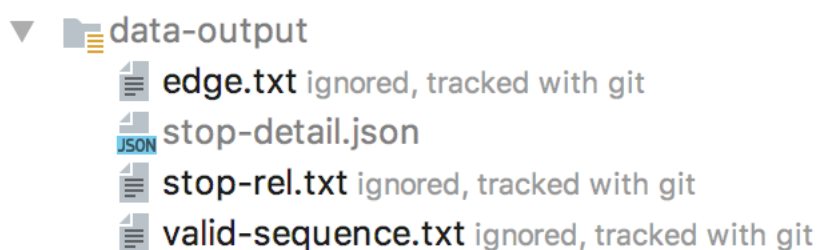
this function read all the input data, and store them into a List<String>.

We also create GTFSParser.java class. in this class, we read the raw data, retrieve and restore the stop detail. Meanwhile:

- map the original stop id with new id;
- map new id with stop name;
- build relationship between new id and stop name.

After reading all the metro lines, we generate the following files in order to avoid the duplication of the same stops save twice in the different line. At the same time, Store only the useful informations, station names, stations coordinates, create stations unique id etc. The output file named:

edge.txt; valid-sequence.txt; stop-rel.txt; stop-detail. json;



Take one output code for example:

```
// original id -> new id
HashMap<String, String> stopIdRel = new HashMap<>();
// new id <-> stop name
```

ALGORITHM

```
HashMap<Integer, Map> stopMap = new HashMap<>();
HashMap<String, String> stopBiMap = new HashMap<>();

// get stops
Integer stopIdIndex = 0;
HashSet<String> stopNames = new HashSet<>();
for (String metroLine : metroLines) {
    String fileName = "data-input/RATP_GTFS_" + metroLine + "/"
    stops.txt";
    FileInputStream stream = new FileInputStream(fileName);
    InputStreamReader streamReader = new
InputStreamReader(stream);
    BufferedReader reader = new BufferedReader(streamReader);
    reader.readLine(); // ignore title row
    Integer beforeIdIndex = stopIdIndex;
    String line;
    while ((line = reader.readLine()) != null) {
        String[] lineData = CSVUtils.parseLine(line).toArray(new
String[0]);
        String stopName =
StringUtils.stripAccents(lineData[2]).toUpperCase();
        String latitude = lineData[4];
        String longitude = lineData[5];
        if (!stopNames.contains(stopName)) {
            stopNames.add(stopName);
            HashMap<String, Object> stopDetail = new HashMap<>();
            stopDetail.put("name", stopName);
            stopDetail.put("latitude", latitude);
            stopDetail.put("longitude", longitude);
            stopMap.put(stopIdIndex, stopDetail);
            stopBiMap.put(stopName, stopIdIndex.toString());
```

ALGORITHM

```
        // build relationship between new id and stop name
        stopIdIndex++;
    }
    String stopId = lineData[0];
    stopIdRel.put(stopId, stopBiMap.get(stopName));
    // build relationship between old id and new id
    // different original id can connect to the same new id
}
reader.close();
streamReader.close();
stream.close();
System.out.println("parsed: " + fileName + ", " + (stopIdIndex
- beforeIdIndex) + " stops.");
}
System.out.println("all stops parsed, " + stopNames.size() + "
stops in total.");

FileWriter writer1 = new FileWriter("data-output/stop-rel.txt");
stopMap.forEach((s1, s2) -> {
    try {
        writer1.write(s1 + "," + s2.get("name") + "," +
s2.get("latitude") + "," + s2.get("longitude") + "\n");
    } catch (IOException e) {
        e.printStackTrace();
    }
});
writer1.close();
```

the output result are as follows:

Output 1: stop-rel.txt

new stop_id, stop_name, stop_latitude, stop_longitude.

ALGORITHM

```

1 0,"NATION",48.84811123157566,2.3980040127977436
2 1,"CONCORDE",48.86567809641779,2.321193771801867
3 2,"BASTILLE",48.85297567465423,2.3692188244433434
4 3,"LOUVRE-RIVOLI",48.86087985759375,2.340973275817903
5 4,"BERAULT",48.84536875015732,2.428244509620486
6 5,"CHAMPS-ELYSEES-CLEMENCEAU",48.867744026068706,2.3141227803741415
7 6,"REUILLY-DIDEROT",48.8473528769588,2.3858425400402448
8 7,"GEORGE V",48.87204561942636,2.3007691856449535
9 8,"ARGENTINE",48.87567248598795,2.289444164481448
10 9,"FRANKLIN-ROOSEVELT",48.869010427937624,2.3102531808960265
11 10,"PALAIS-ROYAL (MUSEE DU LOUVRE)",48.8623718215243,2.3365736007364304
12 11,"PONT DE NEUILLY",48.88550610941816,2.258527470197358
13 12,"SAINT-MANDE",48.84623825662077,2.418999828109256
14 13,"CHATEAU DE VINCENNES",48.84432514436857,2.440552346211822
15 14,"ESPLANADE DE LA DEFENSE",48.88835806016618,2.249937212862802
16 15,"CHATELET",48.85856967247972,2.347933245835307
17 16,"PORTE MAILLOT",48.87800617627899,2.2824656400013326
18 17,"CHARLES DE GAULLE-ETOILE",48.8739310942679,2.295127251650101
19 18,"TUILERIES",48.86478016250008,2.3290949544294275
20 19,"LES SABLONS (JARDIN D'ACCLIMATATION)",48.88129918549113,2.2719151760177914
21 20,"PORTE DE VINCENNES",48.847016518419125,2.410816882334764

```

Output 2: edge.txt

with title as follows: stop_id, stop_id(they are connected), distance(weight):

```

1 74 84 326.2216817462024
2 51 85 383.7573086056194
3 282 291 1085.4944638853387
4 177 178 291.0588168718802
5 35 39 531.6306734915337
6 52 59 683.1478536424372
7 151 15 894.6692711156427
8 297 280 632.9804447661647
9 41 40 641.2582736796169
10 9 5 316.1296080323238
11 230 125 735.7190669908838
12 202 1 496.2964535880419
13 64 35 529.9849596402272
14 50 248 580.0237811420653
15 125 122 345.65814214053495
16 160 98 581.0583803828546
17 125 243 294.45355514590045
18 24 14 954.7067021715384
19 209 220 929.5790408851523
20 239 93 483.21237639456524

```

Output 3: valid-sequence.txt:

with title as follows:

line number:[the sequence of stop in one direction where metro stops]


```
1 METRO_13: [275, 294, 284, 288, 276, 274, 281, 81, 238, 292, 283, 195, 5, 223, 60, 298, 29, 280, 278, 2
2 METRO_13: [296, 295, 279, 285, 293, 278, 280, 29, 298, 60, 223, 5, 195, 283, 292, 238, 81, 281, 274, 2
3 METRO_13: [282, 291, 290, 277, 286, 289, 287, 297, 280, 29, 298, 60, 223, 5, 195, 283, 292, 238, 81, 2
4 METRO_13: [275, 294, 284, 288, 276, 274, 281, 81, 238, 292, 283, 195, 5, 223, 60, 298, 29, 280, 297, 2
5 METRO_14: [60, 202, 157, 15, 21, 117, 299, 300, 301]
6 METRO_14: [301, 300, 299, 117, 21, 15, 157, 202, 60]
7 METRO_2: [0, 41, 40, 28, 30, 43, 46, 33, 34, 42, 37, 27, 32, 38, 26, 36, 29, 47, 35, 39, 45, 44, 17, 2
8 METRO_2: [31, 25, 17, 44, 45, 39, 35, 47, 29, 36, 26, 38, 32, 27, 37, 42, 34, 33, 46, 43, 30, 28, 40,
9 METRO_3: [61, 53, 62, 30, 66, 63, 50, 68, 57, 51, 49, 65, 55, 67, 70, 60, 64, 35, 69, 58, 48, 54, 56,
10 METRO_3: [59, 52, 56, 54, 48, 58, 69, 35, 64, 60, 70, 67, 55, 65, 51, 57, 68, 50, 63, 66, 30, 62,
11 METRO_4: [95, 74, 84, 86, 32, 91, 83, 88, 80, 51, 85, 92, 15, 76, 87, 93, 82, 78, 79, 81, 75, 96, 94,
12 METRO_4: [97, 77, 90, 89, 94, 96, 75, 81, 79, 78, 82, 93, 87, 76, 15, 92, 85, 51, 80, 88, 83, 91, 32,
13 METRO_5: [98, 107, 102, 111, 112, 2, 106, 108, 103, 50, 113, 83, 91, 37, 42, 104, 110, 100, 109, 100,
14 METRO_5: [99, 105, 101, 109, 100, 110, 104, 42, 37, 91, 83, 113, 50, 103, 108, 106, 2, 112, 111, 102,
15 METRO_5: [111, 144, 241, 231, 239, 93, 234, 232, 233, 238, 230, 125, 243, 244, 236, 235, 221, 241, 23
16 METRO_10: [241, 218, 246, 240, 236, 244, 243, 125, 230, 238, 233, 232, 234, 93, 239, 231, 242, 144, 11
17 METRO_10: [245, 237, 218, 246, 240, 236, 244, 243, 125, 230, 238, 233, 232, 234, 93, 239, 231, 242, 14
18 METRO_11: [15, 23, 250, 57, 50, 248, 33, 251, 247, 173, 249, 73, 252]
19 METRO_11: [252, 73, 249, 173, 247, 251, 33, 248, 50, 57, 250, 23, 15]
20 METRO_11: [24, 14, 11, 19, 16, 8, 17, 7, 9, 5, 1, 18, 10, 3, 15, 23, 22, 2, 21, 6, 0, 20, 12, 4, 13]
```

Output 4: stop-detail.txt

This result is reserved for the Noe4j to draw a virtualized map to present the cluster between the metro station.

```

1  ["0":{"latitude":48.84811123157566,"name":"NATION","longitude":2.3986048127797436},"1":{"latitude":48.8565789964179,"name":"CONCORDE","longitude":2.32119371801867},"2":{"latitude":48.85297567465423,"name":"BASTILLE","longitude":2.369218644433434},"3":{"latitude":48.860879859375,"name":"LOUVRE-RIVOLI","longitude":2.340973275817903},"4":{"latitude":48.8453681505732,"name":"BERAULT","longitude":2.42824599620486},"5":{"latitude":48.86774402669786,"name":"CHAMPS-ELYSEES-CLERMENAU","longitude":2.3341227883741415},"6":{"latitude":48.8543528769588,"name":"NEUILLY-DIDEROT","longitude":2.385842648042448},"7":{"latitude":48.8724631242636,"name":"GEORGE V","longitude":2.300769185644935},"8":{"latitude":48.87567248590795,"name":"FRANKLIN","longitude":2.28944164481448},"9":{"latitude":48.869018427395624,"name":"ARMINTE-ROOSEVELT","longitude":2.310231808960265},"10":{"latitude":48.85237182152543,"name":"PALAIS-ROYAL (POUE DU LOUVRE)","longitude":2.33657367634304},"11":{"latitude":48.8558610941618,"name":"PORT DE NEUILLY","longitude":2.25852747087358},"12":{"latitude":48.8432565628777,"name":"CHATEAU DE VINCENNES","longitude":2.44899992111822},"13":{"latitude":48.8443251436857,"name":"CHATEAU DE VINCENNES","longitude":2.44855231211822},"14":{"latitude":48.8585980601618,"name":"ESPLANADE DE LA DEFENSE","longitude":2.249937212862802},"15":{"latitude":48.8585966274972,"name":"CHATELET","longitude":2.347933245835307},"16":{"latitude":48.87086616727899,"name":"PORT MAILLOT","longitude":2.282465640001326},"17":{"latitude":48.8739119946279,"name":"CHARLES DE GAULLE-ETOILE","longitude":2.295127251050101},"18":{"latitude":48.86478016250008,"name":"TUILERIES","longitude":2.320894954608195},"19":{"latitude":48.8812091856131,"name":"LES SABLONS (JARDIN DU)02ACCLIMATATION","longitude":2.27191516771914},"20":{"latitude":48.84701651819125,"name":"PORTE DE VINCENNES","longitude":2.41061682337464},"21":{"latitude":48.8459597798341,"name":"GARE DE LYON","longitude":2.37344924965318},"22":{"latitude":48.

```

2.3 BUILD GRAPH

Before we build graph, we create a class Edge.java. the constructor is

```
public class Edge implements Comparable<Edge> {
    private final int v;
    private final int w;
    private final double weight;
    private int betweenness;

    Edge(int either, int other, double power) {
```

ALGORITHM

```

        v = either;
        w = other;
        weight = power;
        betweenness = 0;
    }

    public int either() {
        return v;
    }

    public int other(int vertex) {
        if (vertex == v) return w;
        else if (vertex == w) return v;
        throw new IllegalArgumentException("Illegal endpoint");
    }

```

Then we create a `EdgedWeightedGraph.java` class to represent `WeightedGraph`. we can set `weight=1`, if we need unweighted Graph.

Consider the graph built in the previous section, we add weights to the edges in order to convert it into a weighted graph. This will make this study more realistic.

Each weight will represent the distance between two subway stops. To calculate the distances, you will consider the geographical position of each stop. You will find, for each subway line, a file named “stop.txt” which contains the position of each stop. The columns “stop_lat” and “stop_long” describe the position coordinates.

The formula used to calculate the distances will be the Euclidean distance between two points. Therefore, you will do abstraction of the shape of the paths by considering them as straight lines.

ALGORITHM

For the weighted graph, we calculate the distance basing the two station latitude and longitude.

```
public static double distance(double lat1, double lat2, double
lon1, double lon2, double el1, double el2) {
    final double R = 6371.0; // Radius of the earth
    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) *
Math.sin(latDistance / 2)
        + Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2))
        * Math.sin(lonDistance / 2) *
Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 -
a));
    double distance = R * c * 1000; // convert to meters
    double height = el1 - el2;
    distance = Math.pow(distance, 2) + Math.pow(height, 2);
    return Math.sqrt(distance);
}
```

3. CALCULATION OF SHORTEST PATHS

In this project, for unweighted and weighted graph, we will use BFS and Dijkstra respectively to find the diameter of the subway.

Once you calculated the diameter, give the path (containing subway stations names) as well as the total length of the path and of each sub-path

ALGORITHM

3.1 BFS algorithm

Breadth-first-search(BFS) is an algorithm for finding shortest paths from a single source vertex to all other vertices.

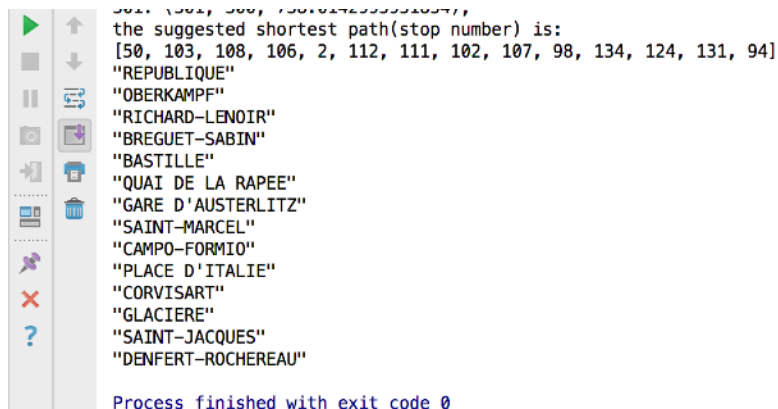
BFS processes vertices in increasing order of their distance(number of vertex) from the root vertex.

The main step are as follows:

1. First all direct neighbors
2. Then neighbors of neighbors
3. Repeat the above steps until iterate all the vertex.

For unweighted graph, we use BFS algorithm to calculate the shortest path. For example,

From « REPUBLIQUE» station to « DENFERT-ROCHEREAU»station, the shortest path is :



```

the suggested shortest path(stop number) is:
[50, 103, 108, 106, 2, 112, 111, 102, 107, 98, 134, 124, 131, 94]
"REPUBLIQUE"
"OBERKAMPF"
"RICHARD-LENOIR"
"BREGUET-SABIN"
"BASTILLE"
"QUAI DE LA RAPEE"
"GARE D'AUSTERLITZ"
"SAINT-MARCEL"
"CAMPO-FORMIO"
"PLACE D'ITALIE"
"CORVISART"
"GLACIERE"
"SAINT-JACQUES"
"DENFERT-ROCHEREAU"
Process finished with exit code 0

```

Conclusion :the total number of metro stations is:14. BFS algorithm only consider the number of stations when it do the search.

3.2 Dijkstra algorithm

On the contrary ,Dijkstra algorithm is greedy. it always considers the shortest distance to « swallow »

The main steps are as follows:

1. Find the vertex at the shortest from the source vertex distance that has not been « marked » yet.
2. Mark it and update the distance of its neighbors.
3. Repeat the first two steps until all vertices are marked.

For the weighted graph, in our case, we will consider the distance between the two stations, thus we use Dijkstra to calculate the shortest path, we still take the same example, From « REPUBLIQUE » station to « DENFERT-ROCHEREAU » station, the shortest path is:

```

the suggested shortest path(distance) is:
[50, 57, 250, 23, 15, 76, 87, 93, 82, 78, 79, 81, 75, 96, 94]
"REPUBLIQUE"
"ARTS-ET-METIERS"
"RAMBUTEAU"
"HOTEL DE VILLE"
"CHATELET"
"CITE"
"SAINT-MICHEL"
"ODEON"
"SAINT-GERMAIN DES PRES"
"SAINT-SULPICE"
"SAINT-PLACIDE"
"MONTPARNASSE-BIENVENUE"
"VAVIN"
"RASPAIL"
"DENFERT-ROCHEREAU"
Process finished with exit code 0

```

Conclusion: the total number of metro stations is:15. BFS algorithm consider the distance first when it do the search.

3.3 Longest Paths

For BFS and Dijkstra, the purpose is to find the diameter of the subway. Once you calculated the diameter, give the path (containing subway stations names) as well as the total length of the path and of each sub-path.

the result are as follows:

the total length of 14 metro lines is length of sub-paths=25603.958102975375 m.

longest path is:

182=>203=>198=>197=>199=>192=>194=>193=>191=>190=>177=>178=>133=

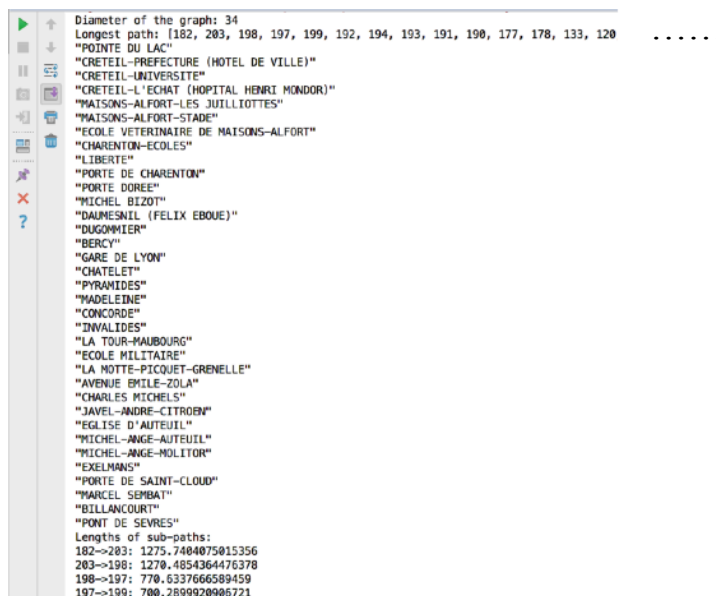
>120=>117=>21=>15=>157=>202=>1=>195=>180=>181=>125=>243=>244=>23
6=>235=> 221=> 218=> 212=> 219=>227=>204=>210

and the corresponding subpath and length is:

POINT DU LAC=>CRETEIL-PREFECTURE : 1275.740 m

CRETEIL-PREFECTURE =>CRETEIL-UNIVERCITE: 1270.485m

CRETEIL-UNIVERCITE=>CRETEIL-L'ACHAT: 770.633m



4. SHORTEST PATHS FOR GRAPH CLUSTERING

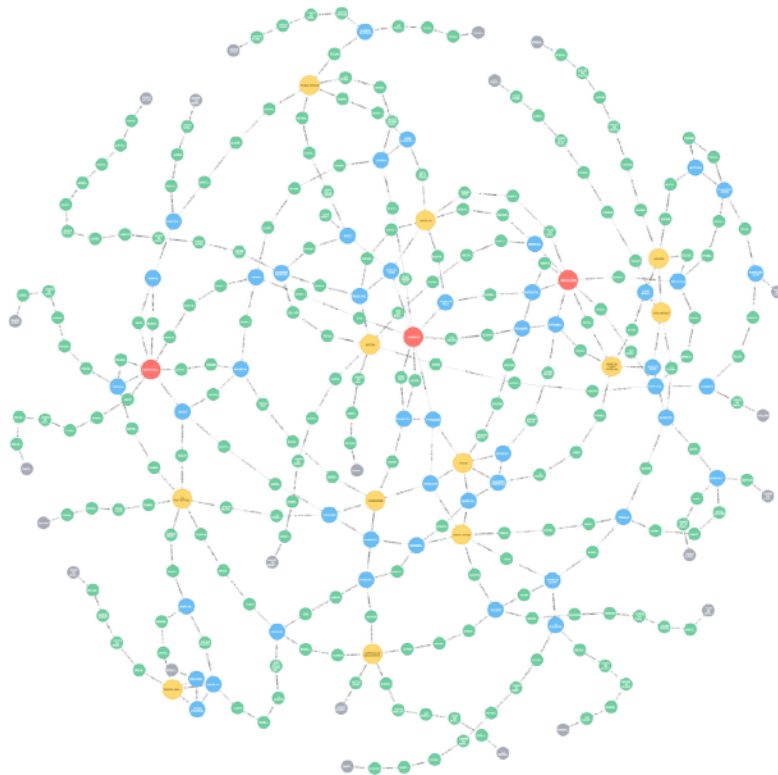
For illustrating the metro station's connectivity, we use the neo4j to present the map before.

In this graph, we calculate degree of each metro station, and set color separately.

For example:

Station Degree	Color	Size
7	Red	Biggest
5	Bleu	Middle
1	Grey	Smallest

In addition, the connection between the metro station is the distance(meter) between them.



Map Paris



More detail

The main steps for executing this program is like below:

1. For all pair of nodes, we perform BFS algorithm from all source nodes.
2. Calculate the betweenness on all edges
3. Choose the highest one, and then remove them.
4. Repeat the above steps until getting two disconnected subgraph.

The result of graph clustering is as below:

```

Remove edge: "MADELEINE"->"CONCORDE", betweenness = 9824
Remove edge: "CHATELET"->"GARE DE LYON", betweenness = 9857
Remove edge: "CHATELET"->"HOTEL DE VILLE", betweenness = 10614
Remove edge: "MIROMESNIL"->"SAINT-LAZARE", betweenness = 7939
Remove edge: "GARE D'AUSTERLITZ"->"QUAI DE LA RAPEE", betweenness = 6925
Remove edge: "SAINT-AUGUSTIN"->"HAVRE-CAUMARTIN", betweenness = 7374
Remove edge: "TUILERIES"->"PALAIS-ROYAL (MUSEE DU LOUVRE)", betweenness = 8176
Remove edge: "CHATELET"->"CITE", betweenness = 7718
Remove edge: "VILLIERS"->"MONCEAU", betweenness = 8962
Remove edge: "JUSSIEU"->"SULLY-MORLAND", betweenness = 13803
Remove edge: "QUAI DE LA GARE"->"BERCY", betweenness = 22317
Connected components:

[0, 20, 41, 6, 228, 207, 114, 12, 40, 176, 21, 175, 214, 226, 116, 4, 28, 133, 117, 2, 186, 213, 222, 13, 30, 120, 178, 299, 22, 185, 112, 106, 224, 209, 62, 66, 43, 177, 300, 23, 183, 108, 103, 220, 53, 72, 63, 46, 190, 301, 250, 184, 50, 216, 61, 71, 33, 191, 57, 248, 113, 68, 80, 73, 34, 251, 193, 51, 83, 88, 187, 249, 252, 42, 247, 194, 85, 49, 91, 140, 142, 188, 173, 104, 168, 37, 146, 192, 92, 65, 32, 145, 189, 171, 170, 110, 172, 148, 27, 199, 15, 55, 38, 86, 162, 67, 158, 169, 100, 166, 197, 151, 157, 149, 3, 26, 84, 202, 70, 109, 167, 198, 163, 10, 261, 262, 36, 74, 269, 271, 60, 101, 161, 203, 263, 256, 29, 95, 268, 298, 266, 64, 105, 152, 182, 47, 280, 264, 35, 99, 155, 297, 278, 69, 139, 287, 293, 58, 289, 285, 48, 286, 279, 54, 277, 295, 56, 290, 296, 52, 291, 59, 282]

[1, 195, 18, 5, 258, 283, 180, 9, 223, 253, 292, 181, 205, 7, 208, 225, 267, 238, 125, 17, 217, 232, 81, 230, 233, 122, 243, 121, 179, 25, 8, 44, 118, 130, 234, 270, 75, 126, 281, 257, 259, 79, 128, 115, 244, 119, 174, 31, 16, 45, 135, 127, 229, 93, 96, 274, 78, 260, 236, 200, 19, 39, 211, 239, 87, 82, 94, 276, 255, 235, 240, 201, 11, 206, 231, 76, 89, 131, 288, 273, 221, 246, 196, 14, 215, 242, 90, 124, 284, 265, 241, 218, 24, 144, 77, 134, 294, 272, 237, 212, 164, 111, 97, 98, 275, 254, 245, 219, 147, 102, 160, 159, 107, 123, 227, 156, 129, 204, 143, 150, 132, 210, 154, 165, 136, 141, 137, 138, 153]

Process finished with exit code 0

```

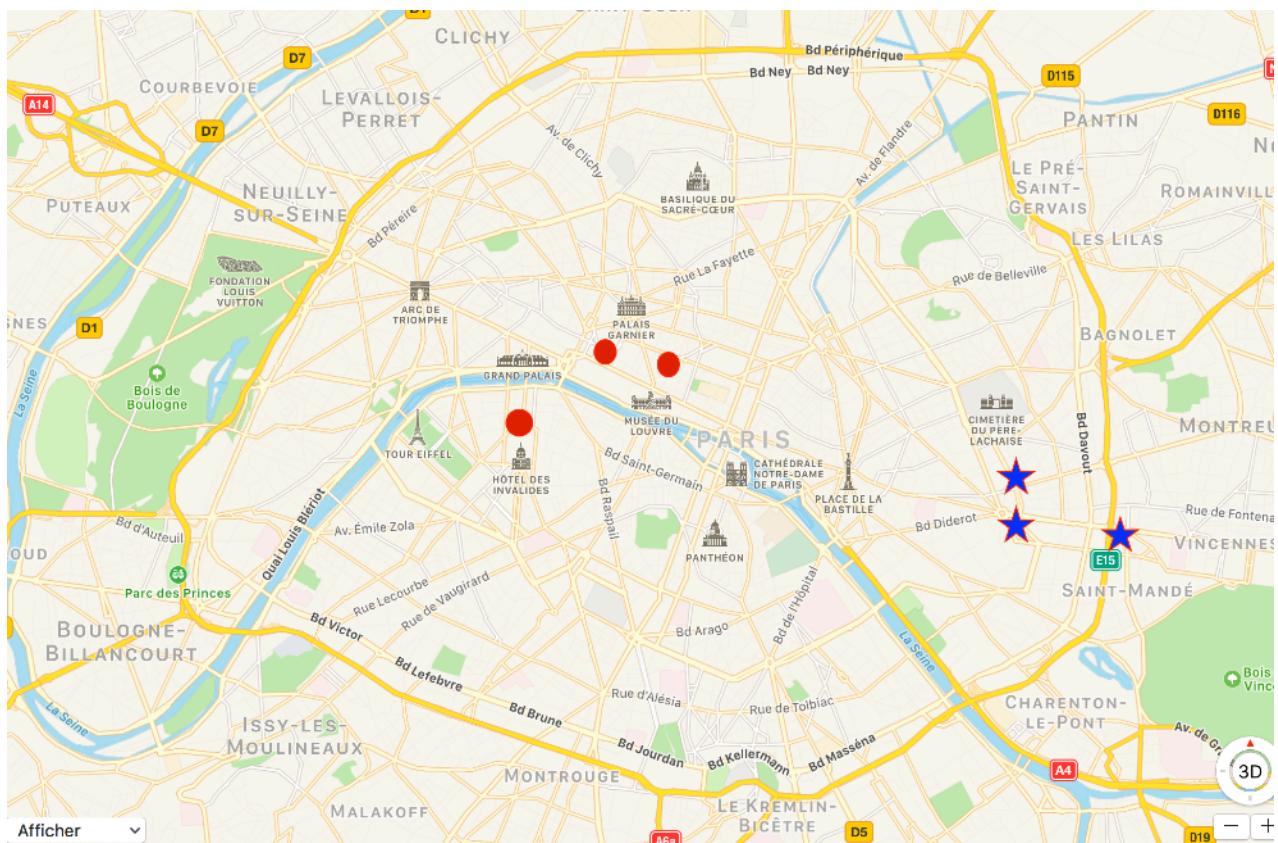
Conclusion: after removing the most high betweenness edge, we get the two connected components.

We predict that the two component divide Paris equally. Lets take some stations to test.

For the first connected component: 0-> NATION; 20->PORTE DE VINCENNES; 41->AVRON.....

For the second connected component: 1->CONCORDE; 195->INVALIDES; 18->TUILERIES.....

Thus, we guess the separate line lie into the direction north east to south west.



5. CODE

Please check the attachement for the code.

6. BIBLIOGRAPHY

<http://algs4.cs.princeton.edu/home/>

https://www.csc2.ncsu.edu/faculty/nfsamato/practical-graph-mining-with-R/slides/pdf/Graph_Cluster_Analysis.pdf

<https://neo4j.com/>