

Sami About

[Adresse de messagerie]

Résumé

Ce document présente les résultats d’une étude algorithmique du réseau de métro parisien.

Projet d’Algorithmique et de programmation avancÉe

Étude de graphe du métro Parisien

Table des matières

[1. Structure du projet 2](#_Toc484515890)

[2. Calcul du plus court chemin 4](#_Toc484515891)

[a. Plus court 4](#_Toc484515892)

[b. Calcul du diamètre 6](#_Toc484515893)

[3. Identification des Clusters 8](#_Toc484515894)

[Annexe 10](#_Toc484515895)

[a. Sortie console : adjacency list 10](#_Toc484515896)

[b. Sortie console : chemin entre deux points 10](#_Toc484515897)

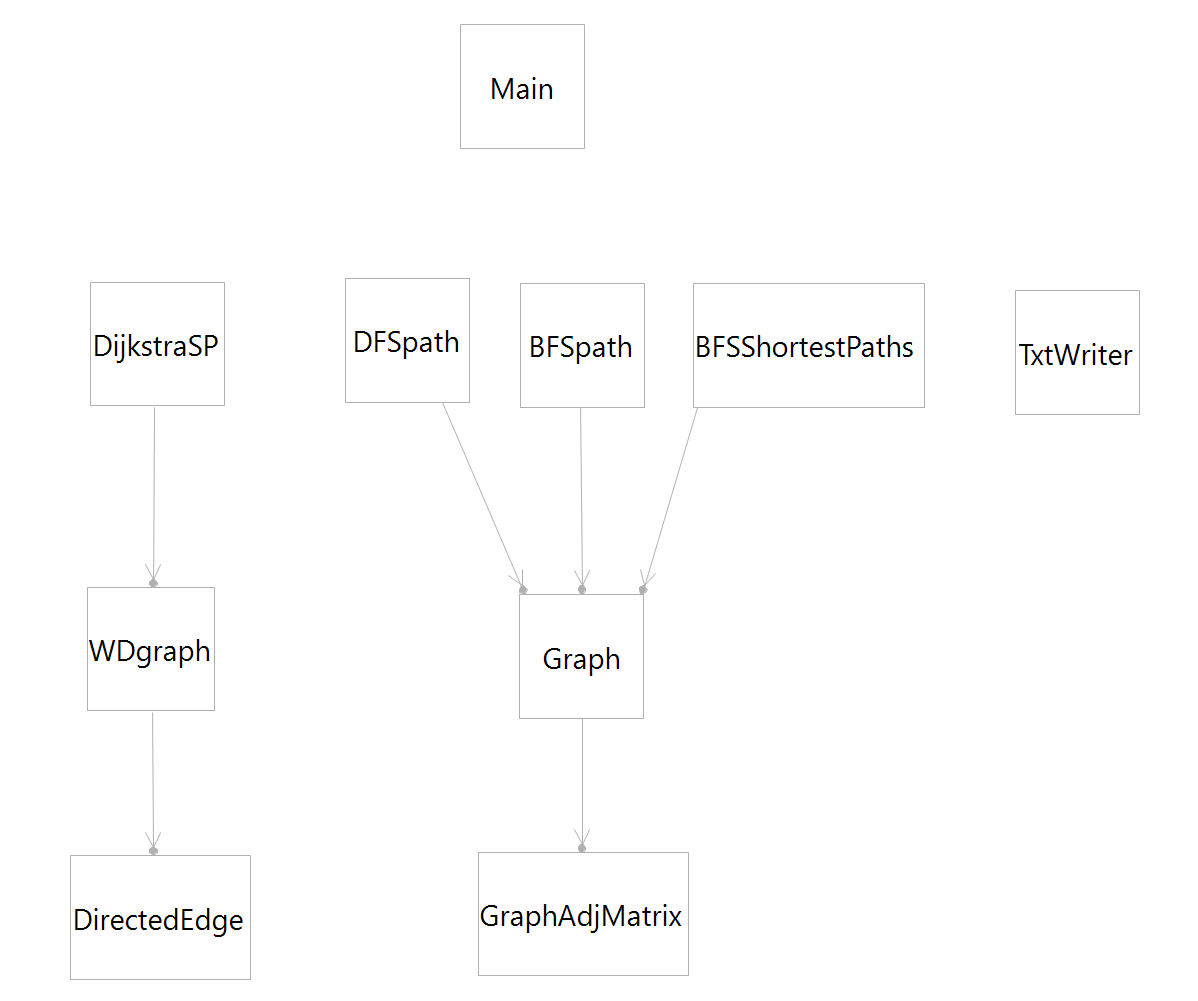
[c. Sortie console : calcul du diamètre 11](#_Toc484515898)

Table des figures

[Figure 1 : Chemin le plus court 3](#_Toc484451078)

# Structure du projet

Voici un diagramme de classe du projet :



L’objet **Graph** permet lorsqu’il est instancié, de récupérer le réseau du métro des fichiers JSON, grâce à la librairie Jackson.

Deux fichiers json sont utilisés pour modéliser le réseau :

* Stations.json : elle contient une liste de l’ensemble des stations ainsi que leurs coordonnées.
* Ligne.json : elle contient des listes de stations, recréant les ligne du métro Parisien.

Ces fichiers ne contiennent que le plan du métro, les stations de tram, RER, funiculaire et Orly ne sont pas incluses.

La classe **ReadJSON** est utilisée dans **Graph**, elle utilise Jackson afin de lire les fichier json. Les stations contenues dans Stations sont instanciées en objets **Node**, les lignes contenues dans lignes permettent de créer l’adjacency list, dans **Graph**.

l’adjacency list (telle quelle est affichée dans la console) est représenté dans le fichier todo.

La classe **ShortestPath** implémente les algorithmes de recherche de plus court chemin. Deux classes héritent de **ShortestPath**, qui utilisent chacune un algorithme différent :

* **BFSShortestPaths** : qui implémente l’algorithme de parcours en largeur (Breadth First Search), il se base sur des graphs non pondérés.
* **DijkstraSP** : qui implémente l’algorithme de Dijkstra des graphs pondérés. Chaque node contient la latitude et la longitude de la station, on obtient la pondération d’un edge en calculant la distance entre les deux stations qu’elle lie.

# Calcul du plus court chemin

## Plus court chemin entre deux points

On peut demander au programme de calculer le chemin le plus court entre deux stations avec les deux méthodes différentes. Le BFS ne prenant pas en compte la position des stations, les résultats peuvent différer.

Ainsi, si on calcule le chemin en Bercy et Hôtel de Ville, on devrait obtenir deux chemins différents.

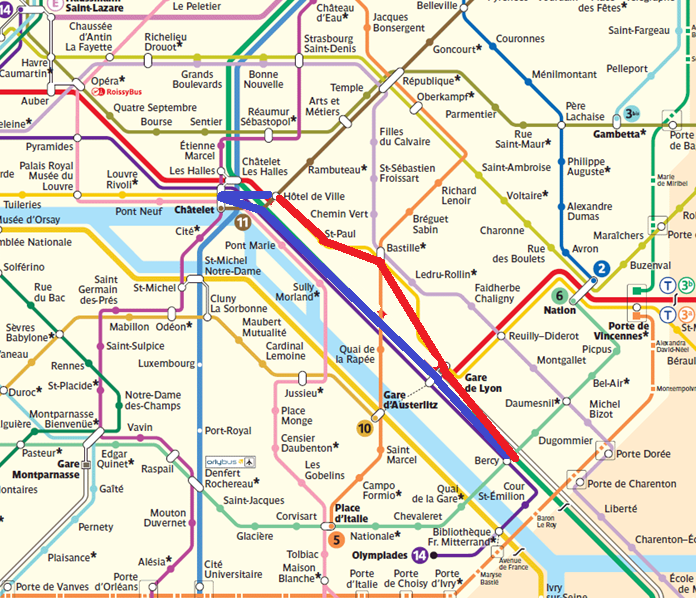


Figure 1 : Chemin le plus court

Le chemin bleu représente de plus court chemin dans le cas d’un graph non pondéré, le chemin rouge représente le plus court chemin, en prenant en compte les distances.

L’algorithme de parcours en largeur donne le résultat suivant :

Bercy

Gare de Lyon

Châtelet

Hôtel de Ville

L’algorithme pondéré Dijsktra donne le résultat suivant :

Bercy

Gare de Lyon distance from previous : 0.7756004347114352

Bastille distance from previous : 1.0545607850857195

Saint-Paul (Le Marais) distance from previous : 0.9813861081176881

Hôtel de Ville distance from previous : 1.0335574164992687

L’algorithme BFS calcule un chemin en 3 stations tant dis que le chemin calculé par l’algorithme de Dijsktra passe par 4 stations mais est plus court en termes de distance brute. On obtient donc bien des résultats cohérents.

La sortie console est disponible dans le fichier todo.

## Calcul du diamètre

On applique à chaque node le BFS et l’algorithme de Dijsktra afin d’obtenir son excentricité. On réitère le précessus sur chaque node, on obtient ainsi le diamètre en choisissant la plus grande excentricitée calculée et le radius en choisissant la plus petites.

On obtient ainsi pour le BFS :

diameter : 34.0

radius : 17.0

Pour Dijsktra

diameter : 32.98664272478596

radius : 16.940912151396866

Dans les deux cas, le diamètre obtenu correspond au chemin entre —Pont de Sèvres— et —Pointe du Lac—.

Le chemin obtenu avec l’algorithme BFS est :

Pont de Sèvres

Billancourt

Marcel Sembat

Porte de Saint-Cloud

Exelmans

Michel-Ange-Molitor

Michel-Ange-Auteuil

Eglise d'Auteuil

Javel-André-Citroen

Charles Michels

Avenue Emile-Zola

La Motte-Picquet-Grenelle

Ecole Militaire

La Tour-Maubourg

Invalides

Concorde

Madeleine

Pyramides

Châtelet

Gare de Lyon

Reuilly-Diderot

Montgallet

Daumesnil (Félix Eboué)

Michel Bizot

Porte Dorée

Porte de Charenton

Liberté

Charenton-Ecoles

Ecole Vétérinaire de Maisons-Alfort

Maisons-Alfort-Stade

Maisons-Alfort-Les Juilliottes

Créteil-L'Echat (Hôpital Henri Mondor)

Créteil-Université

Créteil-Préfecture (Hôtel de Ville)

Pointe du Lac

Le chemin obtenu avec l’algorithme Dijsktra est (calculé avec des distances en km) :

Pont de Sèvres

Billancourt distance from previous : 0.9111677938832308

Marcel Sembat distance from previous : 0.6986808412123223

Porte de Saint-Cloud distance from previous : 1.4808256761481424

Exelmans distance from previous : 0.6005247326488226

Michel-Ange-Molitor distance from previous : 0.3734383197390458

Chardon-Lagache distance from previous : 0.4995918842498141

Mirabeau distance from previous : 0.816454032462486

Javel-André-Citroen distance from previous : 0.5775718685761083

Charles Michels distance from previous : 0.8756496349179784

Avenue Emile-Zola distance from previous : 0.9587303750063835

La Motte-Picquet-Grenelle distance from previous : 0.4141164302304931

Ségur distance from previous : 1.1289549960543264

Duroc distance from previous : 0.9891658648189255

Montparnasse-Bienvenue distance from previous : 0.7773549065889046

Edgar-Quinet distance from previous : 0.4949578291707161

Raspail distance from previous : 0.5132899469430959

Denfert-Rochereau distance from previous : 0.6143691181844467

Saint-Jacques distance from previous : 0.4822071162235533

Glacière distance from previous : 0.8829248148050528

Corvisart distance from previous : 0.641341672488808

Place d'Italie distance from previous : 0.7260308626574583

Nationale distance from previous : 0.7174655086398712

Chevaleret distance from previous : 0.5993162290801676

Quai de la Gare distance from previous : 0.8070112821753982

Bercy distance from previous : 0.7269506227360764

Dugommier distance from previous : 1.0546035873007102

Daumesnil (Félix Eboué) distance from previous : 0.767561406673604

Michel Bizot distance from previous : 0.8448384898771499

Porte Dorée distance from previous : 0.36677566784219867

Porte de Charenton distance from previous : 0.7729471937429111

Liberté distance from previous : 1.09386583711972

Charenton-Ecoles distance from previous : 0.937056007075169

Ecole Vétérinaire de Maison distance from previous : 1.0137233368356438

Maisons-Alfort-Stade distance from previous : 1.8092858857902578

Maisons-Alfort-Les Juilliott distance from previous : 1.1985562460150139

Créteil-L'Echat (Hôpital Hen distance from previous : 0.8691365316905068

Créteil-Université distance from previous : 0.8334589206798152

Créteil-Préfecture (Hôtel de distance from previous : 1.3673698994894

Pointe du Lac distance from previous : 1.5148579736659233

La sortie console est disponible dans le fichier todo.

# Identification des Clusters

Afin d’identifier des Clusters, l’algorithme va calculer pour chaque egde sa « betweenness » et retirer à chaque itération celui ayant la plus grande betweenness », jusqu’à ce que de graph ne comporte que des nodes isolés.

Afin de calculer la « betweenness » des nodes, l’algorithme calcule l’ensemble des chemins les plus courts existants au sein du graph et incrément la « betweenness » d’un edge à chaque passage. Pour calculer tous ces chemins, on calcule pour chacune des 302 stations le chemin emprunté lors de l’exploration du graphe par les algorithmes. Ce chemin nous permet de calculer aisément l’ensemble des chemins entre la station en question et les autres stations et ainsi compléter la « betweenness » des edges.

On peut alors demander au programme d’afficher l’ensemble des betweenness calculées ainsi qu’un graph de répartition en fonction de la taille des betweenness.

La sortie console est disponible dans le fichier todo.

On obtient ainsi pour les edges ayant la plus grande betweenness :

* Pour l’algorithme de BFS (betweenness > 18 000) :

22882 : Madeleine <-> Concorde

19953 : Gare de Lyon <-> Châtelet

18191 : Madeleine <-> Pyramides

18164 : Châtelet <-> Pyramides

* Pour l’algorithme de Dijsktra (betweenness > 9 000) :

11680 : Madeleine <-> Concorde

10120 : Châtelet <-> Cité

9891 : Saint-Michel <-> Cité

9686 : Saint-Michel <-> Odéon

L’algorithme nous donne alors l’edge à retirer :

* Pour l’alogorithme de BFS :

edge to remove :  Madeleine <-> Concorde

* Pour l’algorithme de Dijsktra :

edge to remove :  Madeleine <-> Concorde

L’algorithme retire un edge à chaque itération, il n’est pas nécessaire d’afficher l’ensemble des betweenness de chaque edge.

Il affiche donc le numéro de l’itération de l’algortihme, la taille de subgraph qui a le plus grand diamètre ainsi que son diamètre et l’edge qui contient la plus grande betweenness et qui est retiré du graph à la fin de l’itération.

# Annexe

Désélectionnez le retour charriot ( ) dans voter éditeur de texte, pour mieux visualiser le texte.

## Sortie console : adjacency list

La sortie affiche chaque ligne du tableau de l’adjacency list, par ex pour la première ligne  :

0 : Simplon (2) : Porte de Clignancourt | Marcadet-Poissonniers |

* 0 représente l’index de la liste dans la table.
* Simplon : le nom de la station dont la liste contient les adjacent.
* (2) : le nombre d’adjacents de Simplon.
* Porte de Clignancourt | Marcadet-Poissonniers | : la liste des adjacent à Simplon.

## Sortie console : chemin entre deux points

Le programme exécute tour à tour l’algorithme BFS puis Dijsktra en débutant à l’index dans l’adjacency list correspondant au node —bercy— ( starting point : 266 ).

L’algorithme BFS affiche le chemin (en index) emprunté par l’algorithme (class BFSShortestPaths algorithm path).

Les deux aglorithme affiche trois tableau utilisés lors de l’exécution : marked[] , distances[] et previouses[] .

Enfin, le programme affiche pour chaque algorithme le résultat du plus court chemin en index puis converti en nom de station :

* Pour le BFS :

[266, 156, 244, 167]

Bercy

Gare de Lyon

Châtelet

Hôtel de Ville

* Pour l’algorithme Dijsktra :

[266, 156, 263, 53, 167]

Bercy

Gare de Lyon distance from previous : 0.7756004347114352

Bastille distance from previous : 1.0545607850857195

Saint-Paul (Le Marais) distance from previous : 0.9813861081176881

Hôtel de Ville distance from previous : 1.0335574164992687

## Sortie console : calcul du diamètre

Le programme exécute tour à tour l’algorithme BFS puis Dijsktra.

Pour chaque node, l’excentricité est calculée et enregistrée dans un tableau qui est ensuite affiché ( excentricity of each vertex ).

De ce tableau est déduit le diamètre et le radius du graphe :

* Pour le BFS :

diameter : 34.0

radius : 17.0

* Pour Dijsktra

diameter : 32.98664272478596

radius : 16.940912151396866

Ainsi que la node d’où est issue le chemin du diamètre

* Pour le BFS :

starting point : 63

* Pour Dijsktra

starting point : 63

Enfin, de la même manière que pour l’affichage entre deux point prédéfinis, l’algorithme affiche le chemin suivi par le diamètre.

Nous vous remercions de votre lecture.