Atelier d'approfondissement en informatique: Graphes et Algorithmes

Quentin Garrido

21 avril 2019

Table des matières

1 Introduction

1.1 Objectifs

L'objectif de ce projet est d'implémenter des algorithmes de recherche de plus courts chemins dans un graphe, et plus particulièrement dans un graphe représentant le réseau de métro de Paris.
Nous implémenterons tout d'abord l'algorithme de Dijkstra, puis le modifierons pour qu'ils deviennent l'algorithme A* et nous optimiserons son temps d'éxécution grâce à des structures de données adaptées.

1.2 Utilisation

Tout le code source est disponible à l'adresse suivante : *https ://github.com/garridoq/metro-shortest-path.*

Tout les éxécutables devraient vous être fournis dans le mail et devraient fonctionner sans devoir les recompiler. Dans le cas contraire voici la démarche à suivre :

Un makefile est fourni pour la compilation, il servira à compiler les bibliothèques et les tests. Une fois le code source obtenu il faudra exécuter la commande suivante pour compiler les bibliothèques :

```
> make
```

Vous pourrez alors compiler tous les fichiers de tests de la manière suivante :

```
> make NOM.exe
```

Où NOM est le nom du fichier de test (pour test_heap.c, il faudra entrer make test_heap.exe).

Voici la liste des fichiers de test et leur utilisation :
— test_heap :./test_heap.exe , ce fichier permet de tester l'implémentation du tas binaire et de ses opérations primaires.
— test_dijkstra :./test_dijkstra.exe GRAPH DEBUT FIN , ce fichier va récupérer le graphe dans le fichier GRAPH, puis calculer le plus court chemin de DEBUT à FIN en utilisant l'agorithme de Dijkstra, l'A* et A* avec une file de priorité.
Un fichier EPS sera créé pour chaque algorithme.

À titre de référence, voici à quoi ressemble le graphe entier du réseau de métro, que nous utiliserons par la suite : Les stations sont réprésentées par les sommets et nous avons une chaîne entre deux sommets si ils sont reliés par le métro.

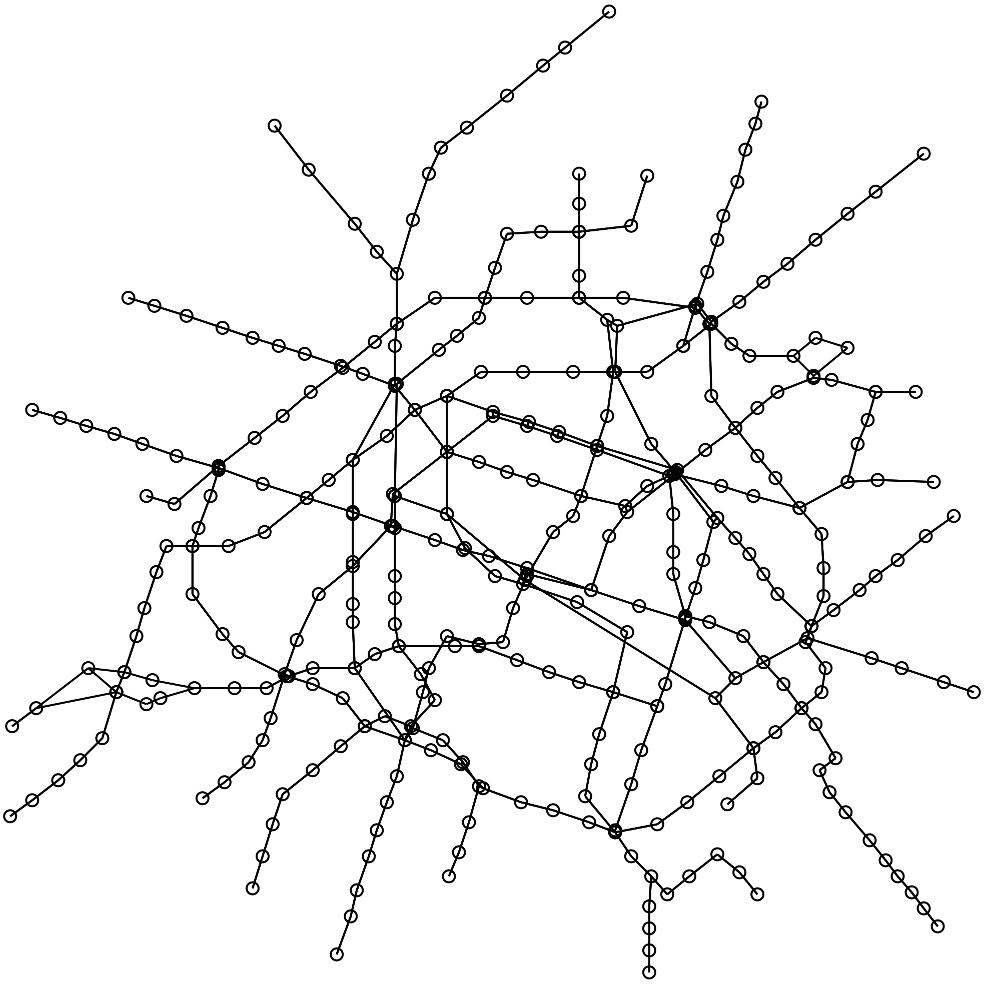

FIGURE 1 – Graphe de référence du métro

2 Algorithme de Dijkstra

2.1 Implémentation

Pour calculer le chemin de plus court d'un point D à A nous allons utiliser la version suivante de l'algorithme de Dijkstra, adaptée depuis le cours de l'unité Graphes et Algorithmes.

Ici nous n'avons pas besoin de calculer les chemins de notre sommet de départ vers tous les autres sommets du graphe et nous nous arrêterons donc dès que nous atteignons notre sommet d'arrivée.

Algorithm 1 Algorithme de Dijkstra

1: **procedure** DIJKSTRA($E, \Gamma, l, d \in E, a \in E$)
2:     S = {d}, $\pi(d) = 0$, k = 1, x_1 = d
3:     **for all** $x \in E \backslash \{d\}$ **do**
4:         $\pi(x) = \infty$
5:     **end for**
6:     **while** $k < n$ et $\pi(x_k) < \infty$ **do**
7:         **for all** $y \in \Gamma(x_k)$ tel que $y \notin S$ **do**
8:             $\pi(y) = \min[\pi(y), \pi(x_k) + l(x_k, y)]$
9:         **end for**
10:        Extraire $x \notin S$ tel que $\pi(x)$=min$\{\pi(y), y \notin S\}$
11:        k = k + 1, $x_k = x$, S = S $\bigcup\{x_k\}$
12:        **if** $x_k = a$ **then**
13:            **break**
14:        **end if**
15:    **end while**
16:    **return** π, S
17: **end procedure**

Nous implémentons S avec un tableau de $n = |E|$ éléments, correspondants aux sommets de notre graphe.
Ainsi nous l'initialiserons entièrement à 0 et S = S $\bigcup\{x_k\}$ correspondra à faire S[k] = 1.
Bien que cette implémentation soit plus coûteuse en mémoire qu'une liste chaînée elle permettra d'implémenter l'appartenance à S en temps constant, opération très utilisée aux lignes 7 et 10.
De plus l'ajout d'un élément sera aussi simplifié car nous n'aurons pas à vérifier l'appartenance avant de l'insérer ou non.

2.2 Résultats

Considérons un trajets des stations Alexandre Dumas (1) à Porte Dauphine (256). Nous trouvons alors le chemin le plus court suivant :

Alexandre Dumas − > Philippe-Auguste − > Père Lachaise − > Ménilmontant − > Couronnes − > Belleville − > Colonel Fabien − > Jaurès − > Stalingrad − > La Chapelle − > Barbès Rochechouart − > Anvers − > Pigalle − > Blanche − > Place de Clichy − > Rome − > Villiers − > Monceau − > Courcelles − > Ternes − > Charles de Gaulle, Étoile − > Victor Hugo − > Porte Dauphine

Nous pouvons observer ce chemin avec la figure suivante. Les chaînes forment le plus court chemin et les sommets sont uniquement ceux visités. Nous en avons visité 329 sur 376.

Comme nous pouvons le voir nous avons parcouru des sommets qui nous éloignaient grandement du résultat uniquement car le coût pour y aller était plus faible (cf ligne 10 de l'algorithme).

FIGURE 2 – Graphe de référence du métro

Cet effet est exacerbé ici car le trajet que nous avons choisi est particulièrement long.
Ce constat reste le même sur tous les trajets, sauf sur ceux très courts.

Bien que le chemin que nous trouvions ne paraisse pas optimal, nous l'avons vérifié via le service Vianavigo de la RATP, où nous trouvons le même chemin. Cela est du au fait que les deux stations sont sur la même ligne et donc que nous avons aucun changement à faire, d'où la plus courte durée du trajet.

Nous avons vérifié plusieurs autres trajets de la même manière avec le même résultat à chaque fois, ainsi notre implémentation semble être correcte, à condition que l'algorithme utilisé par la RATP pour Vianavigo soit correct, ce que nous pouvons affirmer être vrai.

3 Stratégie A*

3.1 Choix de l'heuristique

3.2 Implémentation de l'algorithme

Algorithm 2 Algorithme A*

procedure HEURISTIQUE($E, i \in E, a \in E$) ▷ a est notre point d'arrivée
    return distance_euclidienne(i, a)
end procedure

procedure A*($E, \Gamma, l, i \in E$)
    S = {i}, $\pi(i) = 0$, k = 1, x_1 = i
    for all $x \in E \backslash \{i\}$ **do**
        $\pi(x) = \infty$
    end for
    while $k < n$ et $\pi(x_k) < \infty$ **do**
        for all $y \in \Gamma(x_k)$ tel que $y \notin S$ **do**
            $\pi(y) = \min[\pi(y), \pi(x_k) + l(x_k, y)]$
        end for
        Extraire $x \notin S$ tel que $\pi(x)$=min{$\pi(y)$+heuristique(y), $y \notin S$}
        k = k + 1, $x_k = x$, S = S $\bigcup\{x_k\}$
    end while
    return π, S
end procedure

4 Tas Binaire

4.1 Implémentation classique

4.2 Implémentation pour notre problème

4.3 Autre structures de données possibles

5 Affichage des chemins