

Atelier d'approfondissement en informatique: Graphes et Algorithmes

Quentin Garrido

21 avril 2019

Table des matières

| | | |
|----------|---|----------|
| 1 | Introduction | 2 |
| 1.1 | Objectifs | 2 |
| 1.2 | Utilisation | 2 |
| 2 | Algorithme de Dijkstra | 4 |
| 2.1 | Implémentation | 4 |
| 2.2 | Résultats | 4 |
| 3 | Stratégie A* | 5 |
| 3.1 | Choix de l'heuristique | 5 |
| 3.2 | Implémentation de l'algorithme | 5 |
| 4 | Tas Binaire | 6 |
| 4.1 | Implémentation classique | 6 |
| 4.2 | Implémentation pour notre problème | 6 |
| 4.3 | Autre structures de données possibles | 6 |
| 5 | Affichage des chemins | 7 |

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'exécution 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 exécutable 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'algorithme 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 représentées par les sommets et nous avons une chaîne entre deux sommets si ils sont reliés par le métro.

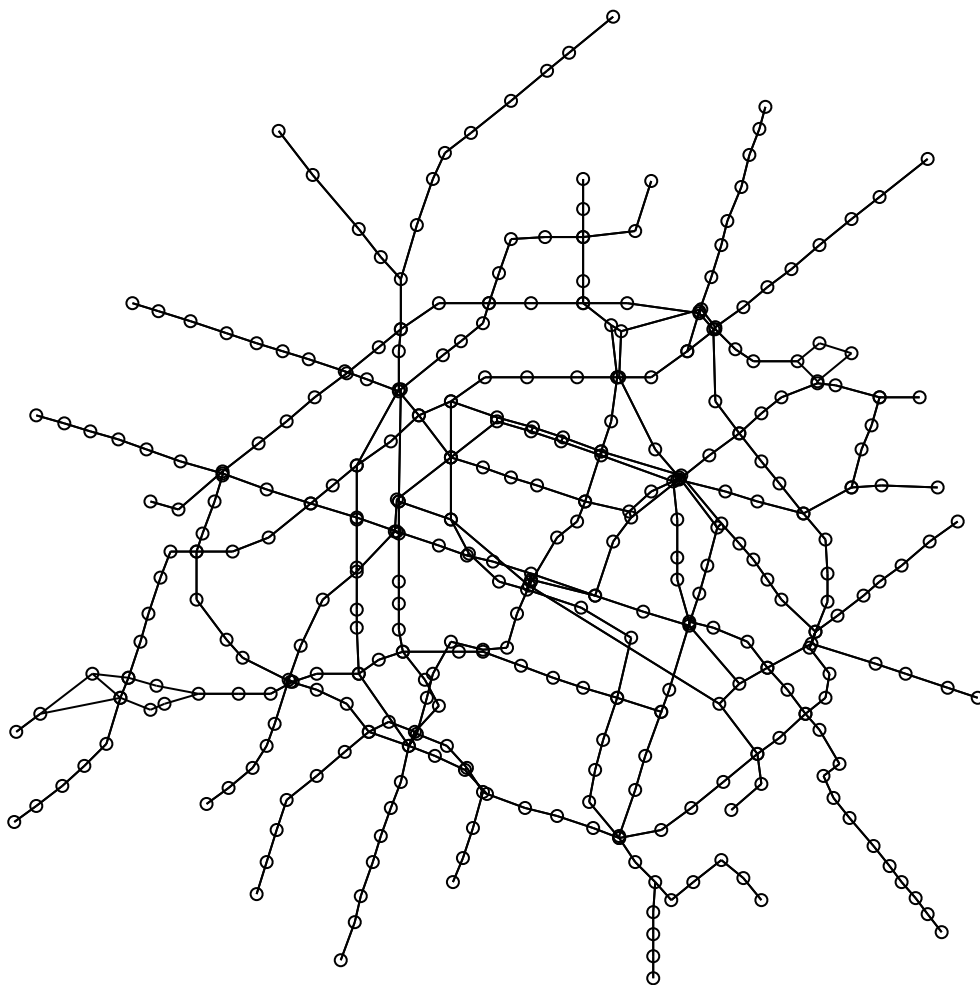


FIGURE 1 – Graphe de référence du métro

2 Algorithmes 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  $x \in E \setminus \{d\}$  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:     $k = k + 1, x_k = x, S = S \cup \{x_k\}$ 
12:    if  $x_k = a$  then
13:      break
14:    end if
15:  end while
16:  return  $\pi, S$ 
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 $S = S \cup \{x_k\}$ 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 Grande arche de la Défense (130) à Place d'Italie (244). Nous trouvons alors le chemin le plus court suivant :

3 Stratégie A*

3.1 Choix de l'heuristique

3.2 Implémentation de l'algorithme

Algorithm 2 Algorithme A*

```

1: procedure HEURISTIQUE( $E, i \in E, a \in E$ )                                 $\triangleright$  a est notre point d'arrivée
2:   return distance_euclidienne(i, a)
3: end procedure
4:
5: procedure A*( $E, \Gamma, l, i \in E$ )
6:    $S = \{i\}, \pi(i) = 0, k = 1, x_1 = i$ 
7:   for all  $x \in E \setminus \{i\}$  do
8:      $\pi(x) = \infty$ 
9:   end for
10:  while  $k < n$  et  $\pi(x_k) < \infty$  do
11:    for all  $y \in \Gamma(x_k)$  tel que  $y \notin S$  do
12:       $\pi(y) = \min[\pi(y), \pi(x_k) + l(x_k, y)]$ 
13:    end for
14:    Extraire  $x \notin S$  tel que  $\pi(x) = \min\{\pi(y) + \text{heuristique}(y), y \notin S\}$ 
15:     $k = k + 1, x_k = x, S = S \cup \{x_k\}$ 
16:  end while
17:  return  $\pi, S$ 
18: 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