SORBONNE UNIVERSITE

KHERDJEMIL Anis

Master 1 IRSI

[4AI02] Projet de C++ Avancé: Application RATP

2018/2019 .

Introduction:

Dans le cadre de ce projet, nous allons rédiger un algorithme qui calcule le chemin le plus court. C'est l'algorithme de Dijkstra qui nous permet de le faire. C'est un algorithme célèbre qui permet de calculer le chemin le plus optimale entre le point A et B. Une base de donnée est fourni qui représente le réseau de la Ratp (`s.csv`, `c.csv`). Le projet consiste à calculer le chemin le plus court entre deux stations du réseau Ratp.

La résolution du problème du plus court chemin :

Le but de ce projet est de faire une application qui établit le trajet le plus court entre deux stations du réseau de métro parisien.

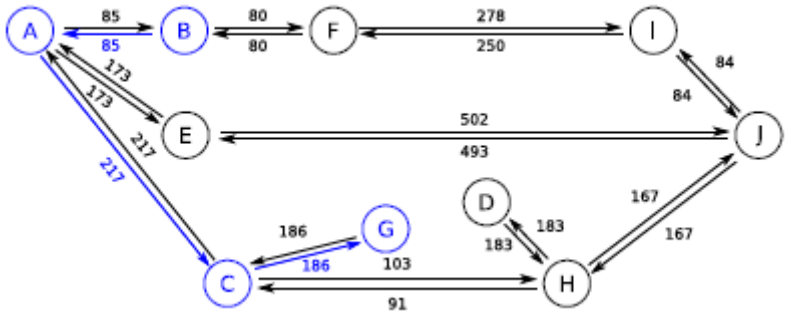

Figure 1: Exemple de graphe orienté

En théorie des graphes, un graphe oriente est un ensemble de points (ici, des entrées de stations de métro) connectés par des liens orientes (ici, les connexions entre les stations, en secondes). Dans le cadre de ce projet, chaque lien représente une possibilité de relier deux points, dans un sens précis, et avec un coût détermine. C'est une connexion possible entre ces points.

Le problème du plus court chemin, dans ce système, est d'évaluer le coût minimal pour aller d'un point A à un point B. L'algorithme de Dijkstra est une méthode de calcul an de résoudre ce problème.

Par exemple, dans le graphe de la Figure 1, le chemin le plus court entre le nœud B et le nœud G passe par les nœuds A et C, pour un coût cumule de 85+217+186=488. Ce chemin peut être exprimé comme une liste de nœuds et le coût cumulé du chemin, ici, best path = [{B : 0};{ A : 85}; {C : 302}; {D : 488}];

Cet algorithme peut s'appliquer sur des graphes orientés, modèle auquel plusieurs réseaux de transports sont disponibles (RATP, SNCF, routes, etc), et est également une porte d'entrée pour le domaine de la théorie des graphes et de la théorie de la décision.

Algorithme de Dijkstra

L'algorithme de Dijkstra sert à résoudre le problème du plus court chemin. Il permet, par exemple, de déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région. Plus précisément, il calcule des plus courts chemins à partir d'une source dans un graphe orienté pondéré par des réels positifs. On peut aussi l'utiliser pour calculer un plus court chemin entre un sommet de départ et un sommet d'arrivée.

Voici les étapes qui cet algorithme suit:

En supposant que le nœud par lequel on commence est appelé le nœud initial. La distance du nœud Y est la distance depuis le nœud le nœud initial à Y. L'algorithme de DIJKSTRA attribue des valeurs de distances initial et les améliore étape après étape.

La première étape de l'algorithme de Dijkstra est de marquer tous les nœuds non visités et créer un ensemble de tous les nœuds non visités appelé ensemble unvisited (Q).

Ensuite, on attribue à chaque nœud une valeur de distance provisoire, cette valeur est égale à zéro pour le nœud initial et égale à l'infini pour tous les autres nœuds et on définit le nœud initial comme étant le nœud courant.

Pour le nœud courant, on considère tous ses voisins non visités et on calcule leurs distances provisoires à travers le nœud courant. on compare la distance provisoire nouvellement calculée à la valeur actuellement assignée, ensuite on affecte la plus petite.
Par exemple, si le nœud actuel A est marqué d'une distance de 6 et que le bord le reliant à un voisin B a un coût 2, la distance entre B et A sera de 6 + 2 = 8. Si B était précédemment marqué avec une distance supérieure à 8, on la remplace par 8, sinon on conserve la valeur actuelle

Lorsqu'on a terminé d'examiner tous les voisins non visités du nœud courant, on marque le nœud courant comme visité et on le supprime de l'ensemble unvisited. Un nœud visité ne sera plus jamais vérifié.

Si le nœud de destination a été marqué comme visité (lors de la planification d'un itinéraire entre deux nœuds spécifiques) ou si la distance minimale entre les nœuds de l'ensemble unvisited est l'infini (lors de la planification d'un parcours complet; survient lorsqu'il n'y a pas de connexion entre le nœud initial et les nœuds non visités restants), on arrête. L'algorithme est terminé.

Sinon, on sélectionne le nœud non visité qui est marqué avec la plus petite distance d'essai, et on le définit le comme nouveau "nœud courant" et on revient à l'étape 3.

Lors de la planification d'un itinéraire, il n'est pas nécessaire d'attendre que le nœud de destination soit "visité" comme ci-dessus: l'algorithme

peut s’arrêter dès que le nœud de destination a la distance minimale entre tous les nœuds "non visités" (et peut donc être sélectionné comme suivant "courant").

Voici un exemple ou on cherche le chemin le plus court entre le nœud M et S.

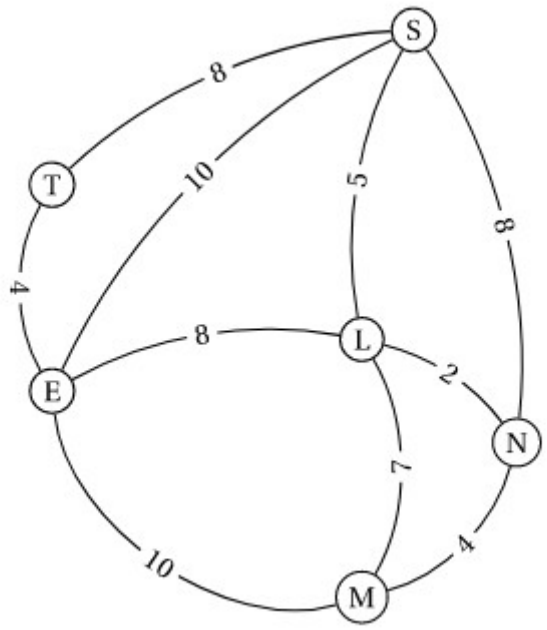

Figure 2: exemple de graphe

Les différentes étapes de calculs cités précédemment sont appliquées et résumées dans les tableaux suivant :

Initialisation

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞

Étape 1

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)						

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞

Étape 2

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)						

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞

Étape 3

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞
L (6)						

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞
L (6)	10_M				11_L	∞

Étape 4

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞
L (6)	10_M				11_L	∞
E (10)						

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞
L (6)	10_M				11_L	∞
E (10)					11_L	14_E

Étape 5

	E	L	M	N	S	T
Départ	∞	∞	0_M	∞	∞	∞
M (0)	10_M	7_M		4_M	∞	∞
N (4)	10_M	6_N			12_N	∞
L (6)	10_M				11_L	∞
E (10)					11_L	14_E

le chemin le plus court trouvé est M-N-L-S

l'algorithme de Dijkstra est traduit en pseudo-code suivant:

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:
6          dist[v] ← INFINITY
7          prev[v] ← UNDEFINED
8          add v to Q
10     dist[source] ← 0
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]
14
15         remove u from Q
16
17         for each neighbor v of u:           // only v that are still in Q
18             alt ← dist[u] + length(u, v)
19             if alt < dist[v]:
20                 dist[v] ← alt
21                 prev[v] ← u
22
23     return dist[], prev[]
```

Figure 3: Pseudo-code de l'algorithme de Dijkstra [1]

Dans notre cas, on cherche le chemin le plus court en une station départ et une station finale. On termine la recherche après la ligne 15 si u=station finale. Le meilleur chemin est lu par itération inverse

```
1  S ← empty sequence
2  u ← target
3  if prev[u] is defined or u = source:          // Do something only if the vertex is reachable
4      while u is defined:                       // Construct the shortest path with a stack S
5          insert u at the beginning of S        // Push the vertex onto the stack
6          u ← prev[u]                           // Traverse from target to source
```

Figure 4 : Pseudo-code du meilleur chemin[1].

Implémentation :

L'implémentation de l'application Ratp est faite suivant une feuille de route, nous avons d'abord implémenter la fonction read_stations dans une classe qui hérite de Generic_station_parser. C'est une fonction qui prends un fichier (.csv) en argument, ce fichier est ouvert en lecture via std::ifstream. Ensuite, les ligne du fichier sont lu par la fonction getline qui récupère tout les caractères jusqu'au délimitateur ' ,' . Les caractère récupérés sont mis dans des variable string. L'id de la station est converti en uint64_t puis mis dans la table de hachage station_hashmap et est associé une station.

Après l'implémentation de la fonction read_stations, la fonction read_connection qui hérite de Generic_connection_parser est implémenté de la même façon, les trois paramètres récupérés sont converti en uint64_t est sont mis dans la table connection_hashmap en associant le une paire de l'id et coût d'une station à une station départ.

La suite du travail repose sur l'implémentation de l'algorithme de Dijkstra et l'affichage. Un vecteur de paire est créé pour le meilleure chemin, une map pour les nœuds non-visités est crée qui prend en argument un nœud et un booléen qui dit si un nœud existe dans cette map ou pas. Une autre map est crée qui associe un nœud à une paire distance_to_next_node et le nœud précédent qui est appelé table_Dijkstra. Après déclaration des maps, le pseudo-code est implémenté sachant que l'initialisation à l'infini est faite par INT_MAX inclus dans la bibliothèque <climits> et les nœuds précédent sont initialisés à -1. Lorsque la map unvisited n'est pas vide, on calcule toujours la distance minimale entre le nœud actuelle et les autres nœuds et on retire la nœud visité de Unvisited. Cette dernière est vérifié par la fonction isNotEmpty(unvisited) qui renvoie true si unvisited n'est pas vide et false si elle est vide. la fonction FindMin cherche la distance minimale entre un nœud est ses successeurs et renvoie le nœud avec la plus petite distance. Lorsqu'on visite la nœud final, on arrête [2].
Le reconstruction du meilleur chemin se fait par itération inverse. Le vecteur best_path renvoie des paire (id,coût) du meilleur chemin.Un exemple est donné ci-dessous:

```
ubuntu@ubuntu:~/Desktop/Gmail$ ./mon_programme
1166840 180
1166838 122
6129304 134
1166834 116
2155    180
2288    104
2522    113
2111    150
2062    0
ubuntu@ubuntu:~/Desktop/Gmail$
```

Après implémentation de la fonction compute_and_display_travel, nous obtenons le résultat suivant :

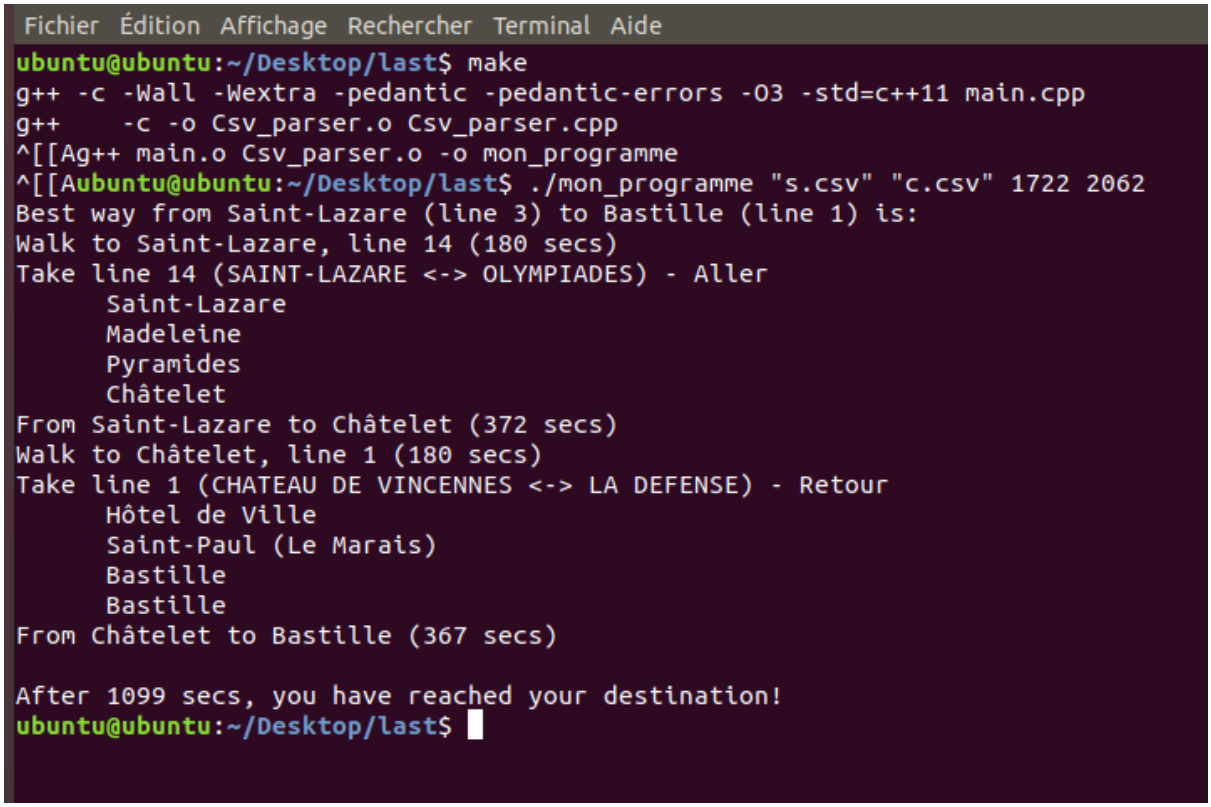

Figure 5 : meilleur chemin entre Saint-Lazzare et Bastille (1722,2062)

Conclusion

Ce projet m’a permit de travailler en autonomie et d’être autodidacte. La construction d’un code étape par étape suivant une feuille de route était une nouvelle expérience pour moi. Cette méthodologie de travail m’a permit d’apprendre une nouvelle façon de programmer. La compréhension de la structure était nécessaire pour la réalisation de ce projet. Sa m’a permit aussi d’appliquer tout ceux qu’a était vu en cours durant le semestre et d’apprendre d’avantage sur le langage de programmation C++

Référence

[1] “ Dijkstra's algorithm“ Wikipédia,2 May 2019

[2] Computer science,Graph Data Structure 4. Dijkstra’s Shortest Path Algorithm,2016