Sorbonne Université
Master Automatique, Robotique
Nom : NEHMAR Anis,
Date : 07/06/2021

Rapport sur l'implémentation d'algorithme de Dijkstra sur C++

Algorithme de DIJKSTRA

L'algorithme de Dijkstra permet de trouver le plus court chemin entre deux points dans un réseau c'est-à-dire dans un **graph à poids** (orienté ou non orienté), Dans notre cas le poids représente le temps de trajet entre deux stations.

l'algorithme **prend en entrée un graph qui représente les différentes stations pondérés par des le temps le trajet**, et on obtient en sortie une séquence de nœuds de plus court chemin pour aller d'une source jusqu'à la station de destination qu'on le construit au fur et à mesure de l' exécution de l'algorithme.

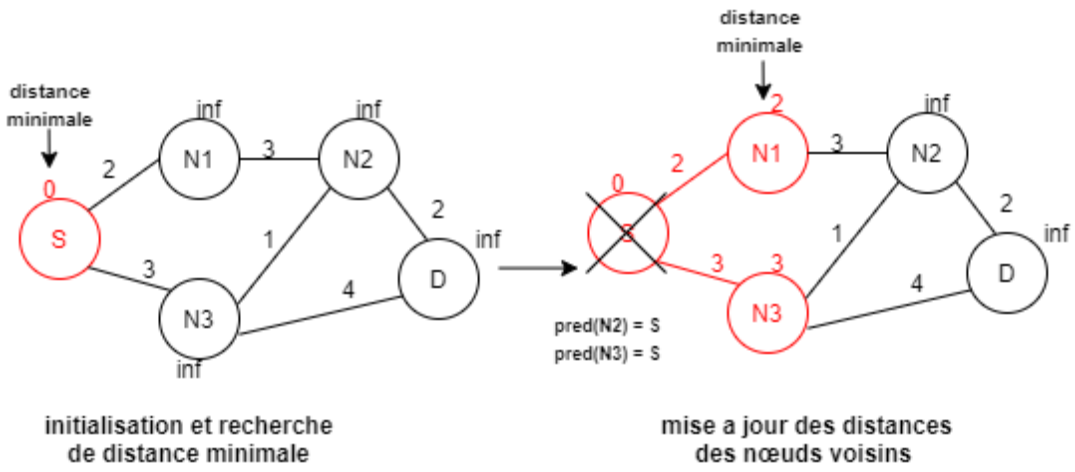
Principe

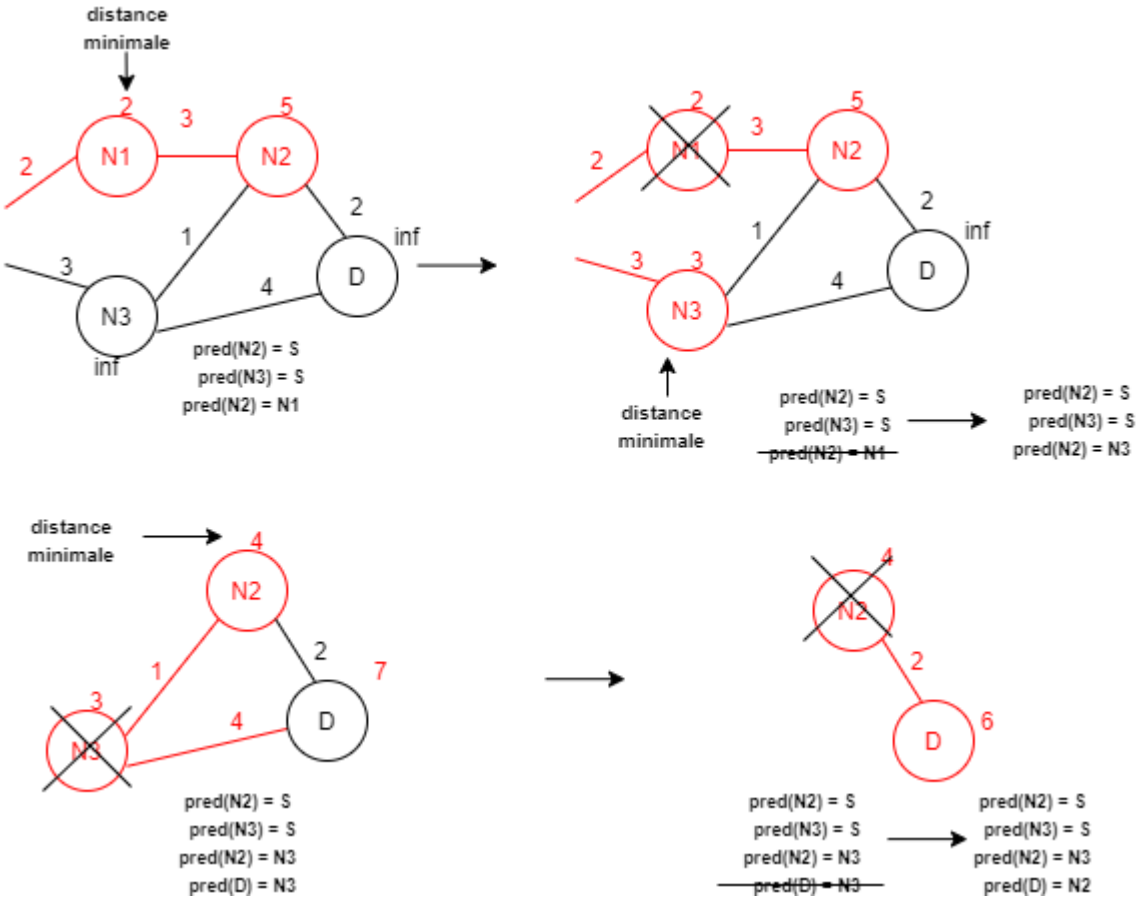
Au départ, on initialise les distance de chaque nœud (station) à l'infini, sauf la station de départ ou la distance est nulle. Apres , à chaque itération on choisit le nœud avec la distance minimale, Ensuite, on mets à jour la distance des nœuds voisins comme suit : la nouvelle distance de nœud voisin est le minimum entre la distance de ce dernier existence et celle obtenu (distance minimale) en ajoutant le poids en question (c'est-à-dire le poids entre le nœud avec la distance minimale et son voisin), et on enlève le nœud dont avec la distance minimale.

On répète ainsi, jusqu'à l'arrivé au nœud d'arrivé ou bien l'épuisement des nœuds.

A chaque mise à jour de la distance d'un nœuds, on enregistre ou bien on mise à jour son prédécesseur (qui représente le nœud avec la distance minimale), à la fin d'exécution du l'algorithme, les prédécesseurs obtenus nous permet de construire le chemin plus court on les parcourant de dernier élément jusqu'à l'arrivé du nœud de départ.

Voici un schéma explicatif : S --> départ, D --> Arrivée





A partir des prédécesseurs obtenues, on peut en conclure le court chemin suivant :

S -> **N3** -> **N2** -> **D**

Pseudo-code

Entrée : graph G (stations, poids :connections entre stations**)**
Nœud S (départ**) , Nœud D (** Arrivée)
Q - -> ensemble vide
Pour chaque n in G.**noeuds** {
Dist**[**n**] =** infini
Q.ajouter(n)
}
Dist**[S] =** 0
Tant que Q **n'est pas vide{**
U <--- Q.noeuds_avec_distance_minimale
Q.supprimer(U)
N <-- G.noeuds_voisins_de_U
Pour chaque n **de** N {
X **=** Dist **[**U**]** + G.poid(U,n)
Si X **<** Dist**[**n**]{**
Dist**[**n**] = X**
Pred**[**n**] = U**
}
}
}
Construire **meilleur chemin** à partir de **pred**
Renvoie meilleur chemin

Problèmes rencontrés

- Pendant la lecture des données de fichier Excel, j'avais du mal à séparer les lignes.
  Solution :
  Ajouter un compteur (Vu qu'on connait le nombre des mots dans chaque ligne), ce qu'il nous permet de connaitre qu'on est ce que on va séparer le flux de données lus par la virgule '**,**', ou bien par la caractère spéciale '**\n**'.

- Pendant l'implémentation d'algorithme de **DIJKSTRA,** je me retrouvais toujours dans une boucle infinie, ce qui a causé ce problème c'est que DISKSTRA a besoin de connaitre les stations et les distances associées, alors j'ai créé une **unordred_map** pour enregistrer les distances tel que : **key** = **id** de la station, et au fur et à mesure j'enlève le nœud avec la distance minimale de structure unordred_map créer, alors l'information sur la distance est aussi supprimé, cela à causé une boucle infinie, donc pour remédier à ce problème, j'ai utilisé deux **unordred_map** :
  La première : pour contient les stations (key = id_station).
  La deuxième : contient les distance (key = id_station).

Choix de containers

- J'ai utilisé la structure **unordred_map** plusieurs fois, vu que j'avais toujours besoin de id de la stations, avec **unordred_map** c'était utile de l'utiliser comme étant un key.
- J'ai aussi utilisé la structure **vector** pour enregistrer les prédécesseurs et le plus court chemin, car il fallait l'utilisation d'une liste chainé pour enregistrer le meilleur chemin dans l'ordre et ajouter le nœuds successivement suivant avec **vector::push_back**.

Conclusion

L'algorithme de Dijkstra est utilisé pour la recherche de graphes. Il est très optimal, et permet de trouver l'unique chemin le plus court. De plus de notre problème (chemin plus court dans le réseau de trafic RATP), et comme tout problème d'optimisation combinatoire peut être formulé comme un problème de plus court chemin, l'algorithme de Dijkstra est également important pour la recherche en IA.