Projet Liv’In-Paris - Etape 2 (4 avril)

Introduction

Liv'in Paris est une application qui facilite le partage de repas entre voisins à Paris. La plateforme permet aux particuliers ou aux commerces locaux de commander des plats faits maison préparés par des cuisiniers inscrits. Les utilisateurs peuvent s'inscrire en tant que cuisinier, client, ou les deux simultanément, en fournissant leurs informations personnelles telles que leur nom, adresse, numéro de téléphone et adresse e-mail.

Ce projet vise à explorer les graphes et leurs fonctionnalités, notamment dans l'optimisation des itinéraires de livraison et l'analyse des interactions entre les utilisateurs. Ce dépôt est un projet destiné aux étudiants en informatique de A2/S4.

Voici le lien vers notre projet : https://github.com/jojo2504/Living-Paris.
Les membres de notre groupe sont représentés par les pseudo suivant sur github :

- ZILBER Benjamin : Bahyamin
- TRAN Jonathan : jojo2504
- THERY Thibault : Tibo-7

Analyse des algorithmes du plus court chemin

1. Analyse théorique

Afin de déterminer le plus court chemin entre 2 adresses Client et Cuisinier, représentées par 2 stations de métro dans notre graphe, nous avons implanté les 3 algorithmes demandés : Dijkstra, Bellman-Ford, Floyd-Warshall. Nous avons également ajouté un algorithme de recherche supplémentaire : A*. Commençons par une analyse théorique de ces différents algorithmes pour trouver le chemin le plus court. Chaque algorithme a des contextes d'utilisation différents. Notre graphe est noté G= (V,E) où V est le nombre de sommets (ou nœuds) du graphe et E le nombre d'arêtes. On donnera à chaque fois la complexité dans le pire des cas. Pour la complexité spatiale, on prend en compte la structure de données du graphe qui est une liste d'adjacence.

Algorithme de Dijkstra

L'algorithme de Dijkstra est utilisé pour trouver le plus court chemin entre un nœud de départ et tous les autres nœuds dans un graphe pondéré à poids non négatif. A partir du sommet source, on maintient une liste des distances les plus courtes possibles de ce sommet à tous les autres sommets, et on les met à jour au fur et à mesure de l'exploration des voisins.

Complexité Théorique :

- **Complexité en temps** : O(E + V*log V)
- **Complexité en espace** : O(V + E), car il faut stocker les distances et les prédécesseurs de chaque nœud, ainsi que les arêtes du graphe. Cette complexité sera la même pour les autres sauf Floyd-Warshall.

L'algorithme est particulièrement adapté pour gérer des graphes denses sans cycle négatif. Dans notre cas, Dijkstra serait efficace pour traiter les parcours entre les stations de métro, ayant de multiples connexions entre elles. Cet algorithme a d'ailleurs comme application réelle les problèmes de navigation ou de routage.

Algorithme de Bellman-Ford

Il est lui aussi utilisé afin de trouver les chemins les plus courts depuis un sommet du graphe, mais cet algorithme a comme particularité de gérer les graphes à poids négatif afin de trouver des cycles négatifs dans les graphes.

Complexité Théorique :

- **Complexité en temps** : O(V * E), chaque arête est vérifiée pour chaque sommet.

- **Complexité en espace** : O(V+E)

Bellman-Ford serait utile si certaines lignes du métro avaient des coûts négatifs (ce qui n'est pas le cas ici). Bellman-Ford serait plus lent que Dijkstra avec notre graphe qui dépasse 300 nœuds, car sa complexité en temps est plus élevée (O(V * E) au lieu de O(V* logE)).

Algorithme de Floyd-Warshall

L' algorithme de Floyd-Warshall est une méthode de programmation dynamique utilisée pour trouver les chemins les plus courts entre toutes les paires de sommets dans un graphe orienté. Il renvoie en effet une matrice contenant les plus courtes distances entre tous les sommets. Il peut être utilisé si le graphe présente des poids négatifs.

Complexité Théorique :

- **Complexité en temps** : $O(V^3)$ L'algorithme doit passer par toutes les triplets (i, j, k), ce qui le rend moins performant que Dijkstra et Bellman-Ford pour les graphes de grande taille.

- **Complexité en espace** : $O(V^2)$, car il faut stocker une matrice de taille V x V des distances entre tous les nœuds.

En raison de sa complexité relativement coûteuse, il vaut mieux l'utiliser dans le cas de petit graphe. De plus, il est particulièrement adapté pour les graphes où tous les chemins entre chaque paire de sommets sont nécessaires (ex: obtenir la

distance entre chaque paire de stations). Floyd-Warshall est optimal si l'on doit connaître les plus courts chemins entre tous les nœuds, et pas seulement entre un couple spécifique.
Il n'est donc pas adapté car nous voulons uniquement le plus court chemin entre 2 stations spécifiques.

Algorithme A*

L'algorithme A* (A star) recherche le chemin le plus court entre un nœud initial et un nœud final, tous deux donnés. Il a comme particularité d'effectuer une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique.

Une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable (pas nécessairement optimale ou exacte) pour un problème d'optimisation difficile. Pour bien fonctionner, elle ne doit pas surestimer le coût réel entre deux nœuds (elle peut sous-estimer mais pas surestimer) : on dit alors qu'elle est admissible. Une heuristique pourrait être par exemple la **distance géographique à vol d'oiseau** (ou **distance euclidienne**), adaptée à un espace plat. Il y a aussi la **formule de Haversine** pour une surface sphérique, en prenant en compte les coordonnées GPS des 2 stations..

Complexité Théorique :

- **Complexité en temps** : O(E*logV). Cela dépend fortement de la qualité de l'heuristique. Si l'heuristique est bien choisie, A* explore moins de nœuds que Dijkstra, ce qui le rend plus rapide.

- **Complexité en espace** : O(V+E)

D'après nos analyses théoriques, l'algorithme A* est le meilleur en termes de complexité en espace et de complexité en temps.

2. Conclusion

Dans le cadre du projet Liv'in Paris où l'on cherche le chemin le plus court entre 2 stations de métro possédant des coordonnées géographiques, nous avons déterminé que l'algorithme A* était le plus efficace et le plus pertinent. En utilisant une heuristique, comme la distance euclidienne entre les coordonnées de 2 stations, A* nous permet de trouver le chemin le plus court en moins d'étapes, car il réduit le nombre de sommets visités par rapport à Dijkstra, tout en assurant un chemin optimal. A* possède la meilleure complexité théorique, se révèle plus rapide en pratique et est donc meilleur que les 3 autres algorithmes.

En effet, bien qu'efficace et approprié dans notre cas, Dijkstra explore des stations inutiles. Bellman-Ford est quant à lui trop lent et sans poids négatif n'est pas intéressant. Floyd-Warshall est inadapté, car on ne cherche pas les chemins les plus courts entre toutes les stations.

Ainsi, A* est l'algorithme le plus pertinent pour optimiser les performances de calcul du plus court chemin dans ce réseau de métro.