Compte rendu

A : complexité des algorithmes de plus court chemin.

Comparons les trois algorithmes Dijkstra, Bellman-Ford, et Floyd-Warshall en termes de complexité algorithmique, rapidité d'exécution et de pertinence.

1. Dijkstra

Complexité algorithmique :

-Dans le cas général : La complexité de l'algorithme de Dijkstra dépend de la structure de données utilisée pour représenter le graphe. La complexité est de **O(E log V)**, où **E** est le nombre d'arêtes et **V** le nombre de sommets.

Rapidité d'exécution :

-Dijkstra est très efficace pour les graphes non négatifs et les graphes peu denses. Il est plus rapide que les autres algorithmes dans ce cas, surtout lorsque le nombre d'arêtes est significativement plus petit que le carré du nombre de sommets.

Pertinence pour ce cas d'application :

-Pertinent : Dijkstra est particulièrement adapté lorsqu'on cherche à trouver les plus courts chemins à partir d'un nœud source unique vers tous les autres nœuds. Il est donc très utile pour des applications de type réseaux de transport ou routing, comme dans le cas de graphes représentant des réseaux de routes ou de communication.

2. Bellman-Ford

Complexité algorithmique :

Complexité : Bellman-Ford a une complexité de **O(V * E)**, ce qui le rend moins efficace que Dijkstra dans les graphes denses ou avec de nombreuses arêtes. Il est plus lent, surtout pour les grands graphes, mais il est capable de gérer des arêtes de poids négatif.

Si les arêtes ont des poids négatifs, Dijkstra échoue, mais Bellman-Ford peut toujours fonctionner.

Rapidité d'exécution :

-Plus lent que Dijkstra dans les graphes sans arêtes négatives. Cependant, pour les graphes avec des poids négatifs, Bellman-Ford reste le meilleur choix, car il est capable de détecter les cycles de poids négatif (et donc de signaler qu'il n'y a pas de solution).

Pertinence pour ce cas d'application :

-Pertinence : Si le graphe contient des arêtes avec des poids négatifs, Bellman-Ford est un bon choix, car il peut gérer ce cas, contrairement à Dijkstra. Pour des applications où les arcs ont des valeurs négatives, comme dans certains types de réseaux économiques ou financiers, Bellman-Ford serait plus approprié. Sinon, il est moins efficace que Dijkstra.

3. Floyd-Warshall

Complexité algorithmique :

-Complexité : L'algorithme de Floyd-Warshall a une complexité de **$O(V^3)$,** ce qui est beaucoup plus coûteux en termes de temps de calcul. Il est donc moins adapté pour les graphes très grands.

-Globalement : Floyd-Warshall calcule tous les plus courts chemins entre toutes les paires de nœuds, ce qui entraîne une croissance exponentielle du temps d'exécution avec le nombre de nœuds dans le graphe.

Rapidité d'exécution :

-Très lent pour des graphes de grande taille ou pour des graphes avec un grand nombre de sommets (V). Il est plus adapté aux petits graphes où le coût en mémoire et en temps n'est pas prohibitif.

-La vitesse d'exécution pour un graphe de taille moyenne à grande est donc bien inférieure à celle de Dijkstra et Bellman-Ford.

Pertinence pour ce cas d'application :

-Pertinence : Floyd-Warshall est adapté si on a besoin de connaître les plus courts chemins entre toutes les paires de nœuds. Cela est utile si l'on souhaite résoudre un problème de type toutes les paires, comme calculer les itinéraires les plus courts entre tous les nœuds d'un réseau.

-Pour les graphes très denses ou avec un petit nombre de nœuds, Floyd-Warshall peut être une bonne solution, mais dans des cas pratiques de graphes plus grands, il devient rapidement impraticable.

Comparaison des trois algorithmes

Algorithme	**Complexité (temps)**	**Rapidité d'exécution**	**Pertinence pour ce cas d'application**
Dijkstra	O(E log V)	Très rapide	Idéal pour les graphes avec arêtes positives et un point de départ unique.
Bellman-Ford	O(V * E)	Plus lent que Dijkstra	Utile si des poids négatifs existent ou pour la détection de cycles négatifs.
Floyd-Warshall	$O(V^3)$	Très lent	Idéal pour les plus courts chemins entre toutes les paires de nœuds, mais impraticable pour des graphes larges.

Conclusion :

Dijkstra est le plus rapide pour les graphes sans poids négatifs et convient parfaitement aux réseaux de transport, où l'on cherche à trouver le chemin le plus court à partir d'une station donnée. Bellman-Ford, bien qu'il puisse gérer des poids négatifs, est plus lent et moins adapté aux grands réseaux comme celui du métro parisien. Quant à Floyd-Warshall, sa complexité en $O(V^3)$ le rend inutilisable pour un réseau aussi vaste que celui du métro de Paris.

Ainsi, pour optimiser les trajets et offrir une réponse rapide aux usagers cherchant le plus court chemin entre deux stations, l'algorithme de **Dijkstra** est le choix le plus adapté au réseau de transport parisien.

B : Utilisation d'IA dans le projet.

Dans de notre projet, nous avons eu recours à l'intelligence artificielle à plusieurs reprises pour nous accompagner dans le développement et l'optimisation de notre code.

Tout d'abord, nous avons utilisé l'IA pour la création de certaines classes clés de notre projet, notamment **DessinerGraph** ainsi que la fin de la partie de la classe **FloydWarshall**. L'IA nous a aidés à structurer ces classes et à générer du code efficace sur des sujets que nous ne maitrisions pas encore.

Ensuite, nous avons fait appel à l'IA pour résoudre plusieurs problèmes sur nos environnements de code sur GitHub. L'IA nous a permis d'identifier rapidement l'origine des erreurs et de proposer des solutions adaptées, nous évitant ainsi de perdre du temps sur des problèmes techniques bloquants.

Enfin, nous avons également utilisé l'IA pour l'optimisation de notre code. À plusieurs endroits du projet, l'IA nous a suggéré des améliorations en termes de performance et de clarté du code, ce qui nous a permis d'obtenir un programme plus efficace et plus lisible.

L'IA s'est donc révélée être un outil précieux tout au long de notre développement, nous aidant à surmonter certains défis techniques et à améliorer la qualité de notre code.