Rapport de projet informatique

Projet : Plus Court Chemin(PCC) dans un graphe

Sommaire

1. Avant propos et mise en contexte

Ce projet vise à calculer le meilleur itinéraire entre 2 villes ou 2 stations de métro en comparant 3 algorithmes calculant le plus court chemin dans un réseau. En outre, il faudra tester ces codes dans des conditions plus réelles où le nombre de données est important et où les problèmes de mémoires et de complexité deviennent alors de réels enjeux d'optimisation. Enfin, au cours de ce projet, nous travaillerons en mode projet. Ainsi, nous développerons donc nos compétences de communication et d'organisation tout en approfondissant notre maîtrise des outils de développement.

2. Implantation

a. Etat du logiciel :

Ce qui fonctionne	Ce qui ne fonctionne pas
Tout semble fonctionner : Le script évaluation a été complété pour pouvoir comparer tous les résultats fournis. On obtient 386/386 tests réussis après exécution du programme.	La complexité des parcours en largeur et en profondeur sont cohérentes contrairement à celle de Dijkstra qui est anormalement trop élevée… Cela peut être dû à une implémentation trop coûteuse, avec des appels de fonctions, ou d'autres procédés ralentissant les temps de calculs. Ce point reste à améliorer.

b. Analyse des performances CPU et mémoires, complexité théorique et expérimentale :

-> Choix et modifications de structures de données :

Structure *graph_t* :

Deux champs ont été ajoutés à la structure de données initiales.
Premièrement, un champ "*oriente*" booléen, représenté par un entier de type int, qui permet de garder en mémoire si le graphe est orienté ou non.
Cela permet lors de la lecture des arcs du graphe de savoir si un seul arc est à ajouter au graphe ou si le retour est aussi à ajouter, dans le cas d'un graphe non orienté.
Le deuxième champ ajouté est aussi un entier, "*initialise*" permet de garder en mémoire si le graphe a été initialisé. Il sert uniquement pour la recherche de plus court chemin avec un parcours en profondeur. Le graphe étant initialisé au début de l'appel de fonction, un appel récursif de la fonction aurait pu causer

une initialisation en boucle du graphe. Ce champ est une manière de remédier à ce type de problèmes.

Structure *vertex_t* :

Trois champs ont été ajoutés à la structure de données initiale. Ce sont trois entiers. Le premier, "*plusproche*" permet de tenir à jour le plus proche voisin, par pcc croissants, d'un sommet lors de la recherche de plus court chemin.Il sert ainsi à simplifier la reconstruction du plus court chemin si ce dernier a été trouvé. Le deuxième, "*visited*" permet de savoir si le sommet courant est présent dans la liste, la file ou le tas des sommets de recherche de plus court chemin. Finalement, le champ "*indiceTas*" tient à jour la position du sommet courant dans le tas de sommet.

Structure *listedge_t* :

Structure usuelle de liste. Utilisée exclusivement pour les listes d'arcs pour les graphes. Les maillons sont composés d'un arc et d'un pointeur sur maillon.

Structure *fifoindice_t* :

Structure usuelle de file, sous forme de liste chaînée. Utilisée exclusivement pour les files d'indice de sommet pour la recherche de plus court chemin par parcours en largeur. Les maillons sont composés d'un entier et d'un pointeur sur maillon.

Les autres structures de données de tas (utilisées pour Dijkstra) et dictionnaires (utilisés pour le métro, détermination d'indice de sommet à part de leur nom) sont usuelles et aucun champ n'a été ajouté.

-> <u>Compléxité :</u>

On calcule la complexité théorique des 3 algorithmes de calcul de plus court chemin dans un graphe qui étaient proposés par le sujet.
Les graphes sont considérés à n sommets et m voisins, on donnera donc la complexité en fonction de n et en fonction de m.

Algo	Complexité dans le pire des cas en fonction de n	Complexité dans le meilleur des cas en fonction de n	Complexité dans le pire des cas en fonction de m	Complexité dans le meilleur des cas en fonction de m
dfs	$O(m^n)$=>O(exp(nlog(m)))	O(1)	$O(m^n)$	O(1)
bfs	$O(m*n^2)=O(n^2)$	O(n*m)=O(n)	O(m)	O(m)
dijkstra	$O(n^2*m)=O(n^2)$	O(n)	O(m)	O(1)

On calcule maintenant la complexité théorique des 3 algorithmes de calcul de plus court chemin dans un graphe que nous avons codé.
Les graphes sont considérés à n sommets et m voisins, on donnera donc la complexité en fonction de n et en fonction de m.

Algo	Complexité dans le pire des cas en fonction de n	Complexité dans le meilleur des cas en fonction de n	Complexité dans le pire des cas en fonction de m	Complexité dans le meilleur des cas en fonction de m
dfs	$O(m^n)$=>O(exp(nlog(m)))	O(1)	$O(m^n)$	O(1)
bfs	$O(n*m^2)=O(n)$	O(n*m)=O(n)	O(m)	O(m)
dijkstra	$O(n^2*m^2)$=O(n2)	O(nlog(n))	$O(m^2)$	O(1)

3. Suivi

c. *Organisation au sein de l'équipe :*

Tout d'abord, nous avons tous les deux travaillé de notre côté sur les structures à mettre en place, ainsi que sur les fonctions sur les listes, files, tas et dictionnaires. Mounsef s'est ensuite chargé plus particulièrement du PCC par BFS, de la lecture de fichiers, de l'affichage des résultats ainsi que l'application du PCC au métro parisien alors que Jean-Baptiste a travaillé sur le PCC par DFS et par Dijkstra ou BFS amélioré.
Systématiquement, nous nous sommes corrigés, relus mutuellement et avons testé, débuggé nos programmes.

d. *Notes (total sur 30)*

Mounsef : 15
Jean-Baptiste : 15

2. Conclusion

Finalement, ce projet nous a permis de comparer trois algorithmes que nous avons pu implémenter pour le calcul du plus court chemin dans un réseau afin de trouver le meilleur itinéraire entre deux villes ou deux stations de métro. Nous avons pu approfondir notre connaissance du langage C et des différentes structures de données. Tout au long de ce projet, nous nous sommes confrontés à des problèmes d'optimisations, de mémoire et de complexité que nous avons dû résoudre. Ce projet a donc été une expérience enrichissante pour nous et a pu confirmer l'interêt de certains outils de développement notamment git et valgrind.