Rapport du projet de C++ avancé

Toufik FERHAT[1]
[1]Master 1 : Ingénierie des systèmes intelligents - Sorbonne Université

559 Mots Numéro d'étudiant : 3872490

Mots clés: C++, POO, Recherche de chemin, Graphe, SDL, Conteneurs, Djikstra

7 juin 2021

Résumé

Ce document a pour but de clarifier les choix faits pour l'implémentation des différentes architectures et la solution aux obstacles rencontrés dans le projet. Le projet en lui-même consiste en l'implémentation d'un city mapper capable de trouver le plus court chemin entre une station de départ et une station d'arrivée à l'aide de l'algorithme de Djikstra.

1 Récupération des informations sur les stations dans une table de hachage

Dans cette partie, nous n'avons pas rencontré beaucoup de problèmes et de bugs, nous avons juste lu la classe Generic Station Parser et utilisé quelques références sur Internet pour lire le fichier et vérifier si la lecture est correcte. Après cela, nous avons récupéré les données séparées par des virgules et les avons ajoutées aux variables de la structure de la station. Nous avons converti l'ID de la station en un nombre entier comme clé et la valeur comme la station pour chaque élément de la table de hachage (Unordered Map) .

L'un des seuls probléme rencontré est l'appel du Generic Station Parser sans le namespace travel.

2 Récupération des données de connexion pour générer une forme de graphe orienté

C'est dans cette partie que nous récupérons les informations de connexion pour chaque station, en d'autres termes, c'est là que nous créons en quelque sorte le graphe nécessaire à l'algorithme de Djikstra.

Nous créons une unorded map où les clés sont l'ID des stations et les valeurs sont des unorded map des stations connectées tel que la clé c'est l'ID et la valeur c'est le cout de transfert en temps.

Ici, l'un des problèmes rencontrés a été l'insertion des éléments. Mais après plus de compréhension de l'architecture et de recherche, nous avons réussi à les insérer sans le insert qui est fournit par la SDL.

3 Mise en œuvre de l'algorithme de Djikstra pour le calcul du plus court chemin

L'algorithme de Djikstra est un algorithme permettant de trouver les plus courts chemins entre les nœuds d'un graphe. Nous utiliserons ici son principe pour trouver le chemin le plus court entre une station de départ et une station d'arrivée.

Nous choisissons d'utiliser des variables pour l'implémentation de l'algorithme comme suit :

— Un nœud pour représenter une station qui sera modélisée comme un std : :pair qui est une sorte de tuple avec seulement deux éléments : le premier est l'ID de la station et le second est le coût associé. Nous avons choisis de garder la même structure que l'énonce pour simplifier.

— Une liste de nœuds formant le chemin le plus court qui sera modéliser par un vecteur de std : :pair. Ici, nous avons choisi d'utiliser la structure std : :Vector car les éléments sont placés dans un espace de stockage contigu et il est donc plus rapide d'insérer ou de récupérer des éléments et même de les traiter que par example un std :Set .
— Une liste des noeuds qui sont en traitement. Ici encore on a choisis le Vecteur pour la même raison que précédemment.
— Une liste de noeuds qui ont été traités
— Un dictionnaire (Table de hachage) tel que chaque clé est un ID de station et sa valeur est son prédécesseur dans le graphe pour garder la trace du chemin. Ici encore utilise une unordered map car c'est l'architecture utilisée depuis le début et qui est plutôt assez accessible.

Nous aurions pu utiliser une std : :priority queu pour améliorer et optimiser la recherche, mais les résultats étaient assez satisfaisants.

L'une des complications était de revenir en arrière à partir du dernier nœud pour former le chemin. L'utilisation du dictionnaire stockant les prédécesseurs et les itérateurs inverses nous a aidé à surmonter ce problème.

4 Améliorations

Une des améliorations a été l'ajout d'une méthode "$choose\_station\_id$" qui vérifie si l'ID d'entrée est correct et qui affiche la station. Si ce n'est pas le cas, l'utilisateur est invité à entrer un autre ID ou à quitter. La méthode elle-même appelle plusieurs fonctions subsidiaires qui gère la validité de l'entrée et compare des chaînes de caractères.