Projet de C++ Avancé: Application RATP

Yang LI

1er mai 2023

1 Introduction

L'objectif principal de ce projet est de fournir aux utilisateurs une solution efficace pour planifier leur itinéraire dans le métro. Pour atteindre cet objectif, j'ai utilisé des structures de données avancées et des algorithmes d'optimisation, tels que l'algorithme de Dijkstra pour le calcul des itinéraires les plus courts. Ce rapport décrit en détail le fonctionnement du code en présentant les principales classes et méthodes utilisées, ainsi que d'illustrer son fonctionnement à travers des exemples d'exécution concrets.

2 Explication du code

Dans ce projet, j'ai créé la classe dérivée MyStationParser et implémenté progressivement les fonctionnalités nécessaires. L'héritage de nouvelles classes de base exigeait l'implémentation de fonctions virtuelles pures pour permettre la compilation. J'ai réussi à remplir toutes les exigences du projet en implémentant diverses fonctions, dont le calcul du chemin le plus court entre les stations et la gestion des erreurs d'entrée. Au lieu de m'appuyer sur l'évaluation par Grade, j'ai utilisé cout pour vérifier la précision des fonctions. Le code exploite les structures de données de la STL de C++, ainsi que la gestion des flux de fichiers et des exceptions. Voici une explication du code :

MyStationParser(const std : :string& _stations,const std : :string& _connections)

Cette fonction est le constructeur de la classe MyStationParser. Elle prend deux arguments, _stations et _connections, qui représentent les chemins vers les deux fichiers contenant les informations sur les stations et les connexions. La fonction appelle les deux méthodes protégées read_stations et read_connections pour lire les informations contenues dans ces fichiers. Cela permet de respecter le principe de l'encapsulation en empêchant les classes externes d'accéder directement aux méthodes read_stations et read_connections.

void read_stations(const std : :string& _filename)

Cette fonction est une fonction privée de la classe MyStationParser, utilisée pour lire le fichier contenant les informations sur les stations. La fonction utilise la classe ifstream pour ouvrir le fichier et en lire le contenu ligne par ligne. Pour chaque ligne, la fonction utilise stringstream pour analyser chaque champ de la ligne et stocke les informations analysées dans une structure Station. Enfin, la fonction stocke chaque objet Station dans un unordered_map, en utilisant l'ID de la station comme clé.

void read_connections(const std : :string& _filename)

Cette fonction privée de MyStationParser lit le fichier des connexions en utilisant une méthode similaire à read_stations. Elle analyse chaque ligne pour obtenir les ID de stations et temps de trajet, puis les stocke dans un unordered_map.

std : :vector<std : :pair<uint64_t, uint64_t» compute_travel(uint64_t _start, uint64_t _end)

Cette fonction est une fonction publique de la classe MyStationParser, utilisée pour calculer le chemin le plus court entre la station de départ _start et la station d'arrivée _end. La fonction utilise l'algorithme de Dijkstra pour calculer le chemin le plus court, en utilisant une priority_queue pour maintenir l'ensemble des stations à traiter et une table de hachage visited_stations pour enregistrer le chemin le plus court de chaque station au point de départ.

std : :vector<std : :pair<uint64_t, uint64_t» compute_and_display_travel(uint64_t _start, uint64_t _end) Cette fonction est également une fonction publique de la classe MyStationParser. Elle a la même fonctionnalité que la fonction compute_travel, mais elle affiche également le chemin le plus court calculé. Plus précisément, la fonction appelle la fonction compute_travel pour calculer le chemin le plus court, puis utilise les tables de hachage stations_hashmap et connections_hashmap pour afficher les détails du chemin.

uint64_t find_station_id_by_name(const std : :string& station_name_and_line)

Cette fonction sert à trouver l'ID de la station en fonction du nom de la station et du nom de la ligne de métro. Plus précisément, la fonction compare le nom de la station et le nom de la ligne de métro entrés avec les champs correspondants de chaque objet Station.

3 Résultat

Dans ma fonction main, j'ai démontré comment utiliser la classe MyStationParser et ses méthodes pour calculer le chemin le plus court entre des stations données. De plus, cela montre comment le programme traite les cas d'entrées invalides (gestion des exceptions).

Dans le "Test 1", nous avons utilisé les identifiants de station pour appeler compute_and_display_travel, calculant et affichant le chemin le plus court entre les stations avec identifiants de 1722 et 2062.

Dans le "Test 2", nous avons testé la gestion des exceptions avec un identifiant inexistant (17) pour appeler compute_and_display_travel.

Dans le "Test 3", nous avons utilisé les noms de station pour appeler compute_and_display_travel, calculant et affichant le chemin le plus court entre "Saint-Lazare,3" et "Bastille,1".

Dans le "Test 4", nous avons testé la gestion des exceptions avec un nom de station mal orthographié ("Saint-Laz") pour appeler compute_and_display_travel. Le résultat est présenté dans la figure 1.

```
============================> Grade 1 <============================
Stations: seems ok
============================> Grade 1 <============================
Stations: seems ok
============================> Grade 2 <============================
Connections: seems ok
Test 1 : compute_and_display_travel avec station id
Best way from Saint-Lazare (line 3) to Bastille (line 1) is:
Walk to Saint-Lazare, line 14 (180 s)
Take line 14(SAINT-LAZARE <-> OLYMPIADES) - Aller
FromSaint-Lazare to Châtelet (372 s)
Walk to Châtelet, line 1 (180 s)
Take line 1(CHATEAU DE VINCENNES <-> LA DEFENSE) - Retour
From Châtelet to Bastille (367 s)

After 1099 secs, you have reached your destination!

Test 2 : gestion d'exception pour l'entrée identifiant de la station
Erreur: La station de départ ou d'arrivée est invalide.
No valid path found.

Test 3 : compute_and_display_travel avec nom de la station
Best way from Saint-Lazare (line 3) to Bastille (line 1) is:
Walk to Saint-Lazare, line 14 (180 s)
Take line 14(SAINT-LAZARE <-> OLYMPIADES) - Aller
FromSaint-Lazare to Châtelet (372 s)
Walk to Châtelet, line 1 (180 s)
Take line 1(CHATEAU DE VINCENNES <-> LA DEFENSE) - Retour
From Châtelet to Bastille (367 s)

After 1099 secs, you have reached your destination!

Test 4 : gestion d'exception pour l'entrée nom de la station
Station name or line not found.
```

FIGURE 1 – Le résultat en exécutant main.cpp

4 Conclusion

En conclusion, ce rapport a fourni une analyse détaillée du planificateur d'itinéraire de métro développé en C++. Nous avons examiné en profondeur les principales classes et méthodes utilisées pour traiter les données des stations et des connexions, calculer les itinéraires les plus courts et afficher les résultats. Les exemples d'exécution ont montré que le code est capable de résoudre efficacement les problèmes de navigation dans les réseaux de transport en commun, en fournissant des résultats précis et facilement compréhensibles.