Rapport de Projet de C++ Avancé

ISI groupe2 : Chenao JIANG (21100538)

I. Introduction

Le but de ce projet est de faire une application qui établit le trajet le plus court entre deux stations du réseau de métro parisien. Il y a déjà un contrat de programmation est écrit, mon objectif est d'implémenter le code de plus en plus complet afin de réaliser le problème du plus court chemin en utilisant l'algorithme de Dijkstra.

Dans les parties suivantes, je vais présenter les problèmes rencontrés et les solutions apportées en suivant la feuille de route, en particulier l'algorithme de Dijkstra.

II. Problèmes et Solutions

1. Généralité

- **namespace** est une région déclarative, il est utilisé pour organiser le code en groupes logiques. Alors le code doit être dans le même namespace, dans mon code j'ai utilisé le nom qualifié complet pour namespace std : *std :: cout*(exemple) et une using pour namespace travel : *using namespace travel*

2. Surcharger les fonctions

- Ma classe ne peut pas être instanciée

J'ai créé une classe qui hérite de la classe mère *Generic_station_parser*, elle ne peut pas être instanciée car la classe *Generic_station_parser* est une classe abstraite. Il faut surcharger la méthode virtuelle pure dans ma classe. (La même problème pour la classe mère *Generic_connection_parser* et *Generic_mapper*).

- La fonction ***insert*** ne peut pas remplacer le couple si la clé existe déjà

Au début, j'ai utilisé la fonction ***insert_or_assign*** pour réaliser mise à jour la map, mais *insert_or_assign* est pour C++17. Alors si on a besoin de mettre à jour la valeur correspondant à une clé existante dans la *unorderedmap*, il faut d'abord supprimer le couple de cette clé(la fonction ***erase***), puis l'ajouter(***insert***).

3. Auto-évaluation

- Utilisation la classe *Grade* et le compilation g++

Dans la fonction principale main, utiliser directement des objets statiques *travel ::evaluate_small* et *travel ::evaluate* dans la classe *Grade*, puis appeler les méthodes de la classe *Grade* pour l'auto-évaluation

Pour exécuter le programme, il faut d'abord utiliser la commande ***g++ -c filename*** pour compiler le fichier et générer le fichier objet, puis il faut utiliser ***g++ -o targetname filename*** pour compiler et lier plusieurs fichiers objets et générer le fichier exécutable avec targetname

- Core dumped

Si une exception est levée, on génère d'abord le fichier **core** (modifie la taille du fichier core, la taille initiale est 0), puis utilise **gdb** pour afficher le fichier et localise les lignes dans le fichier qui ont causé le core dump (les commande : **gdb where, gdb bt**...) pour trouver l'erreur.

4. L'algorithme Dijkstra

- **Initialisation**

1. A partir de la start station, créer deux conteneurs pour enregistrer l'ensemble des stations avec le chemin plus court calculé et l'ensemble des stations non calculées.

J'ai choisi d'utiliser ***unordere_map <uint64_t, uint64_t>*** pour les 2 ensembles **station_yes** et **station_no**.

Pour les stations non connectées à la start station, le cout est initialisé à l'infini ; Pour les stations connectées à la start station, le cout est initialisé à la valeur correspondante des données.

Sur l'exemple simple (A-J: 1-10; start station : 2 ; end station : 7)

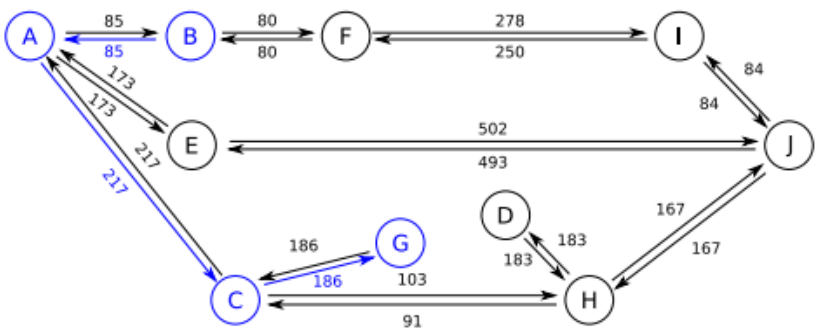


Figure 1: Exemple de graphe orienté

station_yes: {2: 0}

station_no: {1: 85; 3: ∞; 4: ∞; 5: ∞; 6: 80; 7: ∞; 8: ∞; 9: ∞; 10: ∞ }

2. Il a besoin un conteneur supplémentaire appelé **chemins** afin de noter le chemin pour chaque station, j'ai choisi ***unordered_map<uint64_t, vector< pair<uint64_t, uint64_t>>>,*** la clé est l'identifiant de station qui associe le chemin plus court entre la start station et elle-même. La map chemins a le chemin plus court juste pour les station dans la map station_yes, le chemin des autre stations est le chemin plus court précédent. Chaque fois il y a de station ajoutée dans la map station_yes, la map chemins va mettre à jours.

Pour l'exemple simple :

chemins : {1: {2: 0}; 2: {2: 0}; 3: {2: 0}; 4: {2: 0}; 5: {2: 0}; 6: {2: 0}; 7: {2: 0}; 8: {2: 0}; 9: {2: 0}; 10: {2: 0}}

- **Boucle**

3. Il faut rechercher toutes les stations, tous les cas pour trouver le chemin plus court, alors faire une boucle qui passe par toutes les stations sauf la start station.

Chaque cycle, on trouve et supprime la station qui a le cout minimal dans la map **station_no** et l'ajouter dans la map **station_yes**. En même temps, il faut mettre à jours le chemin pour cette station dans la map **chemins**

A ce moment-là, le 1er cycle pour l'exemple simple :

station_yes : {2: 0; 6: 80 }

station_no: {1: 85; 3: ∞; 4: ∞; 5: ∞; 7: ∞; 8: ∞; 9: ∞; 10: ∞ }

chemins : {1: {2: 0}; 2: {2: 0}; 3: {2: 0}; 4: {2: 0}; 5: {2: 0}; 6: {2: 0; 6: 80}; 7: {2: 0}; 8: {2: 0}; 9: {2: 0}; 10: {2: 0}}

4. Ensuite, dans le même cycle, il faut trouver les stations connectées avec la station déplacée (avec min cout), et mettre à jours leur cout dans la map **station_no**

A ce moment-là, le 1er cycle pour l'exemple simple :

station_yes : {2: 0; 6: 80 }

station_no: {1: 85; 3: ∞; 4: ∞; 5: ∞; 7: ∞; 8: ∞; 9: 358; 10: ∞ }

5. Puis il faut mettre à jours le **chemin** pour les stations connectées

chemins : {1: {2: 0}; 2: {2: 0}; 3: {2: 0}; 4: {2: 0}; 5: {2: 0}; 6: {2: 0; 6: 80}; 7: {2: 0}; 8: {2: 0}; 9: {2: 0; 6: 80}; 10: {2: 0}}

Fin de cycle

Après le 2nd cycle pour l'exemple simple :

station_yes : {2: 0 ; 6: 80 ; 1: 85}

station_no: {3: 302; 4:∞; 5: 258; 7: ∞; 8: ∞; 9: 358; 10: ∞ }

chemins : {1: {2: 0; 1: 85}; 2:{2:0}; 3: {2: 0; 1: 85}; 4:{2:0}; 5: {2: 0; 1: 85}; 6:{2:0; 6: 80}; 7: {2: 0}; 8: {2: 0}; 9: {2: 0}; 10: {2: 0} }

Après le 3e cycle pour l'exemple simple :

station_yes : {2: 0 ; 6: 80 ; 1: 85 ; 5: 258}

station_no: {3: 302; 4:∞; 7: ∞; 8: ∞; 9: 358; 10: 751 }

chemins : {1: {2: 0; 1: 85}; 2:{2:0}; 3: {2: 0; 1: 85}; 4:{2:0}; 5: {2: 0; 1: 85; 5: 258}; 6:{2:0; 6: 80}; 7: {2: 0}; 8: {2: 0}; 9: {2: 0}; 10: {2: 0; 1: 85; 5: 258} }

A la fin de boucle, dans la map **chemins** on peut directement trouver le chemin plus court associé à la station end.

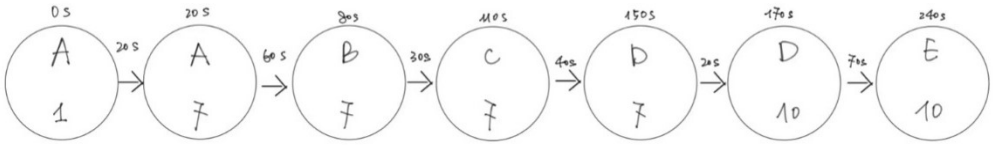
5. Afficher le chemin

Dans la fonction ***compute_and_display_travel***, on peut obtenir le chemin plus court par la fonction ***compute_travel***, et j'ai utilisé **2 itérateurs** comme deux pointeurs pour bien trouver les stations qui sont la même ligne ou même nom, alors il peut afficher le chemin plus clair.

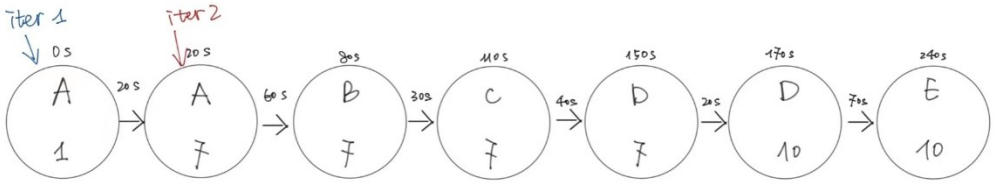
Si les deux stations sont le même nom et afficher l'aller à pied ; Mettre les stations avec la même ligne ensemble et l'afficher une seule fois.

Exemple :

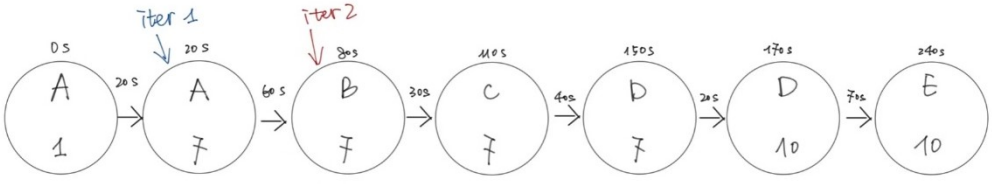
Le chemin plus court: (Station name and line id)



Les 2 itérateurs :



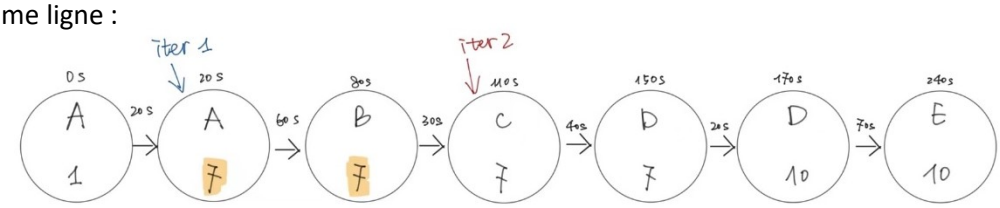
Le même nom :



iter1.name = iter2.name Walk to station A (line 7) (20s)

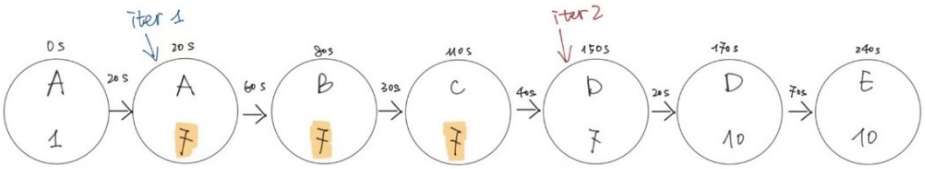
iter1++ ; iter2++ ;

La même ligne :



iter1.line_id = iter2.line_id

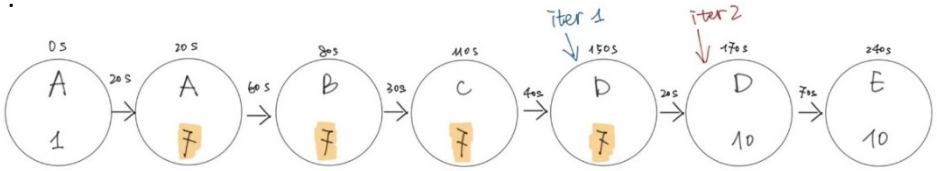
iter2++ ;



iter1 . line_id = iter2. line_id

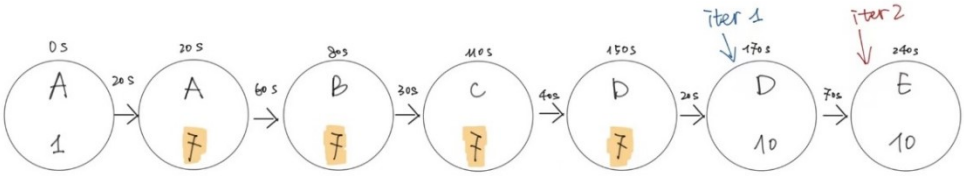
iter2++;

Ensuit :



iter1 = iter2 - 1;

Take line 7
From A to D (130s) 3 stations
(150 − 20 = 130)



iter 2 = end - 1

Boucle Fini.

Take line 10
From D to E (70s) 1 station.

III. Amélioration

Surcharger la fonction ***compute_and_display_travel***, ajouter une étape de convertir le nom de station à l'id de station et les restes sont le même.

S'il ne peut pas trouver la station avec le nom entrée, ***throw*** une erreur ; dans le programme principal ***catch*** les erreurs et demander réessayer.

Utiliser la fonction ***transform*** *(include<algorithme>)* afin de convertir le nom de station en minuscule, alors il est convenable d'être résistant aux erreurs de casse lors de la saisie des noms par l'utilisateur.

IV. Conclusion

Ce projet me permet de mieux comprendre divers containers, tels que map, vector, pair etc. Après ce projet, je peux bien utiliser l'itérateur et les algorithmes communs (find, insert, erase…).

En comprenant les principes de l'algorithme de Dijkstra, puis en utilisant ces outils standards, j'ai implémenté l'algorithme afin de trouver le chemin plus court. Ensuite, j'ai utilisé des itérateurs pour afficher le chemin le plus court de manière plus claire et concise.

A la fin, j'ai optimisé le programme en surchargeant la fonction pour rendre l'utilisateur plus pratique, mais actuellement seules les optimisations insensibles à la casse sont réalisées.