POLYTECH LILLE

Université de Lille

Rapport de projet tutoré PA-SE3 S6

Une application pour analyser des vols aériens internes aux Etats-Unis en 2014.

Réalisé par:
MAZOUZI Janet et YASMINE Chaimae
encadré par :
RUDAMETKIN Walter

Année Universitaire : 2020-2021

projet Tutorat PA

SOMMAIRE

1. Introduction

1.1. Contexte

Nous avons à notre disposition trois fichiers CSV traitant du le trafic aérien interne aux Etats-Unis en 2014:

- airlines.csv : donne des informations (IATA_CODE et nom) sur une quinzaine de compagnies aériennes étasuniennes.
- airports.csv : renseigne des informations sur 322 aéroports du pays. Les informations incluent le code IATA des aéroports, leurs noms et leurs localisations.
- flights.csv : contient les informations sur 58 592 vols des compagnies de airlines.csv, au départ et à l'arrivée des aéroports de airports.csv. Les informations incluent le départ et l'arrivée de ces avions (date,heure, retard,...), les annulations ou déviations .

1.2. Objectifs

Ce projet a pour but de mettre en pratique nos acquis de Programmation Avancée : lecture de fichiers, création de structures de données, de tables de hachage,...

Le projet se constitue de deux étapes:

On doit d'abord réaliser un programme qui charge les fichiers de données CSV dans des structures de données de notre choix, permettant de les interroger facilement. Les structures doivent être optimisées au mieux afin de réduire au plus la complexité des requêtes.

Ensuite, notre application doit être capable de répondre à plusieurs requêtes demandées par l'utilisateur:

- tous les aéroports depuis lesquels la compagnie aérienne <airline id> opère des vols.
- afficher les compagnies aériens qui ont des vols qui partent de l'aéroport <port_id>
- afficher les vols qui partent de l'aéroport à la date, avec optionnellement une heure de début, et limité à xx vols
- donner les 5 vols qui ont subi les plus longs retards à l'arrivée et donner les 5 compagnies aériennes qui ont, en moyenne, le plus de retards.
- donner les 3 compagnies aériennes avec le plus de retard pour arriver à l'aéroport passée en paramètre.
- les vols annulés ou déviés à la date (format M-D)
- calculer le temps de vol moyen entre deux aéroports
- trouver un ou plusieurs itinéraires entre deux aeroports à une date donnée
- trouver un itinéraire multiville qui permet de visiter plusieurs villes

Pour finir, cette application doit aussi avoir une requête qui va permettre à l'utilisateur de sortir de l'application en désallouant toute la mémoire utilisée.

Nous allons maintenant vous présenter la solution que nous avons trouvée.

2. Structures de données(SD)

Dans cette partie nous allons expliciter certaines parties de nos SD.
Afin de gagner du temps nous nous sommes permis de créer plusieurs structures de données basiques tels que la structure date et horaire et autres structures qui regroupent le plus d'informations sur le vol, la compagnie aérienne ainsi que l'aéroport.

2.1. La SD flight

La structure Flight est le nœud d'un arbre. Il y a autant d'arbres que de jours représentés dans l'ensemble des vols (pas de mois d'octobre) . L' arbre est trié par aéroport de départ et comporte toutes les informations sur le vol que nous donne le fichier CSV.

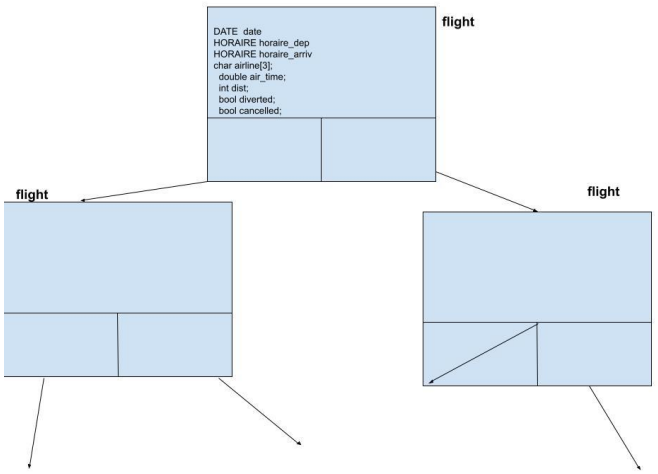

Fig 1:Structure de données flight trier par dep airp

Tous les SD sont définis de la même manière et on peut les retrouver dans le fichier includes/fonctions.h.

2.2. La SD hash_cell_fl

On choisit d'utiliser des tables de hachage principalement pour leur complexité et efficacité. De plus, au vu du nombre significatif de données que l'on doit traiter, les tables de hachages nous évitent de perdre des données et facilitent la programmation des commandes. Alors on initialise un tableau de listes chaînées nommé hash_cell_fl qui contient la date de vol comme clé et un pointeur sur notre arbre flight. On peut visualiser notre SD dans le schéma suivant :

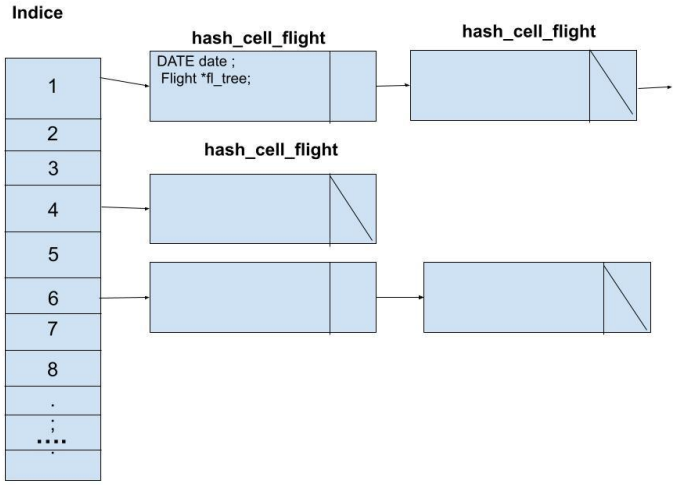

Fig 2: table de hachage des vols

2.3. La SD Airport et airl_infos

Cette SD est construite à partir des données du fichier airports.csv.

De la même manière, on a défini une table de hachage d'aéroports qui contient les informations générales sur l'aéroport en question. De plus, cette structure pointe vers un arbre de airl_infos. Le schéma suivant illustre la table de hachage:

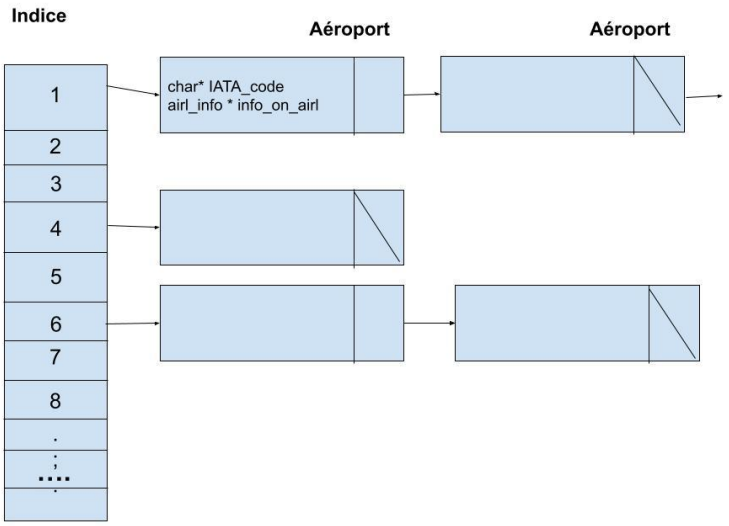

Fig 3 :table de hachage des aéroports

L'arbre de airl_infos est trié par code IATA de compagnies aériennes. Il se remplit avec le balayage du fichier des vols. Elle regroupe les données de toutes les airlines travaillant dans l'aéroport en question:

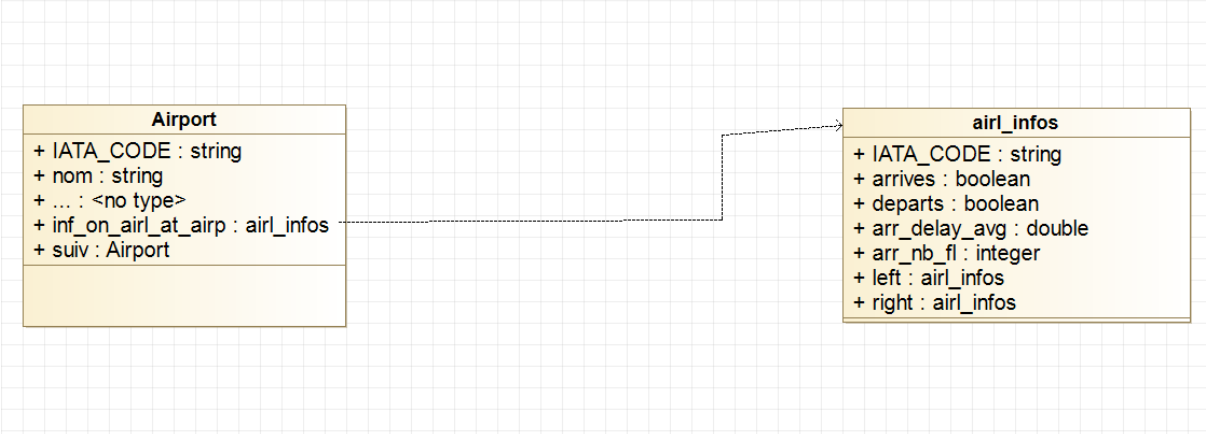

Fig 4 : Schéma de la structures de données airl_infos et sa dépendance à Airport

A titre d'exemple, si l'on rencontre un vol au départ de l'aéroport XXX de la compagnie XX, nous allons à la case de l'aéroport XXX, avant de parcourir l'arbre inf_on_airl_at_airp à la recherche de XX. On y change alors l'attribut booléen *departs* en 1.

2.4. La SD Airlines

Cette SD est un simple nœud d'arbre dans lequel on a le code IATA, le nom, le nombre de vol et le retard moyen de chaque compagnie aérienne. Il est trié par code IATA.

Le retard moyen ainsi que le nombre de vols sont actualisés avec le parcours de tous les vols.

3. Déroulement de l'algorithme:

La première étape du programme est la récupération des données afin de remplir les différentes structures ainsi que certains tableaux pour les retards.

Ensuite, l'application attend que l'utilisateur rentre une requête avec ses arguments. Il faut que ce dernier respecte la syntaxe et l'ordre des arguments comme indiqué dans le README.md.

Maintenant que l'on a les outils nécessaires pour traiter les vols on peut construire nos requêtes.

Exemple la fonction show airports:

Le but : cette fonction permet d'afficher tous les aéroports depuis lesquels la compagnie aérienne opère des vols.

Idée: pour ***airline id*** donnée on parcourt toutes les cases de notre table de hachage de aéroports. Pour chaque aéroport, on parcourt inf_on_airl_at_airp à la recherche de ***airline id***.

Astuces : l' astuce est d'utiliser une fonction intermédiaire qui s'occupe de rechercher un nœud airl_infos dans un ABR d'airl_infos.
De la même manière on a traité les différentes requêtes toujours en passant par des fonctions intermédiaires qui vont faciliter l'affichage des requêtes voulues.

Seule la dernière requête ne fonctionne qu'à moitié : l'application n'affiche que le premier trajet.

Voici quelques précisions:

- Les retards sont calculés comme suit : (somme(delays) pour tous les vols concernés)/(nb total de vols concernés).
- Le nombre d'escale pour find-itinerary et find-multicity-itinerary est limité à 2.
- La durée maximale d'une escale est de 10h.

4. Analyse critique du résultat:

Dans ce projet, le choix de structure de données a un impact énorme sur l'efficacité et la rapidité des requêtes.

Nous avons passé beaucoup de temps sur l'élaboration de ces structures. Elles ne sont pas optimales pour toutes les requêtes mais nous en sommes satisfaites. A titre d'exemple,pour les vols, le tri par date est très efficace. La fonction de hachage que nous avons trouvé nous permet d'avoir au maximum deux collisions sur une case de la table. Cela facilite l'exécution des deux dernières requêtes, bien complexes.

Nous avons fait le choix d'utiliser des arbres mais nous aurions pu aussi utiliser les listes doublement chaînées, ou alors des tables de hachage imbriquées.

Une amélioration nécessaire serait sur la lecture des requêtes. Le programme est encore parfois confus sur la lecture en utilisant strtok. Une solution serait l'utilisation de strsep.

De plus, pour accélérer l'exécution des deux dernières requêtes, il serait intéressant de trouver un moyen de trier les vols par heure de départs. Cela permettrait de ne pas avoir à traverser l'arbre entier à chaque recherche d'arrêt.

Le programme ne procure pas de Warning et passe le test Valgrind.

5. Conclusion

Finalement, nous avons réussi à faire fonctionner 10/11 requêtes. Nous avons utilisé le plus de ressources possible afin d'optimiser au mieux le stockage ainsi que la vitesse d'exécution.

En plus d'être un projet pédagogique il est aussi ludique et nous a donné beaucoup de liberté dans le code et dans la conception.

Pour la suite, nous aimerions incorporer des éléments plus complexes comme les pointeurs de fonctions afin de réduire les répétitions de code, très présentes dans notre programme.