Florian DERLIQUE et Corentin GIELEN SE3 S6

Rapport du Projet de Programmation Avancée - Réalisation d'une application d'analyse de vols

Introduction

Lors de ce tutorat nous avons eu à réaliser une application qui permet d'analyser 58592 vols aux États Unis en 2014 et effectuer différents traitements. Grâce à des requêtes que l'utilisateur pourra renseigner via un terminal.
Pour cela nous devions charger les 3 fichiers fichiers fournis et stocker les données dans une base de données pour ensuite pouvoir les traiter l'intérêt d'utiliser une base de données et de limiter le nombre de chargement pour rendre le programme plus rapide et ne pas dépendre par la suite des fichiers.

Cahier des charges :

1. Charger les données des 3 fichiers CSV dans des structures de données jugées pertinentes.
2. Attendre une commande renseignée par l'utilisateur.
3. Traiter la commande.
4. Afficher le résultat de cette commande.
5. Revenir à l'étape 2.

I Nos choix de structures de données

Dès le départ nous voulions une structure de donnée à la fois rapide et prenant un minimum de place quitte à être plus difficile à coder.

Ensuite pour réaliser notre structures de données à l'agence de la manière la plus optimal nous avons analysé les différentes fonction qu'on nous demander de produire par mots clefs comme ce ci :

show-airports : compagnie aérienne -> aéroports
show-airlines : aéroport -> compagnie aérienne
show-flights : aéroport -> date
most-delayed-flights : retard
most-delayed-airlines : compagnie aérienne -> retard
delayed-airline : compagnie aérienne -> retard
most-delayed-airlines-at-airport : aéroport -> compagnie aérienne-> compagnie
changed-flights : date
avg-flight-duration : aéroport

find-itinerary : aéroport -> date
find-multicity-itinerary : /

Nous avons constaté que les mots clefs qui reviennent le plus souvent son aéroport,compagnie aérienne et date. Partant de ce constat et sur nos connaissances en base de données acquises au semestre derniers et sur notre volonté de faire la base de données la plus optimisée possible .

Nous avons eu l'idée de faire des tables de hachages interconnectées entre elles. ***illustration 1.*** C'est à dire une table principale ou chaque cellule pointe vers une table secondaire unique par cellule, et chaque cellule de la table secondaire pointe vers une dernière table . Pour limiter un maximum la duplication des données et l'espace mémoire occupé nous avons regarder le nombre possible total d'aéroports et de compagnies soit pour les aéroports 26*26*26 = 17576 et 36*36=1296 . Donc nous savions que la table de hache des aéroports serait la table avec potentiellement le plus d'éléments ce sera ainsi notre table principale.

Sachant qu'il n'y a que 3000 aéroports dans le monde et beaucoup moins dans notre cas nous avons décidé de limiter notre table de Hache des aéroports à 375 toutes les collisions seront géré à l'aide de liste chaînée. Nous avons donc dans chaque cellule de la table de Hachage des aéroports une liste chaînée où chaque cellule contient les données sur l'aéroport , un pointeur vers une table de hachage de compagnie unique et un pointeur vers l'aéroport suivant .

Pour la seconde table de Hachage pour limiter la duplication nous avons limité la table à 70 cellules où chaque cellule continent uniquement l'IATA de la compagnie (en entier) un pointeur vers une table de Hachage de date unique et un pointeur vers la compagnie suivante. Nous avons ensuite stocker les donnés de la compagnie notamment son nom dans une table de hachage externe à adressage direct (sans liste chaînée taille = 1296) en effet dans le pire des cas on à 17576 aéroport * 1296 compagnies duplication de nos données compagnies ce qui est conséquent donc chaque octet économiser par cellules et extrêmement bénéfique .

Enfin nous avons une table de hachage de date que nous avons pour les mêmes raisons que précédemment la plus petite possible. c'est pour cela que nous avons choisi une table de longueur 12 représentant le nombre de mois ou chaque cellule est une liste chaînée contenant les vols associer à l'aéroport la compagnie et le mois .

Bien que légèrement complexe notre structures de données pourrait supporter un total de 17576 aéroports et 1296 compagnies sans changement ou plus avec la modification des variables globals maxH. De plus pour limiter la place prise par nos tables de H . Nous avons fait en sorte que chaque table soit initialisée uniquement si les données précédentes existent. Exemple si nous avons un aéroport où il y a une seule compagnie alors seul la table de H des date associer à cette compagnie sera créé de même toute les cellules de la table aéroport vide n'aurons pas de table de H compagnie puis date lier à créer.

Notre structures de données peut donc être imagée par l'illustration ci dessous :

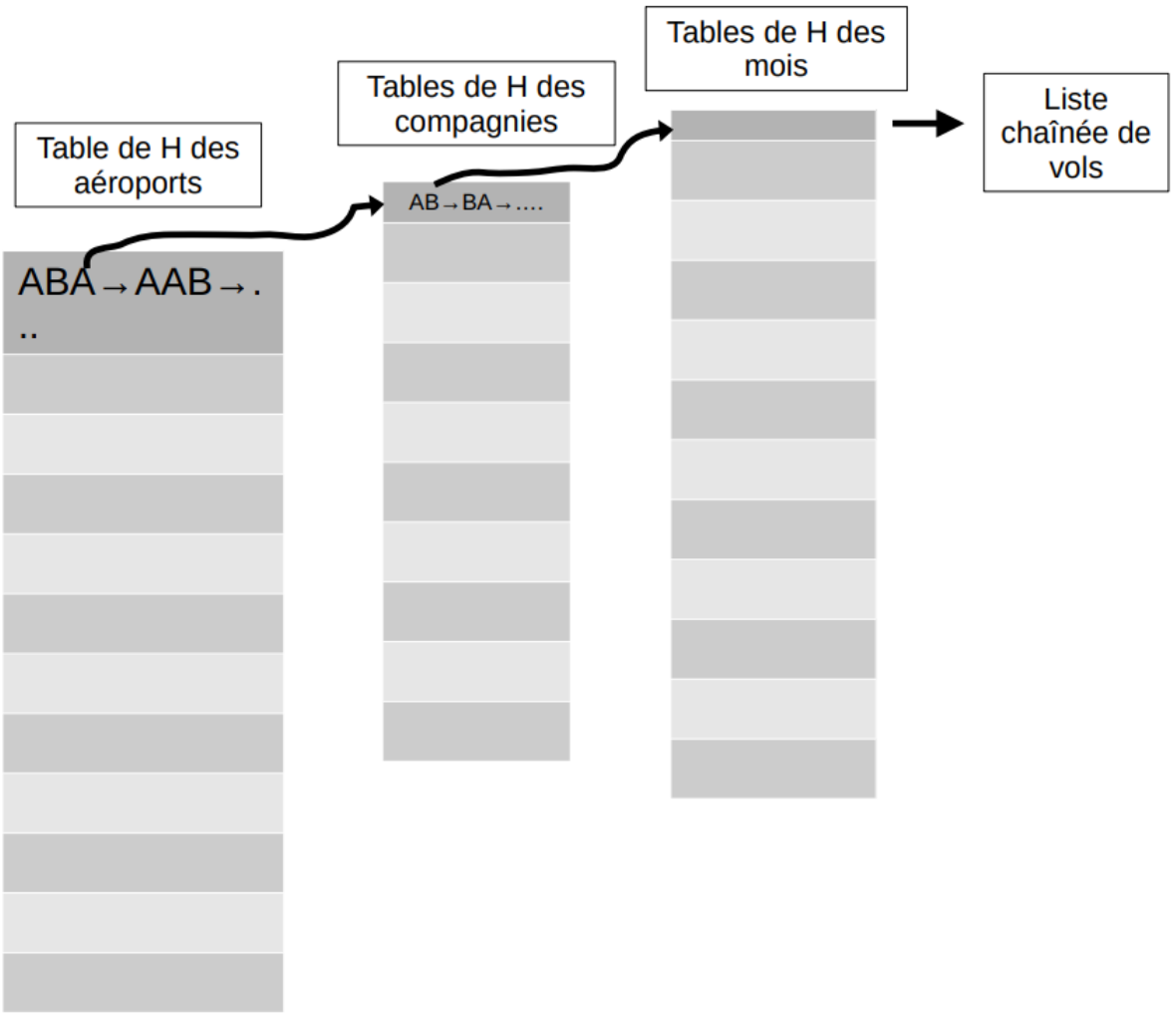

illustration 1

II Nos choix de conception algorithmique

Les requêtes que nous avons réalisées sont principalement toutes basées sur un même algorithme de recherche qui parcourt la TableH des aéroport->la liste chaînée des aéroport -> la Table des compagnies -> la liste chaînée des compagnies -> la TableH des dates puis la liste chaînée des vols . En suite au centre de l'ensemble des boucles nous trouvons généralement une fonction qui est spécifique à chaque requête .

En fonction de la requête nous faisons des accès direct ou un parcours de toutes les tables et nous accédons ou non aux trois tables.

Pour charger nos données, nous avons utilisé des fonctions de calculs d'index de table de hachage puis de recherche de cellules. Si les fonctions de recherches de cellules renvoie NULL, c'est qu'on n'a pas initialisé la table pour ranger le vol, dans ce cas nous effectuons un ajout en tête d'une cellule contenant la suivante, une valeur (soit l'aéroport ou l'airline) et une table de hachage. Ainsi nous pouvons faire un ajout trié par date dans le bon index de table.

Pour l'interface machine, pour l'utilisateur, nous utilisons getline() pour demander les lignes de commandes stdin, c'est assez pratique pour passer de la méthode automatique avec la redirection ou manuelle (sans redirection)

III Fonctionnement de notre programme et temps d'exécution

Dans un premier temps, nous récupérons la ligne de commande, nous séparons les mots à chaque espace de cette ligne et nous la stockons dans une liste contiguë de mots.

Ensuite nous appelons une fonction pour lancer la fonction de la ligne de commande, pour ce faire, nous faisons une recherche dichotomique sur tous les noms de fonctions possibles pour avoir un index, et ainsi utiliser switch case grâce à la l'index obtenu. Selon la fonction, nous lançons une fonction d'initialisation, qui permet en d'autres :

- De convertir les données
- De tester si l'utilisateur ne s'est pas trompé à l'écriture, pour cela nous avons une multitude de fonctions de check mise à disposition. Ceci est pour éviter de lancer la commande avec des faux paramètres.

- De gérer les paramètres optionnels à la création d'un masque si besoin. En effet, nous gérons les paramètres optionnels grâce à un masque dont les bits prennent la valeur 1 si le paramètre est actif, 0 sinon.
-

Nous avons testé notre programme en lançant toutes les fonctions possibles + chargements des données. Notre programme s'exécute en environ 100ms , 100ms le temps pour charger toutes les données triées classées par mois/airport/airlines, puis triées par jour !

IV Limite de nos algorithme et de notre main

Nous avons testé toutes nos fonctions avec les données que nous avions à disposition. Mais nous n'avons pas testé avec des données dans un ordre différents il se peut que certaines fonctions soient des bugs notamment des segmentation fault bien que normalement non. De plus, nos algorithmes sont basés sur notre structures de données et les fonctions de chargement de fichier sont spécifiques à la forme des données que l'on traite . Si la forme des données est amenée à changer, notre solution ne fonctionnera plus sans modification pouvant être lourde .

Pour finir nous avons tenté de limiter au maximum les erreurs d'utilisation lorsque l'utilisateur entre ces requêtes par ailleurs certain cas / action nous on peut être échappée.

IV CONCLUSION

Ainsi nous avons réussi à réaliser le travail demandé lors de ce projet . À travers celui si nous avons pu revoir toutes les notions que nous avons vu durant cette année scolaire comme les structures de données, les boucles , les pointeurs, les différents fichiers , makefile … et des notions supplémentaires lors de nos recherches personnelles notamment des fonctions spécifiques.

Nous avons réalisé la plupart des requêtes demandées dans le cahier des charges ainsi qu'une interface avec l'utilisateur et le reste du cahier des charges. Tout en essayant de réaliser la solution la plus optimale en termes de rapidité d'exécution et de mémoire . Dans la limite de nos connaissances et capacités.