Projet : S1.02

Comparaison d'approches algorithmiques

Rouillon Tom

Welty Alexandre

Année universitaire 2021/2022

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges

Table des matières

I. Introduction

Le but de cet SAÉ est de charger une base de données grâce à différents algorithmes afin de la questionner. La base de données en question est un fichier csv créé par l'INSEE, listant tous les prénoms des enfants nés en France entre 1900 et 2020.

Dans ce compte rendu nous explorerons donc les questions que nous nous sommes posés, notamment sur la structure de donnée à utiliser ainsi que sur la potentielle simplification de la base de données, et nous exposerons ensuite les différentes fonctionnalitées de notre programme mais aussi de ses limites.

II. Analyse du problème et solutions

Comme dit précédemment, l'une des questions les plus importantes à se poser avant même de commencer l'écriture du programme est la structure de donnée à utiliser.

Sur le site de l'INSEE nous pouvons retrouver la documentation pour le fichier que nous utilisons. Elle permet ainsi de nous indiquer les informations essentielles, comme la façon dont est trié le fichier, mais surtout le type des variables ainsi que leur longueur maximale.

Ici, nous voyons que les variables sexe, preusuel, annais et dpt sont des caractères, ce qui veut dire que dans le programme, ces variables-ci devront être stockées en tant que caractères. Seule exception : le nombre de naissances, qui lui peut être stocké en tant qu'entier directement.

Cependant, pour l'année de naissance, il est plus simple et plus rapide de comparer deux entiers que deux chaînes de caractères. Mais l'une des spécifications de cette variable est que si l'année de naissance n'est pas connue, alors la valeur est la chaîne de caractère "XXXX". Ainsi, il ne sera pas possible de la transformer en entier pendant le stockage des données, mais on pourra le faire plus tard, notamment dans la fonction permettant d'avoir les statistiques sur un prénom donné.

Ensuite, en analysant bien ce qu'il nous est demandé, on remarque qu'à aucun moment le département de naissance n'apparaît. Nous pouvons donc simplifier cette base de données en supprimant la variable dpt. Cela nous fera gagner de la place en mémoire, mais aussi un peu de temps d'exécution.

Enfin, après avoir réfléchi à la structure de donnée pour un élément, il faut réfléchir à la structure de donnée pour stocker tous ces éléments. Le critère qui semble le plus pertinent est la performance de l'accès à l'information demandée, en termes de temps de calcul. Ici, on remarque qu'on ne fait qu'accéder à l'information, on ne la supprime pas et on n'en rajoute pas.

Ainsi, on peut d'abord penser à un tableau, où l'accès se fait en O(1), ce qui est donc intéressant. Le seul problème est de savoir combien de place en mémoire on lui réserve. Une première solution envisageable serait de lire le fichier en entier en comptant juste le nombre de ligne et en stockant le résultat dans une variable. L'inconvénient principale est que cela nous pousse à lire entièrement le fichier 2 fois, ce qui pose problème si le fichier contient beaucoup de lignes, ce qui est le cas de ce fichier qui en compte plus de 3 700 000.

Une autre solution serait de stocker les données dans une liste chaînée puisque l'on peut rajouter autant de cellule que l'on veut sans savoir la taille du fichier au préalable. Le problème est que l'accès à l'information se fait en O(n), ce qui ne nous intéresse pas vraiment.

Enfin, une autre solution est de regarder la documentation du fichier sur le site de l'INSEE, qui nous donne le nombre d'enregistrement. Le problème est que si l'on veut lire un autre fichier, il faudra changer la variable en fonction du nombre de ligne donné dans la documentation. C'est cette solution que l'on a retenue, car c'est celle qui nous semblait la moins problématique.

Ainsi, nous avons donc répondu aux principales questions concernant les structures de données à employer.

III. Les fonctionnalités et les limites du programmes

Programme table.c :

La première fonction dont on aura besoin est le stockage des données d'une ligne dans un tableau. Ce qui est intéressant est que le fichier est au format csv (comma-separated values), ce qui veut dire que chaque information est séparée par un point-virgule. On peut donc utiliser la fonction strtok, permettant de séparer une chaîne de caractère en fonction d'un autre caractère, qui sera dans notre cas le point-virgule. Ensuite, on stockera chaque information dans le bon champ de la structure, puis on renverra une variable du type de notre structure.

La deuxième fonction est celle qui s'occupera de lire chaque ligne du fichier et de les stocker dans le tableau. On utilisera donc la fonction d'avant, ainsi que la fonction fgets, permettant de récupérer une ligne dans un fichier.

Pour les fonctionnalités accessibles à l'utilisateur, on a :

L'affichage du menu, qui affiche simplement les fonctionnalités disponibles

Le nombre de naissances, qui affiche le nombre total de naissances de 1900 à 2020. Une manière de la faire est pendant la lecture du fichier et le stockage des données. En effet, il faut juste initialiser une variable, qui sera incrémentée à chaque passage de ligne par le nombre de naissance contenu dans la variable 'nombre' du fichier. Ainsi, il n'y aura pas besoin de relire entièrement le tableau à chaque fois que l'utilisateur demandera le nombre de naissance, mais il suffira d'afficher la variable.

Le nombre de prénoms, qui affiche le nombre de prénoms différents. On demande d'abord si on veut distinguer le genre ou non. Si on distingue le genre, on comptera deux fois le prénom s'il est masculin et féminin, mais il faudra le compter qu'une fois si on ne distingue pas le genre.

On utilisera donc 2 fonctions : une si on distingue le genre, l'autre si on ne distingue pas le genre. La 1ère est la plus simple puisqu'il suffit de regarder le genre puis d'incrémenter de 1 la variable 'boys' s'il vaut 1 et d'incrémenter de 1 la variable 'girls' s'il vaut 2. Ensuite, on crée une boucle permettant d'aller jusqu'au prénom différent suivant. Enfin, on stocke les résultats dans un autre tableau, ce qui nous permet de lancer qu'une seule fois cette fonction.

Pour la 2ème fonction, on stockera chaque nom dans un tableau temporaire pour pouvoir comparer le nouveau nom avec tous ceux déjà présents. S'il est déjà dans le tableau temporaire, alors on le passe, sinon on le stocke dans ce tableau et on incrémente une variable contenant le nombre total. Ensuite, on fait comme dans la 1ère fonction et on boucle jusqu'au prochain prénom différent.

Enfin, on stocke le résultat final dans le même tableau que dans la 1ère fonction pour ne pas avoir à relancer la fonction à chaque fois que l'utilisateur demande le nombre de prénoms.

Pour tester cette fonction, nous avons utilisé un petit fichier csv nommé Test1. Nous y avons placé quelques exemples, et après avoir lancé les fonctions précédentes, les résultats étaient bons. En effet, si on distingue le genre, on trouve 2 prénoms masculins et 3 féminins, et si on ne distingue pas le genre, on trouve 4 prénoms, ce qui est exactement ce qu'on attend. On peut donc être confiant des résultats avec un plus gros fichier.

Les statistiques sur un prénom, permet de donner le nombre de naissance pour un prénom donné puis de donner l'année de première et de dernière apparition.

Pour cela, on aura besoin de 2 fonctions, une si on distingue le genre et une autre si on ne veut pas.

La 1ère est si l'on distingue le genre. On compare ensuite le prénom donné par l'utilisateur et le prénom stocké dans le tableau, puis si ce sont les mêmes, on regarde si c'est une fille ou un garçon et on incrémente une des deux variables suivantes : 'boys' ou 'girls'.

Pour avoir l'année de première et de dernière apparition, on initialise deux variables min et max à 1900 et 2020, et on regarde pour chaque prénom si l'année de naissance est inférieure ou supérieure aux deux variables min et max, et on change ces variables le cas échéant.

Cependant, un problème se pose : si un prénom n'est présent qu'une fois et qu'on ne connaît pas l'année de naissance, la fonction affichera 1900 en année de première apparition et 2020 en année de dernière apparition. C'est pour cela qu'on initialise une variable 'change' à 0, et on la change à 1 si on ne trouve pas d'autre année.

Donc, si 'change' vaut 0, cela veut dire qu'on ne connaît les années de première et dernière apparition.

La 2ème fonction reprend le même principe, à la seule différence que l'on ne regarde pas si le prénom est féminin ou masculin.

Une des limites de ces fonctions est que les prénoms sont stockés en majuscule dans notre tableau, ce qui veut dire que quand l'utilisateur rentre un prénom, il faut le mettre en majuscule. Cependant, les fonctions présentes dans les librairies de base ne permettent pas de gérer les caractères spéciaux tels que le c-cédille ou les accents circonflexes, etc…

L'une des solutions serait de demander à l'utilisateur de taper les prénoms en majuscules, ce qui n'est pas très pratique, ou alors d'importer une bibliothèque permettant de gérer les caractères spéciaux.

Programme listeChained.c

C'est notre première tentative ainsi il y a quelques bugs que nous n'avons pas corrigé dans ce programme (car nous voulions essayer avec une autre structure de donnée) notamment :

-dans la structure Person la variable char dptOfBirth[3] n'est pas utile

-Nombre de naissance : le programme compte le nombre de lignes et non le nombre de naissance.

Fonctions stats : le maximum et le minimum ne fonctionne pas avec tous les prénoms

L'idée était de mettre en pratique les différentes notions vues en cours, nous avons donc essayer de résoudre le problème en utilisant une liste chainée. De plus cela nous permet d'effectuer une comparaison avec le programme table.c

On retrouve une structure person de type Person et une structure node de type Node. La structure node contient 2 types de données : une donnée de type Person ainsi qu'un pointeur donnant accès à l'élément suivant.

Il y a ensuite 3 fonctions qui n'existent pas dans table.c qui permettent respectivement de créer et renvoyer une liste de nœud, d'insérer des données dans la liste et de libérer l'espace dynamiquement alloué.

Le principe des fonctions reste les mêmes que dans le programme table.c, ils sont cependant adaptés afin de pouvoir utiliser le principe des listes chainées.

```
Float temps;
Clock_t t1,t2;
T1=clock();
(programme)
T2=clock();
```

Le temps d'exécution se calcule par la différence entre t2 et t1.

Sans surprise le programme listeChained.c est plus lent que le programme table.c.

Pour fNames la différence est de 0.3s lors de la première exécution uniquement.

Pour stats la différence est de 0.02s à chaque exécution

IV. La répartition du travail :

Pour la répartition du travail, nous avons tous les deux travaillé sur les questions concernant la structure de données des éléments atomiques ainsi que sur la structure de donnée de l'ensemble.

Ensuite, c'est Alexandre qui a écrit les fonctions algorithmiques et c'est Tom qui a écrit les fonctions graphiques.

Puis, nous avons tous les deux travaillé sur la résolution des bogues, ainsi que sur les optimisations des fonctions.

Enfin, c'est Tom qui a rédigé le compte-rendu.