

Algorithmique et structure de données

Projet

Guillaume Magniadas

1 Problematique

Le but de ce projet est d'utiliser 3 méthodes différentes pour stocker le nombre de fois qu'ils apparaissent dans une certaine collection de textes.

Les deux premières méthodes sont l'unordered map (Bibliothèque standard) et une méthode de trie utilisant la bibliothèque standard.

La dernière méthode est aussi un trie mais cette fois-ci avec notre propre implémentation.

2 La lecture des fichiers

Pour lire la suite de fichiers textes qui nous est fourni avec le sujet du projet, j'utiliserais la méthode ifstream de la bibliothèque fstream.

Pour faciliter la lecture de tous les fichiers, je me suis servi d'un array de string contenant le nom de chacun des fichiers que j'ai enregistré dans le fichier "liste.hpp".

Ici, les majuscules ne sont pas prises en compte et les caractères autre que des lettres sont ignorés. On se sert d'un string pour ajouter les caractères consécutifs et à chaque fois qu'un caractère autre qu'une lettre est rencontré, on incrémente le compteur de ce mot dans notre structure et on clear le string.

A la fin quand tous les fichiers ont fini d'être traités, on affiche le temps mis (Pour cela on utilise la méthode system_clock de la bibliothèque "chrono" que l'on appelle une fois avant et une fois après la lecture des fichiers et on fait la différence entre les deux) mais on peut aussi soustraire le temps mis à lire tous les fichiers. (En faisant le test en lisant les fichiers seulement, par exemple, ma machine met environ 16 000 ms) ce que l'on fera pour toutes les méthodes vues dans ce rapport.

3 Unordered Map

Il n'y a pas beaucoup de choses à dire sur ce premier exercice, je me suis servi de l'`unordered_map` (`unordered_map<std::string, int>`) de la bibliothèque du même nom. Son utilisation est simple, j'utilise l'opérateur croché en lui donnant le mot et j'incrmente de 1 ce qu'il me retourne.

Le temps obtenu est très correct (environ 16 000 ms).

3.1 Exemple de résultat

```
[magnat@Pc-Magnat gutenbergs-books]$ g++ -O3 Projet_Hmap_toutfichier.cpp
[magnat@Pc-Magnat gutenbergs-books]$ ./a.out
Unordered Map : 16830ms
[magnat@Pc-Magnat gutenbergs-books]$
```

4 Trie Standart

Pour cette deuxième méthode, nous devons faire un trie utilisant la bibliothèque standard, c'est à dire, un trie utilisant `unordered_map` de caractère et de pointeur sur trie (`std::unordered_map<char, std::unique_ptr<trie>>`) et un compteur de type `int`.

J'ai ajouté à cette classe un `operator[]` qui prend une string et qui retourne l'adresse du compteur correspondant au mot rentré.

Après pour se faire, on utilise un pointeur sur trie qu'on initialise au trie courant puis on parcourt le string et pour chaque caractère jusqu'au dernier, on récupère le pointeur contenu dans l'`unordered map` du caractère actuel (à l'aide de la méthode `get` des uniques pointeurs) sans oublier d'effectuer un `make_unique` dans le cas où le pointeur contenu dans l'`unordered map` serait nul. A la fin il ne reste plus qu'à retourner le compteur pointé par le dernier noeud sauvegardé dans notre pointeur sur trie.

Cette méthode est beaucoup plus lente que la première (environ 38 000 ms)...

4.1 Exemple de résultat

```
[magnat@Pc-Magnat gutenbergs-books]$ g++ -O3 Projet_trie.cpp
[magnat@Pc-Magnat gutenbergs-books]$ ./a.out
Trie : 39168ms
[magnat@Pc-Magnat gutenbergs-books]$
```

5 Mon trie

Pour cette dernière méthode, nous devons encore faire un trie mais cette fois-ci à notre sauce. J'ai donc pour objectif de faire mieux que le trie de la seconde méthode étant donné sa lenteur mais aussi d'essayer de dépasser l'unordered map de la première méthode. Pour se faire, je me suis dit qu'un bon moyen pour accéder rapidement au noeud suivant serait de pouvoir savoir rapidement où trouver l'adresse du prochain trie. Par chance, nous travaillons sur des mot qui, par définition, sont limités en caractères (26 lettres dans notre alphabet), je me suis donc dit qu'utiliser un array de pointeur sur trie de taille 26 serait une bonne solution. Je suis donc parti là-dessus.

Voici les composantes de ma classe trie :

```
std::array<std::unique_ptr<trie>,26> next_;  
int nb;
```

J'ai ensuite défini un operator[] de la même manière que pour le premier trie sauf que cette fois-ci pour trouver l'adresse du prochain trie je n'ai plus à utiliser un unordered map mais juste à regarder à l'index de mon array (next_) correspondant (le caractère actuelle moins la valeur de ascii de a ('a')).

Ensuite partant du principe qu'un mot fait au minimum 1 caractère, j'ai décidé d'utiliser un tableau de ce trie de taille 26 pour commencer directement au deuxième caractère (Par exemple le mot "emotion" sera dans le 5eme trie et on y accèdera en passant "motion" à l'opérateur croché) mais avec cette manière de procéder, j'ai dû définir une nouvelle méthode "ajoutmot" qui elle ne travaille pas sur un string mais sur deux itérateurs de string pour pouvoir directement passer l'adresse du deuxième caractère plutôt que le premier pointer par la méthode begin de string. Cela rend son utilisation plus compliquée mais accélère sa vitesse.

Avec ce trie, mon objectif est atteint, je mets moins de temps que les deux premières méthodes vues ci-dessus (environ 8 000 ms).

5.1 Exemple de résultat

```
[magnat@Pc-Magnat gutenbergs-books]$ g++ -O3 Projet_mon_trie.cpp  
[magnat@Pc-Magnat gutenbergs-books]$ ./a.out  
Mon trie : 7964ms  
[magnat@Pc-Magnat gutenbergs-books]$
```

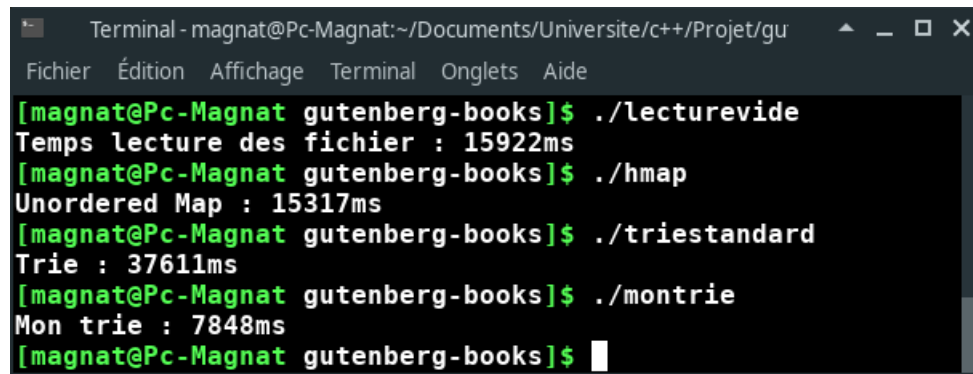
6 Utilisation

L'utilisation des programmes est assez basique, le projet est rendu avec un makefile comprenant 5 methodes :

- make (Qui compilera tout les fichiers)
- make hmap
- make trie
- make montrie
- make lecturevide

Ils correspondent chacun aux 3 méthodes vu plus haut dans le rapport ainsi qu'à un programme pour calculer le temps mis à lire tous les fichiers.

7 Screen d'utilisation



```
Terminal - magnat@Pc-Magnat:~/Documents/Universite/c++/Projet/gu
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[magnat@Pc-Magnat gutenbergs-books]$ ./lecturevide
Temps lecture des fichier : 15922ms
[magnat@Pc-Magnat gutenbergs-books]$ ./hmap
Unordered Map : 15317ms
[magnat@Pc-Magnat gutenbergs-books]$ ./triestandard
Trie : 37611ms
[magnat@Pc-Magnat gutenbergs-books]$ ./montrie
Mon trie : 7848ms
[magnat@Pc-Magnat gutenbergs-books]$
```

Figure 1: Exemple