Algorithmique et structures de données : Mission 3 correction croisée

Groupe 1.2: Ivan Ahad - Jérôme Bertaux - Rodolphe Cambier
Baptiste Degryse - Wojciech Grynczel - Charles Jaquet

07 octobre 2014

Pertinence des produit par rapport aux objectifs : A

Premièrement, Ils utilisent des fonctions permettant de stocker des informations et d'utiliser un dictionnaire. Pour ce faire, ils ont décidé d'implémenter une hashTable. De plus la hashTable utilise bien le principe de buckets pour gérer les collisions comme étudié dans le livre.

Explication du code / qualité de la conception générale: B+

Pour ce qui est du code, ils ont crée 4 classes:

- la principale, étant la classe main permettant l'interaction avec l'utilisateur ainsi que l'initialisation des différentes variables.
- La deuxième est celle qui permet la lecture d'un fichier via un scanner.
- La troisième est la classe journal qui permet de représenter un objet journal.
- Enfin la dernière est celle qui nous interesse le plus et s'appelle HashTableJournals dans laquelle ils utilisent les Hashtable importée directemenent de java.util. Dans l'initialisation de cette classe, il crée une hashtable qu'ils appellent "journals". Ensuite dans le main, ils appellent la fonction fillHashTableJournals() qui va prendre toutes les lignes dans le fichier grâce à la classe "ReadFile". Pour chacune des lignes, ils vont en créer un journal et ensuite l'ajouter dans la Hashtable journals en utilisant le nom du journal comme clé.

Par contre leur code n'est pas vraiment générique (classe Journal et HashTableJournals), et ils n'utilisent pas les boucles à bon escient (pour les tests par exemple).

Qualité des commentaires et des spécifications: A

Le programme est bien commenté et ils ont fait les spécifications pour chaque fonction ce qui rends leur programme lisible et facilement compréhensible. De plus, grâce aux spécifications, il serait facile de réutiliser et/ou modifier leur code pour d'autres fonctionnalités. Bien qu'il faille tout modifier dans le code afin de pouvoir le réutiliser.

Efficacité du code : A

Analyse des différentes compléxité de leur code:

Complexité temporelle

Pour analyser la complexité temporelle de la recherche d'un élément, il faut analyser deux cas :

- **Le meilleur des cas:**
  C'est a dire qu'il n'y a pas eu de collisions, il n'y a donc qu'un seul élément par bucket et donc la complexité est en O(1).

- **Le pire des cas:**
  C'est-à-dire qu'il y a eu des collisions pour tous les éléments de la hashTable. Par exemple si la taille de celle-ci est de 1. Dans ce cas ça va dépendre de comment les éléments sont stockés. Si ils sont stocké dans un tableau alors la complexité est en O(n).

Complexité spaciale

De base la taille du tableau hashTable est de 20714*1,25 éléments. L'interêt est que vu que leur loadFactor est de 0,75, on sait que la taille du tableau hashTable ne vas pas automatiquement grandir lors du remplissage de celui-ci (y compris dans le cas où il n'y aurait pas de collisions).
On peut se permettre de faire ça parce que nous connaissons à l'avance le nombre de journals qu'il faudra stocker. Il l'ont donc codé en dur pour être sur de minimiser cette complexité spaciale.

Nous avons donc, que dans ce programme, la complexité spaciale est de 25893 éléments.

Clareté et pertinence des conclusions tirées dans le rapport : B

Ils n'ont pas vraiment tiré de conclusion dans le rapport. Juste signaler que leur programme peut prendre en compte plusieurs types de fichiers. Mais la modulation se fait juste lors de la lecture du fichier. Leur programme ne pourrait donc pas traiter un fichier similaire contenant autre chose que des journaux avec les mêmes champs. Ils ont indiqué qu'il suffisait de changer la classe journal en cas d'autre fichiers, ou d'en créer une autre selon le besoin. Tout comme la généricité de leur classe HashTableJournal, qui ne fonctionnera pas avec une autre classe sans modification. Sinon, le rapport et le code sont extrêmement clairs et lisibles.