ALGAV

Devoir de programmation

Dictionnaires de mots

KOBROSLI Hassan
13/12/2014

Sommaire

Préambule

L'implémentation du projet a été réalisée en Java avec l'IDE Eclipse. Le projet peut être importé dans Eclipse de cette façon : File > Import > General > Existing projects into workspace > Select archive file : Algav.zip.

Les fichiers sources sont situés dans le dossier Algav/src.

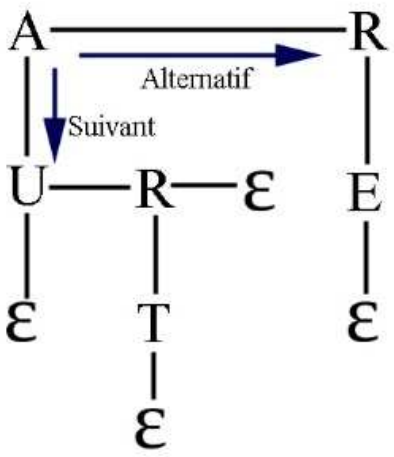
Le projet est composé de 6 packages (dossiers) contenant chacun une classe de test :

- **briandais :** contient les primitives et fonctions avancées sur les arbres de la Briandais.
- **thriehybride :** contient les primitives et fonctions avancées sur les tries hybrides.
- **conversion :** contient les fonctions de conversion d'arbres de la Briandais vers tries hybrides et inversement.
- **utils :** contient les fonctions de lecture de fichiers.
- **Etudeexperimentale :** contient les fonctions de construction d'arbres de la Briandais en parallèle (question 5.13).
- **maintest :** contient le programme principal, permettant d'exécuter les tests des autres packages.

1.1 Structure 1 : Arbres de la Briandais

Question 1.1 : Le caractère utilisé dans cette partie indiquant la fin d'un mot est \0 (code ASCII 00).

Question 1.2 : Un arbre de la Briandais est un Nœud qui contient une clé (ici une lettre), un arbre de la Briandais suivant et un arbre de la Briandais alternatif.



Pour instancier un nouvel arbre de la Briandais il on écrit :

ArbreBriandais arbre = **new ArbreBriandais(char lettre, null, null) ;**

En Java, l'allocation et la désallocation de mémoire se font automatiquement, il n'est donc pas nécessaire d'avoir une fonction permettant de supprimer un arbre. Il suffit d'écrire :

arbre = null;

On dispose de plus de 6 primitives :

- **setLettre(char lettre) :** permet de modifier la lettre (clé) d'un nœud.
- **getLettre() :** permet de récupérer la lettre d'un noeud.
- **setAlternatif(ArbreBriandais arbre) :** permet de modifier le nœud alternatif d'un arbre.
- **getAlternatif() :** permet de récupérer le nœud alternatif d'un arbre.
- **setSuivant(ArbreBriandais arbre) :** permet de modifier le nœud suivant d'un arbre.
- **getAlternatif() :** permet de récupérer le nœud suivant d'un arbre.

Question 1.3 : Afin de rendre la phrase « A quel génial … a ecrire ? » compatible avec le code ASCII, tous les accents des lettres ont été enlevés.

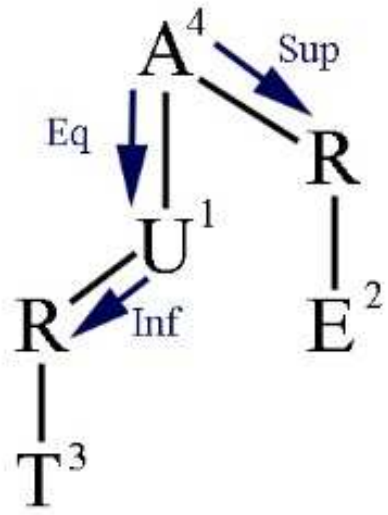
1.2 Structure 2 : Tries Hybrides

Question 1.4 : Un trie hybride est un Nœud contenant une clé, une valeur qui, si elle ne vaut pas **null**, indique une fin de mot, et 3 instances de trie hybride : un trie hybride Inf (respectivement Eq,

respectivement Sup) dont la clé est inférieure (respectivement égale, respectivement supérieure) à la clé du nœud parent.

Pour instancier un nouveau trie hybride on écrit :

TrieHybride arbre = **new TrieHybride(char lettre, null, null, null,null) ;**



2. Fonctions avancées pour chacune des structures

Question 2.6 : Pour les arbres de la Briandais (respectivement tries hybrides), les fonctions avancées sont situées dans la classe Briandais (respectivement Hybride).

3. Fonctions complexes

Question 3.7 : On se donne deux arbres de la Briandais arbre1 et arbre2. Pour fusionner ces deux arbres :

1. On découpe arbre2 selon ses nœuds alternatifs (on se retrouve avec plusieurs nœuds qui n'ont pas de nœud alternatif).
2. On insère chaque nœud découpé à la racine d'arbre1 : s'ils ont tous les deux la même clé, on insère le nœud suivant du nœud dans l'arbre suivant d'arbre1 (en recommençant à l'étape 1, c'est-à-dire en redécoupant le nœud). Sinon, on insère le nœud dans l'arbre alternatif d'arbre1.

La fonction correspondante est : **fusion(ArbreBriandais arbre1,ArbreBriandais arbre2).**

Question 3.8 : Les fonctions de conversion sont situées dans la classe Conversion du package *conversion*.

ArbreBriandais trieHybrideToBriandais(TrieHybride trieHybride)
TrieHybride briandaisToTrieHybride(ArbreBriandais briandais)

La conversion se fait nœud par nœud, récursivement.

Question 3.9 : Le but d'équilibrer un trie hybride est de permettre une insertion ou une recherche plus rapide dans celui-ci. Pour ce faire :

1- On calcule le nombre de nœuds dont le premier caractère est inférieur à la racine (on appelle récursivement **getInf(nœud_courant)** pour compter). On fait de même pour les nœuds supérieurs.
2- Si la différence entre les deux est supérieure à 1, on rééquilibre l'arbre : s'il y a plus de nœuds inférieurs que de nœuds supérieurs, la nouvelle racine sera le nœud inferieur et vice versa.

Le rééquilibrage doit se faire après chaque insertion dans le nœud inférieur ou supérieur de la racine (on rééquilibre d'abord le sous arbre dans lequel on a inséré, puis on rééquilibre l'arbre à la racine).

4. Complexités

A. Arbres de la Briandais

Recherche : A chaque appel récursif, on cherche dans une plus petite partie de l'arbre, la complexité est donc en $O(\log n)$.

Suppression : Même principe que la recherche : $O(\log n)$ appels récursifs.

Comptage Mots : On doit tester chaque nœud de l'arbre pour savoir s'il s'agit de la fin d'un mot : $O(n)$ comparaisons.

Liste Mots : Même principe que le comptage de mots : $O(n)$ comparaisons.

Comptage Nil : Pour chaque nœud de l'arbre, on effectue une comparaison (avec **null**). La complexité est donc en $O(n)$.

Hauteur : Pour déterminer la hauteur d'un arbre, on cherche le maximum des hauteurs des sous arbres. On parcourt donc l'arbre en entier. La complexité est donc en $O(n)$.

Profondeur Moyenne : On cherche toutes les feuilles de l'arbre, il faut donc pour cela parcourir tous les nœuds. La complexité est dont en $O(n)$.

Préfixe : On doit d'abord rechercher le nœud correspondant à la dernière lettre du préfixe, puis compter les mots. La complexité est donc en $O(\log n)$.

Fusion : On doit insérer chaque nœud du deuxième arbre dans le premier soit k le nombre de nœuds du deuxième arbre, la complexité est donc en $O(k \times \log n)$ comparaisons.

B. Tries Hybrides

Recherche : A chaque appel récursif, on divise la taille de l'arbre par 3, la complexité est donc en $O(\log n)$.

Suppression : Même principe que la recherche : $O(\log n)$.

Comptage Mots : On doit tester chaque nœud de l'arbre pour savoir s'il s'agit de la fin d'un mot : $O(n)$ comparaisons.

Liste Mots : Même principe que le comptage des mots : $O(n)$ comparaisons.

Comptage Nil : Pour chaque nœud de l'arbre, on effectue un test. La complexité est donc en $O(n)$.

Hauteur : Pour déterminer la hauteur d'un arbre, on cherche le maximum des hauteurs des sous arbres. On parcourt donc l'arbre en entier. La complexité est donc en $O(n)$.

Profondeur Moyenne : On cherche toutes les feuilles de l'arbre, il faut donc pour cela parcourir tous les nœuds. La complexité est dont en $O(n)$.

Préfixe : On doit d'abord rechercher le nœud correspondant à la dernière lettre du préfixe, puis compter les mots. La complexité est donc en $O(\log n)$.

5. Etude expérimentale

Question 5.11 : Cette question est implémentée dans la classe UtilsTest du package *test*.

Question 5.12 :

Les temps indiqués ci-dessous correspondent à 905534 insertions, 905534 suppressions.

Les profondeur moyenne des feuilles et hauteur sont celles d'un arbre de 23086 mots.

	Arbre de la Briandais	**Trie Hybride**
Temps total d'insertion	618 ms	528 ms
Temps de suppression totale	196 ms	176 ms
Profondeur moyenne des feuilles	5	4
Hauteur	54	34

Question 5.13 : Cette question est traitée dans les classes ListThread et ParallelTest du package *etudeexperimentale*.

On parallélise ici la lecture et l'insertion des mots de chaque fichier dans des arbres de la Briandais différents, puis on fusionne les arbres obtenus. Cela permet d'obtenir un gain de temps de 25 à 50% par rapport à l'insertion de tous les mots un par un dans un seul arbre.