Université de Reims Champagne-Ardenne
Département de Mathématiques et Informatique

Pierre Delisle
Info0501
2018/2019

Travaux Pratiques 4
Arbres binaires de recherche
Analyse de texte – Partie 2

L'objectif de ce TP est d'implémenter le programme du TP3, qui compte le nombre de mots différents dans un texte, à l'aide des arbres binaires de recherche. Pour ce faire, nous allons tout d'abord implémenter une structure d'arbre binaire de recherche permettant de stocker des mots.

1) Créez les fichiers **noeud.c** et **noeud.h** implémentant une structure de noeud permettant de stocker un mot et de le relier à un père, un fils gauche et un fils droit de même type. Définissez les procédures de noeud suivantes :

- **initialiserNoeud** : initialise une cellule à l'aide d'un mot ;
- **détruireNoeud** : libère (si nécessaire) les ressources mémoire d'un noeud.

2) Créez les fichiers **arbre.c** et **arbre.h** et implémentez-y une structure d'arbre binaire de recherche basée sur l'ordre lexicographique des mots (bobard < bobo < gogo < toto) en implémentant les procédures suivantes :

- **initialiserArbre** : initialise un arbre vide ;
- **détruireArbre** : libère (si nécessaire) les ressources mémoire d'un arbre ;
- **insérer** : prend en paramètre un pointeur sur un noeud et insère ce nœud à la bonne position dans l'arbre ;
- **rechercher** : recherche un mot dans un arbre et retourne soit un pointeur sur le noeud qui contient le mot, soit NULL ;
- **supprimer** : prend en paramètre un pointeur sur un nœud d'un arbre et supprime ce noeud de l'arbre.

Notez que pour une implémentation propre qui respecte les principes de la structure d'arbre, les fichiers arbre.c et arbre.h ne devraient pas contenir d'autres fonctions que celles demandées ci-haut. Par conséquent, tout le reste du code que vous développerez dans ce TP doit être à l'extérieur de ces fichiers.

3) Créez les fichiers **outilsArbre.c** et **outilsArbre.h** qui contiendront diverses fonctions utilitaires de manipulation des arbres, et implémentez-y les fonctions **afficherArbreRecursif** et **afficherArbreIteratif** qui affichent à l'écran les informations relatives à votre arbre.

4) Créez un programme de test permettant de vérifier le fonctionnement de votre implémentation en demandant à l'utilisateur de saisir des mots, en les insérant successivement dans un arbre et en affichant l'arbre ainsi créée. Essayez aussi de supprimer un mot, de rechercher un mot et de détruire l'arbre.

L'objectif du programme est de compter le nombre de mots différents dans un texte alors nous allons tenter l'exercice en utilisant notre structure d'arbre binaire de recherche.

5) Récupérez le fichier **texte1.txt** sur le site web du cours (http://cosy.univ-reims.fr/~pdelisle/enseignement.php) et insérez chaque mot différent qui y figure dans votre arbre. Autrement dit, pour chaque mot que vous lisez dans le fichier, recherchez-le dans l'arbre et insérez-le s'il n'y figure pas déjà. Pour l'instant, vous pouvez simplifier le problème en supposant qu'un mot est une chaîne de caractères précédée et suivie d'un blanc (espace ou fin de ligne). Autrement dit, vous pouvez supposer que "Quebec" et "Quebec." sont deux mots différents. Une fois tout le fichier traité, comptez le nombre d'éléments présents dans l'arbre en définissant la fonction **compterArbre** et vérifiez la conformité du résultat.

6) Testez votre programme avec des textes de plus grande taille présents sur le site web du cours. Il est à noter que pour éviter les problèmes de conversion de caractères en C, les caractères accentués ont été remplacés (é → e, à → a, etc.) et certains caractères spéciaux (°, £, etc.) ont été éliminés. De plus, pour comparer le résultat avec vos implémentations en liste et en table de hachage du TP précédent, ne considérez que les mots de 26 caractères et moins. Donnez le nombre total de mots présents dans le fichier, le nombre de mots différents et le temps d'exécution requis pour compter le nombre de mots différents. Vérifiez la similarité des résultats avec vos implémentation en liste et en table de hachage, et comparez les temps d'exécution.

Questions supplémentaires pour les rapides et les motivés

- Écrivez un programme qui compte le nombre d'occurrences de chaque mot d'un texte ;
- Écrivez un programme qui, pour un fichier texte donné, créé un fichier d'index qui donne les lignes où se trouve chaque mot ;
- Améliorez votre programme pour gérer les ponctuations (points, virgules, apostrophes, etc.) ;
- Ajoutez à votre programme une fonction de parcours en largeur de votre arbre ;
- Refaites l'exercice en utilisant un arbre rouge et noir plutôt qu'un arbre binaire de recherche et comparez les résultats obtenus.