INFO0902-1 - Structures de données et Algorithmes

Projet 1: Arbres Binaires de Recherche

Sébastien Collette (sebastien.collette@ulb.ac.be)

Introduction et modalités

L'objectif de ce projet est d'implémenter les arbres binaires de recherche en langage C. Nous vous demandons de remettre le code source correspondant aux désidératas repris ci-dessous, ainsi qu'un bref rapport expliquant ce que vous avez fait et les problèmes rencontrés. Le tout doit être envoyé par e-mail à l'adresse `sebastien.collette@ulb.ac.be` **avant le cours théorique du 31 mars 2011**. Des séances de réponses à vos questions seront organisées pendant toute la durée du projet.

Le fichier "projet1.c" que vous allez remettre devra être compilable sur les ordinateurs du réseau "candi", à savoir les ordinateurs `candiXX.montefiore.ulg.ac.be` qui ont été utilisés lors des laboratoires, via la commande suivante :

```
gcc -Wall -std=c99 -o projet1 projet1.c
```

Il est donc conseillé de compiler et tester son programme sur un des ordinateurs du réseau "candi". Ces ordinateurs sont accessibles à distance (voir `http://www.montefiore.ulg.ac.be/~dethier/INFO2009-1/putty.html`).

Si vous n'avez pas encore d'identifiant pour accéder aux ordinateurs du réseau "candi", rendez-vous personnellement au bureau de Gérard Dethier (Bureau 1/6 au B37) pour vous le procurer. Les identifiants et mots de passe associés ne seront pas transmis par mail.

Finalement, il s'agit d'un travail **personnel** ! Si vous éprouvez des difficultés à réaliser votre projet, rendez-vous aux séances de réponses aux questions, où des assistants et étudiants-moniteurs seront disponibles pour vous aider.

1 Implémentation des arbres binaires de recherche - 12 points

La première partie de ce projet consiste à compléter le code source ci-joint, qui contient déjà les types structurés nécessaires pour implémenter un arbre binaire de recherche classique. Nous vous demandons, en particulier, d'implémenter les opérations suivantes :

`void insertKey(int i, struct tree *T)` qui insère l'entier *i* dans l'arbre binaire de recherche *T*. N'oubliez pas que l'insertion s'effectue *en feuille*.

`void deleteKey(int i, struct tree *T)` qui supprime l'entier *i* dans l'arbre binaire de recherche, pour peu que celui-ci soit présent dans l'arbre. Utilisez la méthode vue au cours : si le nœud contenant *i* dispose d'un fils gauche et d'un fils droit, il faut trouver un *remplaçant* pour *i*.

`bool findKey(int i, struct tree *T)` qui retourne true (=vrai) ou false (=faux) selon que l'élément *i* se trouve effectivement dans l'arbre.

`void inOrder(struct tree *T)` qui affiche un parcours de l'arbre "en-ordre" de façon récursive, comme vue au cours.

Pour chacune de ces opérations, le rapport décrira votre méthode, et indiquera la complexité, en notation grand-O(). Exprimez la complexité dans le pire cas en fonction de la hauteur de l'arbre h et/ou en fonction de n (le nombre d'éléments dans l'arbre), si cela est pertinent.

2 Arbre enrichi - 4 points

Ajoutez deux champs complémentaires au type structuré correspondant aux nœuds de l'arbre : un champ *size* et un champ *height*, qui représenteront le nombre d'éléments dans le sous-arbre dont le nœud est la racine, et la hauteur du sous-arbre. Ces champs devront être mis à jour à chaque insertion et suppression. Adaptez le parcours "en-ordre" pour qu'il affiche, en plus de la valeur du nœud, les champs *size* et *height*.

3 Fusion - 4 points

Etant donnés deux arbres T et U, implémentez une opération de fusion des deux arbres, qui consiste à mettre tous les éléments de U dans T. A l'issue de cette opération, l'arbre U doit être vide. N'oubliez pas de mentionner la complexité de cette opération.

4 Questions Bonus - +1 point à l'examen final

Une première question bonus est d'implémenter la fusion de façon efficace. L'exercice demandé au point précédent nécessite uniquement d'utiliser les fonctions `deleteKey` dans U et `insertKey` dans T. Ne peut-on pas faire mieux ? Par exemple, si les deux arbres sont tels que toutes les clés de T sont plus petites que les clés de U ?

Une deuxième question bonus consiste à implémenter la variante AVL, c'est-à-dire à faire des rotations dans l'arbre, de façon à maintenir un arbre de profondeur $O(\log n)$. Etant donné que les nœuds de vos arbres contiennent déjà la taille des sous-arbres, il suffit de déterminer les cas de déséquilibre et d'appliquer les rotations vues au cours.

Chacune de ces questions vaut 1 point complémentaire lors de l'examen de juin.