TD Noté: Arbres binaires et ternaires de recherche

9 Décembre 2015

Le travail est individuel. Il faut rendre un fichier portant le nom de l'étudiant suivi de l'extension `.c` *(donc* `toto.c` *pour l'étudiant de nom Toto), et le déposer sur e-campus2 dans la boîte de dépôt "abr1" ou "abr2" selon votre groupe. Le travail rendu doit compiler sans erreurs avec la commande* `gcc toto.c` *pour être évalué. Il sera tenu compte de la lisibilité du code.*

Objectif

Dans ce TD, nous considérons des arbres qui contiennent des valeurs de type `int`. Un arbre binaire de recherche (ABR) est un arbre binaire qui satisfait la propriété suivante : quel que soit le sommet de l'arbre, sa valeur est strictement supérieure à la valeur de n'importe quel sommet du sous-arbre gauche, et inférieure ou égale à la valeur de n'importe quel sommet du sous-arbre droit. Un arbre ternaire est sa généralisation à trois sous-arbres par sommet.

Travail à effectuer

Un fichier `etudiant.c` est à récupérer sur e-campus et à compléter. Celui-ci contient une fonction

```
int alea(int n)
```

qui renvoie un nombre aléatoire entre 0 et `n-1`, et un exemple (à tester) de chronomètre dans la fonction `main` pour mesurer le temps d'exécution d'une fonction.

Voici la liste des fonctions et structures à implémenter :

— définir la structure d'arbre binaire `ABR` ;
— fonction qui ajoute une valeur dans l'arbre ;
— fonction qui remplit un ABR avec `NMAX` valeurs aléatoires positives comprises entre 0 et `NMAX-1` ;
— fonction qui affiche les valeurs de l'ABR par ordre croissant ;
— fonction qui supprime complètement un ABR, en libérant la mémoire associée ;

En utilisant le chronomètre, vous mesurerez le temps d'insertion moyen de `NMAX`= 100000 valeurs dans un ABR.

Écrire les mêmes fonctions et faire les mesures pour un arbre *ternaire* de recherche (dénommé ATR) que vous programmerez en suivant les règles décrites après.

Dans un ATR, chaque sommet i de l'arbre peut stocker deux entiers positifs (a_i et b_i avec $a_i \leq b_i$) et trois pointeurs vers des noeuds (`fg`, `fm` et `fd`). Dans ce cas :

— `fg` pointe sur un sous-arbre tel que toute valeur x stockée dans les noeuds du sous-arbre vérifie $x < a_i$.
— `fm` pointe sur un sous-arbre tel que toute valeur x stockée dans les noeuds du sous-arbre vérifie $a_i \leq x \leq b_i$.
— `fd` pointe sur un sous-arbre tel que toute valeur x stockée dans les noeuds du sous-arbre vérifie $b_i < x$.

Une feuille de l'arbre peut ne contenir qu'un seul entier et dans ce cas l'autre entier vaut -1 et les trois pointeurs pointent sur NULL. Lors de l'insertion d'une valeur dans l'arbre, soit cette valeur est rajoutée à un noeud ne contenant qu'une valeur, soit un nouveau noeud est créé.

Quelle structure est la plus efficace pour l'insertion de valeurs ? Vous indiquerez le résultat de vos mesures en commentaire.

Pour terminer, vous écrirez une fonction qui supprime la racine d'un ABR ; il faut que le nouvel arbre soit encore un ABR ; pour cela on pourra utiliser la méthode suivante :

— si la racine est une feuille : il suffit de l'enlever de l'arbre vu qu'elle n'a pas de fils ;

— si la racine a un unique fils : Il suffit l'enlever de l'arbre et de le remplaçant par son fils ;
— sinon il suffit de le remplacer par le sommet de plus petite valeur du sous-arbre droit (ou le sommet de plus grande valeur du sous-arbre gauche) et reconstituer les connexions entre sommets de manière à conserver la structure d'ABR.

Les fonctions doivent être testées, et vos tests doivent être conservés, éventuellement en commentaires.