Semaine 4 – Arbres binaires de recherche

Dans ce TP, vous allez implémenter des fonctions de manipulation d'arbre binaires de recherche sur le même modèle que les listes des TP précédents. Là encore, il ne vous est pas demander de vérifier que les allocation se sont bien passées même si un programme réel devrait faire ces vérifications.

1 Mise en place

Dans ce TP, nous cherchons à implémenter un ensemble de structures de données et de fonctions permettant d'utiliser des arbres binaires de recherche d'entiers. Chaque noeud de l'arbre doit donc contenir une valeur entière ainsi que des pointeurs vers les sous-arbres gauche et droit.

Dans un premier temps, implémentez les éléments de base nécessaires à la création d'un arbre et à son affichage afin de simplifier les tests des fonctions suivantes. Pensez à bien tester vos fonctions !

1. Commencer par définir les deux structures nécessaires à la gestion d'un arbre binaire sur le même modèle que pour les liste chainées.
2. Écrivez maintenant une fonction permettant de créer un arbre vide. Attention de bien initialiser tout ce qui doit l'être.

   ```
   arbre *nouvel_arbre();
   ```

3. Afin de simplifier l'écriture des fonctions suivantes, implémentez aussi une fonction permettant de créer un nouveau noeud de l'arbre avec une valeur fournie en paramètre et initialisant ses enfants à `NULL`.

   ```
   elem *nouvel_elem(int val);
   ```

4. Et enfin, une fonction affichant simplement l'arbre. Vous êtes libre dans le format de cet affichage mais assurez-vous qu'il vous permette de bien visualiser la structure de celui-ci. *Ne passez pas trop de temps sur l'affichage, celui-ci doit juste être fonctionel, vous pourrez y revenir à la fin du TP s'il vous reste du temps.*

   ```
   afficher_arbre(arbre *a);
   ```

2 Fonctions d'édition et de recherche

Pour rappel, dans un arbre binaire de recherche, tous les noeuds respectent les deux propriétés suivantes :

— toutes les valeurs présentes dans les noeuds du sous-arbre gauche sont *strictement inferieures* à la valeur du noeud ;
— toutes les valeurs présentes dans les noeuds du sous-arbre droit sont *supperieures ou égales* à la valeur du noeud.

Ces propriétés permettent une recherche très efficace d'une valeur dans l'arbre mais les fonctions de modification de l'arbre doivent les préservées.

1. Écrivez une fonction permettant d'ajouter une valeur donnée en paramètre à un arbre. Il est possible d'écrire cette fonction soit de manière itérative, soit de manière récursive, vous êtes libre de choisir. *Approfondissement : une fois le reste du TP terminé, implémentez l'autre alternative et comparez les.*
2. Écrivez une fonction permettant de retrouver la plus petite valeur stockée dans un arbre donné en paramètre :

   ```
   int chercher_minimum(arbre *a);
   ```

3. Écrivez une fonction permettant de calculer la profondeur maximale d'un arbre donné :

   ```
   int profondeur(arbre *a);
   ```

   Cette fonction peut elle aussi être implémentée de manière itérative ou récursive. Toutefois, la version itérative de cette fonction est beaucoup plus complexe et il vous est fortement recommandé de choisir ici une implémentation récursive.
4. L'ordre dans lequel les éléments sont ajoutés à un arbre à une forte influence sur la profondeur maximale de celui-ci et donc sur les performances de la recherche. Comparez la profondeur de l'arbre obtenus suivant que les valeurs sont ajoutées en ordre aléatoire ou bien déjà triées.