Travaux Dirigés d'algorithmique n°7

Cours d'Informatique de Deuxième Année

—Deug SCIENCES M3-MI—

Arbres binaires de recherche

L'objet de cette feuille d'exercices est la manipulation d'une nouvelle structure de données récursive : la structure d'*arbre binaire de recherche*. L'algorithmique que l'on utilise sur ces structures est voisine de celle que l'on utilise sur les listes chaînées à ceci près que l'on ne parle plus du "suivant" d'une cellule mais plutôt du "fils gauche" et "fils droit" d'un nœud. De même, on ne parle plus du "début" de la liste mais plutôt de la "racine" de l'arbre.
Nous allons considérer :

- la déclaration de la structure et les algorithmes de manipulation ;
- l'utilisation d'un arbre binaire de recherche pour trier une suite d'éléments.

---

1 Structure d'ABR et algorithmes de base

On considère les déclarations suivantes :

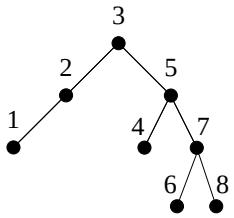
```
typedef struct noeud
{ float Val;                    /* valeur dans le noeud                */
  struct noeud * Gauche;       /* pointeur sur le sous-arbre gauche   */
  struct noeud * Droit;        /* pointeur sur le sous-arbre droit    */
} StrNoeud;                    /* nouveau type : StrNoeud             */

typedef StrNoeud * PtrNoeud;  /* pointeur sur un noeud               */
```

Un arbre binaire est assimilé avec un pointeur sur sa racine. Un arbre binaire, représenté par la structure `PtrNoeud` ci-dessus, est dit *arbre binaire de recherche* ou encore *ABR* si les conditions suivantes sont vérifiées :

- ▷ l'arbre vide est un *ABR* ;
- ▷ les deux sous-arbres Gauche et Droit de la racine de l'arbre sont des *ABR* et la valeur à la racine de l'arbre est plus grande (resp. petite) que toutes les valeurs du sous-arbre gauche (resp. droit).

Exemple :



▸ **Exercice 1. (Fonctions de base)** *Afin de pouvoir manipuler une telle structure de données, nous demandons d'écrire les fonctions élémentaires suivantes :*

1. `CreerFeuille(val)` *qui retournera un pointeur sur un nœud de valeur* `val` *; (une feuille est un nœud dont les deux sous arbre Gauche et Droit sont vides) ;*
2. `EstFeuille(abr)` *qui testera si le nœud pointé par* `abr` *est une feuille ;*
3. `EstVide(abr)` *qui testera si l'arbre pointé par* `abr` *est vide.*

▸ **Exercice 2. (Recherche)** *Donner une fonction de nom* `RechercheBin` *prenant en paramètre un réel* `x` *et un arbre binaire de recherche pointé par* `abr` *et retournant la valeur vrai si l'élément* `x` *est présent dans l'arbre pointé par* `abr` *et faux sinon. Quelle est la complexité de cette fonction ?*

▸ **Exercice 3. (Parcours)** *Donner les fonctions de parcours préfixe, suffixe et infixe, d'un arbre binaire de recherche. Quel parcours donne la suite triée des valeurs ?*

2 Algorithmes complexes

▸ **Exercice 4. (Insertion)** *On considère l'opération d'insertion d'un élément dans un arbre binaire de recherche. Comme pour les listes chaînées triées, cette opération se caractérise par la propriété suivante : si* `abr` *est un arbre binaire de recherche et* `x` *un élément, alors le résultat de l'insertion de* `x` *dans l'arbre* `abr` *est encore un arbre binaire de recherche.*

La création d'un arbre binaire de recherche consiste à insérer successivement dans l'arbre binaire de recherche les valeurs souhaitées si elles ne s'y trouvent pas déjà. Au départ, l'arbre ne contient aucune valeur.

1. *nous demandons tout d'abord de construire à la main les arbres binaires de recherche correspondant aux insertions successives dans l'arbre vide, des valeurs* $3, 5, 7, 2, 4, 1$ *;*
2. *déduire des exemples précédents l'algorithme d'insertion d'un élément* `x` *dans un arbre binaire de recherche pointé par* `abr`*. Quel est la complexité de cet algorithme ?*
3. *donner alors la fonction de nom* `InsertionBin` *prenant en paramètre* `x` *et* `abr`*, réalisant cet algorithme.*

▸ **Exercice 5. (Tri d'un tableau)** *Ecrire une fonction de nom* `TriBin` *de tri d'un tableau d'entiers* `t` *à* `n` *éléments. Cette fonction utilisera un arbre binaire de recherche comme structure intermédiaire. Le résultat du tri sera stocké dans ce même tableau* `t`.

▸ **Exercice 6. (Suppression)**

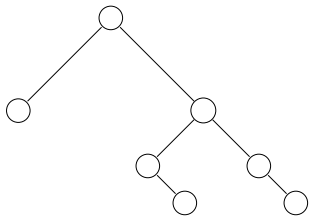
1. *Donner une fonction de nom* `OterMin` *qui enlève le plus petit élément dans un arbre binaire de recherche non vide pointé par* `abr` *et retourne la valeur de ce plus petit élément ainsi que l'arbre modifié.*

2. *En déduire une fonction de nom* `SupBin` *de suppression d'un élément* `x` *dans un arbre binaire de recherche pointé par* `abr`. *Le résultat de cette suppression sera bien évidemment encore un arbre binaire de recherche.*

3 Exercices supplémentaires

▸ **Exercice 7.** *Nous nous intéressons au parcours suffixé d'un arbre binaire (qui dans ce cas, n'est pas un arbre binaire de recherche) :*

- *quel est toujours le dernier sommet (nœud ou feuille) affiché lors du parcours suffixé d'un arbre binaire ?*
- *compléter les 7 nœuds et feuilles de l'arbre binaire suivant, afin que son parcours suffixé soit la suite croissante 1 2 3 4 5 6 7.*



▸ **Exercice 8.** *Donner un algorithme qui teste si un arbre binaire est un arbre binaire de recherche*