Travaux Dirigés d'Algorithmique n°7

Cours d'Informatique de Deuxième Année

—Deug SSM M3-MI—

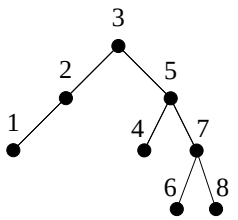
L'objet de cette feuille d'exercices est la découverte d'une nouvelle structure de donnée : la structure d'*arbre binaire de recherche*. L'algorithmique que l'on utilise sur ces structures est voisine de celle que l'on utilise sur les listes chaînées à ceci près que l'on ne parle plus exactement du "suivant"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

On considère les déclarations suivantes :

```
typedef struct noeud
{ int Val;                    /* valeur dans le noeud                */
  int NbOcc;                  /* nombre d'occurences de cette valeur */
  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               */
```

sans oublier la propriété que l'on a imposée à ces arbres, propriété qui en fait des arbres binaires de **recherche**.



▸ **Exercice 1.** *Afin de pouvoir manipuler une telle structure de donnée, nous demandons d'écrire les fonctions élémentaires suivantes :*

1. *`CreerFeuille(val, nbocc)` qui retournera un pointeur sur un noeud contenant `nbocc` occurence(s) de la valeur `val` ;*
2. *`EstFeuille(abr)` qui testera si le noeud pointé par `abr` est une feuille ;*
3. *`EstVide(abr)` qui testera si l'arbre pointé par `abr` est vide.*

▸ **Exercice 2.** *Donner une fonction de nom `RechercheBin` prenant en paramètre un entier `x` et un arbre binaire de recherche pointé par `abr` et retournant le nombre d'occurences de l'élément `x` dans l'arbre pointé par `abr`.*

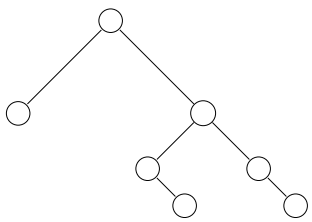
▸ **Exercice 3.** *On considère l'opération d'insertion d'un élément dans un arbre binaire de recherche. Comme pour les listes chaînées triées, cette opération se caractérise par la propriété suivante : si* `abr` *est un arbre binaire de recherche et* `x` *un élément, alors le résultat de l'insertion de* `x` *dans l'arbre* `abr` *est encore un arbre binaire de recherche.*

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`*.*
3. *donner alors la fonction de nom* `InsertionBin` *prenant en paramètre* `x` *et* `abr`*, réalisant cet algorithme.*

▸ **Exercice 4.** (**Parcours**) *Donner les fonctions de parcours préfixe, suffixe et infixe, d'un arbre binaire de recherche.*

▸ **Exercice 5.** (**Utilisation d'un arbre binaire de recherche pour réaliser un tri**) *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.** *Donner une fonction de nom* `SupBin` *de suppression d'un élément* `x` *dans un arbre binaire de recherche pointé par* `abr`*. La fonction supprimera une seule occurence de* `x`*. Le résultat de cette suppression sera bien évidemment encore un arbre binaire de recherche.*

▸ **Exercice 7.** (**exercice supplémentaire**) *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 (noeud ou feuille) affiché lors du parcours suffixé d'un arbre binaire ?*
- *compléter les 7 noeuds et feuilles de l'arbre binaire suivant, afin que son parcours suffixé soit la suite croissante 1 2 3 4 5 6 7.*