



CS316

## TP 2 Arbres binaires de recherche

Récupérer le squelet fournit pour ce TP à partir de l'adresse suivante :  
<http://debbabi.net/teaching/cs316/tp2.tar.gz>

**A rendre à la fin de la séance :**

- Un fichier zip contenant l'ensemble des solutions, à envoyer aux adresses mail suivantes :  
 bassem.debbabi@lcis.grenoble-inp.fr et dbassem@gmail.com

Un *arbre binaire de recherche* ABR, ou *arbre binaire ordonné* est un arbre binaire tel que pour tout noeud  $N$ , toutes les valeurs du sous-arbre gauche de  $N$  sont inférieures ou égales à la valeur de  $N$ , et toutes les valeurs du sous-arbre droit de  $N$  sont supérieures à la valeur de  $N$ .

Soit la structure de donnée suivante pour les arbres binaires de recherche :

```
typedef struct Noeud {
    int valeur;
    struct Noeud* gauche;
    struct Noeud* droit;
} Arbre;
```

Dans les exercices suivants, vous allez implémenter différentes fonctions relatives au fonctionnement d'un arbre binaire de recherche.

Exemple d'utilisation de la structure Arbre :

```
Arbre *arbre = creer(5);
inserer(arbre, 3);
inserer(arbre, 4);
inserer(arbre, 9);
inserer(arbre, 1);
inserer(arbre, 6);
afficher(arbre); // affiche sur le terminal: 1 3 4 5 6 9
detruire(arbre);
```

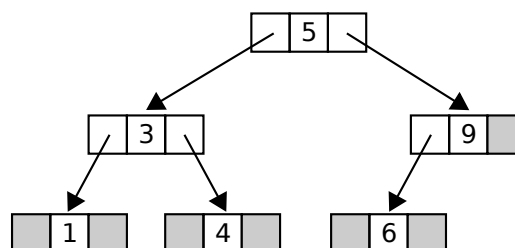


Fig. 1: Arbre binaire de recherche

*Remarques :*

- durant ce TP, les prototypes exacts des fonctions sont donnés dans l'énoncé (et dans le fichier `Arbre.h` sur l'archive fournit). Vous devriez les implémenter dans le fichier `Arbre.c`
- n'oubliez pas de vérifier régulièrement sur des exemples le bon fonctionnement des fonctions que vous écrivez (vous pouvez utiliser le fichier `tp2.c` et la commande `make`)

## Exercice 1

Implémenter la fonction de création qui prend en argument une valeur entière et renvoie un noeud de l'arbre binaire de recherche.

```
Arbre* creer(int valeur);
```

Un espace en mémoire est alloué dynamiquement pour le nouveau noeud créé.

## Exercice 2

Implémenter la fonction de destruction d'un arbre binaire de recherche. Elle libérera l'espace mémoire occupé par les noeuds de l'arbre.

```
void detruire(Arbre* racine);
```

## Exercice 3

Implémenter et tester la fonction d'insertion :

```
struct Noeud* inserer(Arbre* racine, int valeur);
```

Cette fonction permet d'ajouter la valeur **“valeur”** dans la bonne position dans l'arbre binaire ordonné. Elle retourne un pointeur vers le noeud nouvellement créé.

## Exercice 4

Implémenter et tester les fonctions d'affichages suivantes :

```
void afficher(Arbre* racine);
```

Cette fonction permet d'afficher les valeurs de l'arbre binaire par ordre croissant. L'exemple de la figure 1 sera afficher comme suit : 1 3 4 5 6 9

```
void afficher2(Arbre* racine);
```

Cette fonction permet d'afficher les valeurs de l'arbre binaire de manière à voir la structure interne de l'arbre. L'exemple de la figure 1 sera afficher comme suit :  $\{\{\_,1,\_ \},3,\{\_,4,\_ \}\},5,\{\{\_,6,\_ \},9,\_ \}\}$ . Les '\_' indiquent les sous-arbres vides.

## Exercice 5

Implémenter et tester les fonctions suivantes permettant l'obtention de quelques caractéristiques d'un arbre binaire de recherche.

```
int verifie(Arbre* racine);
```

Cette fonction renvoie un entier non nul si et seulement si l'arbre binaire passé en paramètre est correct (c'est à dire qu'il satisfait les propriétés d'arbre binaire de recherche).

```
int taille(Arbre* racine);
```

Cette fonction prend un arbre binaire et rend le nombre de ses noeuds.

```
int hauteur(Arbre* racine);
```

Cette fonction prend un arbre binaire et rend sa hauteur, c'est à dire le nombre de noeuds contenus dans la plus longue branche.

```
int somme(Arbre* racine);
```

Cette fonction prend un arbre binaire et rend la somme de ses valeurs.

```
int moyenne(Arbre* racine);
```

Cette fonction prend un arbre binaire et rend la moyenne de ses valeurs.

## Exercice 6

Implémenter et tester les fonctions de recherche suivantes :

```
struct Noeud* chercher(Arbre* racine, int valeur);
```

Etant donné un arbre binaire de recherche, cette fonction retourne NULL si un noeud avec la valeur donnée en argument est introuvable dans l'arbre. Sinon, elle retourne un pointeur vers le premier noeud trouvé. Cette fonction est récursive.

```
struct Noeud* chercher2(Arbre* racine, int valeur);
```

Pareille que la première fonction, sauf qu'elle n'est pas récursive.

```
struct Noeud* minimum(Arbre* racine);
```

Cette fonction retourne un pointeur sur le noeud contenant la valeur minimale de l'arbre.

```
struct Noeud* maximum(Arbre* racine);
```

Cette fonction retourne un pointeur sur le noeud contenant la valeur maximal de l'arbre.

```
struct Noeud* successeur(struct Noeud* noeud);
```

Cette fonction retourne un pointeur sur le noeud contenant la valeur suivante en ordre croissant par rapport à la valeur du noeud donné en paramètre.

```
struct Noeud* predecesseur(struct Noeud* noeud);
```

Cette fonction retourne un pointeur sur le noeud contenant la valeur précédente en ordre croissant par rapport à la valeur du noeud donné en paramètre.

## Exercice 7

Implémenter et tester la fonction de suppression d'une valeur dans un arbre binaire de recherche.

```
struct Noeud* supprimer(Arbre* racine, int valeur);
```

Cette fonction ne supprime que la première occurrence. Après appel à cette fonction, le noeud trouvé est retiré de l'arbre tout en conservant les propriétés de l'arbre binaire de recherche. Le noeud supprimé est ainsi retourné (ou NULL si la valeur est introuvable). L'algorithme est le suivant (une fois trouvé le noeud contenant la valeur en question) :

- si le noeud à enlever ne possède aucun fils, on l'enlève,
- si le noeud à enlever n'a qu'un fils, on le remplace par ce fils,
- si le noeud à enlever a deux fils, on le remplace par le sommet de plus petite valeur dans le sous-arbre droit, puis on supprime ce sommet.

## Exercice 8

Ecrire une fonction réalisant la fusion de deux arbres binaires de recherche.

```
Arbre* fusion(Arbre* arbre1, Arbre* arbre2);
```

## Exercice 9

Ecrire une fonction qui vérifie si deux arbres binaires de recherche sont équivalents.

```
int equivalents(Arbre* arbre1, Arbre* arbre2);
```

Cette fonction renvoie une valeur positive si et seulement si les deux arbres binaires ont la même structure d'arbre et qu'ils portent les mêmes valeurs aux noeuds se correspondant.

## Exercice 10

Implémenter la fonction d’affichage suivante :

```
void afficher3 (Arbre* racine);
```

Cette fonction permet aussi d’afficher les valeurs de l’arbre binaire de manière à voir la structure interne de l’arbre. L’exemple de la figure 1 sera afficher comme suit :

```

      5
     ~~~
    ,   ,
   3     9
   ~
  ,   ,   ,
 1  4   6
```

Un autre exemple :

```

      5
     ~~~~~
    ,   ,   ,
   3     9
   ~
  ,   ,   ,   ,   ,
 1  4   7     13
      ,   ,   ,   ,
      6  8  10  16
```