

# TP 1 Algorithmique Avancée : Arbres Binaires Équilibrés

CERI - Licence 2 Informatique

Semestre 3

Le TP consiste à implémenter et tester une classe, nommée AVL, représentant un arbre binaire de recherche équilibré. Sa réalisation nécessite de faire les exercices du TD “Arbres binaires (2)”.

## Partie 1.

1/ Créer un fichier avl.h contenant les déclarations des classes suivantes.

- ```
class noeud
{
    friend class AVL;
    int cle;
    int N;
    int d;
    int h;
    noeud * fg;
    noeud * fd;
    noeud * pere;
public :
    noeud(int x);
    ~noeud();
    void affiche();
};
```

Si  $v$  est de type `noeud`, son attribut  $N$  contiendra le nombre de noeuds de l'arbre enraciné en  $v$ ,  $d$  contiendra le déséquilibre en  $v$ , et  $h$  la hauteur de

l'arbre de racine  $v$ . “affiche()” permet d’afficher tous les attributs du noeud.

La classe AVL est définie comme suit.

```
• class AVL
{
    noeud * r;
public :
    AVL(noeud* r);
    ~AVL();
    noeud* root();
    void prefixe(noeud* x);
    int hauteur(noeud* x);
}
```

La méthode “prefixe” permet d’afficher le contenu du sous-arbre enraciné en “x” avec un parcours préfixe, L’affichage des informations d’un noeud doit se faire avec la méthode “affiche()” de la classe noeud.

2/ Ajouter à votre classe “AVL” les méthodes suivantes :

- AVL(char\* filename, bool option) : constructeur permettant de créer un arbre binaire de recherche par insertion des données provenant d’un fichier “filename”. Si option == true, les insertions doivent se faire en feuille, sinon en racine.
- void rotationDroite(noeud\* x) : permet d’effectuer une rotation droite en un noeud  $x$  quelconque de l’arbre.
- void rotationGauche(noeud\* x) : permet d’effectuer une rotation gauche en un noeud  $x$  quelconque de l’arbre.
- void insertionFeuille(noeud\* x, noeud\* y) : permet d’insérer en feuille un nouveau noeud  $y$  dans l’arbre de racine  $x$ .
- void insertionRacine(noeud\* x, noeud\* y) : permet d’insérer en racine un nouveau noeud  $y$ , dans l’arbre de racine  $x$ .
- int noeuds(noeud\* x) : met à jour l’attribut  $N$  de tous les noeuds de l’arbre de racine  $x$ . Le contenu  $x \rightarrow N$  est renvoyé.
- void desequilibres(noeud\* x) : calcule le déséquilibre en chaque noeud d’un arbre de racine  $x$ .
- noeud\* partition(noeud\* x, int k) : partitionne l’arbre par rapport à sa  $k$ -ième plus petite valeur et renvoie l’adresse de celle-ci (voir TD

Arbres Equilibrés).

- void `equilibre(noeud* x)` : équilibre l’arbre en utilisant une procédure de type “diviser pour régner” (voir TD Arbres Equilibrés).

Pour le constructeur “AVL(char\* filename, bool option)”, les fichiers, dans lesquels les données sont lues, auront le format suivant :

- ligne 1 : nombre d’entiers à lire
- ligne 2 : liste des entiers

Par exemple, le fichier “donnees.txt” contenant :

- 8
- 1 2 3 4 6 7 8 9

désigne 8 entiers à lire se trouvant sur la seconde ligne.

3/ Implémenter ces classes dans un fichier AVL.cpp.

4/ Ecrire un fichier principal “main.cpp” testant vos classes. L’exécutable sera nommé “avl.exe”. A l’appel,

*avl.exe donnees.txt*

l’exécutable devra :

- demander à l’utilisateur l’option de création d’arbre souhaitée (en feuille ou en racine),
- lire les données se trouvant dans “donnees.txt” et les insérer dans un arbre, selon l’option d’insertion choisie,
- mettre à jour l’attribut  $N$  de chaque noeud de l’arbre,
- mettre à jour l’attribut  $d$  de chaque noeud de l’arbre,
- faire un affichage “préfixe” de l’arbre,
- équilibrer l’arbre,
- refaire un affichage préfixe pour visualiser le résultat de l’équilibrage.

Notez que, dans le dernier affichage, les attributs  $N$  et  $d$  doivent être à jour.

5/ Vous devez déposer sur le lien de rendu un fichier “zip” avec toutes vos réalisations. Il contiendra :

- le fichier “avl.h”,
- le fichier “avl.cpp” contenant les implémentations commentées des méthodes,
- le fichier “main.cpp”,
- et l’exécutable “avl.exe”.

**Partie 2.** Faire le TP “AB to ABR” de la plateforme caseine.