Chapitre 6

Structures Hiérarchiques

6.1 Les arbres

6.1.1 Introduction

Dans les tableaux nous avons :

+ Un accès direct par indice (rapide)

- L'insertion et la suppression nécessitent des décalages

Dans les listes linéaires chaînées nous avons :

+ L'insertion et la suppression se font uniquement par modification de chaînage

- Accès séquentiel lent

Les arbres représentent un compromis entre les deux :

+ Un accès relativement rapide à un élément à partir de sa clé

- Ajout et suppression non coûteuses

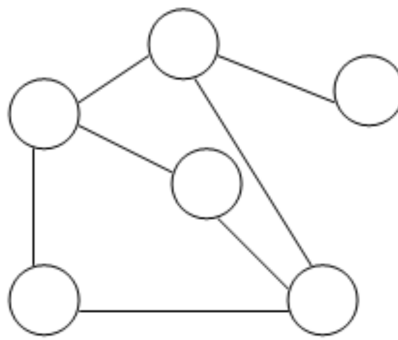
En plus plusieurs traitements en informatique sont de nature arborescente tel que les arbres généalogiques, hiérarchie des fonctions dans une entreprise, représentation des expressions arithmétiques,.. etc.

6.1.2 Définitions

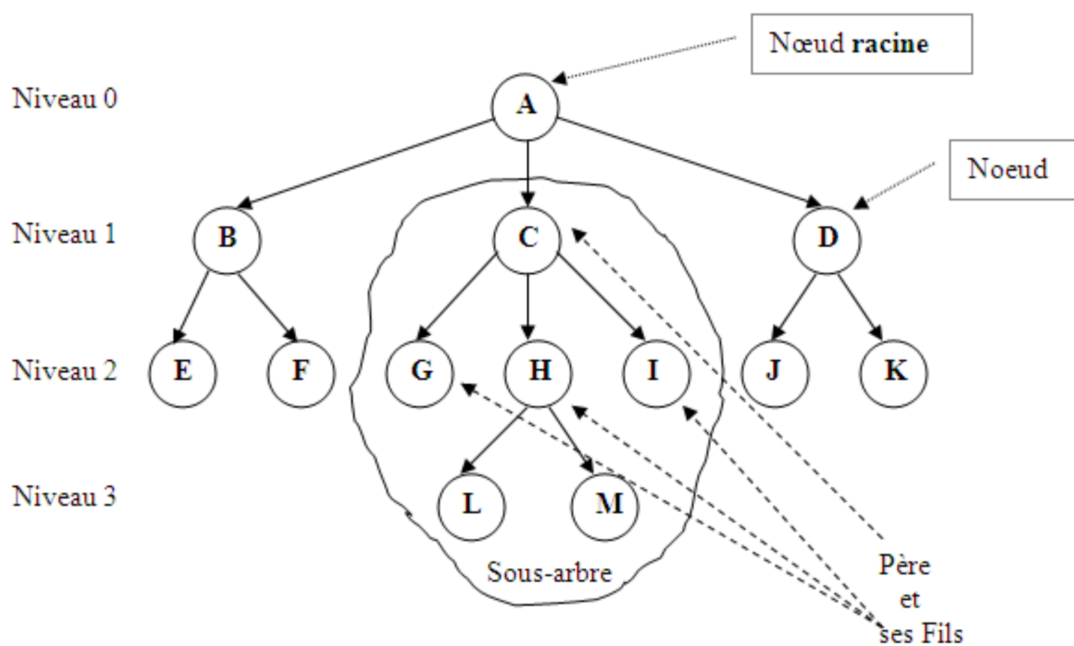
6.1.2.1 Définition d'un arbre

Un arbre est une structure non linéaire, c'est un graphe sans cycle où chaque nœud a au plus un prédécesseur.

Graphe



Arbre



- Le prédécesseur s'il existe s'appelle père (père de C = A, père de L = H)
- Le successeur s'il existe s'appelle fils (fils de A = { B,C,D }, fils de H= {L,M })
- Le nœud qui n'a pas de prédécesseur s'appelle racine (A)
- Le nœud qui n'a pas de successeur s'appelle feuille (E,F,G,L,J,...)
- Descendants de C={G,H,I,L,M}, de B={E,F},...
- Ascendants de L={H,C,A t}, E={B,A},...

6.1.2.2 Taille d'un arbre

C'est le nombre de nœuds qu'il possède.

- Taille de l'arbre précédent = 13
- Un arbre vide est de taille égale à 0.

6.1.2.3 Niveau d'un nœud

- Le niveau de la racine = 0
- Le niveau de chaque nœud est égale au niveau de son père plus 1
- Niveau de E,F,G,H,I,J,K = 2

6.1.2.4 Profondeur (Hauteur) d'un arbre

- C'est le niveau maximum dans cet arbre.
- Profondeur de l'arbre précédent = 3

6.1.2.5 Degré d'un nœud

- Le degré d'un nœud est égal au nombre de ses fils.
- Degré de (A = 3, B =2, C = 3, E= 0, H=2,...)

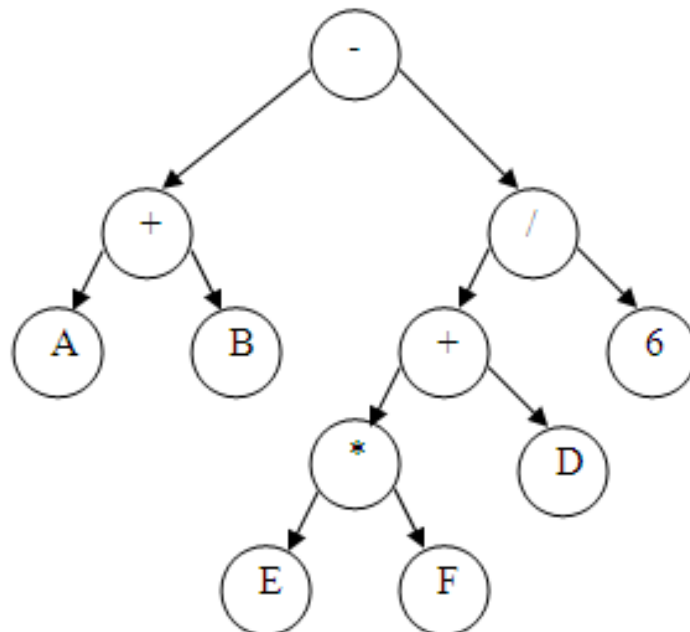
6.1.2.6 Degré d'un arbre

- C'est le degré maximum de ses nœuds.
- Degré de l'arbre précédent = 3.

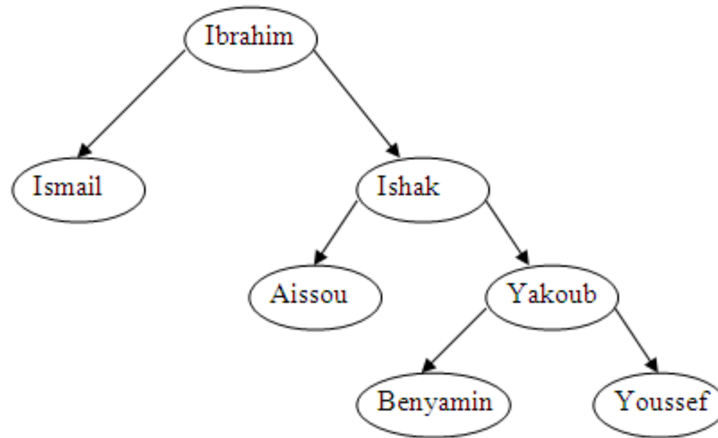
6.1.3 Utilisation des arbres

- Représentation des expressions arithmétiques

(A+B)*c - (d+E*f)/6



– Représentation d'un arbre généalogique



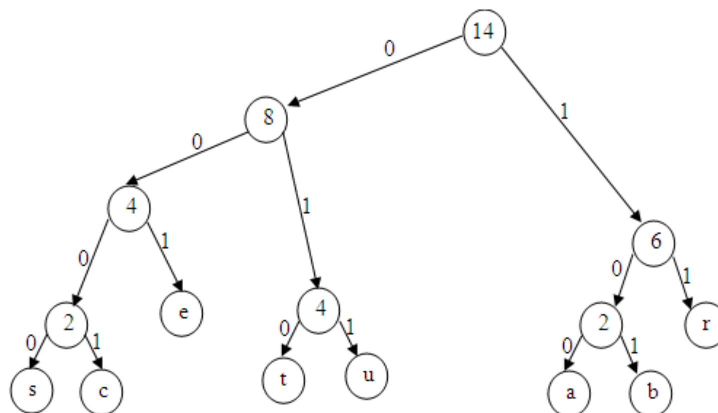
– Codage

Exemple : coder la chaine "structure arbre"

1. Construite la table des fréquences des caractères

Caractère	Fréquence	Caractère	Fréquence
s	1	c	1
t	2	e	2
r	4	a	1
u	2	b	1

2. Construite l'arbre des codes



3. Construire la table des codes

Caractère	Code	Caractère	Code
s	0000	c	0001
t	010	e	001
r	11	a	100
u	011	b	101

4. Coder la chaine :

   "structure arbre" ⇒ 0000 010 11 011 0001 010 011 11 001 100 11 101 11 001

6.1.4 Implémentation des arbre

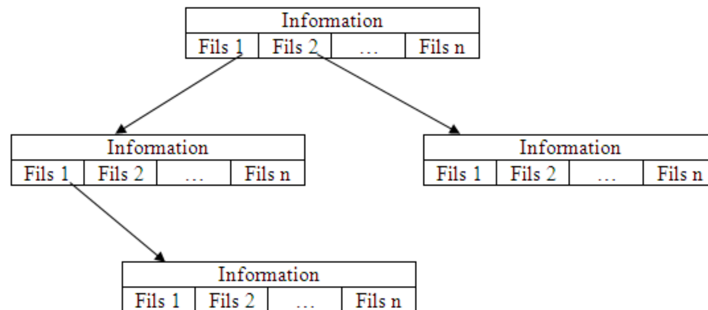
Les arbres peuvent être représentés par des tableaux, des listes non linéaires ou tous les deux :

6.1.4.1 Représentation statique

L'arbre du premier exemple peut être représenté par un tableau comme suit :

Num	Information	Fils 1	Fils 2	Fils 3
1	A	2	3	4
2	B	5	6	0
3	C	7	8	9
4	D	10	11	0
5	E	0	0	0
6	F	0	0	0
7	G	0	0	0
8	H	12	13	0
9	I	0	0	0
10	J	0	0	0
11	K	0	0	0
12	L	0	0	0
13	M	0	0	0

6.1.4.2 Représentation dynamique



Type Tnœud = **Structure**

Info : **typeqq ;**

Fils : **Tableau[1..NbFils] de Pointeur(Tnœud) ;**

Fin ;

Var Racine : **Pointeur(Tnœud) ;**

6.1.5 Modèle sur les arbres

Pour manipuler les structures de type arbre, on aura besoin des primitives suivantes :

- **Allouer (N) :** créer une structure de type TNœud et rendre son adresse dans N.
- **Liberer(N) :** libérer la zone pointée par N.
- **Aff_Val(N, Info) :** Ranger la valeur de Info dans le champs info du nœud pointé par N.
- Aff_Fils_i**(N1,N2) :** Rendre N2 le fils numéro i de N1.
- $Fils_i$**(N) :** donne le fils numéro i de N.
- **Valeur(N) :** donne le contenu du champs info du nœud pointé par N.

6.1.6 Traitements sur les arbres

6.1.6.1 Parcours des arbres

Le parcours d'un arbre consiste à passer par tous ses nœuds. Les parcours permettent d'effectuer tout un ensemble de traitements sur les arbres. On distingue deux types de parcours :

1. Parcours en profondeur

   Dans un parcours en profondeur, on descend le plus profondément possible dans l'arbre puis, une fois qu'une feuille a été atteinte, on remonte pour explorer les autres branches en commençant par la branche "la plus basse" parmi celles non encore parcourues. L'algorithme est le suivant :

```
Procédure PP( nœud : Pointeur(Tnœud));
Début
    Si (nœud ≠ Nil) Alors
        Pour i de 1 à NbFils faire
            PP(Fils_i(nœud))
        Fin Pour;
    Fin Si;
Fin;
```

   Le parcours en profondeur peut se faire en deux manières :
   - Parcours en profondeur Prefixe : où on affiche le père avant ses fils.
   - Parcours en profondeur Postfixe : où on affiche les fils avant leur père.

   Les algorithmes récursifs correspondant sont les suivants :

```
Procédure PPPrefixe( nœud : Pointeur(Tnœud));
Début
    Si (nœud ≠ Nil) Alors
        Afficher(Valeur(nœud)) ;
        Pour i de 1 à NbFils faire
            PPPrefixe(Fils_i(nœud))
        Fin Pour;
    Fin Si;
Fin;
```

```
Procédure PPPostfixe( nœud : Pointeur(Tnœud));
Début
    Si (nœud ≠ Nil) Alors
        Pour i de 1 à NbFils faire
            PPPostfixe(Fils_i(nœud))
        Fin Pour;
        Afficher(Valeur(nœud)) ;
    Fin Si;
Fin;
```

Le parcours en profondeur préfixe de l'arbre du premier exemple donne :

A,B,E,F,C,G,H,L,M,I,D,J,K

Tandis que le parcours en profondeur postfixe donne :

E,F,B,G,L,M,H,I,C,J,K,D,A

2. Parcours en largeur

   Dans un parcours en largeur, tous les nœuds à une profondeur i doivent avoir été visités avant que le premier nœud à la profondeur i + 1 ne soit visité. Un tel parcours nécessite que l'on se souvienne de l'ensemble des branches qu'il reste à visiter. Pour ce faire, on utilise une file d'attente.

```
Procédure PL( nœud : Pointeur(Tnœud));
Var  N : Pointeur(Tnœud) ;
Début
  Si (nœud ≠ Nil) Alors
    InitFile ;
    Enfiler(nœud) ;
    Tant que (Non(FileVide)) faire
      Défiler(N) ;
      Afficher(Valeur(N)) ;
      Pour i de 1 à NbFils faire
        Si (Fils_i(N) ≠ Nil) Alors
          Enfiler(Fils_i(N)) ;
        Fin Si;
      Fin Pour;
    Fin TQ;
  Fin Si;
Fin;
```

L'application de cet algorithme sur l'arbre du premier exemple donne

A,B,C,D,E,F,G,H,I,J,K,L,M

6.1.6.2 Recherche d'un élément

```
Fonction Rechercher( nœud : Pointeur(Tnœud) ; Val : Typeqq) : Booleen;
Var  i : entier ;
     Trouv : Booleen ;
Début
  Si (nœud = Nil) Alors
    Rechercher ← Faux ;
  Sinon
    Si (Valeur(nœud)=Val) Alors
      Rechercher ← Vrai ;
    Sinon
      i ← 1 ;
      Trouv ← Faux ;
      Tant que ((i ≤ NbFils) et non Trouv) faire                ;
        Trouv ← Rechercher(Fils_i(Neoud), Val) ;
        i ← i + 1 ;
      Fin TQ;
      Rechercher ← Trouv ;
    Fin Si;
  Fin Si;
Fin;
```

6.1.6.3 Calcul de la taille d'un arbre

```
Fonction Taille( nœud : Pointeur(Tnœud)) : entier;
Var  i,S : entier ;
Début
    Si (nœud = Nil) Alors
        Taille ← 0 ;
    Sinon
        S ← 1 ;
        Pour i de 1 à NbFils faire
            S ← S + Taille(Fils_i(nœud)) ;
        Fin Pour;
        Taille ← S ;
    Fin Si;
Fin;
```

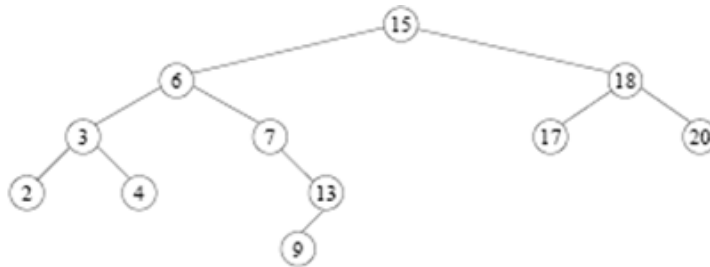
6.1.7 Typologie des arbres

- **Arbre m-aire :** un arbre m-aire d'ordre n est un arbre ou le degré maximum d'un nœud est égal à m.
- **B-Arbre :** Un arbre B d'ordre n est un arbre où :
  - la racine a au moins 2 fils
  - chaque nœud, autre que la racine, a entre n/2 et n fils
  - tous les nœuds feuilles sont au même niveau
- **Arbre binaire :** c'est un arbre ou le degré maximum d'un nœud est égal à 2.
- **Arbre binaire de recherche :** c'est un arbre binaire où la clé de chaque nœud est supérieure à celles de ses descendants gauche, et inférieure à celles de ses descendants droits.

6.2 Les arbres binaires de recherche

6.2.1 Définition

Les arbres binaires de recherche sont utilisés pour accélérer la recherche dans les arbres m-aires. Un arbre binaire de recherche est un arbre binaire vérifiant la propriété suivante : soient x et y deux nœuds de l'arbre, si y est un nœud du sous-arbre gauche de x, alors $clé(y) \leq clé(x)$, si y est un nœud du sous-arbre droit de x, alors $clé(y) \geq clé(x)$.



Un nœud a, donc, au maximum un fils gauche et un fils droit.

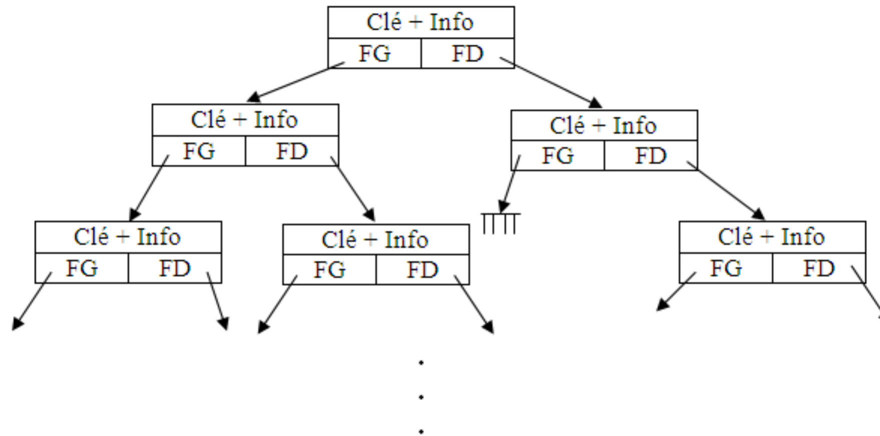
6.2.2 Implémentation des ABR

Les arbres de recherche binaires sont implémentés de la même manière que celles m-aires (statique ou dynamique)

6.2.2.1 Représentation Statique

Num	Information	Fils gauche	Fils droit
1	15	2	3
2	6	4	5
3	18	6	7
4	3	8	9
5	7	0	10
6	17	0	0
7	20	0	0
8	2	0	0
9	4	0	0
10	13	11	0
11	9	0	0

6.2.2.2 Représentation dynamique



Type TNoeud = **Structure**

Clé : **entier ;**

Info : **typeqq ;**

FG,FD : **Pointeur(TNoeud) ;**

Fin ;

Var Racine : **Pointeur(TNoeud) ;**

6.2.3 Modèle sur les ABR

- **Allouer (N) :** créer une structure de type TNoeud et rendre son adresse dans N.
- **Liberer(N) :** libérer la zone pointée par N.
- **Aff_Val(N, Info) :** Ranger la valeur de Info dans le champs info du nœud pointé par N.
- **Aff_Clé(N, Clé) :** Ranger la valeur de Clé dans le champs Clé du nœud pointé par N.
- **Aff_FG(N1,N2) :** Rendre N2 le fils gauche de N1.
- **Aff_FD(N1,N2) :** Rendre N2 le fils droit de N1.
- **FG(N) :** donne le fils gauche de N.
- **FD(N) :** donne le fils droit de N.
- **Valeur(N) :** donne le contenu du champs info du nœud pointé par N.
- **Clé(N) :** donne le contenu du champs Clé du nœud pointé par N.

6.2.4 Traitements sur les ABR

6.2.4.1 Parcours

De même que pour les arbres m-aire le parcours des ARB peut se faire en profondeur ou en largeur :

- En profondeur

```
Procédure PP( noeud : Pointeur(TNoeud));
Début
    Si (nœud ≠ Nil) Alors
        PP(FG(noeud)) ;
        PP(FD(noeud)) ;
    Fin Si;
Fin;
```

Le listage des éléments de l'arbre en profondeur peut se faire en :

- préfixe (préordre) : Pére FG FD,
- infixe (inordre) : FG Père FD,
- postfixe (postordre) : FG FD Père.

```
Procédure PPréfixe( noeud : Pointeur(TNoeud));
Début
    Si (nœud ≠ Nil) Alors
        Ecrire(Valeur(noeud)) ;
        PPréfixe(FG(noeud)) ;
        PPréfixe(FD(noeud)) ;
    Fin Si;
Fin;
```

Trace : 15 6 3 2 4 7 13 9 18 17 20

```
Procédure Infixe( noeud : Pointeur(TNoeud));
Début
    Si (nœud ≠ Nil) Alors
        Infixe(FG(noeud)) ;
        Ecrire(Valeur(noeud)) ;
        Infixe(FD(noeud)) ;
    Fin Si;
Fin;
```

Trace : 2 3 4 6 7 9 13 15 17 18 20

```
Procédure Postfixe( noeud : Pointeur(TNoeud));
Début
    Si (nœud ≠ Nil) Alors
        Postfixe(FG(noeud)) ;
        Postfixe(FD(noeud)) ;
        Ecrire(Valeur(noeud)) ;
    Fin Si;
Fin;
```

Trace : 2 4 3 9 13 7 6 17 20 18 15

- En largeur

```
Procédure PL( noeud : Pointeur(TNoeud));
Var  N : Pointeur(TNoeud) ;
Début
    Si (noeud ≠ Nil) Alors
        InitFile ;
        Enfiler(noeud) ;
        Tant que (Non(FileVide)) faire
            Défiler(N) ;
            Afficher(Valeur(N)) ;
            Si (FG(N) ≠ Nil) Alors
                Enfiler(FG(N)) ;
            Fin Si;
            Si (FD(N) ≠ Nil) Alors
                Enfiler(FD(N)) ;
            Fin Si;
        Fin TQ;
    Fin Si;
Fin;
```

6.2.4.2 Recherche

```
Fonction Rechercher( noeud : Pointeur(TNoeud) ; xClé : entier) : Pointeur(TNoeud);
Var i : entier ;        Trouv : Booleen ;
Début
    Si ((noeud = Nil) ou Clé(noeud)=xClé)) Alors
        Rechercher ← nœud ;
    Sinon
        Si (Clé(noeud) > xClé ) Alors
            Rechercher ← Rechercher(FG(noeud)) ;
        Sinon
            Rechercher ← Rechercher(FD(noeud)) ;
        Fin Si;
    Fin Si;
Fin;
```

6.2.4.3 Insertion

L'élément à ajouter est inséré là où on l'aurait trouvé s'il avait été présent dans l'arbre. L'algorithme d'insertion recherche donc l'élément dans l'arbre et, quand il aboutit à la conclusion que l'élément n'appartient pas à l'arbre (l'algorithme aboutit sur NIL), il insère l'élément comme fils du dernier nœud visité.

6.2.4.4 Suppression

Plusieurs cas de figure peuvent être trouvés : soit à supprimer le nœud N :

Cas			Action
FG(N)	**FD(N)**	**Exemple**	
Nil	Nil	Feuille (2,4,17)	Remplacer N par Nil
Nil	$\neq$ Nil	7	Remplacer N par FD(N)
$\neq$ Nil	Nil	13	Remplacer N par FG(N)
$\neq$ Nil	$\neq$ Nil	6	1- Rechercher le plus petit descendant à droite de N soit P (7) 2- Remplacer Valeur(N) par Valeur(P) (6 $\leftarrow$7) 3- Remplacer P par FD (P) (7 $\leftarrow$13)

Exercice : Donner l'arbre après la suppression de 3 puis de 15.

6.2.5 Equilibrage

Soit les deux ARB suivants :

Ces deux ARB contiennent les mêmes éléments, mais sont organisés différemment. La profondeur du premier est inférieure à celle du deuxième. Si on cherche l'élément 10 on devra parcourir 3 éléments (50, 20, 10) dans le premier arbre, par contre dans le deuxième, on devra parcourir 5 élément (80, 70, 50, 20,10). On dit que le premier arbre est plus équilibré.
Si on calcule la complexité de l'algorithme de recherche dans un arbre de recherche binaire, on va trouver O(h) ou h est la hauteur de l'arbre. Donc plus l'arbre est équilibré, moins élevée est la hauteur et plus rapide est la recherche.
On dit qu'un ARB est équilibré si pour tout nœud de l'arbre la différence entre la hauteur du sous-arbre gauche et du sous-arbre droit est d'au plus égalé à 1. Il est conseillé toujours de travailler sur un arbre équilibré pour garantir une recherche la plus rapide possible. L'opération d'équilibrage peut être faite à chaque fois qu'on insère un nouveau nœud où à chaque fois que le déséquilibre atteint un certain seuil pour éviter le coût de l'opération d'équilibrage qui nécessite une réorganisation de l'arbre.

6.3 Les tas (Heaps)

6.3.1 Introduction

Pour implémenter une file d'attente avec priorité, souvent utilisée dans les systèmes d'exploitation, on peut utiliser :

- Une file d'attente ordinaire (sans priorité), l'insertion sera alors simple (à la fin) en O(1), mais le retrait nécessitera la recherche de l'élément le plus prioritaire, en O(n).
- Un tableau (ou une liste) trié où le retrait sera en O(1) (le premier élément), mais l'insertions nécessitera O(n).

Les tas apportent la solution à ce problème.

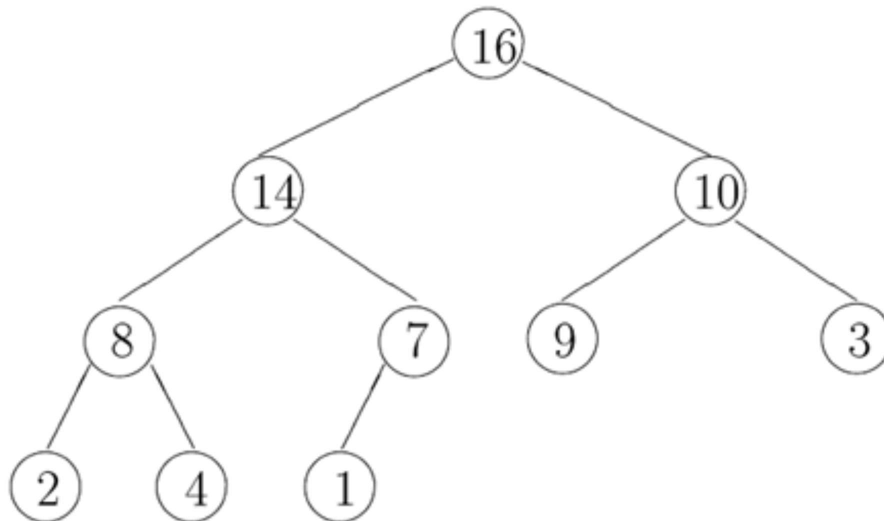
6.3.2 Définition

Un tas (*heap* en anglais) est un arbre qui vérifie les deux propriétés suivantes :

1. C'est un arbre binaire complet c'est-à-dire un arbre binaire dont tous les niveaux sont remplis sauf éventuellement le dernier où les éléments sont rangés le plus à gauche possible.
2. La clé de tout nœud est supérieure à celle de ses descendants.

L'élément le plus prioritaire se trouve donc toujours à la racine.

Exemple d'un tas :



6.3.3 Opérations sur les tas

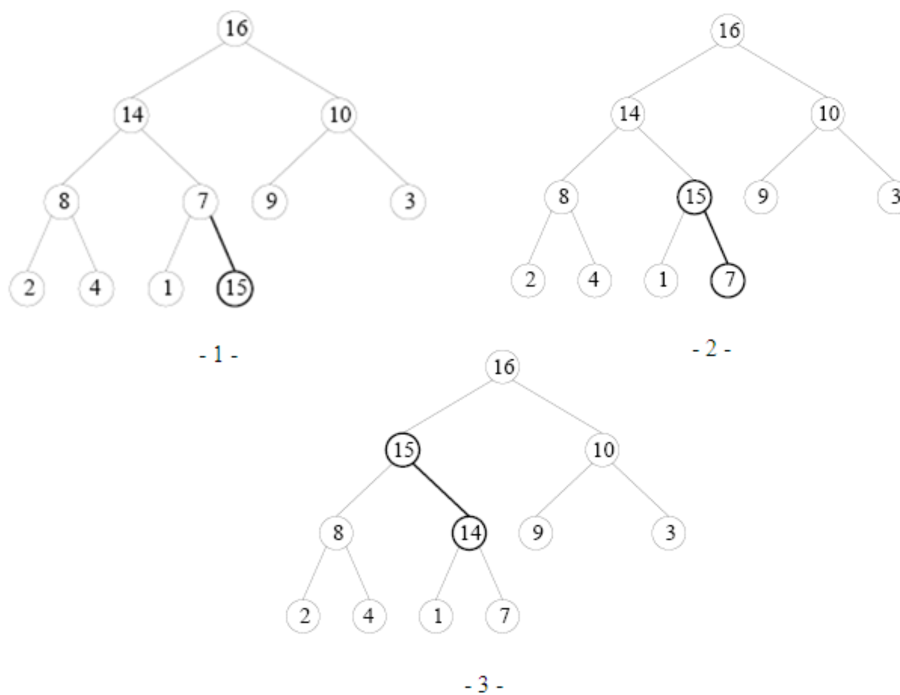
Pour coder une file d'attente avec priorité par un tas, on doit définir les opérations d'ajout et de retrait.

6.3.3.1 Ajout

Pour ajouter un nouvel élément dans la file avec priorité c-à-d dans le tas on doit :

1. Créer un nœud contenant la valeur de cet élément,
2. Attacher ce nœud dans le dernier niveau dans la première place vide le plus à gauche possible (créer un nouveau niveau si nécessaire). On obtient toujours un arbre binaire complet mais pas nécessairement un tas.
3. Comparer la clé du nouveau nœud avec celle de son père et les permuter si nécessaire, puis recommencer le processus jusqu'il n'y ait plus d'éléments à permuter.

Exemple : soit à ajouter l'élément de priorité 15 :



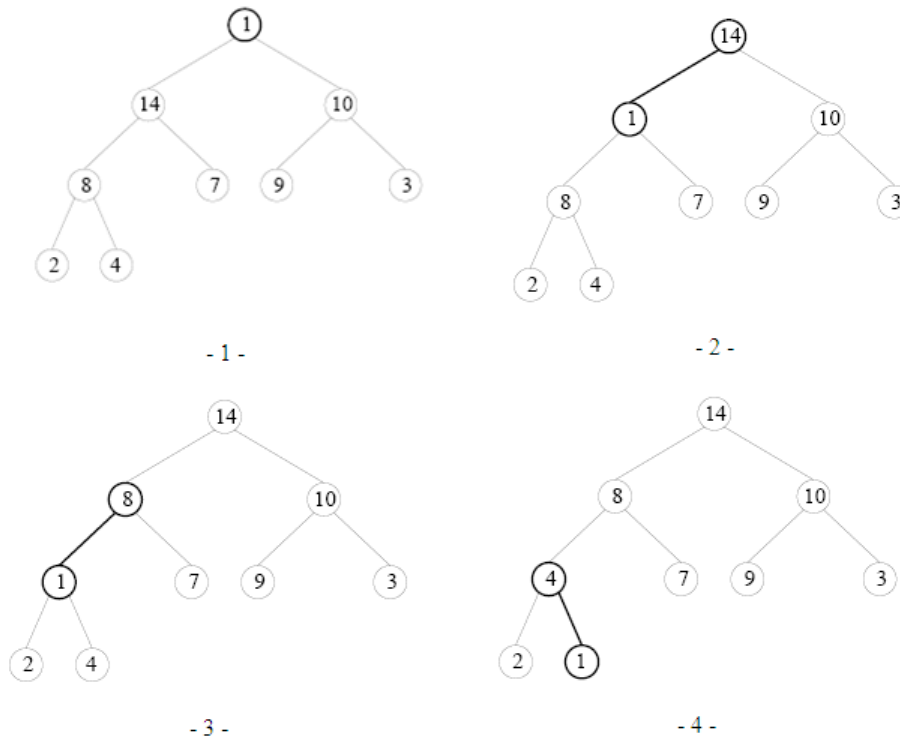
La complexité de cette opération est de $O(h = Log_2(n))$ si n est le nombre d'éléments.

6.3.3.2 Retrait

L'élément le plus prioritaire se trouve toujours à la racine, donc le retrait consiste à lire la racine puis la supprimer. Pour ce faire on doit :

1. Remplacer la valeur de la racine par la valeur de l'élément le plus à droite dans le dernier niveau.
2. Supprimer de l'arbre cet élément (le plus à droite dans le dernier niveau), on obtient un arbre binaire mais pas nécessairement un tas.
3. On compare la valeur de la racine avec les valeurs de ses deux fils et on la permute avec la plus grande. On recommence le processus jusqu'il n'y ait plus d'éléments à permuter.

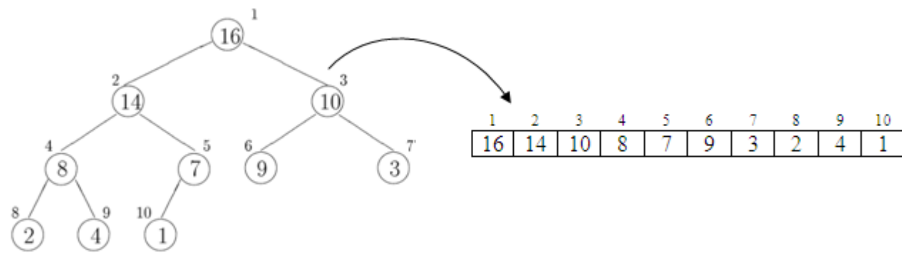
Exemple :



La suppression est aussi en $O(h = Log_2(n))$.

6.3.4 Implémentation des tas

Les tas peuvent être implémentés dynamiquement exactement comme les ARB, et sont utilisés par le même modèle.
Une représentation statique très efficace utilisant des tableaux est très utilisée en pratique, elle consiste à ranger les éléments du tas dans un tableau selon un parcours en largeur :



On remarque sur le tableau obtenu que le fils gauche d'un élément d'indice i se trouve toujours s'il existe à la position $2i$, et son fils droit se trouve à la position $(2i + 1)$ et son père se trouve à la position $i/2$. Les opérations d'ajout et de retrait sur le tas statique se font de la même façon que dans le cas du tas dynamique. Avec ce principe les opérations d'ajout et de retrait se font d'une manière très simple et extrêmement efficace.
Les tas sont utilisés même pour le tri des tableaux : on ajoute les éléments d'un tableau à un tas, puis on les retire dans l'ordre décroissant.