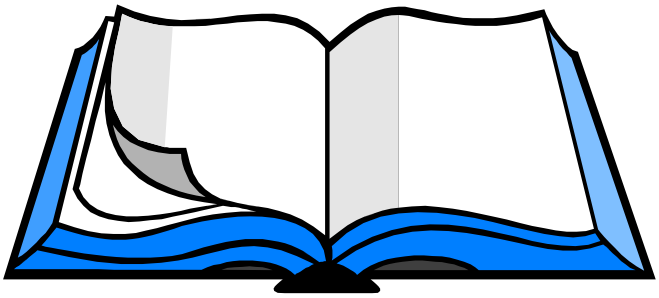
IUT de Lannion
Département Informatique
Semestre 3

Ludovic Liétard

R3.02 Développement efficace

Support de cours
(Les Arbres)



6. Structures arborescentes

6.1 Introduction

6.2 Arbres binaires

6.3 Arbres binaires de recherche

6.4 Arbres équilibrés

6.1 Introduction

Un arbre est un ensemble de *nœuds*, organisés de façon hierarchique, à partir d'un nœud particulier appelé *racine*.

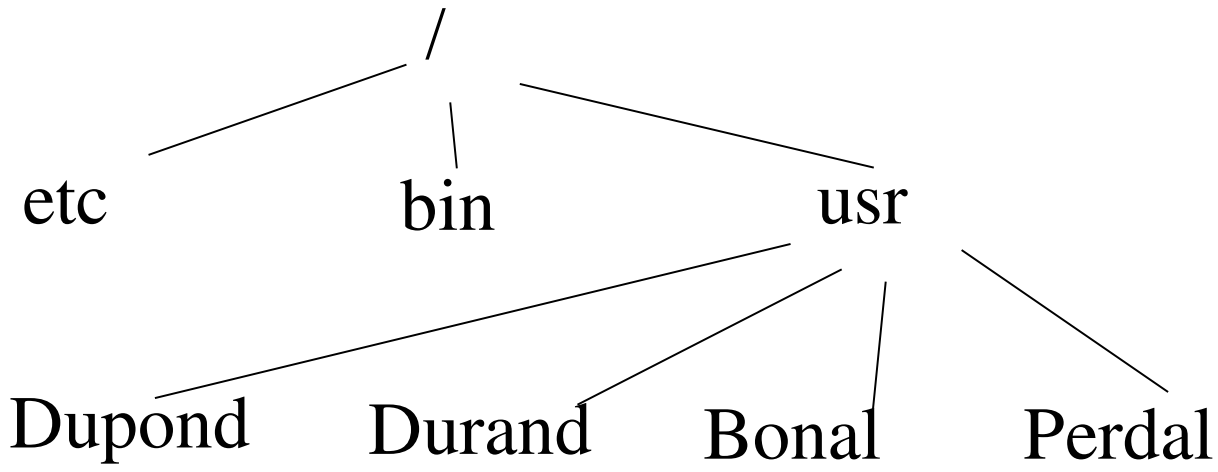
Tout nœud (sauf la *racine*) possède un unique *nœud père* (le père).

Un nœud peut ne pas avoir de *nœud fils* (de fils) ou il peut en avoir plusieurs.

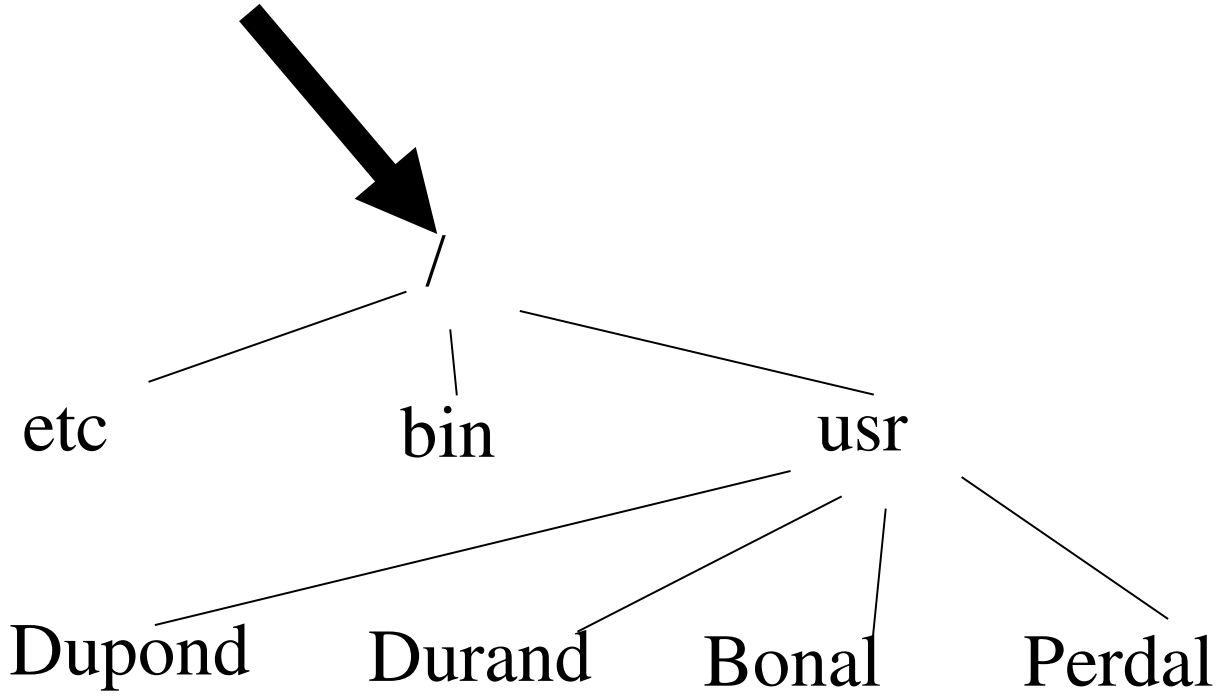
Un nœud qui n'a pas de fils est une feuille.

Exemple 1 :

On peut représenter les répertoires d'Unix par un arbre.



LA RACINE
DE L'ARBRE



La racine a 3 fils, le nœud usr a 4 fils.

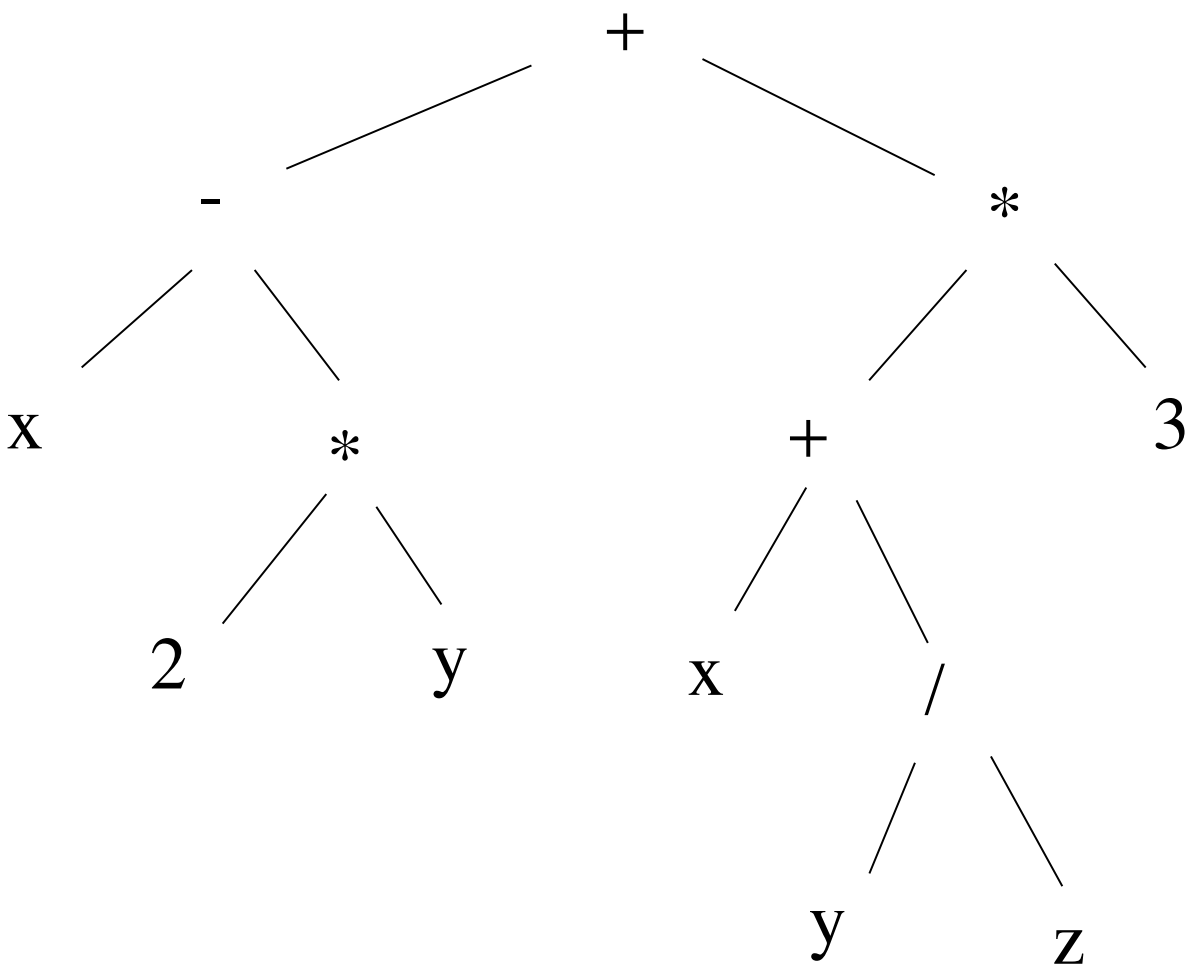
Les autres nœuds sont des feuilles.

Exemple 2 :

On peut représenter l'expression :

(x – (2 * y)) + ((x + (y/z)) * 3)

par un arbre



Taille ou le poids d'un arbre : c'est le nombre des nœuds dans l'arbre.

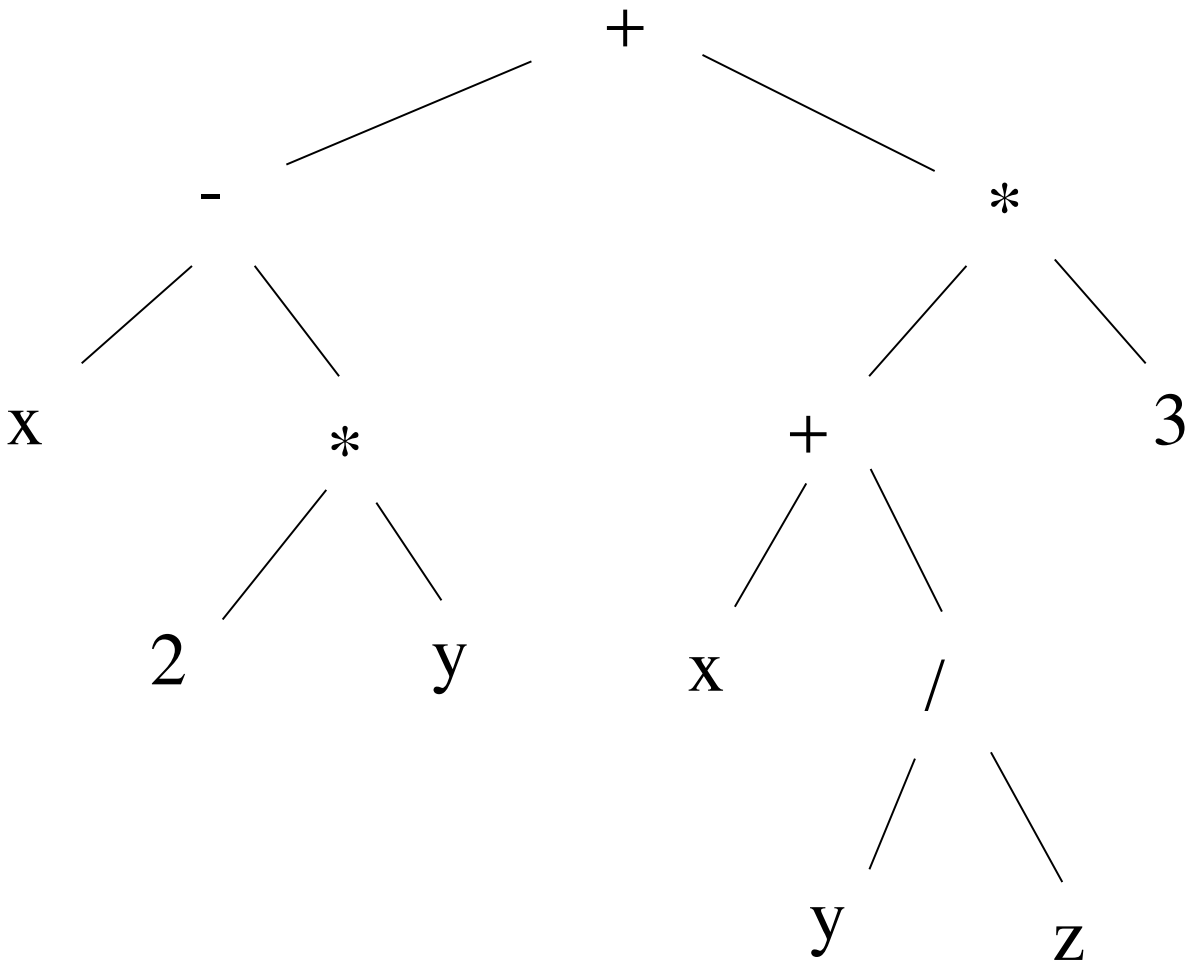
Hauteur (ou profondeur) d'un nœud :

Hauteur(racine) = 0 ;
Hauteur(x) = 1 + Hauteur(père de x).

Hauteur d'un arbre :

C'est la hauteur du nœud de plus grande hauteur.

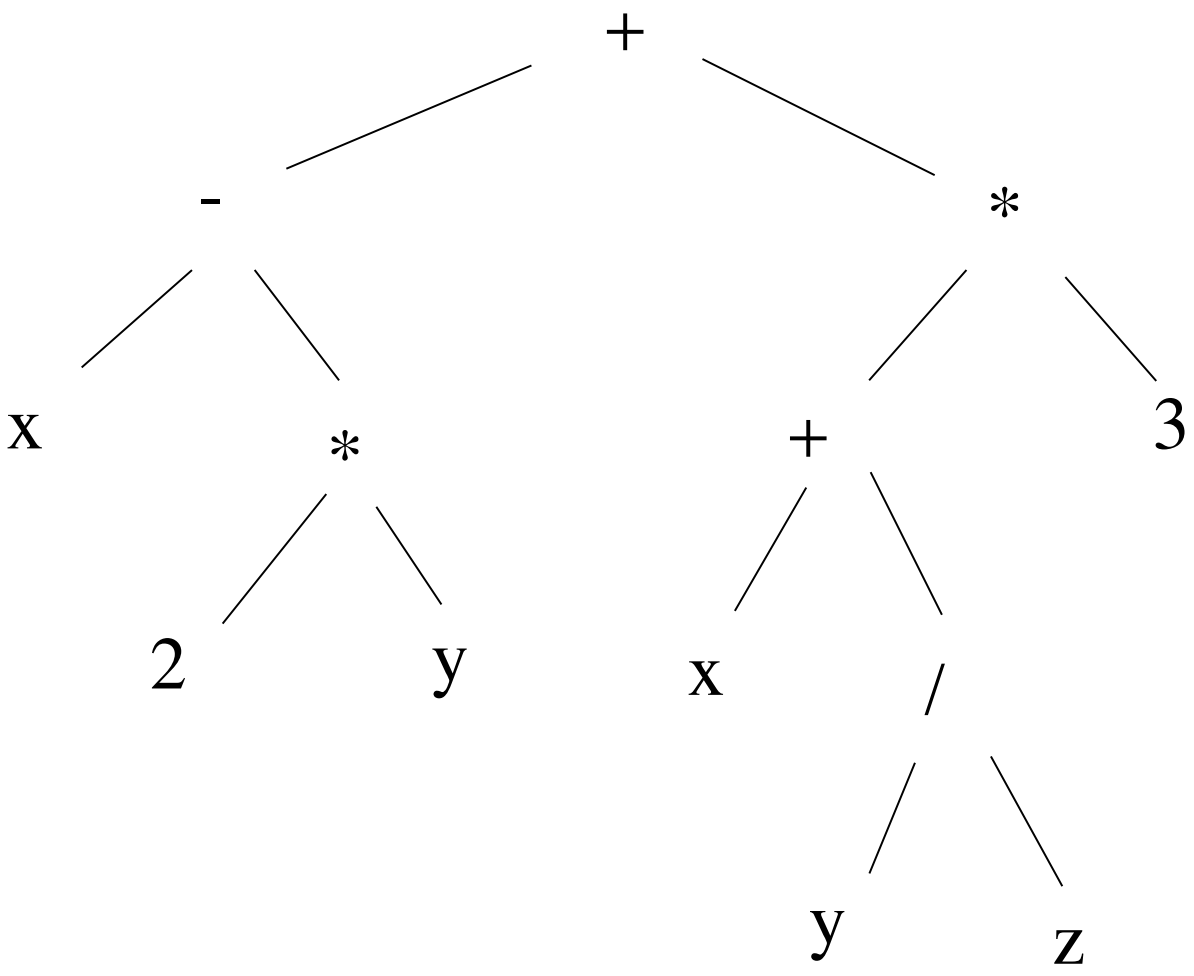
Exemple :



6.2 Arbres binaires

Un arbre binaire est un arbre dont chaque nœud a au plus deux fils.

Exemple :

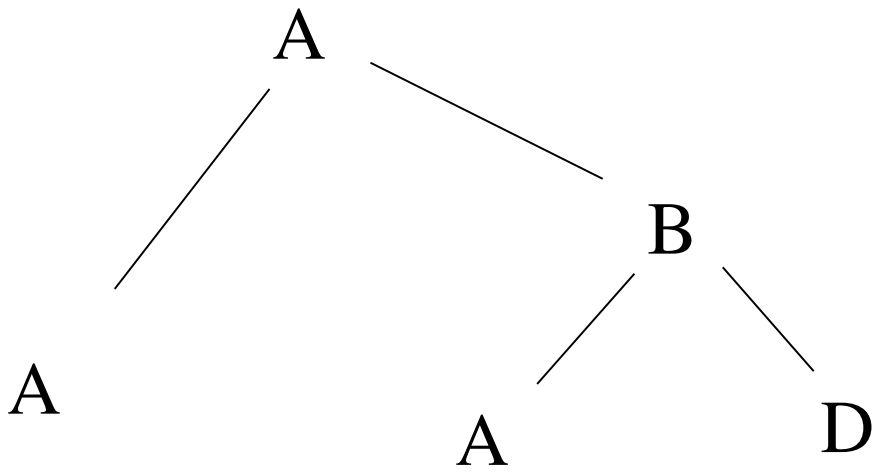


Formellement, un arbre binaire est défini récursivement comme :

1) un arbre vide est un arbre binaire,

2) un arbre dont la racine possède deux fils dont chacun est un arbre binaire.

Exemple :



6.2.1 Implémentation d'un arbre binaire

Un arbre binaire est un pointeur sur une structure (sur un nœud) qui contient la valeur ainsi que les deux sous-arbres.

```
typedef struct noeud * arbre;
typedef struct noeud {
  <type val> val;
  arbre sag;
  arbre sad;
} pnoeud;
```

où <type val> est le type des éléments dans l'arbre (par exemple : int).

L'arbre vide est la valeur NULL.

6.2.2 Parcours d'un arbre binaire

Un arbre binaire se traite naturellement par la récursivité car sa définition est récursive.

```
void proc(arbre a) {

   If (a== NULL){

            // fin traitement

    }else{

            // traiter le nœud pointé par a

              proc(a->sag) ;
              proc(a->sad) ;
    }
}
```

Selon la situation du traitement du nœud par rapport aux appels récursifs, on a un parcours préfixé ou infixé ou postfixé.

Parcours préfixé

```
void proc(arbre a) {

   If (a== NULL){
            // fin traitement
    }else{

            // traiter le nœud pointé par a
              proc(a->sag) ;
              proc(a->sad) ;
    }
}
```

Parcours infixé

```
void proc(arbre a) {

   If (a== NULL){
            // fin traitement
    }else{

            proc(a->sag) ;
            // traiter le nœud pointé par a
            proc(a->sad) ;
    }
}
```

Parcours postfixé

```
void proc(arbre a) {

   If (a== NULL){
            // fin traitement
    }else{

            proc(a->sag) ;
            proc(a->sad) ;
           // traiter le nœud pointé par a

    }
}
```

6.3 Arbres binaires de recherche

Un tel arbre contient des éléments qui sont munis d'une clé.

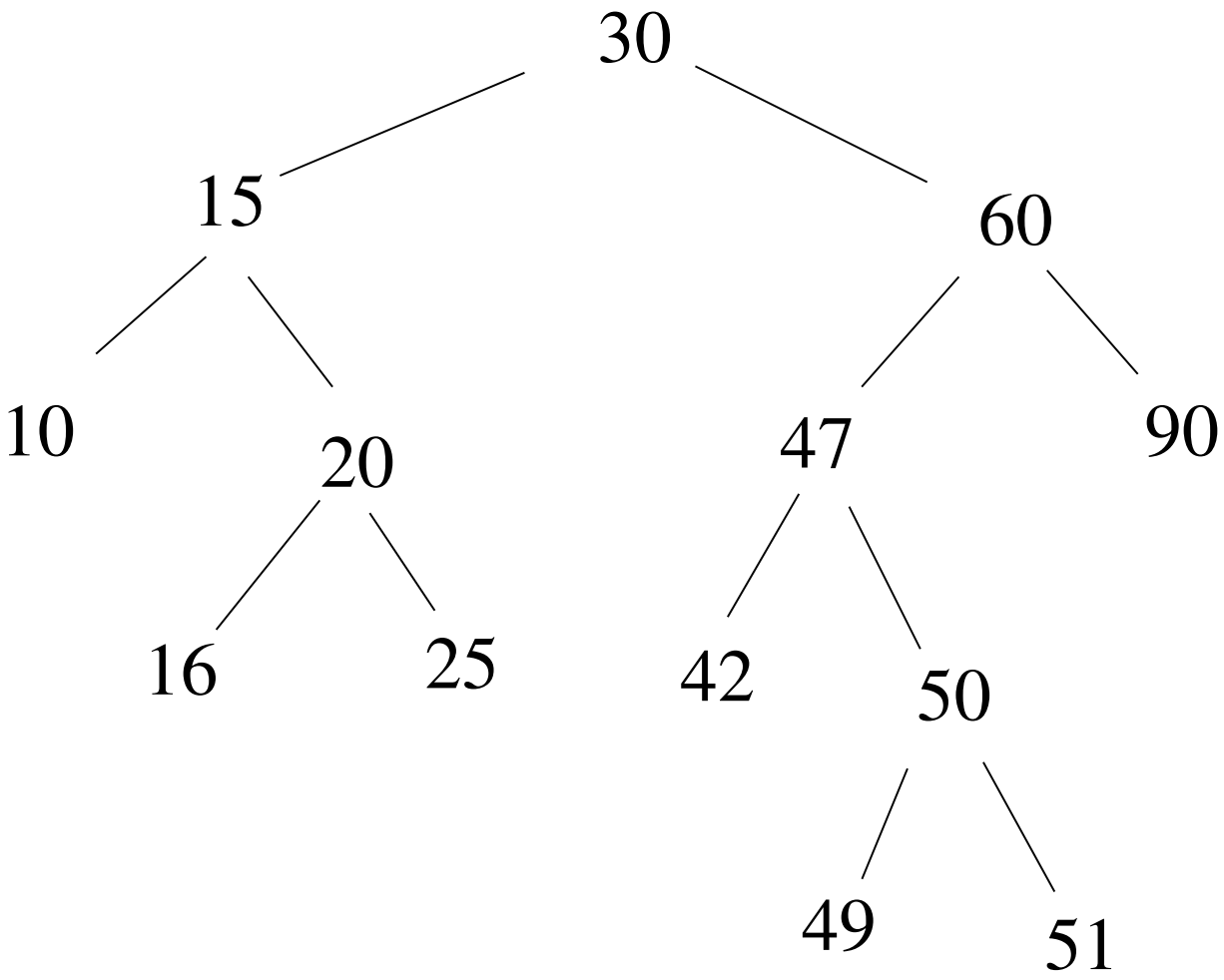
Il existe une relation d'ordre sur la clé.

Les éléments du sous-arbre gauche ont une valeur de clé inférieure à celle de la racine, les éléments du sous-arbre droit ont une valeur de clé qui est supérieure à celle de la racine (récursivement).

```
typedef struct noeud * arbre;
typedef struct noeud {
  int cle;
  <type val> val ;
  arbre sag;
  arbre sad;
} pnoeud;
```

Par la suite seules les clés sont conservées.

Exemple :



6.3.1 Recherche d'un élément dans un arbre binaire de recherche

Soit une clé K.

Quel est l'élément de l'arbre qui possède cette clé ?

On exploite l'organisation de l'arbre :

Si l'arbre est vide alors pas trouvé
sinon si la racine a cette clé alors trouvé
sinon si K < clé de la racine alors
rechercher dans le sag
sinon
rechercher dans le sad.

La fonction recherche retourne un pointeur sur le nœud de clé K ou NULL si un tel nœud n'existe pas.

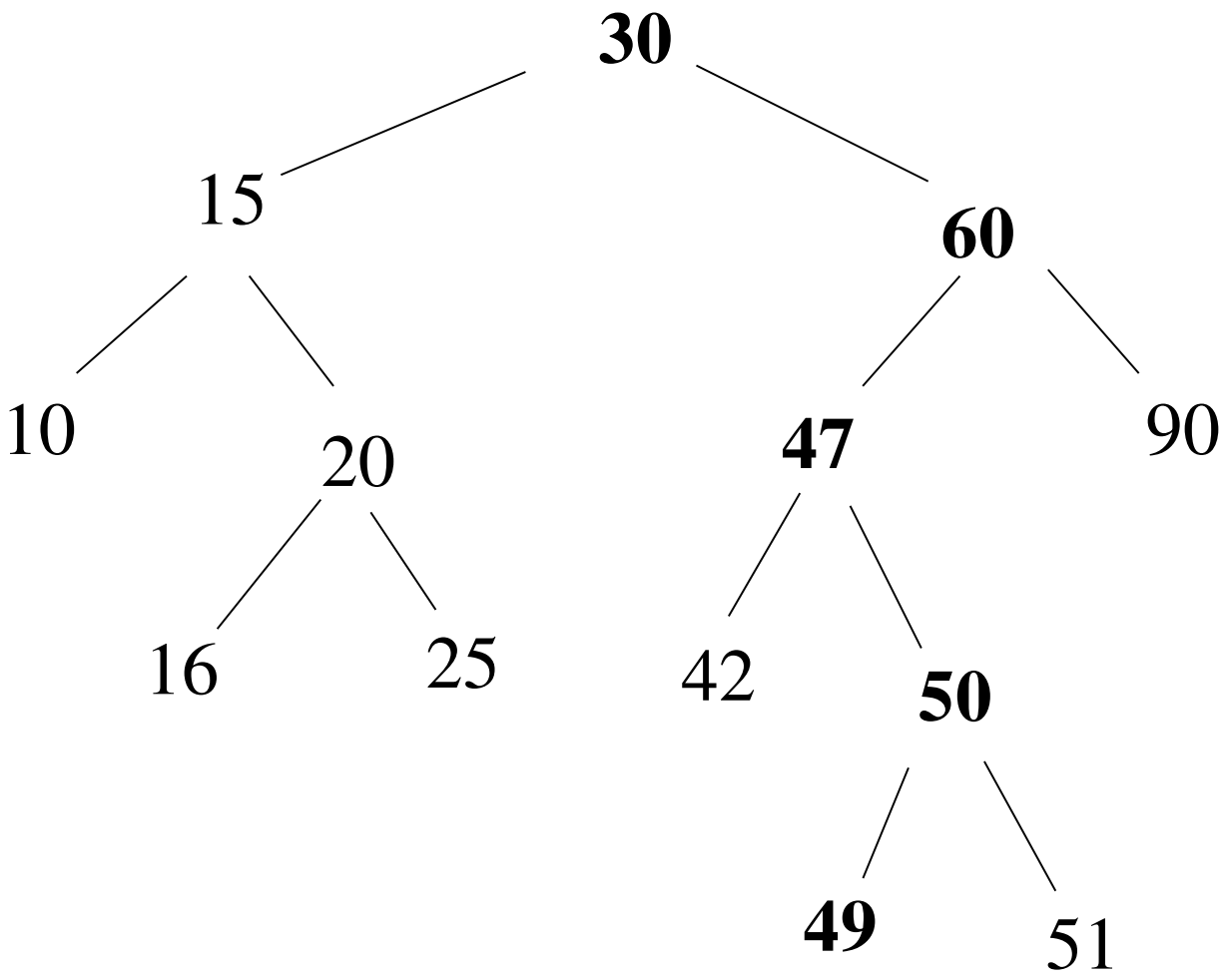
6.3.2 Ajout d'un élément dans un arbre binaire de recherche

Plusieurs algorithmes d'ajout dans un tel arbre sont possibles.

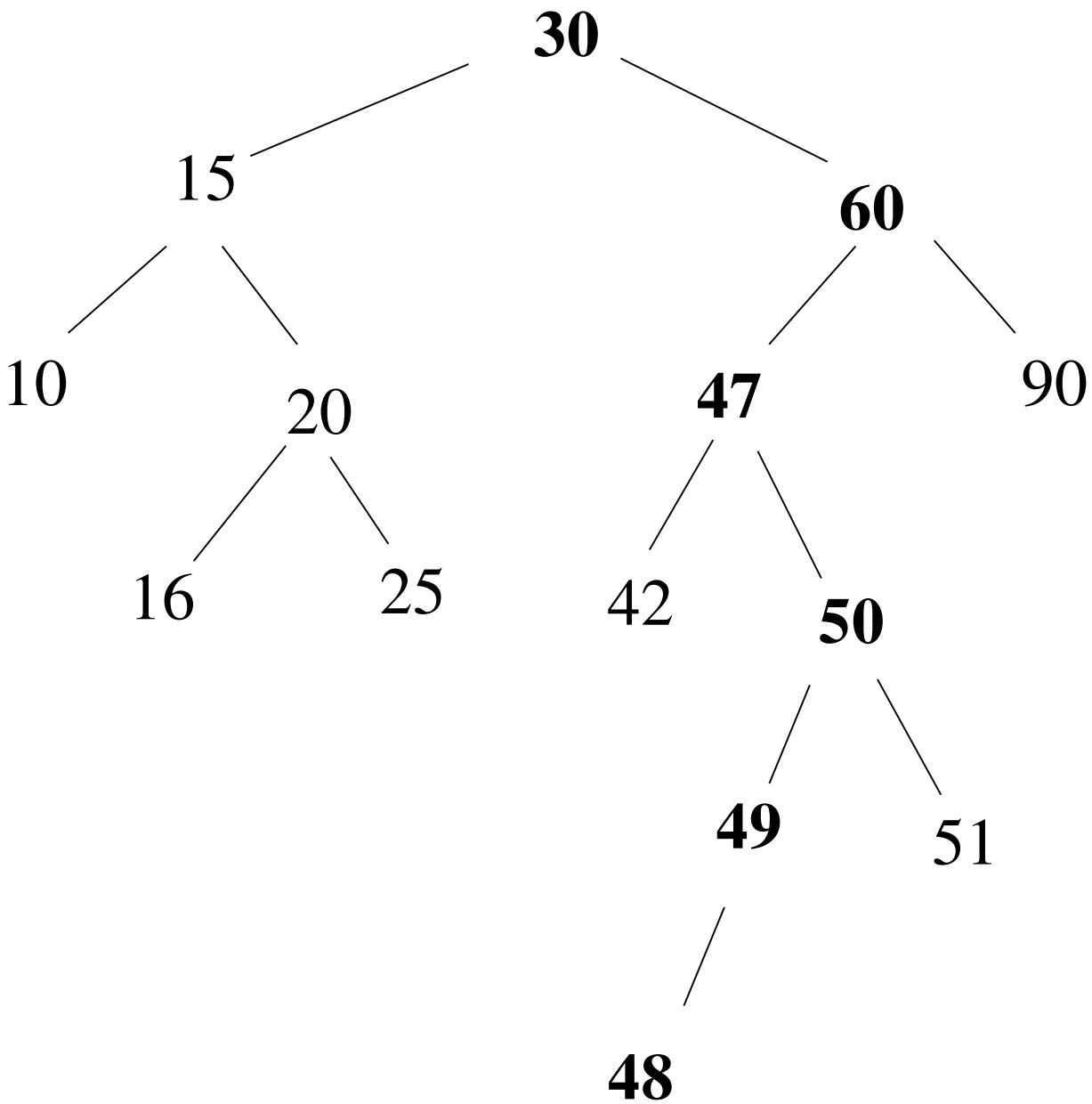
On ne verra que l'ajout en feuille.

La méthode consiste à descendre dans l'arbre jusqu'à ce que la clé soit trouvée (il n'y a rien à faire) ou jusqu'à ce que l'on soit en feuille (arbre vide). Dans ce dernier cas, on crée une nouvelle feuille.

Exemple (ajouter 48) :



Le résultat est :



6.3.3 Supression d'un élément dans un arbre binaire de recherche

Soit K la clé de l'élément à supprimer.

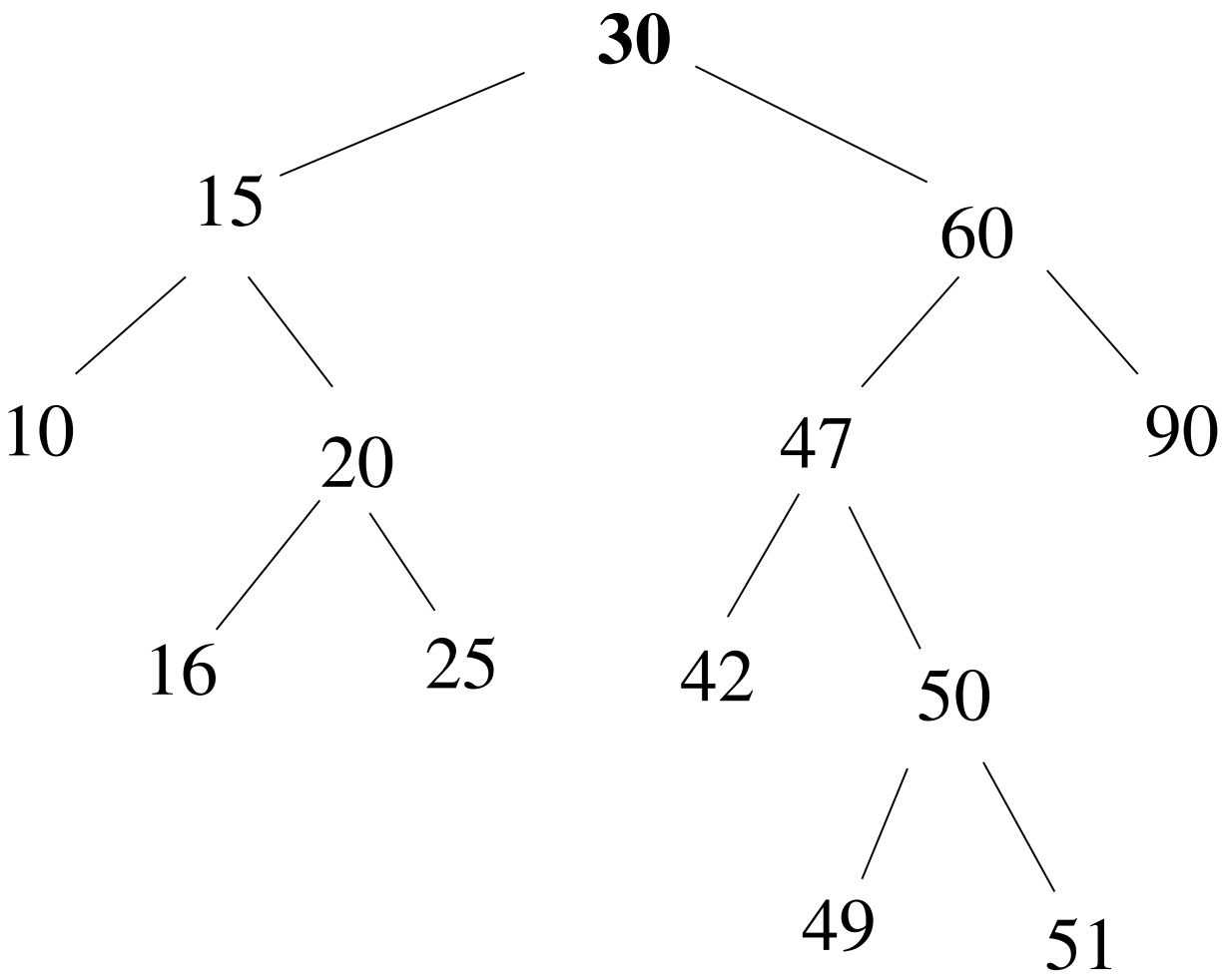
On se situe (dans l'arborescence) sur l'élément de clé K. Trois cas sont à considérer :

Cas 1 : Le nœud n'a aucun fils gauche ou droit. Il devient NULL

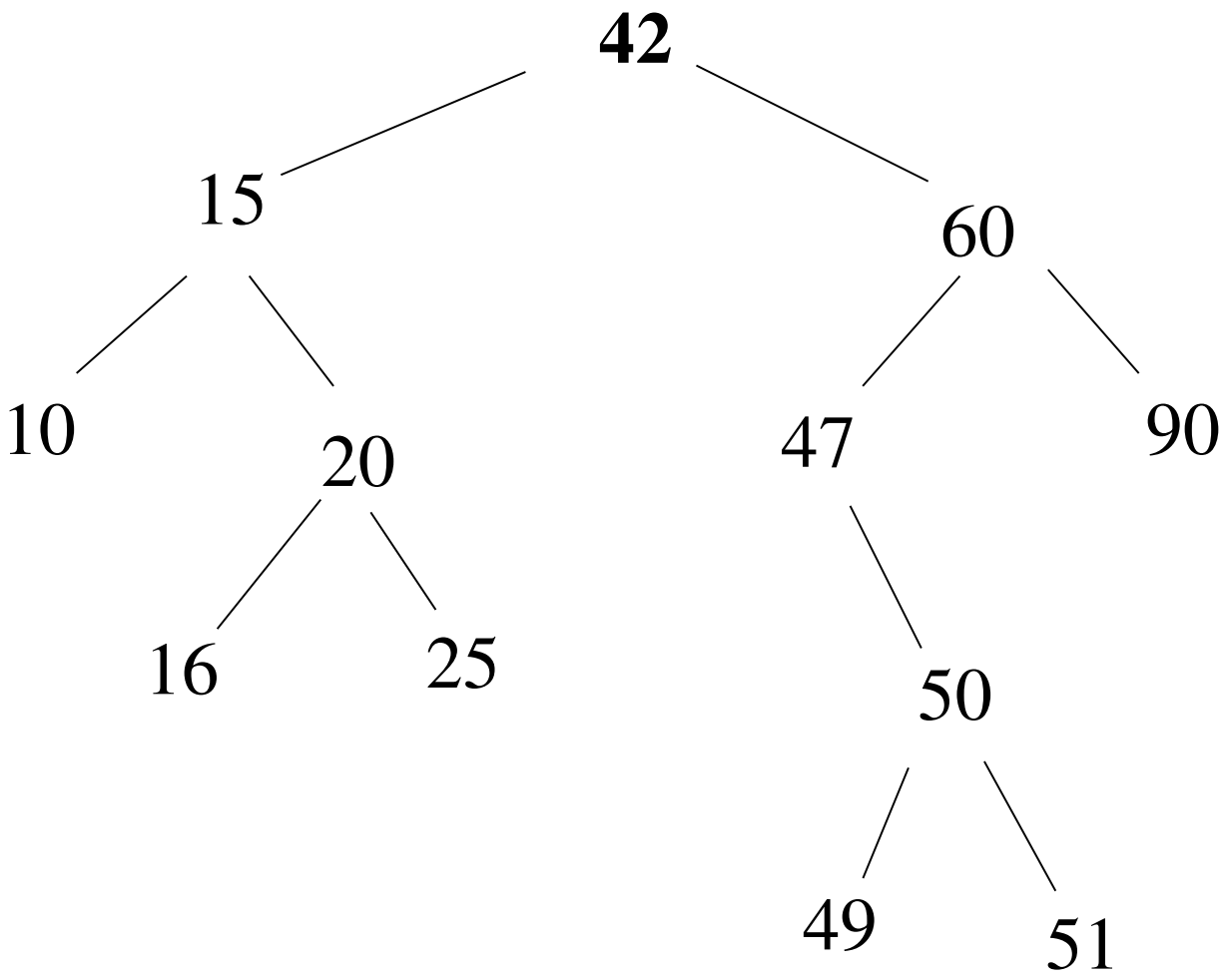
Cas 2 : Le nœud à un seul fils. Il devient ce fils.

Cas 3 : Le noeud à deux fils. On le remplace par le plus petit du sad (le plus petit étant le plus à gauche du sad).

Exemple (supprimer 30) :



On le remplace 30 par 42 et la propriété de l'arbre est conservée.


42
15
60
10
20
47
90
16
25
50
49
51