Chapitre 13

# Arbres binaires de recherche

## 1 Principe

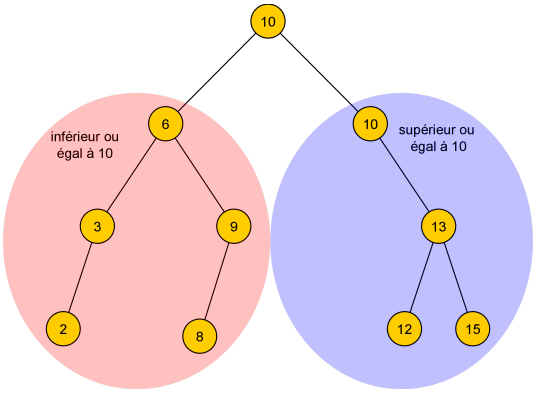
**Définition**

Un arbre binaire de recherche est un arbre binaire

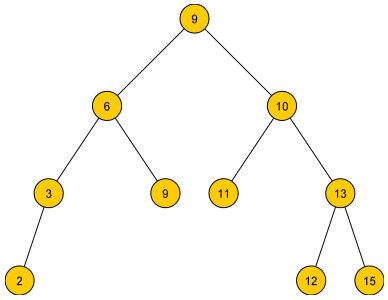
- dont tous les nœuds comportent des valeurs du même type qu'il est possible de comparer (entiers, flottants, chaînes de caractères...) ;
- dont les valeurs de tous les nœuds situés dans le *sous-arbre gauche* d'un nœud sont *inférieures ou égales* à la valeur de ce nœud ;

**Exemples**

Voici un ABR :




Ceci n'est pas un ABR

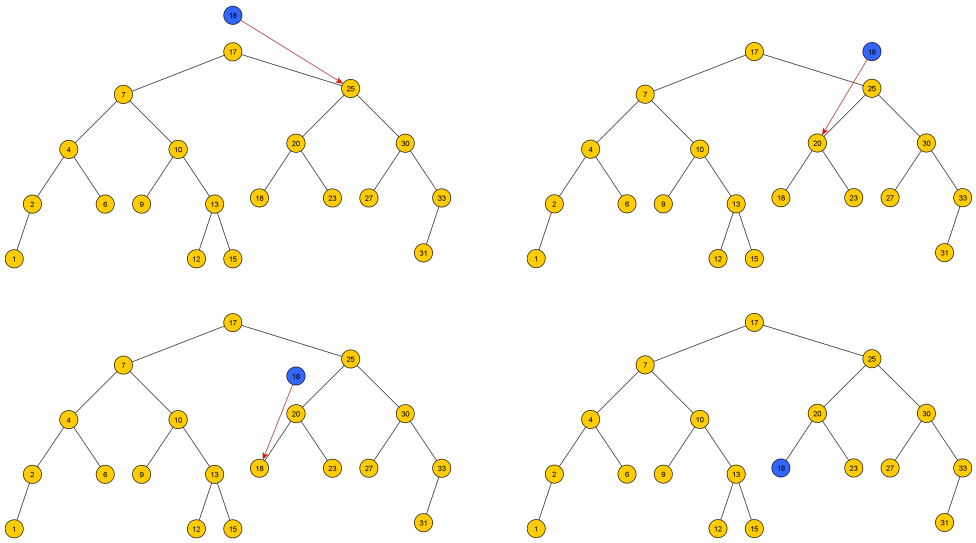



En effet le nœud de valeur 11 n'est pas à la bonne place.

### Méthode : Recherche dans un ABR

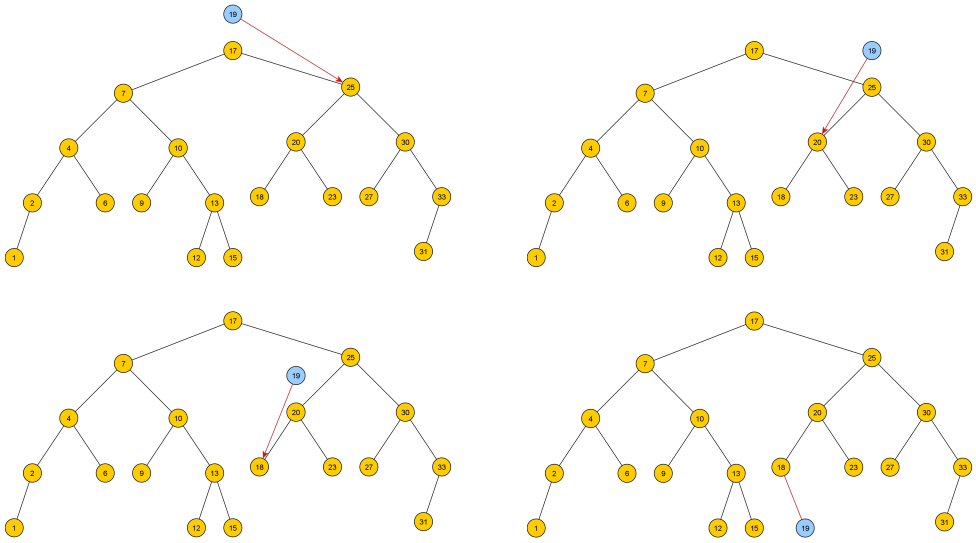
On compare l'élément à trouver avec la valeur de la racine, puis selon les cas

- on s'arrête si on a trouvé la valeur ;
- on essaie d'aller à gauche si la l'élément est plus petit ;
- on essaie à droite s'il est plus grand ;
- si on ne peut continuer, on s'arrête et l'élément ne figure pas dans l'arbre.



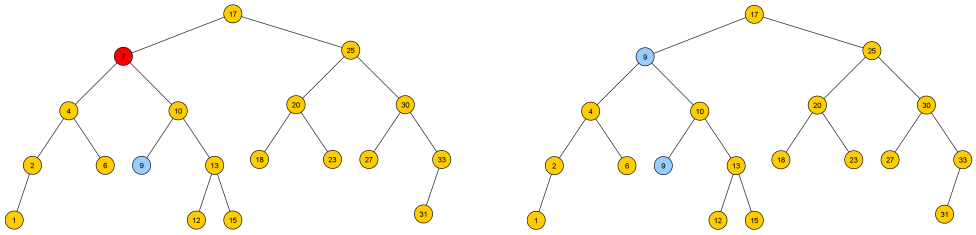
**MéthodeAjout d'un élément dans un ABR**

Le principe est le même que précédemment, quand on ne peut plus continuer, on crée une nouvelle feuille.



**MéthodeSuppression d'un élément dans un ABR (hors programme)**

D'abord on cherchye le nœud contenant l'élément.Si ce n'est pas une feuille, on le remplace par le maximum de son sous-arbre gauche ou par le minimum de son sous-arbre droit en enlevant ce dernier.



## 2 Intérets

Le coût de recherche ou d'ajout d'un élément dans un ABR est proportionnel à sa hauteur.

Dans le pire des cas, où l'ABR est dégénéré, on ne gagne pas grand chose par rapport à une liste.

Le meilleur des cas serait d'avoir un arbre parfait.

Il existe des méthodes pour, lors de l'ajout d'un élément dans un ABR, s'assurer que celui-ci reste bien *équilibré*. Elles ne sont pas au programme de terminale.
Dans ce cas la hauteur $h$ de l'ABR est dite *logarithmique*, c'est à dire que si N désigne le nombre de nœuds, il existe une constante C telle que

$$h \leqslant C.\log_2(N)$$

Alors, avec un ABR équilibré, les opérations de recherche et d'ajouts sont très efficaces.

**Exemple**

> Imaginons un ABR parfait de hauteur $h$, il possède $N = 2^{h+1} - 1$ éléments.
> Le temps de recherche ou d'ajout d'un élément (dans le pire des cas) est de l'ordre de $h$.
>
> Prenons $h = 50$, alors notre ABR comporte 2 251 799 813 685 247 éléments, et on trouve ou on ajoute un nouvel élément en 51 étapes au maximum !