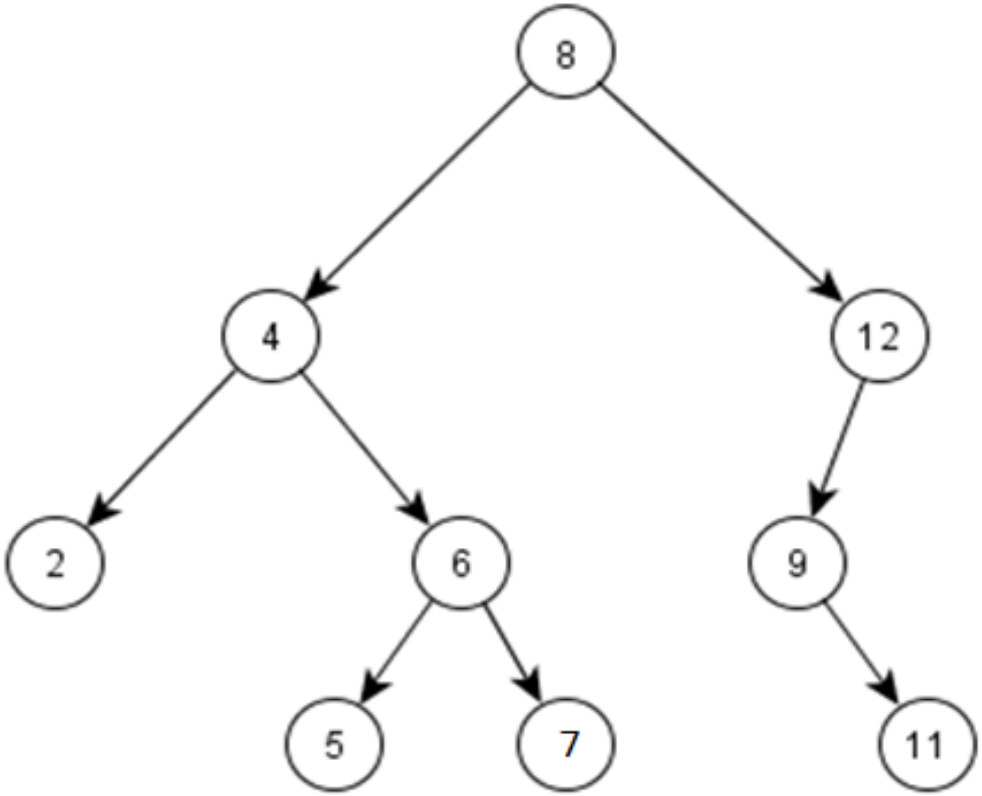
Arbre binaire de recherche

(*Binary search tree*)

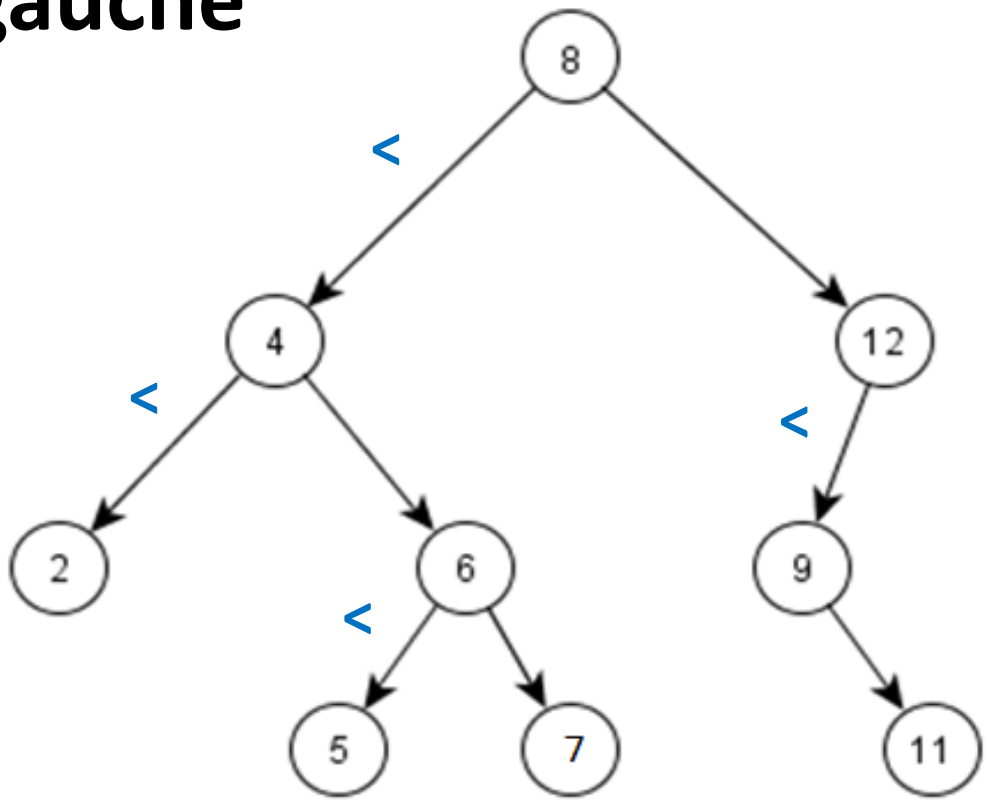
ABR

C’est un arbre binaire



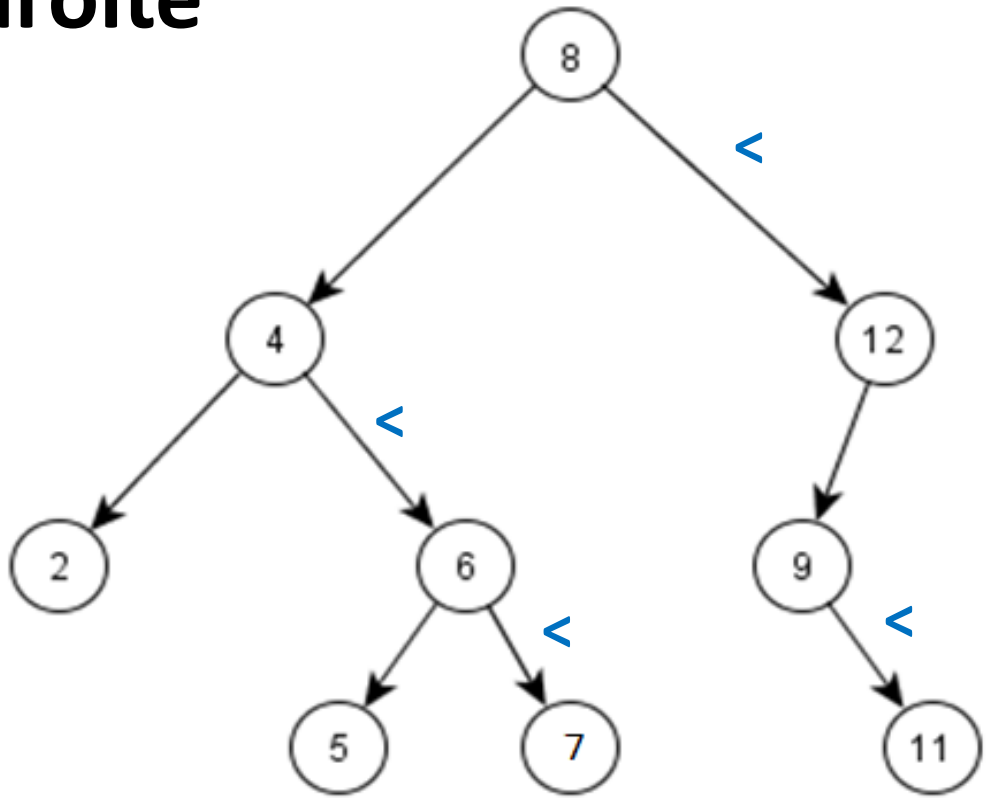
ABR

la descendance **gauche** d'un nœud ne contient que des éléments **inférieurs** à l'élément de ce nœud



ABR

la descendance **droite** d'un nœud ne contient que des éléments **supérieurs** à l'élément de ce nœud



Fonctionnalités ABR

```
boolean estVide()
int taille()
boolean contient(Comparable element)
void insere(Comparable element)
boolean supprime(Comparable element)
String toString()
Iterator iterator()
```

Gestion des doublons

Plusieurs stratégies sont envisageables.

Par exemples :

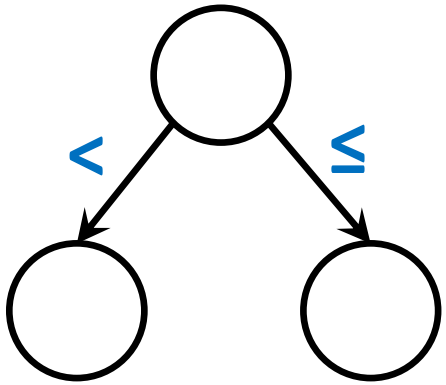
- Refuser les doublons

- Mettre tous les éléments les mêmes dans un même nœud (compteur ou liste)

Gestion des doublons

- Admettre l'insertion et insérer le nœud dupliqué

toujours à droite du nœud de même clé de comparaison



ABR : coût de l'implémentation :

Si l'arbre est équilibré :

```
boolean estVide()
int taille()                                          O(logN)
boolean contient(Comparable element)
void insere(Comparable element)                       O(logN)
void supprime(Comparable element)                     O(logN)
String toString()
Iterator iterator()
```

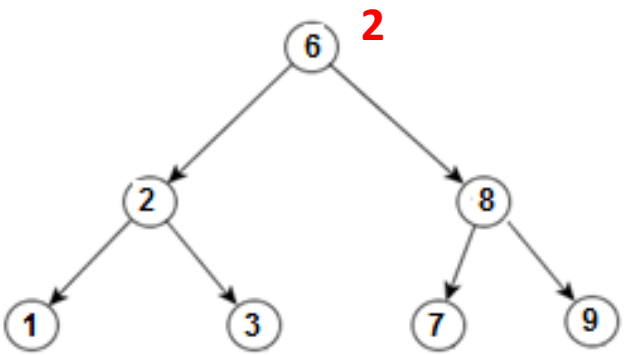
Arbre équilibré

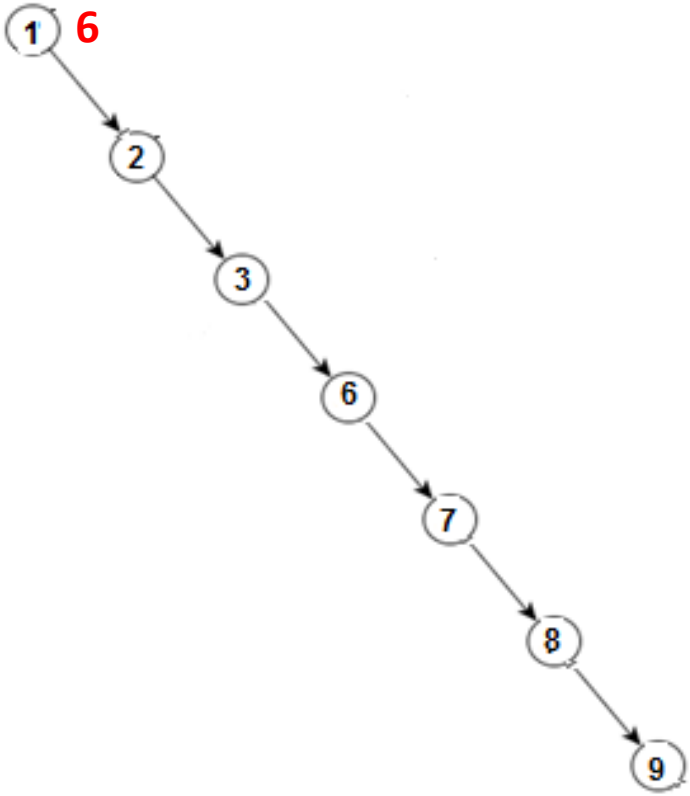
La **hauteur** d'un arbre est la hauteur de sa racine

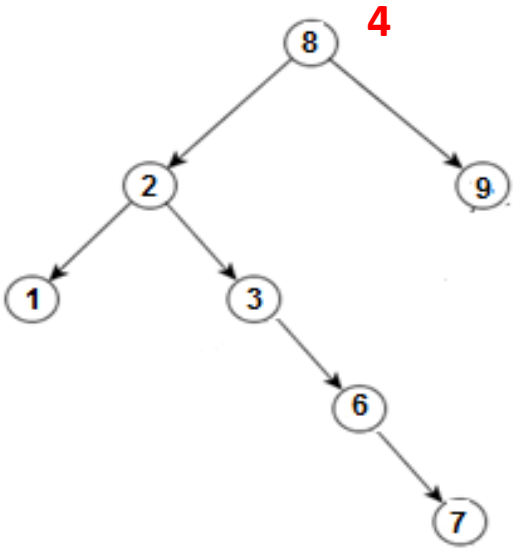
définition récursive

La **hauteur d'un nœud** vaut

- 0 pour une feuille
- un de plus que le maximum des hauteurs de son fils gauche et de son fils droit


2
6
2
8
1
3
7
9


6
1
2
3
6
7
8
9


4
8
2
9
1
3
6
7

Arbre équilibré

La **hauteur** est minimale dans un arbre équilibré

→ log N

Conclusion:

Comme l'arbre binaire de recherche (ABR) n'est pas nécessairement équilibré, on va plutôt utiliser une variante de celui-ci :

- Arbre bicolore ($\rightarrow$ B-arbre binaire symétrique)
- AVL ($\rightarrow$ ABR automatiquement équilibré)
- B-arbre ($\rightarrow$ arbre de recherche équilibré)
- …