

Parcours d'arbre

Programmation

La plupart des méthodes appliquées aux arbres sont récurives

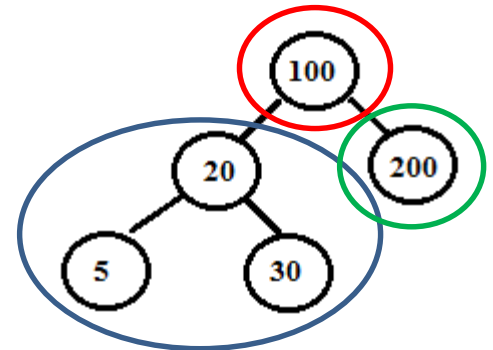
Exemple :

La méthode `somme()` calcule et renvoie la somme des entiers contenus dans l'arbre binaire passé en paramètre

Somme : version 1

```
public int somme () {  
    return somme(racine);  
}  
  
private int somme (Noeud n) {  
    if (n == null) return 0;  
    return n.entier  
        + somme(n.gauche)  
        + somme(n.droit);  
}
```

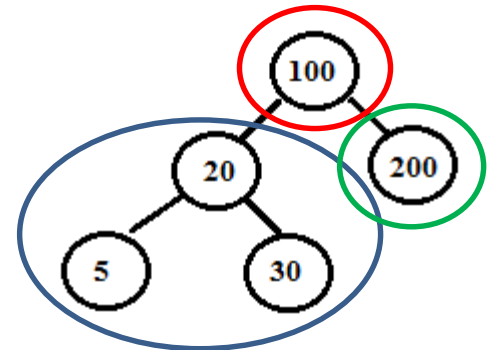
$$100 + ((20 + 5) + 30) + (200)$$



Somme : version 2

```
public int somme () {  
    return somme(racine);  
}  
  
private int somme (Noeud n) {  
    if (n == null) return 0;  
    return  somme(n.gauche)  
        + n.entier  
        + somme(n.droit);  
}
```

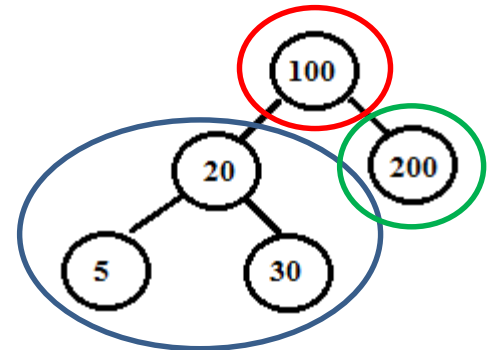
$((5 + 20) + 30) + 100 + (200)$



Somme : version 3

```
public int somme () {  
    return somme(racine);  
}  
  
private int somme (Noeud n) {  
    if (n == null) return 0;  
    return  somme(n.gauche)  
           + somme(n.droit)  
           + n.entier;  
}
```

$$((5 + 30) + 20) + (200) + 100$$



Définition

Le **parcours** d'un arbre est une façon d'aller explorer tour à tour chacun des nœuds de cet arbre.

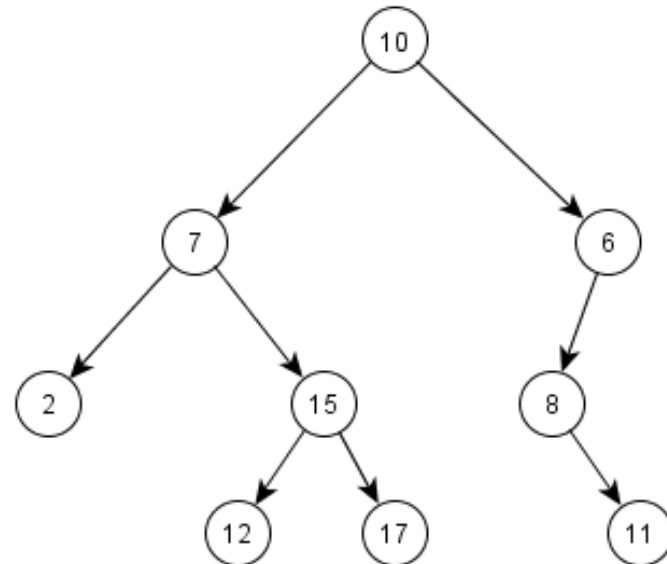
Il existe différents types de parcours, la différence est l'ordre dans lequel on visite les nœuds.

Pré-ordre

Le parcours en **pré-ordre** :

tout élément est visité avant ses enfants :

10, 7, 2, 15, 12, 17, 6, 8, 11

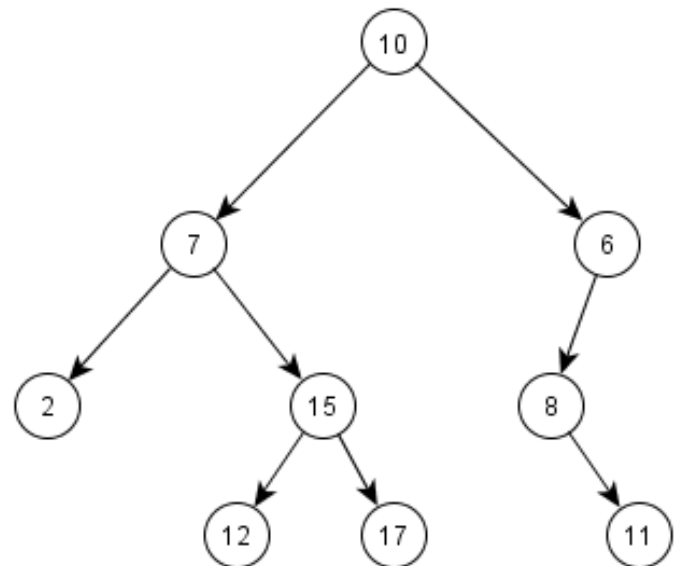


In-ordre

Le parcours en **in-ordre** :

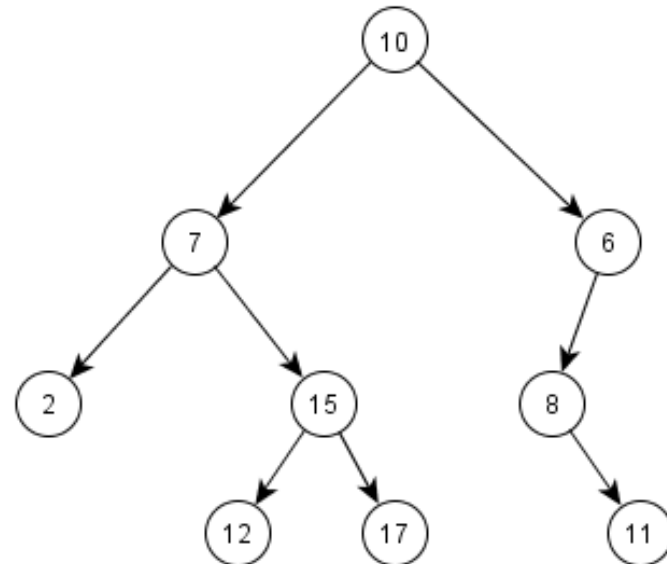
un élément est visité après sa descendance gauche, mais avant sa descendance droite :

2, 7, 12, 15, 17, 10, 8, 11, 6



Post-ordre

Le parcours en **post-ordre** :
un élément sera visité après ses descendants:
2, 12, 17, 15, 7, 11, 8, 6, 10

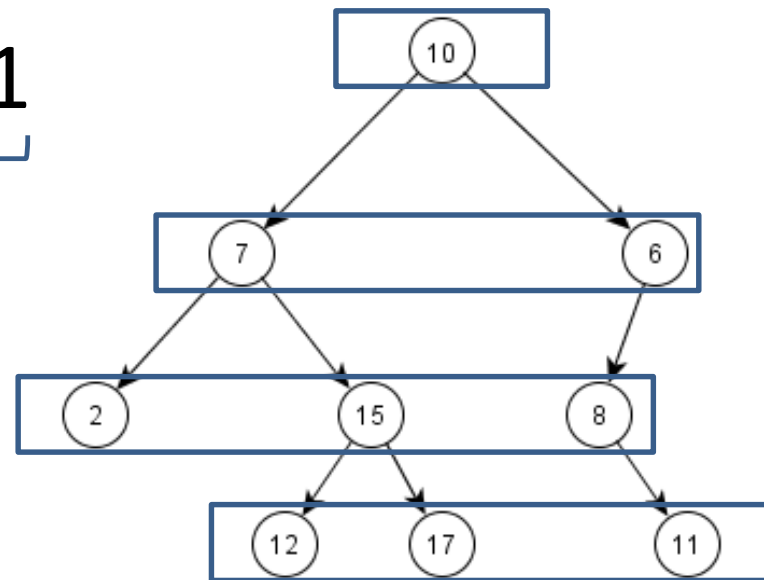


Par niveau

Le parcours **par niveau** :

commence par la racine générale, puis passe à ses deux fils, puis ses quatre petits fils, et ainsi de suite :

10, 7, 6, 2, 15, 8, 12, 17, 11



Programmation

La plupart des méthodes appliquées aux arbres sont récurives.

Toutes ces méthodes peuvent s'écrire de façon itérative en utilisant un itérateur!

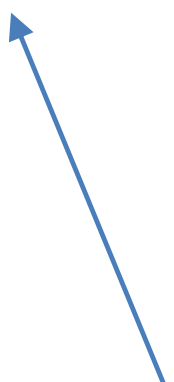
L'itérateur s'écrit de façon récurive!

Somme : version 1

```
public int somme() {  
    int somme = 0;  
    Iterateur it = new PreIterateur();  
    while (it.hasNext()) {  
        somme += it.next();  
    }  
    return somme;  
}
```

Somme : version 2

```
public int somme() {  
    int somme = 0;  
    for (Integer el : this) {  
        somme += el;  
    }  
    return somme;  
}
```



Le « for each » parcourt l'arbre selon l'ordre de l'itérateur renvoyé par la méthode *iterator()* (de l'interface *Iterable*)