Arbres binaires de recherche

Rappel : tas binaire

- topologie : arbre binaire presque complet tassé à gauche
- conditions sur les valeurs : la valeur d'un nœud est supérieure à celle de ses descendants ;

En particulier tout sous-arbre est un tas binaire.

Implémentation des files de priorité par un tas :

- insertion : $O(\log n)$
- extraction de la valeur max (racine) : $O(\log n)$

avec n le nombre d'éléments dans le tas.

Arbre binaire de recherche (ABR)

- topologie : arbre binaire quelconque
- conditions sur les valeurs : pour tout nœud sa valeur est
  - supérieure aux valeurs du sous-arbre gauche
  - inférieure aux valeurs du sous-arbre droit

En particulier tout sous-arbre est un ABR.

Propriété : le parcours infixe affiche les valeurs en ordre croissant.

ABR : quelles opérations ?

- recherche : $O(h)$
- element maximum ou minimum : $O(h)$
- insertion : $O(h)$
- suppression : $O(h)$

Rappel :

$$O(\log n) \le h \le n.$$

Pour garantir que $h = O(\log n)$, il faut en plus que l'arbre soit équilibré

- arbres AVL
- arbres rouge-noir

ABR : quelles opérations ?

- recherche : $O(h)$
- element maximum ou minimum : $O(h)$
- insertion : $O(h)$
- suppression : $O(h)$

Rappel :

$$O(\log n) \leq h \leq n.$$

Pour garantir que $h = O(\log n)$, il faut en plus que l'arbre soit équilibré

- arbres AVL
- arbres rouge-noir

Implémentation des types abstraits

Opération	**ABR équilibré**	**Tas binaire**	**Table de hachage**
Maximum	$O(\log n)$	$O(1)$	$O(n)$
Recherche	$O(\log n)$	$O(n)$	$O(1)\,/\,O(n)$
1 insertion	$O(\log n)$	$O(\log n)$	$O(1)\,/\,O(n)$
n insertions	$O(n \log n)$	$O(n)$	$O(n)\,/\,O(n^2)$
Suppression	$O(\log n)$	$O(\log n)$	$O(1)\,/\,O(n)$

ABR équilibré : alternative à la table de hachage pour implémenter un dictionnaire.

Maximum d'un ABR

Algorithme : max(a : ABR) : entier

Entrées : a : ABR non vide

Sorties : valeur max contenue dans a

si *arbre_vide*(*droite*(a))

  retourner *racine*(a)

retourner *max*(*droite*(a))

Complexité : $O(h)$

Recherche dans un ABR

Algorithme : recherche(a : ABR, v : entier) : booléen

Entrées : a : ABR, v : valeur recherchée dans l'arbre

Sorties : `vrai` si v dans a, `faux` sinon

si *arbre_vide*(a)
    retourner `faux`;
si $racine(a) = v$
    retourner `vrai`;
sinon si $racine(a) > v$
    retourner *recherche*(*gauche*(a), v);
sinon
    retourner *recherche*(*droite*(a), v);

Complexité : $O(h)$

Insertion suppression : modification d'un arbre

Il devient nécessaire de préciser une structure de données :

```
struct noeud{
  val: type des éléments contenus dans l'arbre
  g,d: références vers les noeuds gauches et droits
}
```

- le type Arbre est une référence vers le nœud racine de l'arbre
- l'arbre est vide si cette référence vaut `Nil`

Insertion (au niveau des feuilles)

Algorithme : *insere_arbre*(a : ABR, v : entier) : ABR

Entrées : a : ABR, v : valeur à insérer dans l'arbre

Sorties : Arbre dans lequel la valeur v a été insérée

Variables locales : nd : nœud

si $a = Nil$
- $nd.val \leftarrow v$
- $nd.g \leftarrow Nil$
- $nd.d \leftarrow Nil$
- **retourner** $\uparrow nd$

si $a.val > v$
- `/* on insère à gauche */`
- $a.g \leftarrow insere\_arbre(a.g, v)$;

si $a.val < v$
- `/* sinon on insère à droite */`
- $a.d \leftarrow insere\_arbre(a.d, v)$;

retourner a;

Complexité : $O(h)$

Suppression

Trois cas de figure pour le nœud qui contient la valeur

- le nœud est une feuille
  - supprimer la feuille
- le nœud a un seul fils
  - le supprimer et le remplacer par son fils
- le nœud a deux fils
  - détails à suivre

Suppression d'un nœud qui a deux fils

Par qui remplacer le nœud de valeur v pour conserver la propriété ABR ?

Pa le nœud :

- dont la valeur est la plus grande valeur plus petite que v (prédecesseur)
  - la plus grande valeur du sous-arbre gauche
  - donc celle qui est le plus à droite dans ce sous-arbre
- ou dont la valeur est la plus petite valeur plus grande que v (successeur)
  - la plus petite valeur du sous-arbre droit
  - celle qui est le plus à gauche dans ce sous-arbre

Suppression d'un nœud qui a deux fils

Par qui remplacer le nœud de valeur v pour conserver la propriété ABR ?

Pa le nœud :

- dont la valeur est la plus grande valeur plus petite que v (prédecesseur)
  - la plus grande valeur du sous-arbre gauche
  - donc celle qui est le plus à droite dans ce sous-arbre
- ou dont la valeur est la plus petite valeur plus grande que v (successeur)
  - la plus petite valeur du sous-arbre droit
  - celle qui est le plus à gauche dans ce sous-arbre

Algorithme : *extrait_arbre*(a : ABR, v : entier) : ABR

```
si a = Nil retourner a ;
si v > a.val
    a.d ← extrait_arbre(a.d, v)
sinon si v < a.val
    a.g ← extrait_arbre(a.g, v)
sinon si a.g = Nil
    retourner a.d
sinon si a.d = Nil
    retourner a.g
sinon
    w ← max(a.g)
    a.val ← w
    a.g ← extrait_arbre(a.g, w)
retourner a;
```

Complexité : $O(h)$

arbre équilibré

Définition : un arbre est équilibré si, pour chaque nœud, la différence de hauteur du sous-arbre et droit et du sous-arbre gauche diffère au plus de 1.

Pour tout sous-arbre a on doit avoir

$$|h(a.g) - h(a.d)| \leq 1$$

où $h(a)$ est la hauteur de l'arbre a.

Résultat

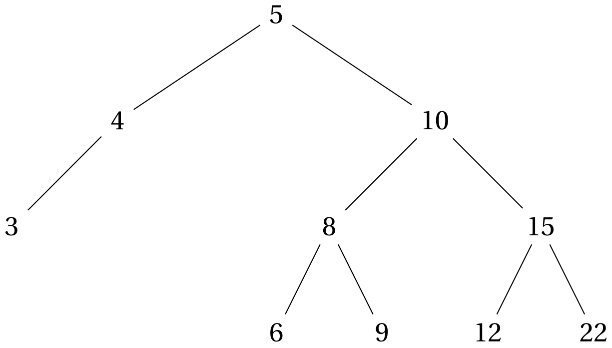
La hauteur h d'un arbre équilibré à n nœuds satisfait

$$h \in O(\log n)$$

Arbre AVL (Adelson-Velsky, Landis 1962)

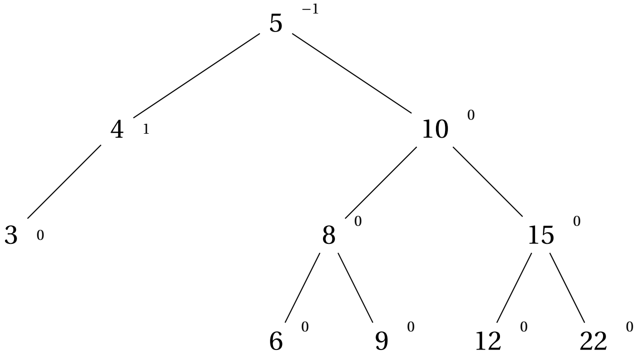
- Arbre ABR
- Arbre équilibré

Exemple d’arbre AVL



On affiche $h(a.g) - h(a.d)$ pour chaque nœud.

Exemple d’arbre AVL



On affiche $h(a.g) - h(a.d)$ pour chaque nœud.

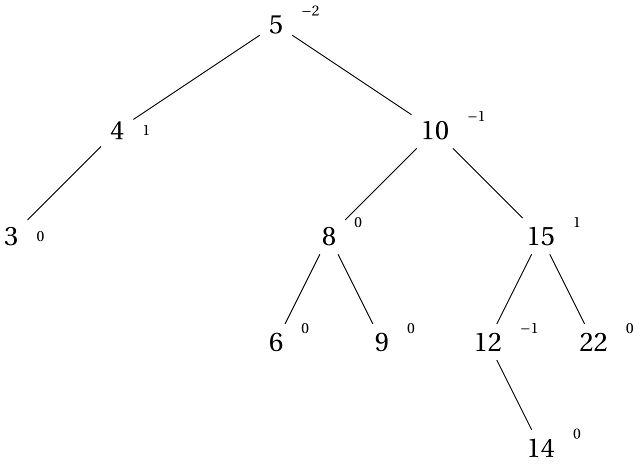
Insertion / suppression dans un AVL

- on insère / supprime comme dans un ABR
- on équilibre si besoin
- complexité $O(h) = O(\log n)$

Conséquence : algo de tri des valeurs en $O(n \log n)$.

Exemple d'arbre AVL

Ajout aux feuilles du noeud 14

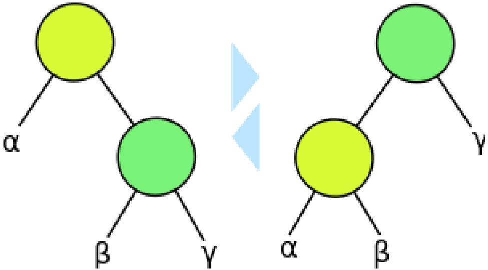


Opération de rotation

Il y a 3 cas de figures pour un déséquilibre de $+2$ (resp. -2) :

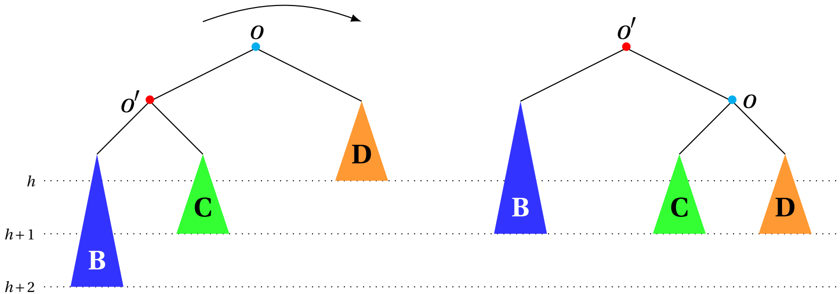
- Un déséquilibre de $+1$ sur le sous-arbre gauche (resp. droit)
- Un déséquilibre de 0 sur le sous-arbre gauche (resp. droit)
- Un déséquilibre de -1 sur le sous-arbre gauche (resp. droit)

Dans tous les cas, on utilise l'opération de **rotation**

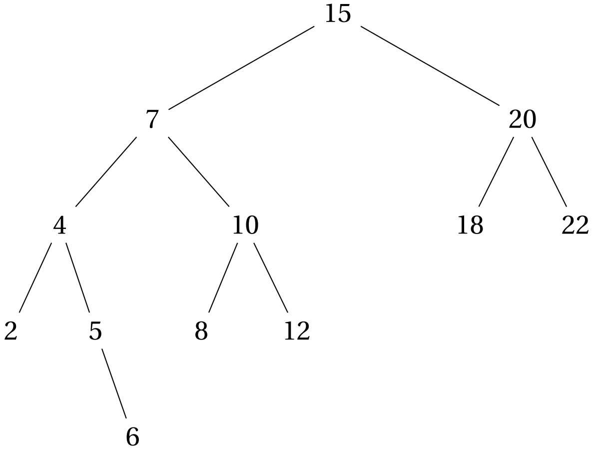


qui conserve la propriété ABR.

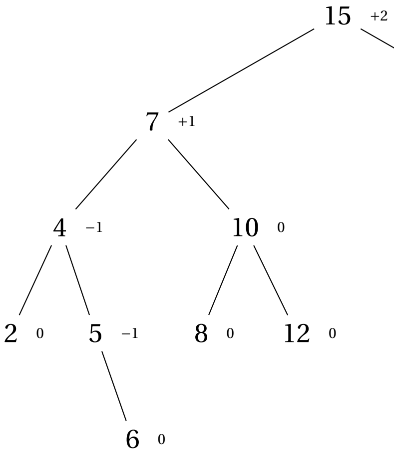
déséquilibre de +2 à la racine, + 1 dans le sous-arbre gauche (insertion ou suppression)



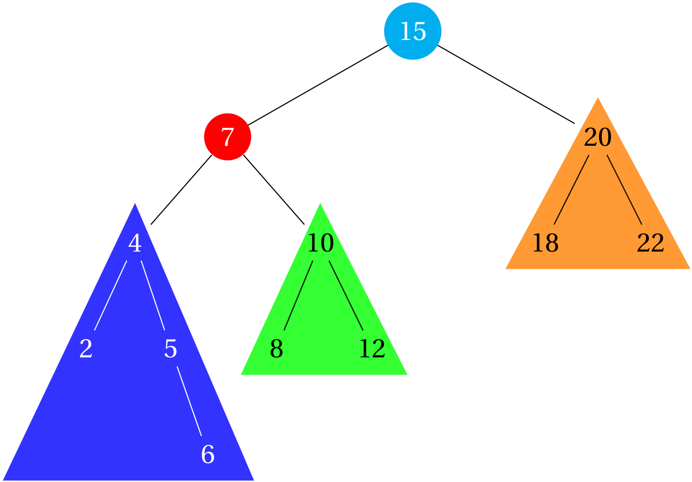
Exemple déséquilibre +1 dans le sous-arbre gauche



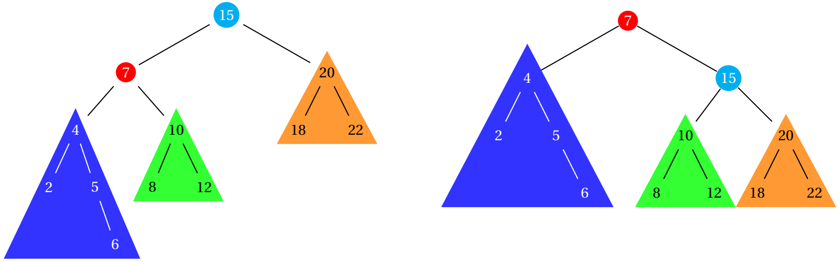
Exemple déséquilibre +1 dans le sous-arbre gauche



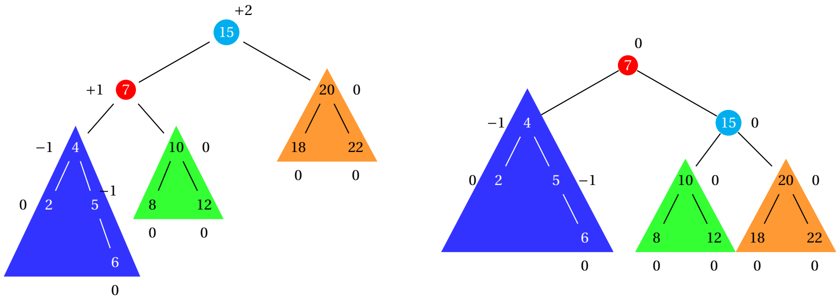
Exemple déséquilibre +1 dans le sous-arbre gauche



Exemple

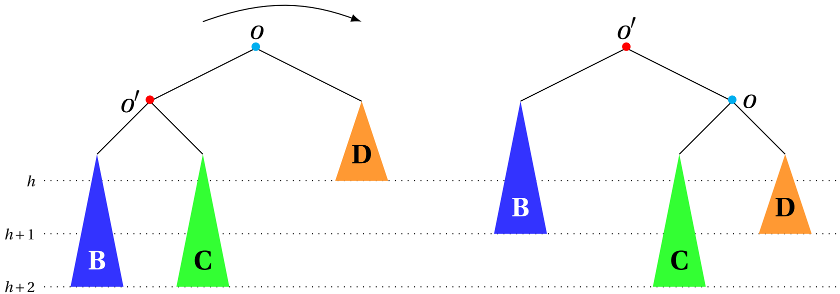


Exemple

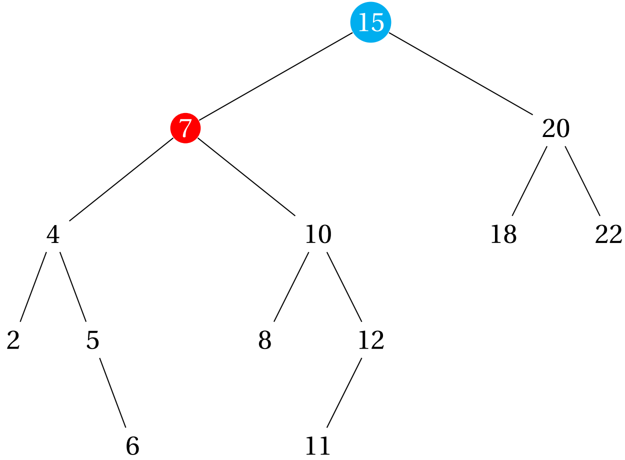


+2
15
+1
7
20
0
−1
4
10
0
18
22
0
0
0
2
5
−1
8
12
0
0
6
0
0
7
−1
4
15
0
0
2
5
−1
10
0
20
0
6
8
12
18
22
0
0
0
0
0

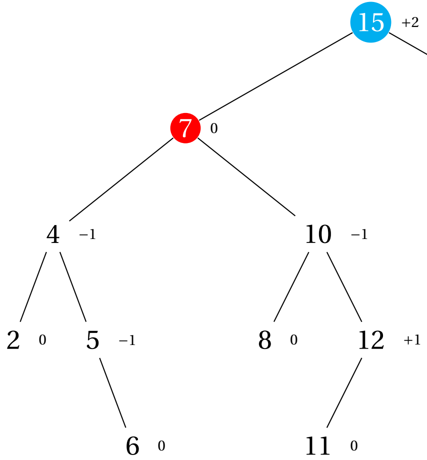
déséquilibre de +2 à la racine, 0 dans le sous-arbre gauche (suppression uniquement)



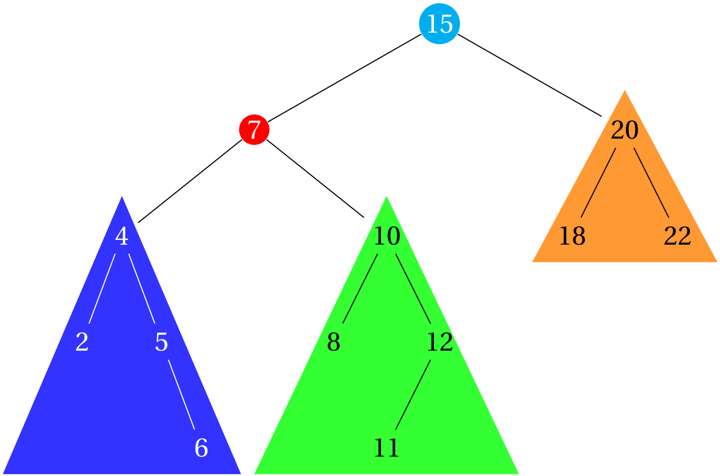
Exemple déséquilibre 0 dans le sous-arbre gauche



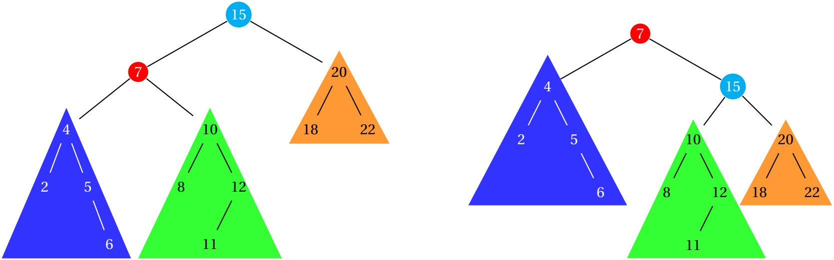
Exemple déséquilibre 0 dans le sous-arbre gauche



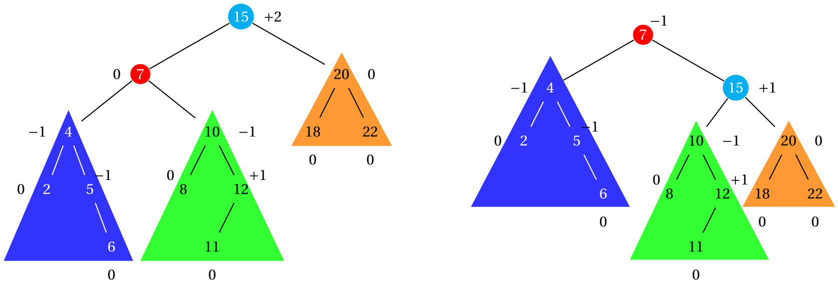
Exemple déséquilibre 0 dans le sous-arbre gauche



Exemple

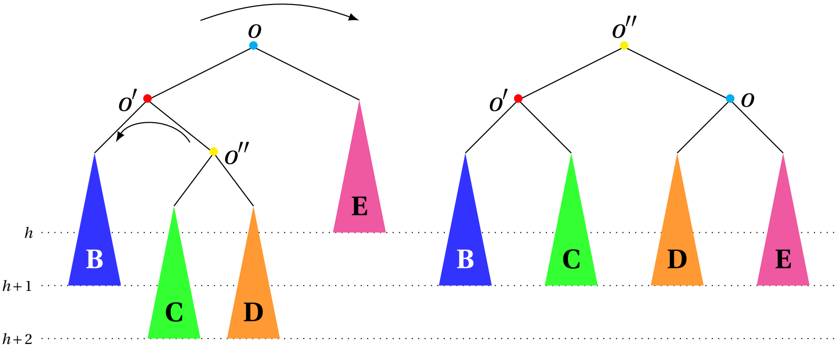


Exemple

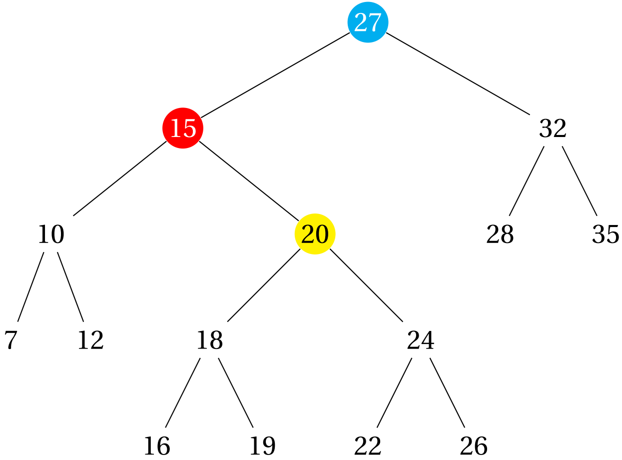


15 +2
0 7
20 0
−1 4
10 −1
18
22
0 0
0 2
5 −1
0 8
12 +1
6
11
0 0
7 −1
−1 4
15 +1
0 2
5 −1
10 −1
20 0
6
0 8
12 +1
18
22
0 0 0
11
0

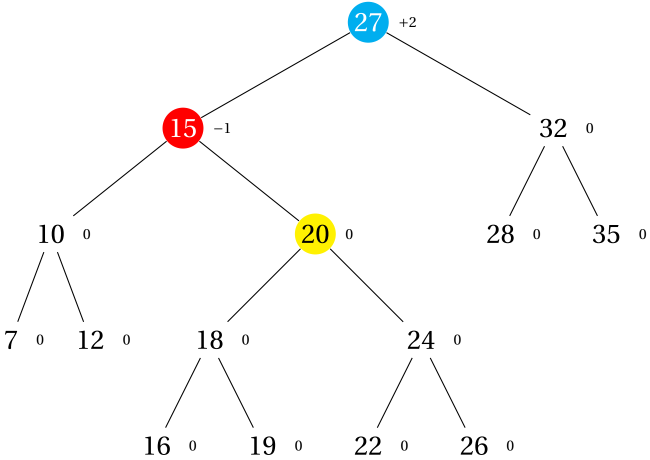
déséquilibre de +2 à la racine, -1 dans le sous-arbre gauche (suppression uniquement)



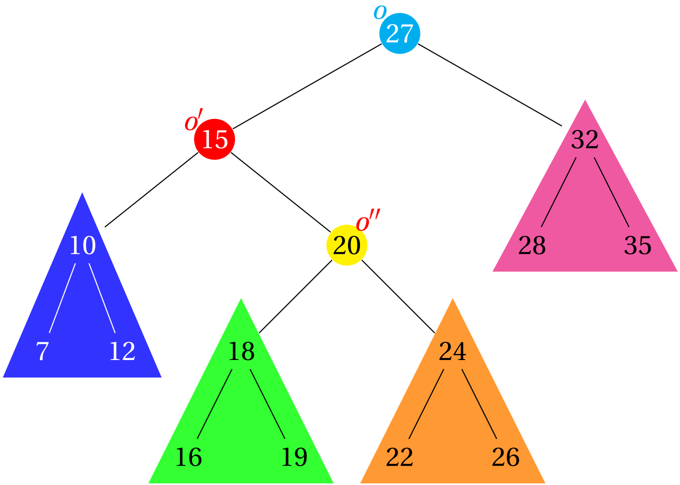
Exemple



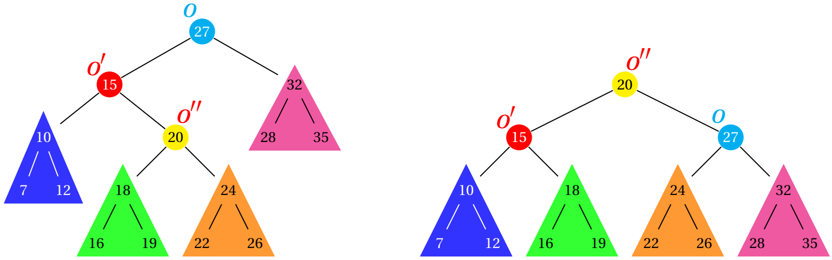
Exemple



Exemple

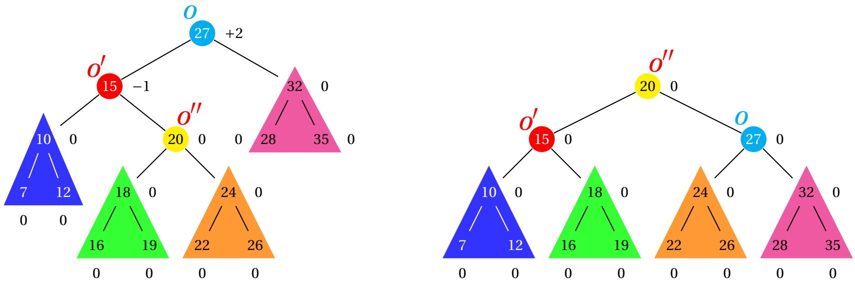


Exemple



o
27
o'
15
o''
20
32
28
35
10
7
12
18
16
19
24
22
26
o''
20
o'
15
o
27
10
7
12
18
16
19
24
22
26
32
28
35

Exemple



o
27
+2
o′
15
−1
32
0
10
0
o″
20
0
0
28
35
0
7
12
0
0
18
0
24
0
16
19
22
26
0
0
0
0
o″
20
0
o′
15
0
o
27
0
10
0
18
0
24
0
32
0
7
12
16
19
22
26
28
35
0
0
0
0
0
0
0
0

Bilan

Insertion : au plus une rotation.

Suppression : au plus $O(\log n)$ rotations où n est la taille de l'arbre.

Dans tous les cas, complexité $O(\log(n))$.