

NFP136- Cours 7

ARBRES DE RECHERCHE

PLAN

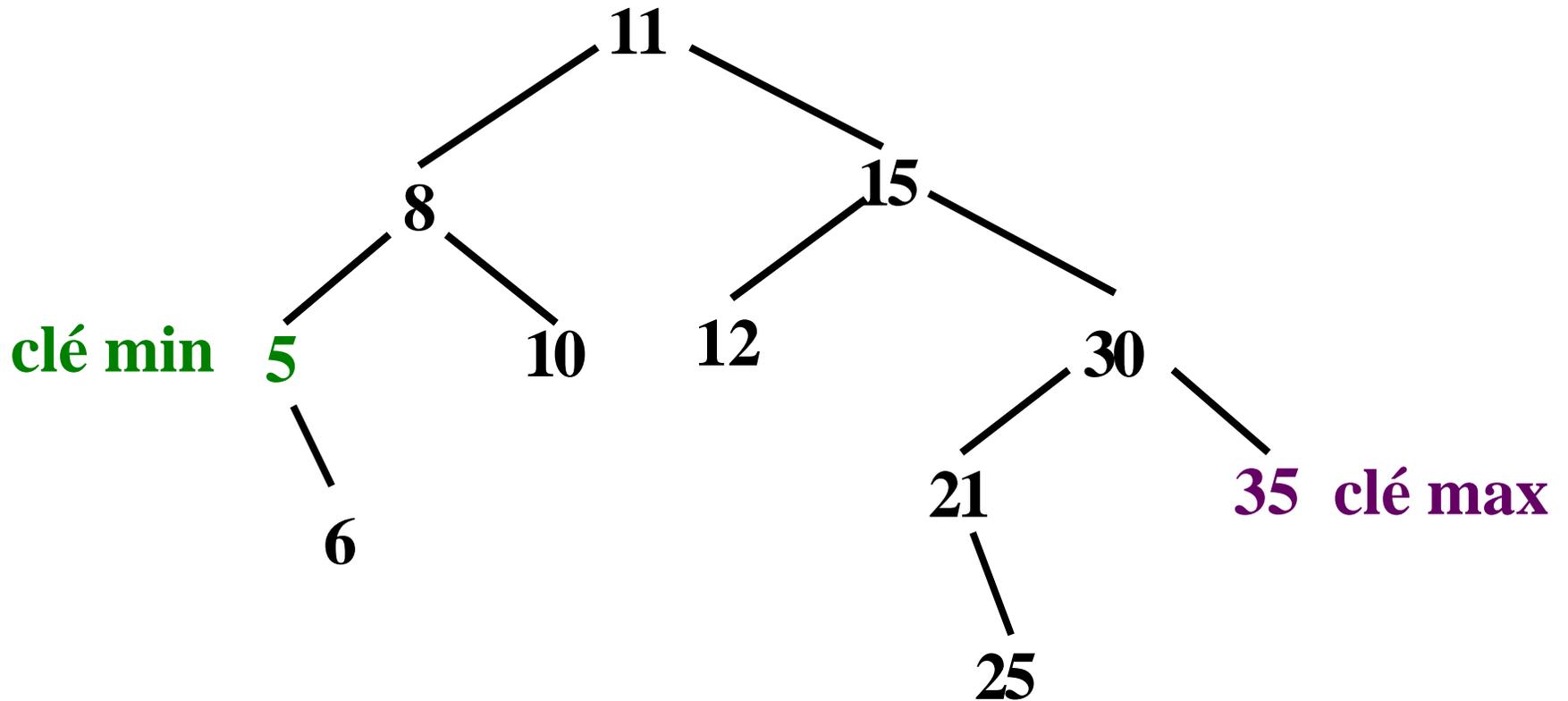
- **Arbres binaires de recherche**
- **Arbre h-équilibrés**

7.1 ARBRES BINAIRES DE RECHERCHE

Définition

Arbre binaire tel qu'en tout nœud la clé du nœud est supérieure à celle de tous ses descendants de gauche et inférieure à celle de tous ses descendants de droite

EXEMPLE



ABR: **structures de données pour grands
volumes variables d'informations
structures dynamiques**

Arbre binaire a

a.g = sous-arbre de gauche de racine(a)

a.d = sous-arbre de droite de racine(a)

Clés **une donnée → une clé**

{clés} ensemble totalement ordonné

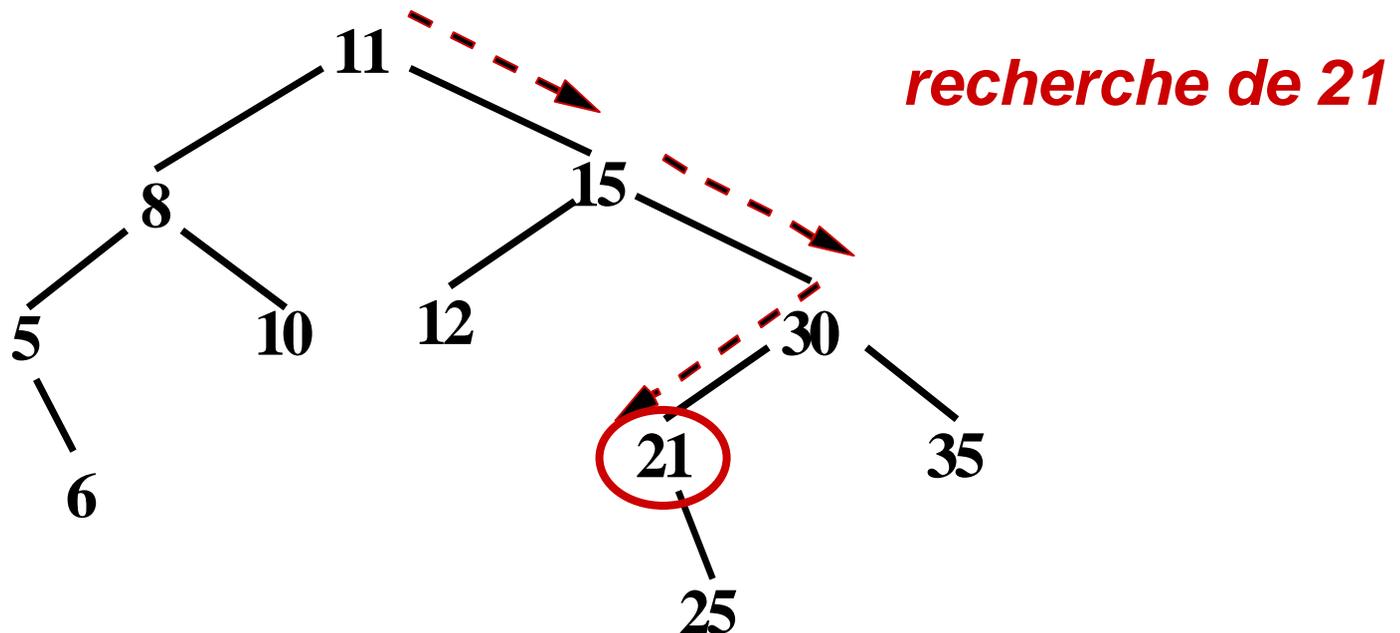
```
class ABR {
    private int cle;
    private String element;
    private ABR gauche;
    private ABR droit;

public ABR(int a, String s, ABR g, ABR d) {
    cle = a;
    element = s;
    gauche= g;
    droit=d;
}
//les méthodes
```

Recherche :

**descente de l'arborescence avec
comparaison en chaque nœud**

EXEMPLE



principe de la recherche de l'élément e de clé c dans l'arbre a

précondition: $c \in a$

recherche(c,a) renvoie String

si $c == a.cle$

alors retourner (a.element)

//élément de la racine de a

sinon

si $c < a.cle$

alors retourner (recherche(c, a.gauche))

sinon retourner (recherche(c, a.droit))

finsi

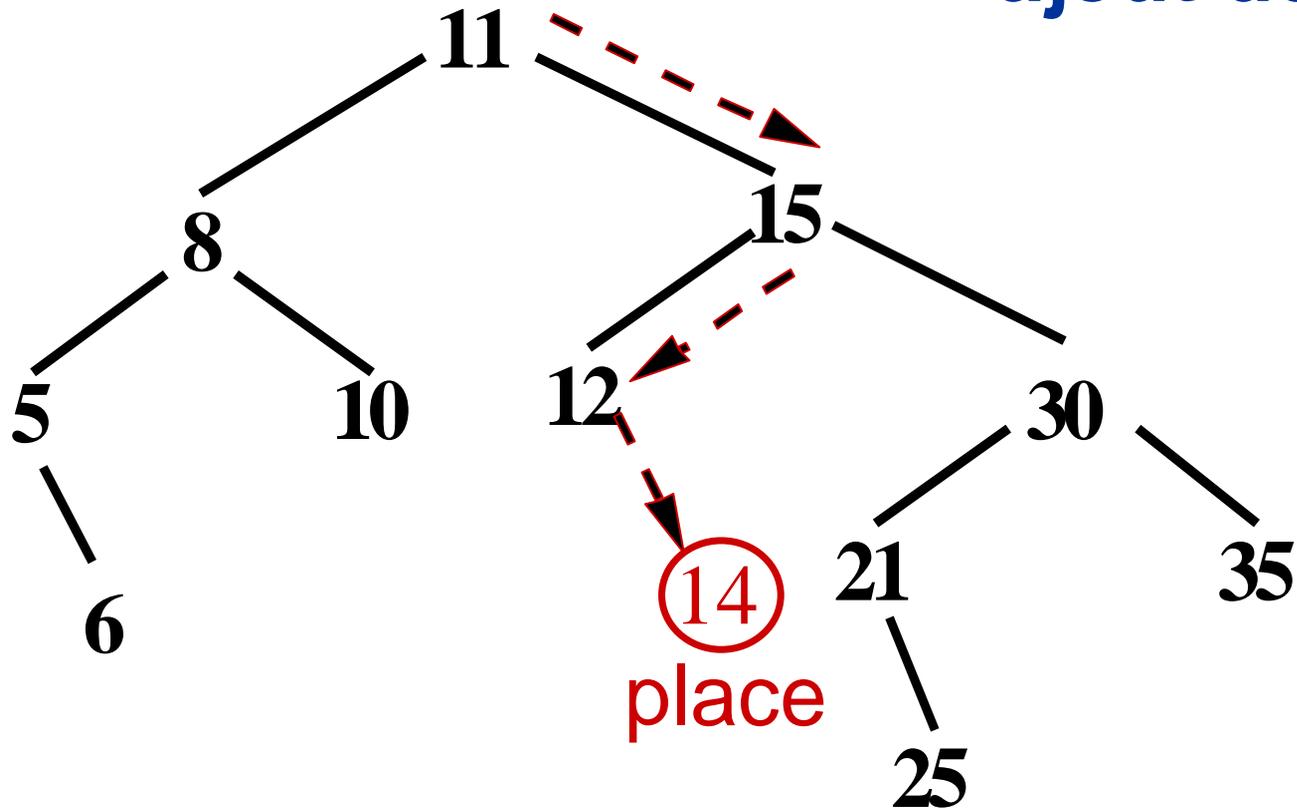
finsi

complexité: $O(h(a))$

ADJONCTION D'UN ELEMENT

EXEMPLE

ajout de 14



complexité: $O(h(a))$

adjonction dans a d'un élément e de clé c

ABR ajouter (ABR a, String e, int c)

b=a;

si b == null alors b=new ABR (c, e, null, null) ;

sinon tant que non place_trouvée faire

si b.cle == c alors "erreur" //élément déjà présent

si b.cle > c

alors //descendre à gauche

si b.gauche == null alors place_trouvée=vrai //fils gauche de b

b.gauche=new ABR (c, e, null, null) ;

sinon b = b.gauche;

finsi;

sinon //descendre à droite

si b.droit == null alors place trouvée=vrai //fils droit

b.droit=new ABR (c, e, null, null) ;

sinon b =b.droit;

finsi;

finsi;

fait;

finsi;

retourner b;

suppression d'un nœud N

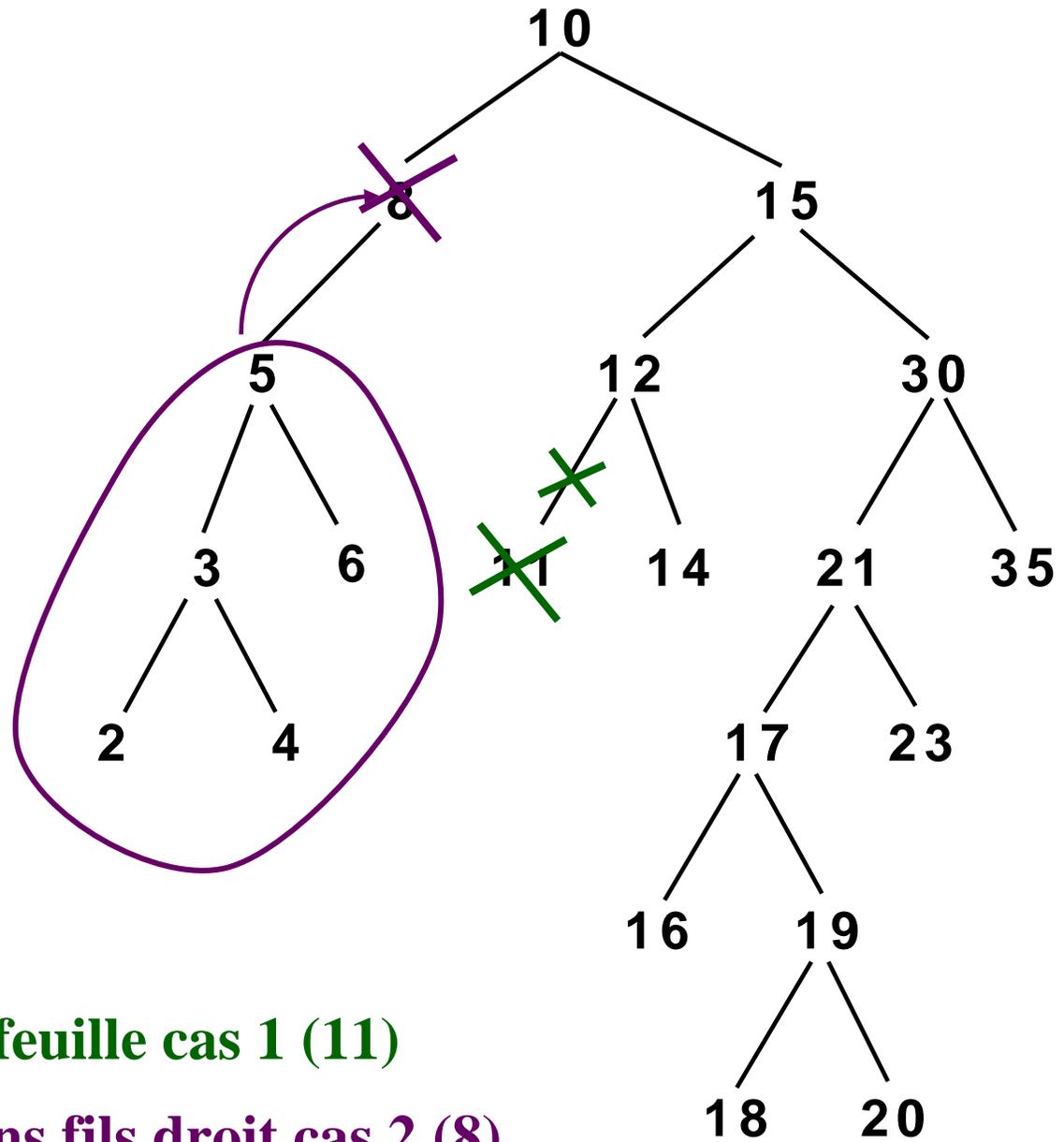
cas 1: N est une feuille

cas 2: N a un seul fils F

cas 3: N a 2 fils

*(2 cas *cas 3-1 cas 3-2*)*

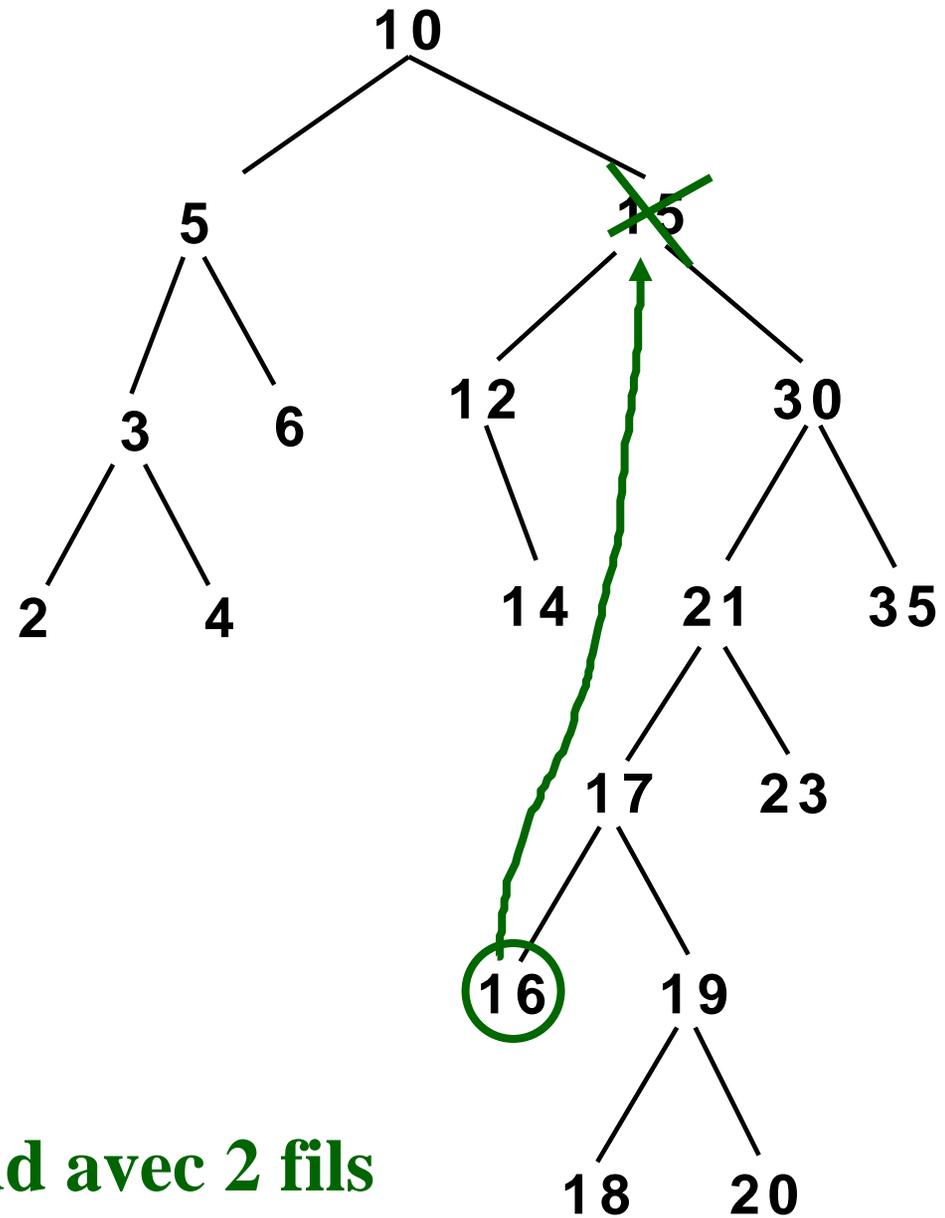
EXEMPLE



suppression d'une feuille cas 1 (11)

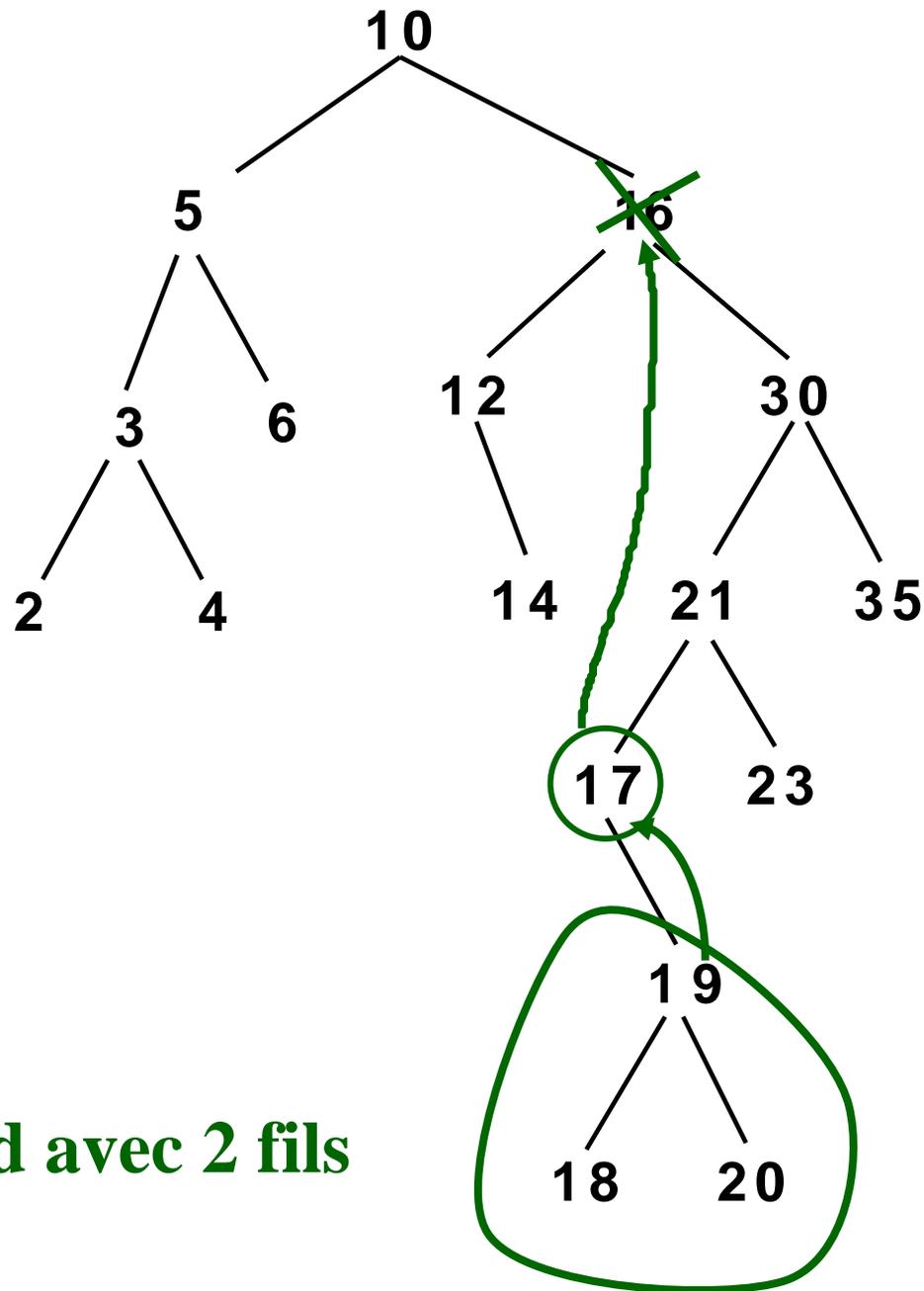
sup. d'un nœud sans fils droit cas 2 (8)

EXEMPLE



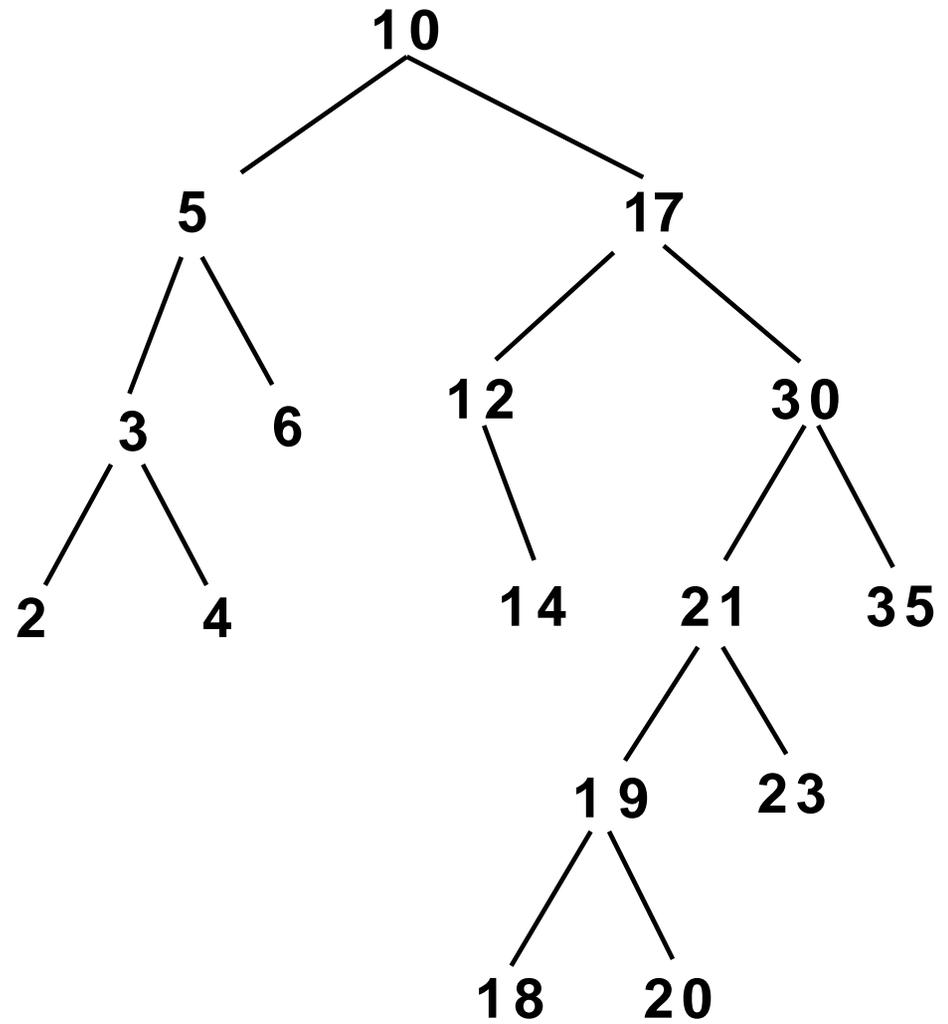
suppression d'un nœud avec 2 fils
cas 3-1 (15)

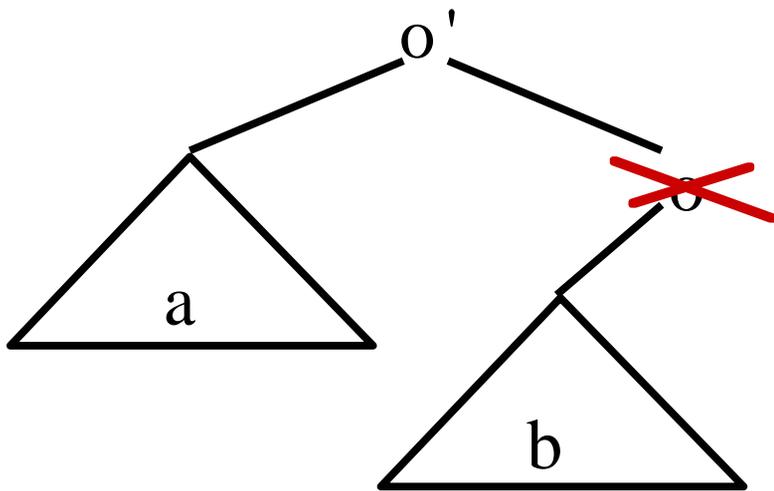
EXEMPLE



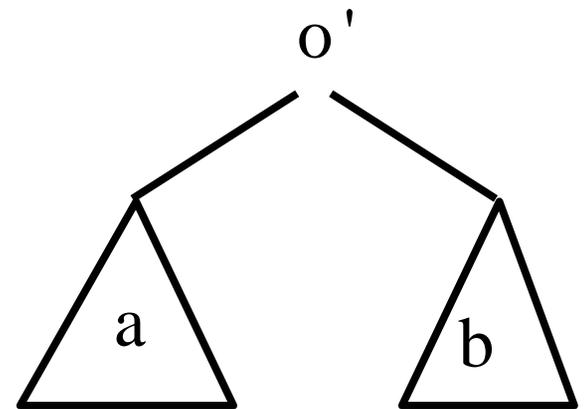
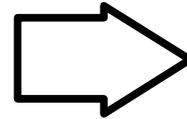
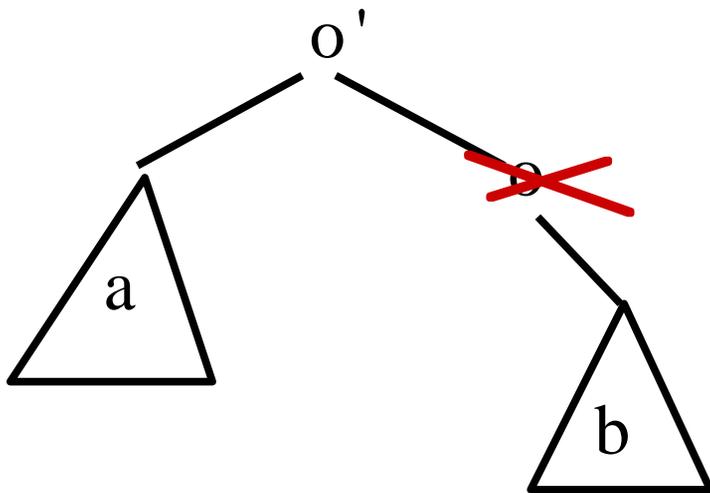
**suppression d'un nœud avec 2 fils
cas 3-2 (16)**

EXAMPLE

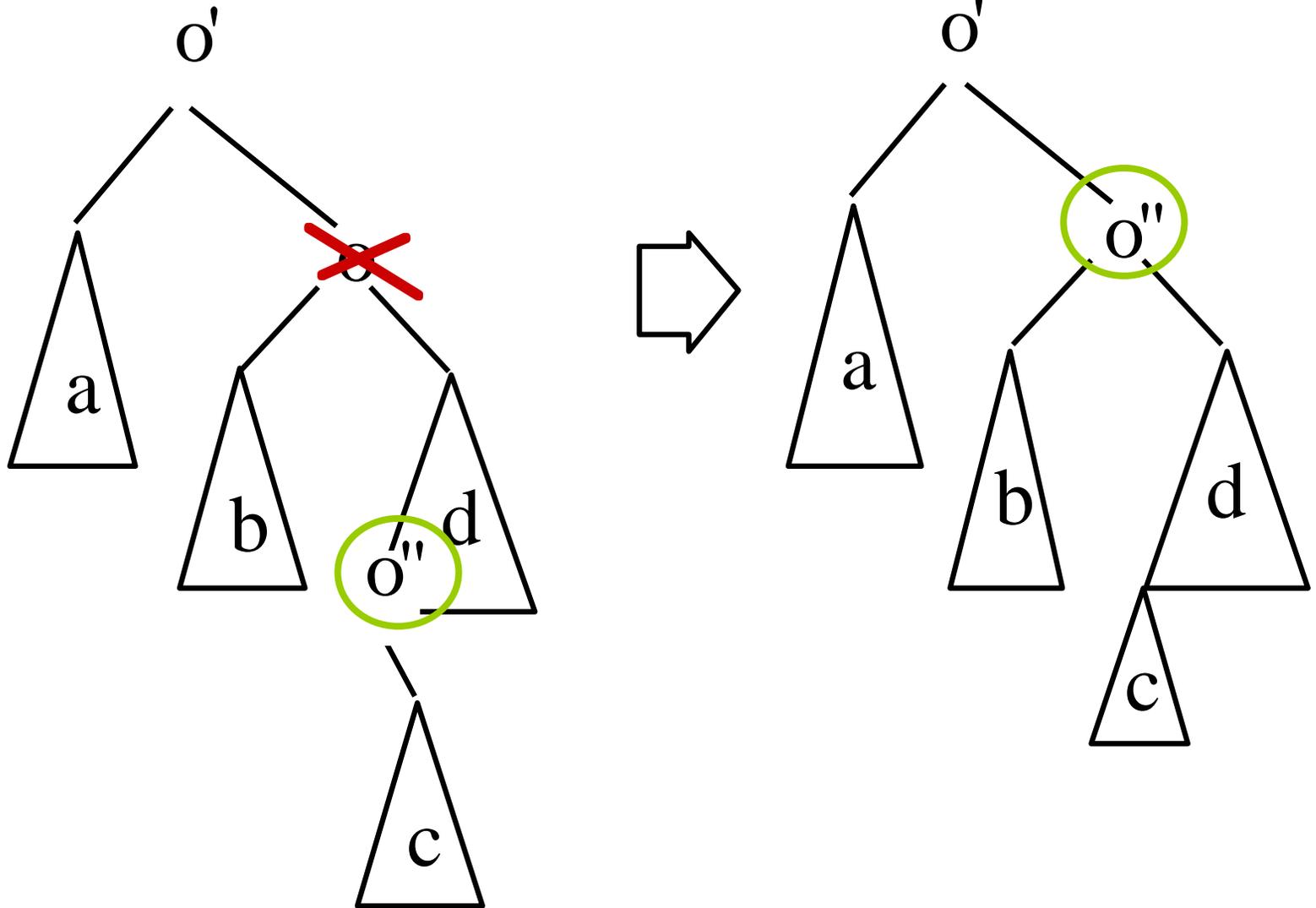




cas 2



Cas 3.2



suppression d'un nœud N

cas 1: N est une feuille

→ suppression simple

cas 2: N a un seul fils F

→ F prend la place de N
(remontée du sous-arbre)

cas 3: N a 2 fils

→ est remplacé par le nœud de plus
petite clé de son sous-arbre de droite

cas 3-1: c'est une feuille

cas 3-2: ce noeud n'a qu'un fils
droit qui prend sa place

cas 3

a a 2 fils:

rechercher le plus petit du sous-arbre de droite

f = a.d;

tant que f.g ≠ ∅ faire f=f.g fait;

a.element = f.element;

a.cle=f.cle; *l'élément de f remplace l'élément supprimé*

f = f.d; *si f n'a pas de fils gauche remonter le fils droit de f;*

f.d = ∅ si f feuille

Complexité de la suppression: $O(h(a))$

Conclusion arbre binaire de recherche

- recherche,
adjonction,
suppression

complexité: $O(h(a))$

avec $\log_2 n \leq h(a) \leq n-1$

complexité en moyenne: $O(\log n)$

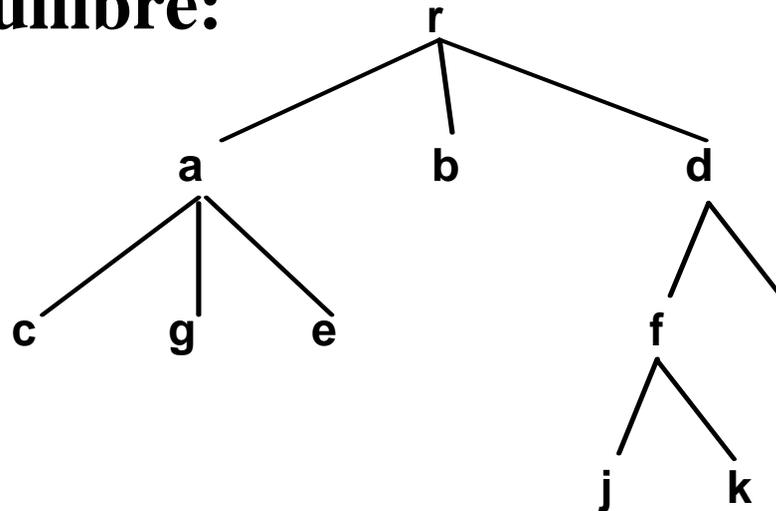
- problème:

éviter les cas défavorables

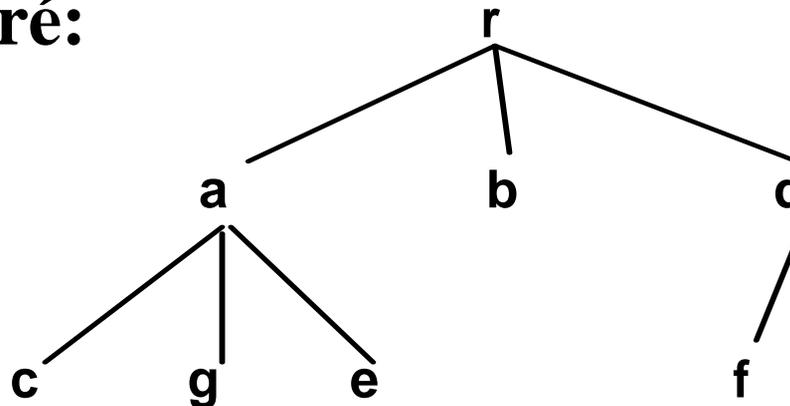
→ arbres équilibrés

équilibre

arbre non équilibré:



arbre équilibré:



les feuilles sont sur au plus 2 niveaux

7.2 ARBRES H-EQUILIBRES

définition: Arbre tel qu'en tout nœud la différence de hauteur entre les sous-arbres gauche et droit est au plus de 1

déséquilibre:

$$\text{déseq}(a) = h(a.g) - h(a.d)$$

$$\text{déseq}(a) = 0 \text{ si } a = \emptyset$$

arbre H-équilibré

pour tout b sous-arbre de a ,

$$\text{déséq}(b) \in \{-1, 0, +1\}$$

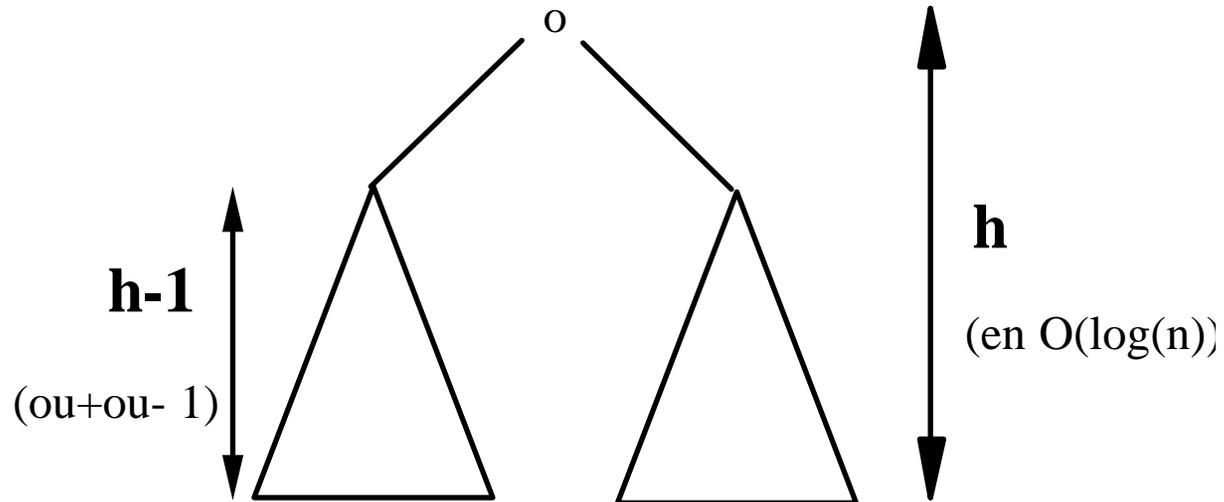
arbre H-équilibré $\Rightarrow \log_2(n+1) \leq h + 1 \leq 1.45 \log_2(n+1)$

arbre AVL:

arbre binaire de recherche

H-équilibré

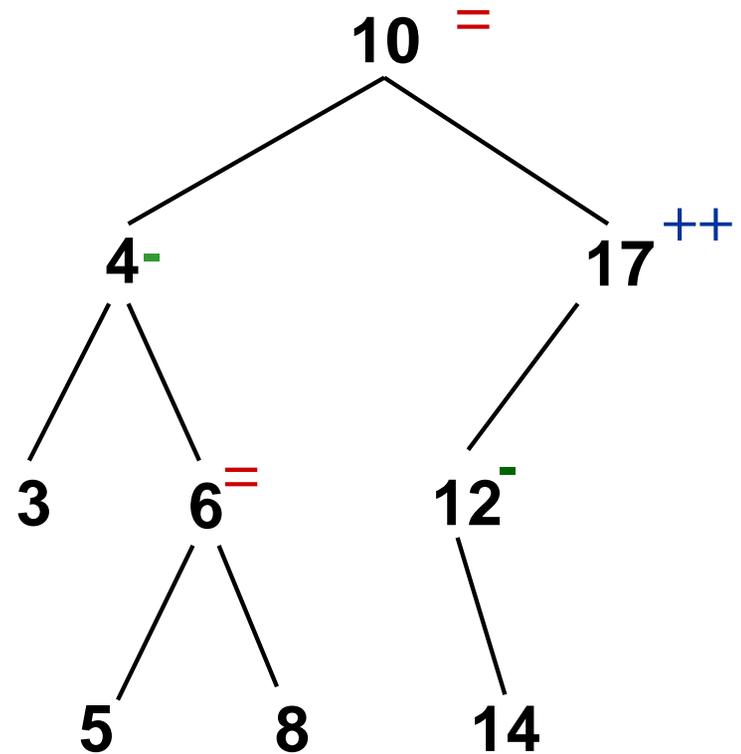
⇒ *la recherche dans un arbre AVL est au pire en $O(\log_2 n)$*



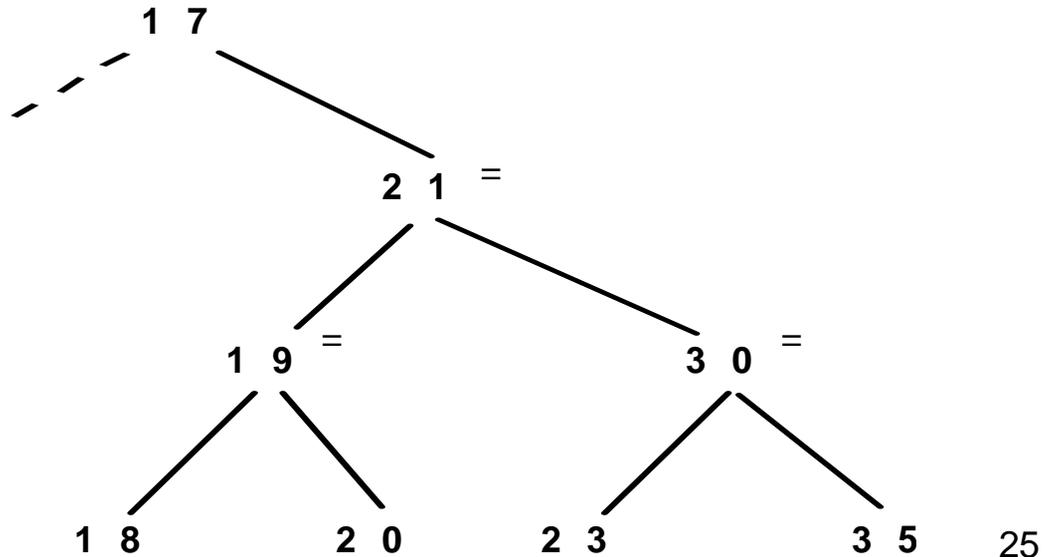
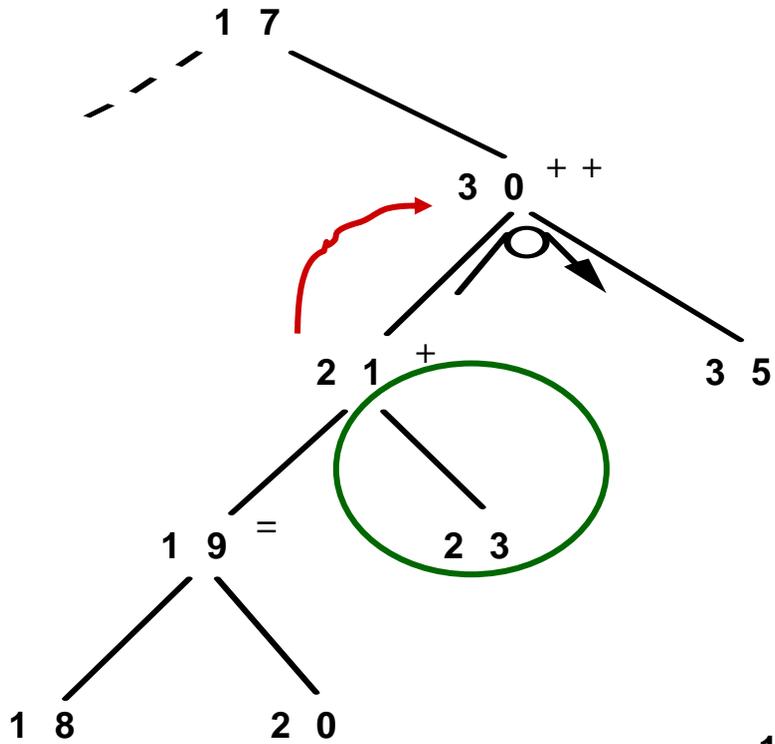
problème: maintenir l'équilibre

notations des déséquilibres:

- =** si $h(a.g) = h(a.d)$
- si $h(a.g) = h(a.d) - 1$
- si $h(a.g) = h(a.d) - 2$
- +** si $h(a.g) = h(a.d) + 1$
- ++** si $h(a.g) = h(a.d) + 2$

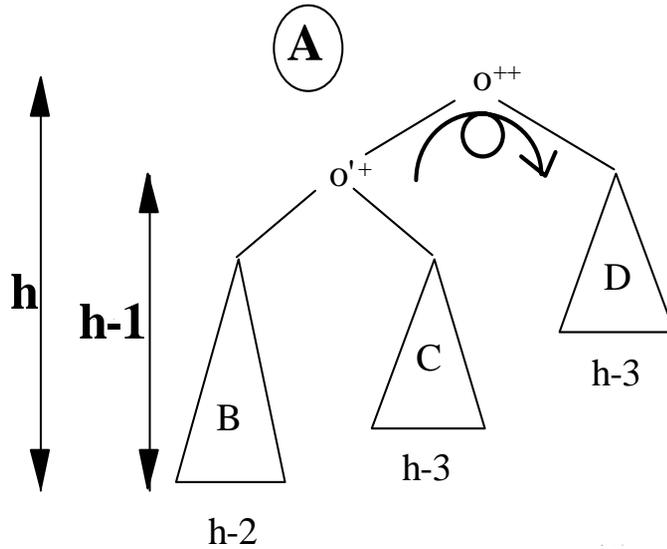


exemple de rotation



Ré-équilibrage 1

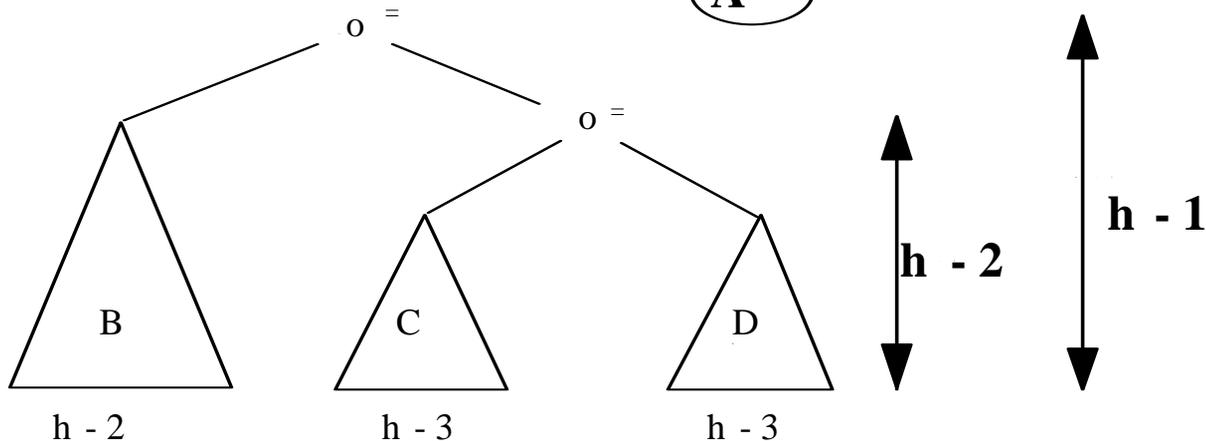
rééquilibrage: rotation à droite

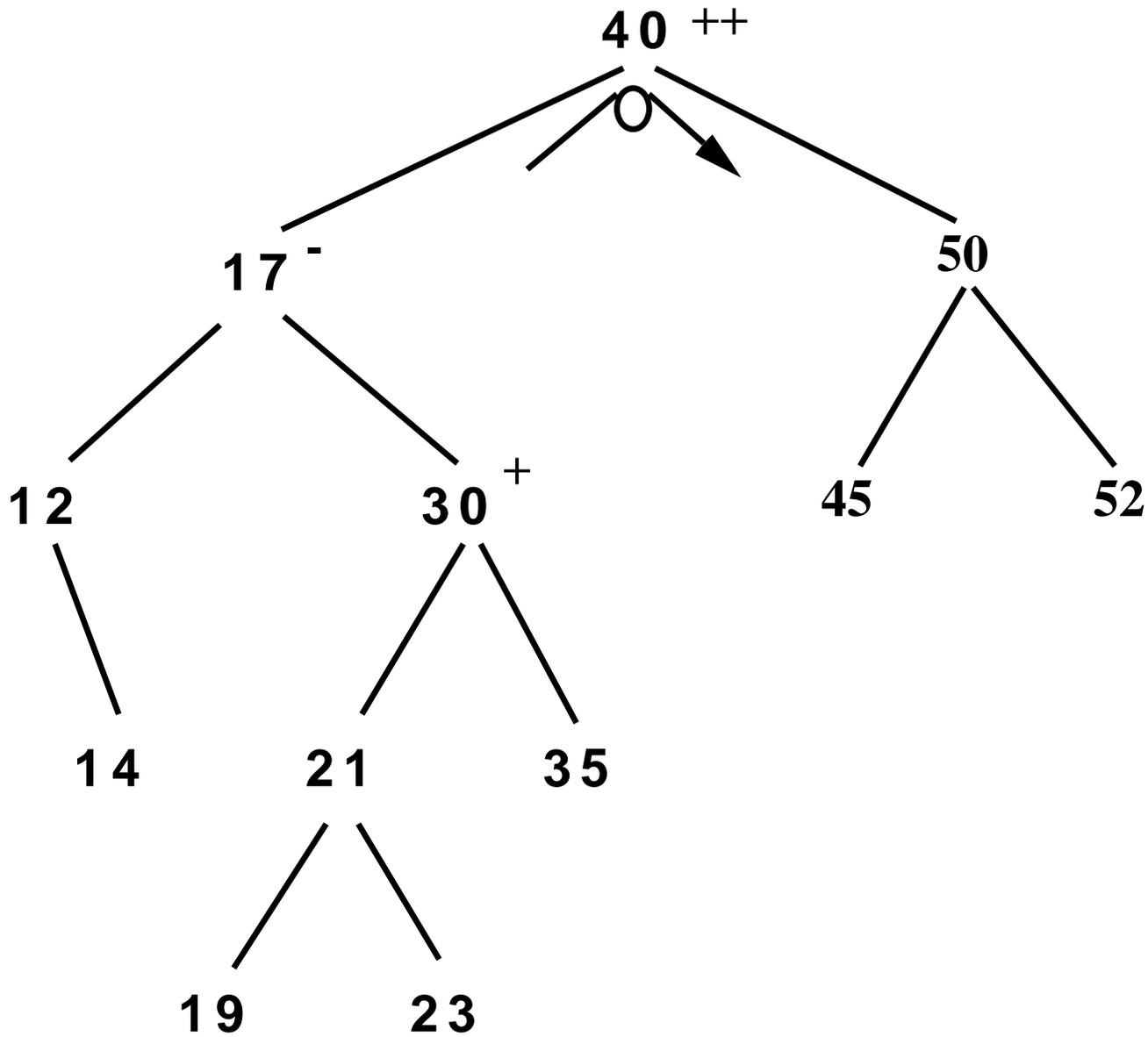


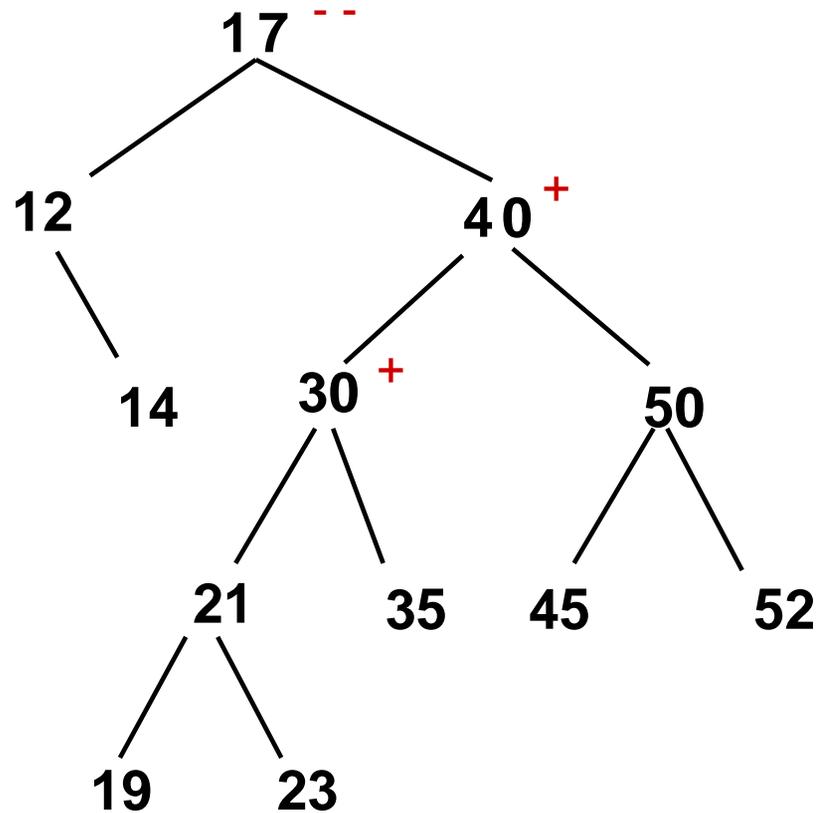
déséquilibre(A)=2, et
déséquilibre(A.g)=1 ou 0 \Rightarrow

A'

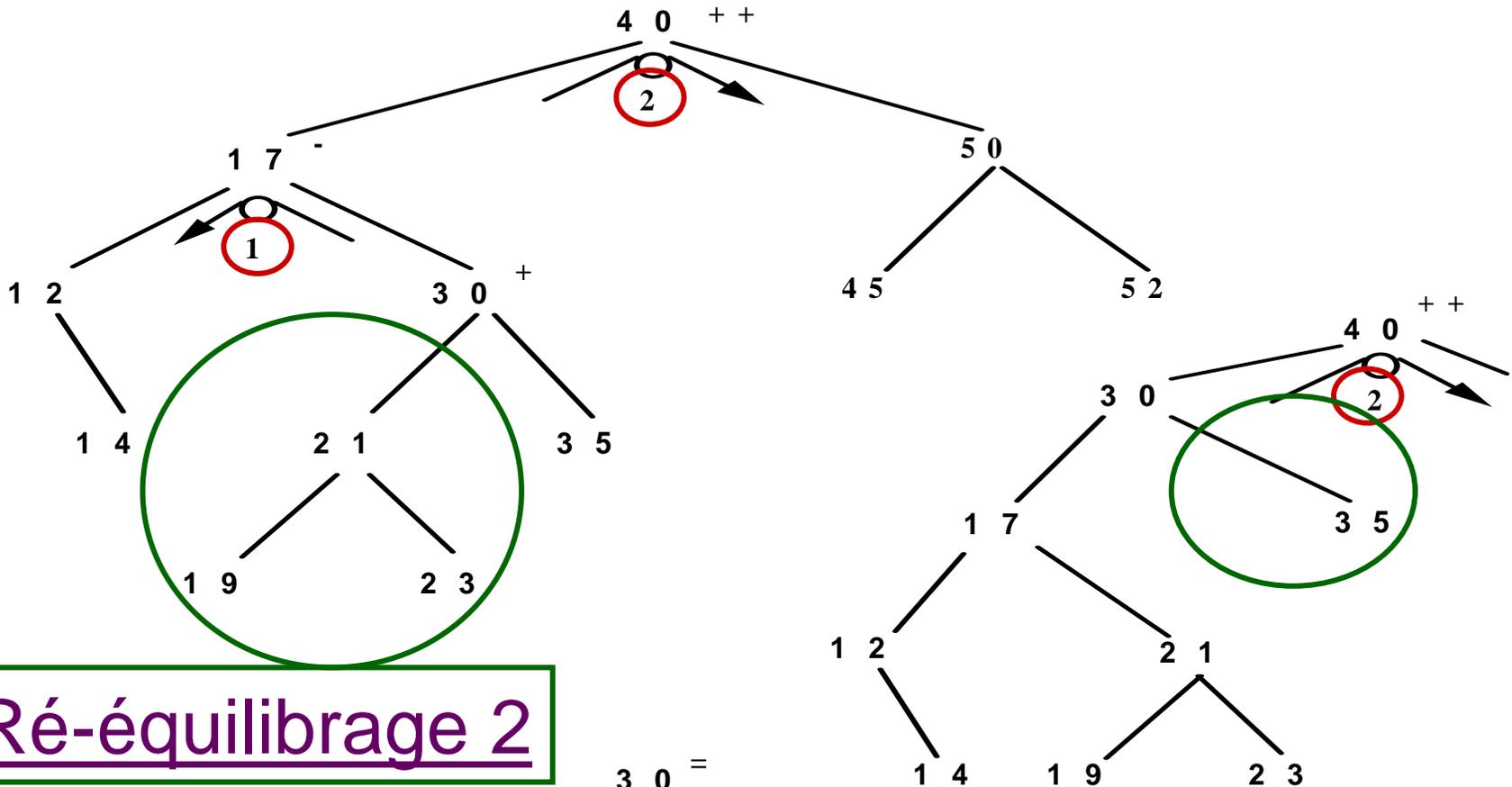
rotation à droite



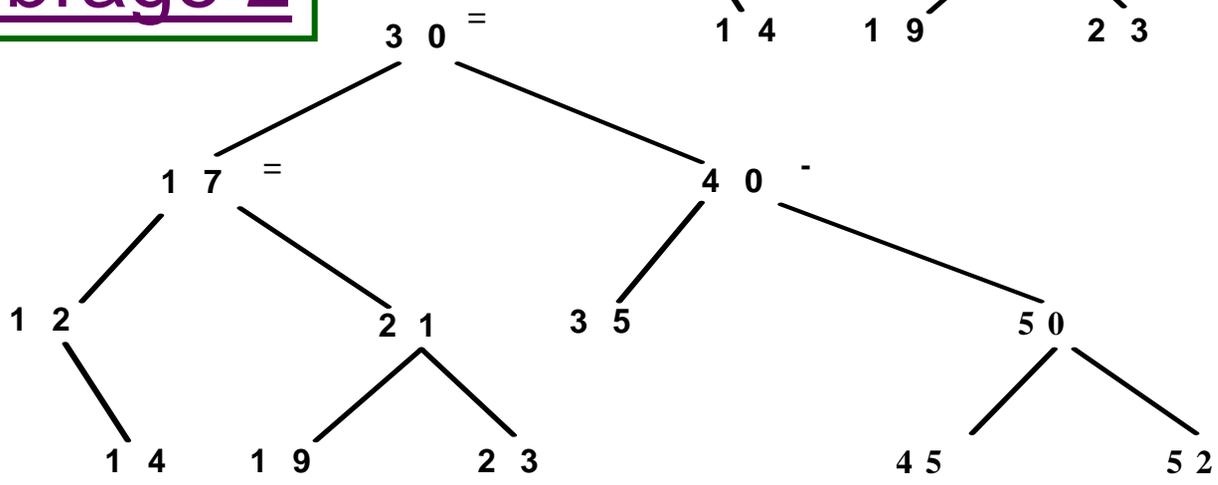




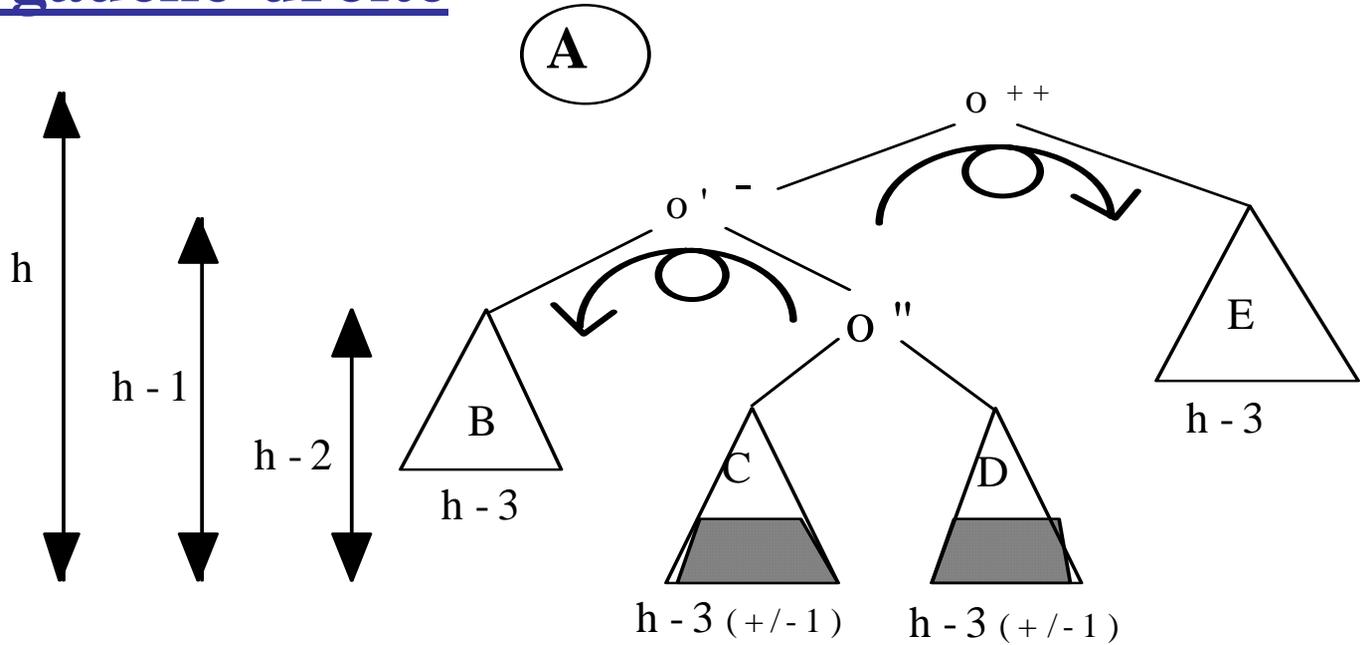
**La rotation simple apporte un nouveau déséquilibre:
double rotation nécessaire**



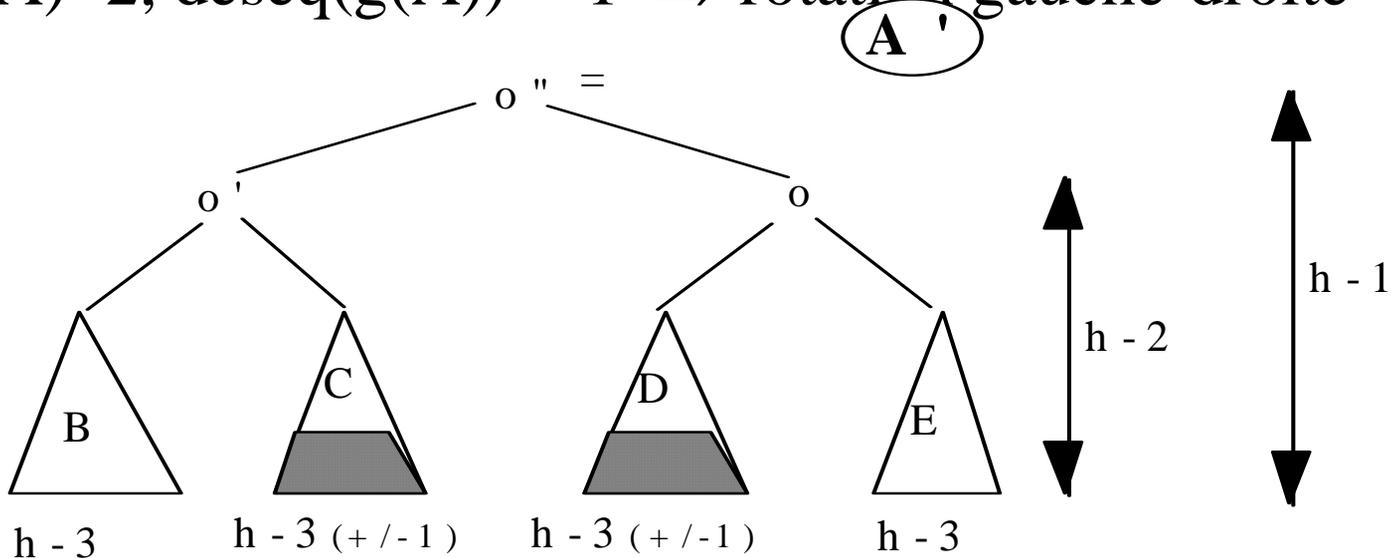
Ré-équilibrage 2



rotation gauche-droite



déséq(A)=2, déséq(g(A))=-1 \Rightarrow rotation gauche-droite



adjonction et suppression

même principe que dans un arbre binaire de recherche mais en rééquilibrant par rotations après l'opération si c'est nécessaire (si a n'est plus H équilibré)

Dans les arbres AVL

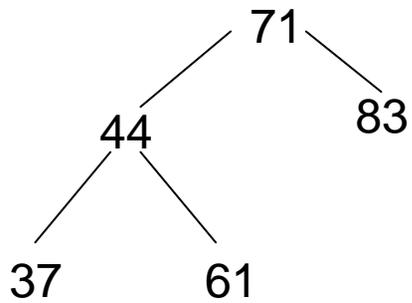
recherche		en $O(\log_2 n)$
adjonction		
suppression		

On constate expérimentalement:

en moyenne 1 rotation pour 2 adjonctions

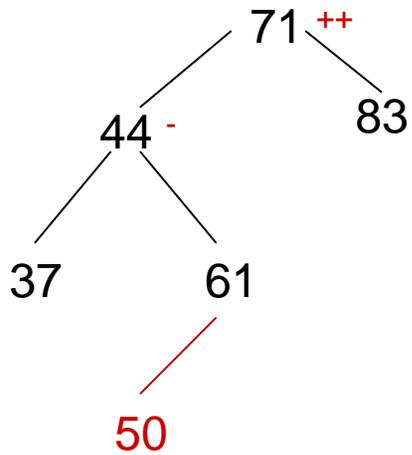
en moyenne 1 rotation pour 5 suppressions

Exemple



C'est bien un AVL

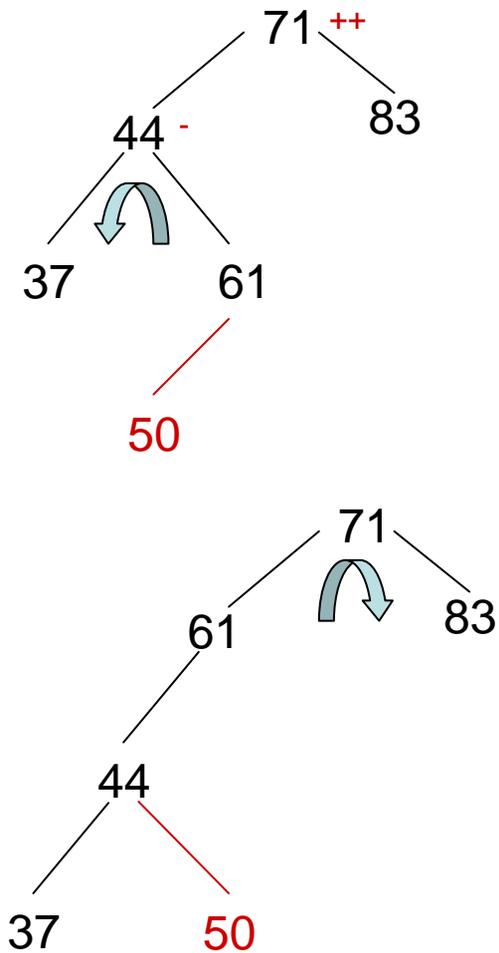
Exemple



Insertion de 50. **ABR** mais non **AVL**

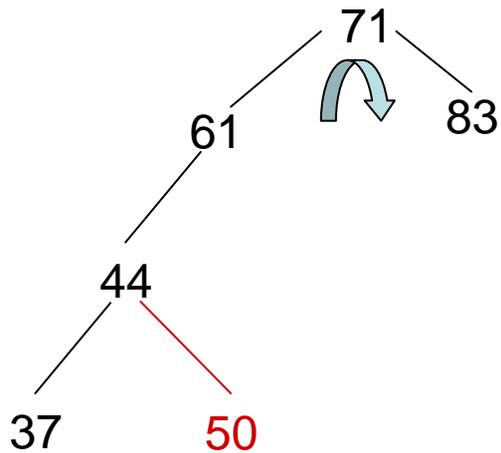
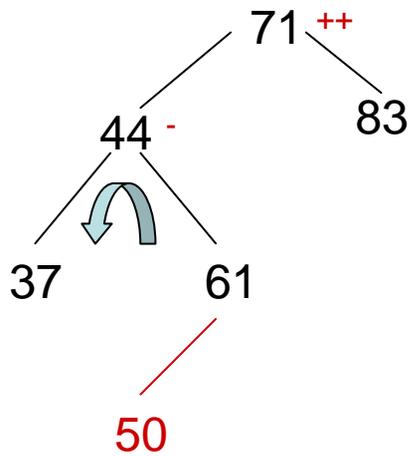
Exemple

Double rotation gauche puis droite

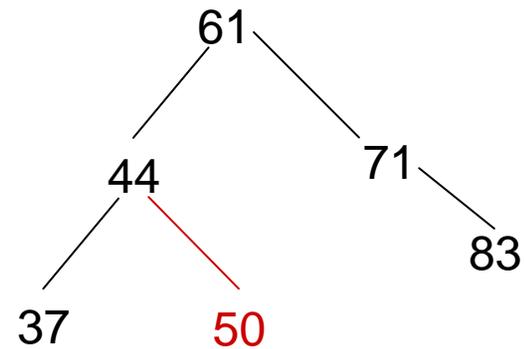


Exemple

Double rotation gauche puis droite



puis



AVL