7
ARBRES DE RECHERCHE

PLAN

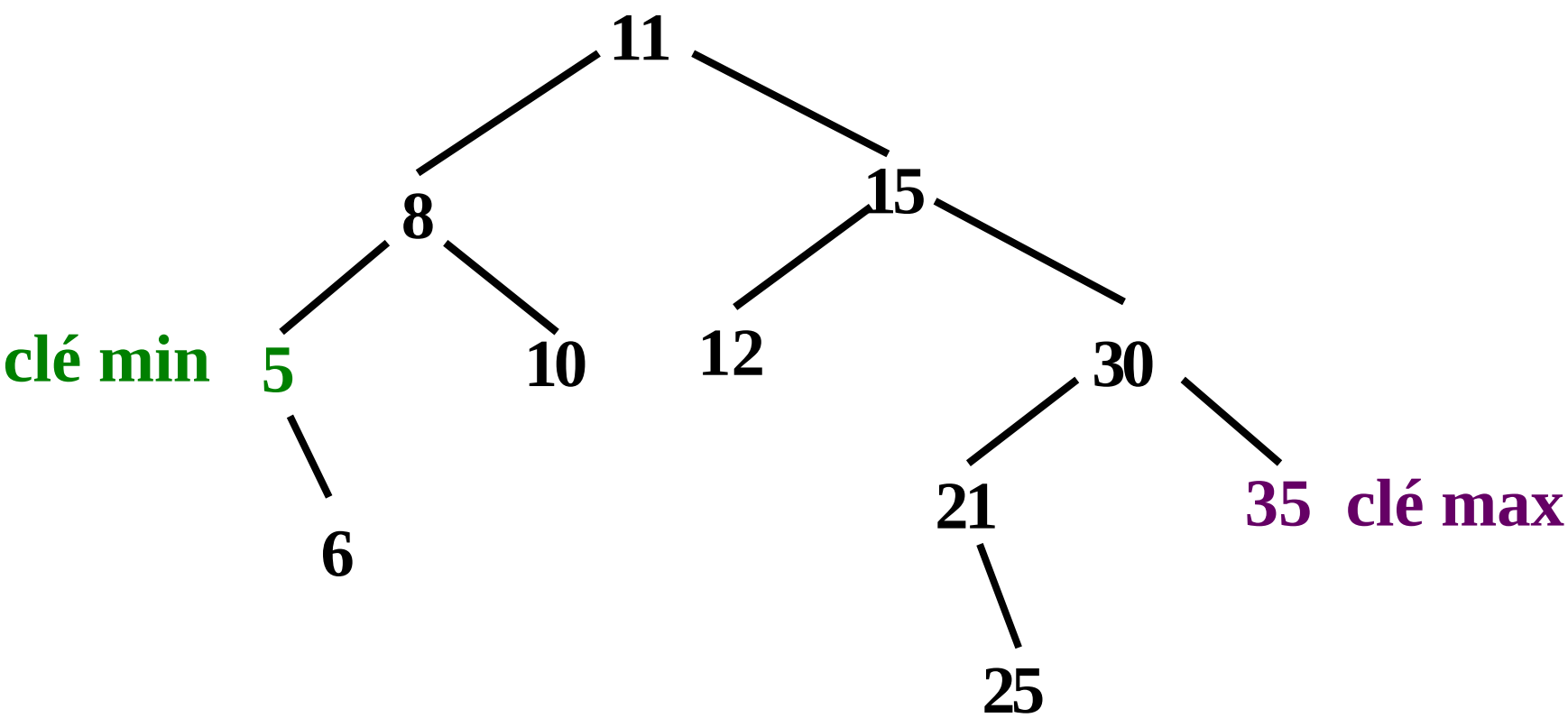
- **Arbres binaires de recherche**
- **Arbre h-équilibrés**
- **Arbres balancés (b-arbres)**
- **Fichiers séquentiels indexés**

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



RAPPEL: Voir cours n° 3 sur les arbres

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

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

```
classe ABR {
Ccle cle;
Elt  element;
ABR  gauche;
ABR  droit;
ABR (Ccle c, Elt e, Arbre g, Arbre d){
    this.cle = c;
    this.element = e;
    this.gauche = g;
    this.droit = d;
    }
//les méthodes}
```

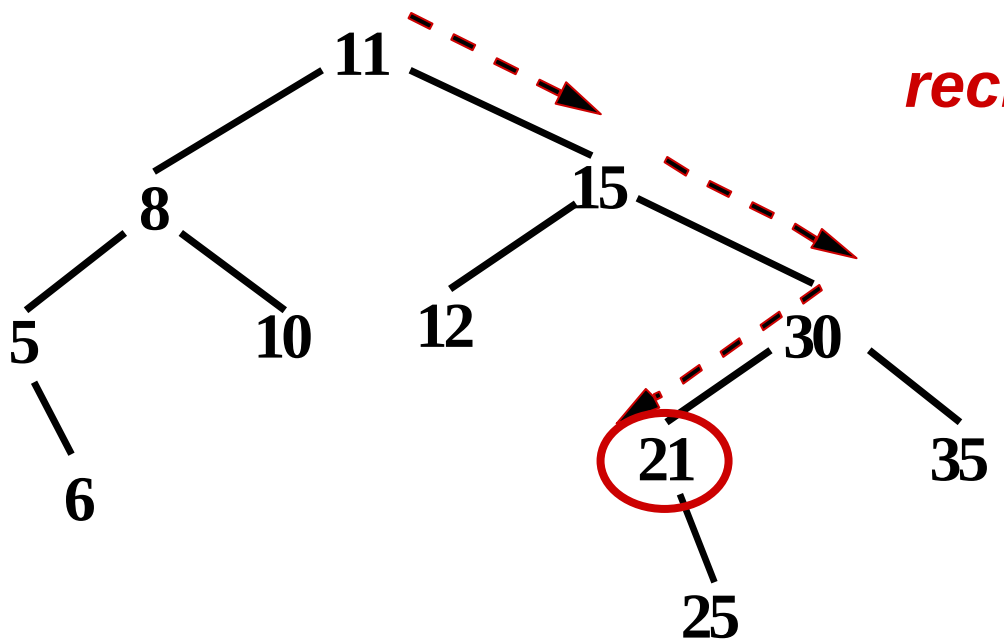
opérations sur un arbre ABR:

- **recherche**
- **adjonction**
- **suppression d'un élément e de clé c**
- **concaténation de 2 arbres en 1 arbre**
- **scission d'un arbre en 2 sous-arbres**

recherche dichotomique:

descente de l'arborescence avec comparaison en chaque nœud

EXEMPLE



recherche de 21

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

précondition: $c \in a$

recherche(c,a,e)

si **c == a.clé**

alors **e = a.element**

//élément de la racine de a

sinon si **c< a.cle**

alors **recherche(c,a.gauche,e)**

sinon **recherche(c,a.droit,e)**

finsi

finsi

complexité: O(h(a))

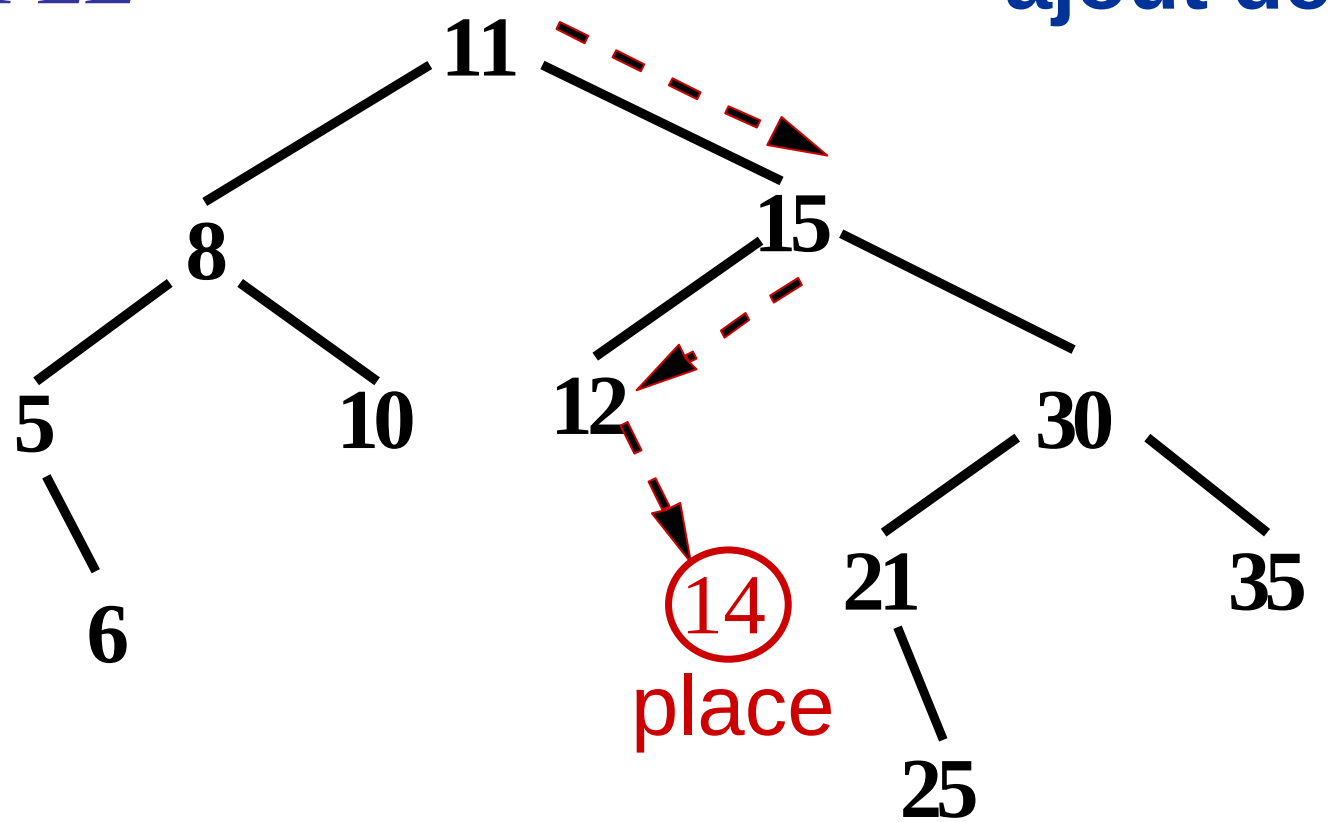
recherche de l'élément de clé c dans l'arbre a

```
Elt recherche (c CCle; a ABR; e Elt)
debut
    si c == a.cle ;
            alors retourner a.element ;
    sinon
          si c < a.cle et a.gauche=Ø
             ou   c > a.cle et a.droit =Ø
                    alors "élément absent";
          sinon   si c< a.cle
                    alors    retourner recherche(c,a.gauche)
                    sinon    retourner recherche(c,a.droit)
                    finsi;
          finsi;
    finsi;
fin
```

ADJONCTION D'UNE FEUILLE

EXEMPLE

ajout de 14



complexité: O(h(a))

adjonction dans a d'un élément e de clé c

```
ABR  ajouter (ABR a, Elt e, CCle c)
b=a;
si b = Ø  alors  b=new ABR (c, e, Ø, Ø ) ;
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 == Ø  alors place_trouvée=vrai //fils gauche de b
                                  b.gauche=new ABR (c, e, Ø, Ø ) ;
          sinon b = b.gauche;  finsi;
        sinon //descendre à droite
          si b.droit == Ø  alors place trouvée=vrai //fils droit
                                  b.droit=new ABR (c, e, Ø, Ø ) ;
          sinon b =b.droit;  finsi;
        finsi;
    fait;
retourner b;
finsi;
```

Appel: a=ajouter(a,e,c);

suppression d'un nœud N

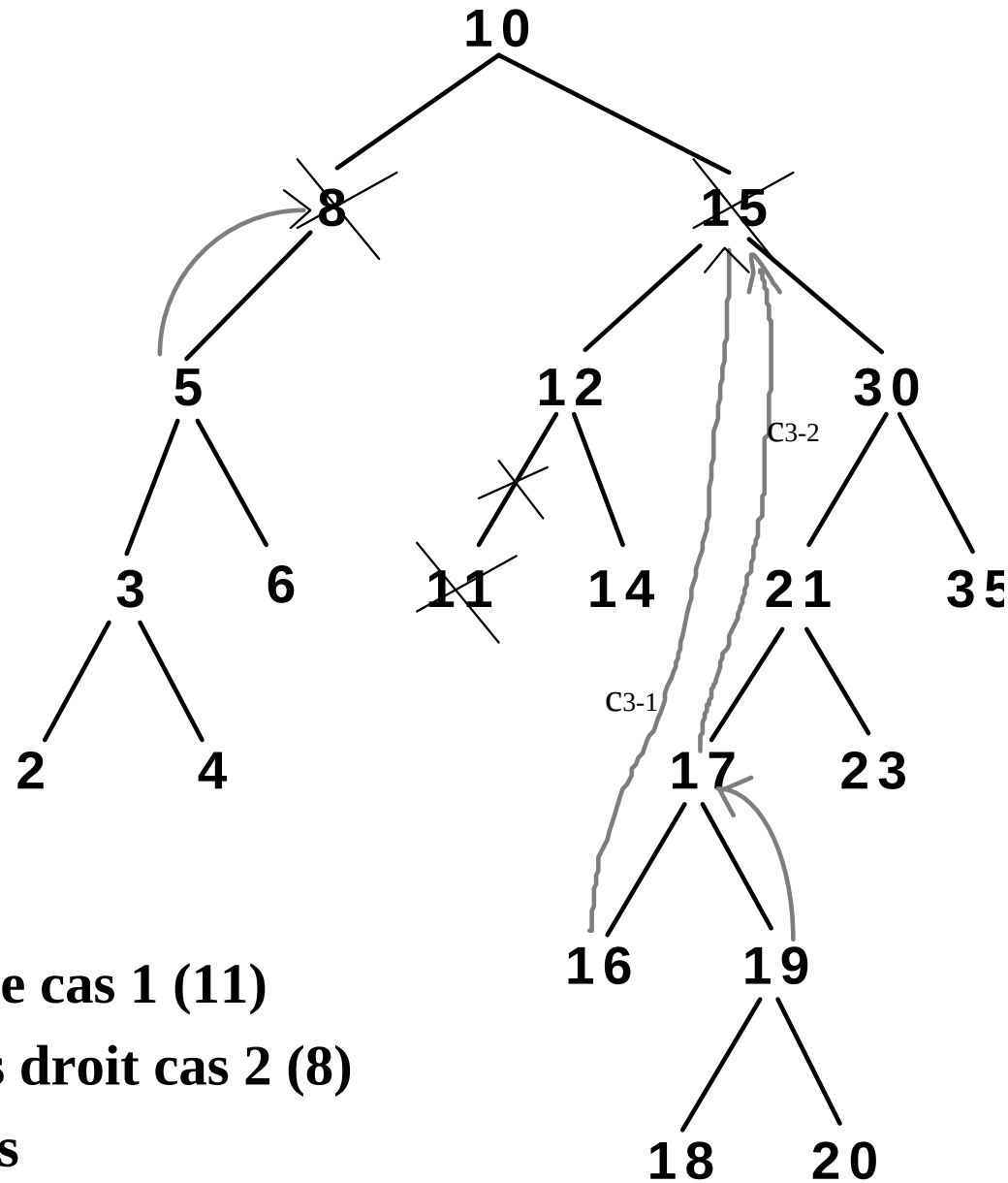
cas 1: **N est une feuille**

cas 2: **N a un seul fils F**

cas 3: **N a 2 fils**

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

EXEMPLE



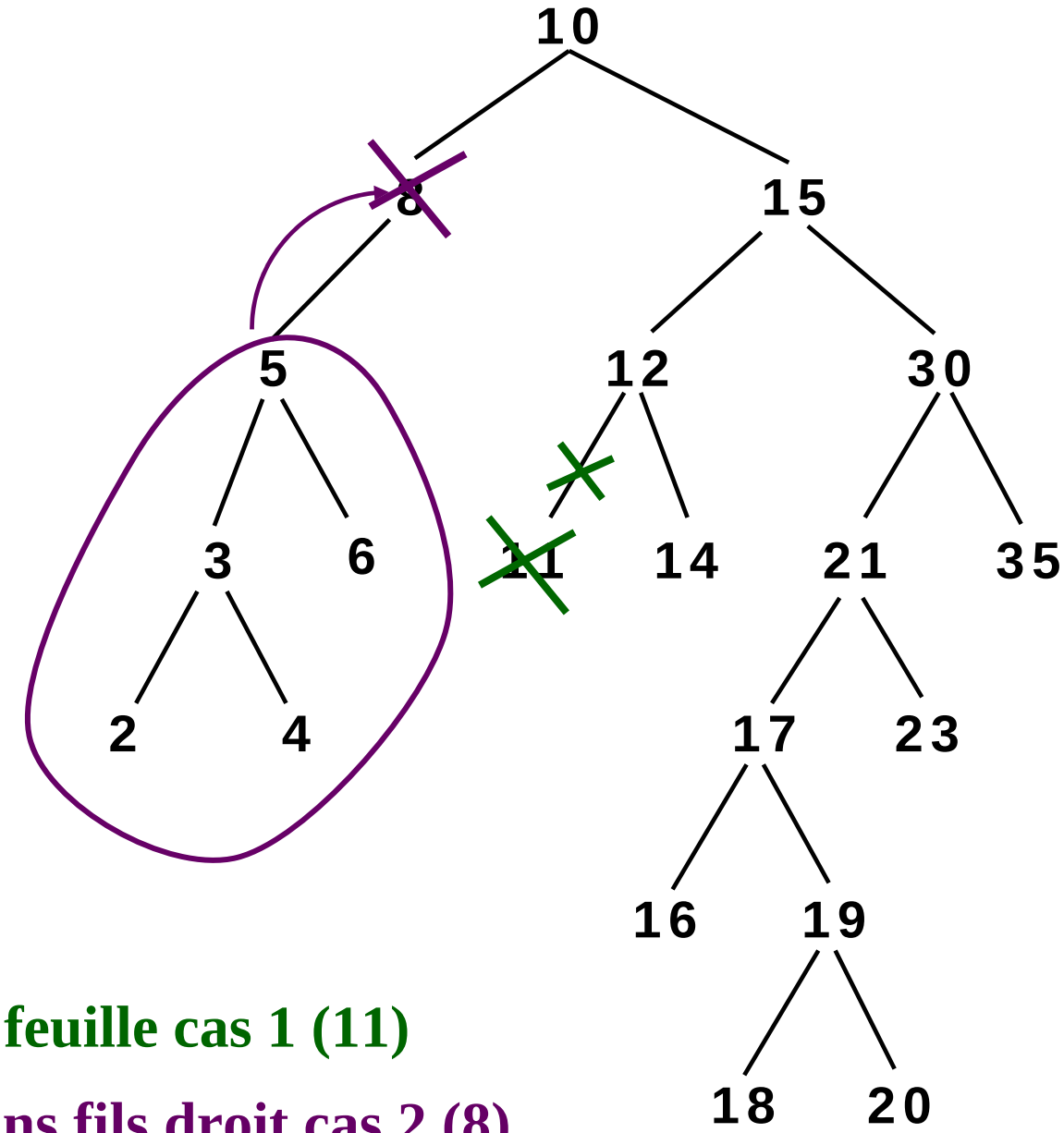
suppression d'une feuille cas 1 (11)

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

sup. de nœuds avec 2 fils

cas 3-1 (15) puis cas 3-2 (16)

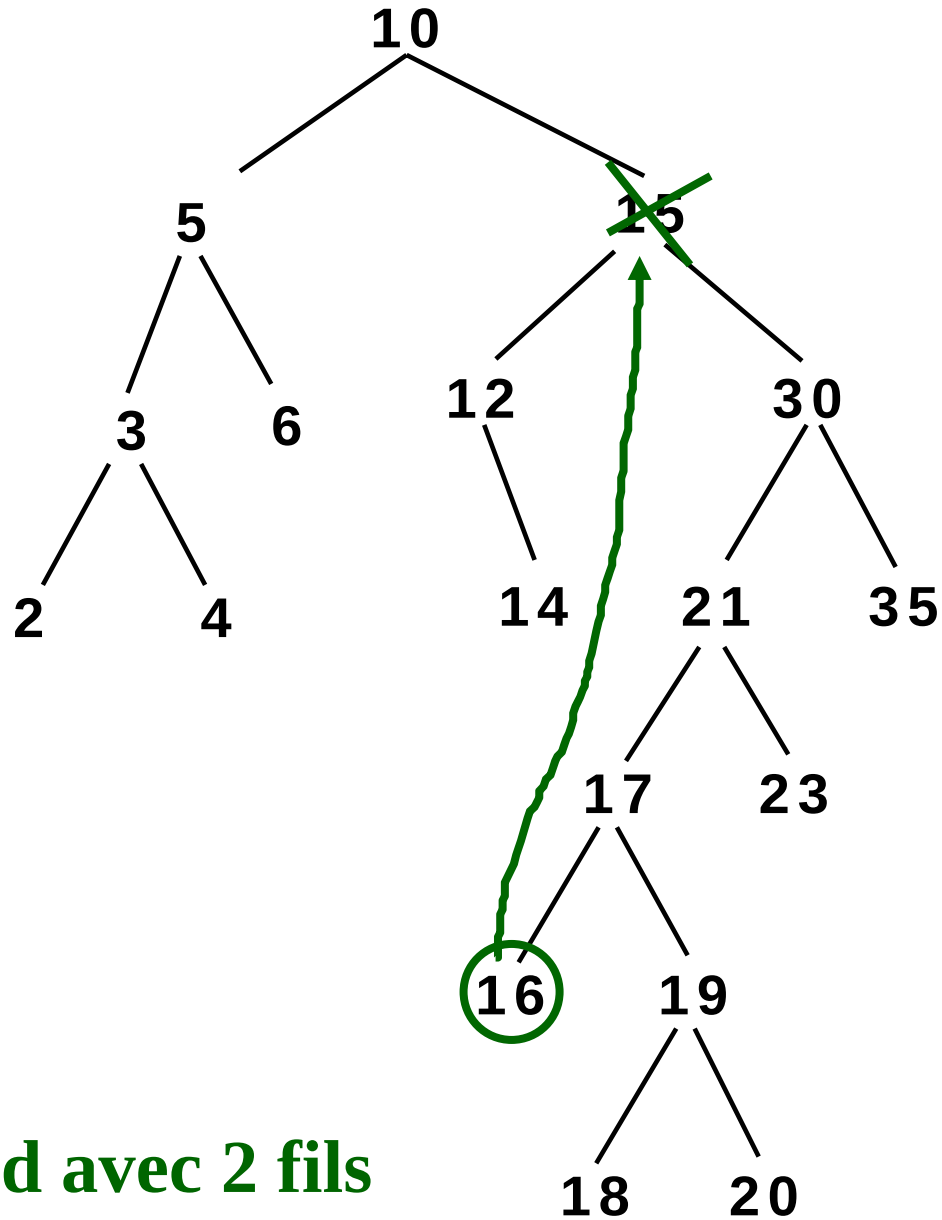
EXEMPLE



suppression d'une feuille cas 1 (11)

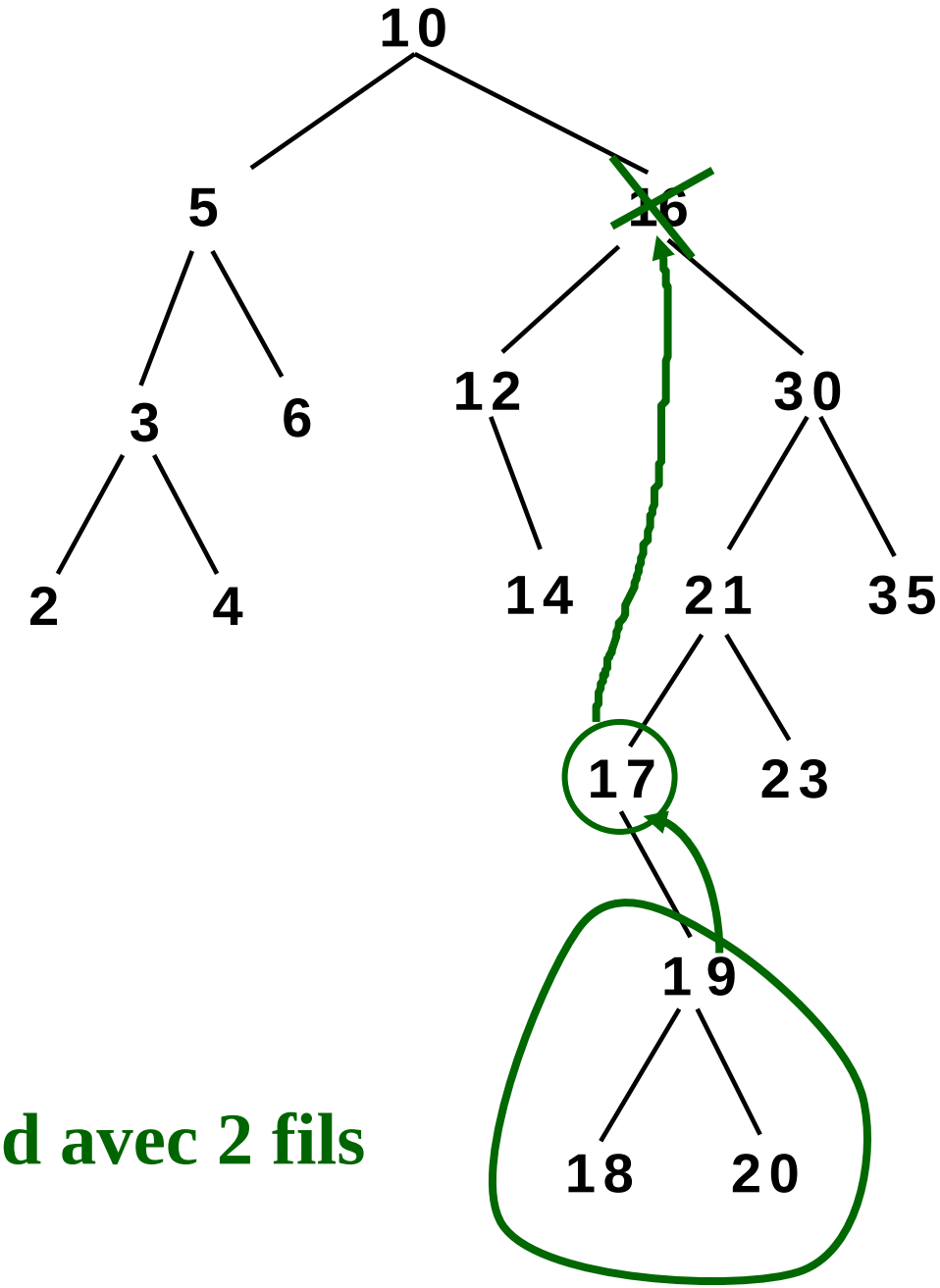
sup. d'un nœud sans fils droit cas 2 (8)

EXEMPLE



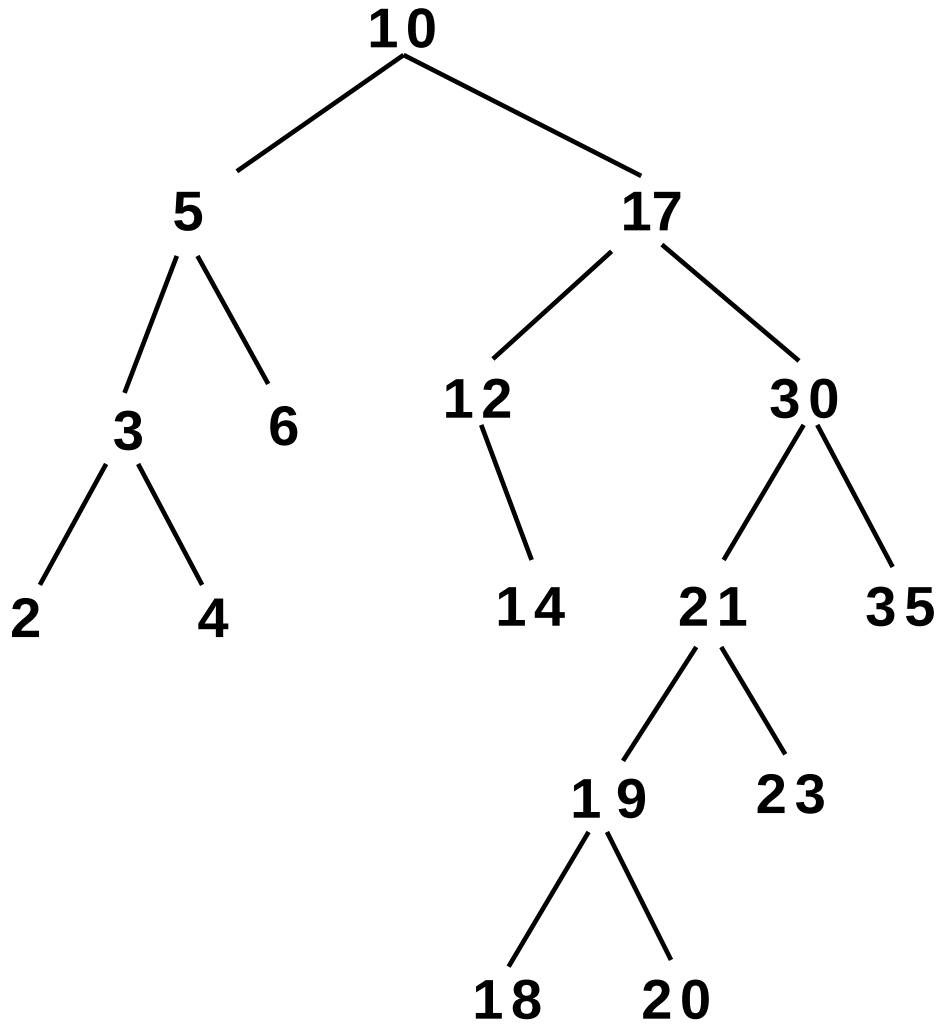
suppression d'un nœud avec 2 fils
cas 3-1 (15)

EXEMPLE

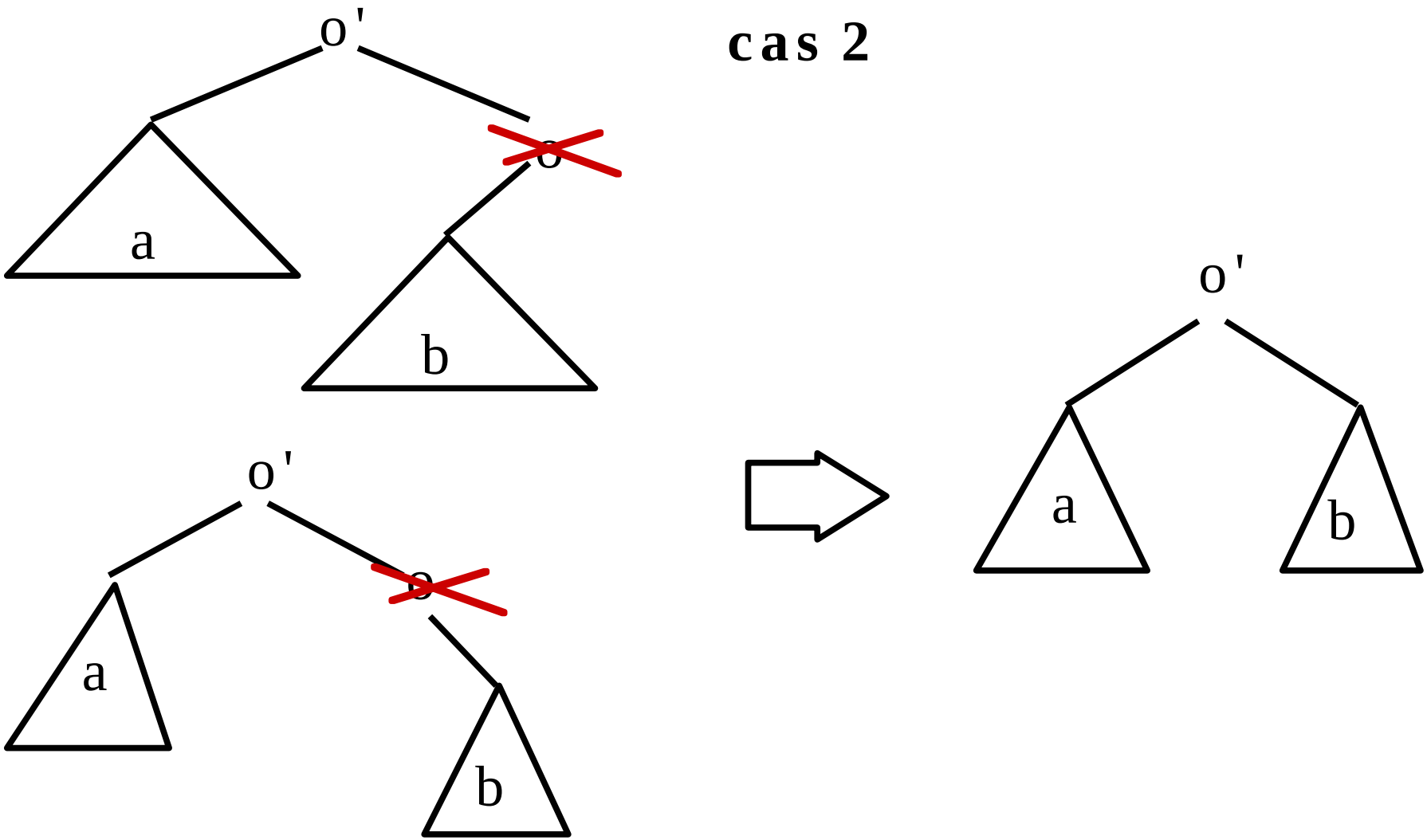


suppression d'un nœud avec 2 fils
cas 3-2 (16)

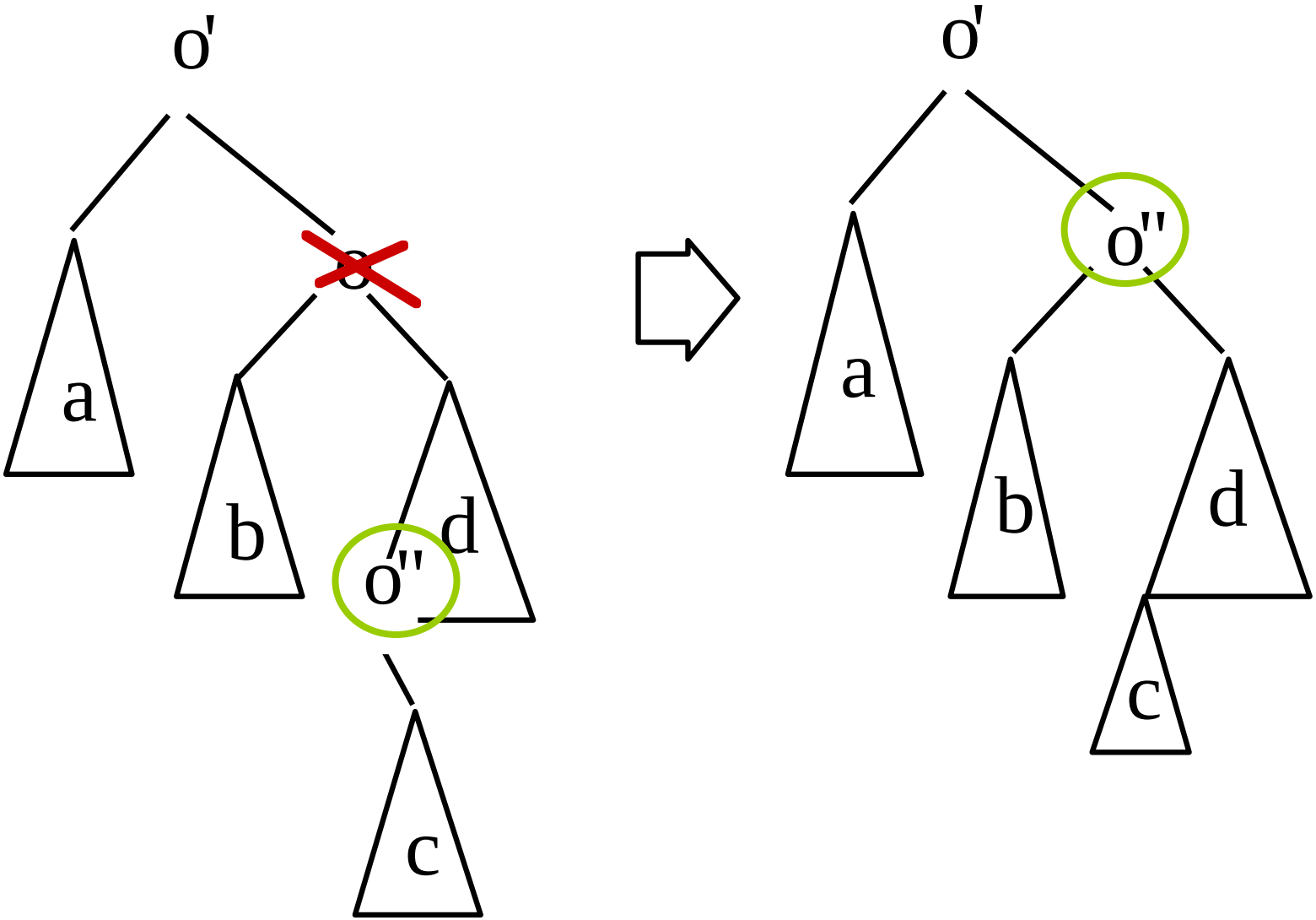
EXEMPLE



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**

```
classe ABR {
Ccle cle;
Elt  element;
ABR  gauche;
ABR  droit;
ABR pere;
ABR (Ccle c, Elt e, ABR g, ABR d, ABR p){
    this.cle = c;
    this.element = e;
    this.gauche = g;
    this.droit = d;
    this.pere = p;
    }
//les méthodes}
```

suppression d'un élément de clé c au nœud a

(obtenu par recherche)

cas 1 **a est une feuille (a.g=a.d=Ø):**

si **a.père.cle > c** alors **a.père.g = Ø**

sinon **a.père.d = Ø**

cas 2 **a n'a pas de fils droit (a.g≠Ø, a.d=Ø):**

a = a.g *on remonte le fils gauche*

ou **a n'a pas de fils gauche (a.d≠Ø, a.g=Ø):**

a = a.d *on remonte le fils droit*

cas 3 **a a 2 fils:**

rechercher le plus petit du sous-arbre de droite

f = a.d;

tant que **f.g≠Ø** 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 =Ø 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)

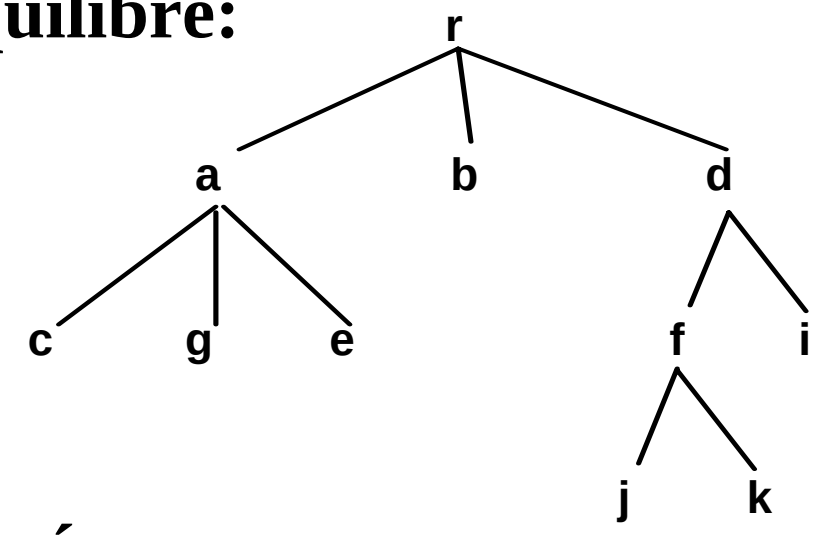
- **problème:**

  éviter les cas défavorables

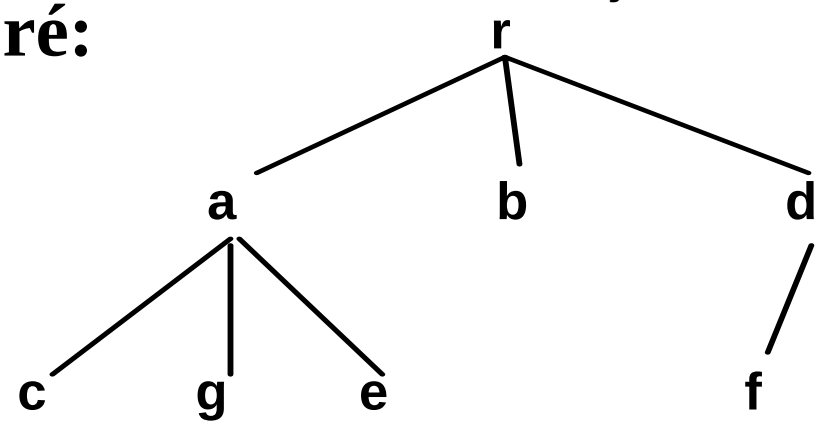
  $\rightarrow$ arbres équilibrés

équilibre

arbre non équilibré:



arbre équilibré:



les feuilles sont sur au plus 2 niveaux

8.3 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:

$$déseq(a) = h(a.g) - h(a.d)$$

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

arbre H-équilibré

pour tout b sous-arbre de a,

$$déséq(b) \in \{-1, 0, +1\}$$

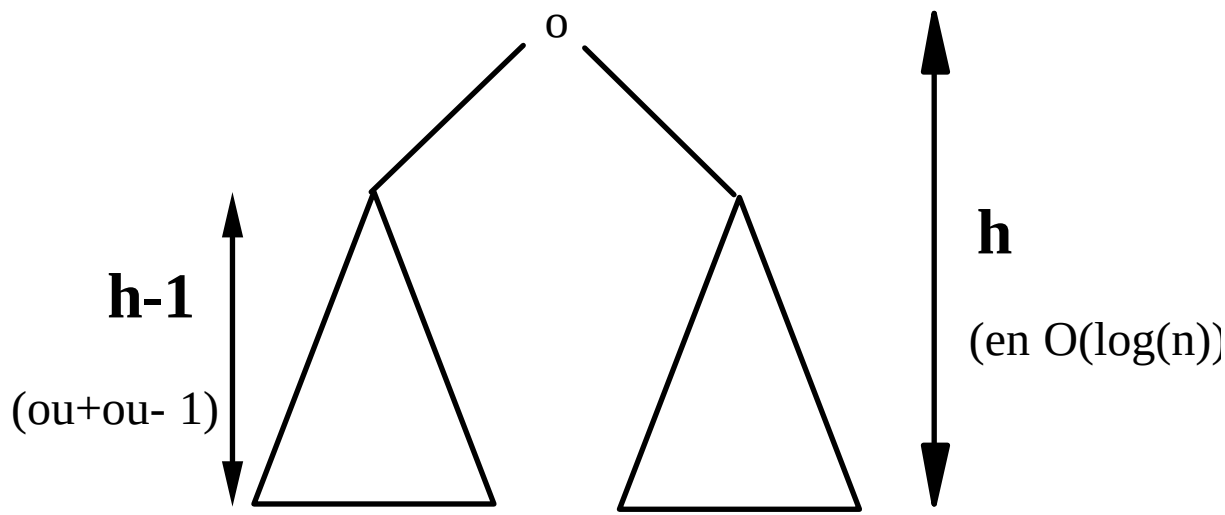
arbre H-équilibré $\Rightarrow \log_2(n+1) \leq h + 1 \leq 1.45\log_2(n+1)$

arbre AVL:

arbre binaire de recherche

H-équilibré

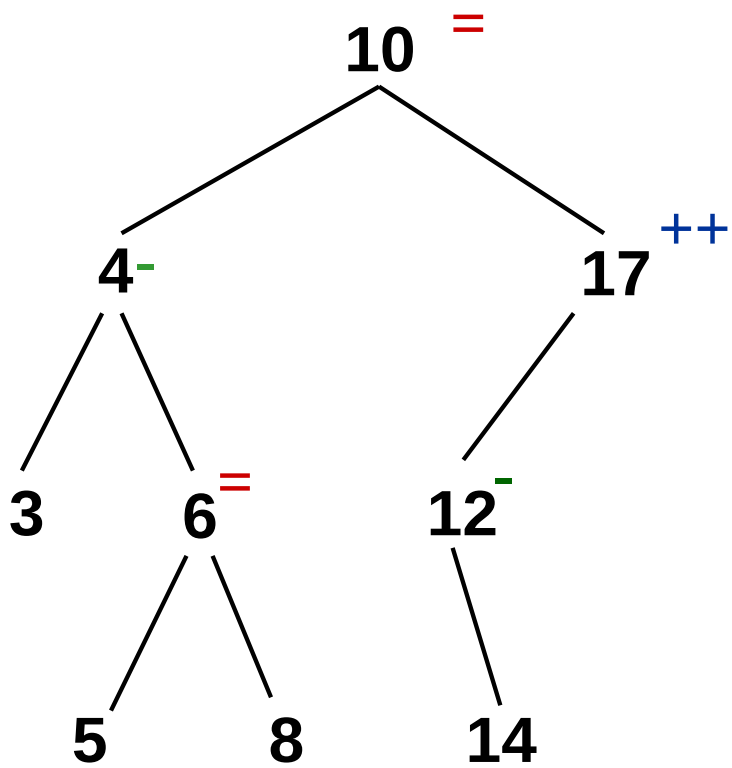
⇒ ***la recherche dans un arbre AVL est au pire en $O(\log_2 n)$***



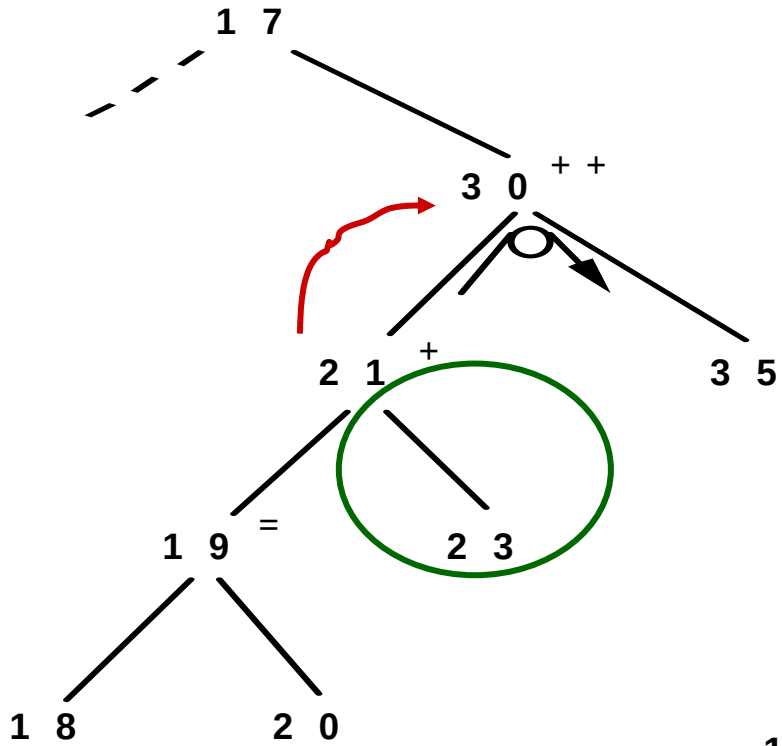
problème: maintenir l'équilibre

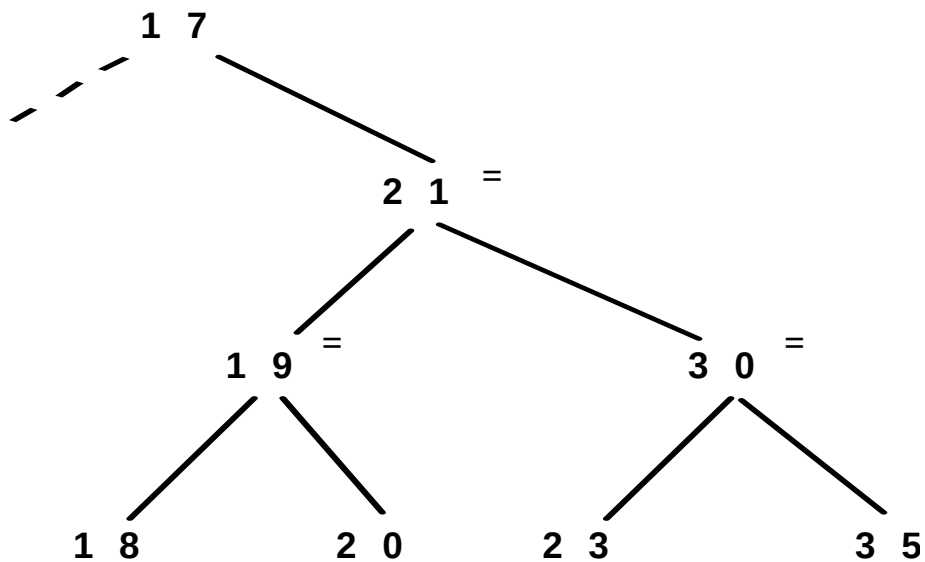
notations des déséquilibres:

=	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



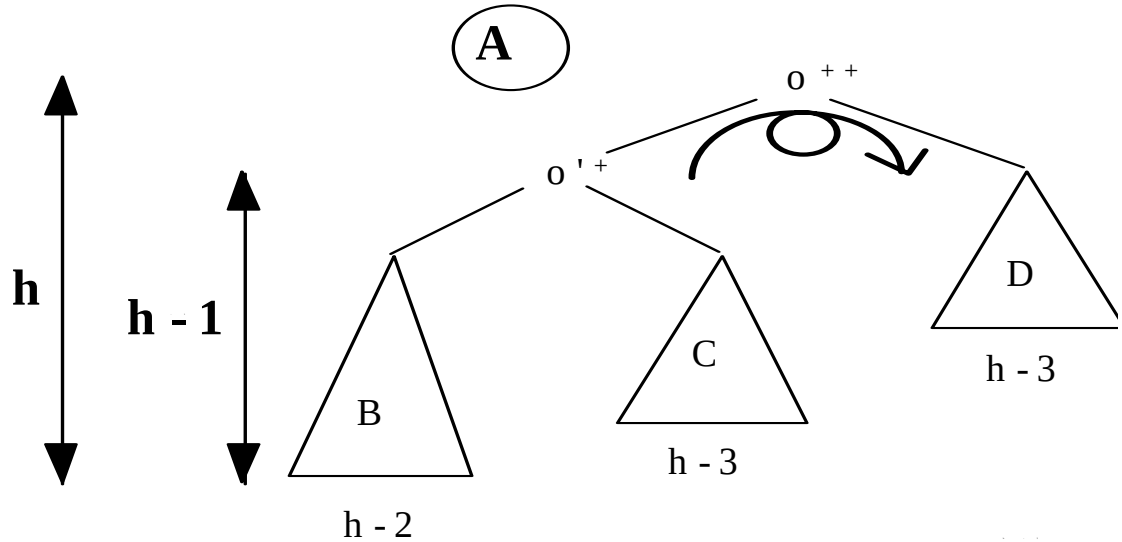
exemple de rotation



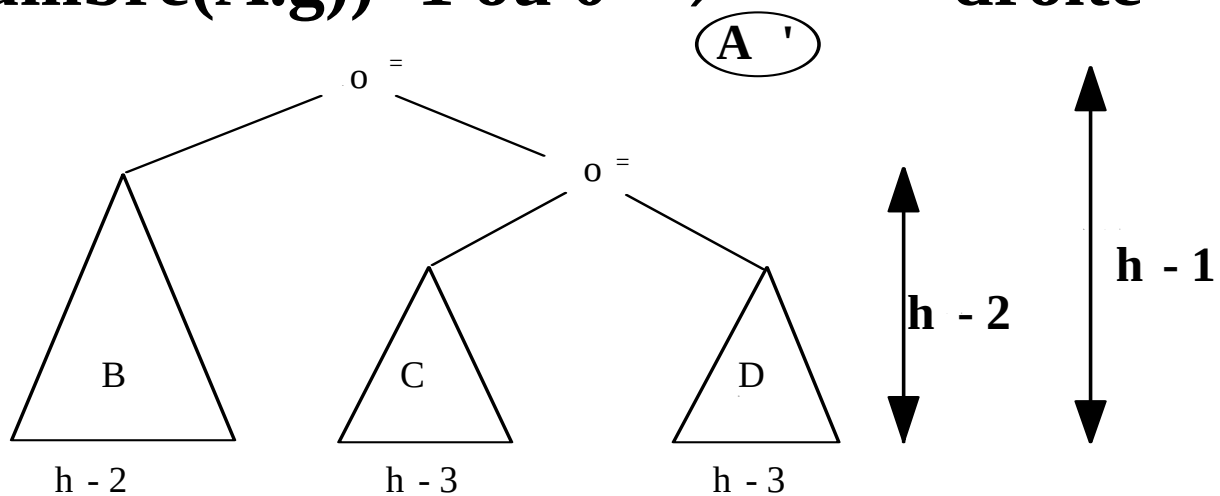


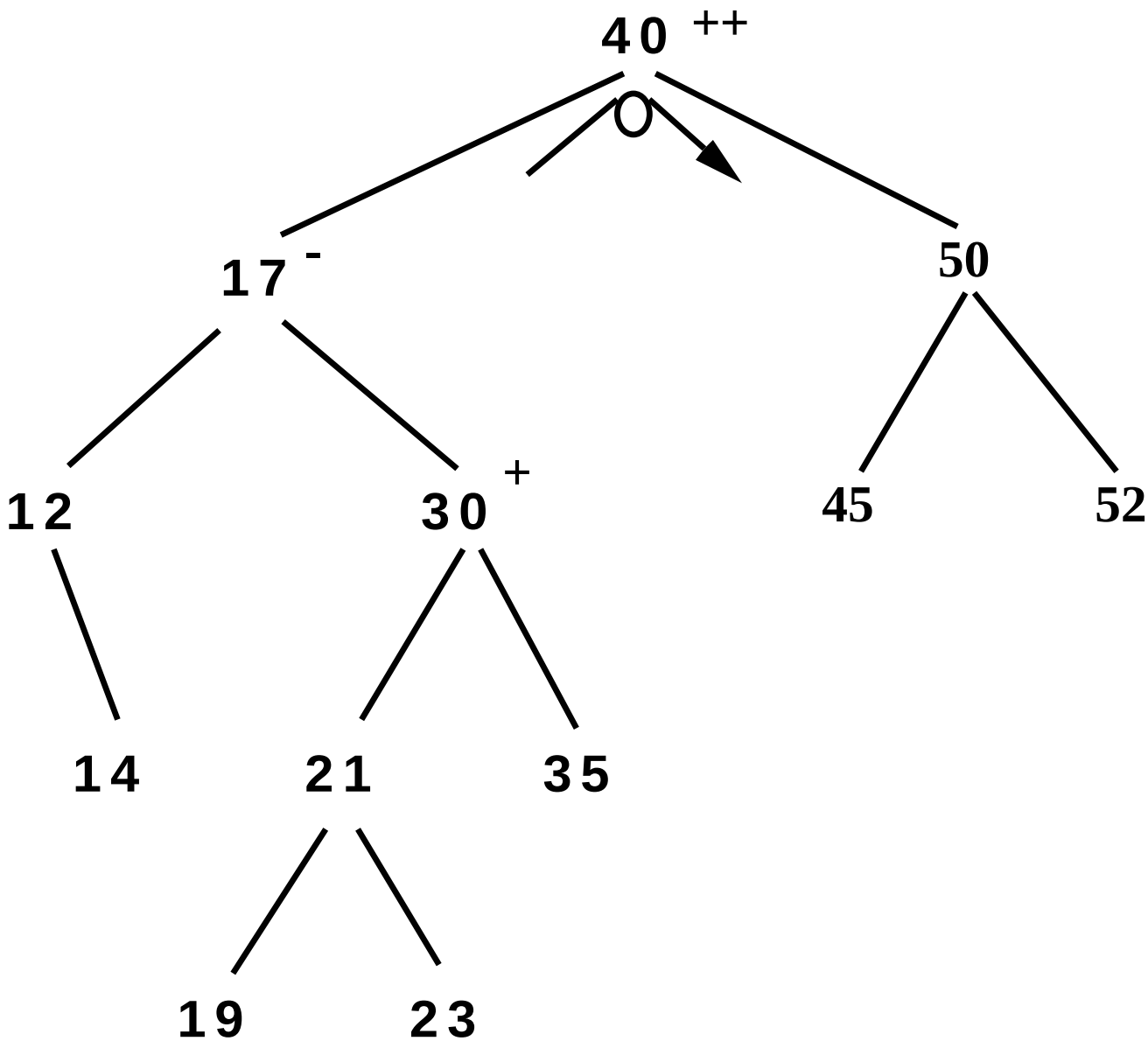
Ré-équilibrage 1

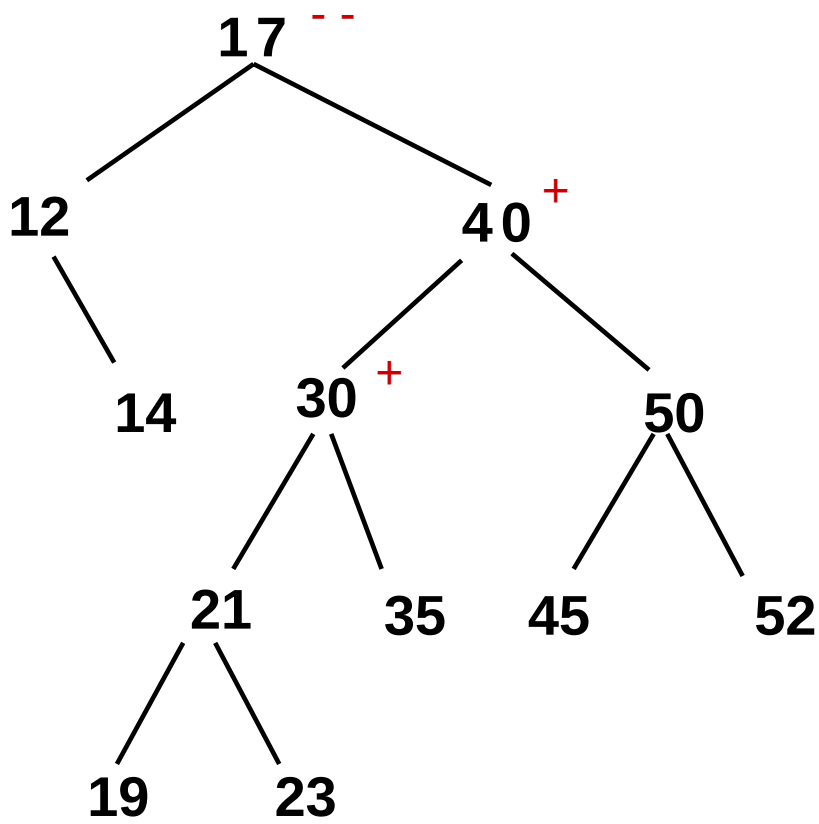
rééquilibrage: rotation à droite



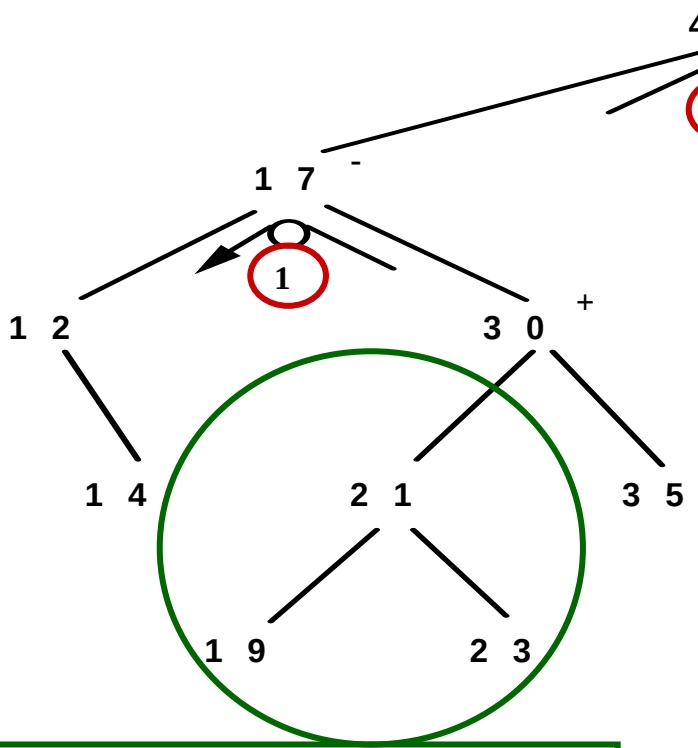
déséquilibre(A)=2, et déséquilibre(A.g))=1 ou 0 ⇒ rotation à droite

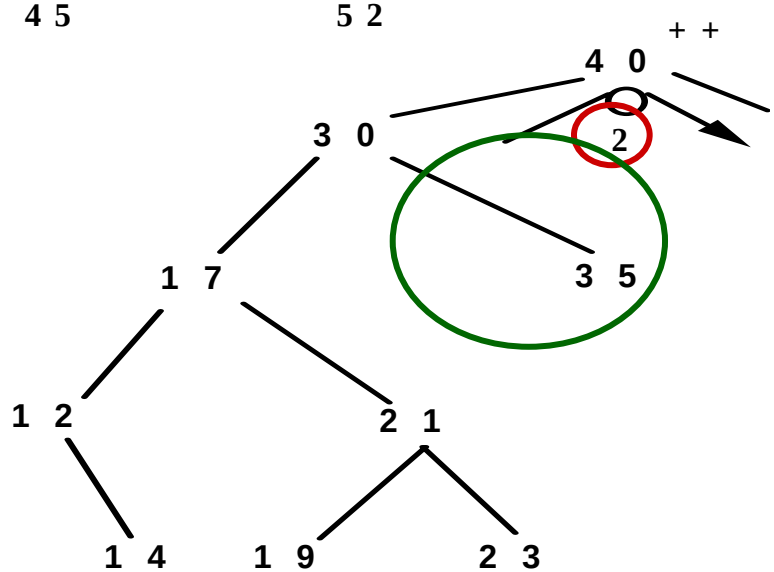



40 ++
17 -
50
12
30 +
45
52
14
21
35
19
23

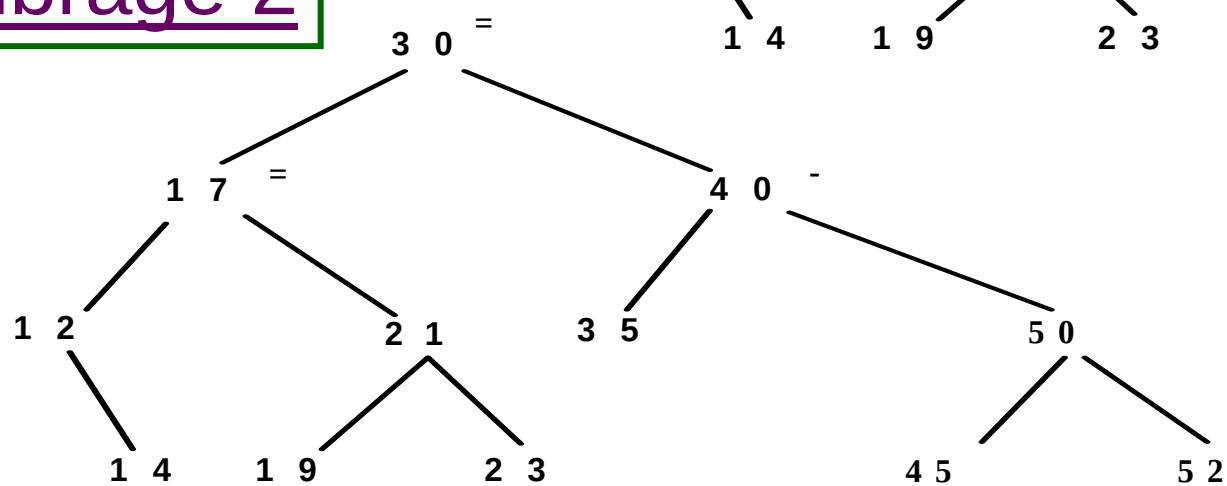


La rotation simple apporte un nouveau déséquilibre: double rotation nécessaire

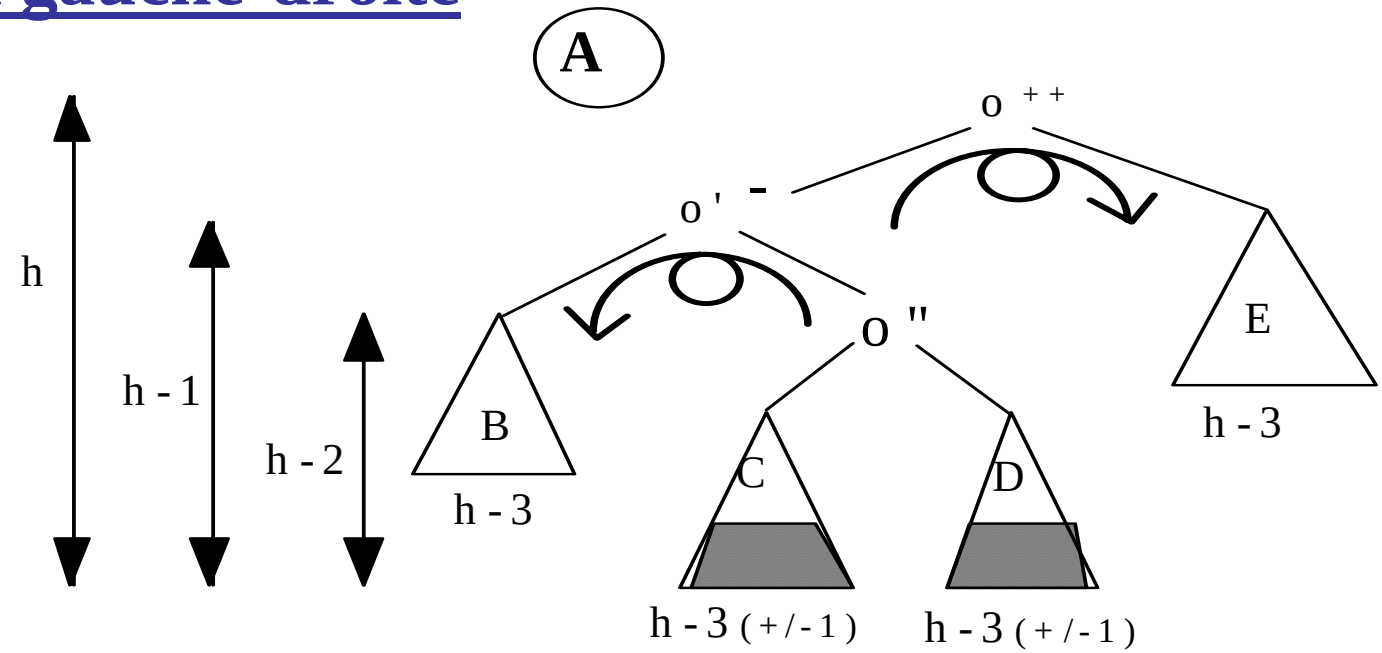




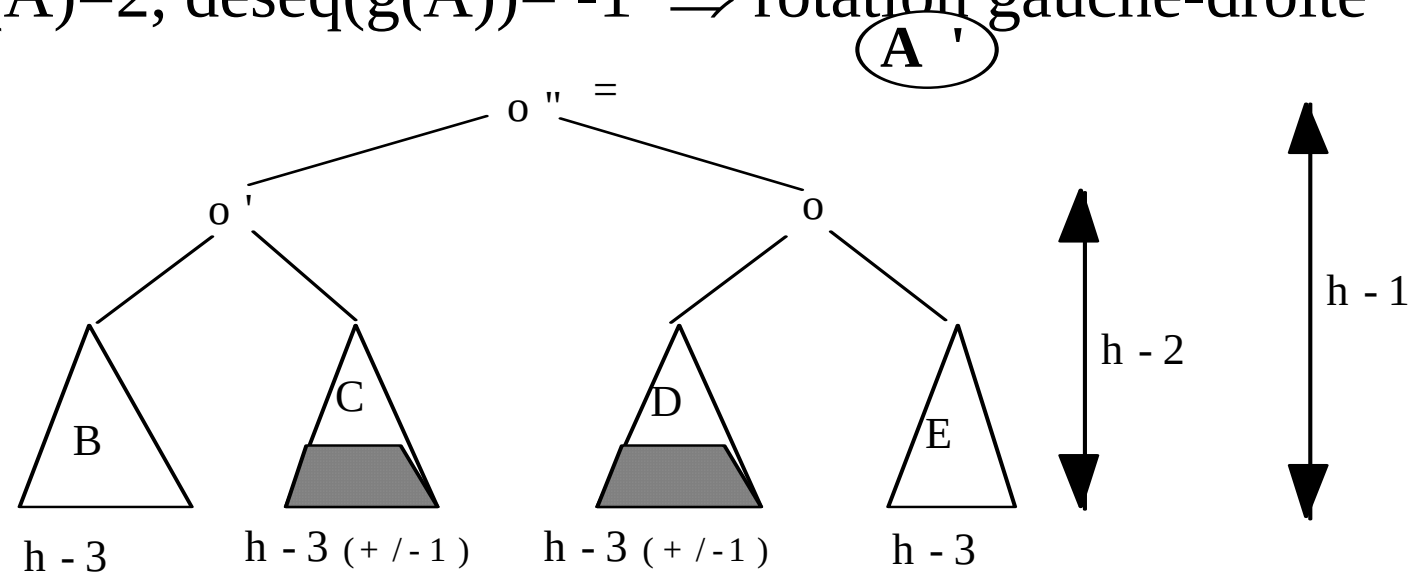
Ré-équilibrage 2



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

On constate expérimentalement:

en moyenne 1 rotation pour 2 adjonctions

en moyenne 1 rotation pour 5 suppressions

8.3 ARBRES BALANCES

idée

- **prendre en compte la taille des blocs disques**
  un bloc peut contenir plusieurs nœuds

 temps d'accès environ 20 ms

- **regrouper les nœuds voisins dans un même bloc**

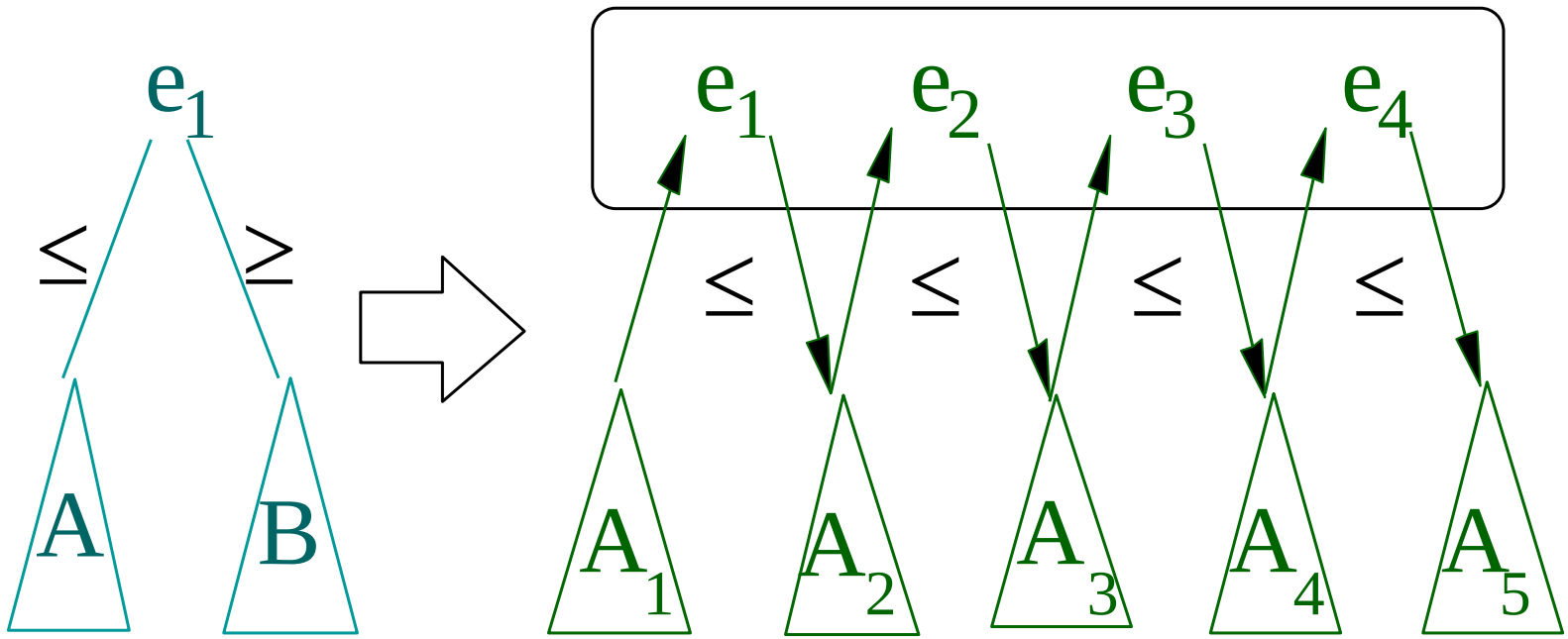
difficile à maintenir

solution

mettre plusieurs valeurs par nœuds

$\Rightarrow$ arbres généraux de recherche

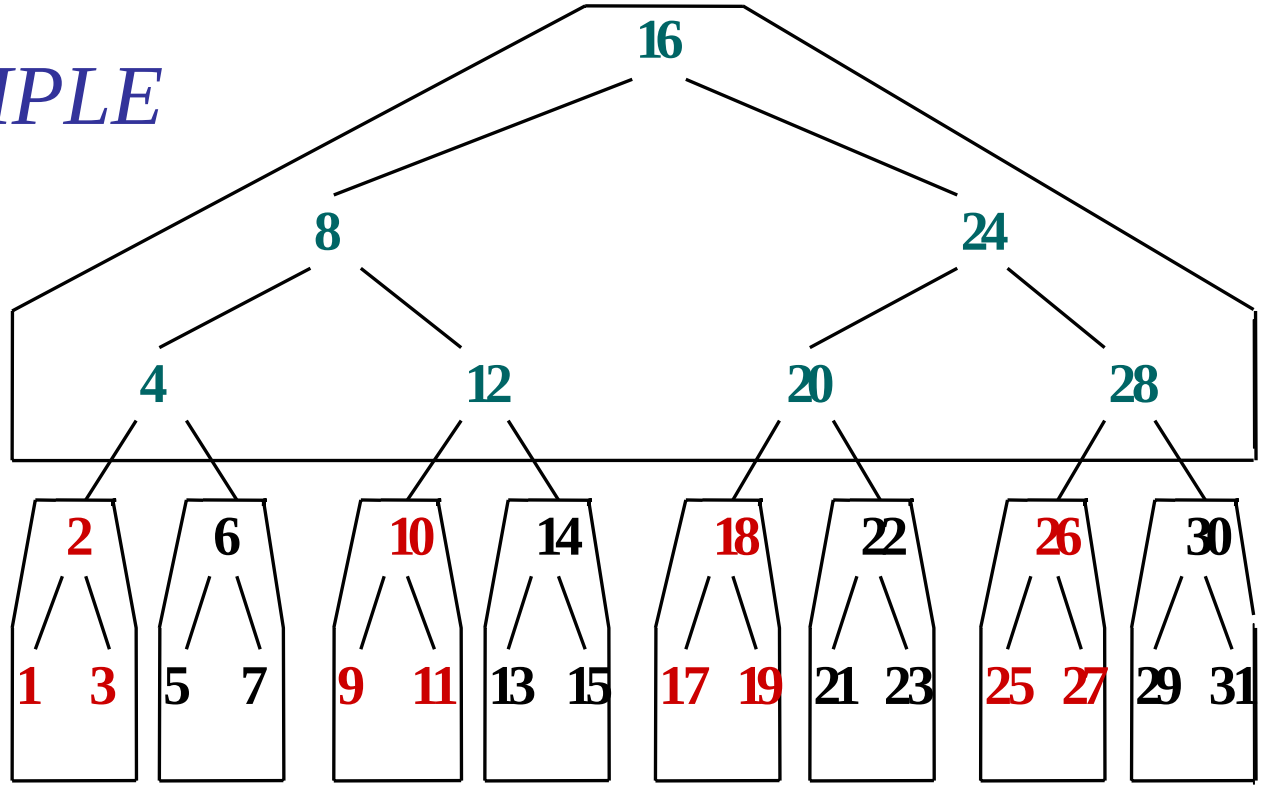
Principe:

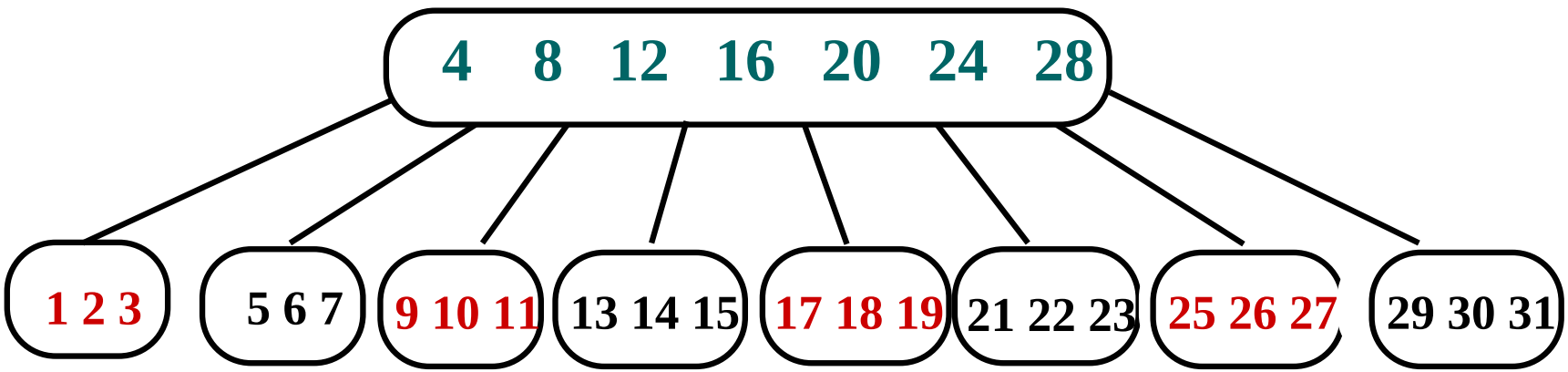


arbre binaire

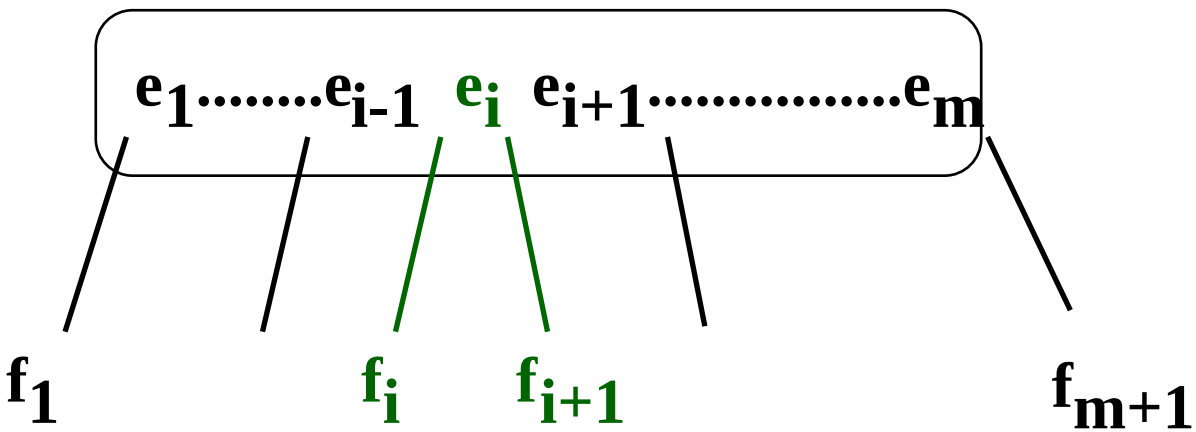
arbre général de recherche

EXEMPLE


16
8
24
4
12
20
28
2
6
10
14
18
22
26
30
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31


4 8 12 16 20 24 28
1 2 3
5 6 7
9 10 11
13 14 15
17 18 19
21 22 23
25 26 27
29 30 31

un nœud de m éléments
a m+1 fils



f_i : ième fils

$$\text{clé}(e_{i-1}) < \text{clés de } f_i < \text{clé}(e_i)$$

c_{i-1} = clé(e_{i-1})

c_i = clé(e_i)

recherche de l'élément de clé c dans l'arbre a

- **recherche de c dans a**

  parcours d'une liste ordonnée

  soit i tel que $c_{i-1} \leq c < c_i$

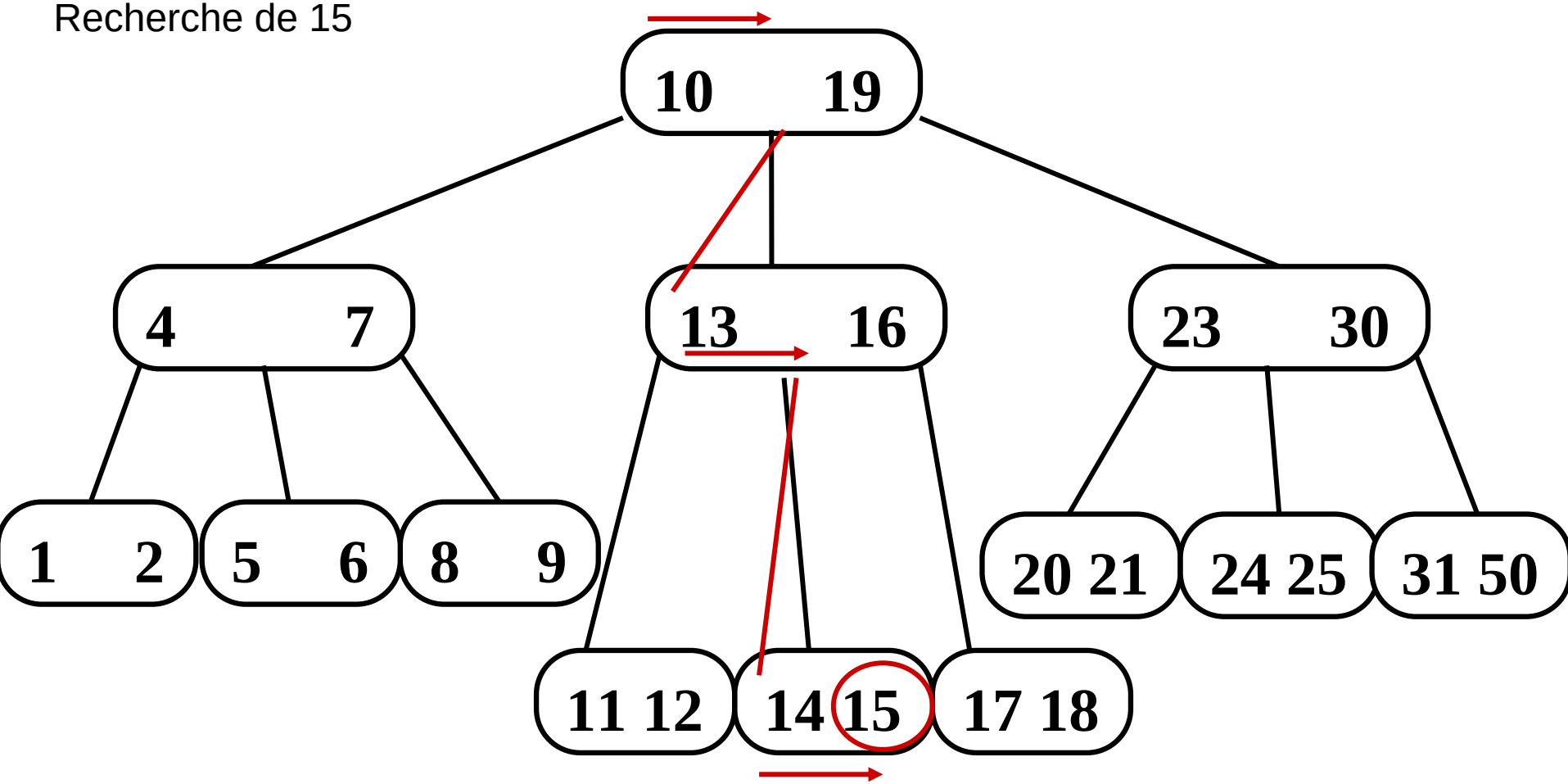
- **si c_{i-1} == c alors « place trouvée »**

  i-1ème élément de a

- **sinon recherche de c dans le ième sous-arbre de a**

  s'il est vide, clé absente

Recherche de 15

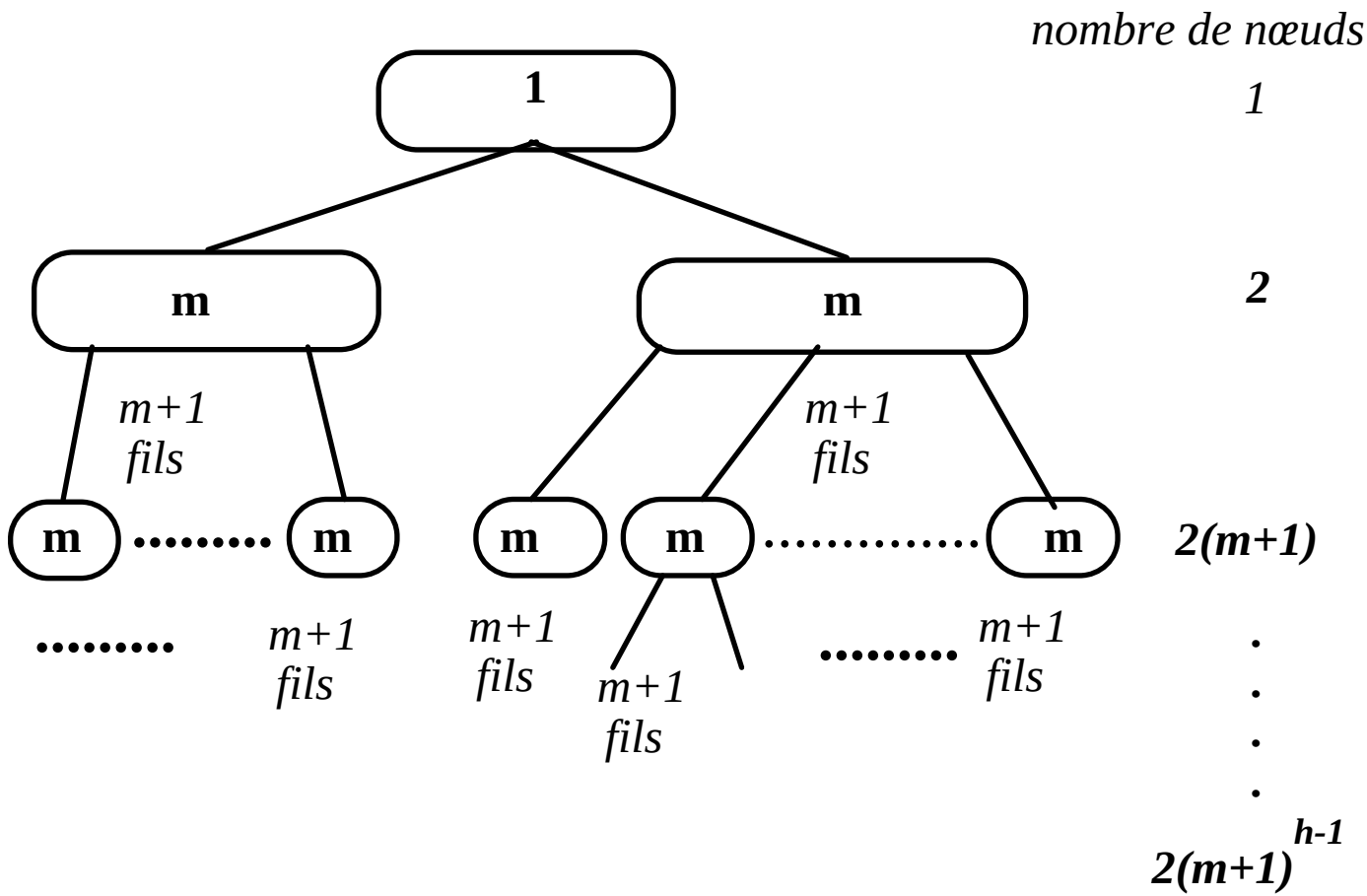


ARBRES BALANCES
ou B-ARBRES d'ordre m

- **arbre général de recherche**
- **toutes les feuilles sont au même niveau**
- **tout nœud contient au plus 2m valeurs**
- **tout nœud sauf la racine contient au moins m valeurs**

A B-arbre d'ordre m $\Rightarrow$

$$\log_{2m+1}(n+1) \leq h(A) \leq \log_{m+1}(n+1)/2$$



EXEMPLE

m = 100 et h = 2 $\Rightarrow$ 20 000 $\leq$ n $\leq$ 8 000 000

m = 100 et h = 3 $\Rightarrow$ 2 000 000 $\leq$ n $\leq$ 1 600 000 000

problème: maintenir le nombre de valeurs par nœud dans des limites après des adjonctions ou suppressions

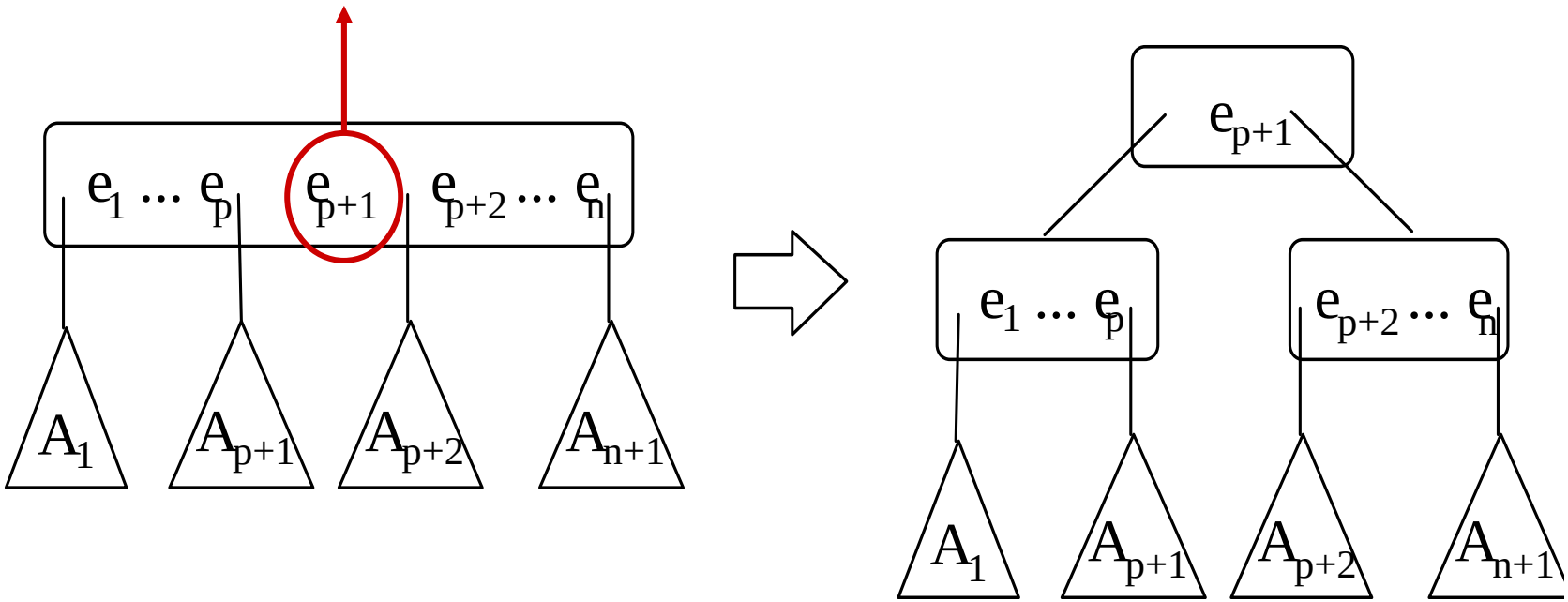
$\rightarrow$ opérations de rééquilibrage

- éclatement

- regroupements

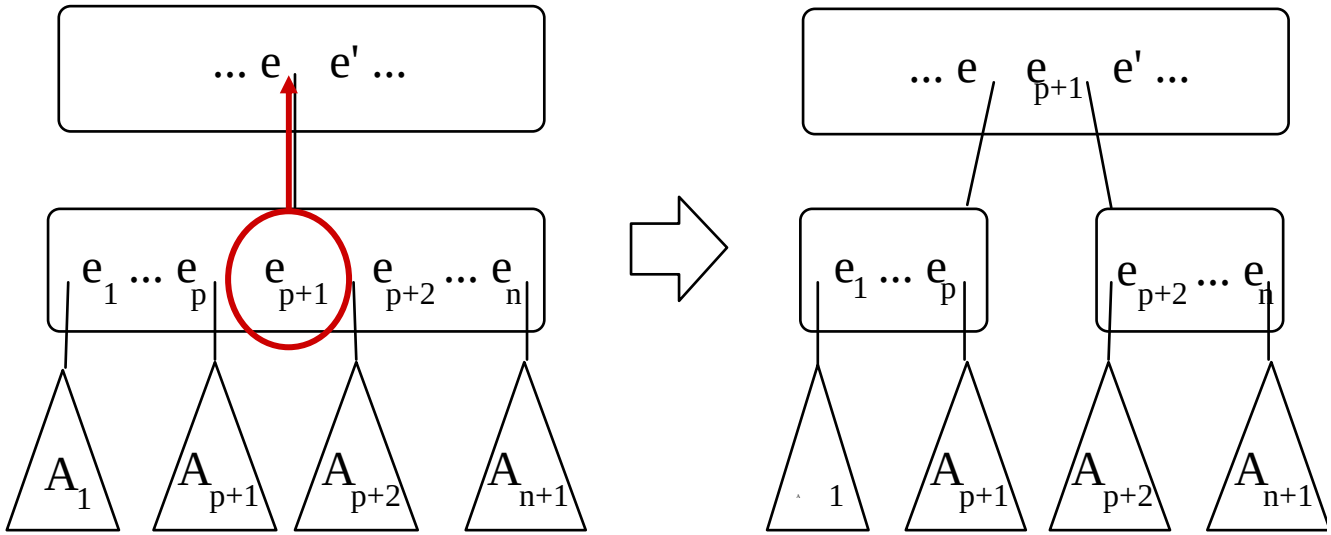
- répartition

éclatement d'une racine



précondition: nombre d'éléments ≥ 3

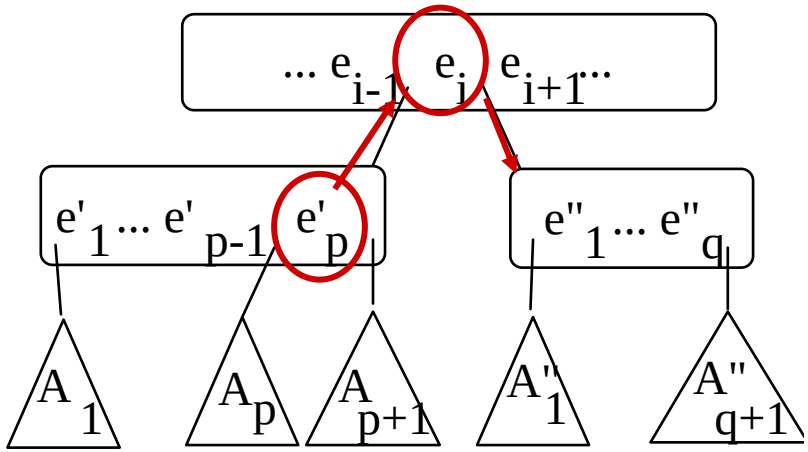
éclatement d'un sous-arbre



éclatement de la racine du sous-arbre

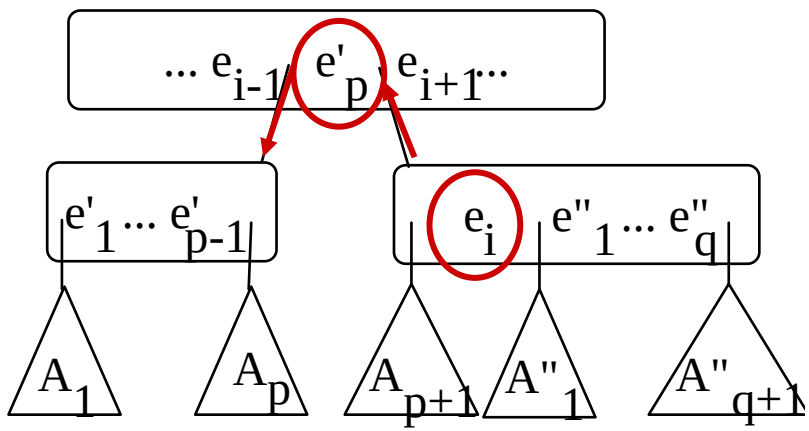
réinsertion de e_{p+1} dans le père de cette racine

transferts et répartition



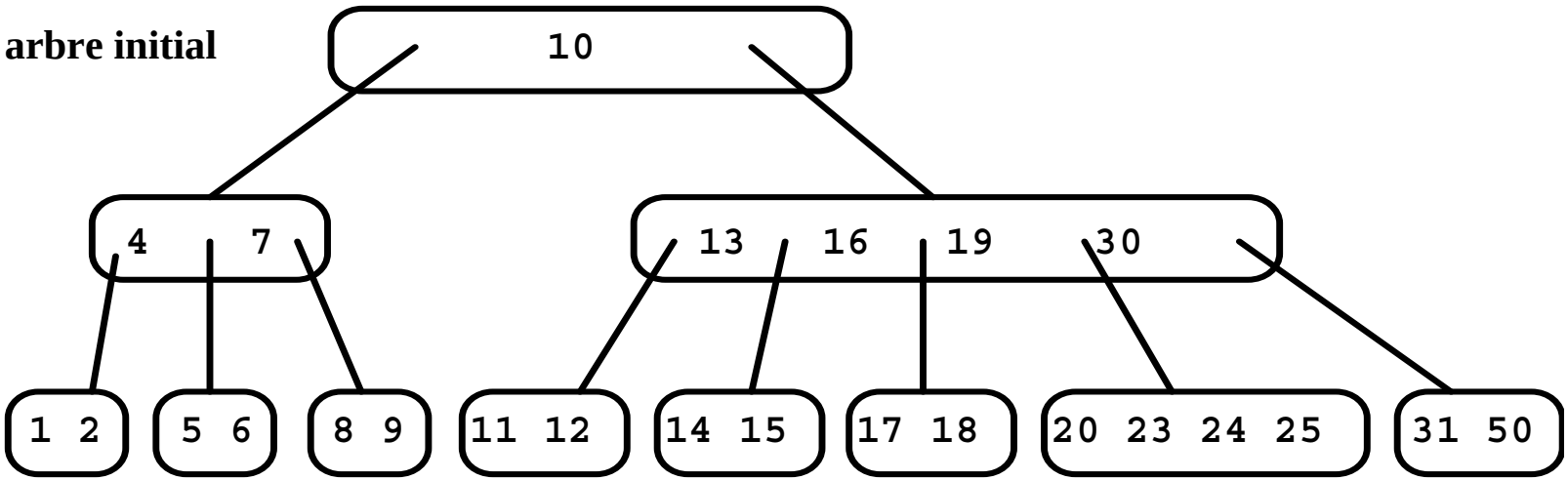
transfert gauche-droite et

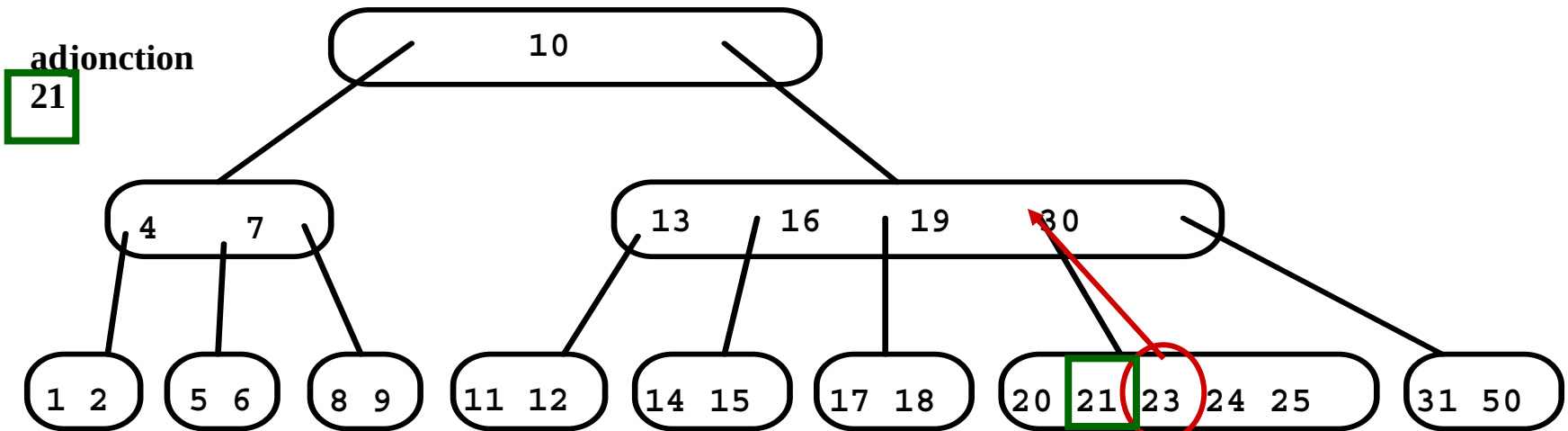
transfert droite-gauche



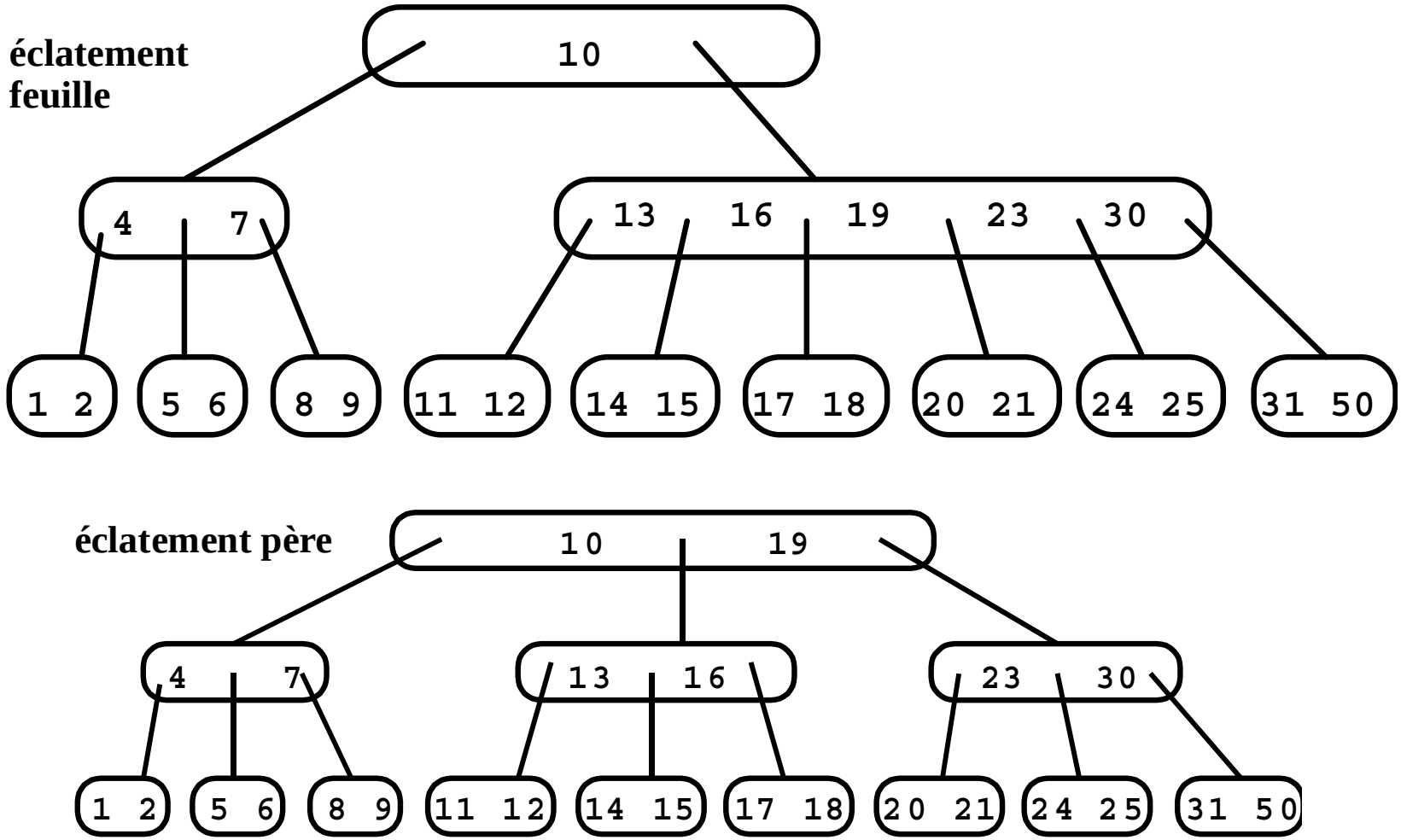
adjonction

(dans un 2-arbre)





adjonction (suite)



complexité: hauteur de l'arbre

suppression

suppression dans un nœud non feuille $\Rightarrow$

remplacer la valeur par la plus petite du $(i+1)^{ème}$ sous-arbre

suppression effective dans une feuille

rééquilibrage éventuel par regroupement

conclusion

maintien de la propriété B-arbre:

adjonction $\Rightarrow$ éclatement ou répartition

suppression $\Rightarrow$ regroupement ou répartition

opérations **au plus $\log_{m+1}(n+1)/2$ accès disque**

taux de remplissage moyen: 0.7

amélioration

B-arbre sur les clés, pour augmenter m

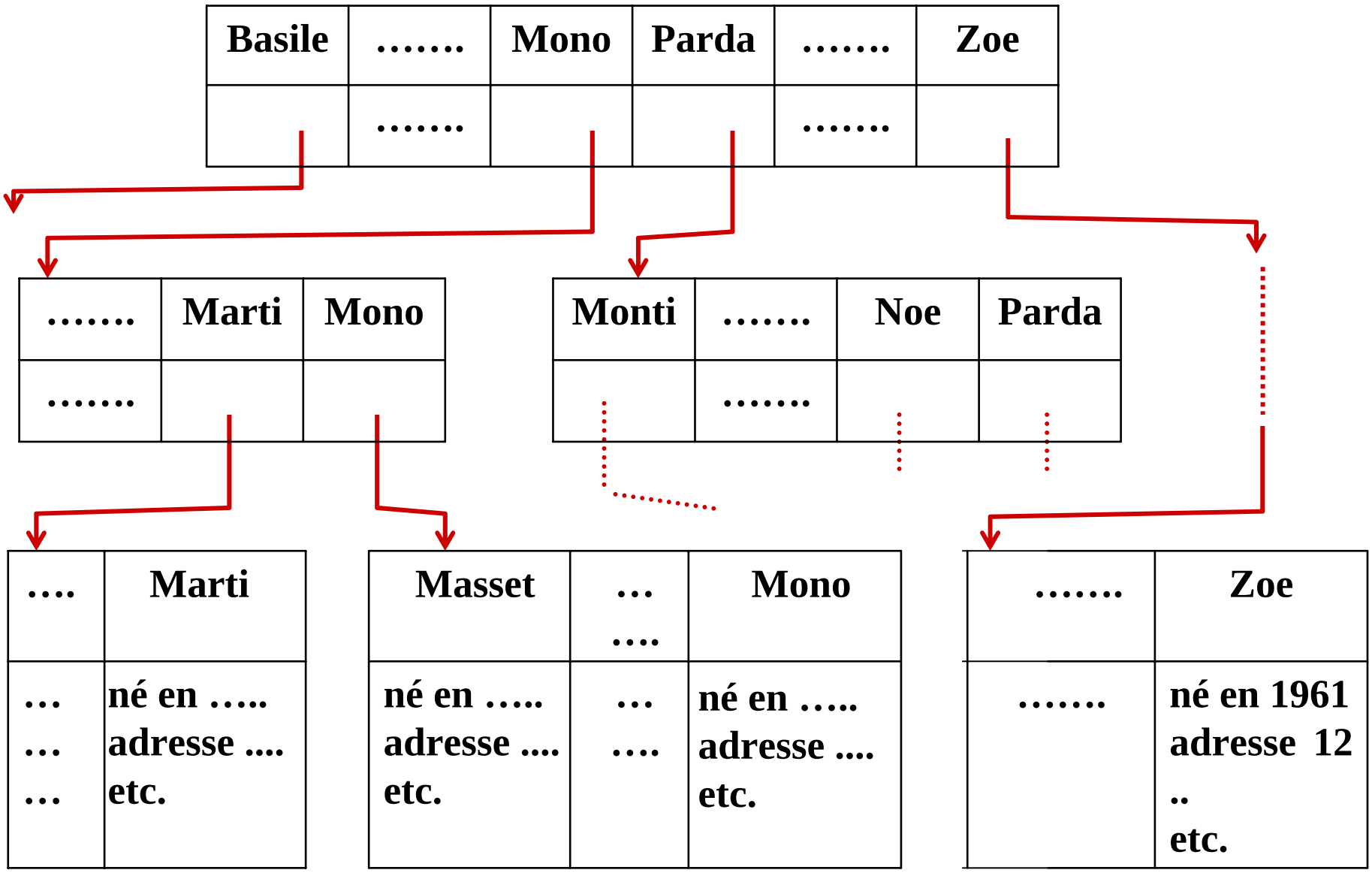
8.3 FICHIERS SEQUENTIELS INDEXES

idée : amélioration des B-arbres: ne stocker que des clés dans les nœuds intermédiaires pour augmenter m. Les informations sont stockées uniquement dans les feuilles.

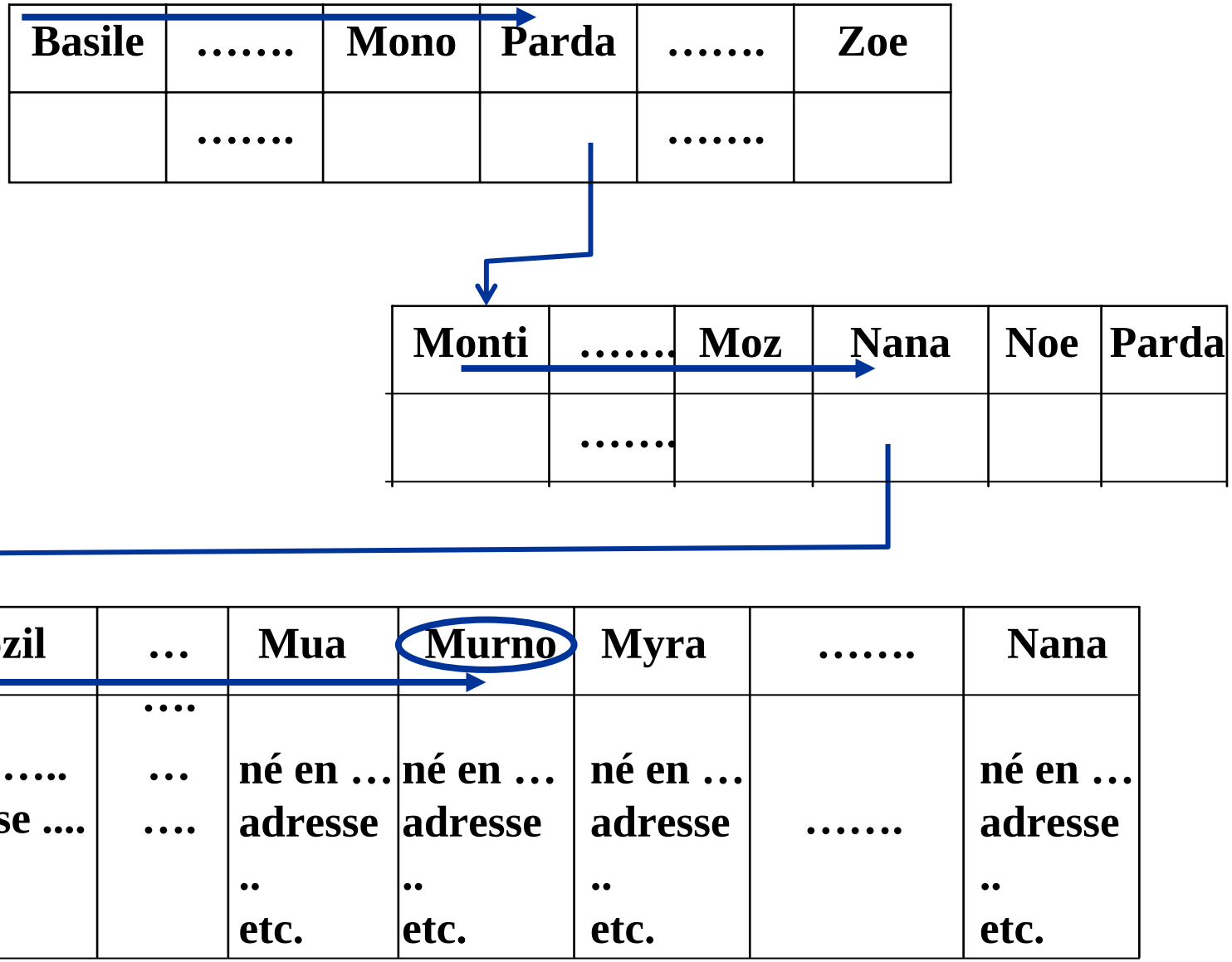
un nœud_intermediaire est une liste ordonnée de clés

une feuille contient les clés et les éléments associés

Un exemple de fichier séquentiel indexé



Recherche de Murno

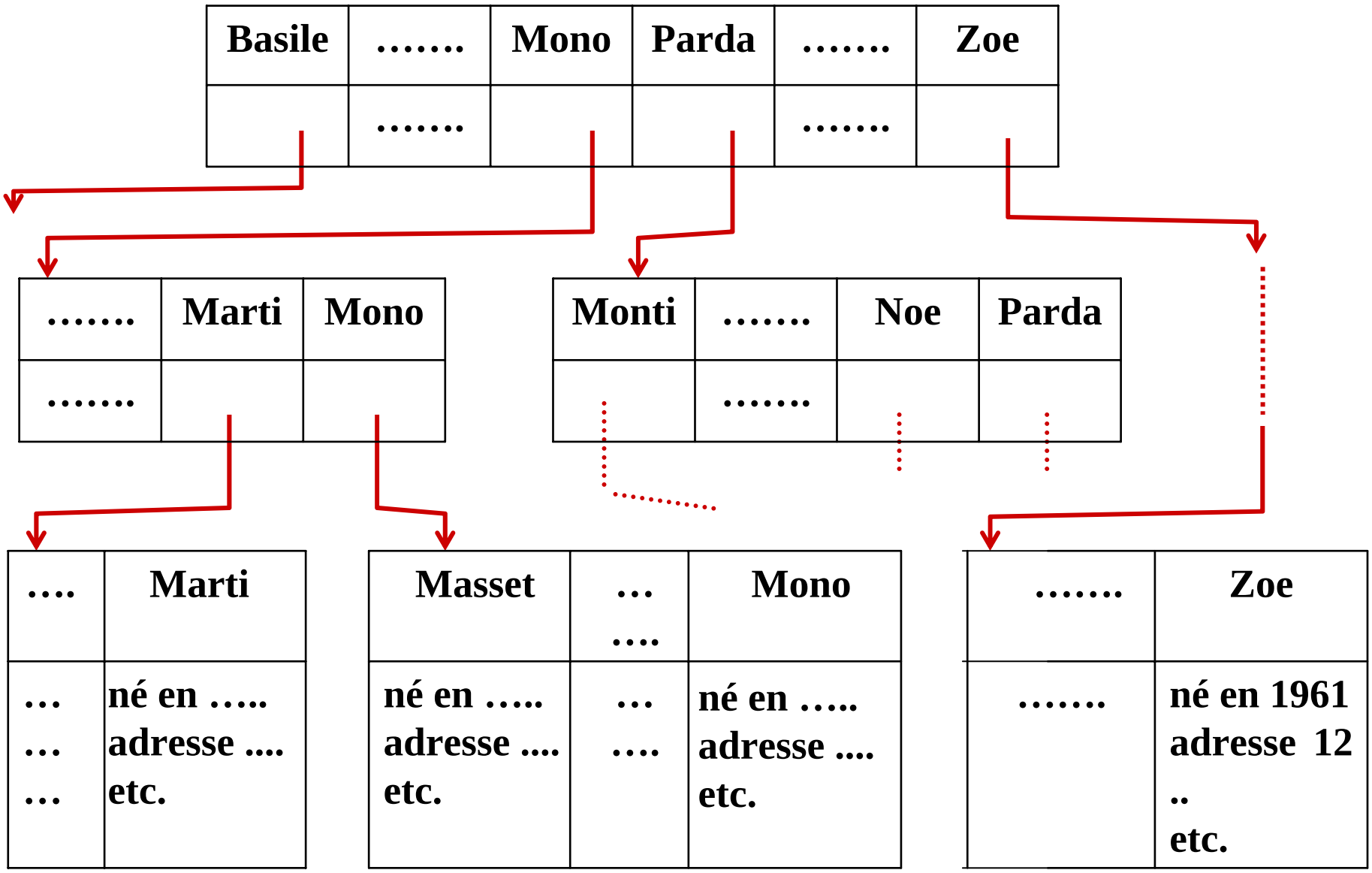


La clé de rang i d'un nœud

→ est la plus grande clé de tous les éléments mémorisés dans les feuilles du i-ème sous-arbre de ce nœud.

→ est la clé du dernier élément de la feuille la plus à droite du i-ème sous-arbre de ce nœud.

Un exemple de fichier séquentiel indexé



si

représentation d'une clé

$\cong$

10% de représentation d'un élément

alors

m $\leftarrow$ 10m

La hauteur de l’arbre a décroît