Arbres binaires de recherche (II)

- Version avec équilibrage (AVL) -

Irena.Rusu@univ-nantes.fr

LINA, bureau 123, 02.51.12.58.16

Temps d'exécution

opérations sur ensemble

implémentation	Ens_vide	Ajouter Enlever	Élément	Min
table	cst	$O(1)$*	$O(n)$	$O(n)$
table triée	cst	$O(n)$	$O(\log n)$	$O(1)$
liste chaînée	cst	$O(1)$*	$O(n)$	$O(n)$
arbre équilibré	cst	$O(\log n)$	$O(\log n)$	$O(\log n)$
arbre	cst	$O(\log n)$	$O(\log n)$	$O(\log n)$
table de hachage	$O(B)$	cst	cst	$O(B)$

en moyenne

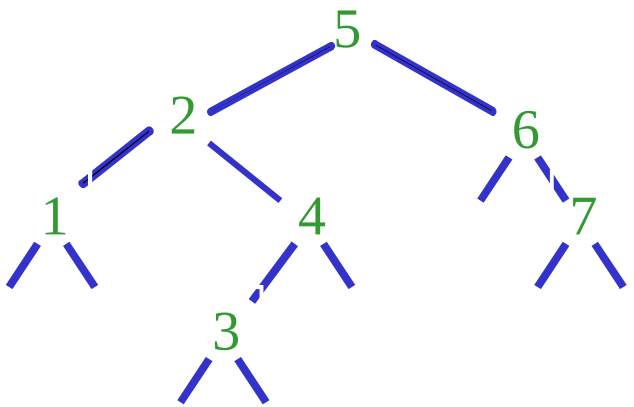
n nombre d 'éléments

$B > n$ taille de la table de hachage

**sans le test d 'appartenance*

Objectifs

- Recherche, insertion, suppression en O(h(A)) chacune, MAIS
- … Maintenir h(A) à une valeur en O(log n), où n est le nombre d'éléments dans l'arbre
- Eviter l' « allongement » de l'arbre ; faire basculer des sous-arbres entiers de gauche vers la droite ou inversement…
- … et cela après chaque opération d'insertion ou suppression



→ Ce rééquilibrage doit aussi être fait en O(log n) au pire pour assurer l'efficacité

Sommaire

- Arbres binaires équilibrés en hauteur (ou AVL)
  - Equilibrage
  - Ajout d’un élément
  - Suppression d’un élément

Sommaire

- Arbres binaires équilibrés en hauteur (ou AVL)
  - Equilibrage
  - Ajout d’un élément
  - Suppression d’un élément

AVLs - Historique

- Auteurs : Georgy Adelson-Velsky (1922 -)

et Evgueny Landis

(1921-1997)

- Publication en 1962 :

  An algorithm for the organization of information, *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

- Ce sont historiquement les premiers arbres de recherche équilibrés algorithmiquement

Arbres binaires de recherche équilibrés (AVLs)

A **arbre AVL ssi A est un ABR et en plus (si A non-vide)**

Soit 1*/* en tout noeud *p* de *A* :

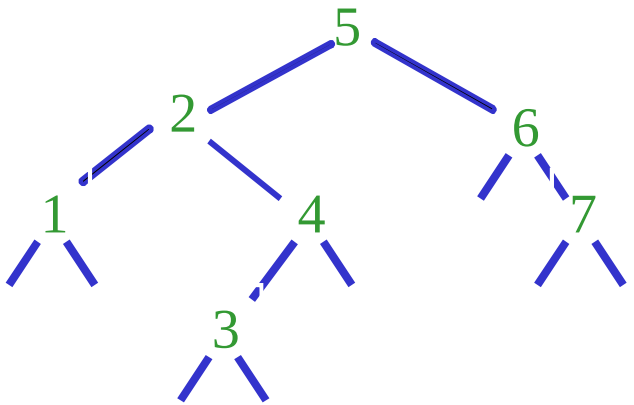
$|h(D(p)) - h(G(p))| \leq 1$

Soit 2*/* *A* est de la forme *A=(r, G, D)* avec:

G, D* sont des *AVLs

$max(G) \leq r < min(D)$

et $|h(D) - h(G)| \leq 1$



bal(*p*) = bal(*A*(*p*)) = h(*D*(*p*)) - h(*G*(*p*)) balance (ou déséquilibre)

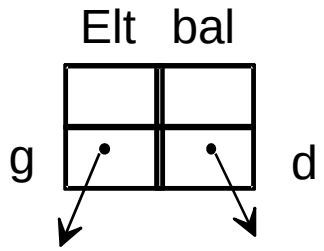
1'*/* en tout nœud *p* de *A* (si *A* non vide),

bal(*p*) = -1, 0 ou + 1

Implémentation des AVLs

avl : structure, comme les ABR, sauf bal :

- **compris entre –1 et +1 si AVL**
- **-2 et +2 temporairement dans la suite**



Fonctions :

(avl,entier) AJOUTER (element x, avl A) ;
/* rend l'arbre modifié et la différence de hauteur : 0 ou +1 */

(avl,entier) ENLEVER (element x, avl A) ;
/* rend l'arbre modifié et la différence de hauteur : -1 ou 0 */

(avl,entier) OTERMIN (avl A) ;
/* rend l'arbre modifié et la différence de hauteur : -1 ou 0 */

Hauteur d'AVL

A **arbre AVL à *n* nœuds**

$$\log_2(n+1)-1 \leq h(A) \leq 1{,}45 \log_2(n+2)$$

$\Rightarrow$ **Implémentation d'ensembles avec opérations :**

MIN (*A*), MAX (*A*)

AJOUTER (*x*, *A*)

ENLEVER (*x*, *A*)

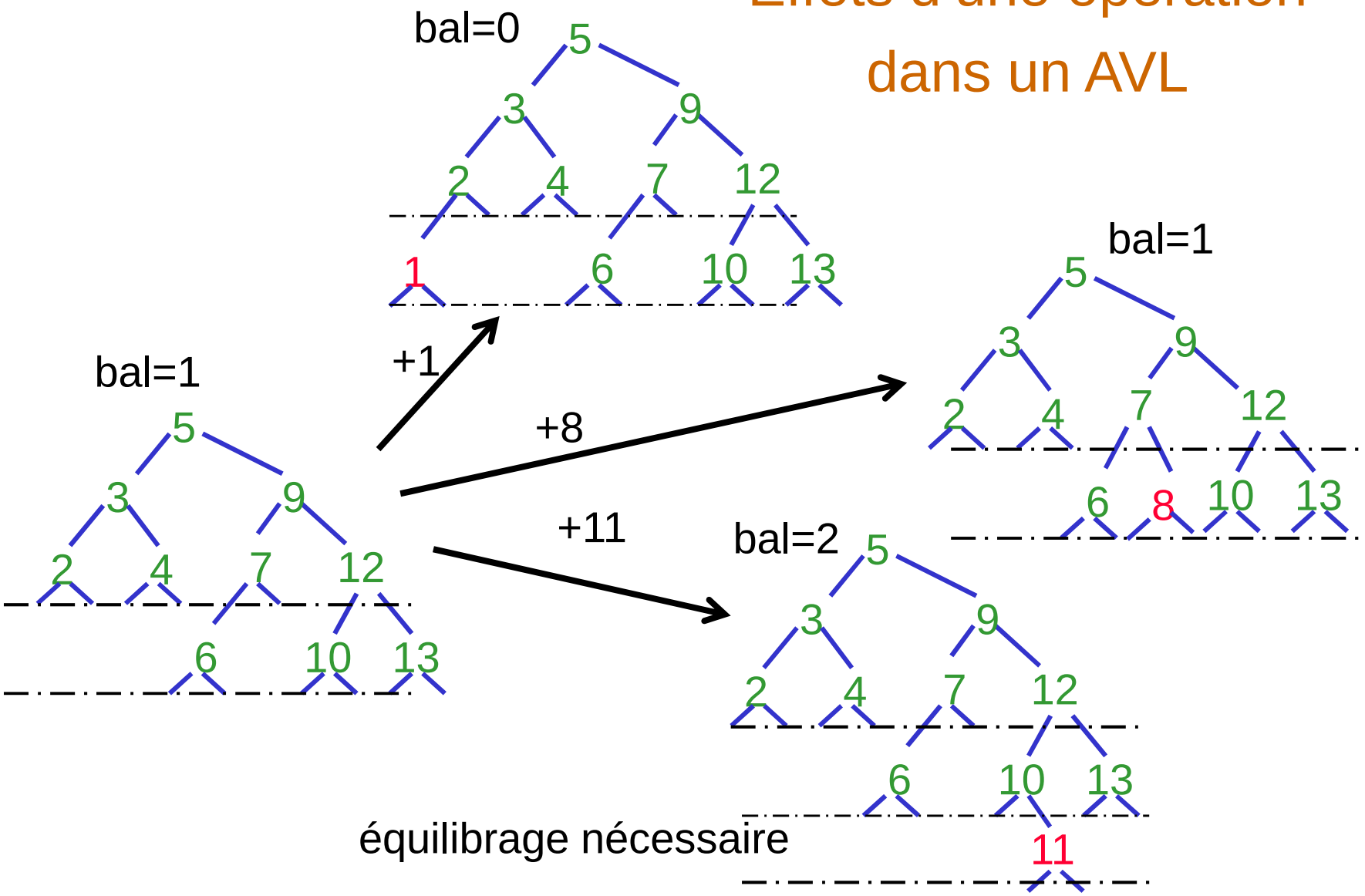
ELEMENT (*x*, *A*)

} **O(log(*n*))**
pire des cas

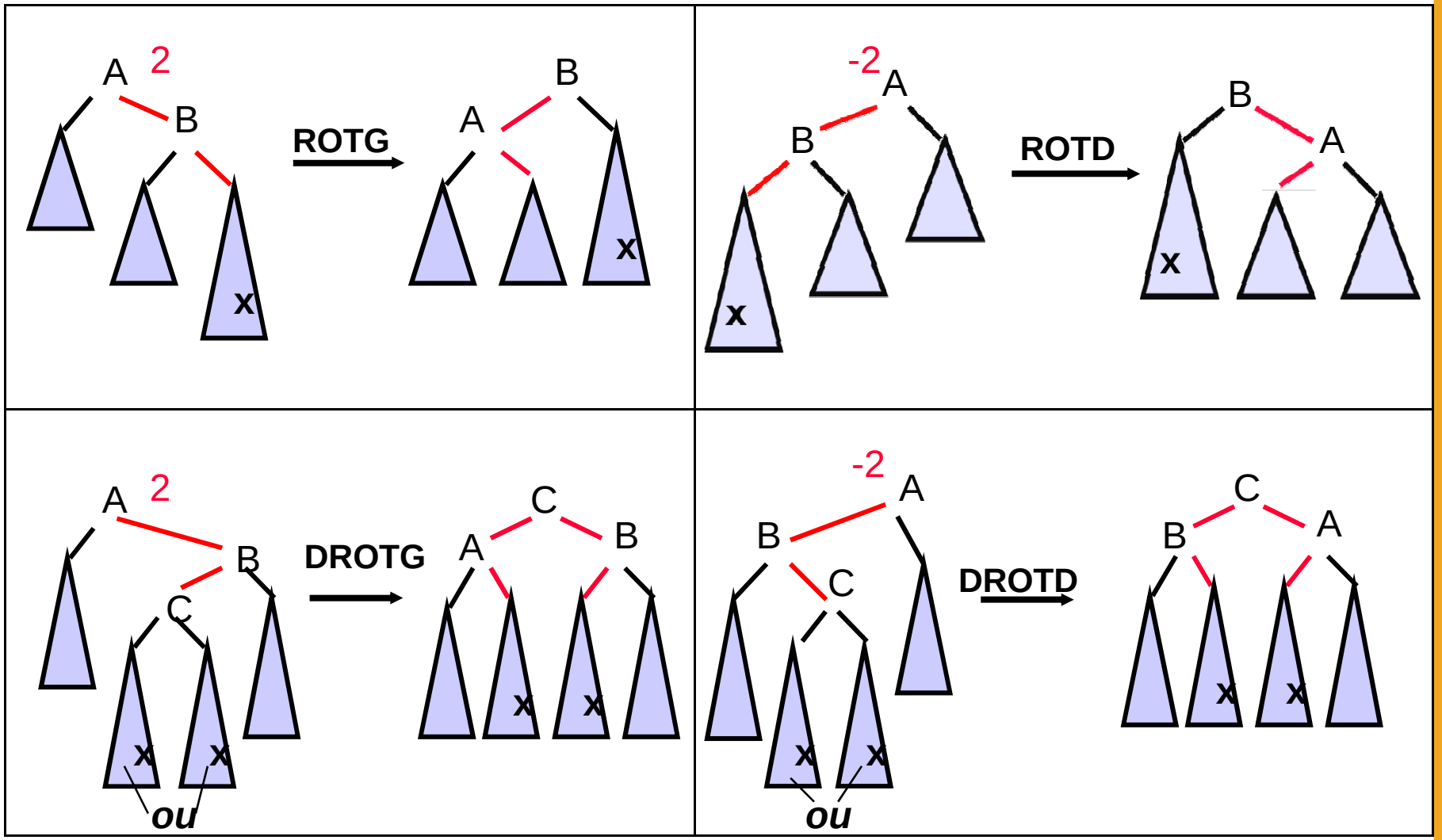
Sommaire

- Arbres binaires équilibrés en hauteur (ou AVL)
  - Equilibrage
  - Ajout d’un élément
  - Suppression d’un élément

Effets d'une opération dans un AVL



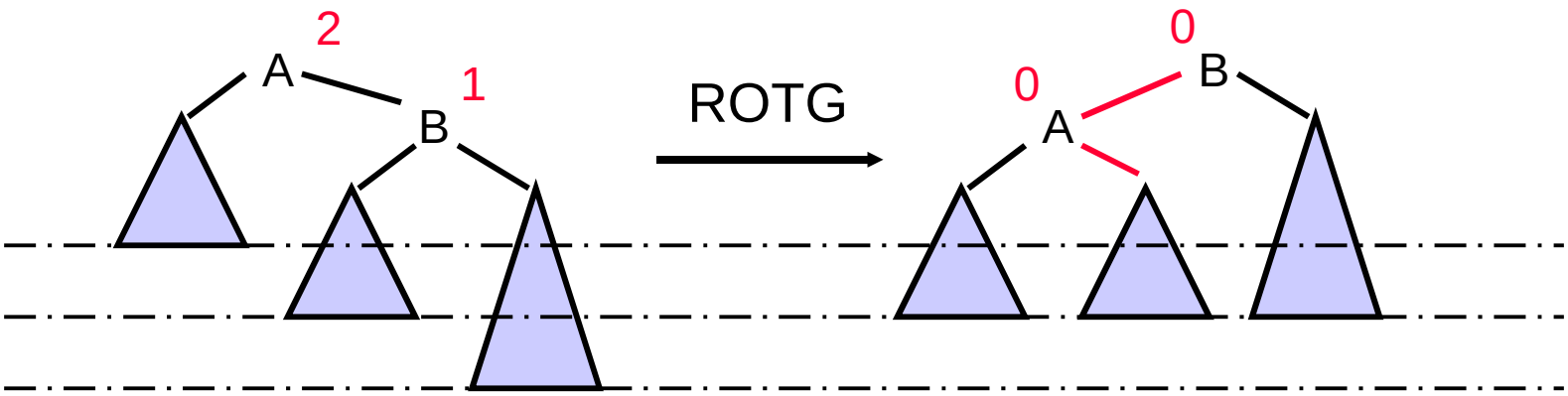
Equilibrage : 4 solutions (à choisir convenablement)



x indique la source du déséquilibre.
Note : A est le nœud le plus bas de déséquilibre +2 ou -2

Rotation gauche : cas 1

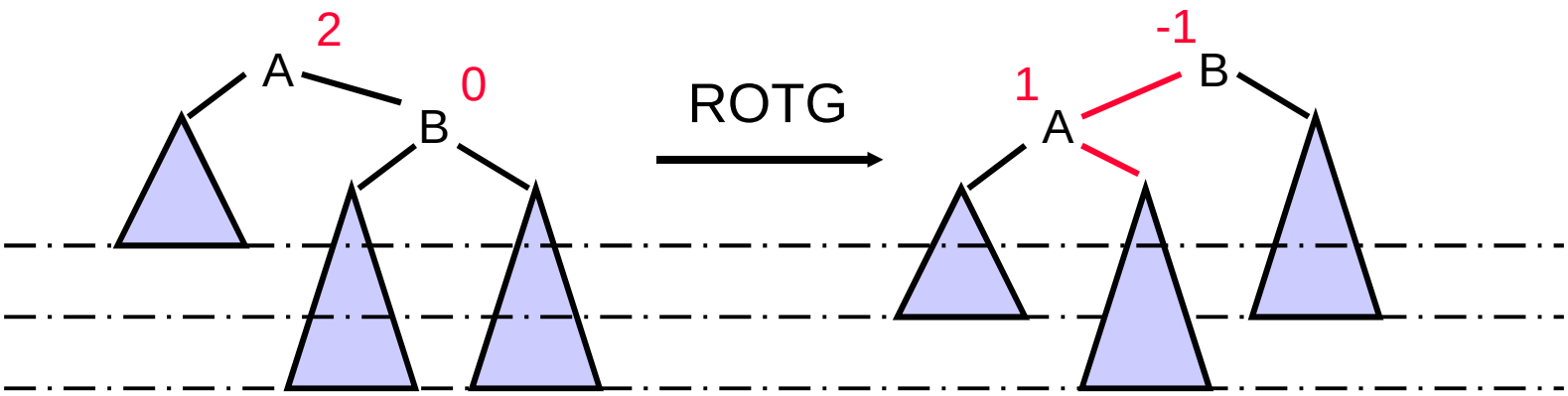
$A = (r, A_g, A_d)$ où A_g, A_d sont AVL
- bal(A) = 2
- bal(A_d) = 1



La rotation préserve l'ordre symétrique
Note : elle diminue la hauteur

Rotation gauche : cas 2

$A = (r, A_g, A_d)$ où A_g, A_d sont AVL
- bal(A) = 2
- bal(A_d) = 0



La rotation préserve l'ordre symétrique
Note : elle ne diminue pas la hauteur

Rotation gauche (suite)

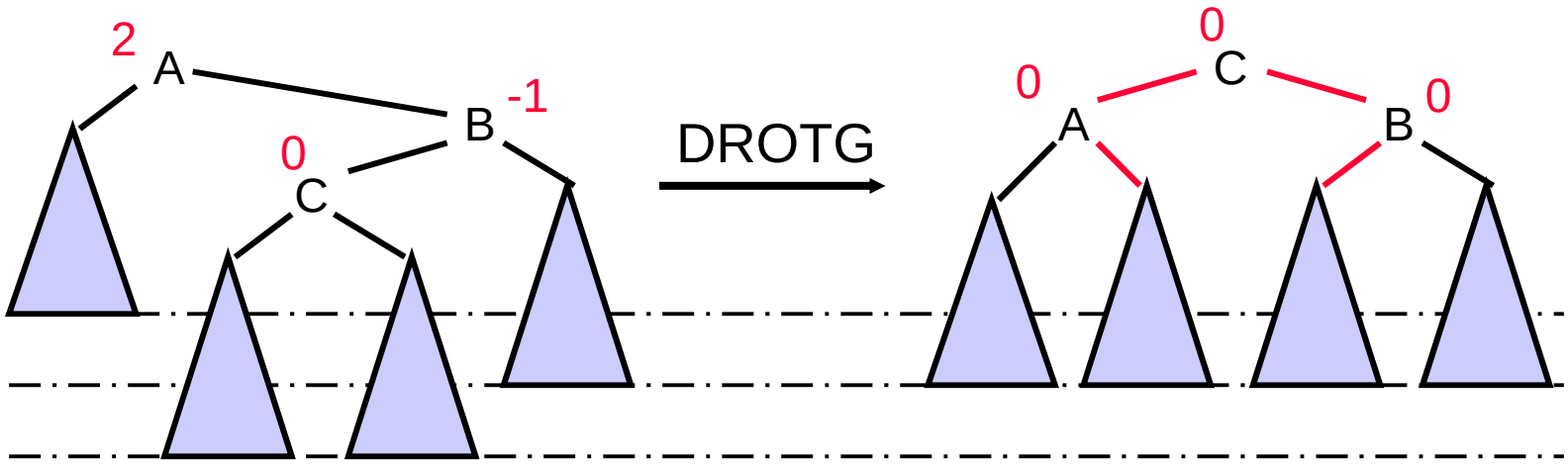
```
avl ROTG(avl A) {
        avl B; int a,b;

        B ← A.d;
        a ← A.bal;  b ← B.bal;
        A.d ← B.g; B.g ← A;  /* rotation */
        A.bal ← a-max(b,0)-1;
        B.bal ← min(a-2,a+b-2,b-1);
        return B;
}
```

Note. Algorithme valable pour tout arbre binaire pour lequel les balances sont calculées (quelles que soient ces balances)

Fonction ROTD similaire.

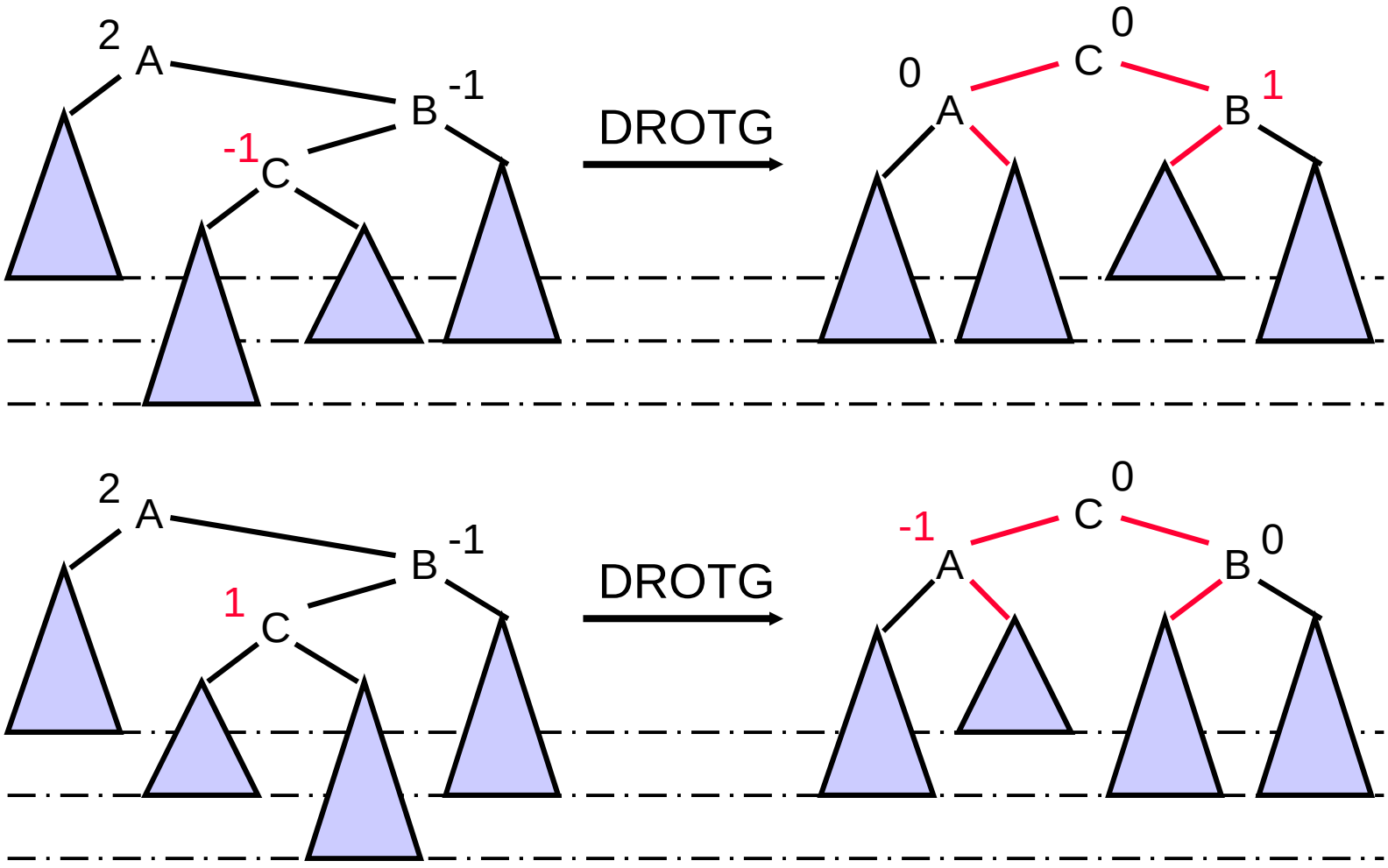
Double rotation gauche : cas 1

$A = (r, A_g, A_d)$ où A_g, A_d sont AVL
- bal(A) = 2
- bal (A_d) = -1

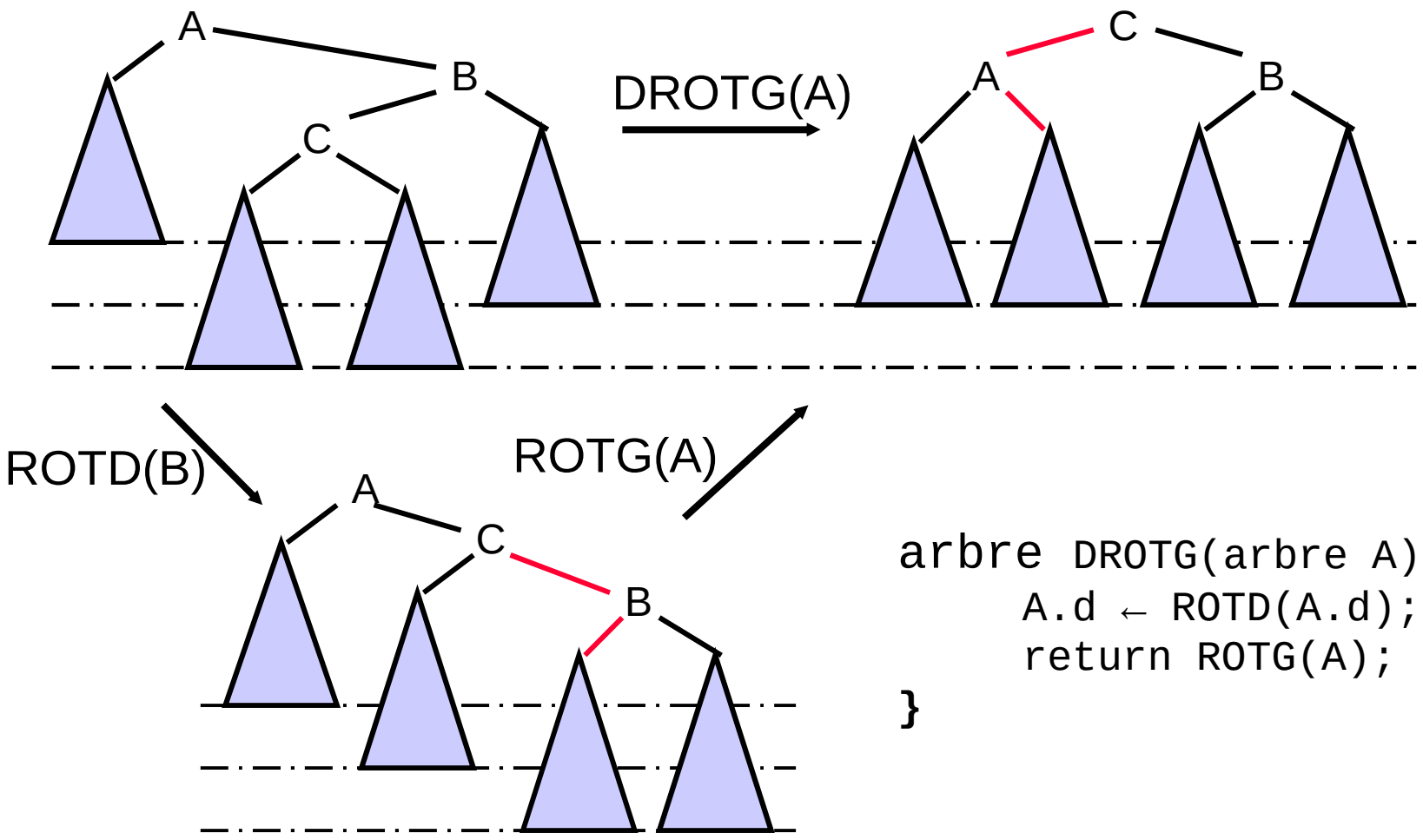


La rotation préserve l'ordre symétrique
Note : elle diminue la hauteur

Double rotation gauche : cas 2, 3



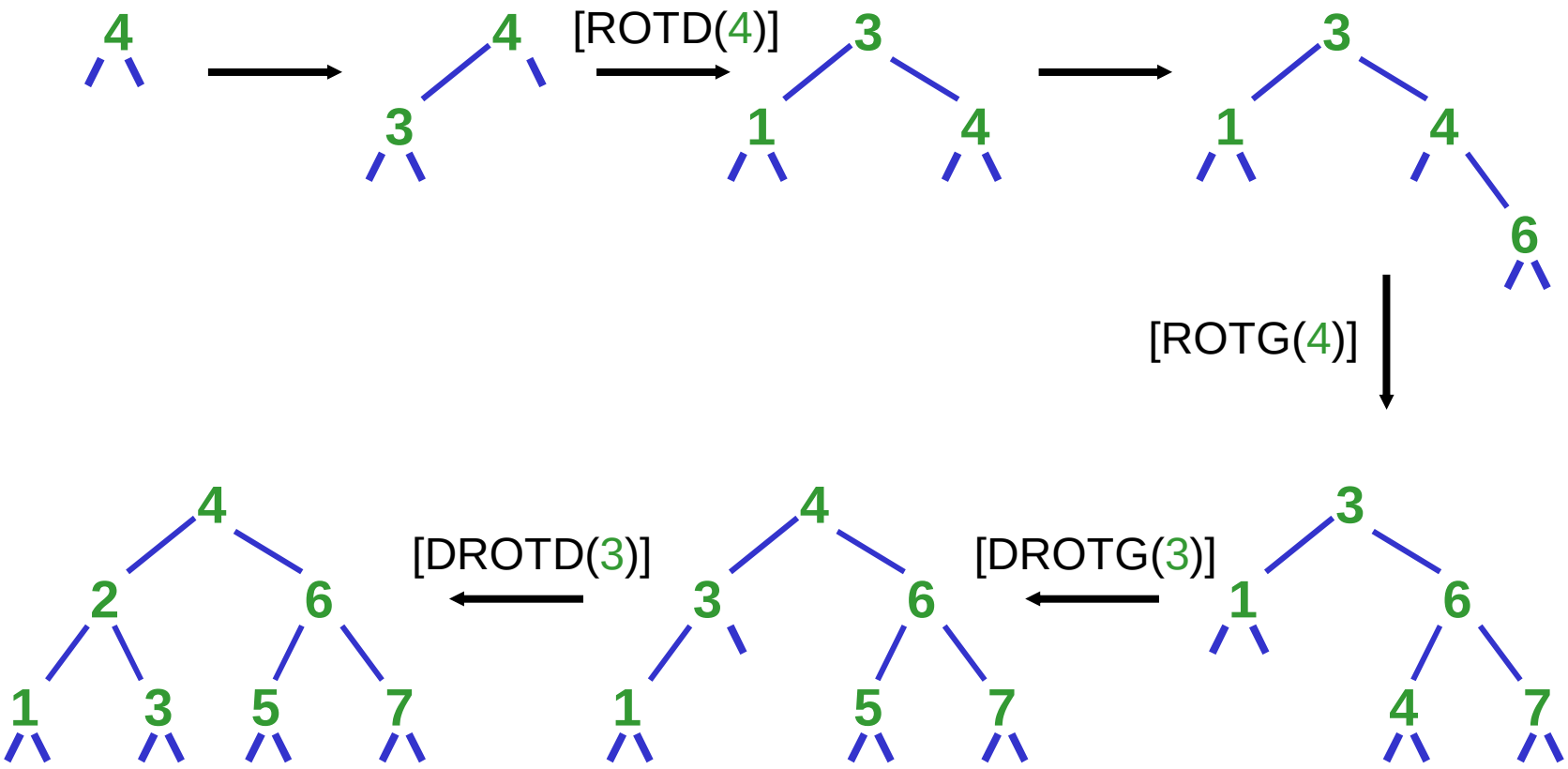
Double rotation gauche (suite)




```
arbre DROTG(arbre A) {
      A.d ← ROTD(A.d);
      return ROTG(A);
}
```

Exemple

Ajouts successifs de **4**, **3**, **1**, **6**, **7**, **5**, **2** dans l'arbre vide



Note. Dans les AVLs, on n'ajoute jamais un élément déjà présent.

Equilibrage

Entrée
A arbre tel que
A_g, A_d sont AVL
$-2 \leq bal(A) \leq 2$

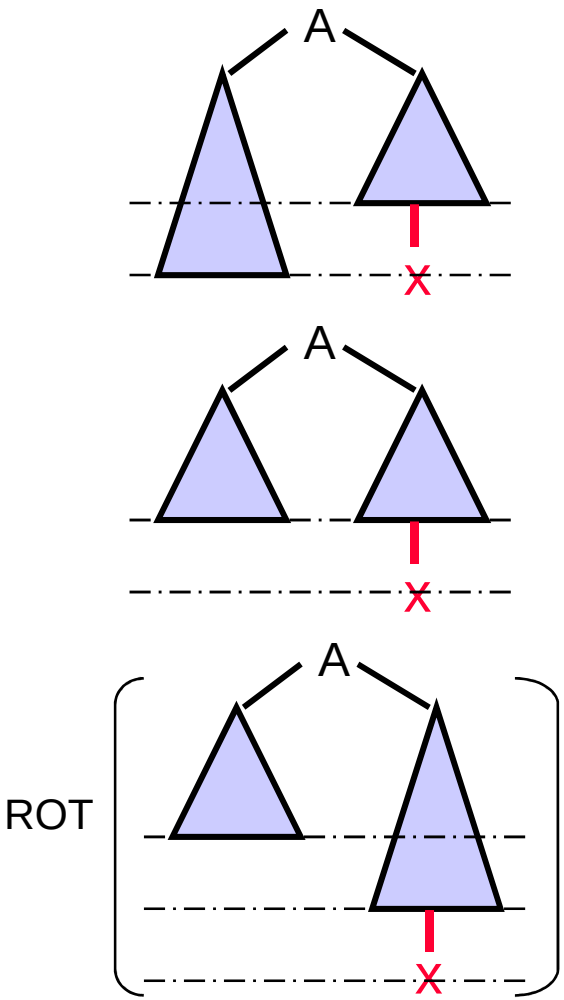
Sortie
A arbre AVL
$[-1 \leq bal(A) \leq 1]$

```
avl EQUILIBRER(avl A) {
     si (A.bal = 2) alors
          si (A.d.bal >= 0) alors
               retour ROTG(A);
          sinon { A.d ← ROTD(A.d);
               retour ROTG(A);
            }
     sinon si (A.bal = -2) alors
          si (A.g.bal <= 0) alors
               retour ROTD(A);
          sinon { A.g ← ROTG(A.g);
               retour ROTD(A);
          }
     sinon retourne A;
}
```

Sommaire

- Arbres binaires équilibrés en hauteur (ou AVL)
  - Equilibrage
  - Ajout d’un élément
  - Suppression d’un élément

Variation de la hauteur après ajout



Équilibre (bal) avant	après	variation de hauteur
-1	0	0
0	1	+1
1	0	0

corrélation !

Conclusion :
variation = 0 ssi bal = 0 après ajout

Ajout d'un élément

```
(avl,int) AJOUTER(element x, avl A)
 /* retourne le nouvel arbre et la variation de la hauteur */

     si (A = NULL) alors {
          créer un nœud A;  A.g ← NULL;  A.d ← NULL;
          A.elt ← x;  A.bal ← 0;
          retour (A,1);}
     sinon si (x = A.elt) alors
             retour (A,0);
           sinon si (x > A.elt) alors
                    (A.d,h) ← AJOUTER(x,A.d);
                 sinon
                   {(A.g,h) ← AJOUTER(x,A.g); h ← -h;}
                 si (h = 0) alors
                    retour (A,0);
                 sinon
                    A.bal ← A.bal + h;
                    A ← EQUILIBRER(A);
                    si (A.bal = 0) alors
                    retour (A,0);
                    sinon retour (A,1);
```

Temps pour un ajout avec équilibrage

Temps d'une rotation : constant

Note : AJOUTER exécute au plus une rotation
car une rotation dans cette situation rétablit la hauteur initiale de l'arbre auquel elle est appliquée

A arbre AVL à *n* nœuds

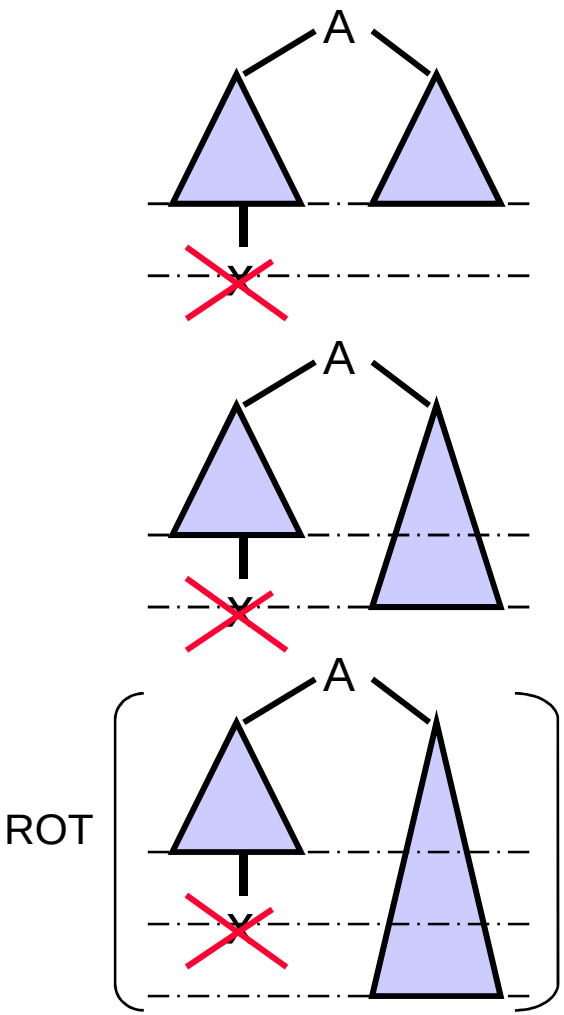
Temps total d'un ajout : $O(h(A)) = O(\log n)$

car une seule branche de l'arbre est examinée

Sommaire

- Arbres binaires équilibrés en hauteur (ou AVL)
  - Equilibrage
  - Ajout d’un élément
  - Suppression d’un élément

Variation de la hauteur après une suppression



Équilibre (bal) avant	après	variation de hauteur
-1	0	-1
0	1	0
1	0	-1
	ou -1	0

corrélation !

Conclusion :

variation = -1 ssi bal = 0 après suppression

Suppression d'un élément

```
(avl,int) ENLEVER(element x, avl A)
\* retourne le nouvel arbre et la variation de la hauteur*\

     si (A = NULL) alors retour (A,0);
     sinon si (x > A.elt) alors
            (A.d,h) ← ENLEVER(x,A.d);
           sinon si (x < A.elt) alors
                  {(A.g,h) ← ENLEVER(x,A.g); h ← -h;}
                 sinon si (A.g = NULL) alors
                         retour (A.d,-1);
                        sinon si (A.d = NULL) alors
                                retour (A.g,-1);
                               sinon {A.elt ← min(A.d);
                                      (A.d,h) ← OTERMIN(A.d);}
          si (h = 0) alors retour (A,0);
            sinon {A.bal ← A.bal + h;
                   A ← EQUILIBRER(A);
                   si (A.bal = 0) alors retour (A,-1);
                  sinon retour (A,0);}
```

Fonction OTERMIN

Entrée
A arbre AVL
non vide

```
(avl,int) OTERMIN(avl A) {
     si (A.g = NULL) alors {
          min ← A.elt;
          retour (A.d,-1);}
     sinon
          {(A.g,h) ← OTERMIN(A.g); h ← -h;}
     si (h = 0) alors
          retour(A,0);
     sinon {
          A.bal ← A.bal + h;
          A ← EQUILIBRER(A);
          si (A.bal = 0) alors
               retour (A,-1);
          sinon
               retour (A,0);
     }
}
```

Temps pour une suppression

Note : ENLEVER et OTERMIN peuvent exécuter une rotation sur chaque ancêtre du nœud supprimé

A arbre AVL à *n* nœuds

Temps total d'une suppression : $O(h(A)) = O(\log n)$

car ENLEVER et OTERMIN examinent une seule branche de l'arbre (et temps d'une rotation constant)