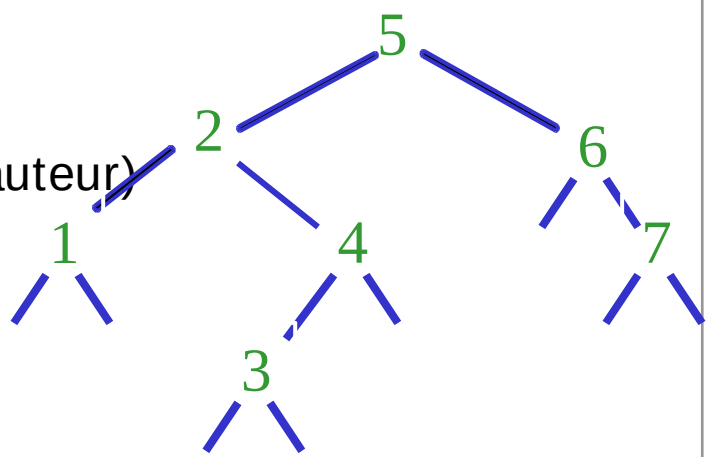
Arbres de recherche AVL

Implémentation d‘ ensembles avec opérations :

MIN (A), MAX (A)
AJOUTER (x, A)
ENLEVER (x, A)
ELEMENT (x, A)

} O(log(|A|)) pire des cas



A arbre (binaire de recherche) AVL (équilibré en hauteur)

1/ en tout noeud p de A (si A non vide) :

$$|h(A_d(p)) - h(A_g(p))| \leq 1$$

2/ A vide ou

$A = (r, A_g, A_d)$, A_g, A_d sont des arbres AVL

et $|h(A_d) - h(A_g)| \leq 1$

$bal(p) = bal(A(p)) = h(A_d(p)) - h(A_g(p))$

Implémentation des AVL

```
struct NOEUD {
     element elt ;
     deuxbits bal ; /* compris entre –1 et +1 */
                    /* -2 et +2 dans la suite */
     struct NOEUD * g, * d ;
} noeud ;
noeud * arbre ;

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

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

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

Hauteur des arbres AVL

A arbre AVL à *n* nœuds

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

ARBRES de FIBONACCI
[nœuds identifiés à leurs étiquettes, 1, 2, …]

$\mathbf{F}_0$ = () = arbre vide — $h(\mathbf{F}_0) = -1$

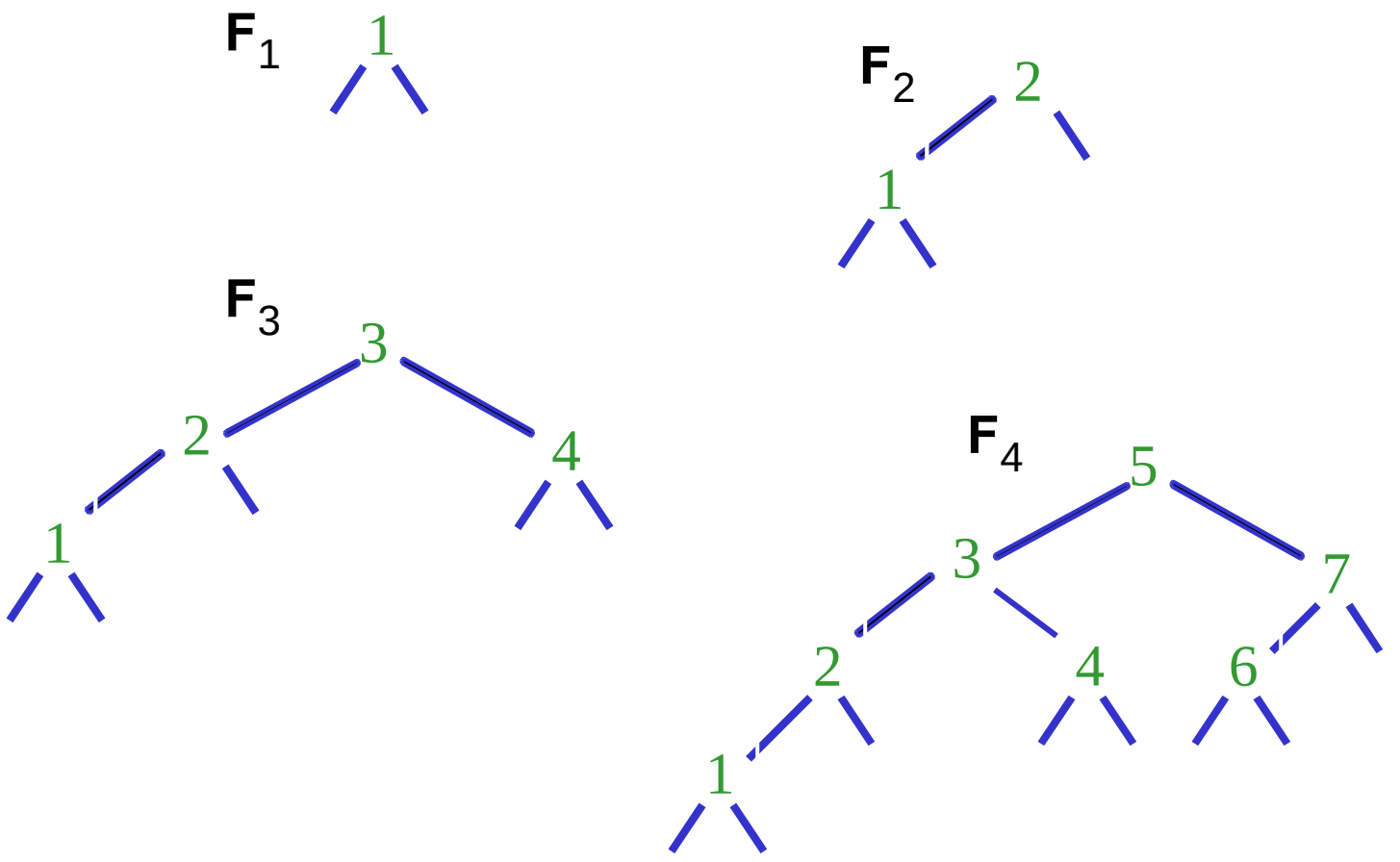
$\mathbf{F}_1 = (1, (), ())$ — $h(\mathbf{F}_1) = 0$

$\mathbf{F}_k = (F_{k+1}, \mathbf{F}_{k-1}, \mathbf{F'}_{k-2})$ si $k>1$ — $h(\mathbf{F}_k) = k-1$

$\mathbf{F'}_{k-2} = \mathbf{F}_{k-2}$ avec étiquettes augmentées de F_{k+1}

[nombres de Fibonacci : $F_0 = 0$, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, …]

AVL de Fibonacci



[$F_0 = 0$, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, $F_6 = 8$, $F_7 = 13$, …]

Propriétés

L'arbre de Fibonacci $\mathbf{F}_k$

- est un arbre AVL
- est étiqueté par 1, 2, ..., F_{k+2}-1
- a pour hauteur k-1

Tout arbre AVL de hauteur k-1 possède au moins

- F_{k+2} -1 nœuds

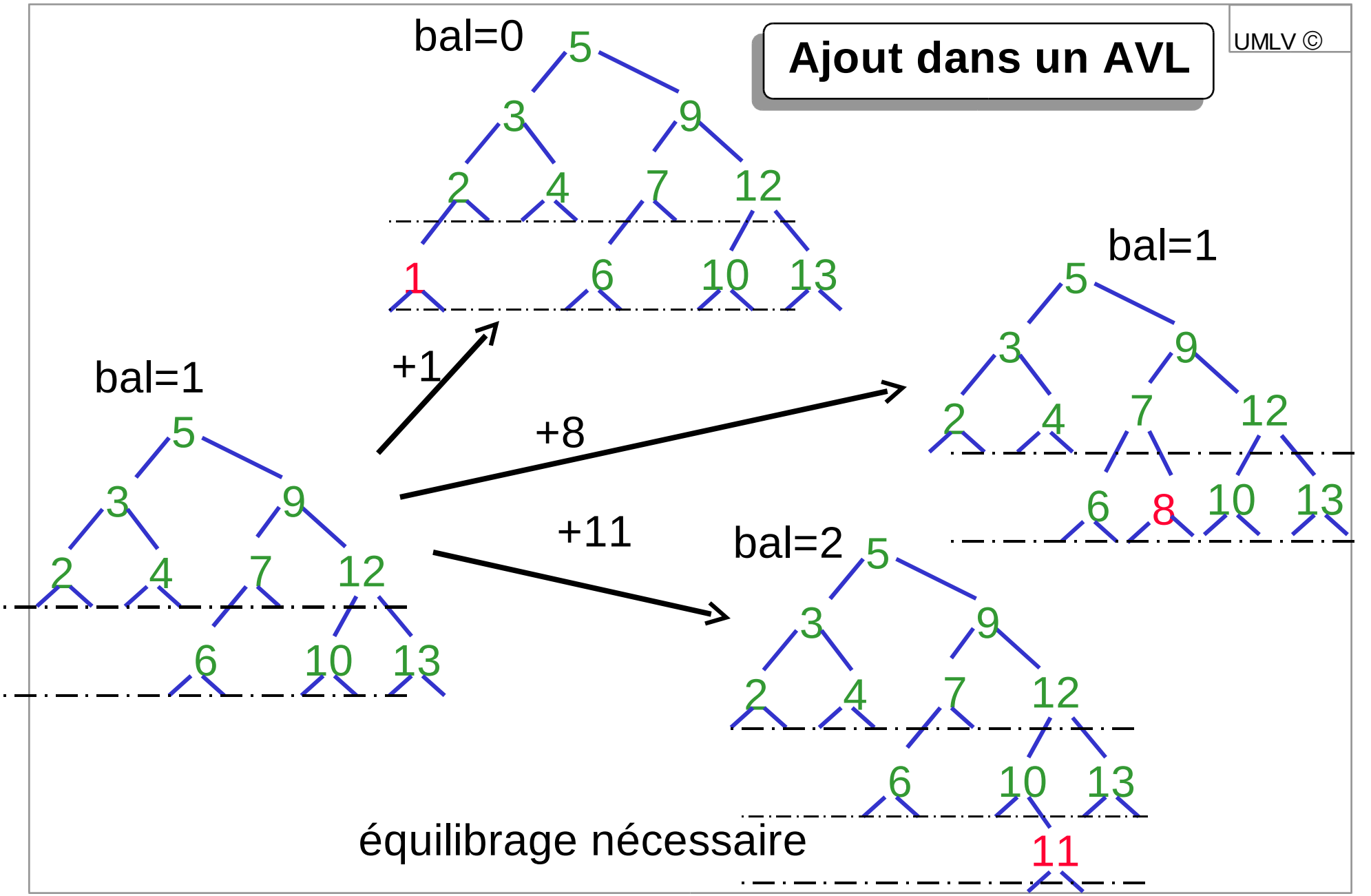
Conséquence : pour A AVL avec n nœuds et hauteur k-1

$$n \geq F_{k+2} - 1 > \Phi^{k+2}/\sqrt{5} - 2$$

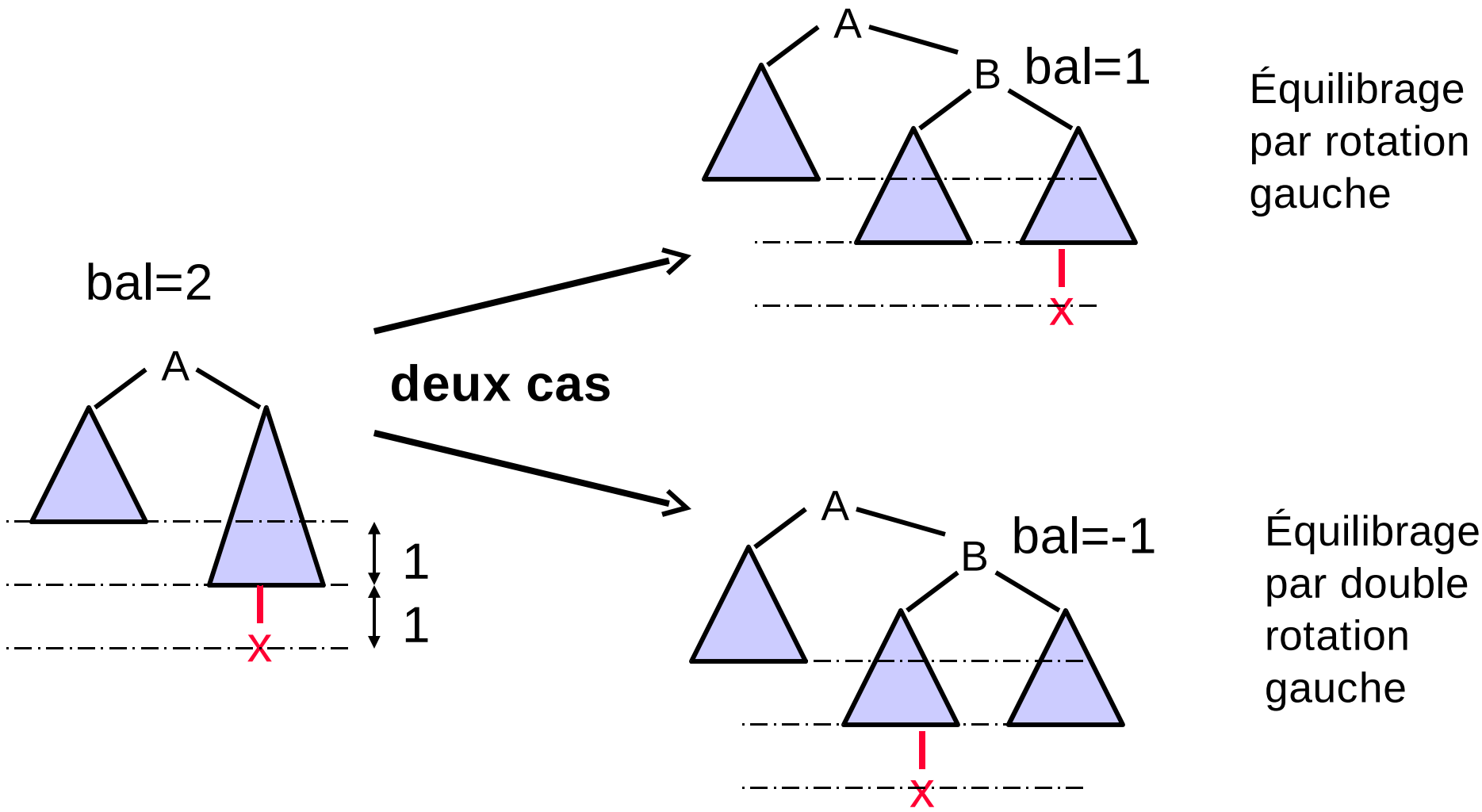
$$h(A) = k - 1 < 1{,}45 \log_2(n+2) - 0{,}33$$

Ajout dans un AVL

UMLV ©



Equilibrage (après AJOUT)



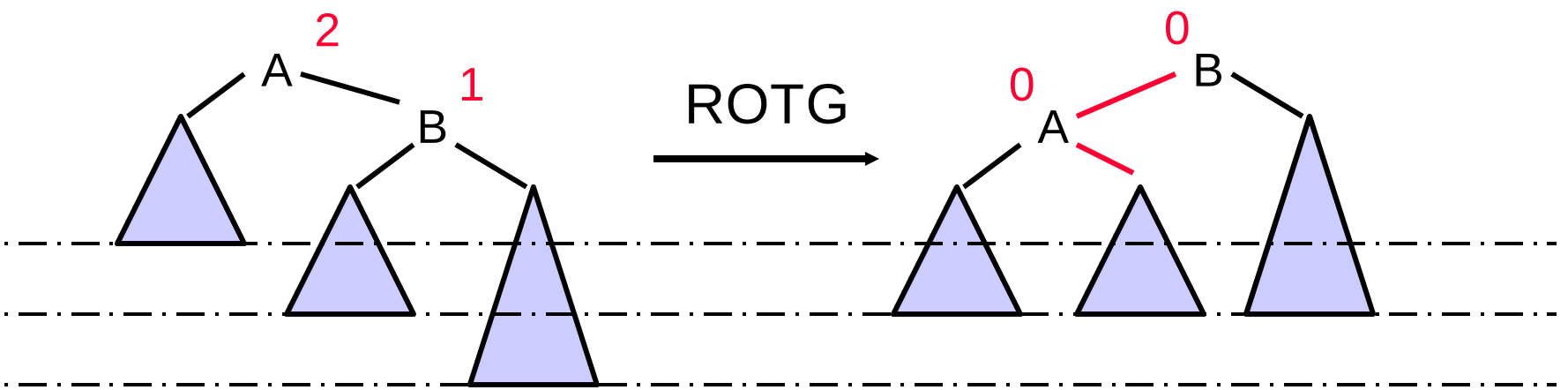
Équilibrage par rotation gauche

Équilibrage par double rotation gauche

Rotation gauche

$A = (r, A_g, A_d)$ avec A_g, A_d sont AVL

- bal(A) = 2
- bal (A_d) = 1



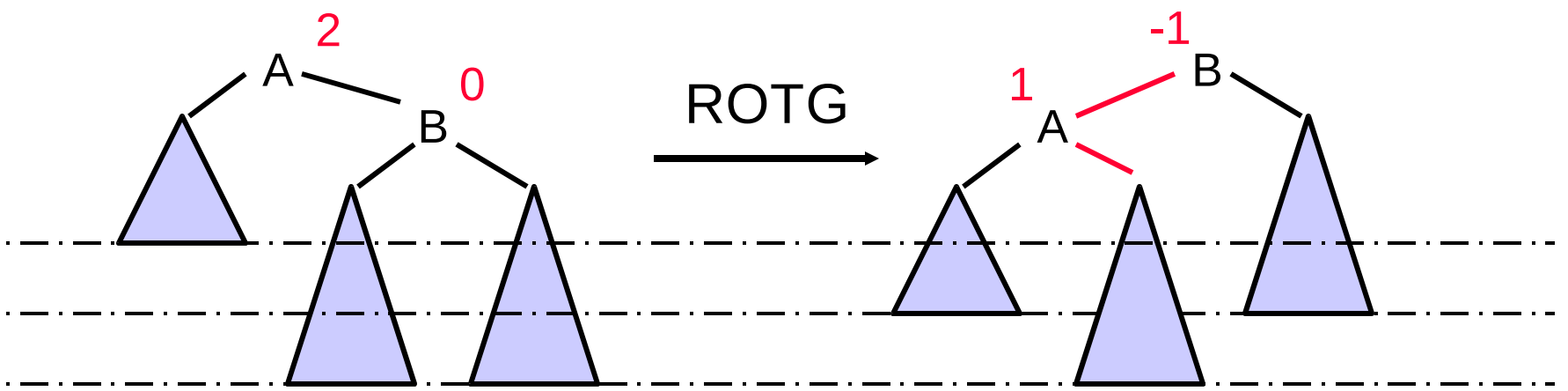
La rotation préserve l'ordre symétrique

Note : elle diminue la hauteur

Rotation gauche (suite)

$A = (r, A_g, A_d)$ avec 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

```
arbre ROTG(arbre A) {
     arbre 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;
}
```

Exercices :

- vérifier les calculs de bal
- écrire la rotation droite ROTD

```
arbre ROTD(arbre A) {
     arbre B; int a,b;
     B = A->g;
     a = A->bal;  b = B->bal;
     A->g = B->d; B->d = A;  /* rotation */
     A->bal = a-min(b,0)+1;
     B->bal = max(a+2,a+b+2,b+1);
     return B;
}
```

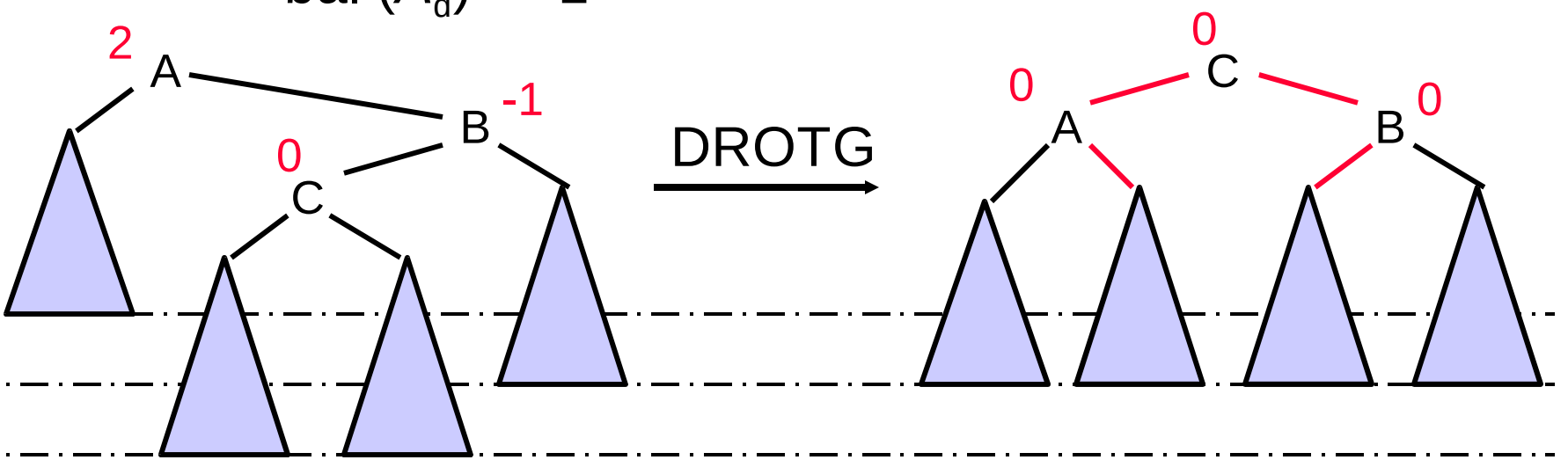
Exercice :

- vérifier les calculs de bal

Double rotation gauche

$A = (r, A_g, A_d)$ avec A_g, A_d sont AVL

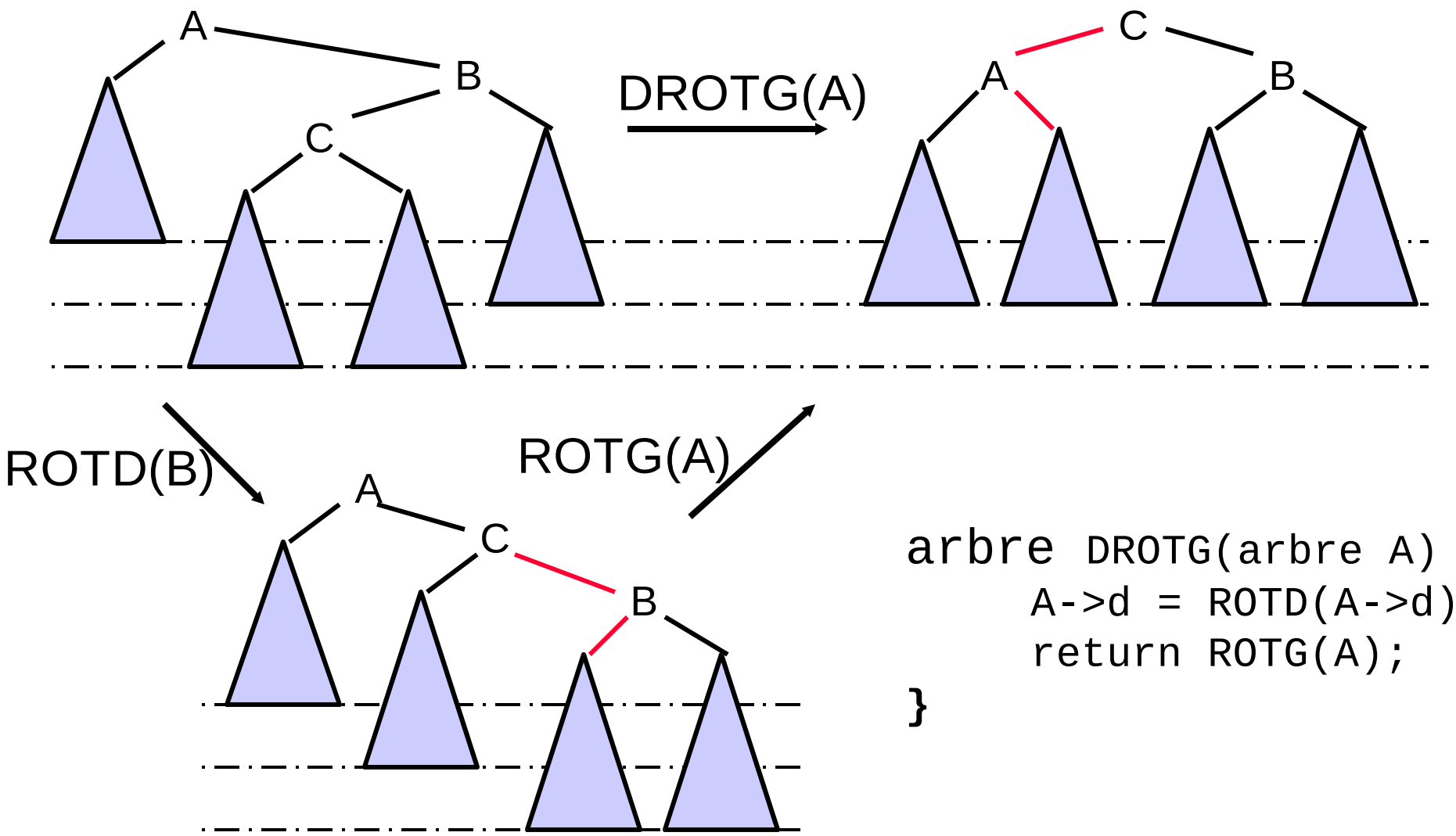
- bal(A) = 2
- bal (A_d) = -1



La rotation préserve l'ordre symétrique

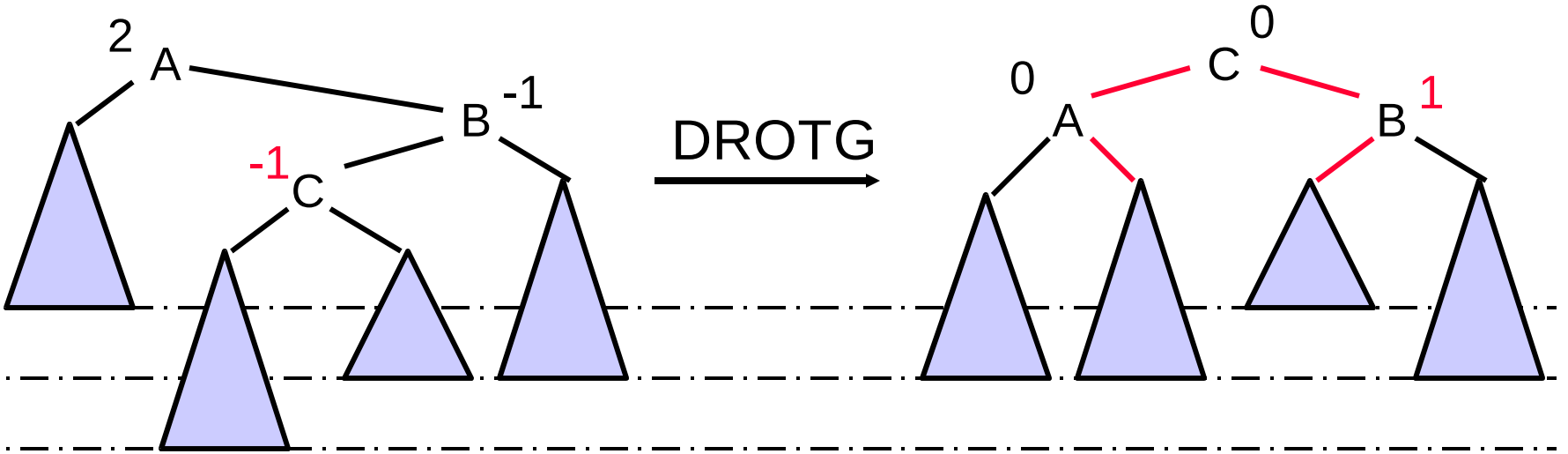
Note : elle diminue la hauteur

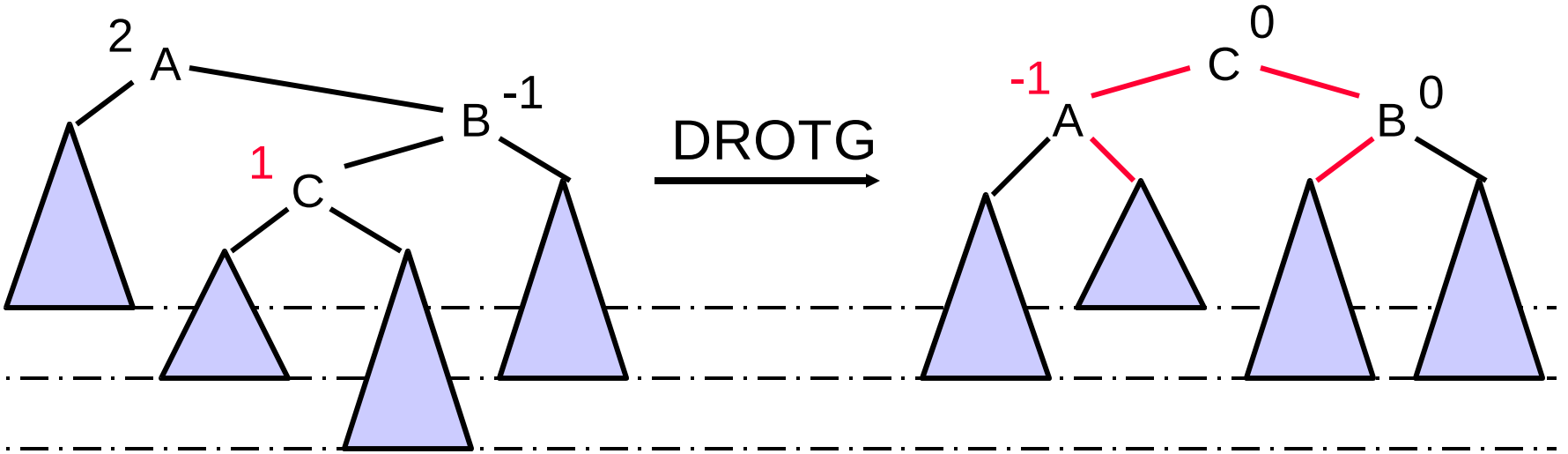
Double rotation gauche (suite)




```
arbre DROTG(arbre A) {
      A->d = ROTD(A->d);
      return ROTG(A);
}
```

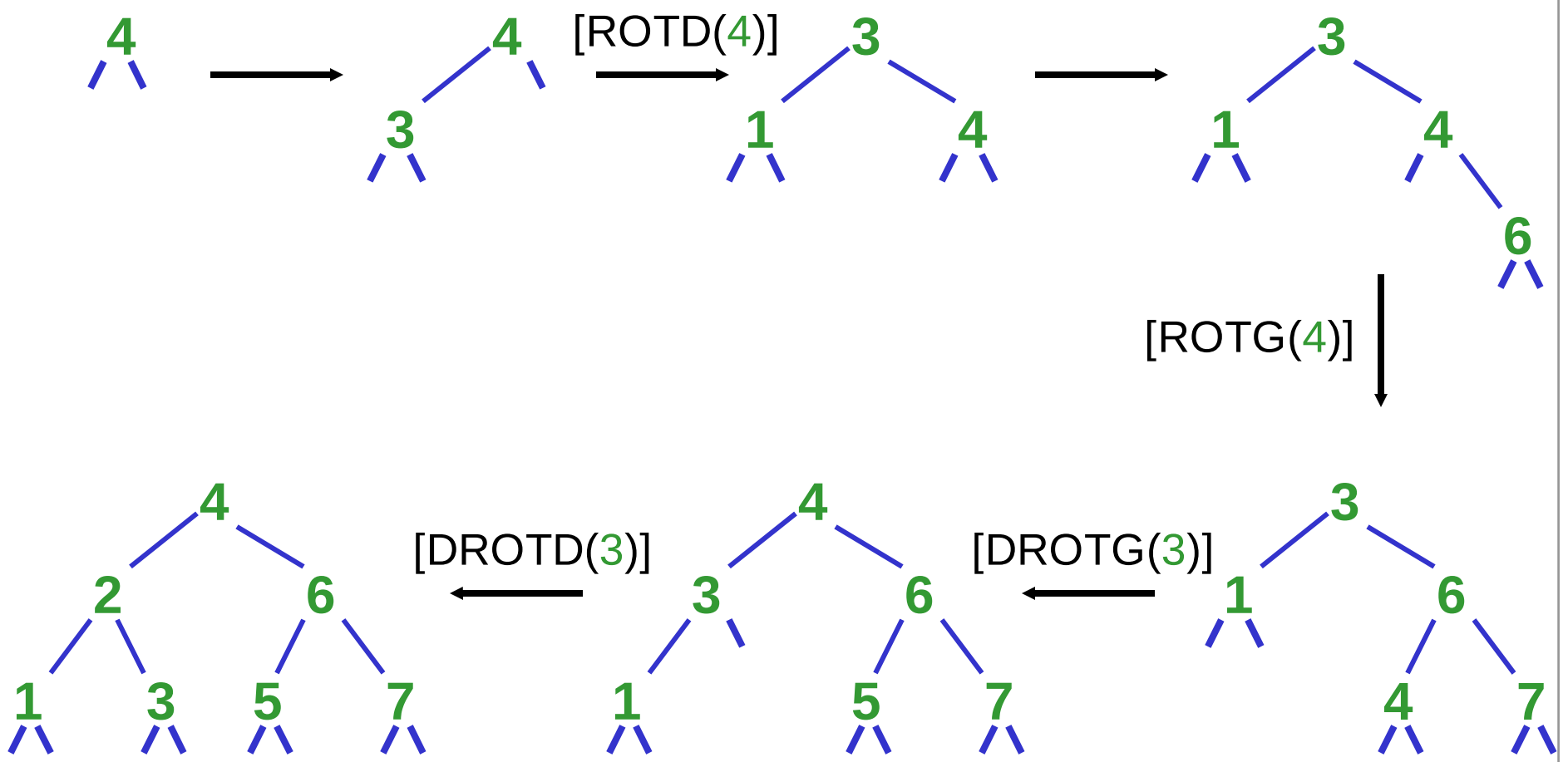
Double rotation gauche (suite)





Exemples de rotations

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



Équilibrage

Entrée

A arbre tel que

A_g, A_d sont AVL

$-2 \leq bal(A) \leq 2$

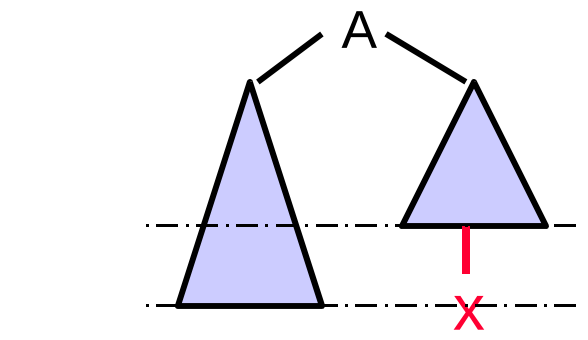
Sortie

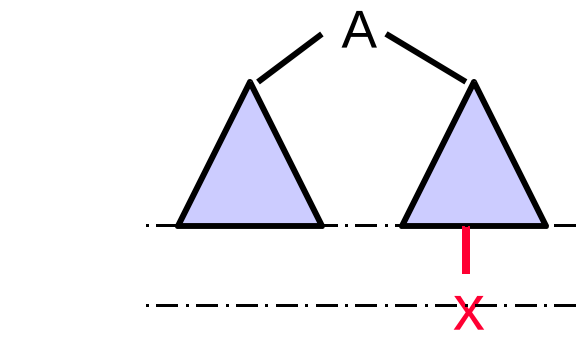
A arbre AVL

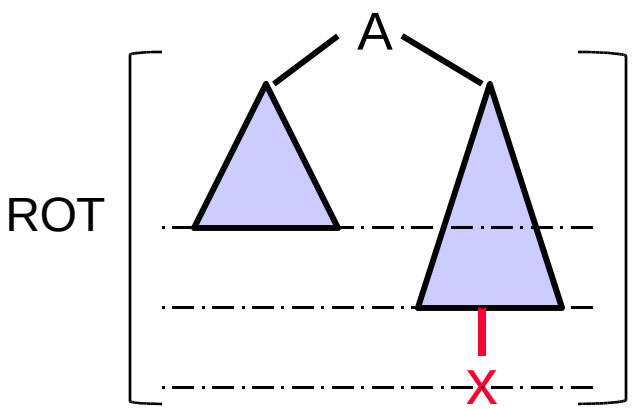
$[-1 \leq bal(A) \leq 1]$

```
arbre EQUILIBRER(arbre A) {
    if (A->bal == 2)
        if (A->d->bal >= 0)
            return ROTG(A);
        else { A->d = ROTD(A->d);
            return ROTG(A);
          }
    else if (A->bal == -2)
        if (A->g->bal <= 0)
            return ROTD(A);
        else { A->g = ROTG(A->g);
            return ROTD(A);
        }
    else return A;
}
```

Variation de 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

```
(arbre,int) AJOUTER(element x, arbre A) {
     if (A == NULL) {
          créer un nœud A;  A->g = NULL;  A->d = NULL;
          A->elt = x;  A->bal = 0;
          return (A,1);
     } else if (x == A->elt)
          return (A,0);
     else if (x > A->elt)
          (A->d,h) = AJOUTER(x,A->d);
     else
          (A->g,h) = AJOUTER(x,A->g); h = -h;
     if (h == 0)
          return (A,0);
     else
          A->bal = A->bal + h;
          A = EQUILIBRER(A);
          if (A->bal == 0) return (A,0); else return (A,1);
}
```

Temps d' un ajout

Temps d' une rotation : constant

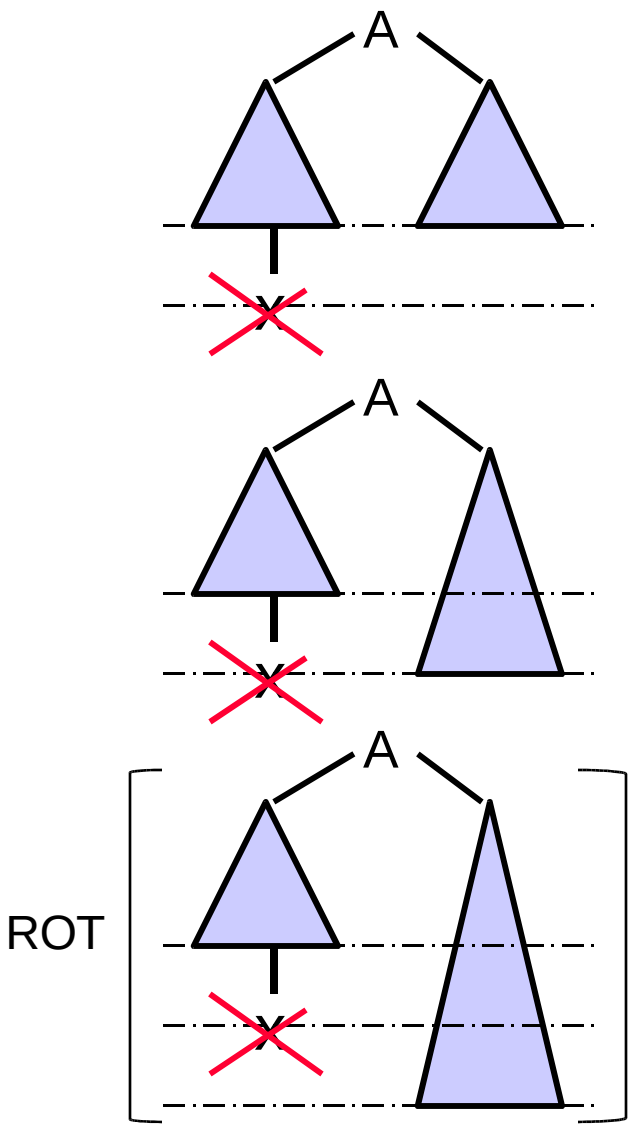
Note : AJOUTER exécute au plus une rotation
car une rotation dans cette situation rétablit la hauteur initiale de l' arbre auquel elle est appliquée

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

Variation de hauteur après RETRAIT



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 retrait

```
(arbre,int) ENLEVER(element x, arbre A) {
     if (A == NULL)
          return (A,0);
     else if (x > A->elt)
          (A->d,h) = ENLEVER(x,A->d);
     else if (x < A->elt)
          (A->g,h) = ENLEVER(x,A->g); h = -h;
     else if (A->g == NULL) {
          /* free */ return (A->d,-1);
     } else if (A->d == NULL) {
          /* free */ return (A->g,-1);
     } else {
          A->elt = min(A->d);
          (A->d,h) = OTERMIN(A->d);
     }
. . . .
```

```
. . . .
     if (h == 0)
          return (A,0);
     else {
          A->bal = A->bal + h;
          A = EQUILIBRER(A);
          if (A->bal == 0)
               return (A,-1);
          else
               return (A,0);
     }
}
```

Entrée

A arbre AVL non vide

```
(arbre,int) OTERMIN(arbre A) {
     if (A->g == NULL) {
          min = A->elt;
          /* free */ return (A->d,-1);
     } else
          (A->g,h) = OTERMIN(A->g); h = -h;
     if (h == 0)
          return (A,0);
     else {
          A->bal = A->bal + h;
          A = EQUILIBRER(A);
          if (A->bal == 0)
               return (A,-1);
          else
               return (A,0);
     }
}
```

Temps d' un retrait

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

Exemple : suppression du maximum dans un arbre de Fibonacci

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)