Chapitre 1

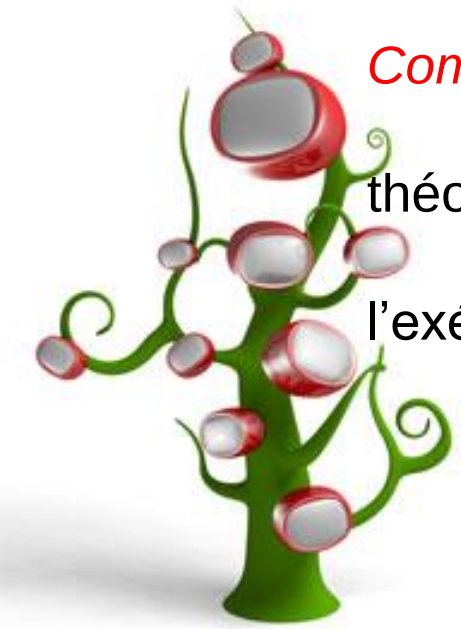
Les Arbres AVL

I. Introduction:

✓Complexité des algorithmes

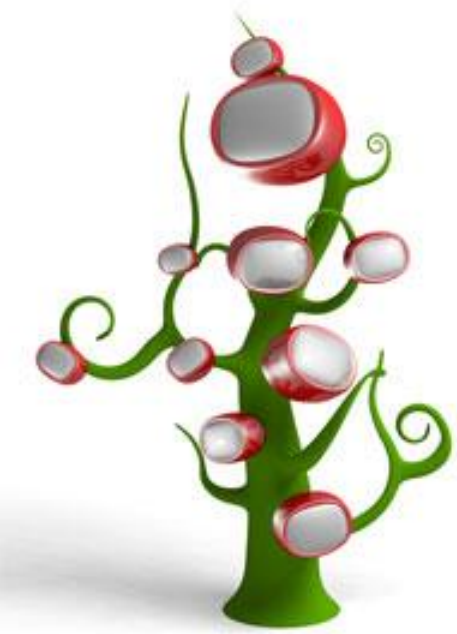
Deux algorithmes produisant les mêmes résultats peuvent être très différents du point de vue des méthodes utilisées, de leur complexité et de leur efficacité.

Complexité théorique : c'est un ordre de grandeur des coûts théoriques indépendant des conditions pratiques de l'exécution.



Soient A et B deux algorithmes traitant le même problème (de taille n) ayant les complexités temporelles (nombre d'opérations effectuées) suivantes:

- A : $O(n^{\alpha}) = a.n^{\alpha} + O(n^{\alpha-1})$
- B : $O(n^{\beta}) = b.n^{\beta} + O(n^{\beta-1})$



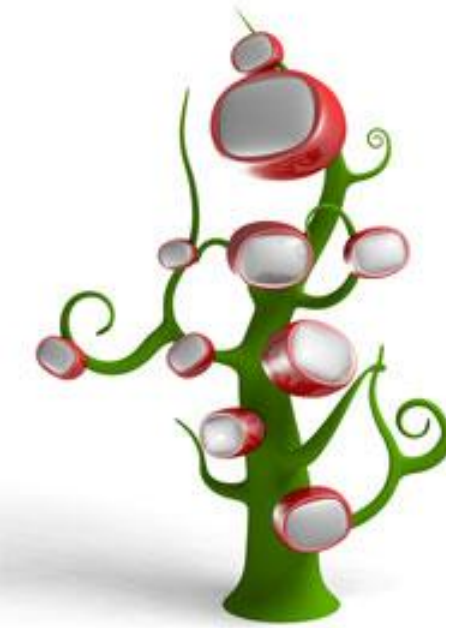
Nous disons que A est meilleure que B si :

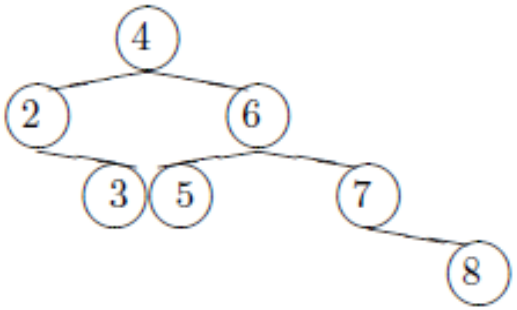
- $\alpha < \beta$ alors A est meilleurs
- $\alpha = \beta$ et $a < b$ alors A est meilleure

✓ Motivation des AVL

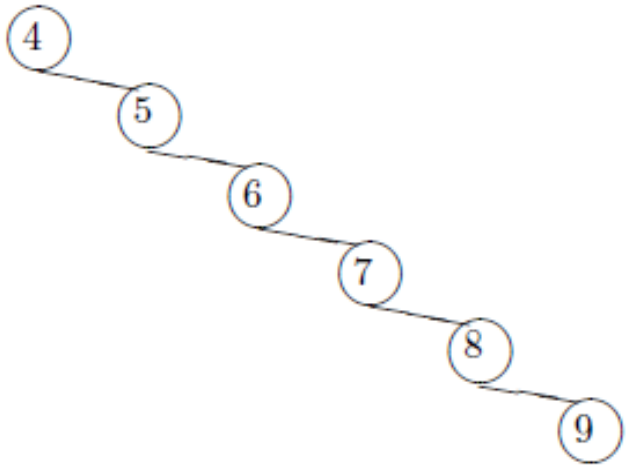
L'efficacité des algorithmes d'insertion et de suppression dans un ABR dépend considérablement de la forme de l'arbre sur lequel on travaille.

Exemple 1 (Formes d'un ABR)





$ABR1$: équilibré



$ABR2$: équivalent à une liste chaînée

La recherche d'un élément dans l'ABR2 (appelé arbre dégénéré) prend un temps en O (n) alors qu'elle prend un temps en O ($\log_2(n)$) dans le cas de l'ABR1 (**n** étant le nombre de nœuds de l'arbre).

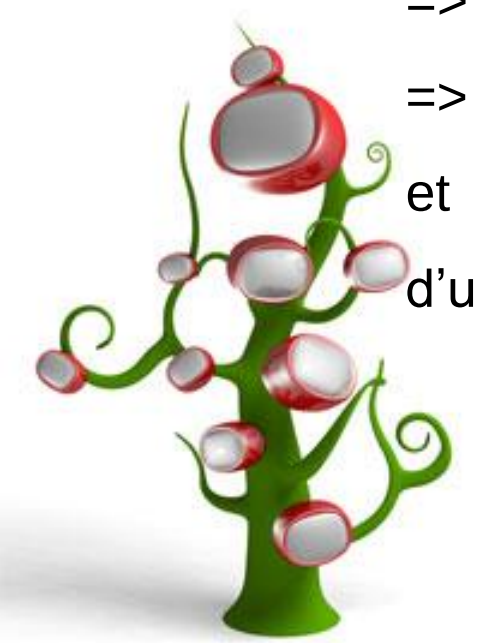
=> On a donc intérêt à avoir des arbres les moins hauts possible.

Remarque

ABR1 est appelé arbre équilibré ou arbre AVL.

=> La recherche dans un AVL est plus efficace.

=> Il faut donc *maintenir l'équilibre de tous les nœuds* au fur et à mesure des opérations d'insertion ou de suppression d'un nœud dans un AVL.



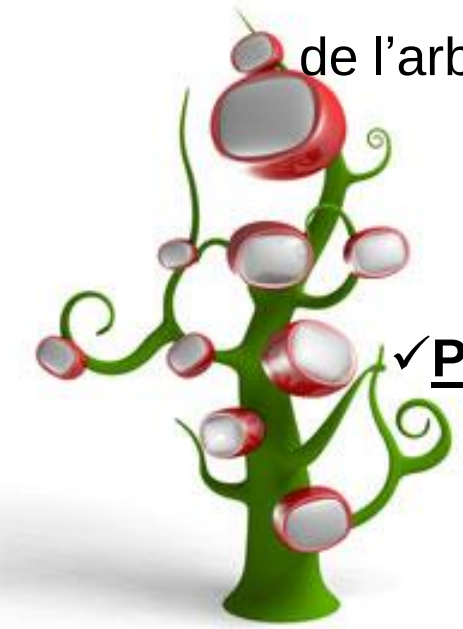
II.Arbres AVL:

✓**Définition**

Un arbre AVL (Adelson-Velskii, Landis, 1962) est un ABR particulier où on contrôle la forme générale de l'arbre afin d'optimiser les requêtes de recherche à tout moment.

Un AVL vérifie la propriété fondamentale suivante :

les hauteurs des deux sous arbres gauche et droite de tout nœud de l'arbre diffèrent de 1 au plus.



$$| h(fg) - h(fd) | \leq 1$$

✓**Propriété:** tout sous-arbre d'un AVL est un arbre AVL.

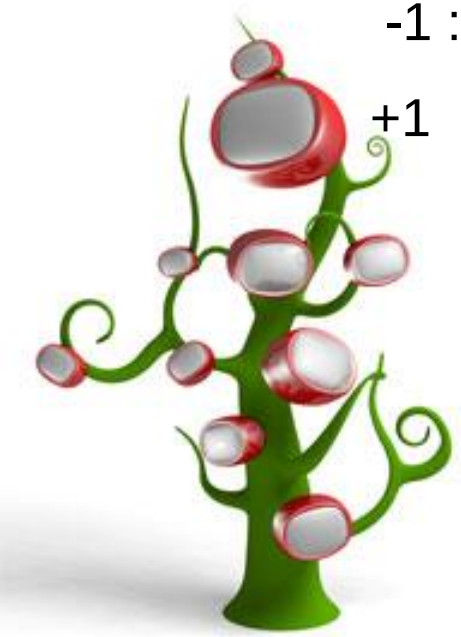
III. Rééquilibrage d'un AVL:

Pour rééquilibrer un AVL après une insertion ou une suppression, on suppose que tout nœud contient un champ annexe «Bal» (comme balance) contenant la différence de hauteur entre le fils droit et le fils gauche :

0 : les deux sous-arbres du nœud ont la même hauteur ;

-1 : le sous-arbre gauche et plus haut que le sous arbre droit

+1 : le sous-arbre droit et plus haut que le sous-arbre gauche.



Après chaque insertion ou suppression, il faut regarder si cela a entraîné un déséquilibre, et si c'est le cas on rééquilibre l'arbre par des « Rotations ».

Rotation sur les AVLs:

Il y a deux sortes de rotations : **Simple** et **Double**.

a. Rotation Simple:

Elle peut être *droite* ou *gauche*.

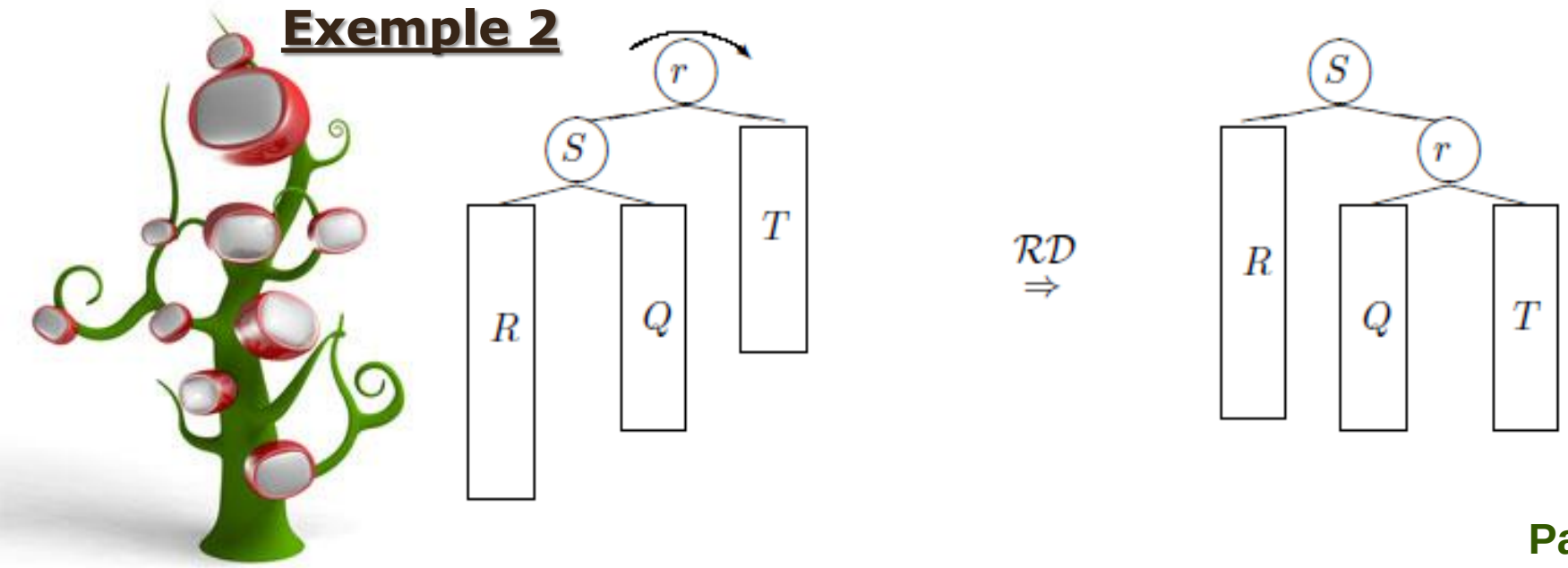
❑ Rotation droite Simple: (RD)

La rotation droite simple est définie par :

RD (r ; (S ; R ; Q); T) = (S ; R ; (r ; Q ; T))

Mettre S à la racine (à la place de r) et réaffecter le sous-arbre droit de S au sous-arbre gauche de r.

Exemple 2



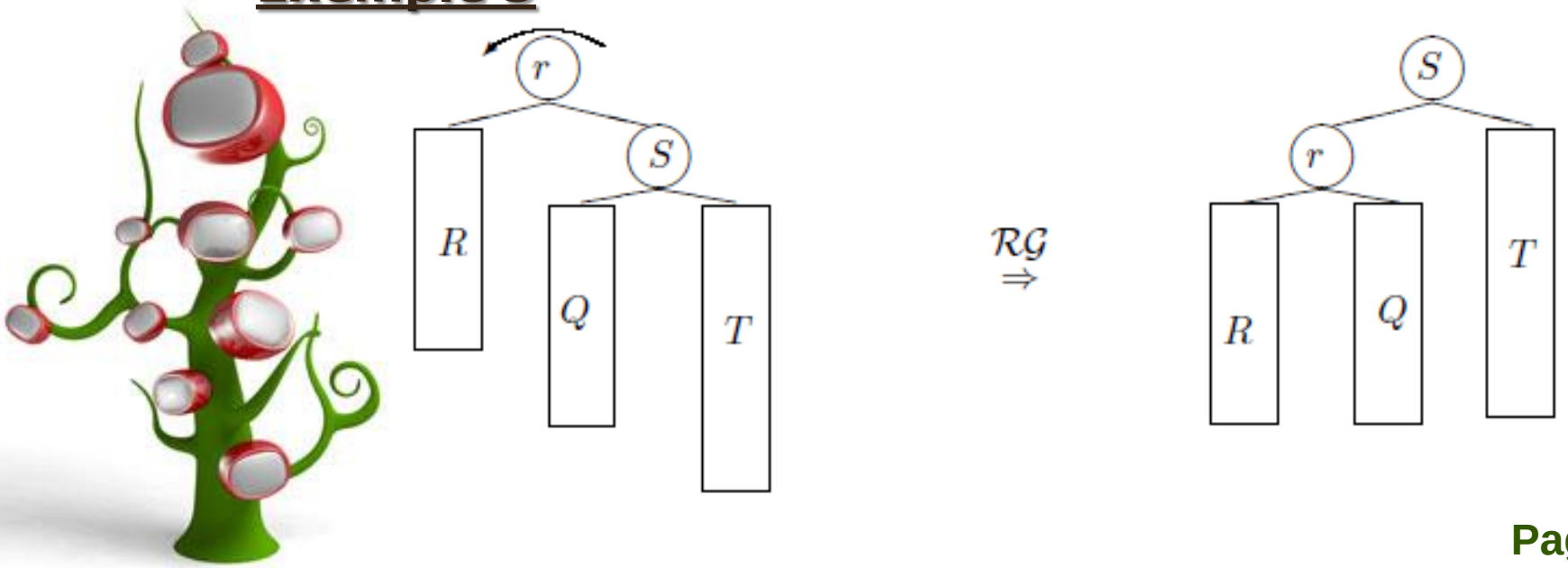
❑ Rotation Gauche Simple: (RG)

La rotation gauche simple est définie par :

RG (r ; R ; (S ; Q ; T)) = (S ; (r ; R ; Q); T)

Mettre S à la place de r et réaffecter le sous-arbre gauche de S au sous arbre droit de r.

Exemple 3



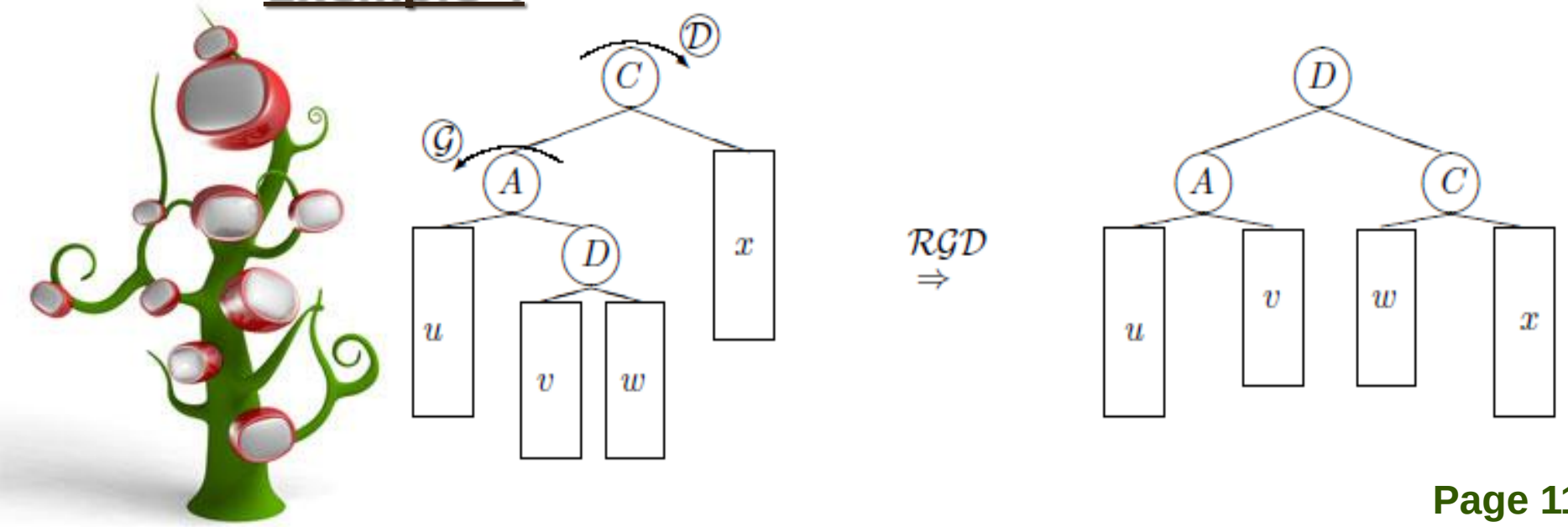
b. Rotation Double:

Une rotation double est simplement composée de deux rotations simples.

❑ Rotation Gauche Droite: (RGD)

Elle correspond à une rotation gauche du sous-arbre gauche (A) suivie d'une rotation droite (C).

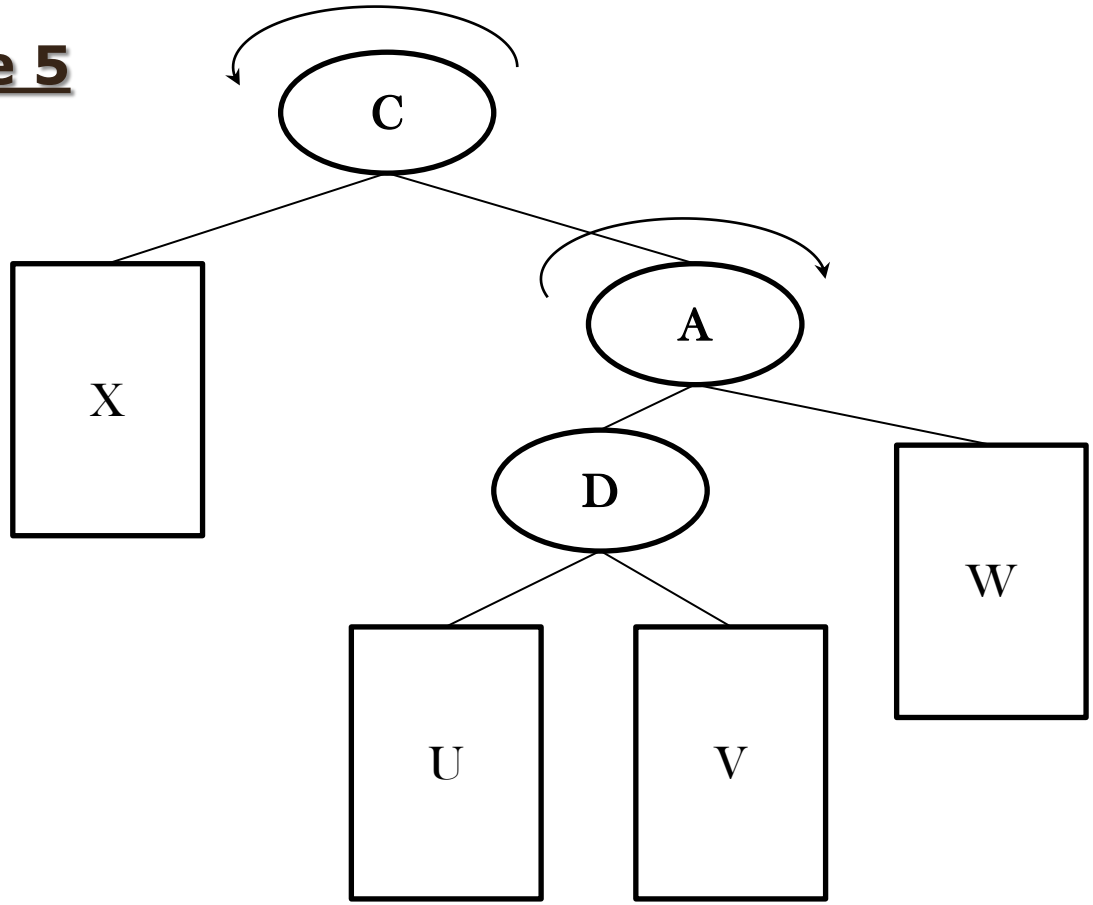
Exemple 4

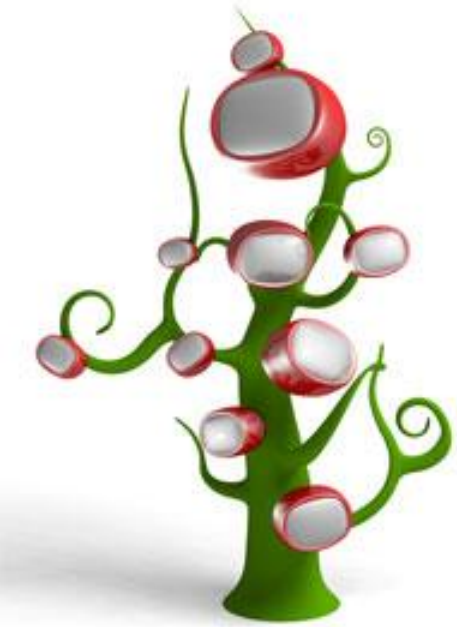


❑ Rotation Droite Gauche: (RDG)

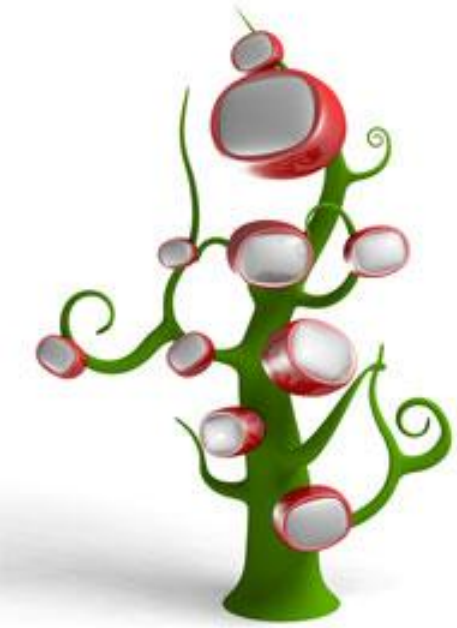
Elle correspond à une rotation droite du sous-arbre droit (A) suivie d'une rotation gauche (C).

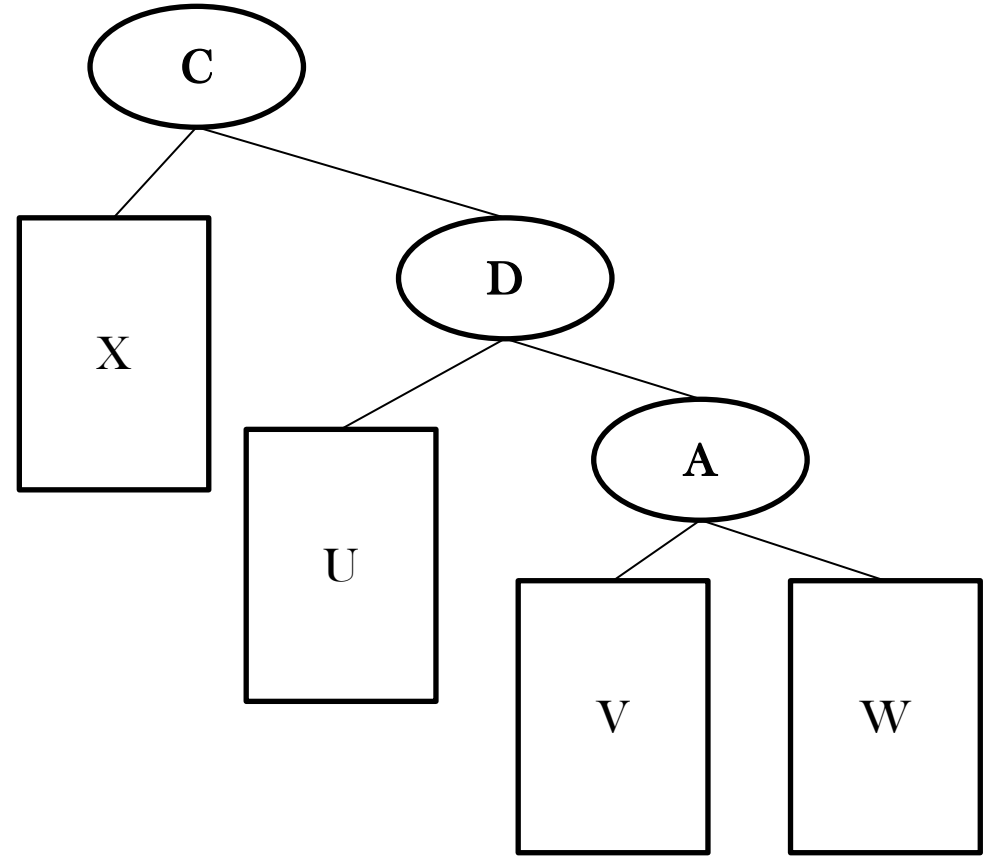
Exemple 5



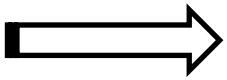


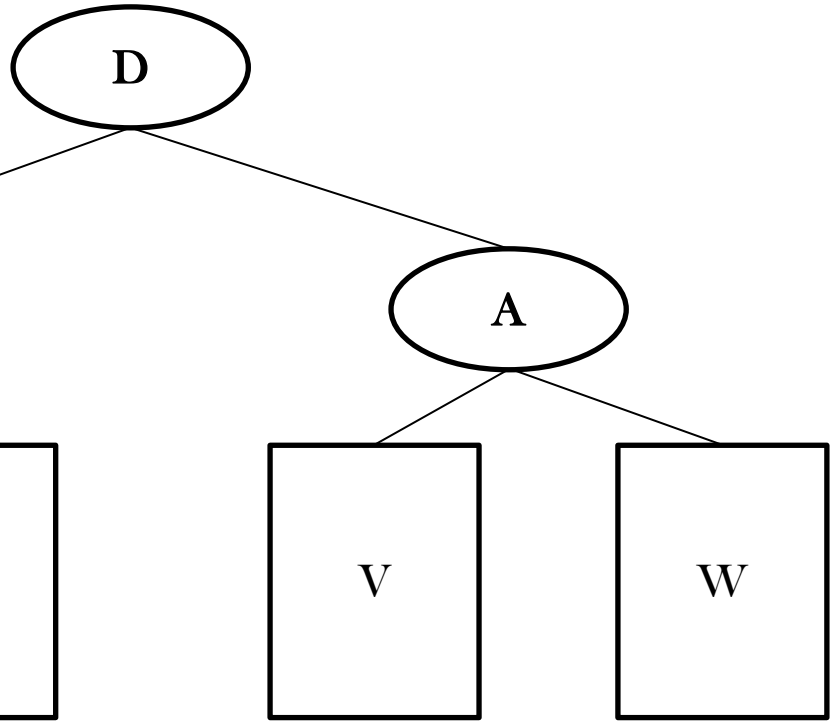
Rotation Droite (A)

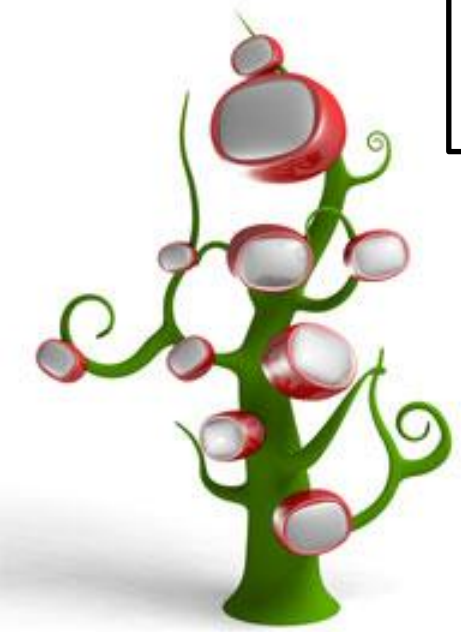




Rotation Gauche (C)





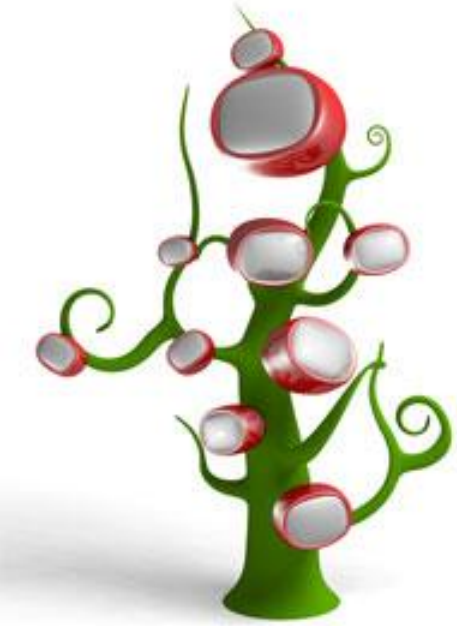


IV. Insertion d'un élément dans un AVL:

Soit à insérer un élément x dans un AVL A (r ; A_g ; A_d).

On suppose que x n'est pas déjà présent dans A et que A était précédemment équilibré.

=> Si x < r alors deux cas se présentent après l'insertion de x :

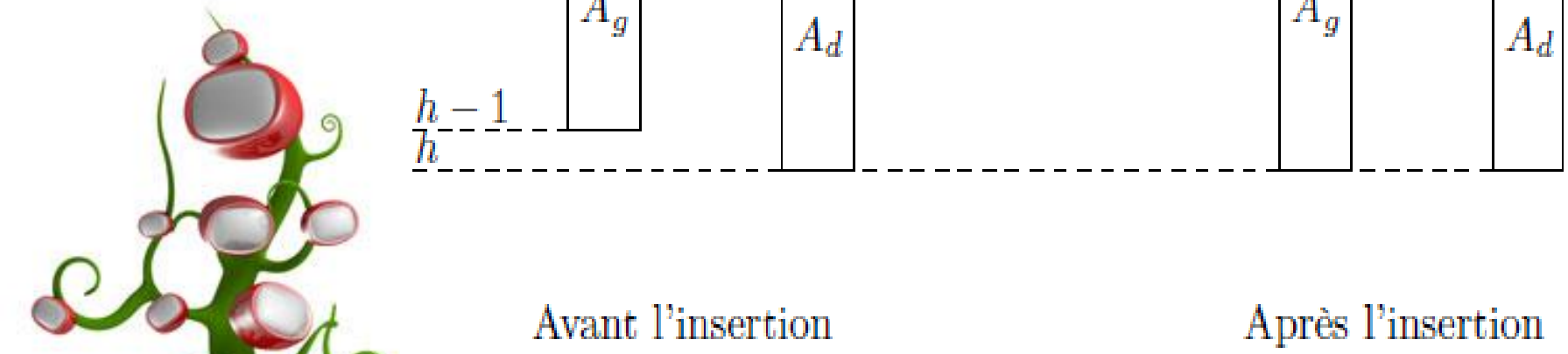


1er cas: la hauteur de A_g n'augmente pas,

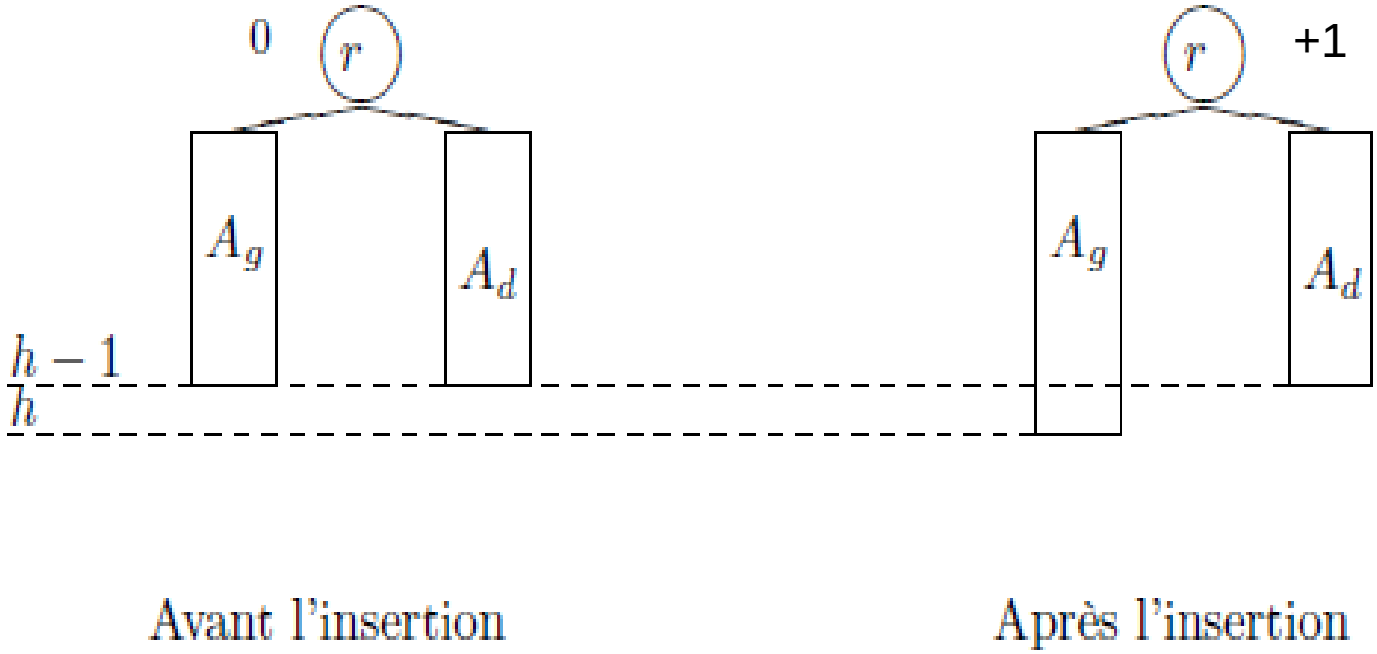
l'arbre reste alors équilibré ;

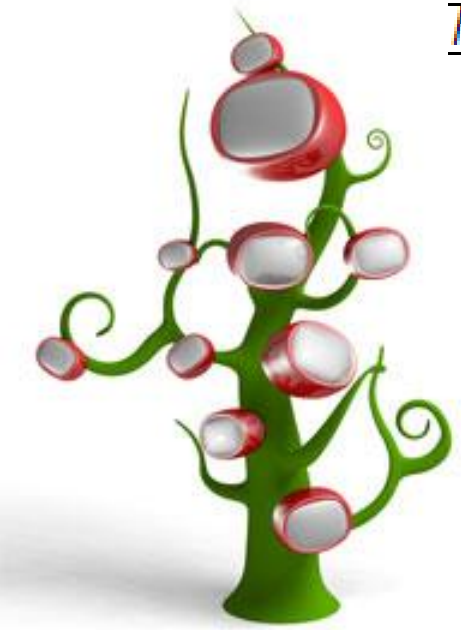
2ème cas: la hauteur de A_g augmente, 3 cas se présentent:

1. *$h(A_g)$ était égale à $h(A_d)$ - 1* : dans ce cas l'arbre devient mieux équilibré qu'avant l'insertion. Ici, la hauteur de l'arbre n'augmente pas.

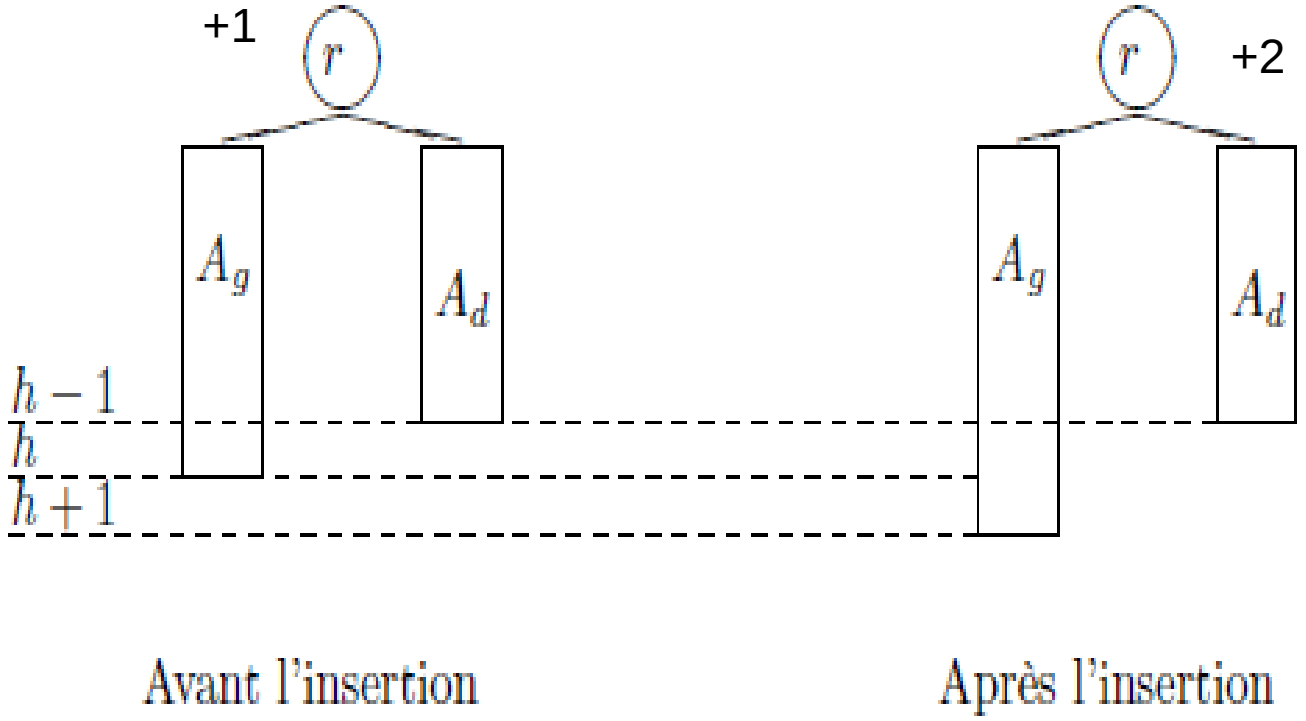


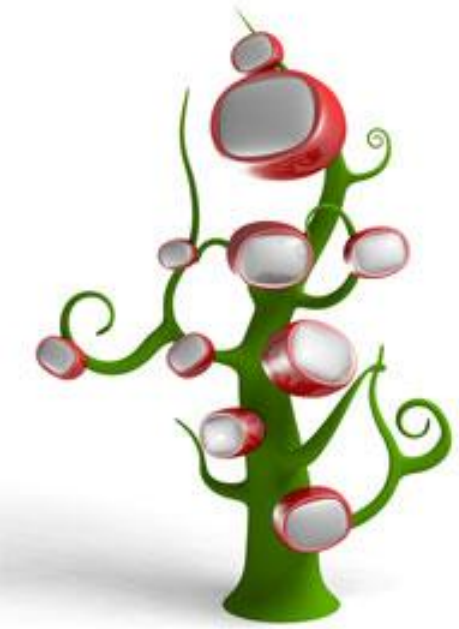
2. *h(A_g) était égale à h(A_d)* : l'arbre reste équilibré





3. *h(A_g) était égale à h(A_d) + 1* : l'arbre devient déséquilibré



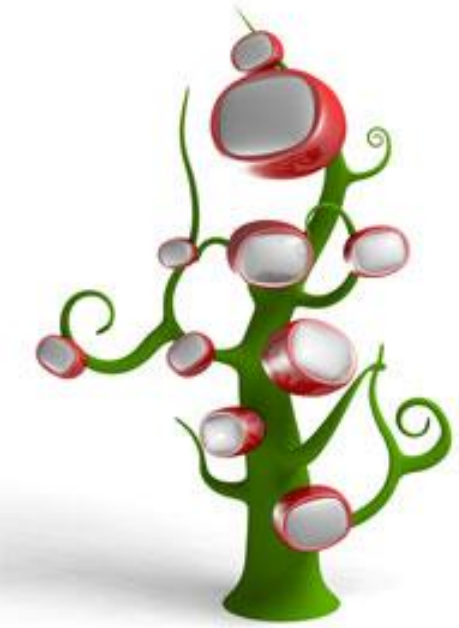


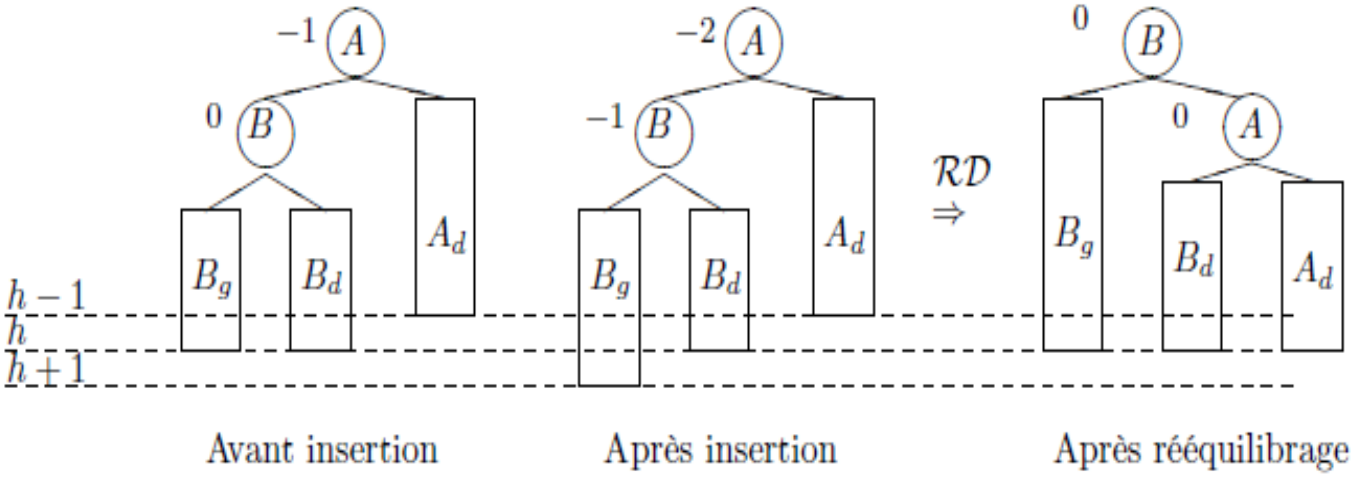
Étudions maintenant les différentes possibilités qui se présentent lorsque l'arbre est déséquilibré.

Supposons qu'avant l'insertion $h(A_d) = h - 1$, $h(A_g) = h$ et que h a augmenté après l'insertion.

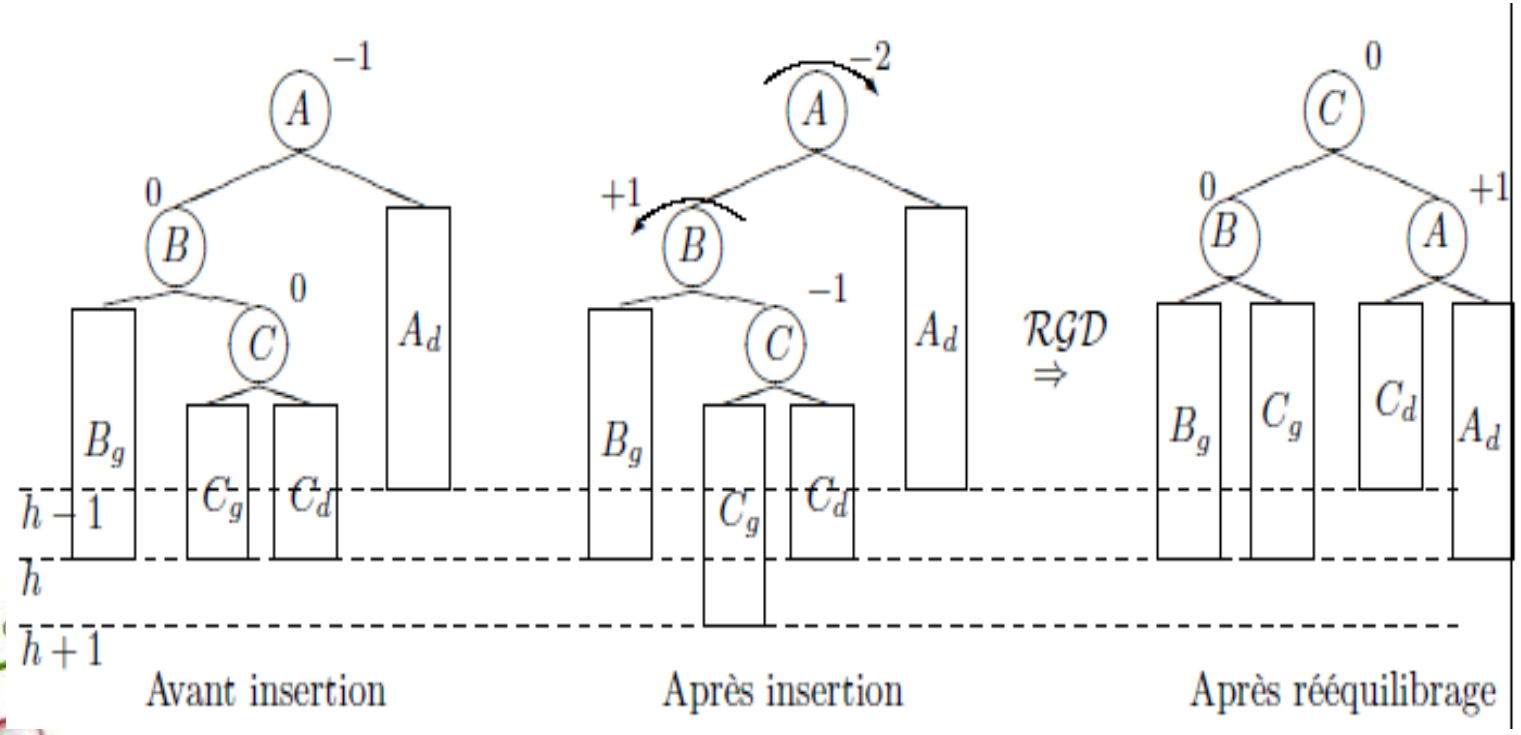
Soit B le sous-arbre gauche de A. Deux cas se présentent alors :

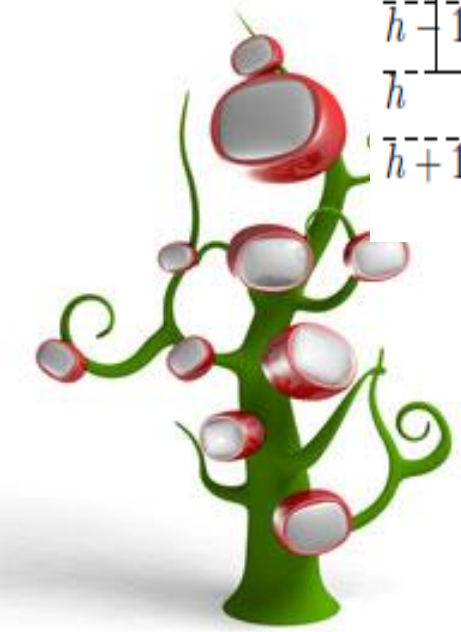
a. <u>B penche gauche</u> : le déséquilibre est causé par le fils gauche de B





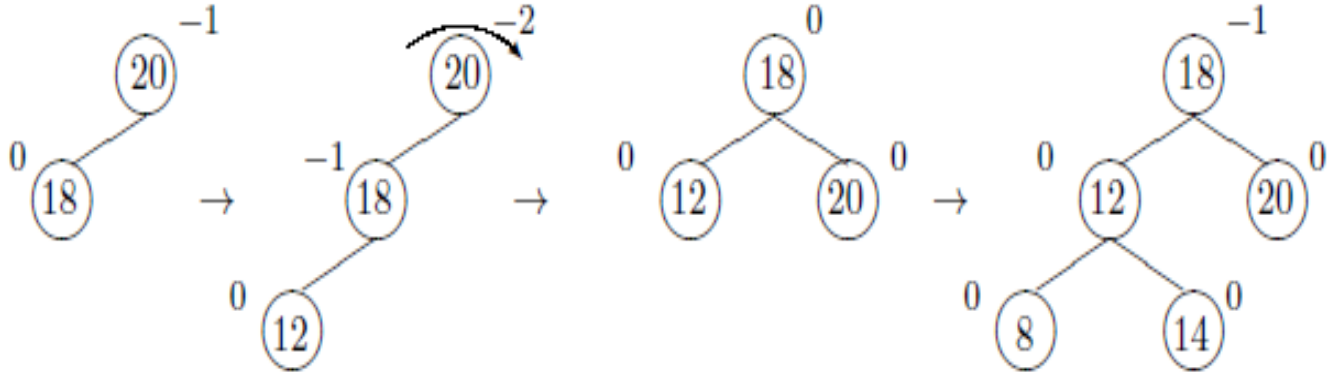
b. B penche droite : le déséquilibre est causé par le fils droit de B



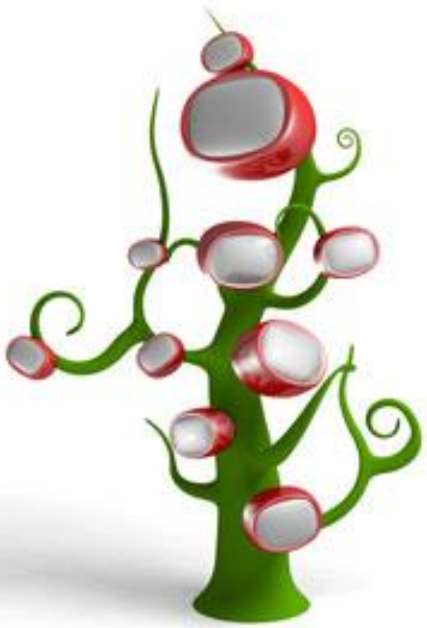


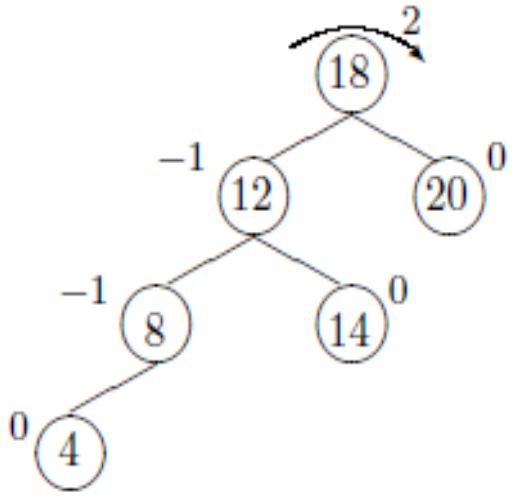
Application 1 ()

Insérer dans l'ordre les élélements 20, 18, 12, 14, 8 et 4. dans un AVL vide.

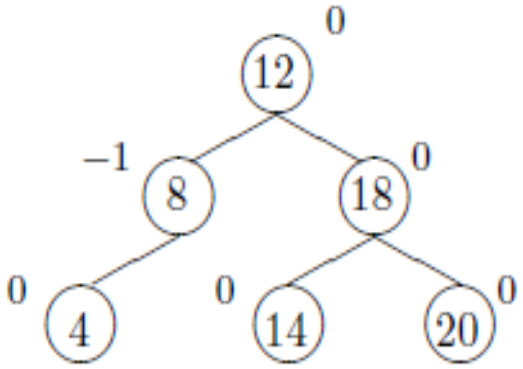


Insertion de 12 Rotation Droite Insertion de 8 et 14 Insertion de 4



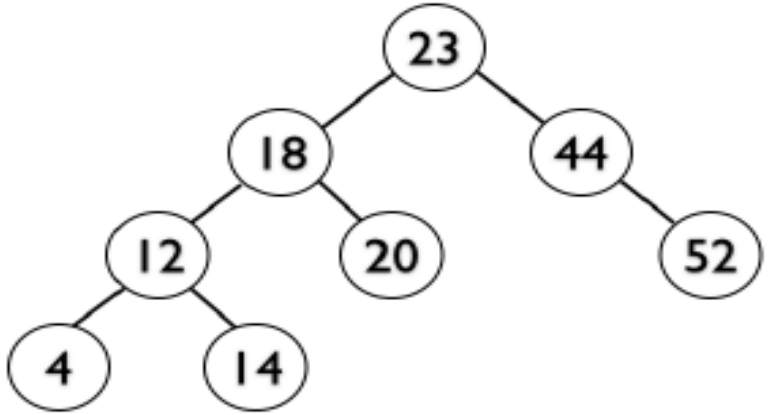


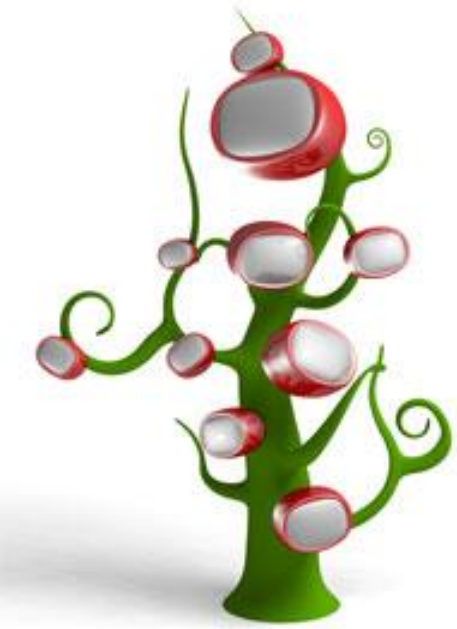
Rotation Droite



Application 2 ()

Insérer l'élément 16 dans l'AVL suivant

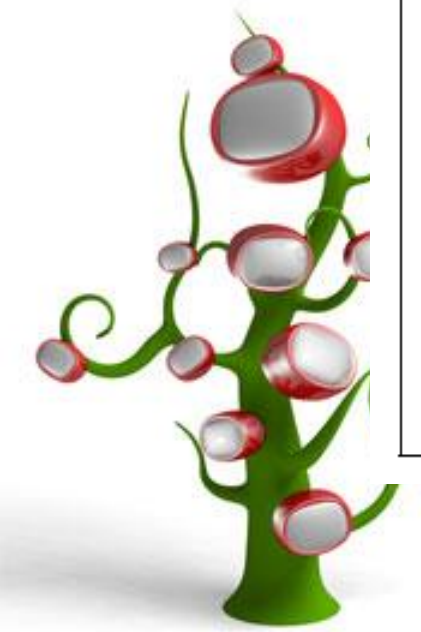




Principe de l'insertion:

Après avoir insérer un élément dans un arbre AVL, on remonte dans l'arbre (en suivant le chemin de l'insertion) et on teste si les nœuds qu'on croise sont balancés.

Dans le cas contraire, on les re-balance par une rotation simple ou une rotation double et on continue à monter vers la racine.



si le déséquilibre et causé par le fils gauche f_g de A

alors on effectue une rotation simple sur A ou bien une rotation double gauche-droite (sur f_g, puis A)

sinon on effectue une rotation simple gauche sur A ou bien une rotation double droite-gauche (sur f_d, puis A)

finsi

Remarque

Une insertion d'un élément dans un AVL donnera lieu à au plus une opération de rotation (simple ou double).

V. Suppression d'un élément dans un AVL:

La suppression dans un AVL repose, comme la suppression dans un ABR sur le remplacement de l'élément à supprimer par un autre élément.

Cette élimination est susceptible de réduire la hauteur du sous arbre et donc de modifier l'équilibre de l'arbre tout entier. Il faut donc rééquilibrer l'arbre.

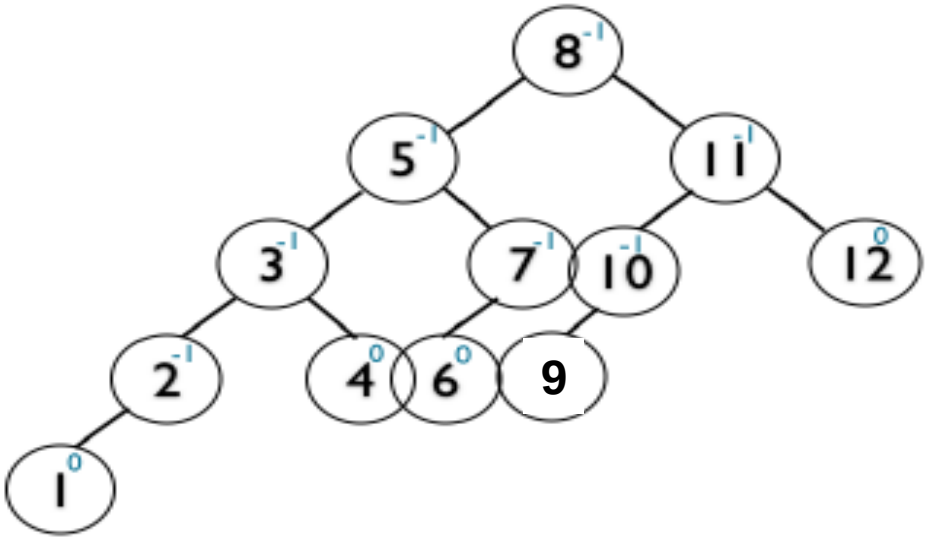
Remarque

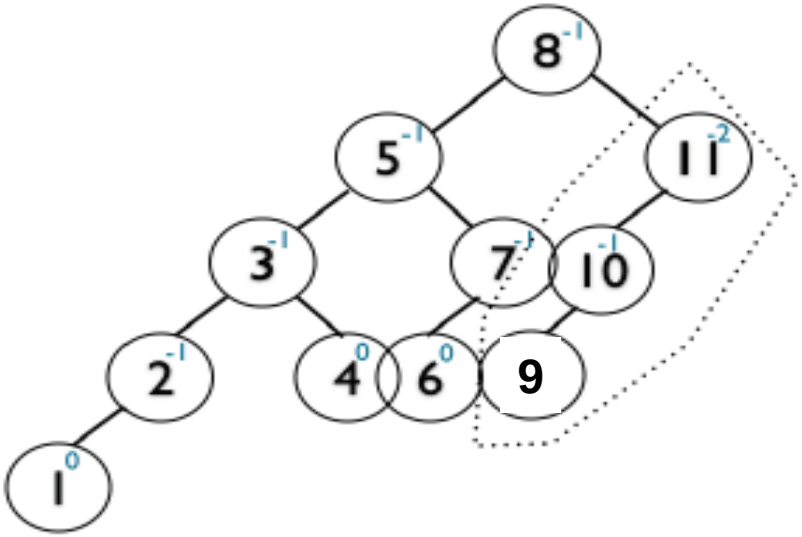
Une suppression peut entraîner plusieurs rotations.

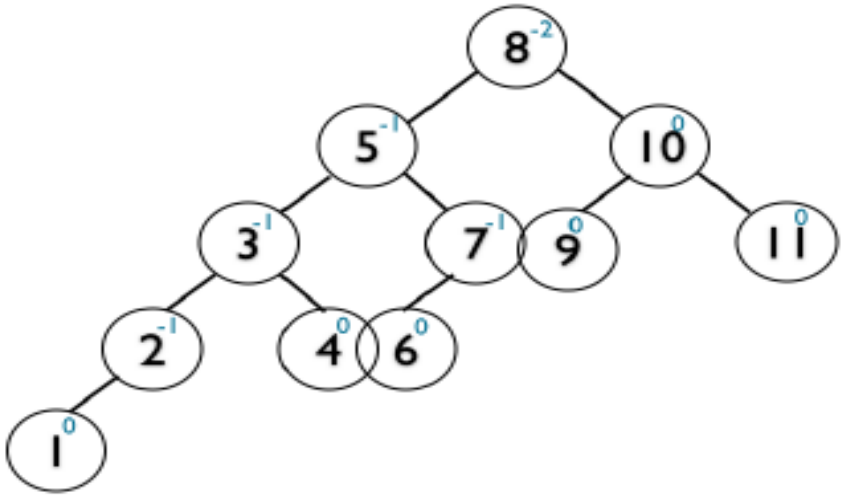
En effet, une seule rotation n'assure pas forcément que l'arbre soit équilibré.

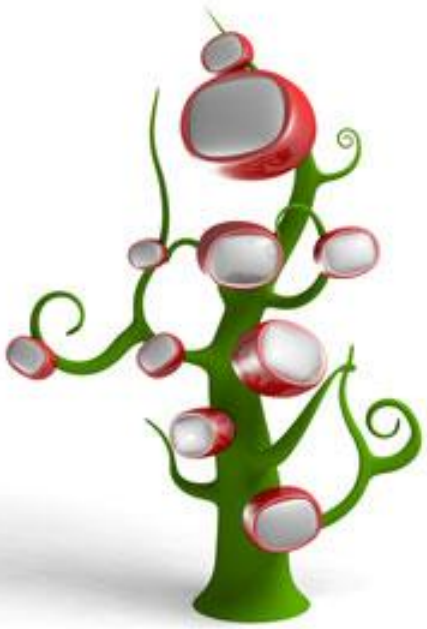
Exemple

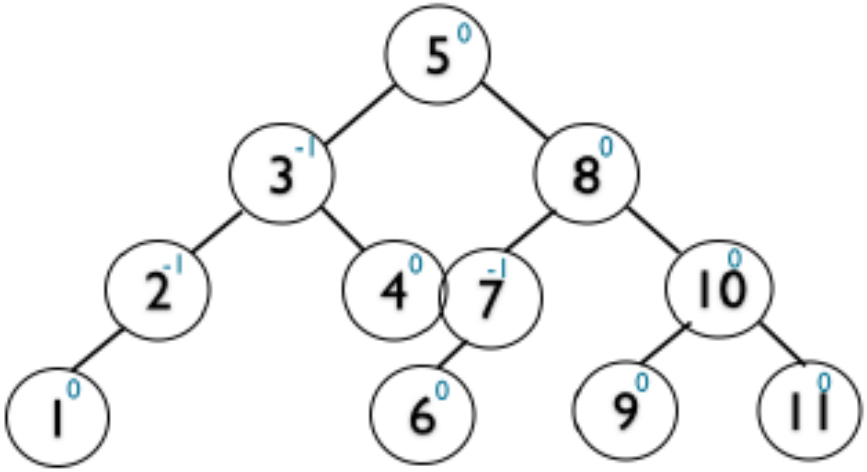
La suppression de l'élément 12 dans l'arbre suivant provoque 2 rotations :




8 -1
5 -1
11 -2
3 -1
7 -1
10 -1
2 -1
4 0
6 0
9
1 0


8 -2
5 -1
10 0
3 -1
7 -1
9 0
11 0
2 -1
4 0
6 0
1 0

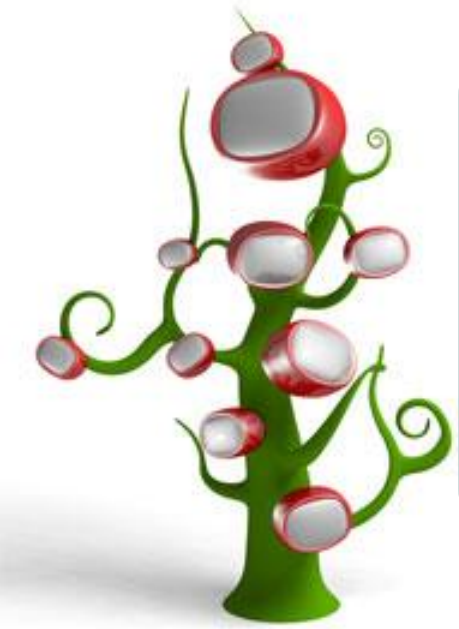



5 0
3 -1
8 0
2 -1
4 0
7 -1
10 0
1 0
6 0
9 0
11 0

VI. Représentation des AVL par les pointeurs:

On suit la même représentation que pour le cas des ABR, en ajoutant un champ « bal »dans chaque nœud de l'arbre pour contenir la différence entre la hauteur du fils droit et le fils gauche.

On définit alors le type NOEUD comme suit :



```
Type  NOEUD = enregistrement
                    val : TypeElement
                    Bal : entier
                    fg, fd : ^NOEUD
                finenregistrement

Type  AVL = ^NOEUD
```

Exercice:

1. Écrire une procédure ***Rotation_Droite(var A: AVL)*** permettant d'effectuer une rotation droite sur A.
2. Écrire une procédure ***Rotation_Gauche(var A: AVL)*** permettant d'effectuer une rotation gauche sur A.
3. Déduire la procédure ***Insertion_AVL(…)***
4. Déduire aussi la procédure ***Suppression_AVL (…)***