TD h-équilibré : les AVL
Correction

deseq(D)	deseq(F)	ΔH	rotation
-2	-1	-1	rg
	0	0	
	1	-1	rdg

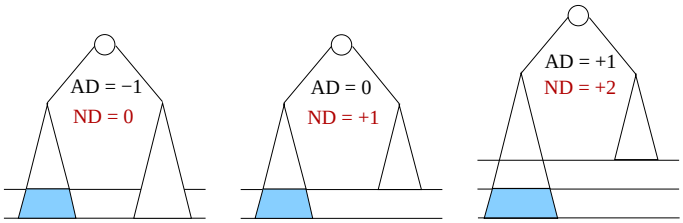
deseq(D)	deseq(B)	ΔH	rotation
2	-1	-1	rgd
	0	0	rd
	1	-1	

Table 1 – Rotations et changements de hauteur

Solution 3.1 **(Insertion)**

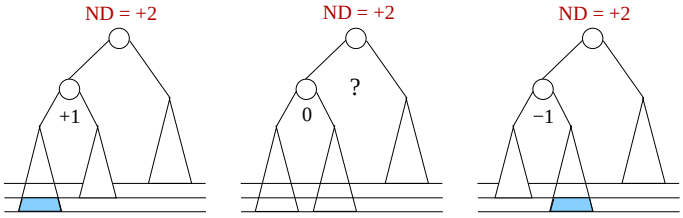
On écrit ici une version récursive de l'ajout d'un élément dans un AVL. Celui-ci sera basé sur le principe de l'ajout en feuilles des arbres binaires de recherche, le rééquilibrage se faisant à la remontée.

1. Changement de hauteur ?
   (a) On considère une insertion dans le sous arbre gauche, qui a augmenté la hauteur de ce sous-arbre.
   Soient :
   AD, l'ancien déséquilibre,
   ND, le nouveau déséquilibre.



$\Delta H = 0$ $\Delta H = 1$ Étude détaillée

Étude du cas où le nouveau déséquilibre (ND) est égal à +2 :



rd $\Delta H = 0$ Cas impossible rdg $\Delta H = 0$

   (b) **Conclusion**
   Soient :
   — h_i la hauteur avant insertion
   — h_f la hauteur après insertion
   — h_r la hauteur après rotation
   Si le nouveau déséquilibre de l'arbre est :

   0 : $h_f = h_i$.

   1 : $h_f = h_i + 1$.

   2 : D'après l'exercice 1.3, si le déséquilibre du fils gauche est :

   1 : $h_r = h_f - 1$, or $h_f = h_i + 1$ donc $h_r = h_i$

   0 : cas impossible après une insertion

   -1 : $h_r = h_f - 1$, or $h_f = h_i + 1$ donc $h_r = H_i$

   Dans tous les cas, après une rotation, l'arbre retrouve sa hauteur initiale.

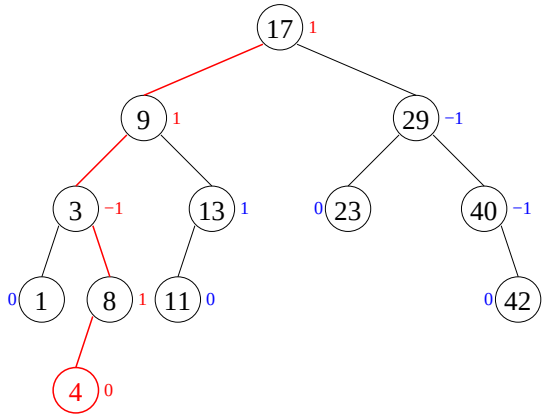
2. *Principe d'insertion dans les AVL :*
   - Recherche de la position d'insertion. Si la clé n'est pas déjà présente :
     - Création du nouveau nœud
     - Variation de hauteur égale à 1
   - A la remontée si la hauteur du sous-arbre (où a eu lieu l'insertion) a changé alors modification du déséquilibre du nœud courant (+1 si retour de gauche, -1 sinon). La nouvelle valeur est :

     0 : pas de variation de hauteur

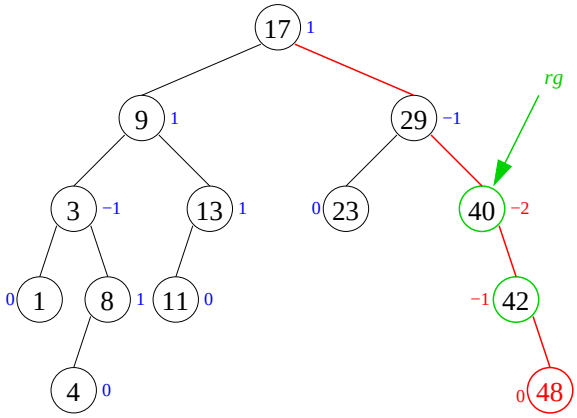
     +1 ou -1 : la hauteur a changé (+1)

     +2 : si déséquilibre du fils gauche est égal à -1 alors RGD sinon RD, pas de variation de hauteur

     -2 : si déséquilibre du fils droit est égal à +1 alors RDG sinon RG, pas de variation de hauteur.

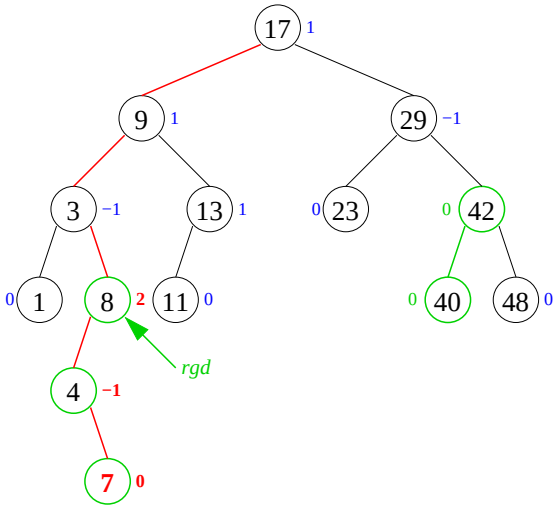


insertion 4

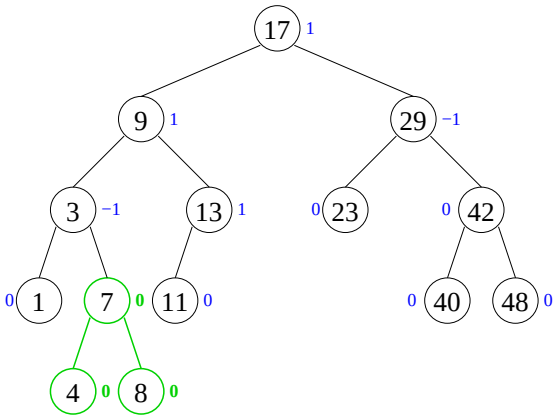


insertion 48

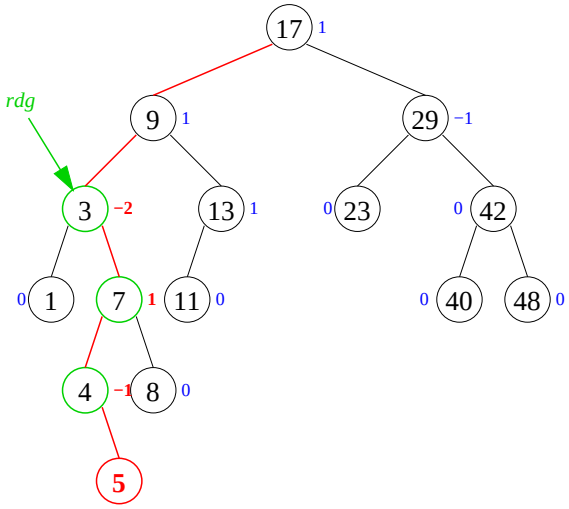
3.



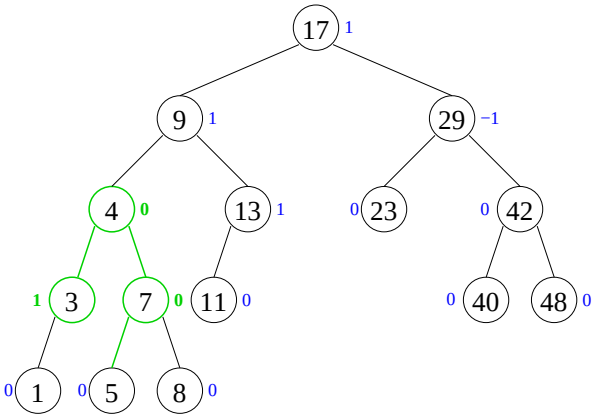
rotation gauche (40) puis insertion 7



rotation gauche-droite (rgd 8)



insertion 5



rotation droite-gauche (3)

4. *La fonction :*

   Spécifications : La fonction `insertAVL(x, A)` insère x dans l'AVL A (sauf si la clé est déjà présente) qu'elle retourne (la fonction récursive retourne en plus un booléen qui indique le changement de hauteur de l'arbre).

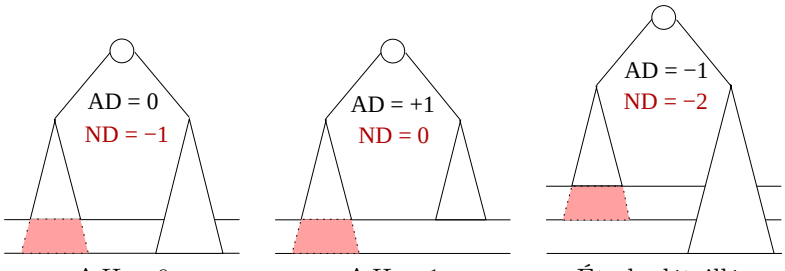
```
def __insertAVL(x, A):
    if A == None:
        return (avl.AVL(x, None, None, 0), True)
    elif x == A.key:
        return (A, False)
    elif x < A.key:
        (A.left, dh) = __insertAVL(x, A.left)
        if not dh:
            return (A, False)
        else:
            A.bal += 1
            if A.bal == 2:
                if A.left.bal == 1:
                    A = rr(A)
                else:
                    A = lrr(A)
            return (A, A.bal == 1)
    else:   # x > A.key
        (A.right, dh) = __insertAVL(x, A.right)
        if not dh:
            return (A, False)
        else:
            A.bal -= 1
            if A.bal == -2:
                if A.right.bal == -1:
                    A = lr(A)
                else:
                    A = rlr(A)
            return (A, A.bal == -1)

def insertAVL(x, A):
    (A, dh) = __insertAVL(x, A)
    return A
```

Solution 3.2 **(Suppression)**

La suppression se fera sur le même modèle que l'insertion :
- La suppression même se fera sur le modèle de celle vue en td pour un arbre binaire de recherche.
- Le rééquilibrage se fera à la remontée.

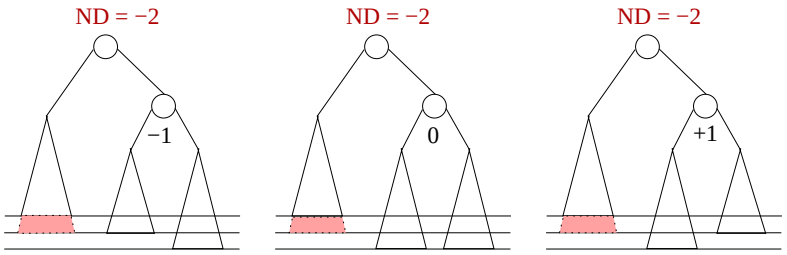
1. On considère une suppression dans le sous-arbre gauche qui a diminué la hauteur de cet arbre.

   Soient : **AD**, l'ancien déséquilibre, **ND**, le nouveau déséquilibre.



$\Delta H = 0$ $\Delta H = 1$ Étude détaillée

   Étude du cas où le nouveau déséquilibre (ND) est égal à -2.



RG $\Delta H = 1$ | RG $\Delta H = 0$ | RDG $\Delta H = 1$

Conclusion :
— h_i la hauteur avant suppression
— h_f la hauteur après suppression
— h_r la hauteur après rotation

Si le nouveau déséquilibre de l'arbre est :

0 : $h_f = h_i - 1$.

-1 : $h_f = h_i$.

-2 : D'après l'exercice 1.3, si le déséquilibre du fils droit est égal à :

-1 : $h_r = h_f - 1$, **et** $h_f = h_i$ donc $h_r = h_i - 1$.

0 : $h_r = h_f$, **et** $h_f = h_i$ donc $h_r = h_i$.

1 : $h_r = h_f - 1$, **et** $h_f = h_i$ donc $h_r = h_i - 1$.

2. *Principe de suppression dans les AVL :*

On considère ici encore que l'arbre ne contient que des valeurs distinctes.

- Recherche de l'élément à supprimer.
  - Si l'élément est en feuille alors suppression physique du nœud le contenant. La hauteur diminue.
  - Si l'élément est dans un point simple alors suppression physique du nœud le contenant : est remplacé par son fils. La hauteur diminue.
  - Si l'élément est dans un nœud double alors remplacer la clé par le maximum du sous-arbre gauche ou le minimum du sous-arbre droit suivant le déséquilibre du nœud courant. Puis, relancer la suppression de la clé remontée dans l'arbre où elle a été prise.

- A la remontée, si la hauteur du sous-arbre (où a eu lieu la suppression) a changé alors modification du déséquilibre du nœud courant (-1 si retour de gauche, +1 sinon). Si son déséquilibre est égal à :

  0 : la hauteur a changé (-1)

  1 ou -1 : pas de variation de hauteur

  2 : si déséquilibre du fils gauche est égal à :

  -1 : RGD, la hauteur a changé (-1)

  0 : RD, pas de variation de hauteur

  1 : RD, la hauteur a changé (-1)

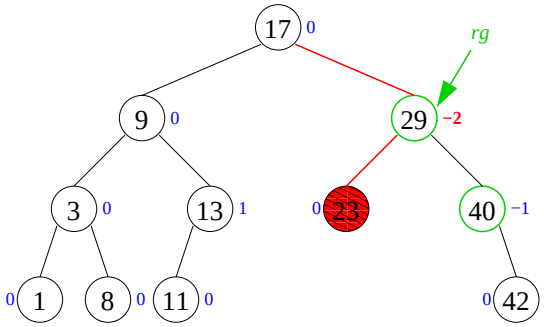
  -2 : si déséquilibre du fils droit est égal à :

  -1 : RG, la hauteur a changé (-1)

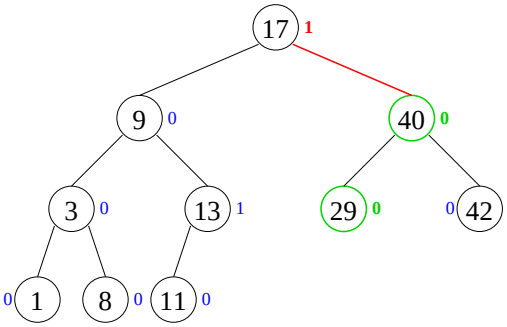
  0 : RG, pas de variation de hauteur

  1 : RDG, la hauteur a changé (-1)

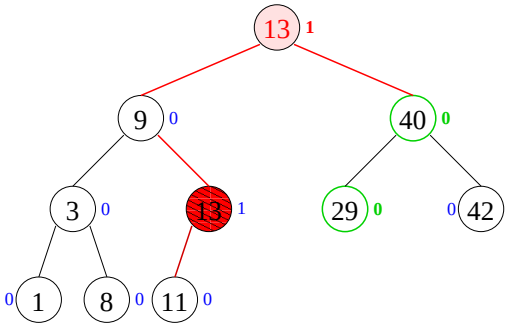
3. Suppression des clés 23, 17, 11 et 1 dans l'arbre de la figure **??**.



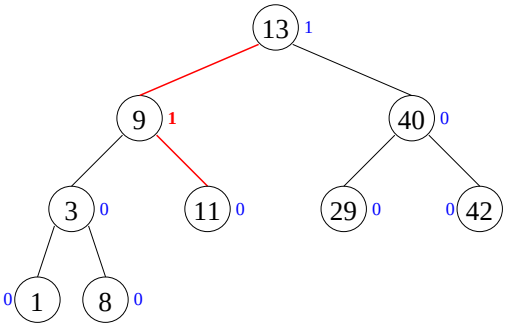
suppression 23



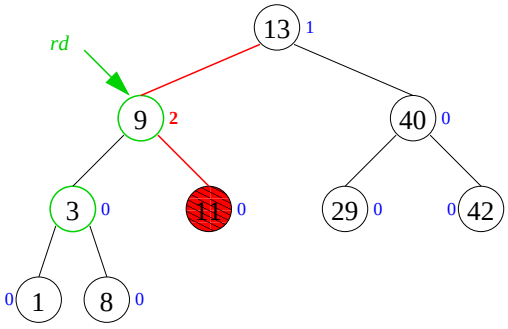
rotation gauche (29)



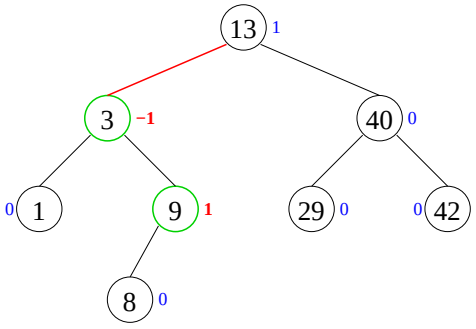
suppression 17
remplacé par 13



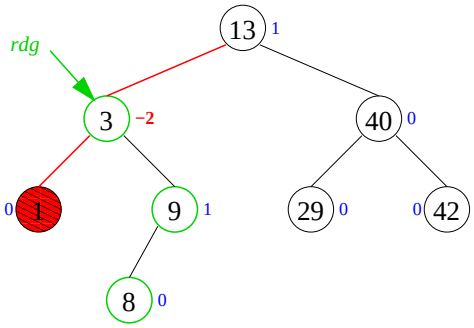
11 remonte



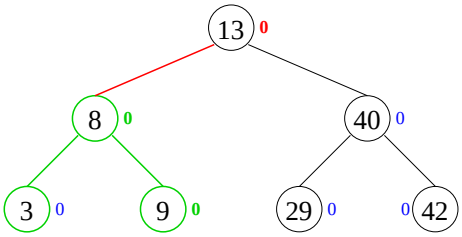
suppression 11



rotation droite (9)



suppression 1



rotation droite-gauche sur 3

4. *La fonction :*

   La version non optimisée...

```python
def maxBST(B):
    while B.right != None:
        B = B.right
    return B.key

def __deleteAVL(x, A):
    if A == None:
        return (None, False)
    elif x == A.key:
        if A.left != None and A.right != None:
            A.key = maxBST(A.left)
            x = A.key
        else:
            if A.left == None:
                return(A.right, True)
            else:
                return(A.left, True)
    if x <= A.key:
        (A.left, dh) = __deleteAVL(x, A.left)
        if not dh:
            return (A, False)
        else:
            A.bal -= 1
            if A.bal == -2:
                if A.right.bal == 1:
                    A = rlr(A)
                else:
                    A = lr(A)
            return (A, A.bal == 0)
    else:   # x > A.key
        (A.right, dh) = __deleteAVL(x, A.right)
        if not dh:
            return (A, False)
        else:
            A.bal += 1
            if A.bal == 2:
                if A.right.bal == -1:
                    A = lrr(A)
                else:
                    A = rr(A)
            return (A, A.bal == 0)

def deleteAVL(x, A):
    (A, dh) = __deleteAVL(x, A)
    return A
```

   L'algorithme ci-dessus peut être optimisé.

   Lorsque l'on remplace la clé courante (la valeur recherchée dans le cas d'un point double) :

   — Il est inutile de tester tous les cas : on peut choisir de quel coté descendre afin d'éviter une rotation à ce niveau.

   — Il serait plus optimal de supprimer la valeur recherchée directement dans les fonction `max_avl` et `min_avl`, et donc d'y intégrer les mises à jour des déséquilibres et rééquilibrages éventuels.