Height-balanced tutorial:AVL
Correction

bal(D)	bal(F)	rotation	ΔH
-2	-1	lr	-1
	0		0
	1	rlr	-1

bal(D)	bal(B)	rotation	ΔH
2	-1	lrr	-1
	0	rr	0
	1		-1

Table 1: Rotations and height changes

***Solution 3.1* (Insertion)**

We write here a recursive version of the element insertion in an AVL. It will be based on the principle of insertion at the leaf in binary search trees, with the rebalancing "in going up".

What information tells us the balance factor has to de updated?
W>e have to know if the tree we come from has changed height.

Why do we add or remove 1 to the balance factor?
If the height of the tree we come from has increased: we add 1 for the left subtree, we remove 1 for the right one.

1. Height change?

   (a) This study allows us to know in which cases the tree height changes after an insertion. It is important to start by the possible cases before the insertion (otherwise, there is a good chance we will have impossible cases)

   We considere an insertion in the left subtree, which has increased the height of this subtree.

   Before insertion, 3 possibilities for the current node:

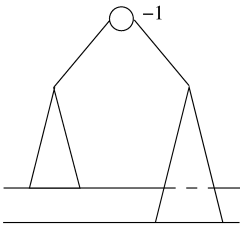


Figure 1: Balance factor: -1

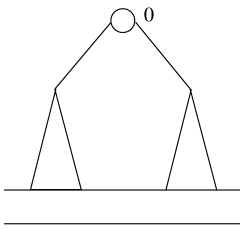


Figure 2: Balance factor: 0

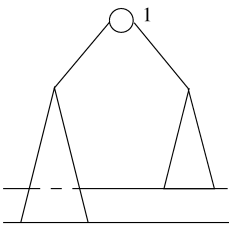
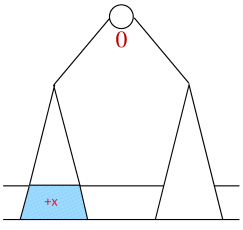
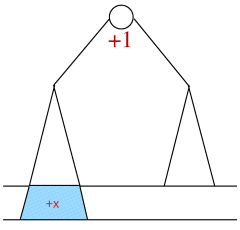


Figure 3: Balance factor: 1

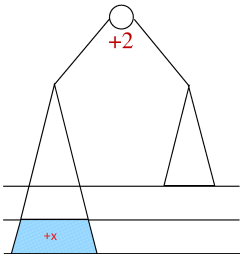
After insertion:



$\Delta H = 0$

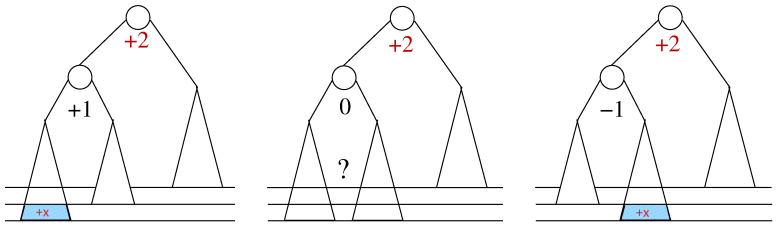


$\Delta H = 1$



Detailed study

Case study where the new balance factor is equal to +2, there are *a priori* 3 possibilities for the left child



after rr
$\Delta H = 0$

Impossible case

after lrr
$\Delta H = 0$

(b) **Conclusion**

Let:

- h_i be the height before insertion
- h_f be the height after insertion
- h_r be the height after rotation

If the new balance factor is:

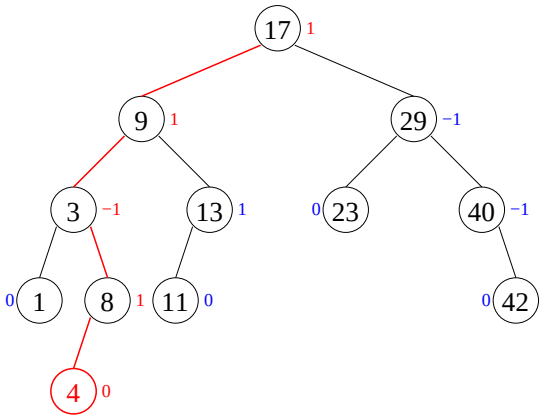
0 : $h_f = h_i$.
1 : $h_f = h_i + 1$.
2 : According to exercise 1.3, if the left child balance factor is:
1 : $h_r = h_f - 1$, but $h_f = h_i + 1$ then $h_r = h_i$
0 : impossible case after insertion
-1 : $h_r = h_f - 1$, but $h_f = h_i + 1$ then $h_r = h_i$
In any case, after a rotation, the tree recovers its initial height.

2. *Principle of the insertion in an AVL:*

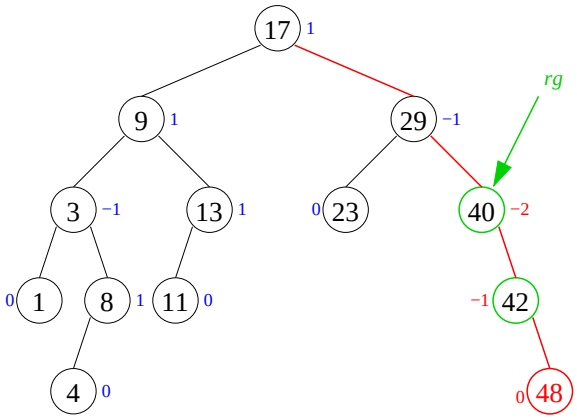
- Search for the insertion position. If the key is not already presented:
  - Creation of the new node
  - Height change equals 1
- Going up if the height of the subtree (where the insertion occured) has changed then the current node balance factor is modified (+1 if back from left, -1 otherwise). The new value is:

0 : no height variation
+1 or -1 : height has changed (+1)
+2 : if the left child balance factor is -1 then LRR else RR, no height variation
-2 : if the right child balance factor is +1 then RLR else LR, no height variation.

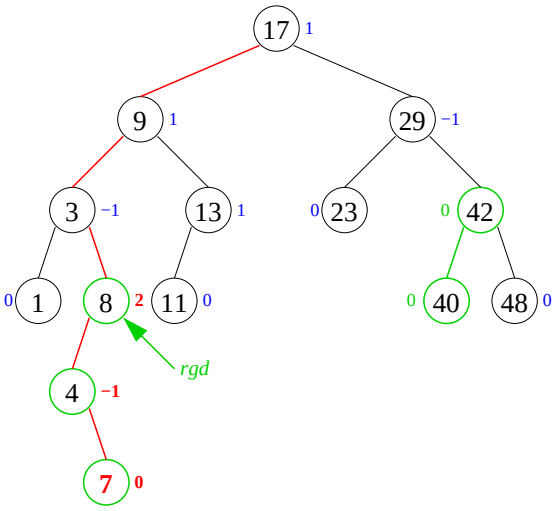
3. Insertions of the keys 4, 48, 7 et 5 in the tree from the subject.



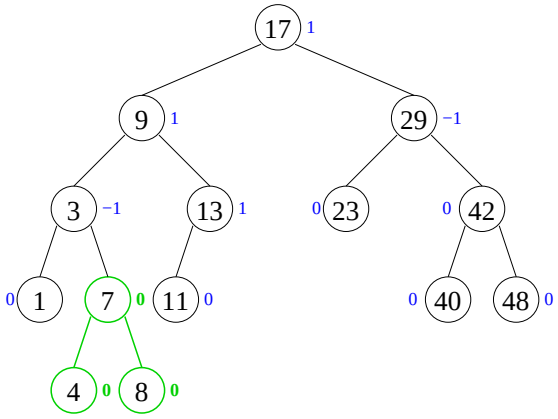
insertion 4



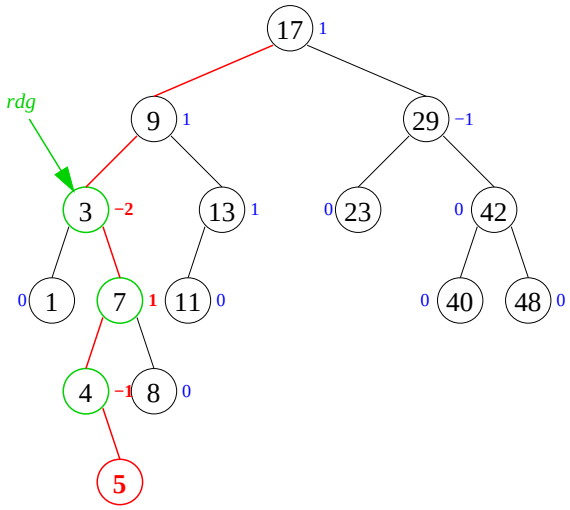
insertion 48



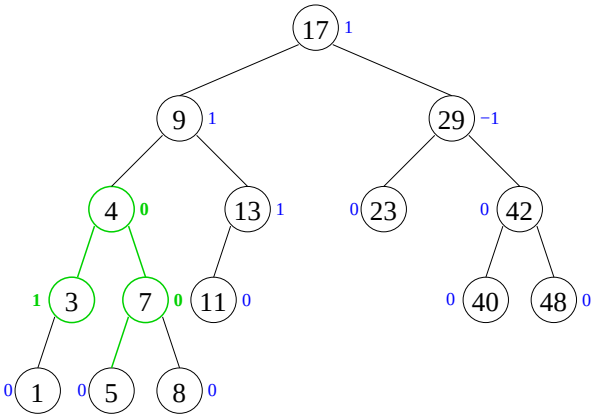
left rotation (rg 40) then insertion 7



left-right rotation (rgd 8)



insertion 5



right-left rotation (rdg 3)

4. *The function:*

   Specifications: The function `insertAVL(x, A)` inserts x in the AVL A (unless x is already in A) and returns the new tree. (The recursive function returns also a boolean that indicates the height change.)

   See the version without "shortcuts" in `AVL_classics.py`

```python
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)  # shortcut!
    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) # shortcut!

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

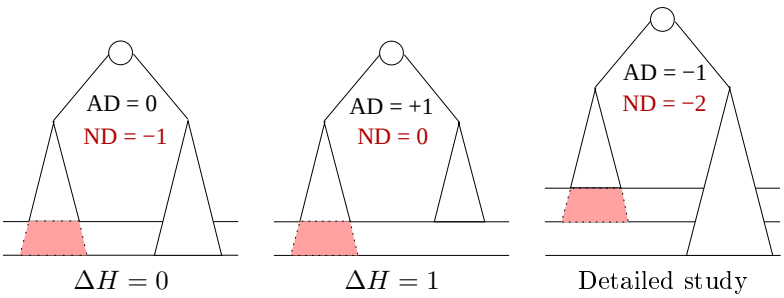
Solution 3.2 (Deletion)

The deletion will be done on the same model as the insertion:

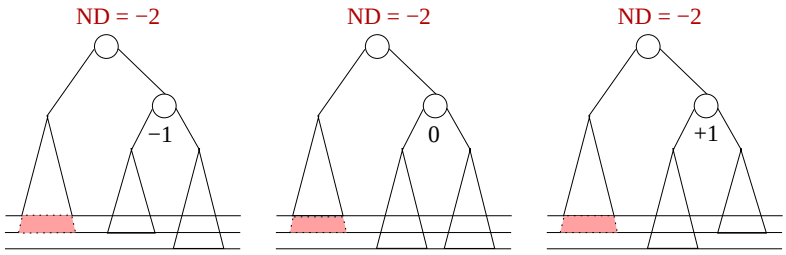
- The deletion itself will be done as on the same model as the one seen in tutorial for binary search trees.
- Rebalacing will be made in going up.

1. We considere a deletion in the left subtree, which has decreased the height of this subtree.

   Let: **AD**, be the old balance factor (ancien déséquilibre), **ND**, be the new balance factor (nouveau déséquilibre).



Case study where the new balance factor (ND) is equal to -2.



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

Conclusion:

- h_i height before deletion
- h_f height after deletion
- h_r height after rotation

If the new balance factor is:

0 : $h_f = h_i - 1$.

-1 : $h_f = h_i$.

-2 : According to exercise 1.3, if the right child balance factor is:

-1 : $h_r = h_f - 1$, **and** $h_f = h_i$ then $h_r = h_i - 1$.

0 : $h_r = h_f$, **and** $h_f = h_i$ then $h_r = h_i$.

1 : $h_r = h_f - 1$, **and** $h_f = h_i$ then $h_r = h_i - 1$.

2. *Principle of deletion in an AVL:*

We consider here the tree contains only distinct values.

- Search for the element to delete.
  - If the element is in a leaf, then deletion of this node. The height decreases.
  - If the element is in a single node, then deletion of this node: replaced by its child. The height decreases.
  - If the element is in a double node then replace the key by the maximum of the left child or the minimum of the right one, depending on the balance factor. Then call again the deletion of the pulled up key in the tree where it was taken.
- Going up if the height of the subtree (where the deletion occurred) has changed then the current node balance factor is modified (-1 if back from left, +1 otherwise). If its new value is:

  0 : height has changed (-1)

  1 ou -1 : no height variation

  2 : if the left child balance factor is:

  -1 : RGD, height has changed (-1)

  0 : RD, no height variation

  1 : RD, height has changed (-1)

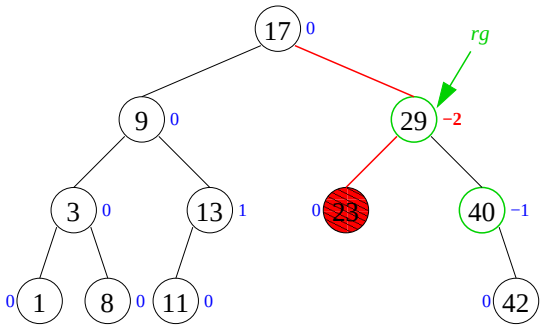
  -2 : if the right child balance factor is:

  -1 : RG, height has changed (-1)

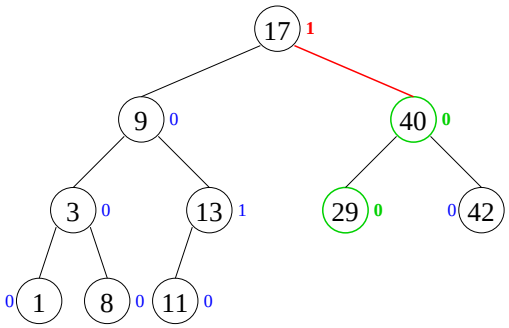
  0 : RG, no height variation

  1 : RDG, height has changed (-1)

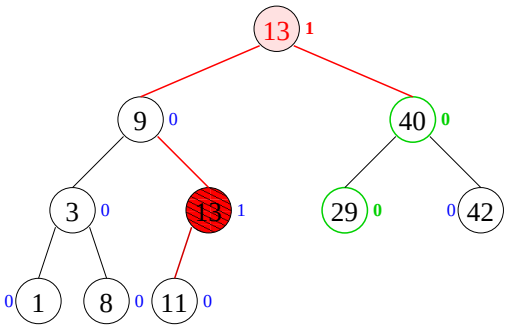
3. Deletions of the keys 23, 17, 11 and 1 in the tree from the subject.



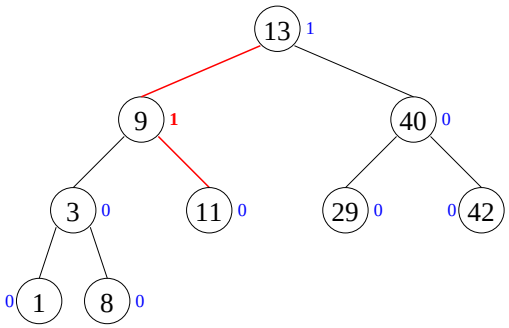
23 deletion



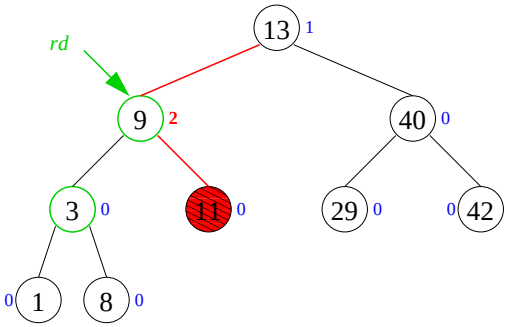
left rotation (rg 29)



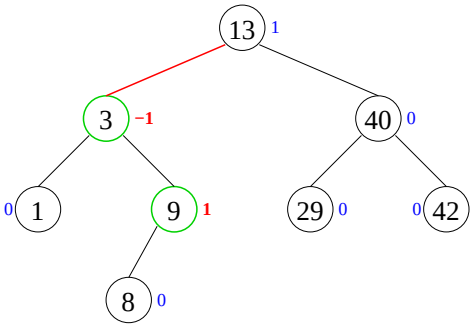
17 deletion
replaced by 13



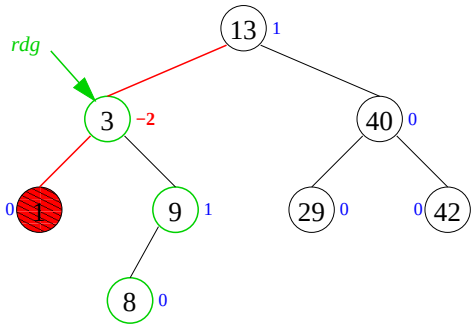
11 goes up



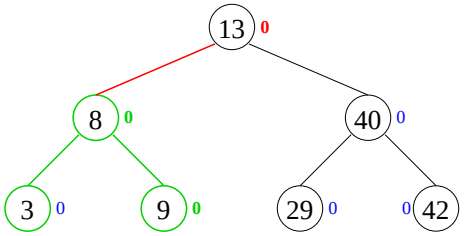
11 deletion



right rotation (rd 9)



1 deletion



right-left rotation (rdg 3)

4. *The function:*

   Not optimized version...

```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     # to use the case x <= A.key below
        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
```

   The above algorithm can be optimized.

   When replacing the current key (the searched value in the case of a double node):

   - We can chose which side going down to prevent rotation at this level.
   - It would be better to delete the searched value directly in the functions `max_avl` and `min_avl` and thus incorporate the balance factor updates and possible rebalancing.