Algorithmique
Correction Partiel n° 2 (P2)

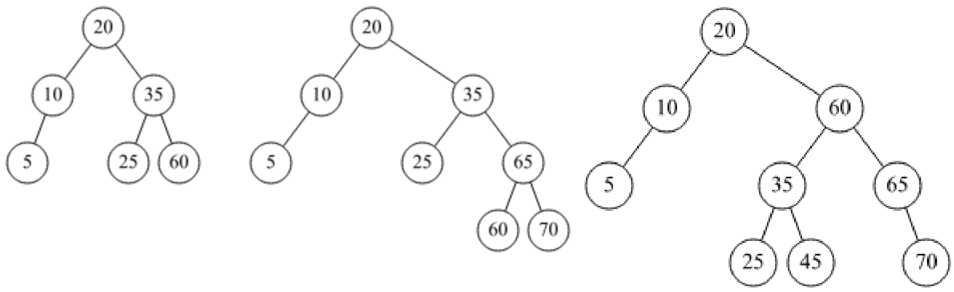
Info-sup S2 – Epita

30 mai 2018 - 14 : 00

Solution 1 **(AVL –** ***3 points*****)**

AVL résultat depuis la liste $[25, 60, 35, 10, 20, 5, 70, 65, 45]$.

AVL final :



Rotations :

```
rlr(25) rdg(25)
lrr(25) rgd(25)
rr(35)  rd(35)
--------------
rlr(60) rdg(60)
--------------
rlr(35) rdg(35)
```

Solution 2 **(Arbres de Léonard –** ***3 points*****)**

1. L'arbre A_5 de Fibonacci est celui de la figure 1 dont les noeuds contiennent leur propre valeur de déséquilibre

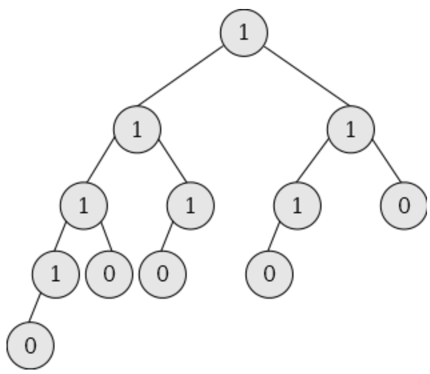


Figure 1 – A_5 de Fibonacci

2. (a) $h_n = n - 1$

   (b) A_0 est réduit à une feuille, donc un arbre h-équilibré.
   La racine de A_1 a pour déséquilibre 1 (une feuille à gauche, rien à droite).
   Pour $n \geq 2$, A_n est un arbre de hauteur $n - 1$. Ses 2 sous-arbres sont A_{n-1} de hauteur $n - 2$ et A_{n-2} de hauteur $n - 3$. Le déséquilibre de la racine de A_n est donc 1 $(n - 2 - (n - 3))$.
   Bref, tous les noeuds internes d'un arbre de Fibonacci ont un déséquilibre de 1 : c'est donc un arbre h-équilibré.

Solution 3 (List → AVL – ***5 points***)

Spécifications :

La fonction `list2avl(L)` retourne un A.-V.L. (class `AVL`) créé à partir de la liste L strictement croissante.

First version :
- Works on $[left, right]$ (as in lecture)
- Recursive function returns the height : to compute balance factors in each node

```python
def __sortedList2AVL(L, left, right):
    """
    L[left, right] -> AVL
    """
    if left > right:
        return (None, -1)
    else:
        mid = left + (right-left) // 2 # or (left + right) // 2
        B = avl.AVL(L[mid], None, None, 0)
        (B.left, hl) = __sortedList2AVL(L, left, mid - 1)
        (B.right, hr) = __sortedList2AVL(L, mid + 1, right)
        B.bal = hl - hr
        return (B, 1 + max(hl, hr))

def sortedList2AVL(L):
    (A, _) = __sortedList2AVL(L, 0, len(L)-1)
    return A
```

Second version :
- Works on $[left, right[$
- Without height computing!

```python
def __sortedList2AVL2(L, left, right):
    """
    L[left, right[ -> AVL
    """
    if left >= right:
        return None
    else:
        mid = left + (right-left) // 2
        A = avl.AVL(L[mid], None, None, 0)
        A.left = __sortedList2AVL2(L, left, mid)
        A.right = __sortedList2AVL2(L, mid + 1, right)
        if right == left + 2:
            A.bal = 1
        return A

def sortedList2AVL2(L):
    return __sortedList2AVL2(L, 0, len(L))
```

***Solution 4* (AVL - Suppression du minimum – *6 points*)**

1. Rotations et changements de hauteur après suppression du minimum :

deseq(racine)	*deseq(fils droit)*	rotation	Δh
-2	-1	rg	1
	0		0
	1	rdg	1

2. **Spécifications :** La fonction `del_min_avl(`A`)` effectue la suppression du nœud contenant la valeur minimale de l'AVL A non vide. Elle retourne un couple : l'arbre modifié et un booléen indiquant si l'arbre a changé de hauteur.

```python
def del_min_avl(A):
    if A.left == None:
        return (A.right, True)
    else:
        (A.left, dh) = del_min_avl(A.left)
        if dh:
            A.bal -= 1
            if A.bal == -2:
                if A.right.bal == +1:
                    A = rlr(A)   # rdg(A)
                else:
                    A = lr(A)    # rg(A)
            return (A, A.bal == 0)
        else:
            return (A, False)

# long version
def del_min_avl2(A):
    if A.left == None:
        return (A.right, True)
    else:
        (A.left, dh) = del_min_avl2(A.left)
        if not dh:
            return (A, False)
        else:
            A.bal -= 1
            if A.bal == 0:
                return (A, True)
            elif A.bal == -1:
                return (A, False)
            else:   # A.bal == -2
                if A.right.bal == -1:
                    A = lr(A)    # rg(A)
                    return (A, True)
                elif A.right.bal == 0:
                    A = lr(A)    # rg(A)
                    return (A, False)
                else:
                    A = rlr(A)   # rdg(A)
                    return (A, True)
```

Solution 5 **(ABR et mystère –** ***4 points*****)**

1. *Résultats retournés ?*
   (a) `call(25,` B`)` : None
   (b) `call(21,` B`)` : 26
   (c) `call(20,` B`)` : 21
   (d) `call(9,` B`)` : 15
   (e) `call(53,` B`)` : None

2. `bst_mystery(x, B)` (`B` ABR quelconque, dont tous les éléments sont distincts).
   À la fin de la partie 1 :

   (a) `B` représente l'arbre de racine x si x présent, il a la valeur None sinon.

   (b) Sur le chemin de recherche de x, `P` est l'arbre dont la racine est le dernier nœud rencontré avant de descendre à gauche (il reste à None si on n'est jamais descendu à gauche).

3. `call(x, B)` : si x est présent dans l'arbre, et n'est pas la plus grand valeur, elle retourne la valeur immédiatement supérieure. Dans les autres cas elle retourne None.
