# Algorithmique
# Correction Partiel n° 2 (P2)

Info-sup (S2) – Epita

*6 juin 2016 - 10 :00*

***Solution 1*** **(Arbres de Léonard – *5 points*)**

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

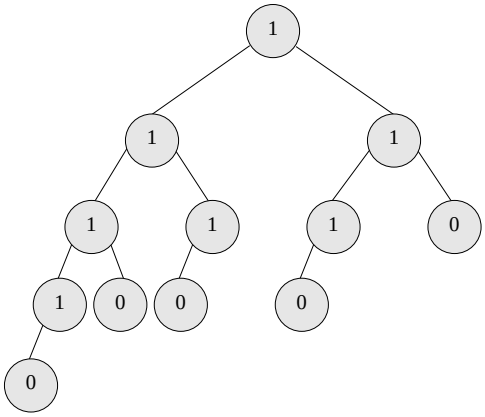


Figure 1 – $A_5$ de Fibonacci.

2. Tableau des valeurs de $H_n$, $T_n$, $F_n$ et $Fib_n$ :

| $n$ | $H\_n$ | $T\_n$ | $F\_n$ | $Fib\_n$ |
|---|---|---|---|---|
| 0 | _ | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 2 | 1 | 1 |
| 3 | 2 | 4 | 2 | 2 |
| 4 | 3 | 7 | 3 | 3 |
| 5 | 4 | 12 | 5 | 5 |
| 6 | 5 | 20 | 8 | 8 |

3. $n \geq 2$ :
   — $H_n = n - 1$ évident par récurrence
   — $T_n = Fib_{n+2} - 1$ par récurrence aussi en notant que $T_n = T_{n-1} + T_{n-2} + 1$
   — $F_n = Fib_n$ Vérifient la même récurrence, donc $F_n = Fib_n = Fib_{n-1} + Fib_{n-2}$

4. $A_0$ 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 2*** **(ABR et mystère – *5 points*)**

1. *Résultat retourné :*
   (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.

---

***Solution 3*** **(La taille en plus – *4 points*)**

```
        def addSize(B):
            if B == None:
                return(None, 0)
            else:
                C = BinTreeSize()
                C.key = B.key
                (C.left, size1) = addSize(B.left)
                (C.right, size2) = addSize(B.right)
                C.size = 1 + size1 + size2
                return (C, C.size)

# another version

        def addSize2(B):
            if B == None:
                return(None, 0)
            else:
                (left, size1) = addSize2(B.left)
                (right, size2) = addSize2(B.right)
                size = 1 + size1 + size2
                return (newBinTreeSize(B.key, left, right, size), size)
```

```
        def copyWithSize(B):
            (C, size) = addSize(B)
            return C
```

***Solution 4* (Médian – *7 points*)**

1. $B$ ABR de $n$ éléments dont le $k^{ème}$ élément ($1 \leq k \leq n$) se trouve en racine :
   — taille(g(B)) $= k - 1$
   — taille(d(B)) $= n - k$

2. *Définition abstraite de l'opération kieme (médian était donné) :*

   **AXIOMES**
   k = *taille*(G)+1 ⇒ *kieme* (<r, G, D>, k) = r
   k ≤ *taille* (G) ⇒ *kieme* (<r, G, D>, k) = *kieme* (G, k)
   k > *taille* (G) +1 ⇒ *kieme* (<r, G, D>, k) = *kieme* (D, k - *taille* (G)-1)

3. **Spécifications :**
   La fonction `nthBST(`$B$`, k)` avec $B$ un ABR non vide et $1 \leq k \leq taille(B)$, retourne l'arbre dont la racine contient le $k^{ème}$ élément de $B$ .

```
        def nthBST(B, k):

            if B.left == None:
                leftSize = 0
            else:
                leftSize = B.left.size

            if leftSize == k - 1:
                return B
            elif k <= leftSize:
                return nthBST(B.left, k)
            else:
                return nthBST(B.right, k - leftSize - 1)


        def nthBST2(B, k):

            if B.left == None:
                if k == 1:
                    return B
                else:
                    return nthBST2(B.right, k - 1)

            else:
                if k == B.left.size + 1:
                    return B
                elif k <= B.left.size:
                    return nthBST2(B.left, k)
                else:
                    return nthBST2(B.right, k - B.left.size - 1)
```

**Spécifications :**
La fonction `median(`$B$`)` retourne la valeur médiane de l'ABR $B$ s'il est non vide, la valeur `None` sinon.

```
        def median(B):
            if B != None:
                return nthBST(B, (B.size+1) // 2).key
            else:
                return None
```