# Algorithmique
# Partiel n° 2 (P2)

Info-sup S2#
Epita

*7 janvier 2020 - 13h-15h*

---

## Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

---

**Exercice 1 (Arbres de Léonard – *3 points*)**

On se propose, pour cet exercice, d'étudier certaines propriétés d'une famille d'arbres binaires, les arbres de Fibonacci. Ceux-ci sont définis récursivement de la manière suivante :

$$\begin{cases} A_0 = \ ArbreVide \\ A_1 = < o, ArbreVide, ArbreVide > \\ A_n = < o, A_{n-1}, A_{n-2} > \ si\ n \geqslant 2 \end{cases}$$

1. Représenter graphiquement l'arbre de Fibonacci $A_5$.
2. (a) Exprimer en fonction de $n \geqslant 2$ la hauteur $h_n$ de l'arbre $A_n$.
   (b) Démontrer que l'arbre $A_n$ est un arbre $h$-équilibré.

**Exercice 2 (Arbres de Léonard, encore – *4 points*)**

Nous reprenons les arbres de Fibonacci définis à l'exercice 1, en ajoutant des étiquettes aux nœuds :

$$\begin{cases} A_0 = \ ArbreVide \\ A_1 = < 1, ArbreVide, ArbreVide > \\ A_n = < fibo(n), A_{n-1}, A_{n-2} > \ si\ n \geqslant 2 \end{cases}$$

Écrire la fonction `leonard_tree(`$n$`)` qui construit l'arbre de Fibonacci $A_n$. Comme indiqués ci-dessus, les nœuds seront étiquetés avec les valeurs de $fibo(n)$ selon la définition suivante :

$$\begin{cases} n \leq 1 \Rightarrow fibo(n) = \ n \\ n > 1 \Rightarrow fibo(n) = \ fibo(n-1) + fibo(n-2) \end{cases}$$

**Exercice 3 (Suppression – *7 points*)**

*Rappel du principe de suppression d'un élément dans un arbre binaire de recherche (ABR) :*

La structure de la fonction est celle de la recherche.
Une fois l'élément trouvé (il est en racine), si le nœud contenant l'élément est :
- une feuille, on la supprime directement : le résultat est l'arbre vide.
- un point simple, on supprime le nœud : on le remplace par son fils unique.
- un point double, on remplace la clé en racine par le maximum de son sous-arbre gauche, et on relance la suppression de cette valeur à gauche.

1. Écrire la fonction `maxBST(`$B$`)` qui retourne la valeur maximale de l'ABR $B$ non vide.
2. Écrire la fonction récursive `delBST(`$B$`,` $x$`)` qui supprime la valeur $x$ (la première trouvée) dans l'ABR $B$ et retourne l'arbre résultat

**Exercice 4 (Construction – *3 points*)**

À partir d'un arbre vide, construire l'AVL en insérant successivement les valeurs 25, 60, 35, 10, 20, 5, 70, 65.

Vous dessinerez l'arbre à deux étapes :
- après l'insertion de 20 ;
- l'arbre final.

Indiquez quelles rotations ont été effectuées, dans l'ordre (par exemple si une rotation gauche a été effectuée sur l'arbre de racine 42, indiquer *rg(42)*).

**Exercice 5 (What is this ? – *3 points*)**

Soient les fonctions suivantes :

```
        def __test(B):
            if B == None:
                return (-1, True)
            else:
                (hl, tl) = __test(B.left)
                (hr, tr) = __test(B.right)
                return (1 + max(hl, hr), tl and tr and abs(hl-hr) < 2)

        def test(B):
            (x, res) = __test(B)
            return res
```

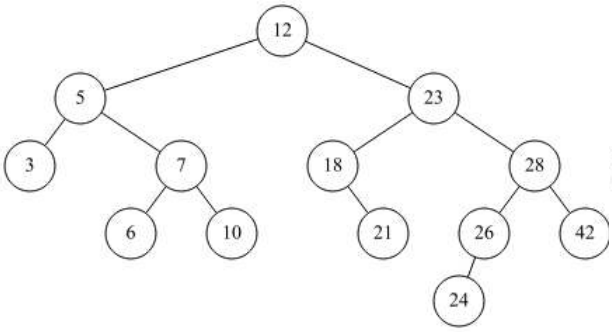


FIGURE 1 – Arbre $B_1$

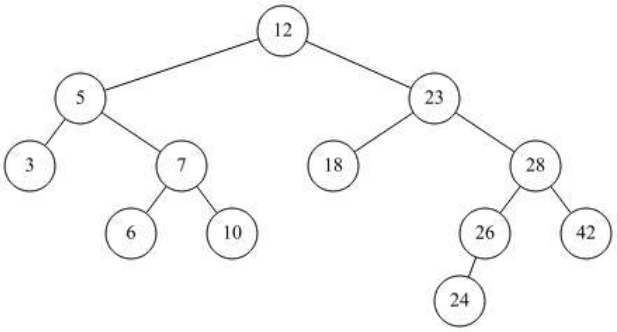


FIGURE 2 – Arbre $B_2$

1. Pour chacun des arbres ci-dessus, quel est le résultat retourné par `test(`$B_i$`)` ?
2. Que fait la fonction `test(`$B$`)` ?
3. Cette fonction peut être optimisée. Comment ?

# Annexes

## Les arbres binaires

```
class BinTree:
    def __init__(self, key, left, right):
        self.key = key
        self.left = left
        self.right = right
```

## Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.