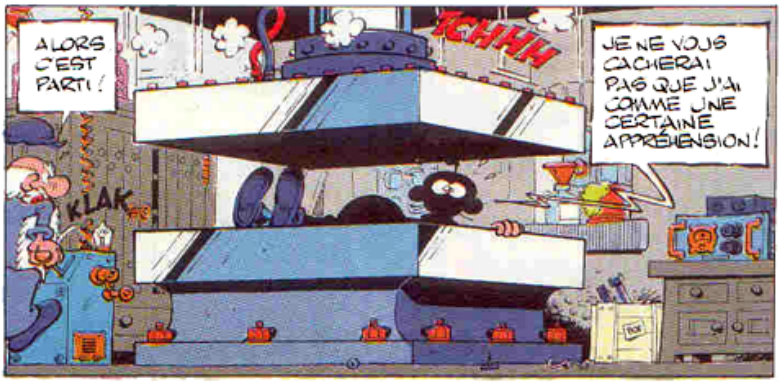
Algorithmique
Partiel n° 2 (P2)

Info-sup S2
Epita

30 mai 2018 - 14 : 00

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 (AVL – *3 points*)

À partir d'un arbre vide, construire l'AVL en insérant successivement les valeurs 25, 60, 35, 10, 20, 5, 70, 65, 45.

— Vous ne dessinerez que 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 2 (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 3 (`List` → `AVL` – *5 points*)

A partir d'une liste d'éléments triés en ordre strictement croissant, on désire créer un *arbre binaire de recherche équilibré* (A.-V.L.). Par exemple, à partir de la liste triée suivante, on veut obtenir un des deux arbres de la figure 1.

0	1	2	3	4	5	6	7	8	9	10	11
1	4	5	7	10	12	13	15	18	20	21	25

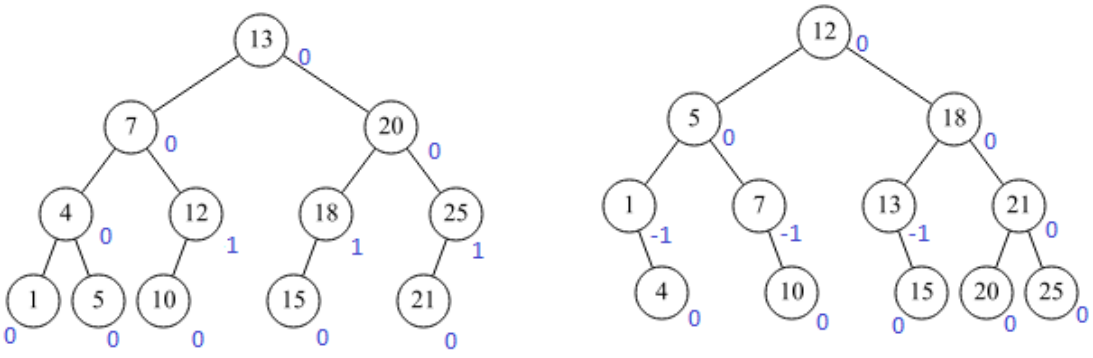


FIGURE 1 – A.-V.L.

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

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

Nous nous intéressons ici à la suppression du minimum dans un AVL avec rééquilibrage.

1. Compléter le tableau donné afin qu'il indique pour chaque cas de déséquilibre (*deseq*), uniquement après la suppression du minimum, quelle est la rotation à effectuer ainsi que le changement éventuel de hauteur induit ($\Delta h = 0$ si l'arbre ne change pas de hauteur après rotation, 1 sinon).

2. Écrire la fonction récursive qui supprime la valeur minimum d'un AVL non vide (avec mises à jour des déséquilibres et éventuelles rotations à la remontée). La fonction retourne l'arbre après suppression et un booléen indiquant si l'arbre a changé de hauteur (un couple).
   Vous pouvez utiliser les fonctions qui effectuent les rotations avec mises à jour des déséquilibres (`rg, rd, rgd, rdg`, voir annexes).

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

```
1  def bstMystery(x, B):
2
3      # first part
4      P = None
5      while B != None and x != B.key:
6          if x < B.key:
7              P = B
8              B = B.left
9          else:
10             B = B.right
11     if B == None:
12         return None
13
14     # second part
15     if B.right == None:
16         return P
17     else:
18         B = B.right
19         while B.left != None:
20             B = B.left
21         return B
```

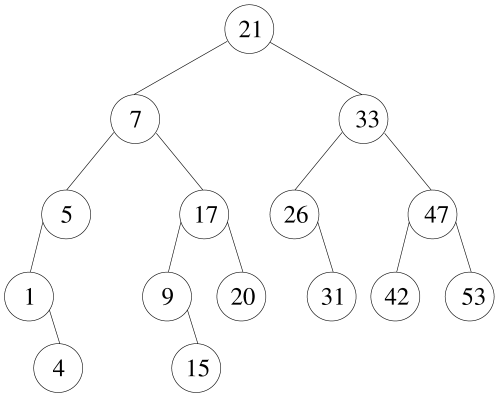


FIGURE 2 – arbre B_1

```
1  def call(x, B):
2      p = bstMystery(x, B)
3      if p == None:
4          return None
5      else:
6          return p.key
```

1. Pour chacun des appels suivants, avec B_1 l'arbre de la figure ci-dessus, quel est le résultat retourné ?
   (a) `call(25,` B_1`)`
   (b) `call(21,` B_1`)`
   (c) `call(20,` B_1`)`
   (d) `call(9,` B_1`)`
   (e) `call(53,` B_1`)`

2. On appelle `bstMystery(x, B)` avec `B` un arbre binaire de recherche quelconque, dont tous les éléments sont distincts.
   Pendant l'exécution, à la fin de la partie 1 :
   (a) Que représente `B` ?
   (b) Que représente `P` ?

3. Que fait la fonction `call(x, B)` ?

Annexes

Les arbres binaires

Les arbres binaires "classiques" :

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

Les AVL, avec les déséquilibres :

Rappel : dans un A-V.L. les clés sont toutes distinctes.

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

Fonctions et méthodes autorisées

Rotations (A:`AVL`) : chaque fonction ci-dessous retourne l'arbre A après rotation et mise à jour des déséquilibres.

- `rg`(A) : rotation gauche
- `rd`(A) : rotation droite
- `rgd`(A) : rotation gauche-droite
- `rdg`(A) : rotation droite-gauche

Sur les listes :

- `len`
- `append`

Autres :

- `abs`
- `min` et `max`, mais uniquement avec deux valeurs entières !

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.