Algorithmique
Partiel n° 2 (P2)

Info-sup S2#
Epita

8 janvier 2018 - 11 : 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 (Combien ? – *3,5 points*)

Écrire la fonction `nb_inter(B, a, b)` qui calcule le nombre de valeurs de l'arbre binaire de recherche B dans l'intervalle $[a, b[$.

Exemples d'applications, avec `B1` l'arbre ci-dessous :

```
>>> nb_inter(B1, 0, 27)
12
>>> nb_inter(B1, 6, 17)
5
```

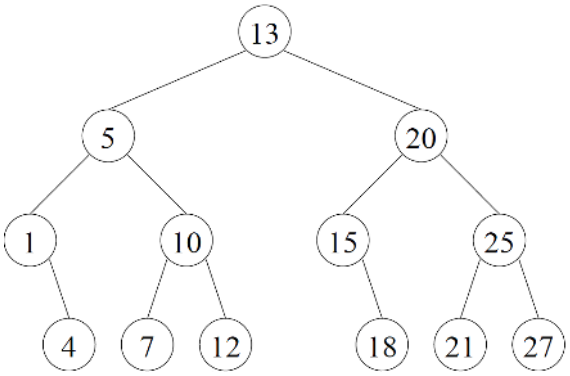

FIGURE 1 – ABR `B1`

Exercice 2 (ABR → AVL – *4,5 points*)

Écrire une fonction qui construit à partir d'un arbre binaire classique (`BinTree`) un arbre équivalent au premier (contenant les mêmes valeurs aux mêmes places) mais avec le déséquilibre (le "champ" *bal*) renseigné en chaque nœud (`AVL`).

Indice : Vous pourriez avoir besoin d'une fonction récursive qui retournera la copie de l'arbre mais également sa hauteur.

Exercice 3 (AVL - Ajout 0 – *5 points*)

Nous travaillons ici avec des AVL toutes les valeurs sont des entiers naturels non nuls.

Écrire la fonction récursive `add0(`A`)` qui insère la valeur 0 dans l'AVL A (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 4 (AVL – *3 points*)

À partir d'un arbre vide, construire l'AVL en insérant successivement les valeurs 13, 20, 5, 1, 15, 10, 18, 25, 4, 21, 7, 12, 23. Vous dessinerez l'arbre à trois étapes :
- après l'insertion de 18 ;
- après l'insertion de 21 ;
- l'arbre final.

Exercice 5 (Arbre 2.3.4 → Arbre bicolore – *2 points*)

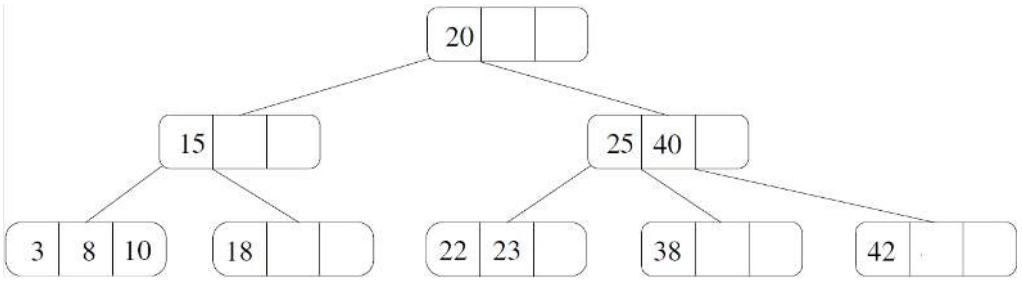

FIGURE 2 – Arbre 2.3.4 à transformer

1. Dessiner l'arbre bicolore correspondant à l'arbre 2.3.4 de la figure 2. Les 3-nœuds devront être représentés penchés à gauche.
2. L'arbre obtenu est-il un AVL ? Justifiez (avec précision et concision) votre réponse.

Exercice 6 (Arbres et mystère – *4 points*)

```
1             def __makeTree(n, i, cur):
2                 if i > n:
3                     return (None, cur)
4                 else:
5                     (left, cur) = __makeTree(n, 2*i, cur)
6                     key = cur+1
7                     (right, cur) = __makeTree(n, 2*i+1, key)
8                     return (binTree.BinTree(key, left, right), cur)
9
10            def makeTree(n):
11                (B, val) = __makeTree(n, 1, 0)
12                return B
```

1. La fonction `makeTree(n)` construit (et retourne) un arbre binaire. Dessiner l'arbre résultat lorsque $n = 13$.
   Si vous ne voyez pas quelles valeurs mettre en clés, dessinez tout de même la structure (l'arbre sans valeurs).
2. On appelle `makeTree(n)` avec `n` un entier strictement positif.
   Donner deux propriétés de l'arbre retourné.

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

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.