Algorithmique
Partiel n° 2 (P2)

Info-sup S2#
Epita

8 Jan. 2018 - 9 : 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

Des ABR avec taille

Pour les exercices qui suivent, on ajoute une nouvelle implémentation des arbres binaires dans laquelle chaque nœud contient la taille (`size`!) de l'arbre dont il est racine.

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

Exercice 1 (La taille en plus – *4 points*)

Écrire la fonction `copyWithSize(`B`)` qui prend en paramètre un arbre binaire B "classique" (`BinTree` sans la taille) et qui retourne un autre arbre binaire, équivalent au premier (contenant les mêmes valeurs aux mêmes places) mais avec la taille renseignée en chaque nœud (`BinTreeSize`).

Exercice 2 (Ajout avec mise à jour de la taille – *3 points*)

Écrire une fonction **récursive** qui ajoute un élément en feuille dans un arbre binaire de recherche (qu'il soit présent ou non).
L'arbre est représenté par le type `BinTreeSize`, il faut donc mettre à jour, lorsque nécessaire, le champ *size* en chaque nœud de l'arbre.

Exercice 3 (Médian – *6 points*)

On s'intéresse à la recherche de la valeur médiane d'un arbre binaire de recherche B, c'est à dire celle qui, dans la liste des éléments en ordre croissant, se trouve à la place $(taille(B) + 1)\ DIV\ 2$.

Pour cela, on veut écrire une fonction `nthBST(`B`, `k`)` qui retourne le nœud contenant le $k^{ème}$ élément de l'ABR B (dans l'ordre des éléments croissants). Par exemple, l'appel à `nthBST(`B_1`, 3)` avec B_1 l'arbre de la figure 1 retournera le nœud contenant la valeur 5 et l'appel `nthBST(`B_1`, 9)` retournera le nœud de racine 18.

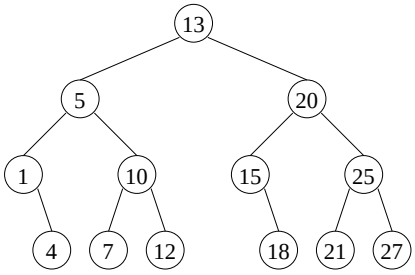

FIGURE 1 – ABR B_1

1. **Un peu d'aide :** Soit B un arbre binaire de recherche contenant n éléments. Si le $k^{ème}$ élément (avec $1 \leq k \leq n$) se trouve en racine, combien d'éléments contiennent les sous-arbres de B ?
2. **Implémentation :** Les fonctions à écrire utilisent des arbres binaires avec la taille renseignée en chaque nœud (`BinTreeSize()`).
   — Écrire la fonction `nthBST(`B`, `k`)` qui retourne l'arbre contenant le $k^{ème}$ élément en racine. On supposera que cet élément existe toujours : $1 \leq k \leq taille(B)$.
   — Écrire la fonction `median(`B`)` qui retourne la valeur médiane de l'arbre binaire de recherche B s'il est non vide, la valeur `None` sinon.

Des arbres équilibrés

Exercice 4 (What is this ? – *3 points*)

```
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
```

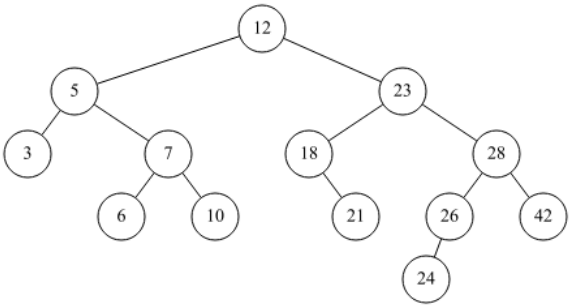

FIGURE 2 – Arbre B_2

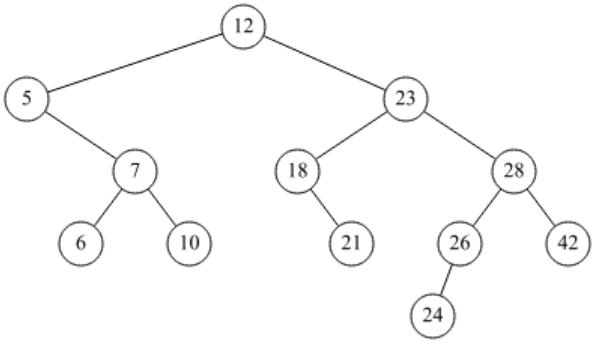

FIGURE 3 – Arbre B_3

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

Exercice 5 (AVL – *2 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.

Exercice 6 (Arbres 2-3-4 – *3 points*)

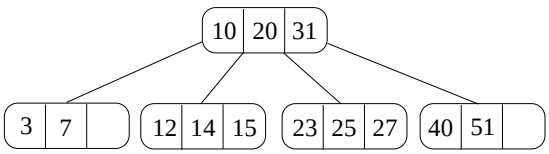

FIGURE 4 – Arbre 2.3.4.

1. Insérer successivement les clés 4, 11, 9 et 18 sur l'arbre de la figure 4 avec **éclatements à la descente**. *(Dessiner les trois arbres 2.3.4. (après chaque ajout) ainsi que l'arbre final).*
2. Représenter l'arbre de la figure 4 en bicolore. Vous considérerez les 3-noeuds penchés à droite.

Annexes

Les arbres binaires

Les arbres binaires "classiques" :

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

Fonctions et méthodes autorisées

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.