

Algorithmique

Partiel n° 2 (P2)

INFO-SUP (S2)
EPITA

6 juin 2016 - 10 : 00 (D.S. 307430.1 BW)

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** !
 - Vos fonctions doivent impérativement respecter les exemples d'applications donnés.
 - ☐ Durée : 2h00
-



Exercice 1 (Arbres de Léonard – 5 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 :

$$\begin{cases} A_0 = \text{ArbreVide} \\ A_1 = \langle o, \text{ArbreVide}, \text{ArbreVide} \rangle \\ A_n = \langle o, A_{n-1}, A_{n-2} \rangle \text{ si } n \geq 2 \end{cases}$$

1. Représenter graphiquement l'arbre de Fibonacci A_5 .
2. Donner, sous la forme d'un tableau, pour chaque arbre A_n avec $0 \leq n \leq 6$ les valeurs de la hauteur H_n de la taille T_n , du nombre de feuilles F_n et du nombre de Fibonacci Fib_n (On considérera $Fib_0 = 0$ et $Fib_1 = 1$).
3. Exprimer en fonction de $n \geq 2$, et éventuellement de Fib_n : la hauteur H_n , la taille T_n et le nombre de feuilles F_n de l'arbre A_n .
4. Démontrer que l'arbre A_n est un arbre h -équilibré.

Exercice 2 (ABR et mystère – 5 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

```

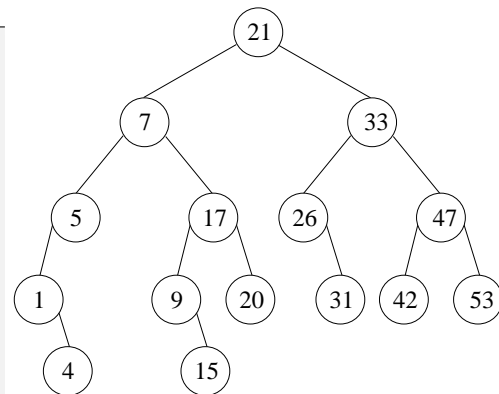


FIGURE 1 – arbre B_1

```

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 1, 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)` ?

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

```
1  class BinTreeSize :
2      """
3      """
4  def newBinTreeSize(key, left, right, size):
5      B = BinTreeSize()
6      B.key = key
7      B.left = left
8      B.right = right
9      B.size = size    # size of the tree
10     return B
```

Exercice 3 (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 4 (Médian – 7 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 $(taille(B) + 1) \text{ DIV } 2$.

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

1. De l'aide pour la suite :

Soit B un arbre binaire de recherche contenant n éléments. Si le $k^{\text{ème}}$ élément (avec $1 \leq k \leq n$) se trouve en racine, combien d'éléments contiennent les sous-arbres de B ?

2. Étude abstraite :

On ajoute à la définition abstraite des arbres binaires (donnée en annexe) l'opération *taille*, définie comme suit :

OPÉRATIONS

taille : ArbreBinaire \rightarrow Entier

AXIOMES

taille (arbrevide) = 0

taille (<0, G, D>) = 1 + *taille* (G) + *taille* (D)

Donner une définition abstraite de l'opération *kieme* (utilisant obligatoirement l'opération *taille*) : compléter les définitions abstraites données.

3. 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 $k^{\text{ème}}$ 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.

Annexes

Type abstrait arbre binaire

SORTE

ArbreBinaire

UTILISE

Nœud, Élément

OPÉRATIONS

arbrevide : \rightarrow ArbreBinaire

$\langle -, -, - \rangle$: $\text{Nœud} \times \text{ArbreBinaire} \times \text{ArbreBinaire} \rightarrow \text{ArbreBinaire}$

racine : $\text{ArbreBinaire} \rightarrow \text{Nœud}$

g : $\text{ArbreBinaire} \rightarrow \text{ArbreBinaire}$

d : $\text{ArbreBinaire} \rightarrow \text{ArbreBinaire}$

contenu : $\text{Nœud} \rightarrow \text{Élément}$

PRÉCONDITIONS

racine(*B*) **est-défini-ssi** $B \neq \text{arbrevide}$

g(*B*) **est-défini-ssi** $B \neq \text{arbrevide}$

d(*B*) **est-défini-ssi** $B \neq \text{arbrevide}$

AXIOMES

racine($\langle o, G, D \rangle$) = *o*

g($\langle o, G, D \rangle$) = *G*

d($\langle o, G, D \rangle$) = *D*

AVEC

G, D : ArbreBinaire

o : Nœud