Algorithmique
Contrôle nº 3 (C3)

Info-spé S3#
Epita

6 mars 2018 - 14 : 15

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 (classes, fonctions, méthodes) est indiqué dans l'énoncé !
  - 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.
- Durée : 2h00

Du hachage

Exercice 1 (Hachages – *2 points*)

Supposons l'ensemble de clés suivant E={beck, cale, clapton, hendrix, hooker, king, richards, vaughan, winter, young} ainsi que le tableau 1 des valeurs de hachage associées à chaque clé de cet ensemble E. Ces valeurs sont comprises entre 0 et 10 ($m = 11$).

beck	10
cale	10
clapton	4
hendrix	5
hooker	6
king	8
richards	3
vaughan	8
winter	1
young	5

TABLE 1 – Valeurs de hachage

Représenter (dessiner) les structures de données et la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans le cas du hachage avec chaînage séparé.

Exercice 2 (Hachage : Tableaux valides – *3 points*)

Supposons les clés de A à G avec les valeurs de hachage données dans la table 2.

clés	A	B	C	D	E	F	G
h(clés) m=7	2	0	0	4	4	4	2

TABLE 2 – Valeurs de hachage

Si celles-ci sont insérées dans un ordre quelconque, selon le principe du hachage linéaire (avec $d = 1$), dans un tableau initialement vide de taille 7, quels tableaux parmi les suivants ne peuvent pas résulter de l'insertion de ces clés ?

	0	1	2	3	4	5	6
Tableau (A)	C	G	B	A	D	E	F
Tableau (B)	F	G	B	D	A	C	E
Tableau (C)	B	C	A	G	E	D	F
Tableau (D)	G	E	C	A	D	B	F

TABLE 3 – Tableaux possibles ?

Des arbres

Exercice 3 (Arité moyenne d'un arbre général – *5 points*)

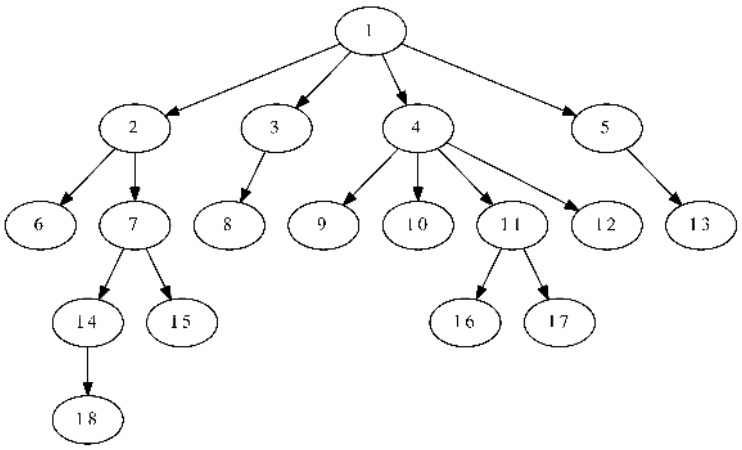

FIGURE 1 – Arbre général

On va s'intéresser à l'arité (nombre de fils d'un nœud) moyenne dans un arbre général. On définit l'arité moyenne comme la somme des nombres de fils par nœud divisée par le nombre de nœuds *internes*.

Par exemple, pour l'arbre de la figure 1, il y a 8 nœuds internes, et lorsque l'on fait la somme des nombres de fils par nœud, on obtient 17 (compter les flèches pour vérifier), l'arité moyenne est donc de $17/8 = 2,125$.

Écrire la fonction `averageArity(`B`)` qui calcule l'arité moyenne de l'arbre général T, avec l'implémentation *premier fils - frère droit*.

Exercice 4 (B-arbres : Représentation linéaire – *5 points*)

Rappel : Un arbre général $A =< o, A_1, A_2, ..., A_n >$ peut être représenté par $(o\ A_1\ A_2\ ...\ A_n)$.

Pour les B-arbres, un nœud o est représenté par la liste de ses clés : $< x_1, ..., x_{k-1} >$.

Par exemple, l'arbre de la figure 2 sera représenté par la chaîne :

$$(< 22 > (< 15 > (< 8, 12 >)(< 18, 19, 20 >))(< 27, 41 > (< 24, 25 >)(< 30, 35, 38 >)(< 45, 48 >)))$$

Écrire la fonction qui construit à partir d'un B-arbre sa représentation linéaire (sous forme de chaîne de caractères).

Exercice 5 (B-Arbres : insertions – *2 points*)

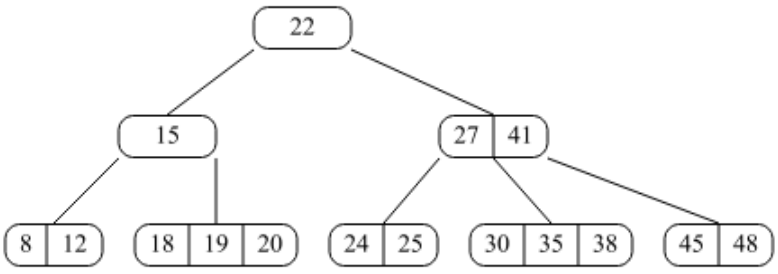

FIGURE 2 – B-arbre d'ordre 2

Insérer **successivement** les clés 36 et 42 dans l'arbre de la figure 2 en appliquant le principe de précaution. Dessiner **uniquement** l'arbre obtenu après chaque insertion.

Exercice 6 (What ? – *3 points*)

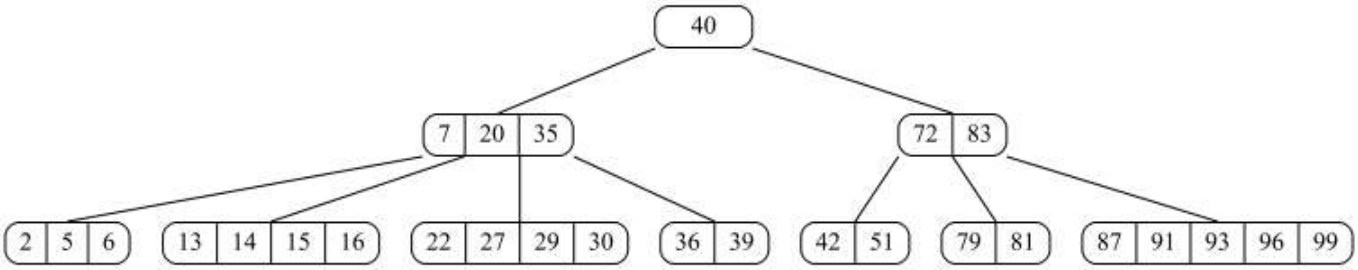

FIGURE 3 – B-tree d'ordre 3

Soit la fonction `what` définie ci-dessous :

```
1   def __what(B, x, y):
2       i = 0
3       while i < B.nbkeys and B.keys[i] <= x:
4           i += 1
5       if i < B.nbkeys:
6           y = B.keys[i]
7       if B.children == []:
8           return y
9       else:
10          return __what(B.children[i], x, y)
11
12  def what(B, x):
13      if B == None:
14          return None
15      else:
16          return __what(B, x, None)
```

Soit B_3 l'arbre de la figure 3. Quel sera le résultat de chacun des appels suivants ?

- `what(`B_3`, 2)`
- `what(`B_3`, 7)`
- `what(`B_3`, 18)`
- `what(`B_3`, 39)`
- `what(`B_3`, 41)`
- `what(`B_3`, 99)`

Annexes

Les arbres généraux

Les arbres (généraux) manipulés ici sont les mêmes qu'en td.

Implémentation *premier fils - frère droit*

Un arbre est un objet de la class `TreeAsBin` :

```
class TreeAsBin:
    def __init__(self, key, child=None, sibling=None):
        self.key = key
        self.child = child
        self.sibling = sibling
```

B-Trees

Les B-arbres manipulés ici sont les mêmes qu'en td.

Un arbre vide est `None`, un arbre non vide est un objet de la classe `BTree` :

```
class BTree:
    degree = None

    def __init__(self, keys=None, children=None):
        self.keys = keys if keys else []
        self.children = children if children else []

    @property
    def nbkeys(self):
        return len(self.keys)
```

Fonctions et méthodes autorisées

Les fonctions que vous pouvez utiliser :

- `len` sur les listes.
- `range`.
- `str` : transforme son paramètre en chaîne.

Rappels, l'opérateur + sur les chaînes et les couples :

```
>>> s = ""
>>> s += str(12)
>>> s
'12'

>>> (1, 2) + (3, 4)
(1, 2, 3, 4)
```

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.