Algorithmique
Contrôle n° 3 (C3)

Info-spé (S3)
Epita

24 octobre 2016 - 14 : 45

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 (Hachage linéaire – *2 points*)

Supposons l'ensemble de clés suivant E={data, kirk, neelix, odo, picard, q, quark, sisko, tuvok, worf} ainsi que la table 1 des valeurs de hachage associées à chaque clé de cet ensemble E. Ces valeurs sont comprises entre 0 et 10 ($m = 11$).

TABLE 1 – Valeurs de hachage

data	4
kirk	5
neelix	3
odo	1
picard	7
q	6
quark	2
sisko	7
tuvok	1
worf	7

Représenter la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans l'ordre de la table 1 (de `data` jusqu'à `worf`) et dans le cas du hachage linéaire avec un coefficient de décalage $d = 4$.

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

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

TABLE 2 – Valeurs de hachage

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

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 ?

TABLE 3 – Tableaux possibles ?

	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

Exercice 3 (Hachage : Questions. . . – *3 points*)

1. Citez trois propriétés que doit posséder une fonction de hachage.
2. A quoi doit-on une collision secondaire ?
3. Collisions mises à part, quel phénomène provoque le hachage linéaire et qu'envisage t-on pour le résoudre ?

Des arbres

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

Implémentation classique

```
class Tree:
    def __init__(self, key=None, children=None):
        self.key = key
        if children is not None:
            self.children = children
        else:
            self.children = []

    @property
    def nbChildren(self):
        return len(self.children)
```

Implémentation *premier fils - frère droit*

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

Exercice 4 (Arité moyenne d'un arbre général – *4 points*)

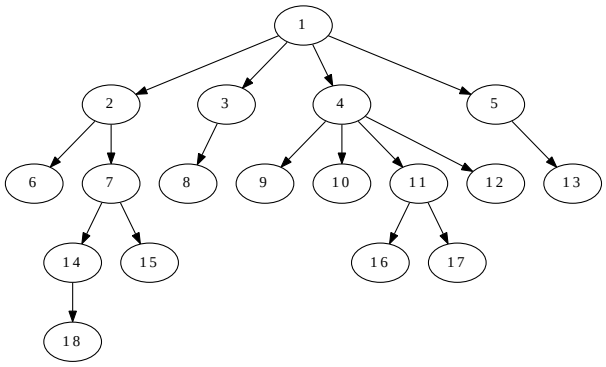

FIGURE 1 – Arbre général

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* (nœuds qui ne sont pas des feuilles).

Par exemple, pour l'arbre de la figure 1, il y a 8 nœuds internes (non feuilles), 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 (attention, il ne doit y avoir qu'un seul parcours de l'arbre), avec l'implémentation *premier fils - frère droit*.

Exercice 5 (Égalité – *5 points*)

Écrire la fonction `same(`T`,` B`)` qui vérifie si T, un arbre général en représentation "classique" et B, un arbre général en représentation *premier fils - frère droit*, sont identiques : ils contiennent les mêmes valeurs dans les mêmes nœuds.

A B-Tree

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

```
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)
```

Exercice 6 (B-arbres et mystère – *3 points*)

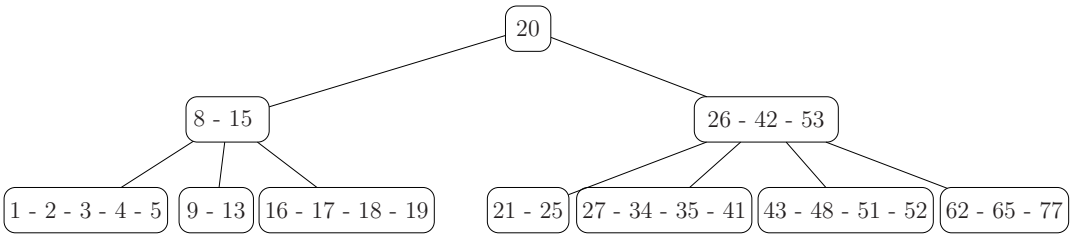

FIGURE 2 – B-arbre B_1

```
def mystery(B, a, b):
    i = 0
    while i < B.nbKeys and B.keys[i] < a:
        i += 1
    c = 0
    if B.children == []:
        while i < B.nbKeys and b > B.keys[i]:
            i += 1
            c += 1
    else:
        c += mystery(B.children[i], a, b)
        while i < B.nbKeys and b > B.keys[i]:
            c += mystery(B.children[i+1], a, b) + 1
            i += 1
    return c
```

1. Pour chacun des appels suivants, avec B_1 l'arbre de la figure 2 :
   — quel est le résultat retourné ?
   — combien d'appels à `mystery` ont été effectués ?

   (a) `mystery(`B_1`, 1, 77)`

   (b) `mystery(`B_1`, 10, 30)`

2. Soient `B` un B-arbre non vide contenant des entiers, et `a` et `b` deux valeurs entières telles que `a` $<$ `b`. Que calcule la fonction `mystery(B, a, b)` ?