Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3#
Epita

5 mars 2019 - 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 en **annexe** !
  - — 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

Exercice 1 (Hachage linéaire – *2 points*)

Supposons l'ensemble de clés suivant E={aragog, buck, crockdur, croutard, dobby, fumseck, hedwige, kreattur, nagini, missteigne} 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

aragog	0
buck	1
dobby	4
fumseck	0
missteigne	0
nagini	5
crockdur	2
croutard	4
kreattur	8
hedwige	8

Représenter la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans l'ordre de la table 1 (de `aragog` jusqu'à `hedwige`) et dans le cas du hachage linéaire avec un coefficient de décalage $d = 5$.

Exercice 2 (Quelques questions – *5 points*)

1. Citez trois propriétés essentielles que doit posséder une fonction de hachage.
2. Quelle méthode de hachage génère des collisions secondaires ?
3. Quelle méthode de hachage permet de résoudre le phénomène de regroupement ou accumulation d'éléments (clustering) généré par le hachage linéaire ?
4. Citer deux méthodes de hachage de base.
5. Citer une méthode de hachage utilisant une fonction d'essais successifs.
6. Quelle méthode de résolution des collisions ne nécessite pas un tableau de hachage de taille supérieure ou égale au nombre de clés à hacher ?

Exercice 3 (Sérialisation – *5 points*)

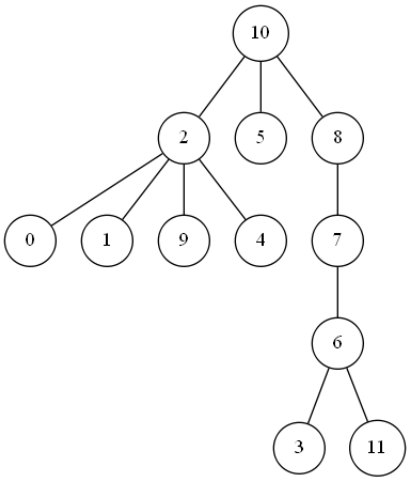

FIGURE 1 – Arbre général T

Nous allons nous intéresser à une représentation alternative des arbres généraux : les vecteurs de pères. Cette représentation est linéaire et peut donc être utilisée pour stocker notre arbre dans un fichier (sérialisation).

Le principe est simple : à chaque nœud de l'arbre on associe un identifiant unique (ici la clé contenue dans chaque nœud), sous la forme d'un entier compris entre 0 et la taille de l'arbre - 1. On construit ensuite un vecteur où la case i contient l'identifiant du père du nœud d'identifiant i. La racine de l'arbre aura pour père -1.

1. Donner le vecteur de pères pour l'arbre de la figure 1.
2. Écrire la fonction `buildParentVect(`T`,` n`)` qui retourne le vecteur (représenté par une liste en Python) de pères correspondant à l'arbre T en **implémentation "premier fils - frère droit"** de taille n. Les clés de l'arbre T sont les entiers dans $[0, n[$ (sans redondance).

Exercice 4 (Croissants – *4 points*)

Écrire une fonction qui construit une liste des clés contenues dans un B-arbre en ordre croissant.

Exemple d'application avec B l'arbre de la figure 2 :

```
>>> btree2list(B)
[3, 5, 11, 13, 18, 25, 32, 35, 40, 44, 46, 49, 50]
```

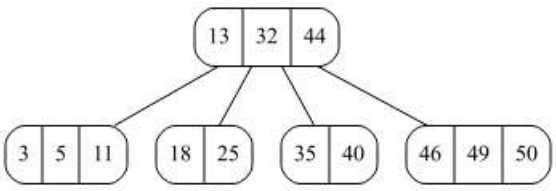

FIGURE 2 – B-Arbre B

Exercice 5 (Mesure sur les B-arbres – *4 points*)

Dans cet exercice on se propose de mesurer la *qualité* d'un B-arbre.

Écrire la fonction `occupation(B)` qui renvoie un réel correspondant au **nombre moyen de clés par nœud** (nombre de clés / nombre de nœuds) dans le B-arbre `B`. La fonction retourne 0 si l'arbre est vide.

Exemple d'application avec B l'arbre de la figure 2 :

```
>>> occupation(B)
2.6
```

Annexes

Les arbres généraux

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

Implémentation classique

```
    class Tree:
        def __init__(self, key, 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
```

B-Trees

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)
```

Fonctions et méthodes autorisées

- `len` sur les listes.
- `range`
- `append` sur les listes

`Queue` :

- `Queue()` retourne une nouvelle file ;
- q`.enqueue(`e`)` enfile e dans q ;
- q`.dequeue()` supprime et retourne le premier élément de q ;
- q`.isempty()` teste si q est vide.

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.