Algorithmique
Contrôle n° 3 (C3)

INFO-SPÉ - S3#
EPITA

17 mars 2021 - 9 : 30

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

Cours

Exercice 1 (The final frontier – *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`) :

1. dans le cas du hachage linéaire avec un coefficient de décalage $d = 3$;
2. dans le cas du hachage avec chaînage séparé.

Exercice 2 (Représentations – *3 points*)

Supposons un graphe simple G non orienté de 9 sommets, numérotés de 0 à 8, représenté par les listes d'adjacence suivantes :

0 : {1, 2, 6}
1 : {0, 3, 4}
2 : {0, 8}
3 : {1}
4 : {1, 6, 5, 7}
5 : {4}
6 : {0, 4, 8}
7 : {4, 8}
8 : {6, 7, 2}

1. Donner la matrice d'adjacence de G.
2. **Propriétés :** le graphe G est-il
   (a) connexe ?
   (b) complet ?
3. Remplir le tableau des degrés des sommets de G.

TD

Exercice 3 (Intervalle – *3 points*)

Écrire la fonction `test_inter(`T`, `a`, `b`)` qui vérifie si les valeurs de l'arbre général T sont bien dans l'intervalle $[a, b[$, avec l'implémentation *premier fils - frère droit*.

Exercice 4 (B-Arbres : insertion – *7 points*)

Nous travaillons ici avec des B-arbres dont toutes les valeurs sont des entiers naturels non nuls.

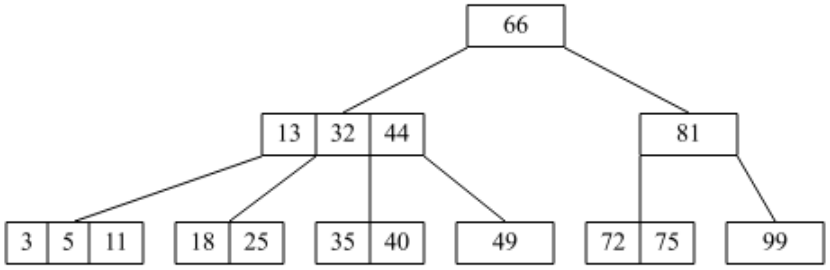

FIGURE 1 – B-arbre d'ordre 2

1. En utilisant le principe "à la descente" vu en td (hors bonus), dessiner l'arbre après insertion de la valeur 0 dans l'arbre de la figure 1.

2. Écrire la fonction `insert0(`B`)` qui insère la valeur 0 dans le B-arbre B (avec principe de précaution). La fonction retourne l'arbre résultat.

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

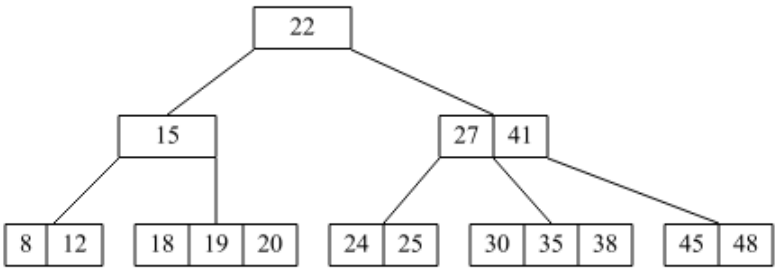

FIGURE 2 – B-arbre d'ordre 2

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 `btree2list(`B`)` qui retourne la représentation linéaire (de type `str`) du B-arbre B s'il est non vide, la chaîne vide sinon.

Annexes

Les arbres généraux

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

Implémentation *premier fils - frère droit*

- `B` : classe `TreeAsBin`
- `B.key`
- `B.child` : le premier fils
- `B.sibling` : le frère droit

B-Trees

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

- L'arbre vide est `None`
- L'arbre non vide est un objet de la class `BTree`, que l'on suppose importée
  - `B.degree` est le degré (l'ordre) des B-arbres que l'on manipule : c'est une constante donnée !
  - `B.keys` : liste des clés
  - `B.nbkeys` = `len(B.keys)`
  - `B.children` : liste des fils (`[]` pour les feuilles)
  - `B = BTree(key_list, child_list)` construit un nouvel arbre

Fonctions données

t est le degré des B-arbres

- La fonction `split(`B`, `i`)` éclate le fils n°i de l'arbre B :
  - L'arbre B existe (est non vide) et sa racine n'est pas un $2t$-nœud.
  - Le fils i de B existe et sa racine est un $2t$-nœud.

Autres fonctions et méthodes autorisées

Comme d'habitude : `len`, `range`, `min`, `max`, `abs`.

En plus, sur les listes :

```
>>> help(list.insert)
...     L.insert(index, object) -- insert object before index

>>> help(list.pop)
...     L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
```

`str` :

Rappel :

```
      >>> s = ""
      >>> s += str(42)
      >>> s
      '42'
```

Vos fonctions

Vous pouvez également écrire vos propres fonctions à condition qu'elles soient documentées : donnez leurs spécifications (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction doit être celle qui répond à la question.