Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3
Epita

9 novembre 2020 - 13 : 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

Exercice 1 (Quelques résultats différents – *5 points*)

1. Considérant que les collisions sont résolues par hachage coalescent, représenter la table de hachage de 11 éléments correspondant à l'application de la fonction de hachage

$$h(x) = (2x + 5)\ mod\ 11,$$

sur les clés $12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5$.

2. En reprenant les éléments de la question précédente et en considérant que les collisions sont résolues par hachage linéaire, représenter la table de hachage de 11 éléments qui en résulte.

3. En reprenant les éléments de la première question et en considérant que les collisions sont résolues par double hachage à l'aide de la deuxième fonction de hachage

$$d(x) = 7 - (x\ mod\ 7)$$

qui combinée avec $h(x)$ donne la table 1 d'essais successifs associés à chaque clé, représenter la table de hachage de 11 éléments qui en résulte.

TABLE 1 – valeurs d'essais successifs

Elt	h(x)	2	3	4	5	6	7	8	9	10	11
12	7	9	0	2	4	6	8	10	1	3	5
44	5	10	4	9	3	8	2	7	1	6	0
13	9	10	0	1	2	3	4	5	6	7	8
88	5	8	0	3	6	9	1	4	7	10	2
23	7	1	6	0	5	10	4	9	3	8	2
94	6	10	3	7	0	4	8	1	5	9	2
11	5	8	0	3	6	9	1	4	7	10	2
39	6	9	1	4	7	10	2	5	8	0	3
20	1	2	3	4	5	6	7	8	9	10	0
16	4	9	3	8	2	7	1	6	0	5	10
5	4	6	8	10	1	3	5	7	9	0	2

Exercice 2 (Cherche la somme – *4 points*)

Écrire la fonction `find_sum(`B`, sum)` qui vérifie s'il existe une branche dans l'arbre B (`TreeAsBin`) dont la somme des valeurs (entières) est *sum*.

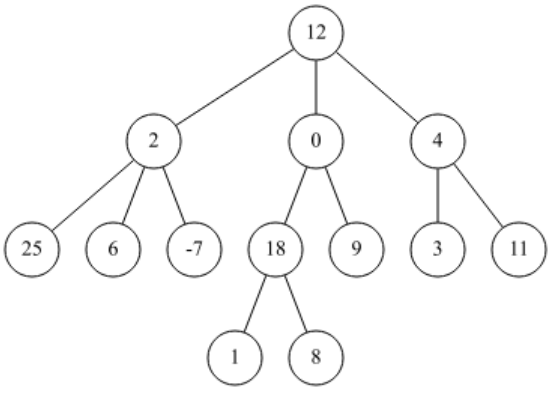
Exemples d'applications :

```
>>> find_sum(T4, 20)
True

>>> find_sum(T4, 10)
False

>>> find_sum(T4, 7)
True

>>> find_sum(T4, 17)
False
```



Arbre `T4`

Exercice 3 (Gap maximum – *4 points*)

Pour chaque nœud d'un B-arbre on appelle *gap* l'écart maximum entre deux clés consécutives. Le *gap maximum d'un B-arbre* sera donc le maximum des *gaps* des ses nœuds.

Par exemple, dans l'arbre de la figure 1, le gap maximum est 15 (35 - 20).

Écrire la fonction `maxgap` qui calcule le gap maximum d'un B-arbre. La fonction retournera 0 si l'arbre est vide.

Exercice 4 (What ? – *4 points*)

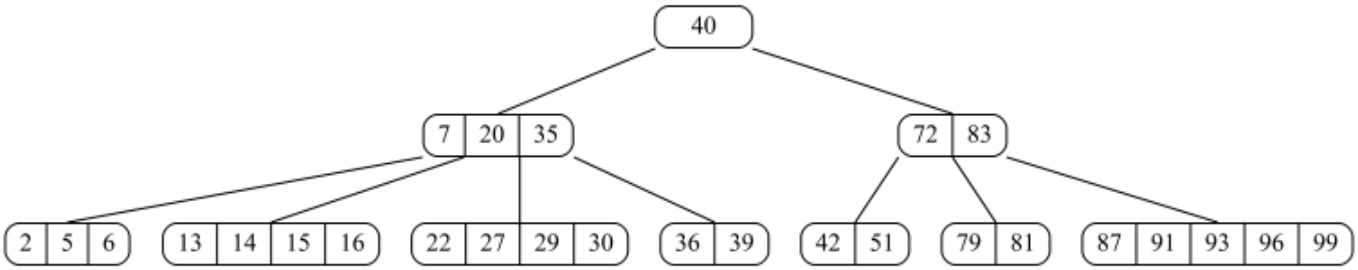


FIGURE 1 – B-arbre de degré 3

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)
```

1. Soit B_3 l'arbre de la figure 1. Quel sera le résultat de chacune des applications suivantes ?
   - `what(`B_3`, 2)`
   - `what(`B_3`, 7)`
   - `what(`B_3`, 18)`
   - `what(`B_3`, 39)`
   - `what(`B_3`, 41)`
   - `what(`B_3`, 99)`

2. Soit B un B-arbre non vide et x un entier. Que retourne `what(`B`,` x`)` ?

Exercice 5 (B-arbre : insertion et supression – *3 points*)

1. En utilisant le principe "à la descente" vu en td (hors bonus), dessiner l'arbre après insertion de la valeur 39 dans l'arbre de la figure 2.

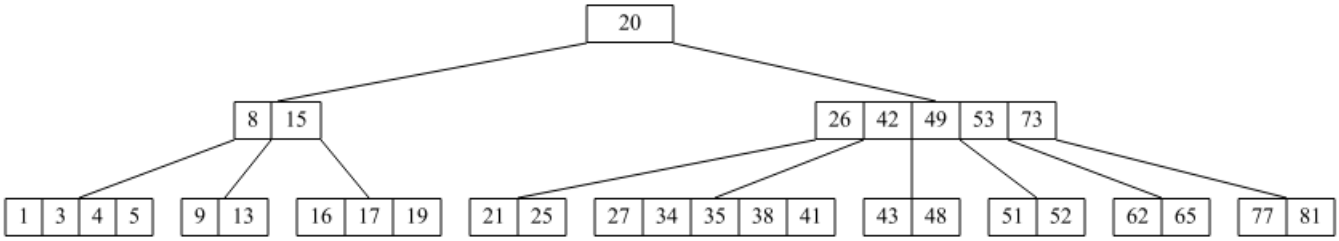


FIGURE 2 – B-arbre pour insertion

2. En utilisant le principe "à la descente" vu en td (hors bonus), dessiner l'arbre après suppression de la valeur 72 dans l'arbre de la figure 3.

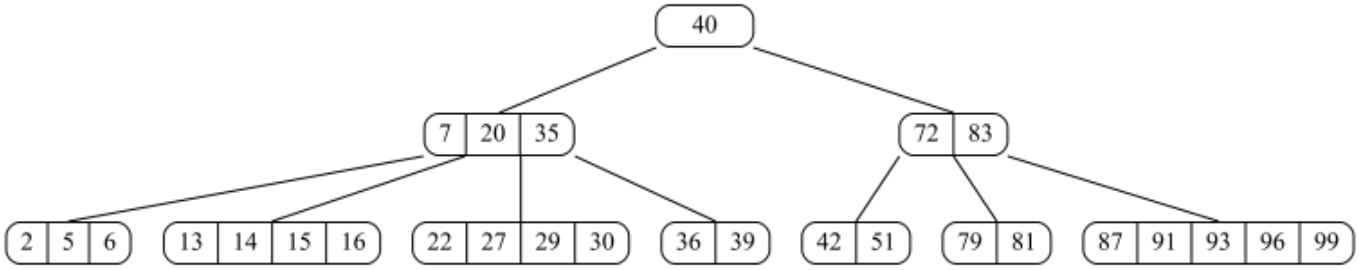


FIGURE 3 – B-arbre pour suppression

Annexes

Les arbres généraux : Implémentation *premier fils - frère droit*

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

```
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.

Rappel :
— 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 !

```
class BTree:
    degree = t    # given constant
    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

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

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.