Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3
Epita

29 octobre 2018 - 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 (Hachage fortement connecté – *4 points*)

1. Citer une méthode de hachage direct.
2. Quelle méthode de résolution des collisions ne nécessite pas un tableau de hachage de taille supérieure ou égale au nombre de clés à hacher ?
3. Quelle type de recherche est incompatible avec le hachage ?
4. Avec quelle méthode de résolution des collisions apparaissent des collisions secondaires ?

   Soit le graphe `G=<S,A>` orienté avec :
   `S={1,2,3,4,5,6,7,8,9,10}`
   et `A={(1,2),(1,6),(1,7),(2,3),(2,6),(3,1),(3,5),(4,3),(4,8),(4,9),(4,10),(5,1),(7,6),(8,5),(8,10),(10,9)}`
5. Représenter graphiquement le graphe correspondant à `G`.
6. Donner le tableau `DemiDegréIntérieur` tel que $\forall\ i \in [1, Card(S)]$, `DemiDegréIntérieur[i]` soit égal au demi-degré intérieur de `i` dans `G`.

Exercice 2 (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.

Exercice 3 (Levels – *4 points*)

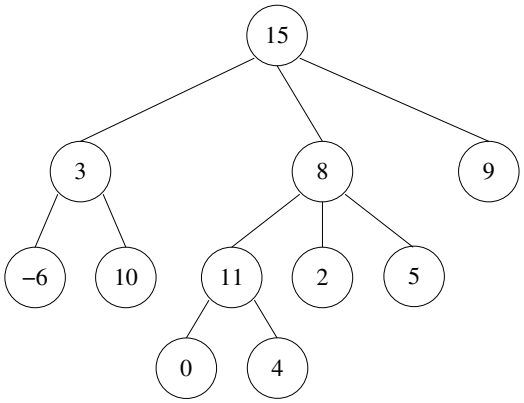

FIGURE 1 – T

Écrire une fonction qui construit une liste des clés d'un arbre général en représentation *premier fils - frère droit* niveaux par niveaux : une liste de listes, chaque sous-liste contenant les clés d'un niveau.

Exemple d'application, avec `T` l'arbre de la figure 1 :

```
>>> levels(T)
[[15], [3, 8, 9], [-6, 10, 11, 2, 5], [0, 4]]
```

Exercice 4 (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 2, le gap maximum est 16 (42 - 26).

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

Exercice 5 (B-arbres et mystère – *3 points*)

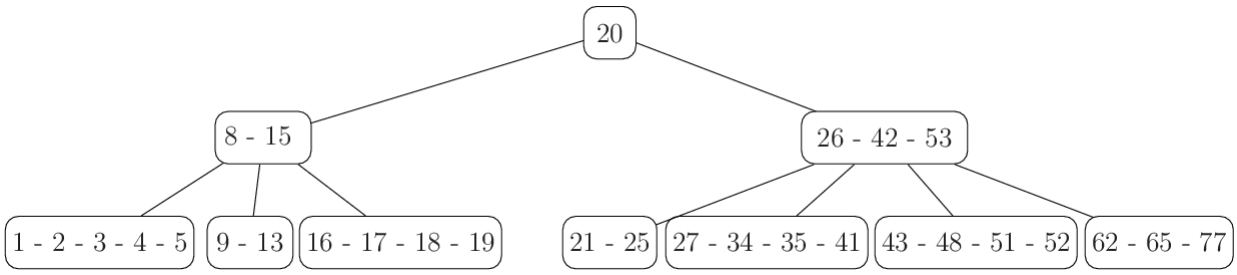

FIGURE 2 – B-arbre B_1

```
1  def mystery(B, a, b):
2      i = 0
3      while i < B.nbkeys and B.keys[i] < a:
4          i += 1
5      c = 0
6      if B.children == []:
7          while i < B.nbkeys and b > B.keys[i]:
8              i += 1
9              c += 1
10     else:
11         c += mystery(B.children[i], a, b)
12         while i < B.nbkeys and b > B.keys[i]:
13             c += mystery(B.children[i+1], a, b) + 1
14             i += 1
15     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)` ?

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.