Algorithmique
Contrôle n° 3 (C3)

Info-spé - S3
Epita

novembre 2019

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 (Haches et graphes... – *5 points*)

1. Donnez deux méthodes de hachage indirect.
2. Avec quelle méthode de résolution des collisions apparaissent des collisions secondaires ?
3. Qu'est-ce qu'une collision secondaire ?
4. Qu'est-ce que l'ordre d'un graphe orienté ?
5. Comment nomme t-on un sommet de degré nul ?

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)}`

6. S'ils existent, quels sont les sommets de `G` de demi degré extérieur égal à 0 ?
7. S'ils existent, quels sont les sommets de `G` de demi degré intérieur égal à 1 ?

Exercice 2 (Arité moyenne d'un arbre général – *5 points*)

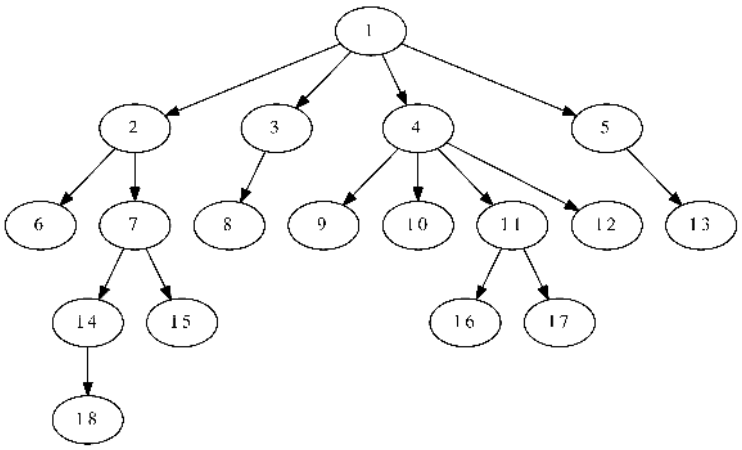

FIGURE 1 – Arbre général

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*.

Par exemple, pour l'arbre de la figure 1, il y a 8 nœuds internes, 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(`T`)` qui retourne l'arité moyenne de l'arbre général T si $taille(T) > 1$, 0 sinon, avec l'implémentation *premier fils - frère droit*.

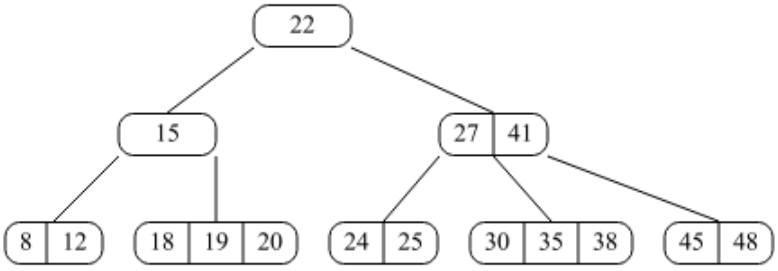

FIGURE 2 – B-arbre d'ordre 2

Exercice 3 (B-Arbres : insertions – *8 points*)

1. Insérer **successivement** les clés 36 et 42 dans l'arbre de la figure 2 en appliquant le principe de précaution (avec éclatements à la descente). Dessiner **uniquement** l'arbre obtenu après chaque insertion.

2. Écrire la fonction récursive `__insert(`B`, `x`)` qui insère x dans le B-arbre non-vide B sauf x est déjà présente.
   - ▷ Utiliser le principe de précaution (à la descente).
   - ▷ Utiliser la procédure d'éclatement dont les spécifications sont les suivantes :
     La procédure `split(`B`, `i`)` éclate le fils n°i de l'arbre B (avec t le degré du B-arbre) :
     — 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.
   - ▷ On suppose implémentée la fonction `search_pos` dont les spécifications sont les suivantes :
     La fonction `search_pos(`L`, `x`)` cherche la valeur x dans la liste L non vide. Elle retourne la position de x dans L s'il est présent, sa position "virtuelle" (celle où il devrait être) dans le cas contraire.
   - ▷ La fonction `__insert` sera appelée par la fonction suivante, qui retourne l'arbre après insertion éventuelle et un booléen indiquant si l'insertion a eu lieu :

```
    def insert(B, x):
        if B ==  None:
            return (BTree([x]), True)
        else:
            if B.nbkeys == 2 * B.degree - 1:     # root split
                B = BTree([], [B])
                split(B, 0)
            ok = __insert(B, x)
            return (B, ok)
```

Exercice 4 (B-arbres et mystère – *2 points*)

```
    def build(nodes, degree):
        BTree.degree = degree
        B = BTree(nodes[0])
        end = B.nbkeys + 2
        q = Queue()
        q.enqueue((B, 1))
        while not q.isempty():
            (cur, pos) = q.dequeue()
            if pos < len(nodes):
                for i in range(pos, pos + cur.nbkeys + 1):
                    new = BTree(nodes[i])
                    cur.children.append(new)
                    q.enqueue((new, end))
                    end += new.nbkeys + 1
        return B
```

Quels paramètres ont été passés à la fonction `build` pour obtenir l'arbre de la figure 2 ?

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`.

Les méthodes de la classe `Queue`, que l'on suppose importée :
— `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.

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.

>>> help(list.append)
...    L.append(object) -> None -- append object to end
```

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.