Algorithmique
Contrôle n° 2

S2# – Epita

D.S 307831.62 BW (31 Octobre 2016 - 09 :00)

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 n'avez le droit d'utiliser que ce qui a été vu en TD et autorisé en **annexe**
  - ☐ Vos fonctions doivent impérativement respecter les exemples d'applications donnés.

- ☐ Durée : 2h00 (May the force...)

Exercice 1 (Arbre Binaire : Construction - *2 points*)

Soit un arbre binaire B dont les parcours *infixe* et *suffixe* affichent les séquences suivantes :

infixe `H D I B E A F C J G`
suffixe `H I D E B F J G C A`

1. Représenter graphiquement l'arbre B correspondant à ces deux parcours.
2. Donner le parcours *préfixe* de l'arbre B.

Exercice 2 (Arbre Binaire de Recherche - *4 points*)

1. En utilisant les opérations définies dans le type abstrait ABR[1], donner les axiomes et éventuelles préconditions de l'opération `recherche : Elément` $\times$ `ABR` $\rightarrow$ `Booléen`.
2. Les suites suivantes correspondent aux valeurs rencontrées lors d'une recherche dans un arbre binaire de recherche. Lesquelles sont valides ?
   - `88, 65, 64, 11, 59, 54, 13, 33, 51, 34, 46, 39, 40, 45, 44, 42`
   - `17, 89, 19, 57, 54, 26, 32, 36, 41, 46, 47, 93, 48, 60, 74, 88`
   - `94, 76, 74, 17, 63, 57, 52, 41, 39, 19, 35, 22, 31, 27, 26, 23`
   - `92, 32, 91, 36, 55, 56, 59, 79, 76, 73, 61, 10, 44, 11, 22, 31`

Exercice 3 (Matrices : Symétrique - *4 points*)

La matrice transposée d'une matrice A est la matrice $A^\intercal$, obtenue en échangeant les lignes et les colonnes de A.

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \text{ alors } A^\intercal = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Une matrice symétrique est une matrice carrée qui est égale à sa propre transposée.

Écrire la fonction `isSymmetric(A)` qui teste si une matrice non vide est symétrique.

Exercice 4 (Arbre Binaire : Similarités - *5 points*)

Écrire la fonction `checkPostOrder(A, B)` qui vérifie si les deux arbres binaires `A` et `B` ont la même liste de valeurs en ordre suffixe de rencontre. Vous avez le droit de faire d'autres fonctions si cela vous est nécessaire.

Exercice 5 (Arbre Binaire : PME - *6 points*)

En utilisant un parcours en largeur d'un arbre binaire, écrire la fonction `PME(B)` qui calcule la profondeur moyenne externe de l'arbre `B` supposé non vide.

---

1. cf. annexes

Algorithmique

Type algébrique abstrait : arbre binaire de recherche

```
TYPE
    ABR

UTILISE
    Noeud, Elément

OPERATIONS
    arbre_vide : → ABR
    <_, _, _>  : Noeud × ABR × ABR → ABR
    racine     : ABR → Noeud
    contenu    : Noeud → Elément
    g          : ABR → ABR
    d          : ABR → ABR

PRECONDITIONS
    racine(B) est-défini-ssi B≠arbre_vide
    g(B) est-défini-ssi B≠arbre_vide
    d(B) est-défini-ssi B≠arbre_vide

AXIOMES
    racine(<o, B1, B2>)= o
    g(<o, B1, B2>)= B1
    d(<o, B1, B2>)= B2

AVEC
    o         : Noeud
    B, B1, B2 : ABR
```

Python : classe, fonctions et méthodes autorisées

Arbre Binaire

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

- `None` est l'arbre vide.
- L'arbre non vide a 3 attributs : `key`, `left`, `right`.

```
>>> B = BinTree()
>>> B.key = 'une clef'
>>> B.left = None
>>> B.right = None
```

Les files

```python
import queue

def newQueue():
    return queue.Queue()

def isEmpty(q):
    return q.empty()

def enqueue(e, q):
    q.put(e)

def dequeue(q):
    if isEmpty(q):
        raise Exception("Empty Queue")
    return q.get()
```

Longueur [2]

```
>>> l = [0,1,2,3,4]
>>> a = len(l)
>>> print(a)
5
```

Ajout

```
>>> l = [1,2]
>>> l.append(5)
>>> l
[1,2,5]
```

Suppression

```
>>> l = [1,2,5]
>>> l.pop()
5
>>> l
[1,2]
```

2. la longueur n'est pas définie sur les files