Algorithmique
Contrôle n° 2 (C2)

Info-sup S2
Epita

4 mars 2019 - 9 : 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 (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

Exercice 1 (Un peu de cours... – *4 points*)

Soit l'arbre `B = {`ε`,0,1,01,11,010,011,110,0101,1100,1101}`.

1. Mesures :
   (a) Quelle est la taille de l'arbre `B` ?
   (b) Quelle est la hauteur de l'arbre `B` ?
   (c) Quelle est la longueur de cheminement de l'arbre `B` ?
   (d) Quelle est la profondeur moyenne externe de l'arbre `B` ?
2. En utilisant la numérotation hiérarchique, donner, dans l'ordre, la liste des nœuds de l'arbre `B`.

Exercice 2 (Maximum gap - *4 points*)

Pour cet exercice, on définit le *gap* (écart) d'une liste comme étant l'écart maximum entre deux valeurs de la liste. Par exemple, dans la matrice ci-dessous le *gap* de la première ligne est 13.

Écrire la fonction `maxgap` qui retourne le gap maximum des lignes d'une matrice non vide.

Exemple d'application avec la matrice ***Mat1*** *ci-contre :*

```
>>> maxgap(Mat1)
19
```

1	10	3	0	-3	2	8
-1	0	1	8	5	0	-4
10	9	14	1	4	-5	1
10	-3	7	11	6	3	0
7	8	-5	1	5	4	10

`Mat1`

En effet le gap maximum est celui de la ligne du milieu (19 = 14 - (-5)).

Exercice 3 (Recherche – *4 points*)

Écrire la fonction `searchMatrix(M, x)` qui retourne la position (i, j) de la première valeur `x` trouvée dans la matrice non vide `M`. Si `x` n'est pas présent, la fonction retourne $(-1, -1)$.

Exemples d'applications avec la matrice ***Mat1*** *ci-dessus :*

```
>>>  searchMatrix(Mat1, -5)
(2, 5)
>>> searchMatrix(Mat1, 5)
(1, 4)
>>> searchMatrix(Mat1, 15)
(-1, -1)
```

Exercice 4 (Tests – *4 points*)

Écrire la fonction `equal(`$B1$`, `$B2$`)` qui vérifie si les arbres binaires $B1$ et $B2$ sont identiques : ils contiennent les mêmes valeurs aux mêmes places.

Exercice 5 (Feuilles – *2 points*)

Écrire la fonction `leaves(`B`)` qui calcule le nombre de feuilles de l'arbre binaire B.

Exercice 6 (Mystery – *3 points*)

La fonction `mystery` ci-dessous construit un arbre binaire à partir d'une liste.

```
def build(L, a, b):
    if a > b:
        return None
    else:
        c = (b - a) // 2 + a
        return BinTree(L[c], build(L, a, c-1), build(L, c+1, b))

def mystery(L):
    return build(L, 0, len(L)-1)
```

1. Dessiner l'arbre, résultat de l'application de la fonction `mystery` à la liste suivante
   `L = [4, 8, 2, 9, 5, 10, 1, 6, 11, 3, 12, 7, 13]`
2. Quelles propriétés doit avoir la liste pour que le résultat soit :
   (a) un arbre binaire de recherche (ABR) ?
   (b) un arbre complet ?

Annexes

Les arbres binaires

Les arbres binaires manipulés ici sont les mêmes qu'en td.
— `None` est l'arbre vide.
— L'arbre non vide est un objet de la class `BinTree` avec 3 attributs : `key`, `left`, `right`.

```
class BinTree:
    def __init__(self, key, left, right):
        self.key = key
        self.left = left
        self.right = right
```

Fonctions et méthodes autorisées

Sur les listes :

— `len`

Autres :

— `range`
— `abs`
— `min` et `max`, mais uniquement avec deux valeurs entières !

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.