Algorithmique
Contrôle n° 2 (C2)

Info-sup S2
Epita

5 mars 2018 - 8 : 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 (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

Cours

Exercice 1 (Un peu de cours... – *5 points*)

Soit l'arbre `B = {`ε`,0,1,00,01,10,11,000,011,110,111,0001,0110,0111,1101}`.

1. Représenter graphiquement l'arbre `B` en donnant comme étiquette aux noeuds leur numéro d'ordre hiérarchique.
2. Quels sont *(en ordre hiérarchique)* les noeuds externes de l'arbre `B` ?
3. Quelle est la longueur de cheminement externe de l'arbre `B` ?
4. Quelle est la profondeur moyenne interne de l'arbre `B` ?
5. Quelle est la particularité d'un arbre localement complet ?

Exercice 2 (Arbre Binaire : Ordres – *2 points*)

Soit un arbre binaire B dont les traitements *infixe* et *suffixe* du parcours profondeur main gauche affichent les séquences suivantes :

infixe `C H R I S T O P H E`
suffixe `C R H S I O H E P T`

1. Représenter graphiquement l'arbre B correspondant à ces deux séquences.
2. Donner les valeurs de l'arbre B dans l'ordre de rencontre *préfixe*.

Matrices

Exercice 3 (Symétrie – *5 points*)

Écrire la fonction `v_symmetric` qui vérifie si une matrice (supposée non vide) est symétrique selon un axe horizontal (symétrie verticale).

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
0	8	1	1	1
3	1	4	7	5
10	0	9	3	8
1	1	10	10	7

FIGURE 1 – `Mat1`

1	1	10	10	7
10	0	9	3	8
3	1	4	7	**5**
0	8	1	1	1
3	1	4	7	**0**
10	0	9	3	8
1	1	10	10	7

FIGURE 2 – `Mat2`

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
3	1	4	7	5
10	0	9	3	8
1	1	10	10	7

FIGURE 3 – `Mat3`

Exemples d'applications sur les matrices des figures 1, 2 et 3 :

```
>>> v_symmetric(Mat1)
True
>>> v_symmetric(Mat2)
False
>>> v_symmetric(Mat3)
True
```

Arbres binaires

Exercice 4 (Test implémentation hiérarchique – *5 points*)

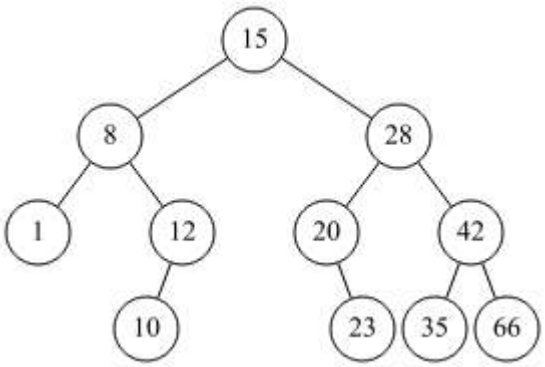

FIGURE 4 – Binary Tree

Rappel :

Implémentation hiérarchique (linéaire) : On peut utiliser un simple tableau (une liste en Python) pour représenter un arbre binaire. Il suffit de stocker chaque valeur à la position correspondant au numéro en ordre hiérarchique du noeud la contenant. Ici, toutes les cases non "utilisées" ont la valeur $\emptyset$ (`None` en Python).

Par exemple, voici la *représentation hiérarchique* de l'arbre de la figure 4 en Python :

```
L = [None, 15, 8, 28, 1, 12, 20, 42, None, None, 10, None, None, 23, 35, 66]
```

Écrire la fonction `object_vs_list(`B`, `L`)` qui vérifie si les deux arbres B, en représentation classique ("objet"), et L, en *implémentation hiérarchique*, sont identiques.

Exercice 5 (Père et fils – *4 points*)

La classe `BinTreeParent` ci-dessous permet de représenter les arbres binaires avec en chaque nœud les liens vers les fils, mais également le lien vers le père (`None` pour la racine).

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

Écrire la fonction `copywithparent` qui construit à partir de l'arbre binaire "classique" B (`BinTree`) un arbre binaire équivalent (contenant les mêmes valeurs aux même places) mais avec le père renseigné en chaque nœud (`BinTreeParent`).

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`

— `append`

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.