Algorithmique
Contrôle n° 2 (C2)

Info-sup S2#
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 (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

Du cours

Exercice 1 (Arbres binaires – *4 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 à chaque noeud son numéro d'ordre hiérarchique.
2. Quelle est la longueur de cheminement interne de l'arbre `B` ?
3. Quelle est la profondeur moyenne externe de l'arbre `B` ?

Exercice 2 (ABR : chemin de recherche – *2 points*)

Parmi les séquences suivantes, lesquelles peuvent correspondre à la suite des étiquettes des noeuds rencontrés dans un arbre binaire de recherche lors de la recherche de 42 ?

① 50 - 15 - 48 - 22 - 46 - 42
② 48 - 15 - 45 - 22 - 47 - 42
③ 15 - 22 - 45 - 43 - 35 - 42
④ 22 - 45 - 43 - 15 - 35 - 42

Des matrices

Exercice 3 (Transposée - *3 points*)

La matrice transposée d'une matrice est la matrice A^T obtenue en échangeant les lignes et les colonnes de A.

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

Écrire la fonction `transpose(`M`)` qui construit et retourne la matrice transposée de la matrice M non vide.

Exercice 4 (Symétrie verticale – *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
```

Des arbres binaires

Exercice 5 (Maximum Path Sum – *2 points*)

Dans un arbre binaire, nous définissons la *valeur d'une branche* comme étant la somme des valeurs de nœuds sur cette branche.

Écrire la fonction `maxpath(`B`)` qui détermine la plus grande valeur des branches de l'arbre binaire B (dont les clés sont des entiers).

Par exemple, dans l'arbre de la figure 4, la somme retournée sera $26 = 13 + 8 + 5$

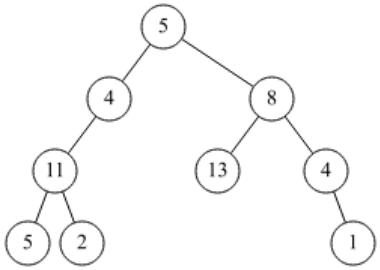

FIGURE 4 – maximum = 26

Exercice 6 (Full ? – *3 points*)

Écrire la fonction `full(`B`)` qui vérifie si l'arbre binaire B est localement complet. On considérera l'arbre vide comme localement complet.

Exercice 7 (Mystery – *2 points*)

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

```
1   def what(B):
2       R = []
3       if B != None:
4           q = Queue()
5           q.enqueue(B)
6           q2 = Queue()
7           L = []
8           while not q.isempty():
9               B = q.dequeue()
10              L.append(B.key)
11              if B.left != None:
12                  q2.enqueue(B.left)
13              if B.right != None:
14                  q2.enqueue(B.right)
15              if q.isempty():
16                  R.append(L)
17                  L = []
18                  (q, q2) = (q2, q)
19      return R
```

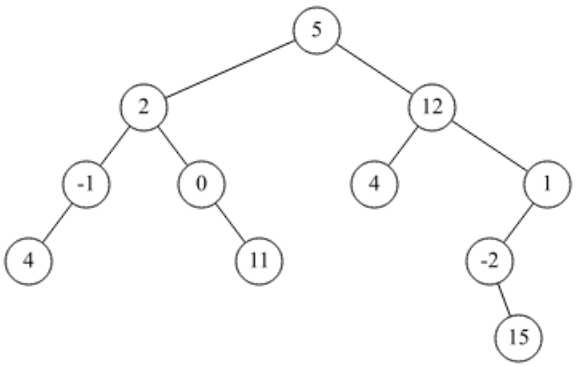

FIGURE 5 – Arbre `B`

Donner le résultat de l'application de `what` à `B` l'arbre de la figure 5.

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 !

Les files pour le code à lire

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.

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.