Algorithmique
Arbres binaires et généraux

SUP S2 EPITA

Examen B3

10 mars 2026

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet.**
  - **Indiquez de manière lisible vos NOM (en majuscules), prénom, UID et classe.**
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées.**
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes notés sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- **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.**
  - Les seules classes, fonctions, méthodes que vous pouvez utiliser sont données **ci-dessous.**
  - Vos fonctions doivent impérativement respecter les exemples d'applications donnés.
  - 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.
  - Comme d'habitude l'optimisation est notée. Si vous écrivez des fonctions non optimisées, vous serez notés sur moins de points. [1]
- Durée : 2h00

---

1. Des fois, il vaut mieux moins de points que pas de points.

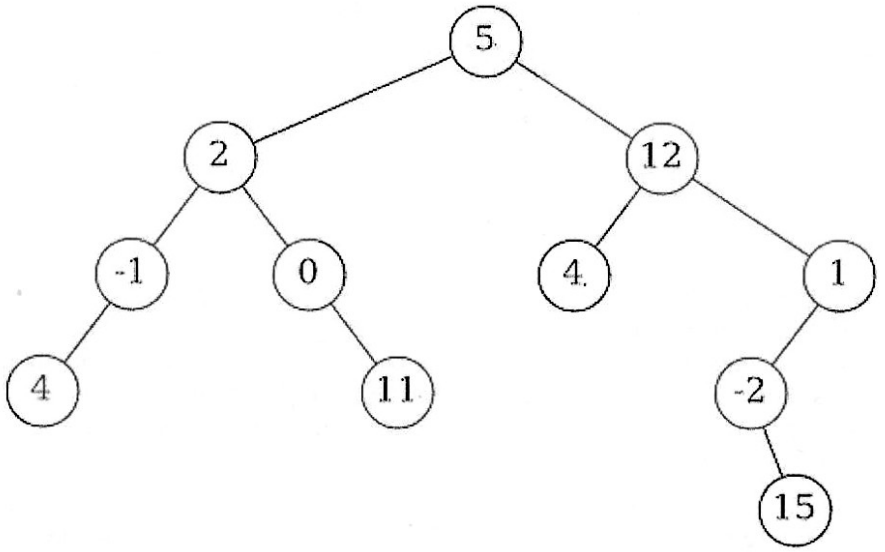
Exercice 1 (`size_levels(B, level, k)` – ***5 points***)

Écrire la fonction `size_levels(`B`,` *level*`,` k`)` qui prend en paramètres :

- B, un arbre binaire
- *level*, un entier positif ou nul représentant un niveau de l'arbre
- k, un entier positif ou nul

et renvoie `True` si le nombre de nœuds de l'arbre binaire B, en ne prenant en compte que les nœuds situés aux niveaux supérieurs ou égaux à *level*, est strictement supérieur à k. La fonction renverra `False` dans le cas contraire.

On considère que le niveau de la racine est `0`.



B1

Exemples d'applications :

```
>>> size_levels(None, 0, 0)
False

>>> size_levels(B1, 0, 0)
True
>>> size_levels(B1, 0, 5)
True
>>> size_levels(B1, 0, 11)
False
>>> size_levels(B1, 0, 15)
False

>>> size_levels(B1, 2, 5)
True
>>> size_levels(B1, 2, 8)
False

>>> size_levels(B1, 3, 0)
True
>>> size_levels(B1, 3, 5)
False
```

Exercice 2 (`add_trees(`$B1$`, `$B2$`)` – ***5 points***)

Écrire la fonction `add_trees(`$B1$`, `$B2$`)` qui prend deux arbres binaires $B1$ et $B2$ dont les nœuds contiennent des entiers et "additionne" ces deux arbres en construisant un nouvel arbre binaire.

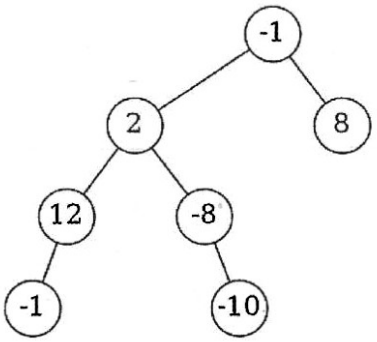
La règle d'addition est la suivante :

- si deux nœuds se superposent, la valeur du nœud résultant est la somme des valeurs des deux nœuds ;
- sinon, le nœud non vide est directement utilisé dans le calcul de la valeur correspondante.

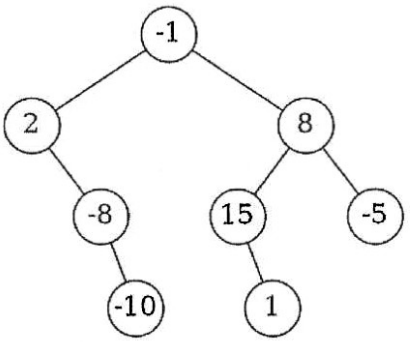
Chaque nœud du nouvel arbre doit être explicitement créé (aucun nœud des arbres $B1$ ou $B2$ ne peut être réutilisé).

La fonction retourne le nouvel arbre binaire ainsi construit.

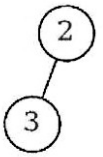
La résolution de cet exercice doit obligatoirement utiliser une approche récursive.



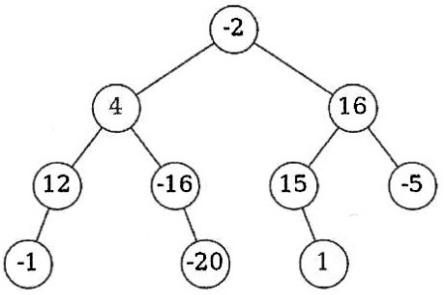
B1



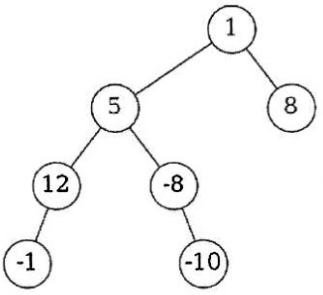
B2



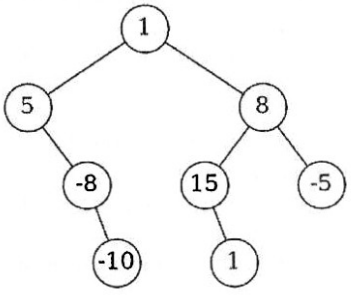
B3



`add_trees(`$B1$`, `$B2$`)`



`add_trees(`$B1$`, `$B3$`)`



`add_trees(`$B2$`, `$B3$`)`

Exercice 3 (`positive_sequence(B, k)` – ***7 points***)

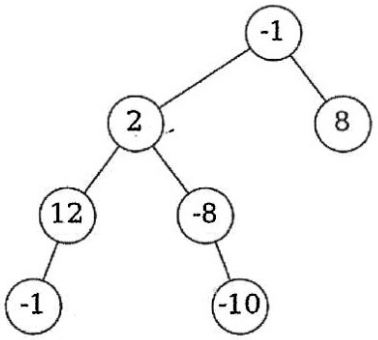
Écrire la fonction `positive_sequence(`B`, `k`)` qui prend en paramètres :

- B, un arbre binaire dont les nœuds contiennent des entiers ;
- k, un entier strictement positif.

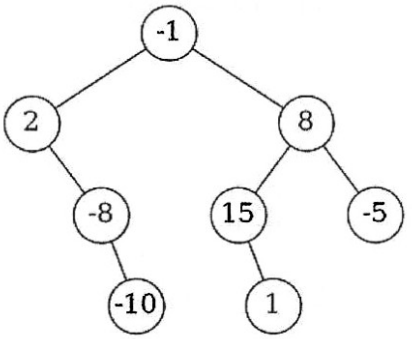
La fonction retourne `True` s'il n'existe **aucune** séquence successive de nœuds strictement positifs de longueur supérieure ou égale à k dans l'arbre binaire B, et `False` sinon.

Une séquence successive est définie comme un chemin allant d'un nœud à l'un de ses descendants, tel que tous les nœuds du chemin contiennent des valeurs strictement positives.

La résolution de cet exercice doit obligatoirement utiliser une approche récursive.



B1



B2

Exemples d'applications :

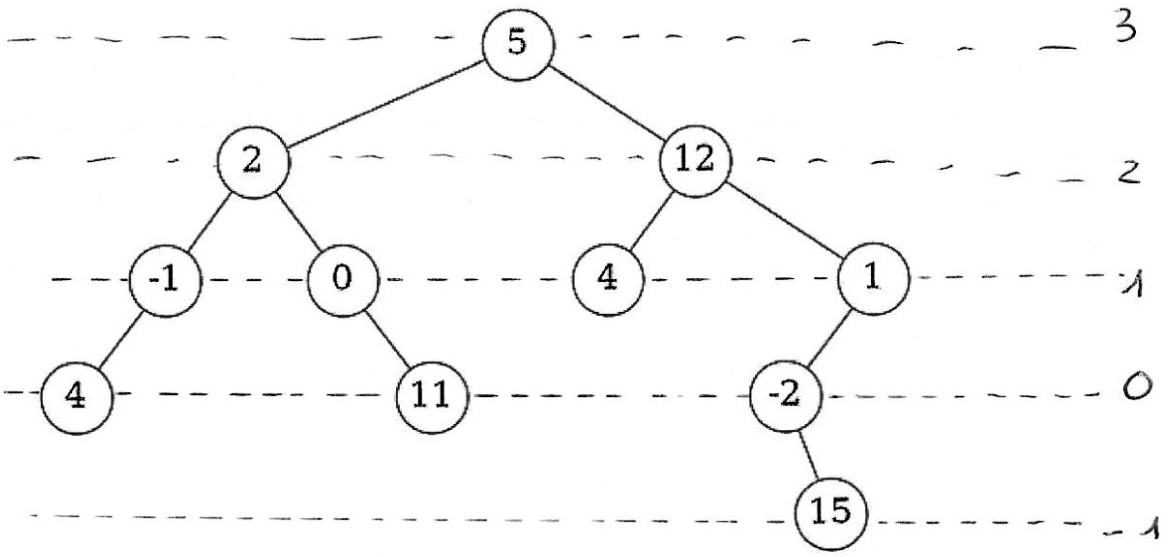
```
>>> positive_sequence(B1, 1)
False
>>> positive_sequence(B1, 2)
False
>>> positive_sequence(B1, 3)
True

>>> positive_sequence(B2, 1)
False
>>> positive_sequence(B2, 2)
False
>>> positive_sequence(B2, 3)
False
>>> positive_sequence(B2, 4)
True
```

Exercice 4 (Mystery – *3 points*)

```
1     def aux_mystery(B, a):
2         if B.left == None:
3             if B.right == None:
4                 if a == 0:
5                     return None
6                 else:
7                     return B
8             else:
9                 if a == 0:
10                    return B.right
11                else:
12                    B.right = aux_mystery(B.right, a-1)
13        else:
14            if B.right == None:
15                if a == 0:
16                    return B.left
17                else:
18                    B.left = aux_mystery(B.left, a-1)
19            else:
20                B.left = aux_mystery(B.left, a-1)
21                B.right = aux_mystery(B.right, a-1)
22        return B
23
24    def mystery(B, a):
25        if B == None:
26            return None
27        else:
28            return aux_mystery(B, a)
```

Dessiner sur la feuille de réponses l'arbre résultat de l'application de `mystery(B, 3)` avec B l'arbre binaire ci-dessous.



B

Annexes : types, méthodes et fonctions autorisées

Les arbres binaires

— L'arbre vide est `None`

— L'arbre non vide est (une référence sur) un objet de la class `BinTree` avec 3 attributs : `key`, `left`, `right`.

— `B` : classe `BinTree`
— `B.key` : contenu du nœud racine
— `B.left` : le sous-arbre gauche
— `B.right` : le sous-arbre droit

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

Pour créer un nouveau nœud contenant la clé `k`, de sous-arbres gauche `L` et droit `R` :

```
>>> N = BinTree(k, L, R)
```

Files

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.

Fonctions et méthodes autorisées

Les fonctions `min` et `max`, mais uniquement avec deux valeurs entières !

Vos fonctions

Vous pouvez écrire des fonctions 'intermédiaires' / 'supplémentaires', dans ce cas vous devez donner leurs spécifications : on doit savoir ce qu'elles font.

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.