Algorithmique
Arbres de recherche
SUP S2 EPITA

Examen B4
11 mai 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 (`insert_before_depth(B, x, d)` – ***6 points***)

Écrire la fonction `insert_before_depth(`B`, `x`, `d`)` qui prend en paramètres :
- B, un arbre binaire de recherche dont les nœuds contiennent des entiers ;
- x, un entier à insérer ;
- d, un entier positif ou nul représentant une profondeur maximale d'insertion.

La fonction insère x en feuille dans l'ABR B en suivant le chemin d'insertion standard, **uniquement si la profondeur d'insertion est inférieure ou égale à** d. Si la profondeur d'insertion nécessaire est strictement supérieure à d, x n'est pas inséré.

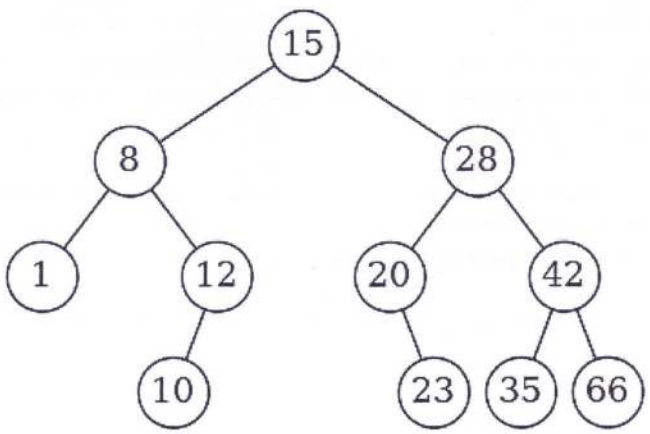
On considère que la racine de l'arbre est à la profondeur 0.

La fonction retourne un couple `(root, res)` où `root` est l'arbre binaire de recherche résultat et `res` vaut `True` si x a été inséré, `False` sinon.

Exercice 2 (`check_list(L, B, x)` – ***6 points***)

Écrire la fonction `check_list(`L`, `B`, `x`)` qui prend en paramètres :
- L, une liste d'entiers ;
- B, un arbre binaire de recherche dont les nœuds contiennent des entiers ;
- x, un entier.

La fonction retourne `True` si la liste L correspond exactement à la séquence de nœuds rencontrés lors de la recherche de x dans l'ABR B, et `False` sinon.



B

Exemples d'applications :

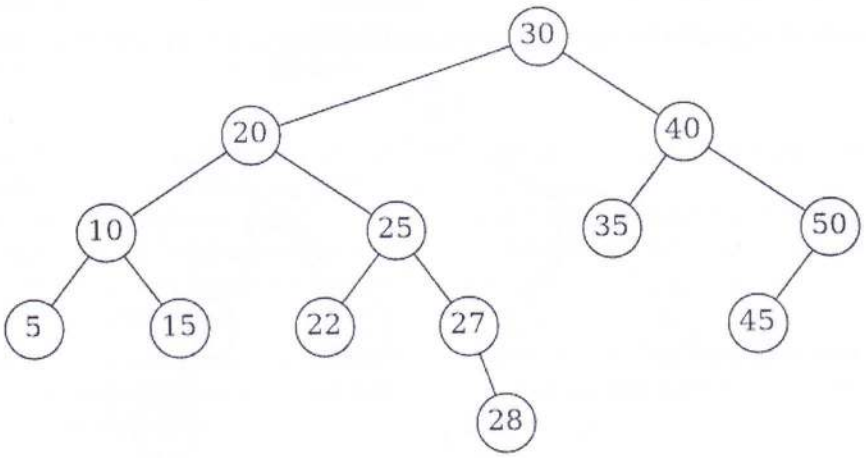
```
>>> check_list([], None, 42)
True
>>> check_list([15, 28, 20, 23], B, 23)
True
>>> check_list([15, 28, 23], B, 23)
False
>>> check_list([15, 28, 20, 23], B, 22)
True
>>> check_list([15, 8, 12, 10], B, 10)
True
>>> check_list([15, 8, 10], B, 10)
False
>>> check_list([15], B, 15)
True
```

Exercice 3 (Dessins – *4 points*)

1. Construire l'AVL correspondant aux ajouts successifs des valeurs $\{10, 5, 15, 2, 1, 7, 6, 0, 9, 8\}$ à partir de l'arbre vide.
   — Vous dessinerez 2 arbres : l'arbre après l'insertion de 6 puis l'arbre final
   — Indiquez quelles rotations ont été effectuées, dans l'ordre (par exemple si une rotation gauche a été effectuée sur l'arbre de racine 42, indiquer $rg(42)$ et si une rotation droite-gauche a été effectuée sur l'arbre de racine 42, indiquer $rdg(42)$).
2. A partir de l'AVL de la figure 2, construire l'AVL résultant de la suppression de la clé 40.

   Indiquez quelles rotations ont été effectuées, dans l'ordre (par exemple si une rotation gauche a été effectuée sur l'arbre de racine 42, indiquer $rg(42)$ et si une rotation droite-gauche a été effectuée sur l'arbre de racine 42, indiquer $rdg(42)$).

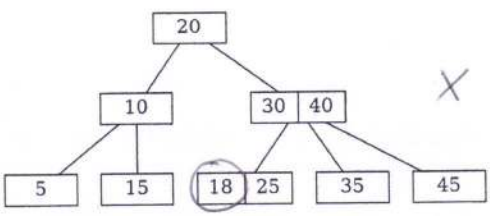
   Lors de la suppression dans un point double, on prendra forcement l'élément maximum du sous-arbre gauche.



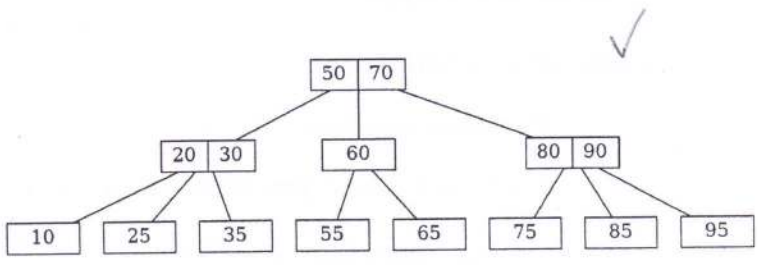
A-V.L. pour la suppression

Exercice 4 (Arbres de recherche – *2 points*)

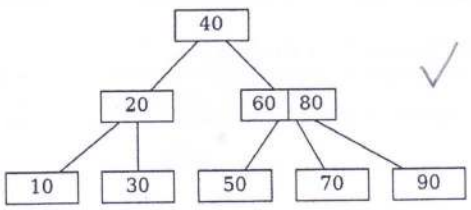
Lesquels des arbres suivants sont des arbres 2-3-4 ?



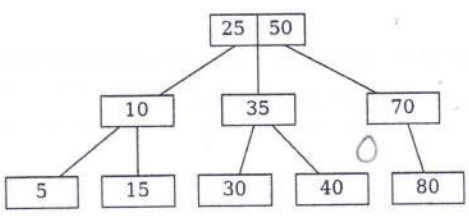
B_1



B_2



B_3



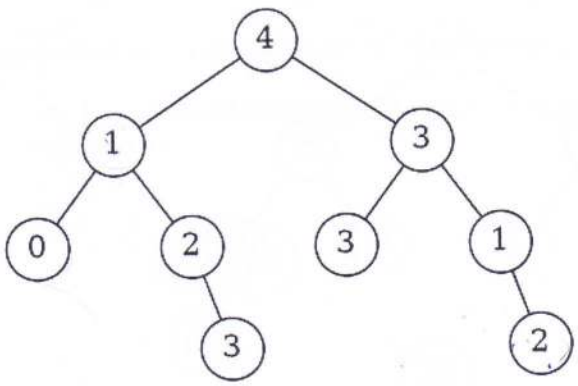
B_4

Exercice 5 (Mystery – *2 points*)

```
def myst(B, a):
    if B == None:
        return 0
    else:
        b = myst(B.right, a)
        if b == 0:
            B.key = B.key + a
        B.key = B.key + b
        b = myst(B.left, B.key)
        if b == 0:
            return B.key
        else:
            return b

def mystery(B):
    myst(B, 0)
```

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



B

Annexes : types

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)
```

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.