Nom	
Prénom	
Groupe	

Note	

Algorithmique
Arbres binaires et généraux

Sup S2 Epita

Examen B3

5 mars 2025

Consignes (à lire) :

- ☐ Vous devez répondre directement **sur ce sujet**.
  - 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 noté 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 en **annexe**.
  - 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 : 1h30

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

Exercice 1 (Liste – *6 points*)

Écrire la fonction `list_sum_limit(`*B*`, `*limit*`)` qui prend en paramètres :

- `B`, un arbre binaire contenant des entiers positifs
- `limit`, un entier

et construit et retourne une liste contenant la somme des éléments de chaque niveau uniquement si cette somme est strictement plus petite que `limit`.

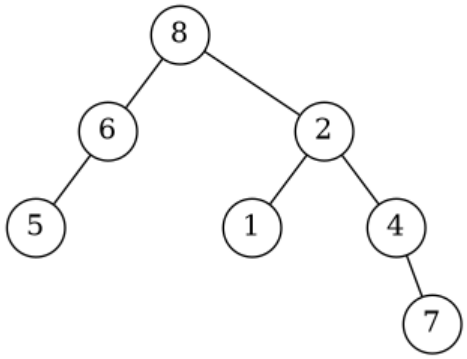

Figure 1 – B5

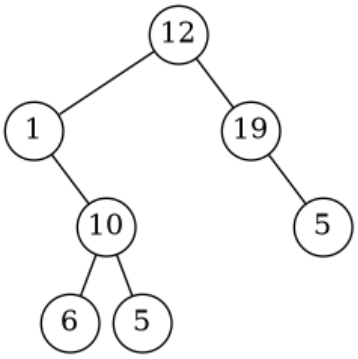

Figure 2 – B7

Exemples d'applications :

```
>>> list_sum_limit(None, 42)
[]
>>> list_sum_limit(B5, 5)
[]
>>> list_sum_limit(B5, 8)
[7]
>>> list_sum_limit(B5, 10)
[8, 8, 7]
>>> list_sum_limit(B5, 12)
[8, 8, 10, 7]
>>> list_sum_limit(B7, 11)
[]
>>> list_sum_limit(B7, 13)
[12, 11]
>>> list_sum_limit(B7, 16)
[12, 15, 11]
>>> list_sum_limit(B7, 40)
[12, 20, 15, 11]
```

Exercice 2 (Point simple – *6 points*)

Écrire la fonction `get_first_single(`B`)` qui retourne la clé du premier point simple rencontré dans l'arbre binaire `B` et -1 si `B` ne contient pas de point simple (l'arbre binaire `B` ne contient que des entiers positifs). Le **parcours profondeur** doit obligatoirement être utilisé.

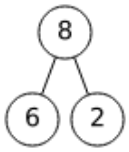

Figure 3 – B1

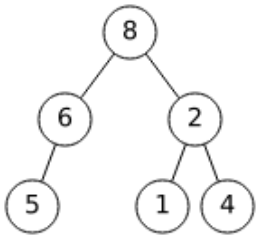

Figure 4 – B2

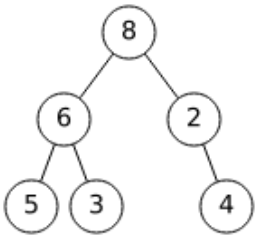

Figure 5 – B4

Exemples d'applications :

```
>>> get_first_single(None)
-1
>>> get_first_single(B1)
-1
>>> get_first_single(B2)
6
>>> get_first_single(B4)
2
```

Exercice 3 (Ordres – *4 points*)

Écrire la fonction `get_root_order(`B`)` qui retourne le numéro d'ordre de rencontre (1er, 2ème..) pour chacun des 3 ordres induits du parcours profondeur (sous la forme d'un triplet (préfixe, infixe, suffixe)) de la racine si elle existe, `None` sinon. Le **parcours profondeur** doit obligatoirement être utilisé.

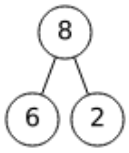

FIGURE 6 – B1

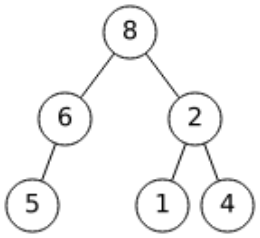

FIGURE 7 – B2

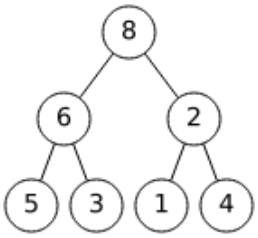

FIGURE 8 – B3

Exemples d'applications :

```
    >>> print(get_root_order(None))
    None
    >>> get_root_order(B1)
    (1, 2, 3)
    >>> get_root_order(B2)
    (1, 3, 6)
    >>> get_root_order(B3)
    (1, 4, 7)
```

Exercice 4 (Arbre général – *4 points*)

Soit l'arbre général **A** représenté sous la forme binaire **premier fils - frère droit** dans la figure 9 :

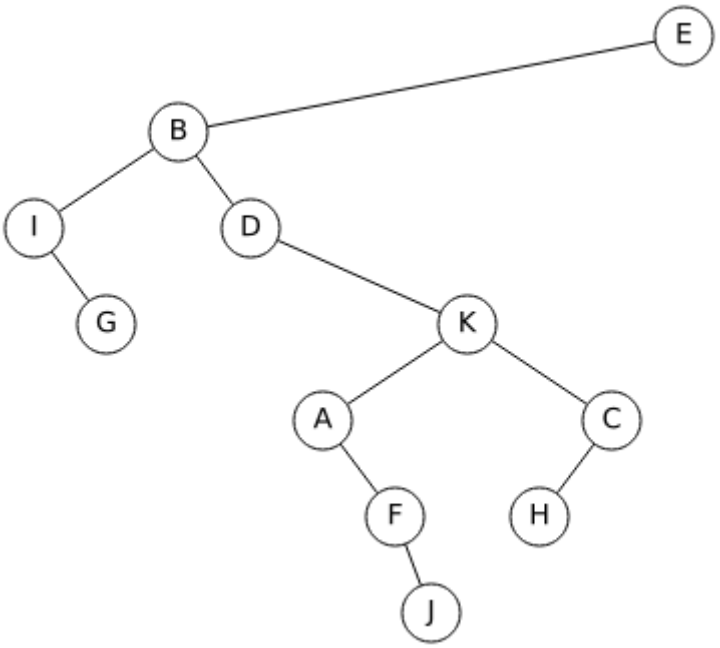

Figure 9 – Arbre général **A** sous forme binaire **premier fils - frère droit**

1. Quelle est la taille de l'arbre `A` ?
2. Quelle est la hauteur de l'arbre `A` ?
3. Quelle est la profondeur moyenne interne de l'arbre `A` ?
4. Donner la liste des valeurs des nœuds de l'arbre `A` rencontrés en ordre préfixe lors du parcours profondeur.
5. Donner la liste des valeurs des nœuds de l'arbre `A` rencontrés en ordre suffixe lors du parcours profondeur.

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
```

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

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes ainsi que la fonction `range`.

```
>>> L = []

>>> for i in range(5):
        L.append(i)

>>> L
[0, 1, 2, 3, 4]

>>> len(L)
5

>>> for i in range(5, 10):
        L.append(i)
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> for i in range(9, -1, -1):
        print(i, end=" ")
9 8 7 6 5 4 3 2 1 0
```

Les fonctions `min` et `max`, mais uniquement avec deux valeurs entières !
Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).

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.