Algorithmique
Arbres et Recherche II

SPÉ S3 EPITA

Examen B5

21 octobre 2025

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet.**
  Indiquez de manière lisible vos noms, prénoms, 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 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 : 2h00

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

Exercice 1 (Level up – *5 points*)

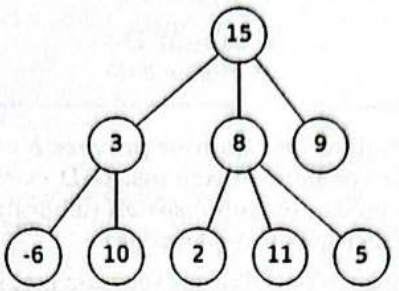

FIGURE 1 – Arbre général `T_level1`

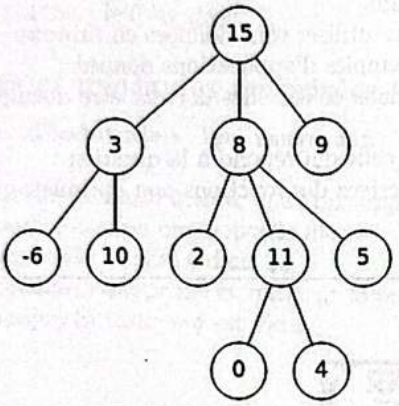

FIGURE 2 – Arbre général `T_level2`

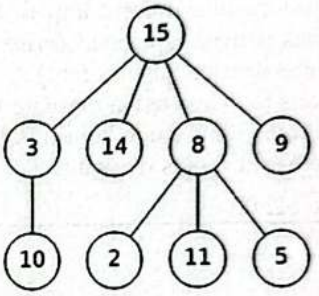

FIGURE 3 – Arbre général `T_level3`

Écrire la fonction `level_up(T:Tree)` qui vérifie que chaque niveau de l'arbre général `T` (implémentation par "n-uplets") contient strictement plus de nœuds que le niveau précédent.

Exemples d'applications :

```
>>> level_up(tree.Tree(42, []))
True

>>> level_up(T_level1)
True

>>> level_up(T_level2)
False

>>> level_up(T_level3)
False
```

Exercice 2 (Check sum – *6 points*)

42

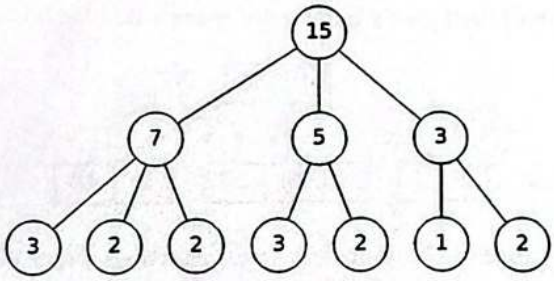

FIGURE 4 – Arbre général Bsum1

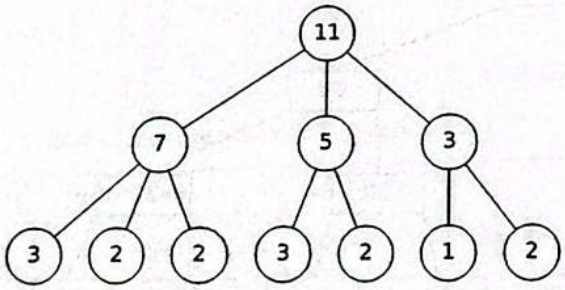

FIGURE 5 – Arbre général Bsum2

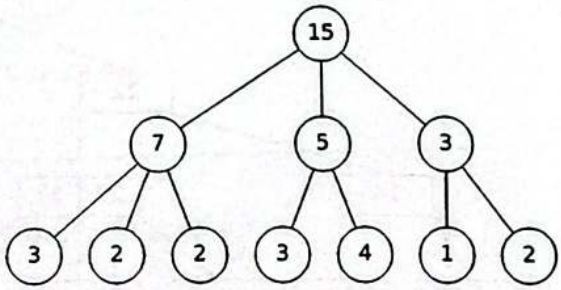

FIGURE 6 – Arbre général Bsum3

Écrire la fonction `check_sum(B:TreeAsBin)` qui vérifie si dans l'arbre général `B`, en implémentation *premier fils - frère droit*, la clé de chaque nœud interne est égale à la somme des clés de ses fils. Le **parcours profondeur** doit obligatoirement être utilisé.

Exemples d'applications :

```
>>> check_sum(TreeAsBin(42, None, None))
True

>>> check_sum(Bsum1)
True

>>> check_sum(Bsum2)
False

>>> check_sum(Bsum3)
False
```

Exercice 3 (B-arbre : insertions et suppression – *3 points*)

Pour chaque question, utiliser le principe "à la descente" (principe de précaution) vu en td (hors bonus).

1. Dessiner l'arbre final après insertions successives des valeurs 42, 12 et 2 dans l'arbre B1 de la figure 7.

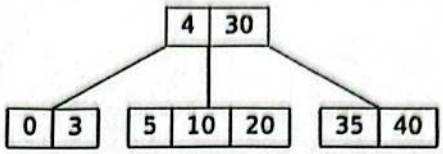

FIGURE 7 – B-arbre B1 pour insertions, degré 2

2. Dessiner l'arbre après suppression de la valeur 20 dans l'arbre B2 de la figure 8.

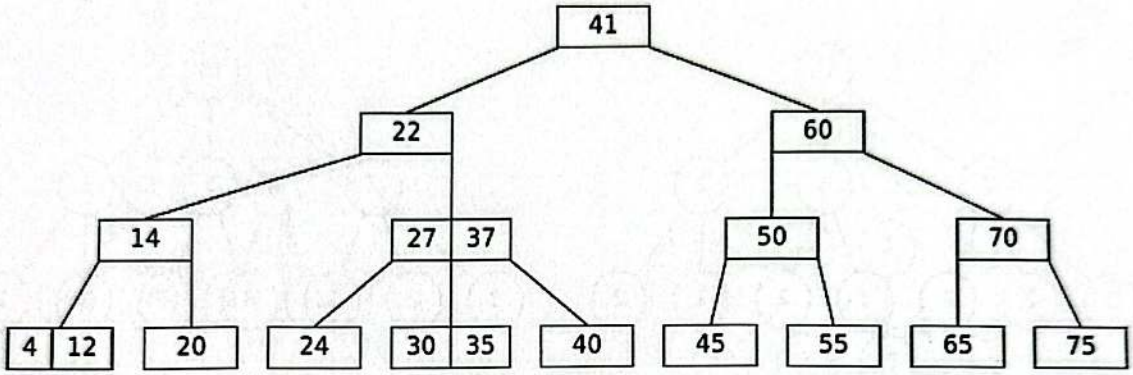

FIGURE 8 – B-arbre B2 pour suppression, degré 2

Exercice 4 (We are Groot – *3 points*)

Supposons l'ensemble de clés suivant E={Star-Lord, Rocket, Groot, Cosmo, Drax, Kraglin, Gamora, Mantis, Nebula, Yondu} ainsi que la table 1 des valeurs de hachage associées à chaque clé de cet ensemble E. Ces valeurs sont comprises entre 0 et 10 ($m = 11$).

Star-Lord	7
Rocket	7
Groot	4
Cosmo	5
Drax	6
Kraglin	2
Gamora	3
Mantis	2
Nebula	1
Yondu	5

TABLE 1 – Valeurs de hachage

Représenter la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans l'ordre de la table 1 (de `Star-Lord` jusqu'à `Yondu`) :

1. Dans le cas du hachage linéaire avec un coefficient de décalage $d = 3$;
2. Dans le cas du hachage coalescent, avec une zone de hachage primaire de taille p = 8 et une zone de réserve de taille r = 3.

Exercice 5 (What ? – *3 points*)

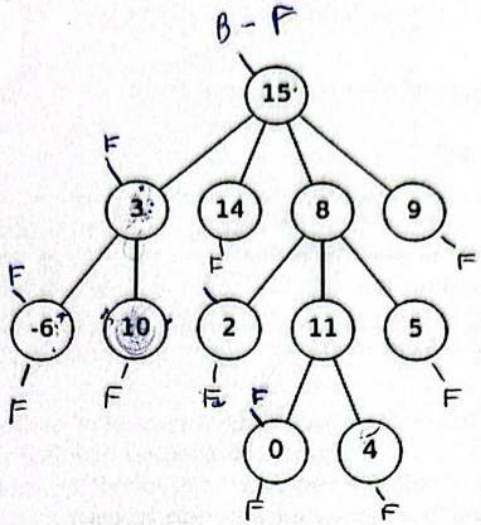

FIGURE 9 – Arbres Bwhat

Soit la fonction what définie ci-dessous :

```
1  def __aux(B, F, x, L):
2      C = B.child
3      while C and C.key != x and not __aux(C, B.child, x, L):
4          C = C.sibling
5      if C != None:
6          if C.key == x:
7              while F:
8                  if F.key != B.key:
9                      L.append(F.key)
10                 F = F.sibling
11         return True
12     else:
13         return False
14
15
16 def what(B, x):
17     L = []
18     __aux(B, B, x, L)
19     return L
```

Soit Bwhat l'arbre de la figure 9. Quel sera le résultat de chacune des applications suivantes ?

```
1  >>> what(Bwhat, 15)
2
3  >>> what(Bwhat, 8)
4
5  >>> what(Bwhat, 10)
6
7  >>> what(Bwhat, 4)
```

Annexes

Les arbres généraux

Les arbres (généraux) manipulés ici sont les mêmes qu'en td.

Implémentation par "n-uplets"

- `T` : classe `Tree`
- `T.key`
- `T.children` : listes des fils (`[]` pour les feuilles)
- `T.nbchildren` = `len(T.children)`

Implémentation *premier fils - frère droit*

- `B` : classe `TreeAsBin`
- `B.key`
- `B.child` : le premier fils
- `B.sibling` : le frère droit

Fonctions et méthodes autorisées

Comme d'habitude : `len`, `range`, `min`, `max`, `abs`

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 à condition qu'elles soient documentées : **donnez leurs spécifications** (on doit savoir ce qu'elles font, la signification des paramètres).

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