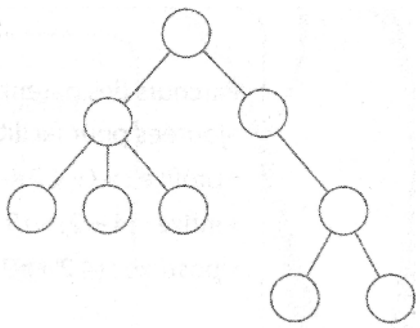
SDD_Arbres_Exercices_Partie_I

<u>Exercice 1</u> : QCM, vocabulaire des arbres (une seule réponse possible)

On considère l'arbre non étiqueté ci-contre :



1. Quelle est sa hauteur ?

☐ a. 9 ☐ b. 3
☐ c. 2 ☐ d. 8

2. Quelle est sa taille ?

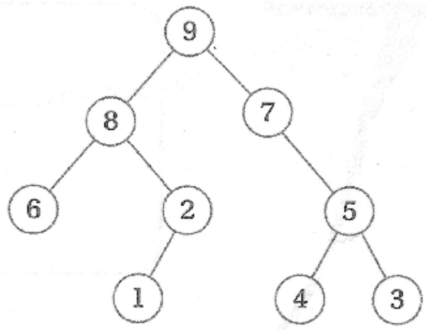
☐ a. 9
☐ b. 8
☐ c. 4

3. Quel est l'arité maximale parmi les nœuds de l'arbre ?

☐ a. 1 ☐ b. 2
☐ c. 3 ☐ d. 4

<u>Exercice 2</u> : QCM, vocabulaire du parcours d'arbres (une seule réponse possible)

On considère l'arbre binaire ci-contre étiqueté par des entiers.



1. Dans quel ordre seront examinés les nœuds lors d'un parcours en largeur ?

☐ a. 6–1–2–8–4–3–5–7–9
☐ b. 9–8–7–6–2–5–1–4–3
☐ c. 6–8–1–2–9–7–4–5–3
☐ d. 9–8–6–2–1–7–5–4–3

2. Dans quel ordre seront examinés les nœuds lors d'un parcours préfixe ?

☐ a. 6–1–2–8–4–3–5–7–9 ☐ c. 6–8–1–2–9–7–4–5–3
☐ b. 9–8–7–6–2–5–1–4–3 ☐ d. 9–8–6–2–1–7–5–4–3

3. Dans quel ordre seront examinés les nœuds lors d'un parcours infixe ?

☐ a. 6–1–2–8–4–3–5–7–9 ☐ c. 6–8–1–2–9–7–4–5–3
☐ b. 9–8–7–6–2–5–1–4–3 ☐ d. 9–8–6–2–1–7–5–4–3

4. Dans quel ordre seront examinés les nœuds lors d'un parcours postfixe ?

☐ a. 6–1–2–8–4–3–5–7–9 ☐ c. 6–8–1–2–9–7–4–5–3
☐ b. 9–8–7–6–2–5–1–4–3 ☐ d. 9–8–6–2–1–7–5–4–3

<u>Exercice 3</u> : QCM, arbre d'expression arithmétique (plusieurs réponses possibles)

Dans un arbre binaire représentant une expression arithmétique, que peut-on affirmer ?

a) Chaque nœud interne a exactement deux fils.
b) Chaque nœud a un ou deux fils.
c) Les feuilles contiennent des nombres.
d) Les nombres sont disposés sur des feuilles ou des nœuds internes.
e) Les opérateurs ne peuvent pas être sur des feuilles.

Exercice 4 : Reconstruire un arbre

Un arbre binaire est étiqueté avec des lettres. Un parcours préfixe de l'arbre donne : ALORHGIMET. Un parcours infixe donne : OLHRAMIEGT.

a) **Reconstruire** l'arbre binaire ayant produit ces deux résultats.
b) Qu'obtient-on en faisant un **parcours en largeur** d'abord ? Et en faisant un **parcours postfixe** ?

Exercice 5 (*) : Ajout d'un élément dans un ABR

Il y a plusieurs façons d'implémenter un ABR. On peut utiliser une structure de classe qui peut servir un arbre binaire quelconque.

Un nœud a deux enfants au maximum (pouvant être `None`) et, excepté à la racine à la racine, possède un parent unique. Chaque nœud possède une valeur. On crée donc **une classe `Node`** avec quatre attributs :

- Une valeur.
- Un parent.
- Un enfant à gauche.
- Un enfant à droite.

```python
# Définition de la classe `Node`
class Node :
    def __init__(self, val) :
        self.value = val
        self.parent = None  # Valeur par défaut
        self.left = None    # Valeur par défaut
        self.right = None   # Valeur par défaut


    # Surcharge de l'instruction `print`. Affiche la valeur du noeud
    def __str__(self) :
        return str(self.value)


    #######################################################
    ################# A COMPLETER #########################
    #######################################################
    def add(self, val) :
        pass

    #######################################################

tree = Node(15) # Racine de l'arbre
print(tree)     # Affiche la valeur de l'objet `tree`
```

1/ Sur Jupyter, **recopier** le programme précédent. Vérifier que l'on obtient bien `15` à l'exécution.
2/ (*) **Ecrire** la méthode `***add(self, val)***` qui permet d'ajouter une valeur à l'arbre en tenant compte des propriétés d'un ABR (on supprimera l'instruction `*pass*`), les valeurs étant toutes différentes.

Aide : procéder par récursivité en étudiant deux cas :

- si la valeur ajoutée est inférieure à la valeur du nœud parent, on l'ajoute au sous-arbre gauche,
- si la valeur ajoutée est supérieure à la valeur du nœud parent, on l'ajoute au sous-arbre droit.
- Dans chaque cas, si le sous-arbre (à gauche ou droite) est `*None*`, on crée alors un nouveau nœud et le nœud courant devient son parent.