Chapitre 8 :
Arbres binaires — Exercices

Exercice 1 ★ :

Représenter tous les arbres binaires ayant respectivement 3 ou 4 noeuds. Parmi ces arbres, préciser si certains sont des arbres binaires parfaits.

Exercice 2 ★ :

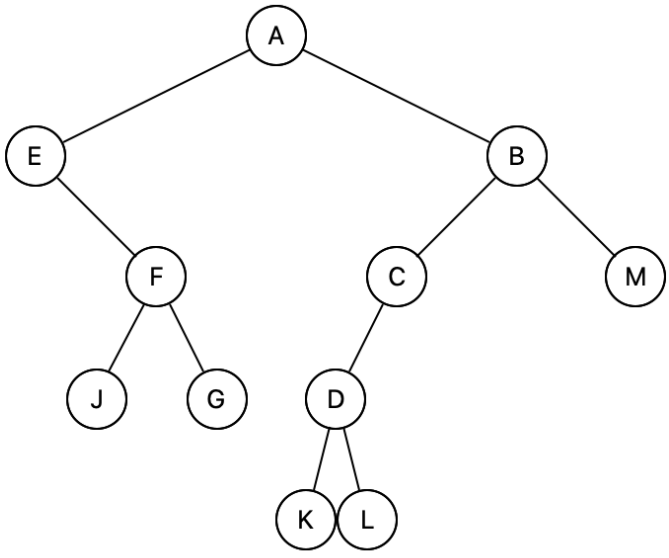
On sait qu'il existe :

- 1 arbre binaire vide ;
- 1 arbre binaire à un noeud ;
- 2 arbres binaires à deux noeuds ;
- 5 arbres binaires à trois noeuds ;
- 14 arbres binaires à quatre noeuds ;

Dénombrez le nombre d'arbres binaires à 5 noeuds. Attention : on ne demande pas de tous les construire ! Pensez aux différentes manières de répartir les noeuds dans les sous-arbres droits et gauches, puis utilisez les résultats ci-dessus.

Exercice 3 ★ :

Nous avons étudié dans le cours le parcours **infixe** d'un arbre binaire. Dans cet exercice, nous nous proposons de découvrir les modes de parcours suffixe et préfixe. Nous allons en particulier l'arbre ci-dessous :



a) Rappelez à quoi correspond la méthode de parcours infixe en donnant l'ordre des noeuds parcourus dans l'arbre.

b) On dispose de l'algorithme page suivante.
   - En appliquant cet algorithme à l'arbre ci-dessus, expliquez précisément pourquoi cet algorithme est appelé **parcours préfixe**.

- Traduisez cet algorithme en langage Python.

```
PARCOURS-PREFIXE(T) :
  si T != None :
    x ← T.racine
    affiche x.clé
    PARCOURS-PREFIXE(x.gauche)
    PARCOURS-PREFIXE(x.droit)
  fin si
```

c) On souhaite maintenant parcourir notre arbre par la méthode dite **suffixe.**
- Par analogie avec le parcours infixe et le parcours préfixe, que donnerait un parcours suffixe sur l'arbre de la page précédente?
- Écrire un tel algorithme puis le traduire en langage Python.

Exercice 4 ★★ :

Écrire une fonction **parfait(h)** qui prend pour argument un entier naturel non nul et renvoie un arbre binaire parfait de hauteur h. La valeur stockée dans chaque noeud sera sa hauteur.

Exercice 5 ★★ :

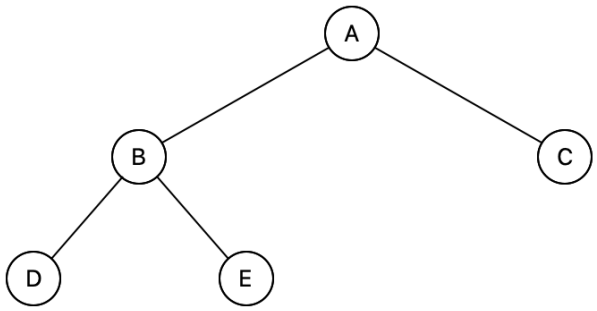
Partie A

Écrire une fonction **affiche(a)** qui affiche un arbre sous la forme suivante :
- pour un arbre vide, on n'affiche rien
- pour un noeud, on imprime un crochet ouvrant, son sous-arbre gauche, sa valeur, son sous-arbre droit et un crochet fermant.

Ce processus est bien sûr récursif.

Par exemple, pour l'arbre ci-contre, on va afficher :

```
[[[D]B[E]]A[C]]
```



On pourra s'inspirer de l'algorithme sur le parcours infixe du cours.
print(……, **end=""**) empêche le retour à la ligne.

Partie B

a) Dessiner l'arbre binaire associé à l'affichage obtenue par la fonction **affiche(a)** suivante :

```
[1[[2]3]]
```

b) À partir de cet exemple, expliquer précisément comment reconstruire un arbre à partir de n'importe quelle "formule".

c) Écrire une fonction Python permettant de réaliser cette opération. Vous créerez une fonction récursive **construireArbre** qui prendra une formule comme argument et renverra un Arbre constitué de Noeud. Il sera utile de créer une fonction auxiliaire comptant le nombre de crochets ouvrants et fermants.

Exercice 6 ★★ :

Donner cinq arbres différents de taille 3 dont les noeuds contiennent les valeurs 1,2, 3 et pour lesquels la fonction **parcoursInfixe** du cours affiche exactement à chaque fois :

1
2
3

Exercice 7 ★ :

Expliquez pourquoi le dictionnaire utilisé comme exemple introductif au début du chapitre contient 17576 pages.

Exercice 8 ★ :

Donnez tous les ABR formés de trois noeuds et contenant les entiers 2, 4 et 6.

Exercice 9 ★ :

Dans le cours, nous avons écrit une fonction **ajoute(x, A)** qui peut potentiellement créer des doublons si **A.valeur** est déjà présent dans l'ABR. Modifiez cette fonction afin d'ajouter l'élément x uniquement si celui-ci n'est pas déjà dans A.

Exercice 10 ★★ :

a) Où se situe le plus petit élément d'un ABR ?
b) En déduire une fonction récursive **minimum(A)** qui renvoie le plus petit élément de l'ABR A. Si l'arbre A est vide, alors cette fonction renvoie **None**.

Exercice 11 ★★★ :

Dans cet exercice, nous allons ajouter une fonction **compte(x, A)** qui renvoie le nombre d'occurrences de x dans l'ABR A.
L'ABR A a, à priori, été construit de manière quelconque : une valeur égale à la racine peut se trouver dans le sous-arbre gauche ou le sous-arbre droit.

a) En étudiant toutes les branches de l'arbre, écrire la fonction **compte(x, A)**.
b) Raffinez votre fonction **compte(x, A)** afin de ne pas descendre dans les sous-arbres dans lesquels vous êtes certain que la valeur **x** ne peut apparaitre.
c) Nous allons à présent intégrer cette fonction dans la classe ABR construite en cours. Modifiez la fonction **compte** afin de l'intégrer dans cette classe en Orienté Objet.