Fonctions usuelles sur les arbres binaires

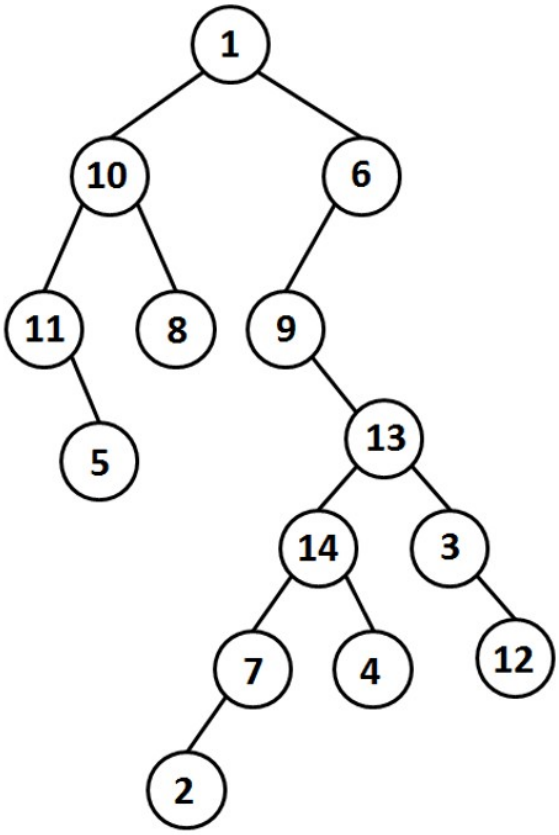
I. Quelques mesures

1. Coder et tester les deux fonctions ci-dessous.

- **est_vide(arbre)** : retourne **True** si l'arbre est vide **False** sinon
- **est_une_feuille(noeud)** : retourne **True** si le nœud est une feuille, **False** sinon

Soit l'algorithme récursif ci-dessous.

```
algo(arbre) :
    si arbre est vide :
        retourner la valeur 0
    sinon :
        retourner 1
            + algo(sous arbre gauche de arbre)
            + algo(sous arbre droite de arbre)
```



2. Que fait-il ? Expliquer.

3. Implémenter l'algorithme en lui donnant un nom adapté à sa fonction. Le tester sur l'arbre ci-contre.

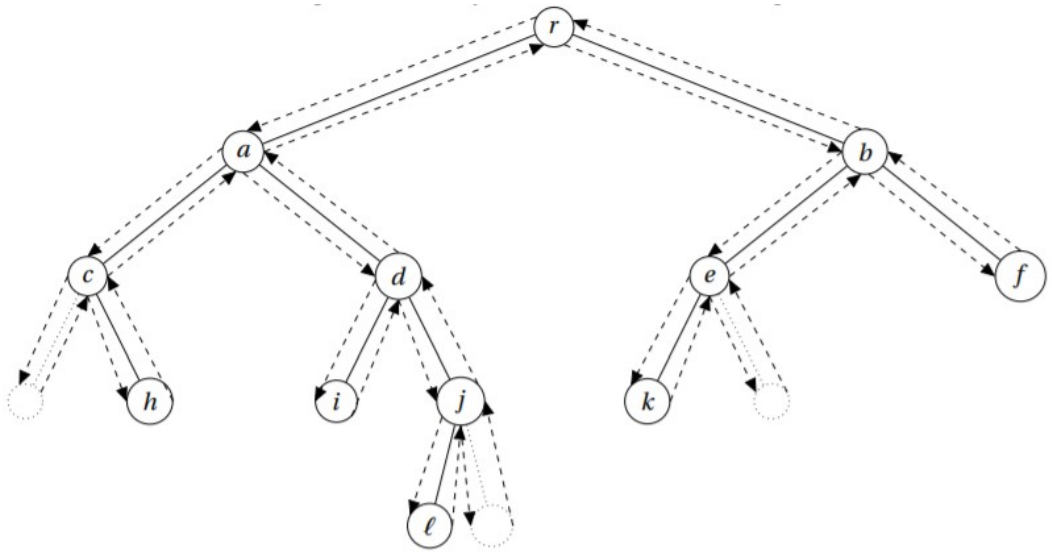
4. En vous aidant de l'algorithme précédent, proposer un algorithme qui donne la hauteur d'un arbre binaire.

5. Implémenter l'algorithme dans une fonction nommée **hauteur(arbre)** et le tester.

6. Enfin, coder une fonction **profondeur(arbre, etiquette)** qui donne la profondeur du nœud d'étiquette spécifiée. Si le nœud n'est pas trouvé la fonction retournera **None**.

II. Parcours en profondeur

Parcourir un arbre consiste à visiter tous ses nœuds. Pour chaque nœud visité, un traitement est effectué. Le traitement le plus simple peut être d'afficher la valeur de son étiquette. Lors d'un parcours en profondeur la visite d'un nœud parent est suivie récursivement de la visite de ses nœuds enfants jusqu'à atteindre un noeud vide ou une feuille.



Ainsi on parcourt l'arbre de haut en bas en visitant toujours les feuilles les plus profondes de gauche avant celles de droites.

On distingue trois parcours en profondeur en fonction de la position du traitement par rapport à la visite des noeuds enfants.

- *Parcours préfixe :* la traitement est effectué avant la visite des nœuds enfants

parcours_prefixe(arbre) :
    traitement
    parcours_prefixe(sous arbre gauche)
    parcours_prefixe(sous arbre droit)

- *Parcours infixe :* la traitement est effectué entre l visite du nœud enfant gauche et du nœud enfant droit

parcours_infixe(arbre) :
    parcours_infixe(sous arbre gauche)
    traitement
    parcours_infixe(sous arbre droit)

- *Parcours postfixe (ou suffixe) :* la traitement est effectué après la visite des nœuds enfants

parcours_postfixe(arbre) :
    parcours_postfixe(sous arbre gauche)
    parcours_postfixe(sous arbre droit)
    traitement

1. Que donnent comme résultats les trois parcours appliqués à l'arbre du **I**, le traitement correspondant à l'affichage de l'étiquette du nœud visité.

2. Implémenter les trois fonctions de parcours en profondeur. Vérifier ainsi vos réponses à la question précédente.

3. Proposer de nouvelles fonctions qui stockent les étiquettes des nœuds parcourus dans une liste plutôt que de les afficher, en utilisant, puis en utilisant pas, les propriétés de référencement des paramètres d'entrée des fonction Python.

III. Parcours en largeur d’abord

Ce parcours visite tous les nœuds d’un niveau de gauche à droite avant de passer au niveau suivant. En suivant ce parcours, on va d'abord visiter la racine, puis les nœuds à la profondeur 1, puis 2, ...

Pour l’implémentation de ce parcours, on a besoin d’une structure de file d'attente pour stocker les nœuds enfant de chaque niveau en cours de visite.

L’algorithme est le suivant :

```
parcours_largeur(arbre) :
    initialiser une file vide
    enfiler le nœud racine
    tant que la file n’est pas vide :
        défiler un nœud
        si le nœud n’est pas vide :
            traitement
            enfiler tous ses nœuds enfants
```

1. Implémenter le parcours en largeur et le tester sur l’arbre du **I**.

2. Proposer de nouvelles fonctions qui stockent les étiquettes des nœuds parcourus dans une liste plutôt que de les afficher, en utilisant, puis en utilisant pas, les propriétés de référencement des paramètres d’entrée des fonctions Python.

3. Reprendre tous les algorithmes de ce TP avec tous les arbres codés jusqu’à présent. D’abord manuellement puis en **Python**.