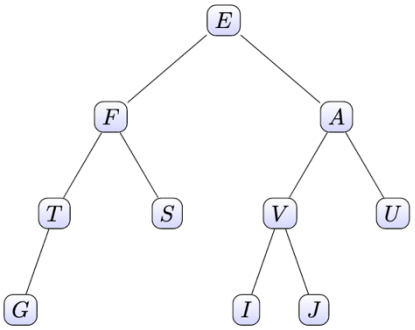
Exercices de révisions : arbres binaires

1. GENERALITES



On considère l'arbre représenté ci-contre.

a) Quelle est la taille de cet arbre ? **10**

b) Quelle est la valeur de sa racine ? **E**

c) Quelle est la hauteur de cet arbre ? **4**

d) Quel est le nombre de feuilles cet arbre? **5**

e) Quel est le père du nœud T ? Donnez un frère du nœud U.

F est le père du nœud de valeur T.
V est la valeur du nœud frère du nœud de valeur U.

2. UN ALGORITHME

On considère l'algorithme suivant :

```
Algorithme Mystere(T,x)
Entrée : T : un arbre et x : un noeud de cet arbre
Sortie : Un entier
1 si x = x.racine alors
2     retourner 0
3 sinon
4     retourner 1 + Mystere(T,x.pere)
```

a) Quelle sera la valeur retournée par l'appel `Mystere (arbre, 'V')` ?
`arbre` désigne l'arbre de l'exercice précédent.

Cet appel retournera la valeur 2 car l'appel récursif suivant sera `1+mystere(arbre,A)` puis ensuite `1+1+mystere(arbre,E)`. Or `mystere(arbre,E)` renvoie 0 (c'est le cas de base) donc le retour obtenu est 1 + 1 + 0 = 2.

b) Expliquez ce que ce que calcule cet algorithme.

La fonction `Mystere` calcule la profondeur du noeud passé en paramètre.

3. PARCOURS

Donnez l´affichage obtenu lors des différents parcours de l´arbre de l´exercice 1 :

a) Préfixe: **E F T G S A V I J U**

b) Postfixe (ou suffixe): **G T S F I J V U A E**

c) Infixe: **G T F S E I V J A U**

d) Largeur: **E F A T S V U G I J**

4. ARBRES BINAIRES DE RECHERCHE

a) Complétez cette définition d´un arbre binaire de recherche :

Un *arbre binaire de recherche* est un arbre binaire tel que :

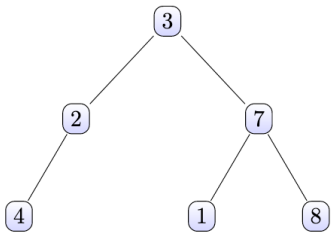
- Les clés (ou valeurs) du sous-arbre gauche sont **inférieures ou égales à celle de la racine.**

- Les clés* du sous-arbre droit sont **supérieures strictement à celle de la racine.**

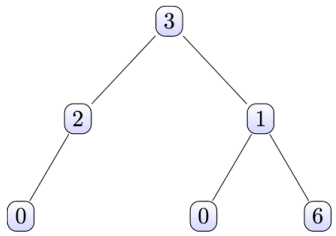
- Les deux sous-arbres sont **eux même des arbres binaires de recherche.**

*** Les mots *clé*, *valeur* ou *étiquette* sont équivalents.**

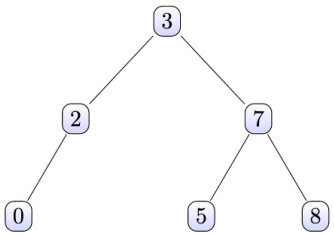
b) Parmi les arbres ci-dessous, lesquels ne sont pas des *arbres binaires de recherche* ?



Arbre 1



Arbre 2



Arbre 3

c) On dispose de la classe `Node` rappelée ci-dessous :

```python
class Node():
    def __init__(self, val):
        self.val = val
        self.gauche = None
        self.droit = None
```

Complétez le code de la méthode récursive d´insertion d´un nœud dans un *arbre binaire de recherche*.

```python
def insert(arbre, v):
    if arbre is None:
        .............
    else:
        if arbre.val == v:

            ........................
        elif arbre.val < v:

            ........................
        else:

            ........................
    return arbre
```

Ce code a été vu dans le cours sur les arbres binaires de recherche:

```python
def insert(arbre, v):
    if arbre is None:
        return Node(v)
    else:
        if arbre.val == v:
            return arbre
        elif arbre.val < v:
            arbre.droit = insert(arbre.droit, v)
        else:
            arbre.gauche= insert(arbre.gauche, v)
    return arbre
```

5) PARCOURS EN LARGEUR

Complétez le code de parcours en largeur d´un arbre binaire :

```python
def BFS(arbre):
    if arbre == None:
        return
    # On crée une file vide
    q= File()
    # On enfile l´arbre
    q.enfiler(arbre)
    while not q.estVide():
        # On affiche la valeur de l´arbre en tête de file
        print(q.tete().val,end=' - ')

        node = ....................

        if node.gauche is not None:

            ........................

        if node.droit is not None:

            ........................
```

Ce code a aussi été vu dans le cours sur le parcours des arbres binaires:

```
def BFS(arbre):
    # Si l´arbre est vide, on sort
    if arbre == None:
        return

    # On crée une file vide
    q= File()
    # On enfile l´arbre
    q.enfiler(arbre)

    while not q.estVide():
        # print(q) # pour avoir une trace de la pile
        # On affiche la valeur de l´arbre en tête de file
        print(q.tete().val,end=' - ')
        # on la défile
        node = q.defiler()

        # On enfile le sous-arbre gauche si il existe
        if node.gauche is not None:
            q.enfiler(node.gauche)

        # On enfile le sous-arbre droit si il existe
        if node.droit is not None:
            q.enfiler(node.droit)
```