Algorithmes sur les arbres

Table des matières

1) Rappels

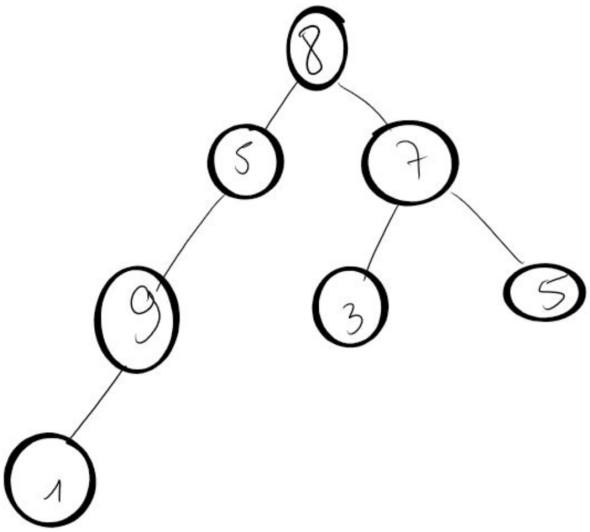
L'implémentation d'un arbre est la suivante :

```
class ArbreBinaire:
    def __init__(self,val):
        self.valeur=val
        self.gauche=None
        self.droite=None

    def insererGauche(self,valeur):
        if self.gauche == None:
            self.gauche = ArbreBinaire(valeur)
        else:
            nouveauNoeud = ArbreBinaire(valeur)
            nouveauNoeud.gauche = self.gauche
            self.gauche =nouveauNoeud


    def insererDroite(self,valeur):
        if self.droite==None:
            self.droite=ArbreBinaire(valeur)
        else:
            nouveauNoeud = ArbreBinaire(valeur)
            nouveauNoeud.droite = self.droite
            self.droite=nouveauNoeud

    def affiche(self):
        """permet d'afficher un arbre avec None si le noeud n'a pas d'enfants """
```

```
        if self==None: # si l'arbre est vide
            return None
        else :
            return [self.valeur,ArbreBinaire.affiche(self.gauche),ArbreBinaire.affiche(self.droite)]

    def taille(self):
        """donne la taille d'un arbre cad le nombre de feuilles """
        if self==None:
            return 0
        else :
            return 1+ArbreBinaire.taille(self.gauche)+ArbreBinaire.taille(self.droite)

    def hauteur(self):
        """donne la haureur de l'arbre (la racine n'étant pas comptée)"""
        if self==None:
            return 0
        elif self.gauche==None and self.droite==None:
            return 0
        else :
            return 1+max(ArbreBinaire.hauteur(self.gauche),ArbreBinaire.hauteur(self.droite))
    def getValeur(self):
        return self.valeur

    def getGauche(self):
        return self.gauche

    def getDroite(self):
        return self.droite
```

Exercice 1 :

— Déterminer la racine ; le nombre de feuilles ; le nombre de branches ; l'arité ; sa taille : T(B) ; la hauteur de B : H(B) ; LC(B) ; LCE(B) ; LCI(B) ; PM(B) ; PME(B) ; PMI(B) pour l'arbre ci-dessous.



— Implémenter cet arbre.

— Tester les différentes méthodes sur cet arbre.

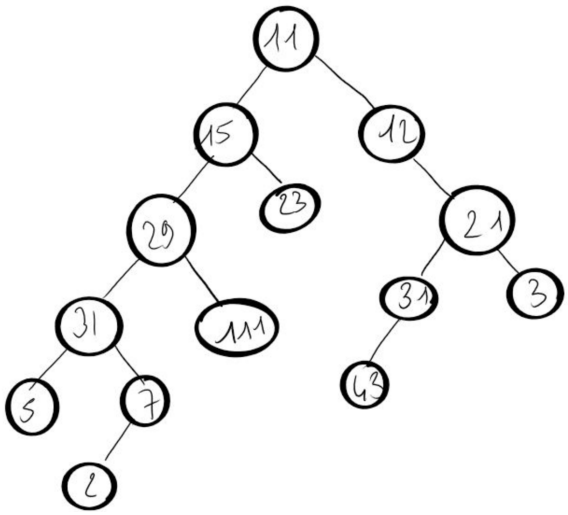
2) Différents parcours

Nous allons écrire une fonction qui affiche les valeurs contenues dans tous les noeuds de l'arbre. l'ordre dans lequel le parcours est effectué devient important.

2.1) Parcours infixe

Le parcours infixe consiste à parcourir le sous-arbre gauche puis afficher la valeur de la racine puis enfin parcourir le sous-arbre droit.

Exercice 2 : Réaliser à la main le parcours infixe de cet arbre :



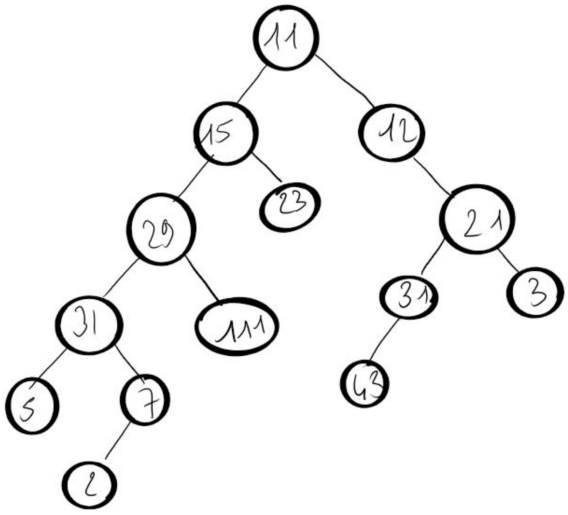

```
def parcours_infixe(a:ArbreBinaire):
    """affiche les éléments de a dans un parcours infixe"""
    if a ==None:
        return  None
    parcours_infixe(a.gauche)
    print(a.getValeur(),end=' ')
    parcours_infixe(a.droite)
```

Efficacité : La fonction `parcours_infixe` directement proportionnelle au nombre de noeud de l'arbre. Tout comme les méthodes `taille` et `hauteur`, elle fait un nombre d'opérations fini sur chaque noeud (ici l'affichage de la valeur) et parcourt une fois et une seule chaque noeud.

2.2) Parcours préfixe

Dans ce parcours on note tous les nœuds en commençant par la racine puis par le sous-arbre gauche puis le sous-arbre droit.

Exercice 3 : Réaliser à la main le parcours préfixe de cet arbre :

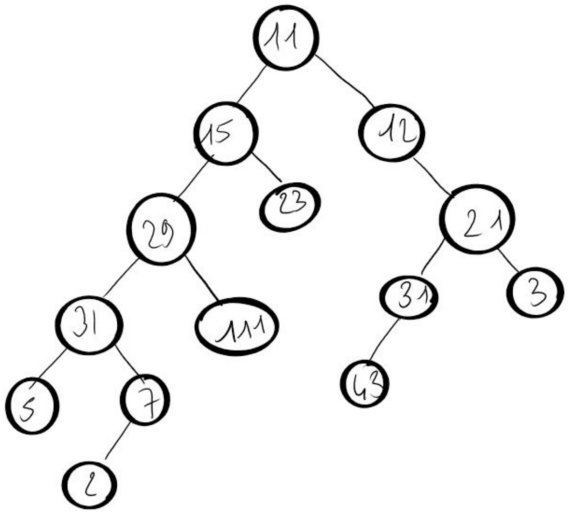



```
def parcours_prefixe(a:ArbreBinaire):
    """affiche tours les éléments de a dans un parcours préfixe"""
    if a==None:
        return None
    else :
        print(a.getValeur(),end=' ')
        parcours_prefixe(a.gauche)
        parcours_prefixe(a.droite)
```

2.3) Parcours suffixe

Dans ce parcours on note tous les nœuds en commençant par le sous arbre gauche puis le sous arbre droit et enfin la racine .

Exercice 4 : Réaliser à la main le parcours sufixe de cet arbre :



Exercice 5 :

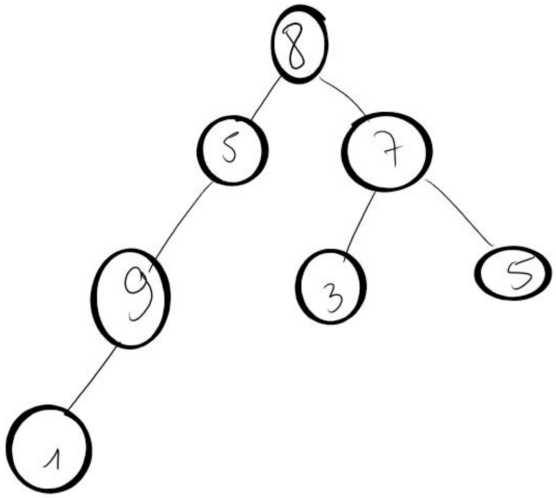
Écrire la fonction du parcours suffixe

2.4) Le parcours en largeur d'abord

Quand on parcourt en largeur un arbre, on note chaque sommet niveau par niveau et en commençant par la gauche.

Ce parcours est parfois noté *BPS* pour Breadth-First Search.

Exemple : le parcours en largeur de cet arbre donne 8-5-7-9-3-5-1



Pour réaliser l'implémentation de ce parcours, nous avons besoin de définir deux classes :

— une classe élément

— une classe file

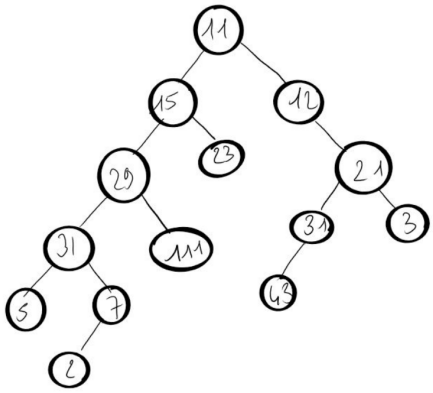
```
class Element:
    #chaque élément a pour attribut : le précédent , le suivant et la valeur de l'élément
    def __init__(self,x):
        self.val=x
        self.precedent=None
        self.suivant=None
    def __str__(self): #methode qui permet de lancer un print sur un tel objet
        return str(self.val)+"-"+str(self.suivant)
```

```
class File:
    #ici une file est la donnée de deux attributs : la file complète de type Element et le dernier
    #élément de la file de type Element
    def __init__(self):
        self.tete=None
        self.queue=None
    def enfile(self,x):
        e=Element(x) #on transforme l'élément à ajouter en un objet Element de listes doublement Chainées
        if self.tete==None:
            self.tete=e #file vide la tete est remplacée par l'élément e
        else:
            e.precedent=self.queue #le précédent de l'élément pointe sur l'ancienne queue de la file
            self.queue.suivant=e #l'ancienne queue de la file pointe sur e avec suivant
        self.queue=e #on redéfinit self.queue par e.
```

```
    def file_vide(self):
        return self.tete is None #renvoie True si None et False sinon

    def defile(self):
        if not self.file_vide():
            e=self.tete #on stocke l'élément à defiler
            if e.suivant is None: #cas où il n'y a qu'un élément
                self.tete=None
                self.queue=None
            else:
                self.tete=e.suivant
                self.tete.precedent=None
            return e.val

    def __str__(self):
        return str(self.tete)
```

```
def BFT(arbre):
    f=File()
    f.enfile(arbre)
    l=[]
    while not f.file_vide():
        a=f.defile()
        l.append(a.valeur)
        if a.gauche!=None:
            f.enfile(a.gauche)
        if a.droite!=None:
            f.enfile(a.droite)
    return l
```

Exercice 6 : Tester l'algorithme sur cet arbre.



3) Algorithmes sur les arbres binaires de recherche

3.1) Implémentation d'un arbre binaire de recherche

L'implémentation des arbres binaires de recherche peut être la suivante :

```
class ABR:
    def __init__(self,val):
        self.valeur=val
        self.gauche=None
        self.droite=None


    def inserer(self,x):
        if x<self.valeur:
            if self.gauche!=None:# si il y a un noeud à gauche
                self.gauche.inserer(x)
            else:
                self.gauche=ABR(x)
        else:
            if self.droite!=None:
                self.droite.inserer(x)
            else:
                self.droite=ABR(x)

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None: # si l'arbre est vide
            return None
        else :
            return [self.valeur,ABR.affiche(self.gauche),ABR.affiche(self.droite)]

    def taille(self):
        """donne la taille d'un arbre cad le nombre de feuilles """
        if self==None:
            return 0
        else :
            return 1+ABR.taille(self.gauche)+ABR.taille(self.droite)

    def hauteur(self):
        if self==None:
            return 0
        elif self.gauche==None and self.droite==None:
            return 0
        else :
            return 1+max(ABR.hauteur(self.gauche),ABR.hauteur(self.droite))

    def getValeur(self):
        return self.valeur
```

3.2) Recherche d'une clé

Exercice 7 :

Écrire la méthode `recherche(self),val)` qui renvoie `True` si `val` est une valeur de l'arbre et `False` sinon.

Remarque : le principe est celui de la dichotomie, on élimine grâce à la structure des arbres binaires de recherche la moitié des noeuds restant à chaque étape.

3.3) Insertion d'une clé

Exercice 8 : La méthode qui insère un élément dans un arbre binaire de recherche est déjà écrite. Quelle est-elle ? Commenter cette méthode.

3.4) Recherche d'extremum

Exercice 9 : Écrire une méthode `minimum(self)` qui renvoie la clé minimale d'un arbre.

Exercice 10 : Écrire une méthode `maximum(self)` qui renvoie la clé maximale d'un arbre.

3.5) Exercices sur les arbres

Exercice 11 : Donner tous les arbres binaires de recherche formés des trois nœuds : 7, 52, 40

Exercice 12 : Écrire une fonction listeEnArbre(l) qui en paramètre reçoit une liste d'entiers et qui renvoie un arbre binaire de recherche contenant les éléments de la liste l.