Chapitre 4

Arbre

Version avec preuves.

Table des matières

1 Définition **2**
1.1 Généralité sur les arbres 2
1.2 Arbre binaire 3

2 Parcours d'un arbre binaire **4**
2.1 Parcours en pré-ordre (préfixe) 4
2.2 Parcours en ordre (infixe) 5
2.3 Parcours en post-ordre (postfixe) 5
2.4 Parcours militaire (parcours en largeur d'abord) 5

3 Propriétés des arbres binaires **6**

1 Définition

1.1 Généralité sur les arbres

Définition 1 :

On appelle **arbre général** un ensemble non vide et fini d'objets appelés nœuds et liés par une "relation de parenté" notée $\mathcal{F}$ définie par :

$x\mathcal{F}y \Leftrightarrow x$ est un fils de y

$\Leftrightarrow y$ est le père de x

et une relation étendue notée $\mathcal{F}^*$:

$x\mathcal{F}^*y \Leftrightarrow x$ est un descendant de y

$\Leftrightarrow y$ est un ascendant x

$\Leftrightarrow x = y$ ou il existe un chemin : $(x_0 = x)\,\mathcal{F}x_1\mathcal{F}\ldots\mathcal{F}\,(x_n = y)$

Et vérifiant les propriétés suivantes :

- Il existe un unique élément n'ayant pas de père, appelé racine de l'arbre.
- Tout élément à part la racine a un et un seul père.
- Tout élément est un descendant de la racine.

Définition 2 :

Autres définitions

- Un nœud n'ayant pas de fils est appelé **feuille** ou **nœud terminal** et un nœud ayant au moins un fils est appelé **nœud interne** ou **nœud intérieur**.
- Les fils d'un même père sont appelés **frères**.
- Le **degré** d'un nœud est son nombre de fils, le **degré maximal** de l'arbre est le plus grand des degrés des nœuds.
- La **profondeur** d'un nœud est le nombre d'ascendant strict de ce nœud, la **hauteur** d'un arbre est la profondeur maximale de ses nœuds.
- Le nombre de nœuds dans un arbre est la **taille** de l'arbre.
- L'ensemble des descendants d'un nœud x forme un **sous-arbre** de racine x.
- Une **forêt** est un ensemble fini d'arbre sans nœuds communs ; l'ensemble des descendants stricts d'un nœud x forme une forêt, vide si le nœud est terminal. Les arbres de cette forêt sont appelés branches issues de x.
- Un arbre **ordonné** est un arbre pour lequel l'ensemble des branches issues d'un nœud est totalement ordonné.
- Un arbre est **étiqueté** lorsqu'à chaque nœud est associée une information appelé **étiquette** du nœud. Des nœuds distincts peuvent porter la même étiquette.

1.2 Arbre binaire

Définition 3 :

Un **arbre binaire** est un ensemble fini, éventuellement vide, de nœuds liés par une relation de parenté orientée notée $\mathcal{F}_d$ et $\mathcal{F}_g$ définie par :

$x\mathcal{F}_d y \Leftrightarrow x$ est le fils droit de y
$\Rightarrow y$ est le père de x

et

$x'\mathcal{F}_g y \Leftrightarrow x'$ est le fils gauche de y
$\Rightarrow y$ est le père de x'

Et vérifiant les propriétés suivantes :

- Si l'arbre est non vide alors il existe un unique élément n'ayant pas de père appelé racine de l'arbre.
- Tout élément, à part la racine, a un et un seul père.
- Tout élément a au plus un fils droit et au plus un fils gauche.
- Tout élément est descendant de la racine.

Remarque : un arbre binaire est :

- soit vide
- soit composé d'un élément, d'un arbre fils gauche et d'un arbre fils droit, chacun des deux fils pouvant éventuellement être un arbre vide.

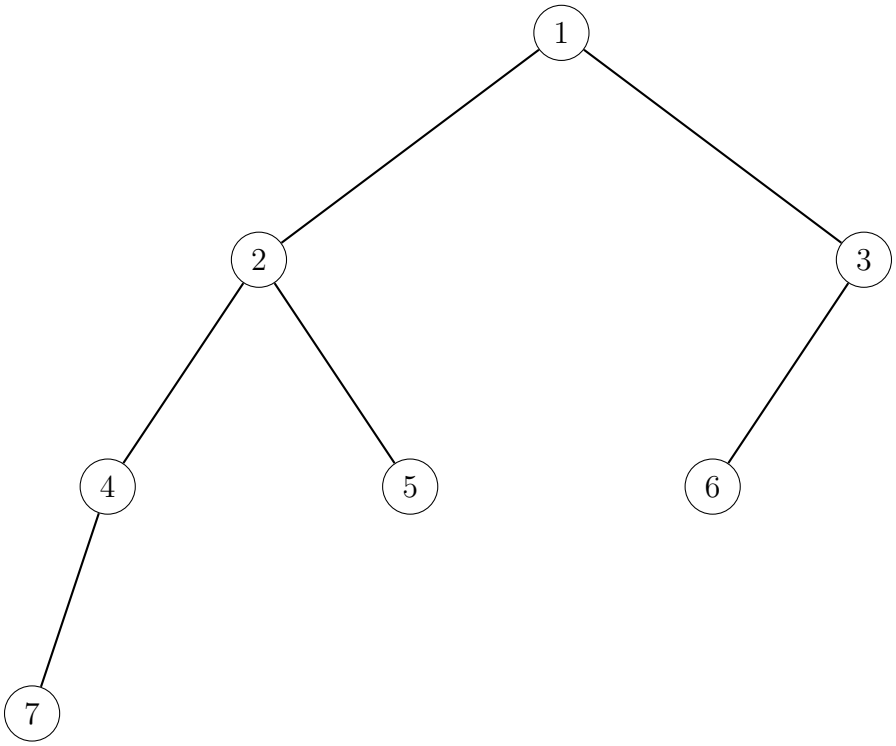
Implémentation d'un arbre binaire :

```
class Arb_bin :
    def __init__(self, racine = None, arb_gauche = None , arb_droit = None ) :
        self.racine = racine
        self.arb_gauche = arb_gauche
        self.arb_droit = arb_droit
```

Exemple :

```
>>> v = Arb_bin() # arbre vide
>>> a = Arb_bin(8,v,v)
>>> b = Arb_bin(7,v,v)
>>> c = Arb_bin(15,a,b)
>>> d = Arb_bin(18,c,v)
```

Exercice 1 : Définir l'arbre **exemple_1** suivant :



Exercice 2 :

1. Ecrire la méthode `test_vide` qui renvoie le booléen correspondant à l'arbre vide.
2. Ecrire les méthodes getter pour la racine, le sous-arbre gauche, et le sous-arbre droit.
3. Ecrire la méthode `taille` qui renvoie la taille d'un arbre binaire.
4. Ecrire la méthode `hauteur` qui renvoie la hauteur d'un arbre binaire.

2 Parcours d'un arbre binaire

2.1 Parcours en pré-ordre (préfixe)

Définition 4 :

Parcourir un arbre binaire non vide en pré-ordre consiste à traiter la racine de cet arbre, puis à parcourir en pré-ordre l'arbre fils gauche de cette racine et enfin, à parcourir en pré-ordre l'arbre fils droit de cette racine. (parcourir un arbre binaire vide en pré-ordre est une opération nulle (pas de traitement)).

Exercice 3 : Parcourir en pré-ordre l'arbre **exemple_1**.

Exercice 4 : En utilisant l'implémentation précédente, écrire la méthode `prefixe` qui renvoie, sous forme d'une chaine de caractère, le parcours en pré-ordre de l'arbre.

2.2 Parcours en ordre (infixe)

Définition 5 :

Parcourir un arbre binaire non vide en ordre (ou de façon infixe) consiste à parcourir en ordre l'arbre fils gauche de cet arbre, puis à traiter la racine, et enfin, à parcourir en ordre l'arbre fils droit de cette racine. (parcourir un arbre binaire vide en pré-ordre est une opération nulle (pas de traitement)).

Exercice 5 : Parcourir en ordre l'arbre **exemple_1**.

Exercice 6 : En utilisant l'implémentation précédente, écrire la méthode `infixe` qui renvoie, sous forme d'une chaine de caractère, le parcours en ordre de l'arbre.

2.3 Parcours en post-ordre (postfixe)

Définition 6 :

Parcourir un arbre binaire non vide en post-ordre (ou de façon postfixe) consiste à parcourir en post-ordre l'arbre fils gauche de cette arbre, puis à parcourir en post-ordre l'arbre fils droit, et enfin à traiter la racine. (parcourir un arbre binaire vide en pré-ordre est une opération nulle (pas de traitement)).

Exercice 7 : Parcourir en post-ordre l'arbre **exemple_1**.

Exercice 8 : En utilisant l'implémentation précédente, écrire la méthode `postfixe` qui renvoie, sous forme d'une chaine de caractère, le parcours en post-ordre de l'arbre.

2.4 Parcours militaire (parcours en largeur d'abord)

Définition 7 :

Il faut ajouter l'ordre militaire, qui consiste à lire profondeur par profondeur, de gauche à droite. C'est un parcours en largeur d'abord.

Exercice 9 : Parcourir en largeur l'arbre **exemple_1**.

Exercice 10 : En utilisant l'implémentation précédente, écrire la méthode `militaire` qui renvoie, sous forme d'une chaine de caractère, le parcours en largeur d'abord de l'arbre.

Correction

```
# exercice 6 ( sur les files )

def cree_file() :
    return []

def ajout(f,element):
    f.append(element)

def file_vide(f) :
    return f== []  # ou len(f) == 0

def retire(f) :
    if file_vide(f) :
        print("la file est vide")
    else :
        a = f.pop(0)
        return a



class Arb_bin :
    def __init__(self,racine = None,arb_gauche =  None , arb_droit = None):
        self.racine= racine
        self.arb_gauche=arb_gauche
        self.arb_droit = arb_droit

    def test_vide (self ):
        return self.racine == None and \
               self.arb_gauche == None and \
                self.arb_droit == None

    def get_racine(self) :
        return self.racine


    def get_arb_gauche(self) :
        return self.arb_gauche

    def get_arb_droit(self) :
        return self.arb_droit

    def taille(self) :
        if self.test_vide() :
            return 0
        else :
            taille_1 = self.arb_gauche.taille()
            taille_2 = self.arb_droit.taille()
            return taille_1 + taille_2 +1
```

```
    def hauteur(self) :
        if self.test_vide() :
            return -1
        else :
            h_g = self.arb_gauche.hauteur()
            h_d = self.arb_droit.hauteur()
            return max(h_g,h_d) +1

    def prefixe(self) :
        if self.test_vide() :
            return ""
        else :
            parcours_gauche = self.arb_gauche.prefixe()
            parcours_droit =self.arb_droit.prefixe()
            rac = self.racine
            return str(rac) +parcours_gauche +  parcours_droit

#exercice 6
    def infixe(self) :
        if self.test_vide() :
            return ""
        else :
            parcours_gauche = self.arb_gauche.infixe()
            parcours_droit =self.arb_droit.infixe()
            rac = self.racine
            return parcours_gauche + str(rac)  +  parcours_droit

#exercice 8
    def postfixe(self) :
        if self.test_vide() :
            return ""
        else :
            parcours_gauche = self.arb_gauche.postfixe()
            parcours_droit =self.arb_droit.postfixe()
            rac = self.racine
            return parcours_gauche  +  parcours_droit + str(rac)


#exercice 10
    def militaire(self) :
        f = cree_file()
        ajout(f,self)
        def traitement() :
            if file_vide(f) :
                return ""
            else :
                arbre = retire(f)
                if arbre.test_vide() :
                    return traitement()
                else :
```

```
                    rac = arbre.racine
                    arbre_gauche = arbre.arb_gauche
                    arbre_droit = arbre.arb_droit
                    ajout(f,arbre_gauche)
                    ajout(f,arbre_droit)
                    return str(rac) + traitement()
        return traitement()


v = Arb_bin()

exemple_1 = Arb_bin(1 , Arb_bin(2 , \
    Arb_bin(4 , Arb_bin(7,v,v), v) , \
Arb_bin(5,v,v)),Arb_bin( 3,Arb_bin(6,v,v),v))
```

3 Propriétés des arbres binaires

Propriété 1 :

- Un arbre binaire à n nœuds possède $n+1$ branches vides.
- Dans un arbre binaire à n nœuds, le nombre de nœuds sans fils est inférieur ou égal à $\frac{n+1}{2}$. Il y a égalité si et seulement si tous les nœuds ont zéro ou deux fils.
- Pour h la hauteur d'un arbre binaire non vide à n nœuds, on a la relation :

$$\lfloor \log(n) \rfloor \leqslant h \leqslant n-1$$

Preuve :