Math93.com

Cours - NSI
Les arbres binaires (Correction)

Vous écrirez tous les codes de ce chapitre dans le même fichier.

1 Définition

Définition 1

Un arbre binaire est un arbre de degré 2 au plus (dont les noeuds sont de degré 2 au plus).
Vocabulaires :

- Les enfants d'un noeud sont lus de gauche à droite et sont appelés : fils gauche et fils droit.
- Les noeuds qui n'ont pas de fils sont des feuilles.
- Chaque noeud qui n'est pas une feuille possède un sous-arbre gauche et un sous-arbre droit.

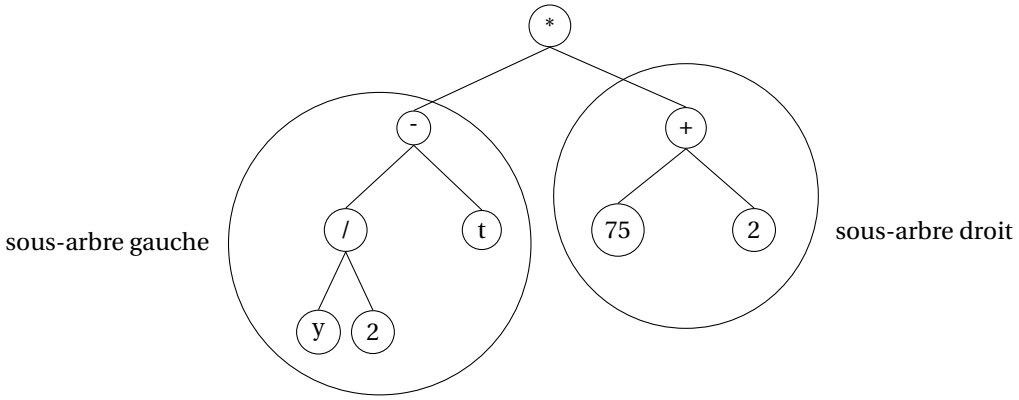


FIGURE 1 – Arbre d'une expression mathématiques $\left(\frac{y}{2}-t\right)(75+t)$ utilisant la priorité des opérations

Remarque

Les arbres binaires forment une structure de données qui peut se définir de façon récursive. Un arbre binaire est :

- Soit vide.
- Soit composé d'une racine portant une étiquette (clé) et d'une paire d'arbres binaires, appelés sous-arbres gauche et droit

Arbre binaire

1. On appelle **taille d'un arbre** le nombre de noeuds présents dans cet arbre.
2. Dans un **arbre binaire**, un noeud possède au plus 2 fils.
3. On appelle **profondeur d'un nœud** ou d'une feuille dans un arbre binaire
   - Soit le nombre de nœuds du chemin qui va de la racine à ce nœud. La racine d'un arbre de taille 1 est à une profondeur 1.
   - Soit le nombre de branches qui va de la racine à ce nœud. La racine d'un arbre de taille 1 est à une profondeur 0.
4. On appelle **hauteur d'un arbre** la profondeur maximale des nœuds de l'arbre.

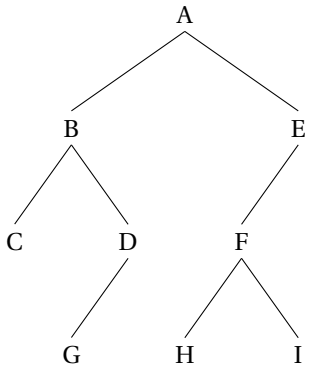
2 Représentation par un tableau (Une liste en python)

Exercice 1 , Sujet 0 Bac , 2021

Dans cet exercice, on utilisera la convention suivante : la hauteur d'un arbre binaire ne comportant qu'un nœud est 1.

Question 1

Déterminer la taille et la hauteur de l'arbre binaire suivant :



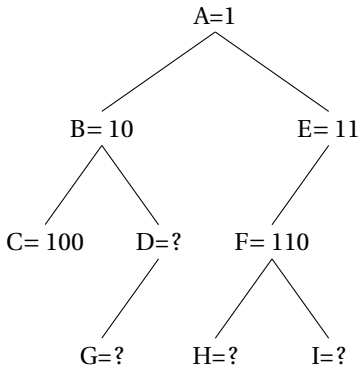
Corrigé

- Par définition, la taille d'un arbre est le nombre de noeud qu'il contient. Ici il y en a 9 donc la taille est 9.
- Par définition la hauteur est le nombre de noeuds du chemin le plus long dans l'arbre, ici les chemins les plus long sont ABDG, AEFH et AEFI. Comme la hauteur d'un arbre ne contenant qu'un noeud est 1, la hauteur de cet arbre est 4.

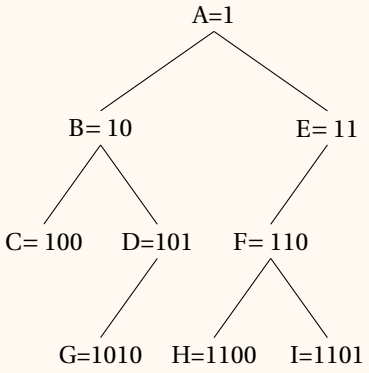
Question 2

On décide de numéroter en binaire les nœuds d'un arbre binaire de la façon suivante :
— la racine correspond à 1 ;
— la numérotation pour un fils gauche s'obtient en ajoutant le chiffre 0 à droite au numéro de son père ;
— la numérotation pour un fils droit s'obtient en ajoutant le chiffre 1 à droite au numéro de son père.

1. **Dans l'exemple précédent, quel est le numéro en binaire associé au nœud G ?**



Corrigé



On a donc G :1010.

2. **Quel est le nœud dont le numéro en binaire vaut 13 en décimal?**

Corrigé

$13_{10} = 1101_2$ or I :1101 donc cela correspond au noeud I.

3. **En notant h la hauteur de l'arbre, sur combien de bits seront numérotés les nœuds les plus en bas?**

Corrigé

A chaque niveau de l'arbre on rajoute 1 bit donc les numéros des noeuds les plus bas (les feuilles) contiennent h bits.

4. **Justifier que pour tout arbre de hauteur h et de taille $n \geq 2$, on a : $h \leq n \leq 2^h - 1$.**

Corrigé

- Soit un arbre de hauteur h. Il contient un maximum de noeuds si il est complet. Or un arbre binaire complet de hauteur h contient $1+2+2^2+...+2^{h-1}$ noeuds. C'est la somme d'une suite géométrique de raison 2 donc :

$$1+2+2^2+...+2^{h-1} = \frac{2^h-1}{2-1} = 2^h-1$$

  On a donc $n \leq 2^h - 1$.
- Soit un arbre de hauteur h. La hauteur maximale que l'on peut obtenir avec un minimum de noeuds est le cas où chaque père n'a qu'un seul fils . Dans ce cas la hauteur est égale à n. Donc $h \leq n$.
- Donc $h \leq n \leq 2^h - 1$

Question 3

Un arbre binaire est dit complet si tous les niveaux de l'arbre sont remplis. On décide de représenter un arbre binaire complet par un tableau de taille $n+1$, où n est la taille de l'arbre, de la façon suivante :

- La racine a pour indice 1 ;
- Le fils gauche du nœud d'indice i a pour indice $2 \times i$;
- Le fils droit du nœud d'indice i a pour indice $2 \times i + 1$;
- On place la taille n de l'arbre dans la case d'indice 0.

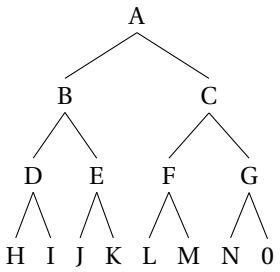
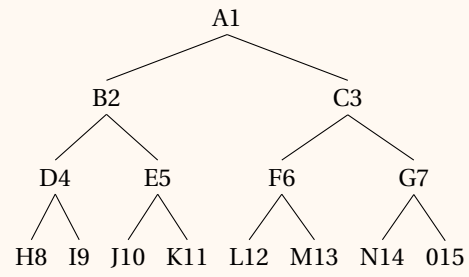


FIGURE 2 – Arbre binaire complet

- Dans le cas où l'arbre n'est pas complet, on fait comme s'il était complet et on remplace la valeur des noeuds manquants par None.

1. **Déterminer le tableau qui représente l'arbre binaire complet de l'exemple de la figure 2 précédent.**

Corrigé



On a donc le tableau

$$[15, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O]$$

2. **Déterminer le tableau qui représente l'arbre binaire incomplet de l'exemple de la figure 3 suivant.**

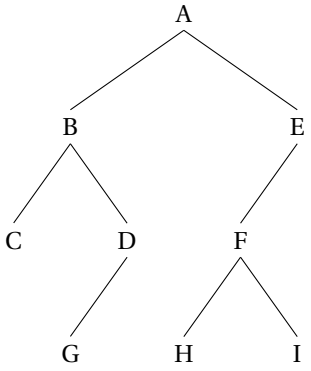
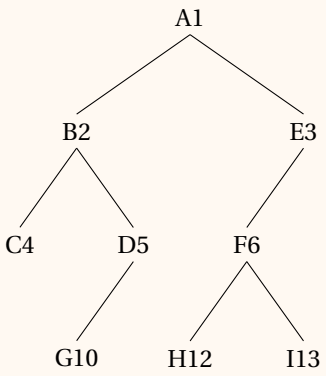
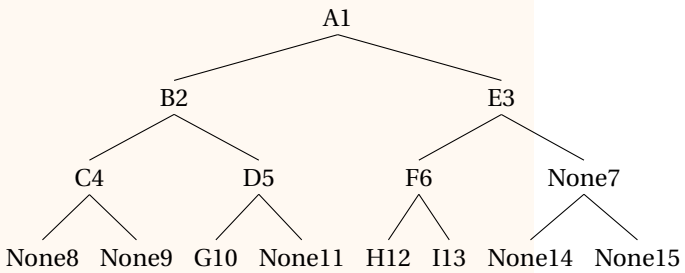


FIGURE 3 – Arbre binaire incomplet

Corrigé





On a donc le tableau

$$[9, A, B, E, C, D, F, None, None, None, G, None, H, I, None, None]$$

3. **On considère le père du nœud d'indice i avec $i \geq 2$. Quel est son indice dans le tableau ?**

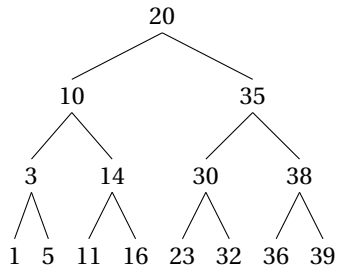
Corrigé

Le père d'un fils d'indice i a pour indice $i/2$ si i est pair et $(i-1)/2$ sinon.

Sous python on peut dans ce cas utiliser la fonction // .
$a//b$ donne le quotient de la division euclidienne de a par b .

Question 4 : Arbre binaire de recherche

On se place dans le cas particulier d'un arbre binaire de recherche complet où les nœuds contiennent des entiers et pour lequel la valeur de chaque noeud est supérieure à celles des noeuds de son arbre gauche, et inférieure à celles des noeuds de son arbre droit.
On a par exemple un arbre de ce type :



$$A = [15, 20, 10, 35, 3, 14, 30, 38, 1, 5, 11, 16, 23, 32, 36, 39]$$

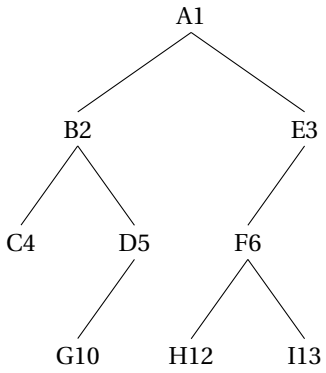
Écrire une fonction recherche ayant pour paramètres un arbre *arbre* et un élément *element*.
Cette fonction renvoie *True* si *element* est dans l'arbre et *False* sinon. L'arbre sera représenté par un tableau comme dans la question précédente. Attention, il faut utiliser la structure des arbres binaires de recherche et ne pas visiter tous les noeuds.

Corrigé

```
def recherche(arbre,element):
    '''In : arbre et element entier
      Out : true si element est dans la liste'''
    taille = arbre[0]
    i=1
    while i<=taille:
        if element==arbre[i]:
            return True
        elif element>arbre[i]:
            i=2*i+1
        else:
            i=2*i
    return False
```

Exercice 2

C'est un exercice sur machine. Voici un code Python qui crée la liste représentant l'arbre de l'exercice 1 question 3.2



Corrigé

https://replit.com/@fduffaud/TP-arbres-binaires-Exercice-2

```
def creation_arbre(r,profondeur):
    """ r : la racine (str ou int). la profondeur de l'arbre (int)"""
    if profondeur==0:
        return [0]
    Arbre = [2**profondeur-1,r]+[None for i in range(2**(profondeur)-2)]
    return Arbre
def insertion_noeud(arbre,n,fg,fd):
    """Insére les noeuds et leurs enfants dans l'arbre"""
    indice = arbre.index(n)
    arbre[2*indice] = fg
    arbre[2*indice+1] = fd
# création de l'arbre
arbre = creation_arbre("A",4)
# ajout des noeuds par niveau de gauche à droite
insertion_noeud(arbre,"A","B","E")
insertion_noeud(arbre,"B","C","D")
insertion_noeud(arbre,"D","G",None)
insertion_noeud(arbre,"E","F",None)
insertion_noeud(arbre,"F","H","I")
#pour vérifier
print(len(arbre))
print(arbre)
```

1. Écrire le code précédent puis compléter la fonction suivante qui retourne le parent d'un noeud s'il est dans l'arbre :

```
def parent(arbre,p):
    if p in arbre:
        indice =...
        if indice==1:
            return ...
        if ...:
            return ...
        else:
            return ...
```

2. Créer les fonctions suivantes :
   (a) Une fonction qui retourne True si l'arbre est vide.
   (b) Une fonction qui retourne les enfants d'un noeud.
   (c) Deux fonctions qui retournent le fils gauche d'un noeud et son homologue le fils droit s'ils existent.
   (d) Une fonction qui retourne True si le noeud est la racine de l'arbre.
   (e) Une fonction qui retourne True si le noeud est une feuille.
   (f) Une fonction qui retourne True si le noeud a un frère gauche ou droit.

3 Représentation par un tableau de tableau (Une liste de listes en python)

Chaque arbre est un tableau contenant la racine, un tableau contenant le sous-arbre gauche et un tableau contenant le sous-arbre droit. On remarque que l'on utilise clairement la structure récursive d'un arbre binaire. Un arbre vide est une liste vide.

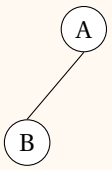
Exemple

Pour un arbre contenant qu'un seul noeud, on le représente par

["A" ,[ ] , [ ]]

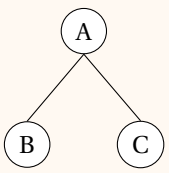
Pour un arbre contenant deux noeuds, on le représente par

["A" ,["B" , [ ] , [ ]] , [ ]]



Pour un arbre contenant trois noeuds, on le représente par

["A" ,["B" , [ ] , [ ]] ,["C" , [ ] , [ ]]]

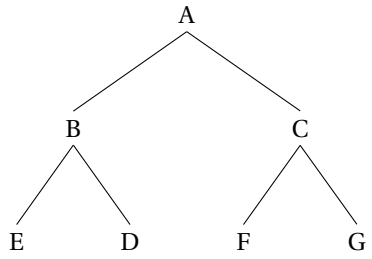


En résumé un arbre est un tableau de trois éléments contenant l'étiquette de la racine, le sous-arbre gauche et le sous arbre droit.

$$[\ Etiquette\ , Sous-arbre\ gauche, Sous-arbre\ droit]$$

Exercice 3

Faire le tableau représentant l'arbre suivant :



Corrigé

```
[ "A" ,[ "B" , [ "E" , [ ] , [ ] ] , [ "D" , [ ] , [ ] ] ] ,[ "C" , [ "F" , [ ] , [ ] ] , [ "G" , [ ] , [ ] ] ] ]
```

Exercice 4

Le code Python ci-dessous, construit l'arbre ci-dessous de manière récursive.

- Les noeuds sont représentés par un dictionnaire.
- L'arbre se construit depuis la racine en construisant les sous-arbres des fils gauche et droit.

Sur machine, écrire le code précédent puis créer une fonction qui retourne True si l'arbre est vide

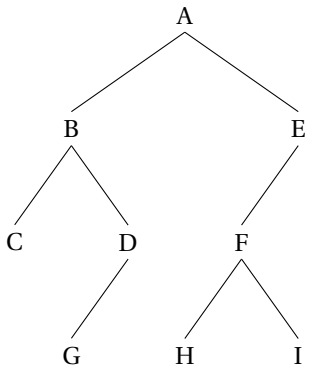


FIGURE 4 – Mon arbre

```
def noeud(nom, fg = None, fd = None) :
    return {"racine": nom, "fg" : fg, "fd": fd}
def construit(arbre) :
    if arbre == None :
        return []
    else:
        return [arbre["racine"],construit(arbre["fg"]),construit(arbre["fd"])]
# création des noeuds
c = noeud("C")
g = noeud("G")
h = noeud("H")
i = noeud("I")
d = noeud("D",g,None)
f = noeud("F", h, i)
b = noeud("B",c,d)
e = noeud("E", f, None)
racine = noeud("A", b, e)
# création de l'arbre

arbre1=construit(racine)
print(arbre1)
```

Corrigé

On a créé une fonction qui retourne True si l'arbre est vide

```
def est_vide(arbre):
    return arbre == []
```

4 Hauteur, parcours en largeur, parcours en profondeur

4.1 Calcul de la hauteur : un algorithme récursif

Dans la suite, on utilise la définition :

Rappel Définition (Hauteur ou profondeur d'un noeud , **première définition**)

La **hauteur** (ou profondeur ou niveau) d'un noeud X est égale au nombre d'arêtes qu'il faut parcourir à partir de la racine pour aller jusqu'au noeud X.

Le principe est le suivant :

- Si l'arbre est vide la hauteur vaut -1 ;
- Sinon la hauteur vaut 1 auquel il faut ajouter le maximum entre les hauteurs des sous-arbres gauche et droit.
- Ces sous-arbres sont eux mêmes des arbres dont il faut calculer la hauteur.

Algorithme 1 Fonction hauteur(arbre)

ENTRÉE : Un arbre représenté comme tableau de tableaux
SORTIE : Un entier, hauteur de l'arbre
  Si L'arbre est vide **alors**
    renvoyer -1
  Sinon
    $h1 \longleftarrow$ 1+hauteur(arbre gauche)
    $h2 \longleftarrow$ 1+hauteur(arbre droit)
    Renvoyer max($h1, h2$)
  Fin Si

Exercice 5

Écrire cette fonction dans le code précédent et vérifier que la hauteur est 3

Corrigé

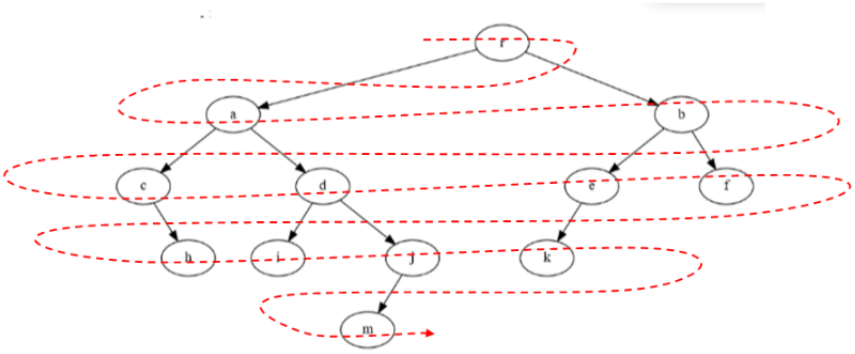
On a créé une fonction suivante :
https://replit.com/@fduffaud/arbres-binaires-Exercices-4-5

```
def hauteur(arbre):
    if arbre == []:
        return -1
    else:
        h1 = 1 + hauteur(arbre[1])
        h2 = 1 + hauteur(arbre[2])
        return max(h1, h2)

hauteur_arbre = hauteur(arbre1)
print(f"La hauteur de l'arbre est {hauteur_arbre}")
```

4.2 Parcours en largeur

Le parcours d'un arbre en largeur consiste à partir de la racine on visite ensuite son fils gauche puis son fils droit, puis le fils gauche du fils gauche etc.. Comme le montre le schéma ci-dessous :



On utilise une **File**.

- On met l'arbre dans la file.
- Puis tant que la file n'est pas vide :
- On défile la file.
- On récupère sa racine.
- On enfile son fils gauche s'il existe.
- On enfile son fils droit s'il existe.

Algorithme 2 Fonction parcoursLargeur(arbre)

ENTRÉE : arbre binaire
SORTIE : La liste des noeuds visités
f une file vide
h une file vide
On met l'arbre dans la file f
Tant que la file f n'est pas vide **faire**
On défile la file f dans une variable tmp
On ajoute tmp[0] (L'etiquette de la racine) à la file h
Si tmp[1] n'est pas vide **alors**
On enfile tmp[1] (le sous arbre gauche) dans f
Fin Si
Si tmp[2] (le sous arbre droit) n'est pas vide **alors**
On enfile tmp[2] (le sous arbre droit) dans f
Fin Si
Fin Tant que
Renvoyer h

Exercice 6

Implémenter cette fonction **parcoursLargeur(arbre)** en récupérant le code avec la classe File ci-dessous pour pouvoir utiliser une file... En reprenant l'arbre de la figure 4, vous devez obtenir : ['A', 'B', 'E', 'C', 'D', 'F', 'G', 'H', 'I']
On rappelle le code de la classe File ci-dessous.

```
class File:
    """ classe File
    création d"une instance File avec une liste """
    def __init__  (self):
        """Initialisation d"une file vide"""
        self.file=[]
    def vide(self):
        """teste si la file est vide"""
        return self.file==[]
    def defiler(self):
        """défile"""
        assert  not  self.vide(), "file vide"
        return self.file.pop(0)
    def enfiler(self,x):
        """enfile"""
        self.file.append(x)
    def afficher(self):
       print( str(self.file))
```

Corrigé

On a créé une fonction suivante :
https://replit.com/@fduffaud/arbres-binaires-Exercices-4-5

```
def parcoursLargeur(arbre):
    '''
    Parcours en largeur de l'arbre défini avec la classe arbre
    '''
    f = File()
    h = File()
    f.enfiler(arbre)
    while not f.vide():
        tmp = f.defiler()
        h.enfiler(tmp[0])
        if tmp[1]!=[]:
            f.enfiler(tmp[1])
        if tmp[2]!=[]:
            f.enfiler(tmp[2])
    return h
```

4.3 Parcours en profondeur

On se balade autour de l'arbre en suivant les pointillés.
Dans le schéma ci-dessus, on a rajouté des "branches fantômes" pour montrer que l'on peut considérer que chaque noeud est visité 3 fois :

- Une fois par la gauche.
- Une fois par en dessous.
- Une fois par la droite.

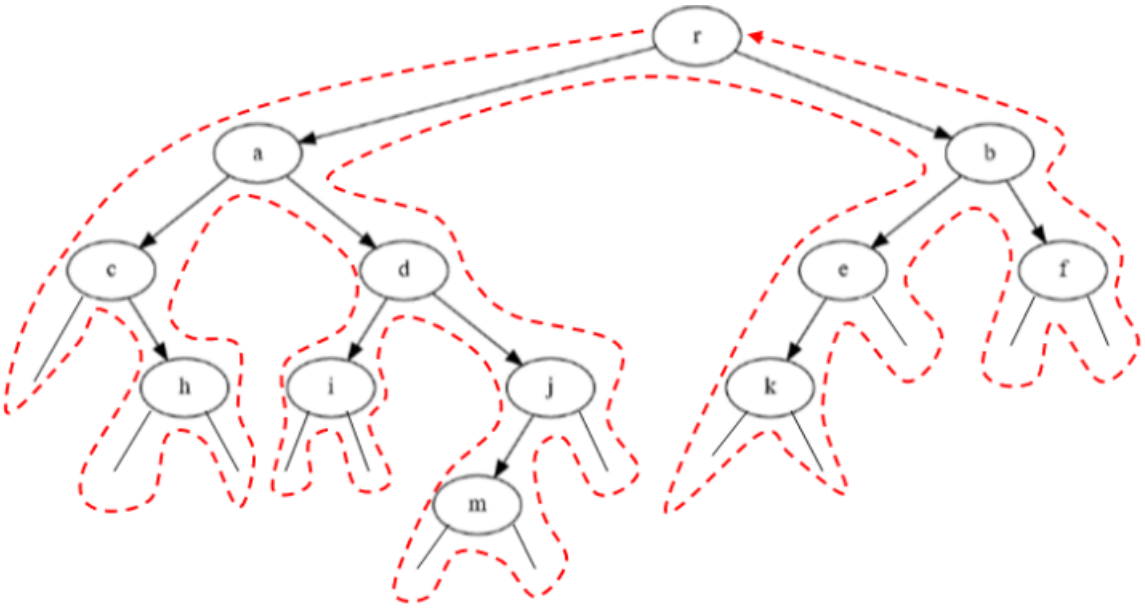


FIGURE 5 – Le parcours en profondeur

Définition 2 (Parcours préfixe)

Dans un parcours préfixe, on liste le noeud la première fois qu'on le rencontre.

Définition 3 (Parcours infixe)

Dans le sens antihoraire, dans un parcours infixe, on liste le noeud la seconde fois qu'on le rencontre.

- Ce qui correspond à : on liste chaque noeud ayant un fils gauche la seconde fois qu'on le voit et chaque noeud sans fils gauche la première fois qu'on le voit.

Définition 4 (Parcours suffixe ou postfixe)

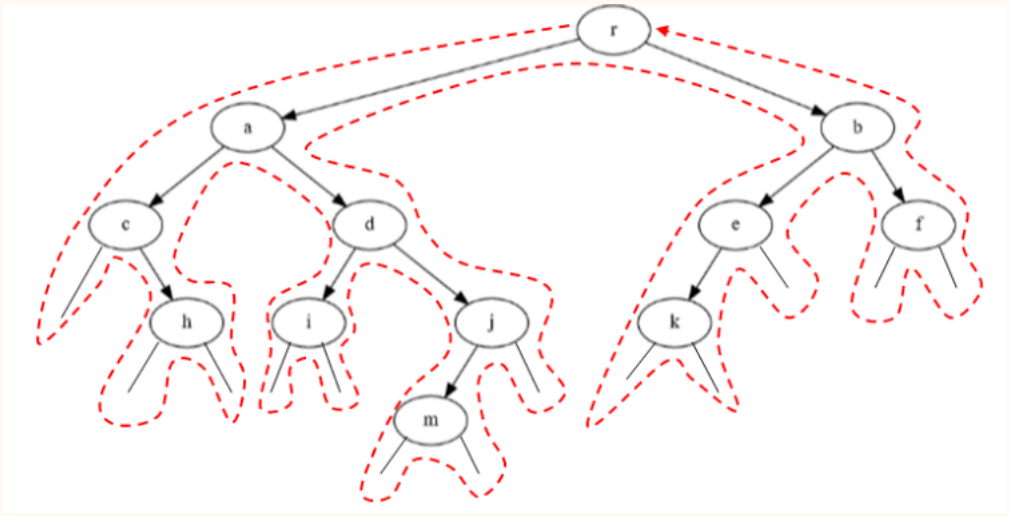
Dans un parcours suffixe ou postfixe, on note le noeud la dernière fois qu'on le rencontre.

Exercice 7

Écrire la suite des sommets, pour l'arbre de la Figure 5, dans chaque cas :

- Parcours préfixe :
- Parcours infixe :
- Parcours suffixe :

Corrigé



- Parcours préfixe : r , a , c , h , d , i , j , m , b , e , k , f .
- Parcours infixe : c , h , a , i , d , m , j , r , k , e , b , f .
- Parcours suffixe ou postfixe : h , c , i , m , j , d , a , k , e , f , b , r .

Exercice 8

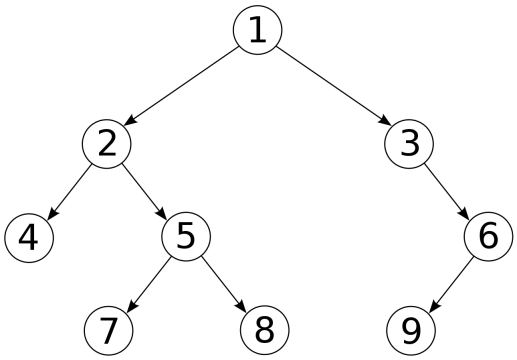
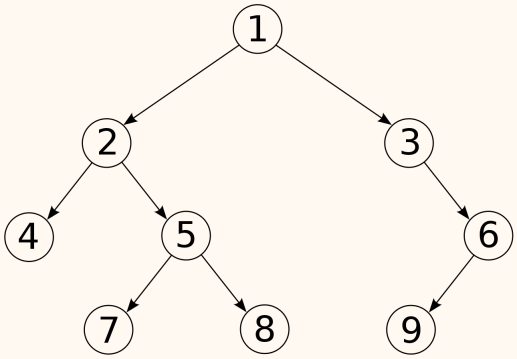


FIGURE 6 – Le parcours en profondeur

Écrire la suite des sommets, pour l'arbre de la Figure 6, dans chaque cas :

- Parcours préfixe :
- Parcours infixe :
- Parcours suffixe :
- Parcours en largeur :

Corrigé



- Parcours préfixe : 1, 2, 4, 5, 7, 8, 3, 6, 9
- Parcours infixe : 4, 2, 7, 5, 8, 1, 3, 9, 6
- Parcours suffixe ou postfixe : 4, 7, 8, 5, 2, 9, 6, 3, 1
- Parcours en largeur : 1, 2, 3, 4, 5, 6, 7, 8, 9

Exercice 9

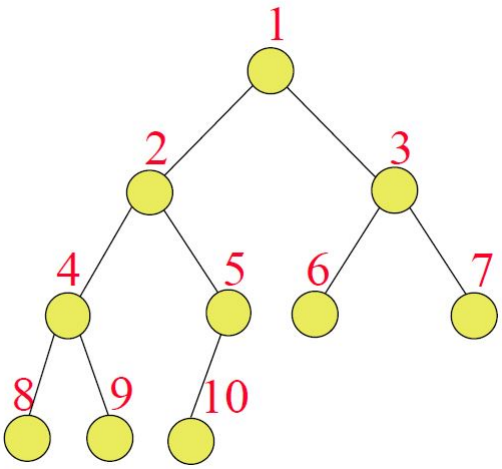
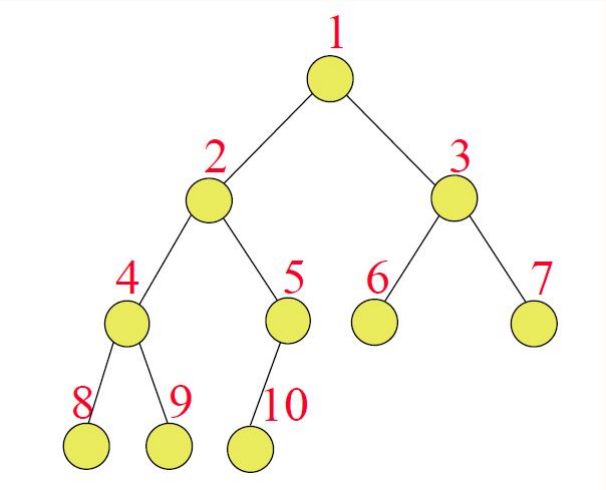


FIGURE 7 – Le parcours en profondeur

Écrire la suite des sommets, pour l'arbre de la Figure 7, dans chaque cas :

- Parcours préfixe :
- Parcours infixe :
- Parcours suffixe :
- Parcours en largeur :

Corrigé



- Parcours préfixe : 1, 2, 4, 8, 9, 5, 10, 3, 6, 7
- Parcours infixe : 8, 4, 9, 2, 10, 5, 1, 6, 3, 7
- Parcours suffixe ou postfixe : 8, 9, 4, 10, 5, 2, 6, 7, 3, 1
- Parcours en largeur : 1, 2, 3, 4, 5, 6, 7, 8, 9 , 10

Exercice 10

Voici trois algorithmes de parcours, associez chaque algorithme au type de parcours en profondeur

Algorithme 3 Fonction parcours(arbre)

```
ENTRÉE : arbre binaire
  Si L'arbre n'est pas vide alors
    parcours(sous-arbre gauche)
    parcours(sous-arbre droit)
    Afficher racine
  Fin Si
```

Algorithme 4 Fonction parcours(arbre)

```
ENTRÉE : arbre binaire
  Si L'arbre n'est pas vide alors
    parcours(sous-arbre gauche)
    Afficher racine
    parcours(sous-arbre droit)
  Fin Si
```

Algorithme 5 Fonction parcours(arbre)

```
ENTRÉE : arbre binaire
  Si L'arbre n'est pas vide alors
    Afficher racine
    parcours(sous-arbre gauche)
    parcours(sous-arbre droit)
  Fin Si
```

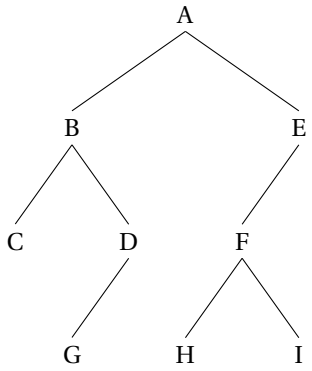
Corrigé

- Parcours préfixe : Algorithme 5

- Parcours infixe : Algorithme 4
- Parcours suffixe ou postfixe : Algorithme 3

Exercice 11

Implémenter les trois parcours précédents.
Vous pourrez les tester avec l'arbre déjà implémenté dans l'exercice 4, Figure 4



- Parcours préfixe : ['A', 'B', 'C', 'D', 'G', 'E', 'F', 'H', 'I']
- Parcours suffixe ou postfixe : ['C', 'G', 'D', 'B', 'H', 'I', 'F', 'E', 'A']
- Parcours infixe : ['C', 'B', 'G', 'D', 'A', 'H', 'F', 'I', 'E']

Exercice 12 (Complement)

Sur machine, écrire le code précédent puis créer les fonctions suivantes :

1. Une fonction qui retourne les enfants d'un noeud.
2. Deux fonctions qui retournent le fils gauche d'un noeud et son homologue le fils droit s'ils existent.
3. Une fonction qui retourne True si le noeud est la racine de l'arbre.
4. Une fonction qui retourne True si le noeud est une feuille.
5. Une fonction qui retourne True si le noeud a un frère gauche ou droit.

Corrigé

https://replit.com/@fduffaud/arbres-binaires-Exercices-4-5

5 Arbre binaire avec une classe

Nous allons créer une classe Noeud dont les attributs d'instances seront :

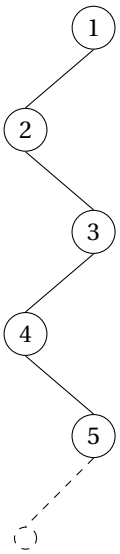
- Le nom (ou valeur) de la racine.
- Son fils gauche (vide par défaut).
- Son fils droit (vide par défaut).

De plus on rajoute une méthode spécifique, qui permet d'afficher la racine du noeud avec la fonction print() Ci-dessous la classe Noeud et la représentation de l'arbre.(La racine possède deux fils qui sont eux mêmes des noeuds possédant deux fils qui.....)

```
class Noeud:
    # Le constructeur
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
    # Méthode qui permet d'afficher la valeur
    # de la racine avec la fonction print
    def __str__(self):
        return str(self.value)
# création des noeuds
c = Noeud("C")
g = Noeud("G")
h = Noeud("H")
i = Noeud("I")
d = Noeud("D",g,None)
f = Noeud("F", h, i)
b = Noeud("B",c,d)
e = Noeud("E", f, None)
# création de l'arbre
arbre = Noeud("A", b, e)
print(arbre) # affiche r
```

Exercice 13

Ecrire un code qui construit l'arbre suivant, sachant qu'il contient 100 noeuds :



Exercice 14

1. Écrire Une méthode qui retourne vrai si le noeud est une feuille faux sinon.
2. Écrire une **fonction** qui retourne la hauteur de l'arbre. Attention je dis bien une **fonction**.
3. Maintenant écrire une méthode qui retourne la hauteur de l'arbre. C'est plus délicat qu'une fonction.