Spécialité NSI en terminale
Arbres binaires et algorithmes

1 Parcours d'un arbre binaire

On parcourt un arbre pour calculer sa taille ou sa hauteur, pour chercher une valeur particulière ou pour afficher les différentes valeurs, etc. Il existe différentes façons d'effectuer ce parcours. Nous distinguons un parcours en profondeur d'abord et un parcours en largeur d'abord.

1.1 Parcours en profondeur d'abord

En anglais, on parle de Depth-First Search (DFS).

Principe : on explore chaque branche complètement avant d'explorer la branche voisine. La programmation récursive est indiquée :

- si l'arbre est non vide, on parcourt de manière récursive son sous-arbre gauche puis son sous-arbre droit ;
- sinon, c'est terminé.

On distingue trois cas suivant le moment où est traitée une racine d'un sous-arbre. Traiter une racine signifie par exemple afficher la valeur.

- Si la racine est traitée avant ses deux sous-arbres, il s'agit d'un ordre préfixe.
- Si la racine est traitée entre ses deux sous-arbres, il s'agit d'un ordre infixe.
- Si la racine est traitée après ses deux sous-arbres, il s'agit d'un ordre postfixe.

Nous reprenons toujours le même arbre exemple et nous complétons l'interface afin de pouvoir traiter les deux implémentations, (liste et classe), en même temps. Nous souhaitons afficher les différentes valeurs des sommets.

```
def affiche_valeur(arbre):
    if not vide(arbre):
        if isinstance(arbre, list):
            print(arbre[0], end=" ")
        else:
            print(arbre.valeur, end=" ")
```

L'interface est donc composée des fonctions `vide`, `gauche`, `droit`, et `affiche_valeur`.

Nous définissons alors les fonctions de parcours.

```
def dfs_prefixe(arbre):
    if not vide(arbre):
        affiche_valeur(arbre)
        dfs_prefixe(gauche(arbre))
        dfs_prefixe(droit(arbre))

def dfs_infixe(arbre):
    if not vide(arbre):
        dfs_infixe(gauche(arbre))
        affiche_valeur(arbre)
        dfs_infixe(droit(arbre))
```

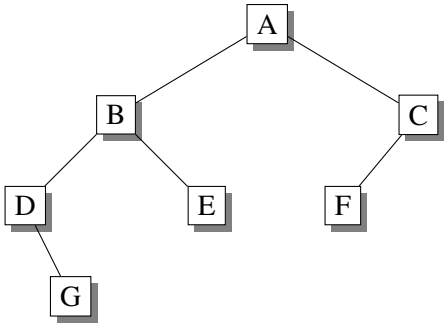
```
def dfs_postfixe(arbre):
    if not vide(arbre):
        dfs_postfixe(gauche(arbre))
        dfs_postfixe(droit(arbre))
        affiche_valeur(arbre)
```

Les programmes sont exécutés dans l'interpréteur Python avec l'arbre exemple et les deux implémentations. Les fonctions de parcours donnent les résultats suivants :

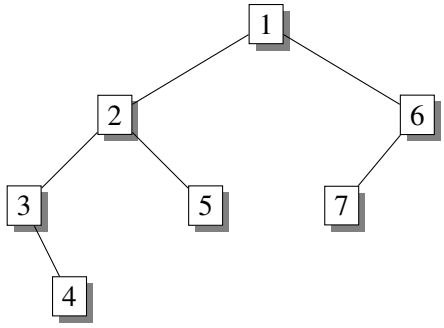
```
>>> dfs_prefixe(a)
A B D G E C F
>>> dfs_prefixe(b)
A B D G E C F
>>> dfs_infixe(a)
D G B E A F C
>>> dfs_infixe(b)
D G B E A F C
>>> dfs_postfixe(a)
G D E B F C A
>>> dfs_postfixe(b)
G D E B F C A
```

Nous obtenons des parcours, identiques pour les deux implémentations de l'arbre exemple, qui sont représentés ci-dessous dans les trois cas.

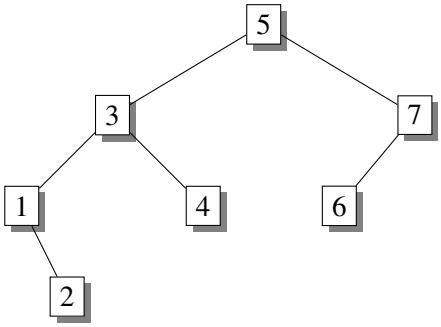
Arbre



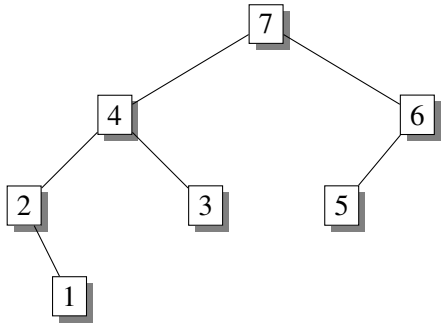
Ordre préfixe



Ordre infixe



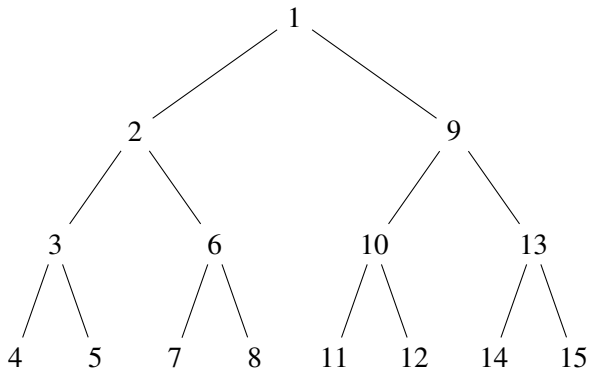
Ordre postfixe



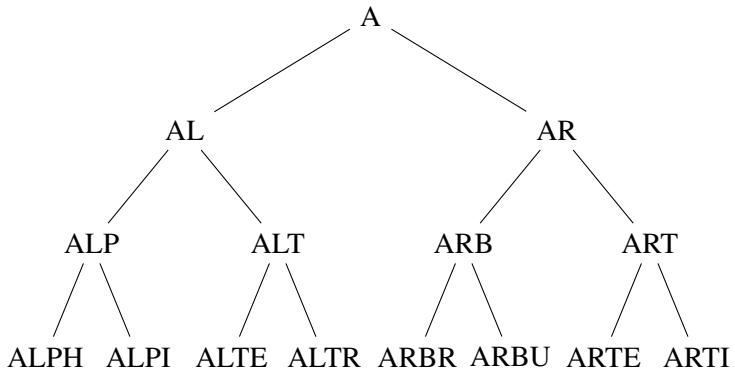
Comme cela a été fait précédemment, il est possible avec l'implémentation de l'arbre à l'aide d'une classe de définir dans cette classe trois méthodes `dfs_prefixe`, `dfs_infixe`, `dfs_postfixe`. Ceci

est proposé en exercice.

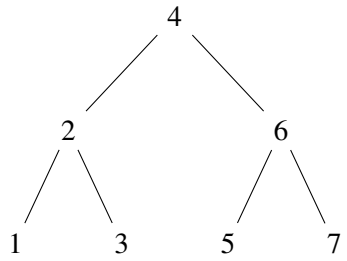
De manière générale, l'ordre d'un parcours préfixe est le suivant :

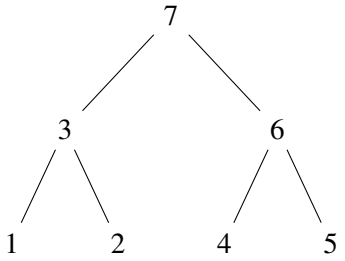


Un parcours préfixé est comparable à l'ordre alphabétique sur les préfixes de mots :



Un parcours infixe et un parcours postfixe :





1.2 Parcours en largeur d'abord

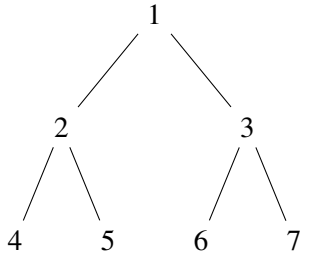
Le cas d'un parcours en largeur, (BFS pour Breadth-First Search en anglais), pose quelques difficultés. Il s'agit de parcourir un arbre niveau par niveau en considérant tous les sommets de chaque niveau.

On commence donc par la racine, puis les deux racines de chacun des deux sous-arbres, puis les quatre racines de chacun des quatre sous-arbres et ainsi de suite.

Parcours premier niveau

Parcours deuxième niveau

Parcours troisième niveau



La méthode la plus pratique consiste à utiliser une structure de file. Il peut être utile de consulter le chapitre 3 sur les structures de données. Pour mémoire, le principe d'une file est « premier entré, premier sorti ».

Description de l'algorithme

On suppose l'arbre non vide.

- On place l'arbre dans la file.
- Tant que la file n'est pas vide, on défile un élément, qui est un arbre, on affiche la valeur de la racine et on place dans la file chacun de ses deux sous-arbres s'ils ne sont pas vides.

Il s'agit d'un algorithme itératif.

Nous reprenons l'arbre exemple avec la classe `Arbre`. Plusieurs possibilités sont offertes pour représenter une file (voir chapitre 3). Nous utilisons ici le module `queue`.

Définition d'un arbre :

```
class Arbre:
    def __init__(self, val):
        self.valeur = val
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, val):
        self.gauche = Arbre(val)

    def ajout_droit(self, val):
        self.droit = Arbre(val)

    def taille(self):
        tg = self.gauche.taille() if self.gauche else 0
        td = self.droit.taille() if self.droit else 0
        return 1 + tg + td

b = Arbre('A')
b.ajout_gauche('B')
b.ajout_droit('C')
b.gauche.ajout_gauche('D')
b.gauche.ajout_droit('E')
b.gauche.gauche.ajout_droit('G')
b.droit.ajout_gauche('F')
```

Définition de la fonction `parcours_largeur` :

```
from queue import *

def parcours_largeur(arbre):
    f = Queue(arbre.taille()) # taille de la file
    f.put(arbre) # l'arbre est placé dans la file
    k = 0
    while not f.empty():
        a = f.get() # un élément est retiré de la file
```

```
        print(a.valeur, end=" ")
        if a.gauche is not None:
            f.put(a.gauche)
        if a.droit is not None:
            f.put(a.droit)
```

Test de la fonction :

```
>>> parcours_largeur(b)
A B C D E F G
```

Regardons quel est le contenu de la file. Si on représente un sous-arbre par sa racine, on obtient :

- on place 'A' dans la file ; 'A'
- on retire 'A' et on place 'B' et 'C' ; 'B' 'C'
- on retire 'B' et on place 'D' et 'E' ; 'C' 'D' 'E'
- on retire 'C' et on place 'F' ; 'D' 'E' 'F'
- on retire 'D' et on place 'G' ; 'E' 'F' 'G'
- on retire 'E' ; 'F' 'G'
- on retire 'F' ; 'G'
- on retire 'G' ;

Remarque : si on utilise une pile à la place d'une file, on obtient un parcours en profondeur préfixe. Cette méthode est proposée en exercice.