Parcours d’arbres binaires

Exercices d’application

Mesurer les arbres

On considère l’arbre non étiqueté fig 1 :

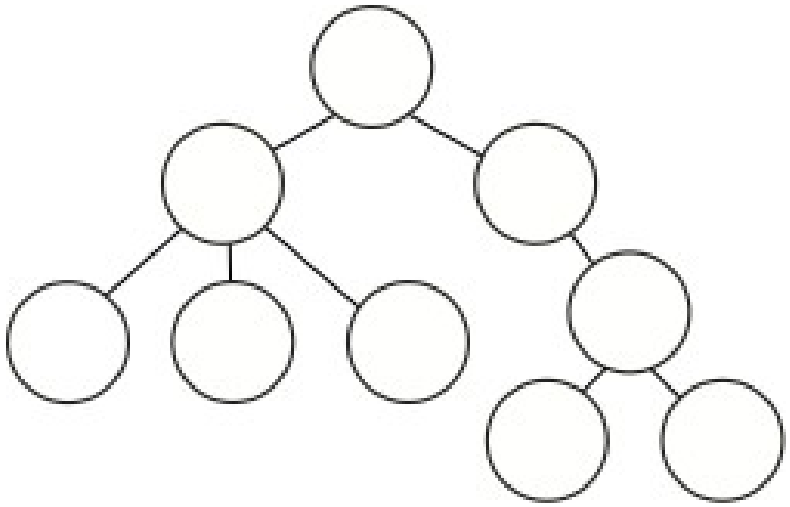


Figure 1: Arbre non étiqueté

1. Quelle est sa hauteur ?
   - ☐ 9
   - ☐ 2
   - ☐ 3
   - ☐ 8

2. Quelle est sa taille ?
   - ☐ 9
   - ☐ 8
   - ☐ 4

3. Quelle est l’arité maximale ?
   - ☐ 1
   - ☐ 2
   - ☐ 3
   - ☐ 4

Parcourir un arbre binaire

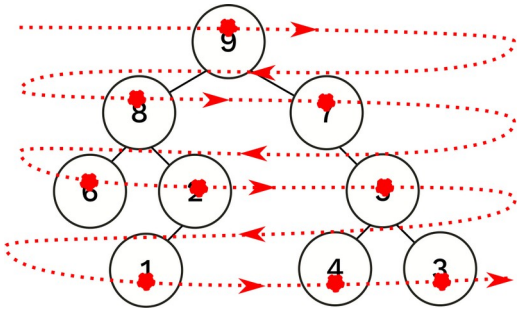


Figure 3: Parcours en largeur

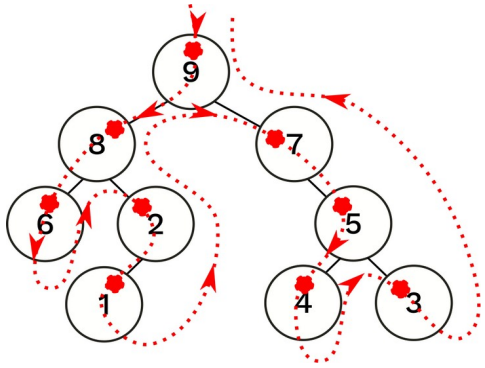


Figure 2: Parcours en profondeur préfixe

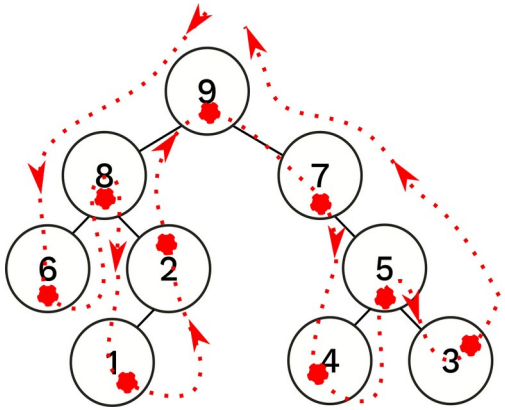


Figure 4: Parcours infixe

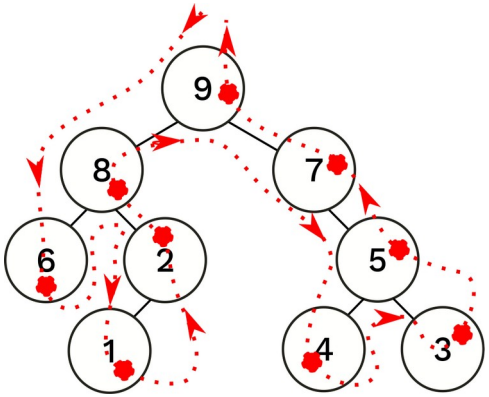


Figure 5: Arbre binaire étiqueté

4. Dans quel ordre seront examinés les nœuds lors de son parcours en largeur ?
   - ☐ 6-1-2-8-4-3-5-7-9
   - ☐ 9-8-7-6-2-5-1-4-3
   - ☐ 6-8-1-2-9-7-4-5-3
   - ☐ 9-8-6-2-1-7-5-4-3

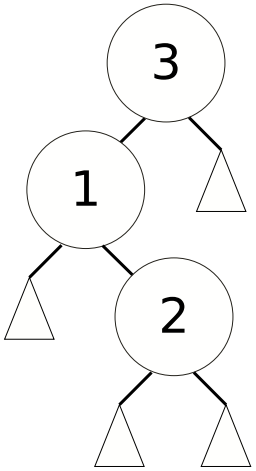
5. Dans quel ordre seront examinés les nœuds lors de son parcours préfixe ?
   - ☐ 6-1-2-8-4-3-5-7-9
   - ☐ 9-8-7-6-2-5-1-4-3
   - ☐ 6-8-1-2-9-7-4-5-3
   - ☐ 9-8-6-2-1-7-5-4-3

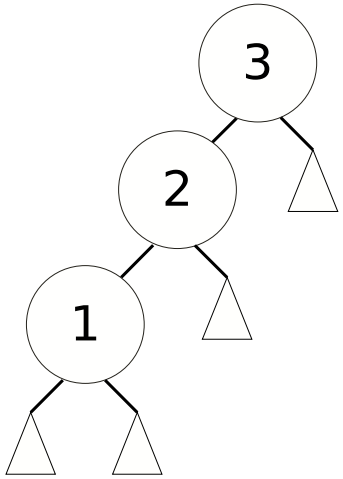
6. Dans quel ordre seront examinés les nœuds lors de son parcours infixe ?
   - ☐ 6-1-2-8-4-3-5-7-9
   - ☐ 9-8-7-6-2-5-1-4-3
   - ☐ 6-8-1-2-9-7-4-5-3
   - ☐ 9-8-6-2-1-7-5-4-3

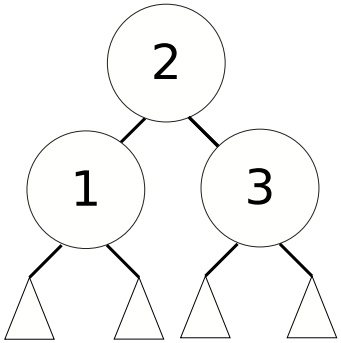
7. Dans quel ordre seront examinés les nœuds lors de son parcours postfixe?
   - ☐ 6-1-2-8-4-3-5-7-9
   - ☐ 9-8-7-6-2-5-1-4-3
   - ☐ 6-8-1-2-9-7-4-5-3
   - ☐ 9-8-6-2-1-7-5-4-3

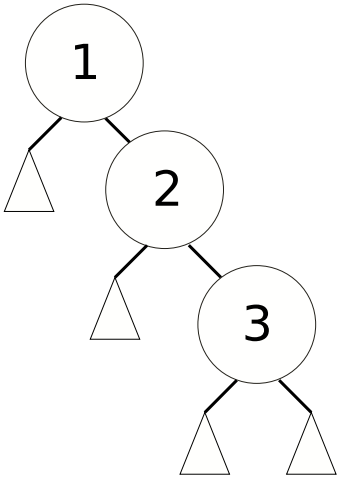
Construire des arbres ayant des ressemblances

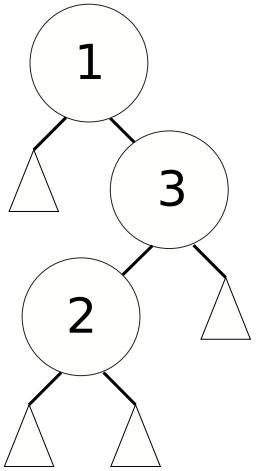
1. **Donner** 5 arbres de taille 3, différents, dont les nœuds internes contiennent les valeurs 1, 2, 3 et pour lesquels le parcours infixe donne 1-2-3.











On considère la classe Arbre suivante (prog 1) dont le fonctionnement a déjà été étudiée lors des activités précédentes. La méthode parcours_xx() affiche l'ensemble des nœuds de l'arbre dans un ordre qui illustre le parcours de l'arbre effectué.

```
class Arbre :
    def __init__(self,d, gauche=None, droit=None) :
        '''Construit un arbre constituie d'un noeud d'étiquette d, d'un
        sous arbre gauche et un sous arbre droit.'''
        self.__data = d
        self.__sag = gauche
        self.__sad = droit

    def est_feuille(self) :
        '''Retourne vrai si le noeud est une feuille'''
        return self.__sag==None and self.__sad==None

    def parcours_xx(self) :
        '''Affiche les etiquettes des noeuds en suivant un parcours XXX'''
        if self.est_feuille() :
            print(self.__data, end = ' ' )
            return
        else :
            self.__sag.parcours_xx()
            print(self.__data, end = ' ')
            self.__sad.parcours_xx()
```

Prog 1 : Classe Arbre (*fichier disponible sur \donnee\NSI\parcours_arbres\arbre.py*)

1. **Déterminer** le type de parcours effectué par cette méthode puis renommer la en conséquence. Indiquer la complexité de ce programme.

   Parcours infixe. L'évaluation de la racine est placée entre l'exploration du sous arbre gauche et l'exploration du sous arbre droit

2. **Tester** cette méthode avec l'arbre défini dans le cours.

Le cours 'Parcours d'arbres binaires' détaille quatre manières de parcourir un arbre.

3. **Définir** puis **tester** dans deux méthodes différentes, les deux parcours en profondeur restants.

```
def parcours_postfixe(self) :
    '''Affiche les etiquettes des noeuds en suivant un parcours postfixe'''
    if self.est_feuille() :
        print(self.__data, end = ' ' )
        return
    else :
        self.__sag.parcours_postfixe()
        self.__sad.parcours_postfixe()
        print(self.__data, end = ' ')

def parcours_prefixe(self) :
    '''Affiche les etiquettes des noeuds en suivant un parcours prefixe'''
    if self.est_feuille() :
        print(self.__data, end = ' ' )
        return
    else :
        print(self.__data, end = ' ')
        self.__sag.parcours_prefixe()
        self.__sad.parcours_prefixe()
```

Le parcours en largeur nécessite l'utilisation d'une file d'attente disponible dans le fichier file.py.
Au départ, on place l'arbre dans la file, puis, tant que la file contient des éléments, on défile un élément, on affiche son étiquette racine, on ajoute les deux sous-arbres fils dans la file et on recommence.

4. **Ecrire** et **tester** une méthode parcours_largeur() parcourant l'arbre dans sa largeur.

```
def parcours_largeur(self) :
    '''Affiche les etiquettes des noeuds en suivant un parcours en largeur'''
    f = file.File()
    f.enfiler(self)
    while not f.est_vide() :
        abr = f.defiler()
        if not abr.est_feuille() :
            f.enfiler(abr.__sag)
            f.enfiler(abr.__sad)
        print(abr.__data)
```

Evloution de la file

```
    Contenu file : ['H']
H
    Contenu file : ['Y', 'N']
Y
    Contenu file : ['N', 'P', 'T']
N
    Contenu file : ['P', 'T', 'O', '.']
P
    Contenu file : ['T', 'O', '.']
T
    Contenu file : ['O', '.']
O
    Contenu file : ['.']
.
```

Gras : Affichage de la méthode parcours en largeur