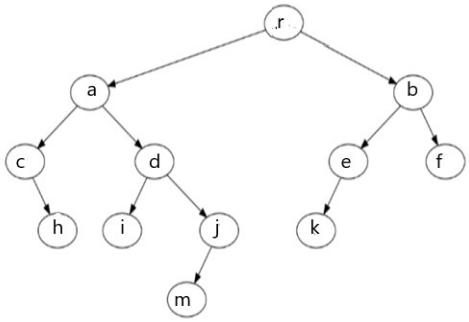
# Les parcours d'arbres

Nous allons voir **4 méthodes de parcours d'arbres** en fonction de l'ordre dans lequel on rencontre les nœuds.




Ces parcours s'appliquent aux arbres quelconques mais on ne considèrera que des arbres binaires dans la suite.

## A) 3 parcours en profondeur : Préfixe, Infixe et Suffixe

➔ **Préfixe** : Dans un **parcours préfixe**, chaque noeud est visité avant que ses enfants soient visités.
Cela signifie que l'on affiche la racine de l'arbre, on parcourt tout le sous arbre de gauche, une fois qu'il n'y a plus de sous arbre gauche on parcourt les éléments du sous arbre droit. Ce type de parcours peut être résumé en trois lettres : R G D (pour Racine Gauche Droit).

Dans l'exemple cela donne [ 'r' , 'a', 'c',…

➔ **Infixe** : Dans un **parcours infixe,** chaque noeud est visité après son enfant gauche mais avant son enfant droit.
Le parcours infixe affiche la racine après avoir traité le sous arbre gauche, après traitement de la racine, on traite le sous arbre droit (c'est donc un parcours de type G R D : Gauche Racine Droite).

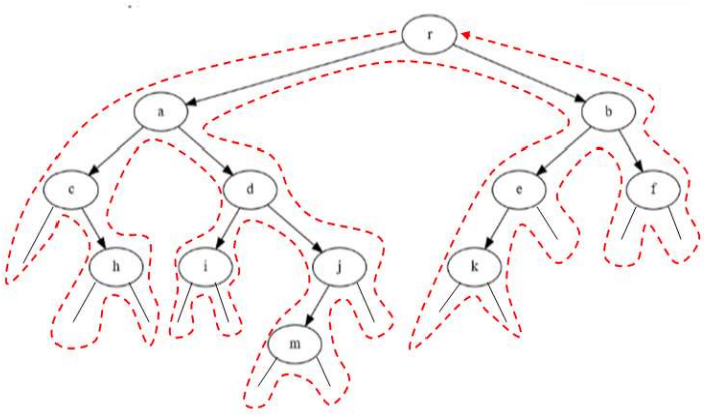
Dans l'exemple cela donne [ 'c' , 'h', 'a',….

➔ **Suffixe (ou appelé aussi Postfixe)** : Dans un parcours suffixe, chaque noeud est visité après que ses enfants sont visites. Le parcours suffixe effectue schématiquement le parcours suivant : sous arbre gauche, sous arbre droit puis la racine, c'est donc un parcours G D R (Gauche Droite Racine).

Dans l'exemple, cela donne [ 'h', 'c', 'i' ,….

Une autre façon d'obtenir ces 3 parcours :
On se balade autour de l'arbre en suivant les pointillés. Dans le schéma ci-contre, on a **rajouté** des "nœuds fantômes" pour montrer que l'on peut considérer que chaque nœud est visité 3 fois

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




On peut définir les parcours comme suit :

- **préfixe**, on affiche le nœud la **première fois** qu'on le rencontre.

- **infixe**, on affiche 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.

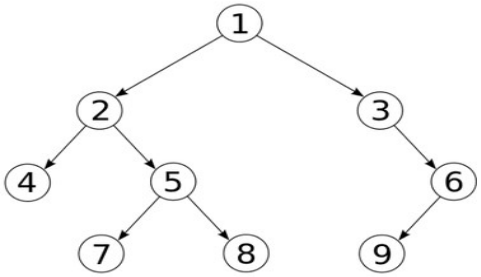
- **suffixe**, on affiche le noeud la **dernière fois** qu'on le rencontre.

*Exercice 1 : Donner les 3 parcours pour l'arbre ci- contre*

Préfixe :

Infixe :

Suffixe :

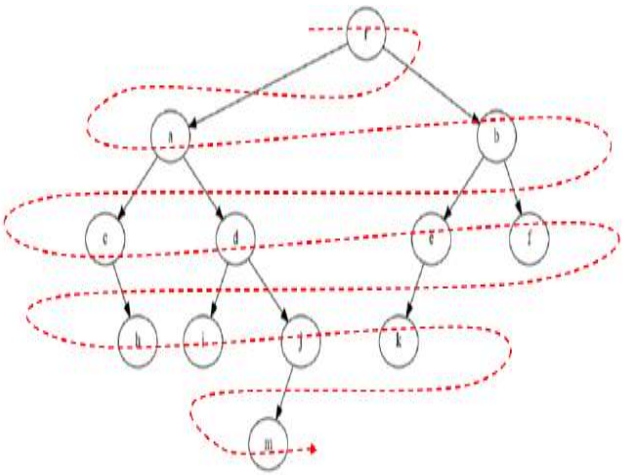


Exercice 2 : Voici un algorithme, dire quel parcours il réalise et écrire les 2 autres parcours.

| | | |
|---|---|---|
| Fonction parcours ……..(arbre)<br>Données : arbre binaire<br>Si L'arbre n'est pas vide alors<br>parcours(sous-arbre gauche)<br>parcours(sous-arbre droit)<br>Afficher : La racine de l'arbre<br>Fin si | Fonction parcours ……..(arbre)<br>Données : arbre binaire<br>Si L'arbre n'est pas vide alors<br><br><br><br>fin si | Fonction parcours …… …(arbre)<br>Données : arbre binaire<br>Si L'arbre n'est pas vide alors<br><br><br><br>Fin si. . . |

| | |
|---|---|
| Exercice 3 : Reprendre les 2 notebooks d'implémentation (tuples et dictionnaires) et coder ces 3 fonctions et confirmer les réponses sur l'arbre1.<br>Pour une des 3 fonctions, au lieu d'un affichage, on retournera le parcours dans une liste L.<br>Faire de même pour l'implémentation en objet | Aide pour l'implémentation en objet :<br>`def prefixe(self):`<br>`  if self != None:`<br>`  print(self.valeur, end=" ; ")`<br>`  if self.fils_gauche !=None:`<br>`  self.fils_gauche.prefixe()`<br>`  if self.fils_droit !=None:`<br>`  self.fils_droit.prefixe()` |

- **Le parcours en largeur :**



Le principe de ce parcours est de visiter le nœud le plus proche de la racine qui n'a pas déjà été visité, ainsi on va d'abord visiter la racine, puis les nœuds à la profondeur 1, puis 2, etc…

D'où le nom **parcours en largeur**.

Comme le montre le schéma ci-contre:

On obtient la liste suivante :

['r', 'a', 'b', 'c', 'd', 'e', 'f', 'h', 'i', 'j', 'k', 'm']

L'idée est la suivante : On utilise une File.
On met la racine de 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.

```
f = File  # on crée une file vide
f.enfiler(noeud)  # on place la racine dans la file
tant que f non vide :
    n = f.defiler()
    visiter(n)   # on l'affiche
    si n.gauche n'est pas vide
            f.enfiler(n.gauche)
    fin si
    si n.droit n'est pas vide
            f.enfiler(n.droit)
    fin si
fin tant que
```

*Exercice 4 :*
*Implémenter l'algorithme ci-contre et tester- le*

*On peut aussi retourner le parcours dans une liste*