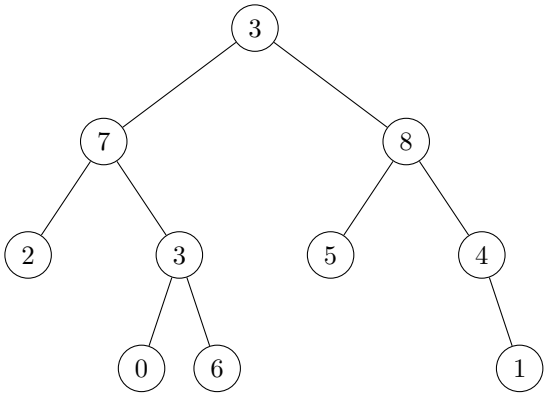
1- Parcours en largeur.

1. Exécuter l'algorithme suivant avec l'arbre A donné ci-dessous :

```
Algorithme ? ParcoursLargeur(A)
'''
Entrée : un arbre binaire A
Sortie : ???
'''
    F   file vide
    x est la racine de l?arbre A
    Si x  n'est pas vide, l?ajouter à F
    Tant que F est non vide :
        y  est le sommet de F
        défiler un élément de F
        Afficher y
        Si le fils gauche de y  n'est pas vide
            l'ajouter à F
        Si le fils droit de y  n'est pas vide
            l'ajouter à F
```

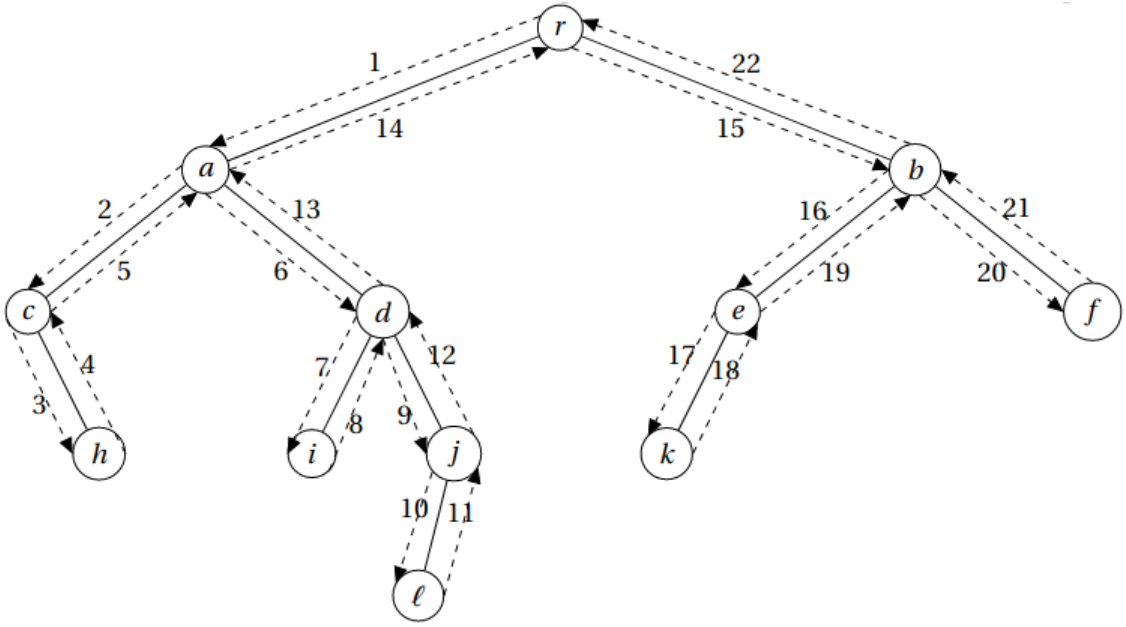


Réponse :

2. Expliquer l'ordre de visite d'un parcours en largeur :

2- Parcours en profondeur.

On se balade autour de l ?arbre en suivant les pointillés dans l ?ordre des numéros indiqués



A partir de ce contour, on définit trois parcours des sommets de l ?arbre :

1. **l'ordre préfixe** : on liste chaque sommet la première fois qu ?on le rencontre dans la balade.

Ce qui donne :

2. **l ?ordre postfixe** : on liste chaque sommet la dernière fois qu ?on le rencontre.

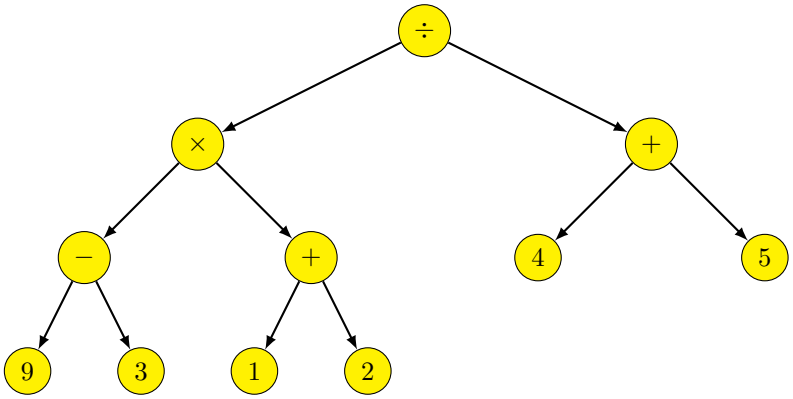
Ce qui donne :

3. **l'ordre infixe** : on liste chaque sommet ayant un fils gauche la seconde fois qu ?on le voit et chaque sommet sans fils gauche la première fois qu ?on le voit.

Ce qui donne :

Exercice n° 1

Écrire les sommets de l'arbre ci-dessous pour chacun des ordres postfixe, préfixe, infixe



Pour le parcours infixe, on ajoute la convention suivante : on ajoute une parenthèse ouvrante à chaque fois qu ?on entre dans un sous-arbre et on ajoute une parenthèse fermante lorsqu'on quitte ce sous-arbre.

Préfixe (Notation polonaise) :

Postfixe (Notation polonaise inversée) :

Infixe :

Exercice n° 2 Implémentation en Python

1. Compléter les algorithme `ParcoursPostfixe(x)` et `ParcoursInfixe(x)`

   a. Parcours préfixe : Dans un parcours préfixe (preorder traversal), chaque nœud est visité avant que ses enfants soient visités.

   ```
   Algorithme -  ParcoursPrefixe(x)
       # Entrée : un arbre binaire A
       Si x n'est pas vide
         Afficher val(x)
           ParcoursPrefixe(fils gauche de x)
           ParcoursPrefixe(fils droit de x)
   ```

   b. Parcours suffixe (ou postfixe) : Dans un parcours postfixe (postorder traversal), chaque nœud est visité après que ses enfants sont visités.

   ```
   Algorithme - ParcoursPostfixe(A)
       # Entrée : un arbre binaire A
   ```

   c. Parcours infixe : Dans un parcours infixe (inorder traversal), chaque nœud est visité après son fils gauche mais avant son fils droit.

   ```
   Algorithme - ParcoursInfixe(x)
       # Entrée : un arbre binaire A
   ```

2. Créer un fichier `parcours.py` et implémenter ces algorithme en Python puis tester sur l'arbre suivant :

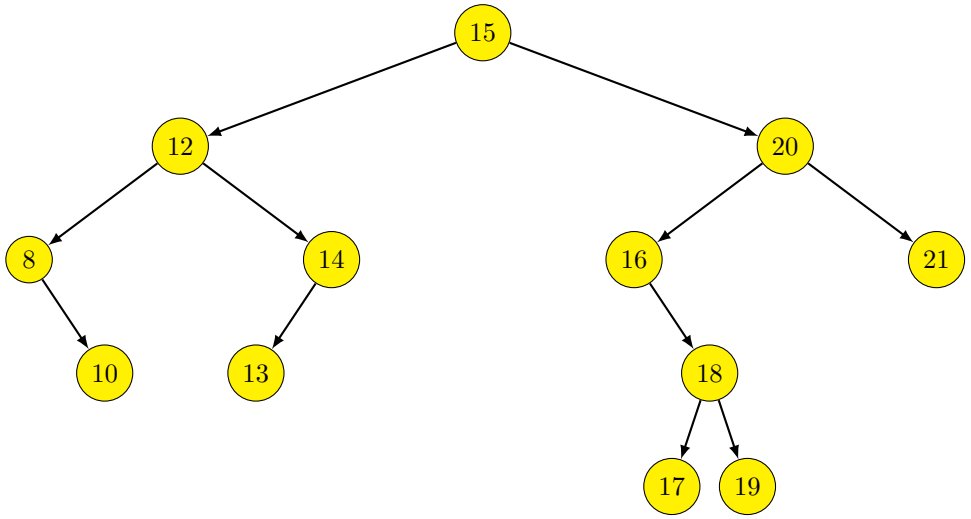
   ```
   ["*", ["+",[3,[],[]],["/",[4,[],[]],[2,[],[]]]],["-",["*",[2,[],[]],["3",[],[]]],["8",[],[]]]]
   ```

3. Dessiner l'arbre.

Exercice n° 3

Expressions arithmétiques : Représenter l'opération mathématique $(\frac{y}{2} - t) \times (75 + z)$ sous forme d'arbre binaire. (les valeurs de départ sont les feuilles, la dernière opération effectuée est la racine)

Exercice n° 4



Donner l'ordre d'affichage des nœuds dans les cas :
- d'un parcours en profondeur prefixe
- d'un parcours en profondeur infixe
- d'un parcours en profondeur suffixe (ou postfixe)
- d'un parcours en largeur

Exercice n° 5

1. Quelle est la hauteur minimale d'un arbre à 83 nœuds ? la hauteur maximale ?
2. Quel est le nombre maximal de feuilles d'un arbre de hauteur 8 ?
3. Quel est le nombre maximal de nœuds d'un arbre de hauteur 8 ?

Travaux Pratiques : Les marmottes au sommeil léger

Un groupe de marmottes, moyennement satisfaites de leur terrier actuel, décide de concevoir un nouveau terrier et de le creuser avant l ?hiver. Pour ce faire les marmottes doivent respecter trois règles.

1. A partir de l ?entrée on peut construire deux couloirs, et au bout de chaque couloir on peut faire un embranchement vers deux autres couloirs, mais pas plus (au risque de faire s'écrouler l'édifice).
2. Les marmottes vont chacune occuper une salle différente (pour ne pas se réveiller les unes les autres) et forcément une salle qui est tout au bout d'un couloir. Pour des marmottes au sommeil léger il est inenvisageable de dormir dans une salle à un embranchement, car les marmottes qui seraient au-delà de cet embranchement leur marcheraient dessus en entrant/sortant, et cela ruinerait leur hibernation.
3. Chaque marmotte se réveille un nombre précis de fois dans l'hiver. Comme les pas de marmottes émettent de légères vibrations et que nos marmottes ont vraiment le sommeil léger, on va vouloir minimiser les déplacement de l'ensemble du groupe. Pour cela on va compter les déplacements de la façon suivante. Une marmotte dormant à 4 couloirs de l'entrée se réveillant 5 fois dans l'hiver va parcourir 4 x 5 = 20 couloirs aller et retour (pour simplifier on ne va compter que les allers). On va faire la somme des déplacements de toutes les marmottes et **essayer de rendre cette somme la plus petite possible.**

Et maintenant, creusez !

1. Quelle stratégie a permis que le nombre total de déplacement soit le plus petit possible ?
2. Écrire un algorithme en Python qui prend une liste de "marmottes " et qui retourne un arbre binaire correspondant au plan de construction du terrier.
   Exemples : `FamilleMarmottes=[2,3,5,7,8]`