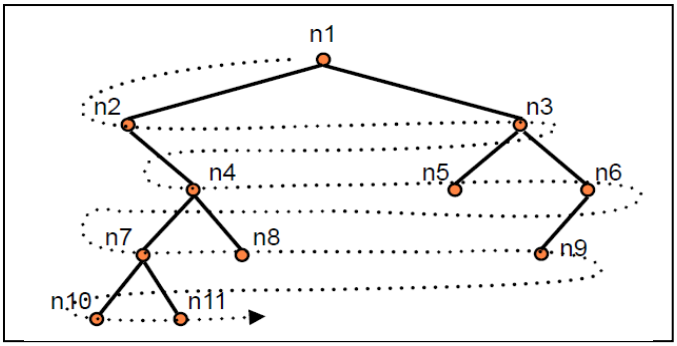
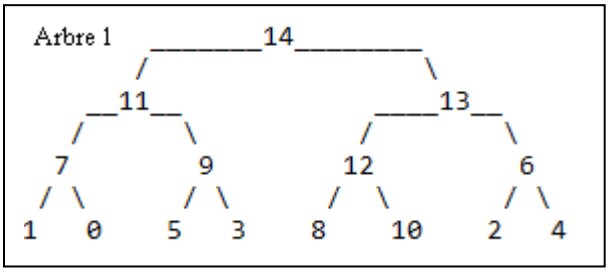
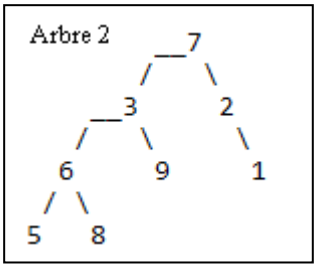
Parcours en largeur d'un arbre

I- Principe :



Lors du parcours d'un arbre en largeur d'abord, il faut visiter tous les nœuds de la ligne de profondeur n avant de passer à la ligne n+1
Sur notre exemple, cela donne :
n1,n2,n3, n4,n5,n6, n7,n8,n9, n10,n11





Donner la liste des nœuds pour un parcours en largeur des deux arbres ci-contre

Réponse :
Arbre 1 :

Arbre 2 :

II- Etude de l'algorithme :

On va utiliser une FILE type FIFO (first In, First Out) pour stocker les noeuds à visiter.
La notion de FILE est largement utilisée dans les process (voir lien ci-dessous):
https://www.youtube.com/watch?v=ek7TVIAV7JE (1mn40 , au-delà pas la peine pour le moment)

Voyons l'application pour le parcours de notre arbre

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

On suppose que Enfiler et Defiler gèrent correctement la file FIFO et permettent donc d'ajouter (enfiler) ou retirer (défiler) les éléments de la file.
Traitement(x) permet juste ici d'afficher x.

Compléter les feuilles suivantes pour chaque étape du déroulement de l'algorithme pour un appel de la fonction avec arbre2 :

→ parcours_Largeur(arbre2)

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

```
F=[]
F=[7]

F pas vide, j'entre dans boucle :
      x=7   (et F=[])
      → 7 (affichage)
      filsgauche(7) = 3 donc pas vide :
          F=[3]

      Filsdroite(7) = 2 donc pas vide :
          F=[3,2]
Boucler
```

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

```
F pas vide, je continue dans boucle :
      x=    (et F=[ ])
      →
      filsgauche(3) = 6 donc pas vide :
          F=[  ,  ]

      Filsdroite(3) = 9 donc pas vide :
          F=[  ,   ,   ]
Boucler
```

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

```
F pas vide, je continue dans boucle :
      x=2   (et F=[  ,  ])
      → 2
      filsgauche( ) = None donc vide
          --pas exécuté--

      Filsdroite( ) =  donc pas vide :
          F=[  ,  ,   ]
Boucler
```

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

```
F pas vide, je continue dans boucle :
      x=    (et F=[  ,   ])
      →
      filsgauche(   ) = 5 donc pas vide
          F=[9,1,5]

      Filsdroite(   ) = 8 donc pas vide :
          F=[  ,   ,   ,   ]
Boucler
```

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

F pas vide, je continue dans boucle :
x= (et F=[, ,])
→
filsgauche( ) = **None** donc vide
--pas exécuté--

Filsdroite( ) = **None** donc vide :
--pas exécuté--
Boucler

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

F pas vide, je continue dans boucle :
x= (et F=[,])
→
filsgauche( ) = **None** donc vide
--pas exécuté--

Filsdroite( ) = **None** donc vide :
--pas exécuté--
Boucler

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

F pas vide, je continue dans boucle :
x= (et F=[ ])
→
filsgauche( ) = **None** donc vide
--pas exécuté--

Filsdroite( ) = **None** donc vide :
--pas exécuté--
Boucler

```
fonction Parcours_Largeur(a)
  Initialement la file F est vide
  Enfiler le noeud a dans F
  Tantque F n'est pas vide faire
    x ← Défiler(F)
    Traitement(x)
    Si filsgauche(x) n'est pas l'arbre vide
      Enfiler filsgauche(x) dans F
    Si filsdroit(x) n'est pas l'arbre vide
      Enfiler filsdroit(x) dans F
  Fin Tantque
```

F pas vide, je continue dans boucle :
x= (et F=[ ])
→
filsgauche( ) = **None** donc vide
--pas exécuté--

Filsdroite( ) = **None** donc vide :
--pas exécuté--
Boucler

III- Implémentation en python

```
1   from binarytree import tree,bst,heap,build,Node
2
3   def Parcours_Largeur(arbre):
4       # File F vide initialement
5       F = []
6       F = F+[arbre]        # enfile 1er noeud dans la liste F
7
8       while F != []:
9
10          x = F[0]    # on récupére 1er élément de la liste = premier entré
11          F = F[1:]   # on l'enlève de la liste
12          print ("Noeud",x.value ,end=" ")
13
14          if x.left != None:
15              F = F + [x.left]
16
17          if x.right != None:
18              F = F + [x.right]
19
20
21
22  valeurs = [7, 3, 2, 6, 9, None, 1, 5, 8]
23  racine2 = build(valeurs)                # construction de l'arbre
24  print (racine2)
25  Parcours_Largeur(racine2)
```

On utilise une liste pour simuler une file, ce qui est très facile vu toutes les méthodes disponibles pour gérer les listes en python …

1- Tester le programme et vérifier qu'il affiche bien les nœuds d'un parcours en largeur de l'arbre 2.
2- Construire l'arbre 1 (il suffit de remplir la liste des nœuds ligne par ligne et on indique None si un fils est manquant et vérifier qu'il affiche correctement les nœuds pour parcours en largeur.

3- A partir de votre propre classe Arbre vue dans les cours/TD précédents sur les arbres, ajouter la fonction parcours_largeur ci-dessus (quelques adaptations à faire)