Les structures de données arborescentes

Reconnaître et représenter un arbre

Exercice 1

La figure 1 représente différentes structures relationnelles.

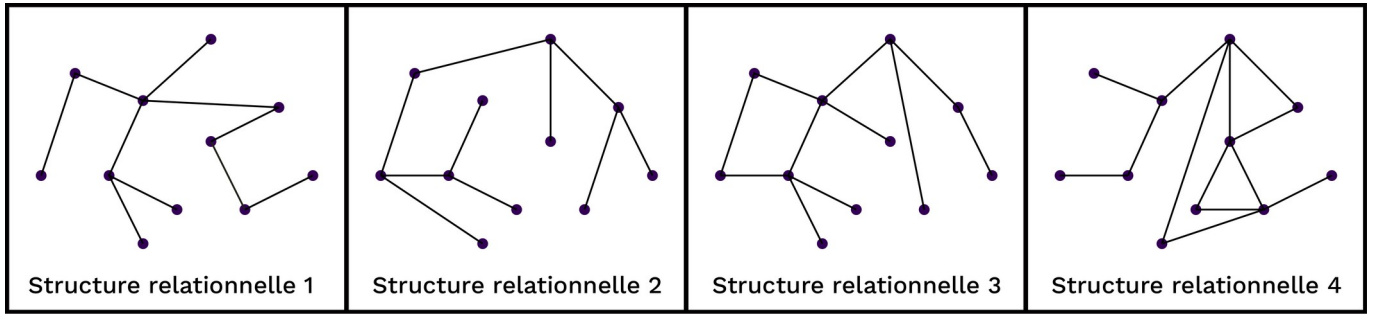


Figure 1: Exemples de structures relationnelles

1. **Identifier** les représentations qui correspondent à des arbres. **Justifier** vos réponses

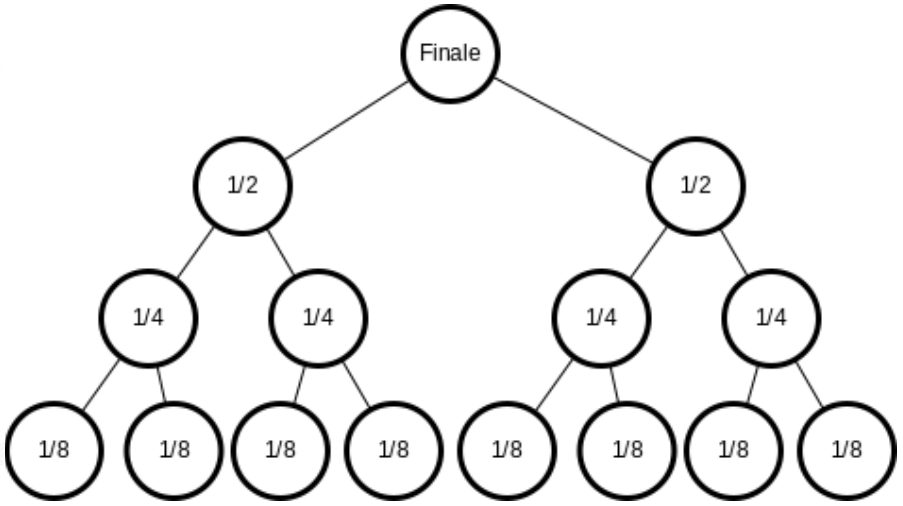
Structures relationnelles 1 et 2 : arbres
Structures relationnelles 3 et 4 : graphes → Présence de cycles

Exercice 2

Beaucoup de tournoi sportif sont organisés avec une structure qui représente les matchs joués ou à jouer dans le tournoi. Le principe de base est que chaque branche rassemble deux joueurs (ou deux équipes). Le gagnant avance et le perdant est éliminé. C'est en tout cas le fonctionnement avec élimination directe.

1. **Représenter** sous la forme d'un graphe, la la planification représentée figure 2.

Figure 2: Planification des matchs d'un tournoi



2. **Déterminer** la hauteur, la taille et l'arité maximale de cet arbre.

Hauteur : 3
Taille : 15
Arité max : 2 → Arbre binaire

Exercice 3

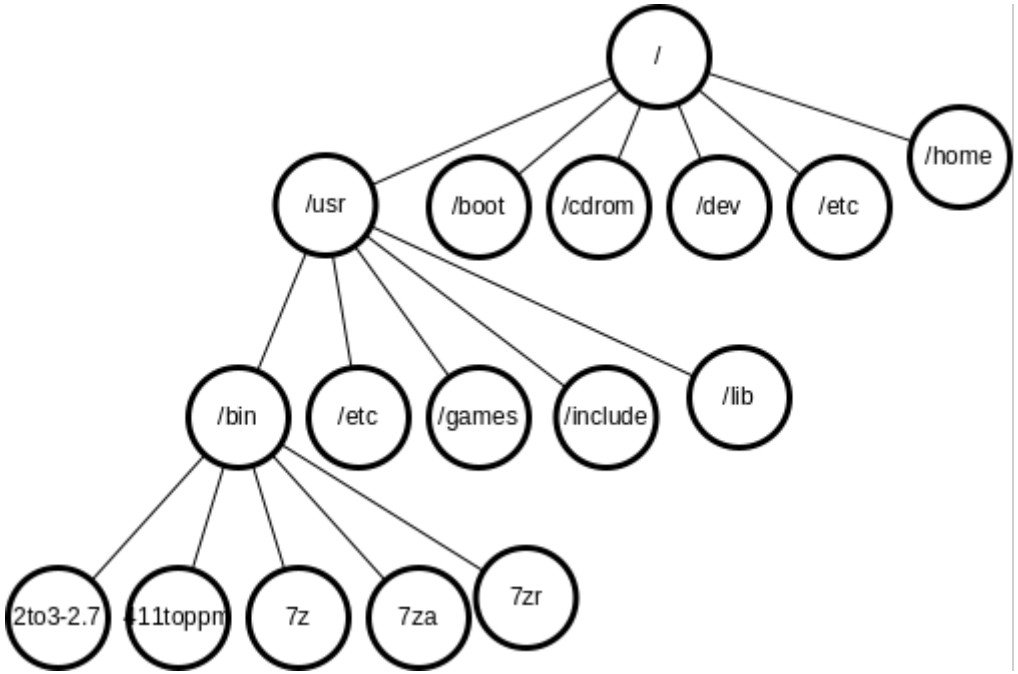
Linux est le système d'exploitation privilégié des serveurs. Il offre pour avantage, en plus d'être opensource, d'offrir une gestion avancée des tâches par ligne de commande.
La commande **`ls -l`** permet de lister les fichiers et répertoires contenus dans un répertoire passé en argument. Par exemple :
`ls - l /usr` renvoie la liste des fichiers et répertoires du dossier /usr.

```
$ ls -l /
total 176
drwxr-xr-x   2 root root 12288 avril 15 19:14 usr
drwxr-xr-x   3 root root  4096 avril 17 09:35 boot
drwxrwxr-x   2 root root  4096 oct.  21  2017 cdrom
drwxr-xr-x  20 root root  4940 avril 17 08:53 dev
drwxr-xr-x 202 root root 12288 avril 16 19:19 etc
drwxr-xr-x   8 root root  4096 juin  14  2019 home
etc...
```

```
$ ls -l /usr
total 248
drwxr-xr-x   2 root root 131072 avril 16 19:18 bin
drwxr-xr-x   3 root root   4096 janv. 20  2018 etc
drwxr-xr-x   2 root root   4096 avril  5 20:22 games
drwxr-xr-x 116 root root  20480 avril  8 19:04 include
drwxr-xr-x 197 root root  20480 avril 16 19:17 lib
etc...
```

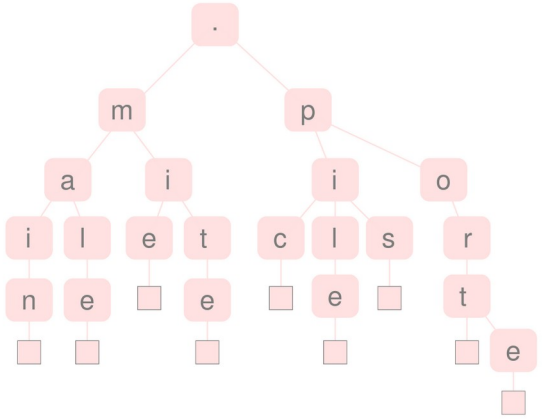
```
$ ls -l /usr/bin
total 1424264
-rwxr-xr-x 1 root root          96 avril  7 14:05 2to3-2.7
-rwxr-xr-x 1 root root       10104 avril 23  2016 411toppm
-rwxr-xr-x 1 root root          39 août   9  2019 7z
-rwxr-xr-x 1 root root          40 août   9  2019 7za
-rwxr-xr-x 1 root root          40 août   9  2019 7zr
etc...
```

1. **Représenter** sous la forme d'un graphe l'arborescence partielle décrite par les trois commandes **`ls -l /` ; `ls -l /usr` ; `ls -l /usr/bin`**



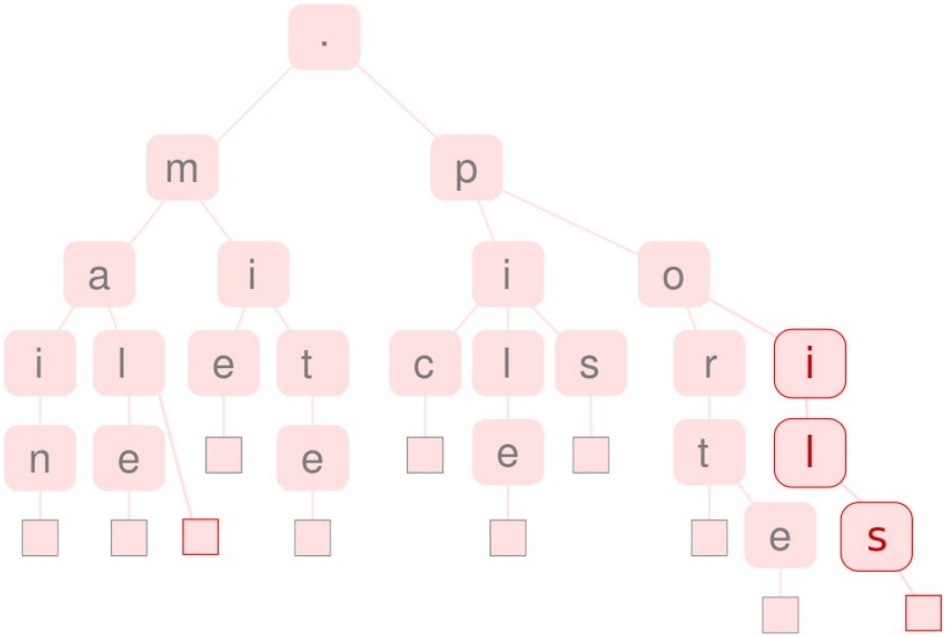
Exercice 4 :

Un arbre lexicographique, ou arbre en parties communes, ou dictionnaire, représente un ensemble de mots. Les préfixes communs à plusieurs mots apparaissent une seule fois dans l'arbre, ce qui se traduit par un gain d'espace mémoire. De plus la recherche d'un mot est assez efficace, puisqu'il suffit de parcourir une branche de l'arbre en partant de la racine, en cherchant `a chaque niveau parmi les fils du nœud courant la lettre du mot de rang correspondant.



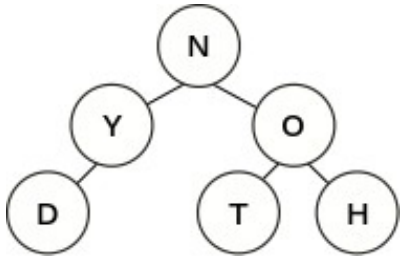
main
male
mie
mite
pile
pis
port
porte

1. **Rajouter** à cet arbre les mots **poils** et **mal**



Représentation d'un arbre binaire avec une liste

Un arbre binaire équilibré peut être implémenté par une liste. La liste comportera le nom des nœuds/feuilles classés par ordre de profondeur et de gauche à droite.
L'absence de fil sur un nœud est précisé avec la valeur `None` à sa place. La figure suivante illustre ce codage :



Arbre **arb**

Exemple d'arbre binaire

```
abr = ['N', 'Y', 'O', 'D', None, 'T', 'H']
```

Liste représentant ce graphe

2. **Exprimer** la longueur de la liste en fonction de la profondeur de l'arbre.

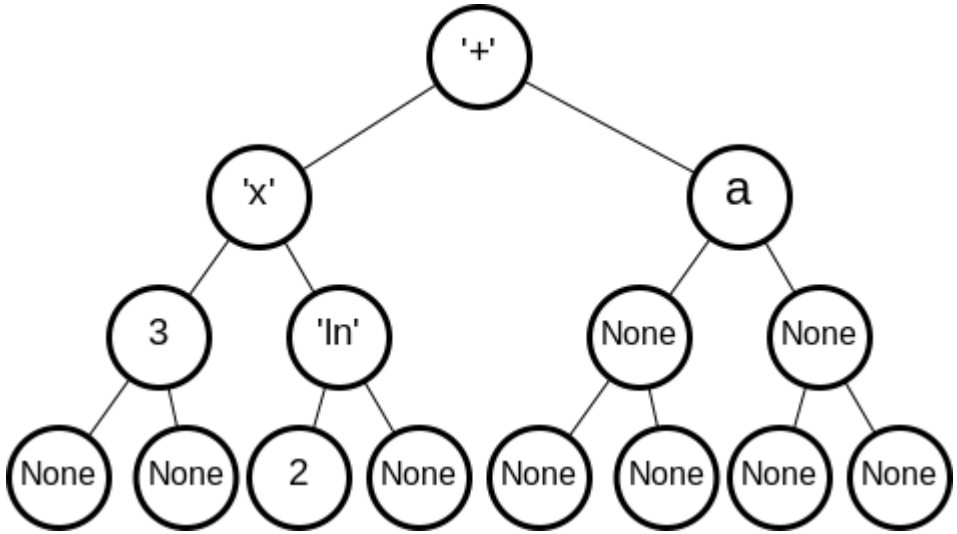
$$\textbf{longueur} = 2^{\textbf{profondeur}+1}-1$$

On donne la liste suivante :

```
a = ['+', 'x', a, 3, 'ln', None, None, None, None, 2, None, None, None, None, None]
```

3. **Dessiner** l'arbre correspondant à cette liste. **Décrire** ce que représente cet arbre.

Cet arbre représente la formule mathématique *a + 3.ln(2)*



La programmation suivante implémente sous la forme d'une liste l'arbre binaire équilibré représenté figure 3

```
def creation_arbre(r,profondeur):
    ''' r : la racine (str ou int). la profondeur de l'arbre (int)'''
    arbre = [r]+[None for i in range(2**(profondeur+1)-2)]
    return arbre
def insertion_noeud(arbre,n,fg,fd):
    '''Insére les noeuds et leurs enfants dans
l'arbre'''
    indice = arbre.index(n)
    arbre[2*indice+1] = fg
    arbre[2*indice+2] = fd

# création de l'arbre
arbre = creation_arbre("r",3)
# ajout des noeuds par niveau de gauche à droite
insertion_noeud(arbre,"r","a","b")
insertion_noeud(arbre,"a","c","d")
insertion_noeud(arbre,"b","e","f")
insertion_noeud(arbre,"c",None,"h")
insertion_noeud(arbre,"d","i","j")
insertion_noeud(arbre,"e","k",None)
insertion_noeud(arbre,"f",None,None)
```

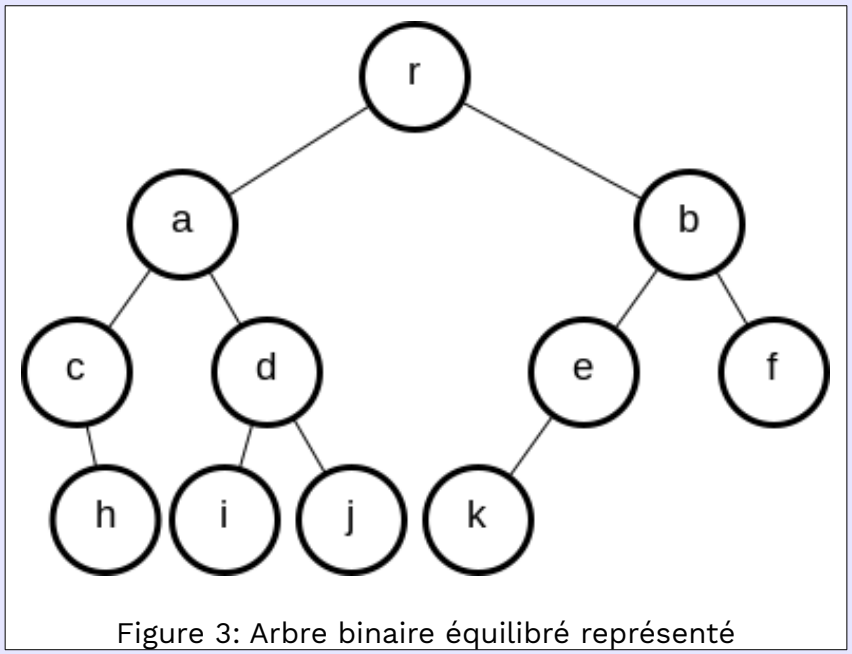


Figure 3: Arbre binaire équilibré représenté

La fonction suivante retourne le parent d'un nœud s'il existe :

```
def parent(arbre,p):
    if p in arbre:
        indice = arbre.index(p)
        if indice%2 == 0: return arbre[(indice-2)//2]
        else: return arbre[(indice-1)//2]
```

Programme disponible sur **\donnee\NSI\arbres\arbre_liste.py**

4. **Programmer** et **tester** les fonctions suivantes :
   - ➔ Une fonction qui retourne vrai si l'arbre est vide.
   - ➔ Une fonction qui retourne les enfants d'un nœud.
   - ➔ Deux fonctions qui retournent le fils gauche d'un nœud et son homologue le fils droit s'ils existent.
   - ➔ Une fonction qui retourne vrai si le nœud est la racine de l'arbre.
   - ➔ Une fonction qui retourne vrai si le nœud est une feuille.
   - ➔ Une fonction qui retourne vrai si le nœud a un frère gauche ou droit
   - ➔

```
def creation_arbre(r,profondeur):
    ''' r : la racine (str ou int). la profondeur de l'arbre (int)'''
    arbre = [r]+[None for i in range(2**(profondeur+2)-2)]
    return arbre

def insertion_noeud(arbre,n,fg,fd):
    '''Insére les noeuds et leurs enfants dans l'arbre'''
    indice = arbre.index(n)
    arbre[2*indice+1] = fg
    arbre[2*indice+2] = fd

def parent(arbre,p):
    '''Retourne le parent du noeud p de l'arbre'''
    if est_racine(arbre,p) : return None
    if p in arbre:
        indice = arbre.index(p)
    if indice%2 == 0:
        return arbre[(indice-2)//2]
    else:
        return arbre[(indice-1)//2]
```

```
def est_vide(arbre) :
   '''Test si l'arbre est vide'''
   return arbre[0]==None

def enfants(arbre,p):
   '''Retourne les enfants du noeud p de l'abre'''
   if p in arbre :
        indice = arbre.index(p)
        if est_feuille(arbre,p) : return (None, None)
        else : return (arbre[2*indice + 1], arbre[2*indice + 2])
   else : return (None, None)

def fils_gauche(arbre,p) :
   '''Retourne le fils gauche du noeud p de l'arbre'''
   return enfants(arbre,p)[0]

def fils_droit(arbre,p) :
   '''Retourne le fils droit du noeud p de l'arbre'''
   return enfants(arbre,p)[1]

def est_racine(arbre,p) :
   '''Test si le noeud p de l'arbre est sa racine'''
   return arbre.index(p) == 0

def est_feuille(arbre,p) :
   '''Test si le noeud p de l'arbre est une feuille'''
   indice = arbre.index(p)
   if 2*indice+1 > len(arbre) or (arbre[2*indice + 1], arbre[2*indice + 2]) == (None, None):
        return True
   else :
        return False

def frere(arbre,p) :
   '''Retourne le frere du noeud p s'il existe'''
   if est_racine(arbre,p) : return None
   pere = parent(arbre,p)
   e = enfants(arbre,pere)
```

```
    return e[1-e.index(p)]

# création de l'arbre
arbre = creation_arbre("r",3)
# ajout des noeuds par niveau de gauche à droite
insertion_noeud(arbre,"r","a","b")
insertion_noeud(arbre,"a","c","d")
insertion_noeud(arbre,"b","e","f")
insertion_noeud(arbre,"c",None,"h")
insertion_noeud(arbre,"d","i","j")
insertion_noeud(arbre,"e","k",None)
insertion_noeud(arbre,"f",None,None)
```