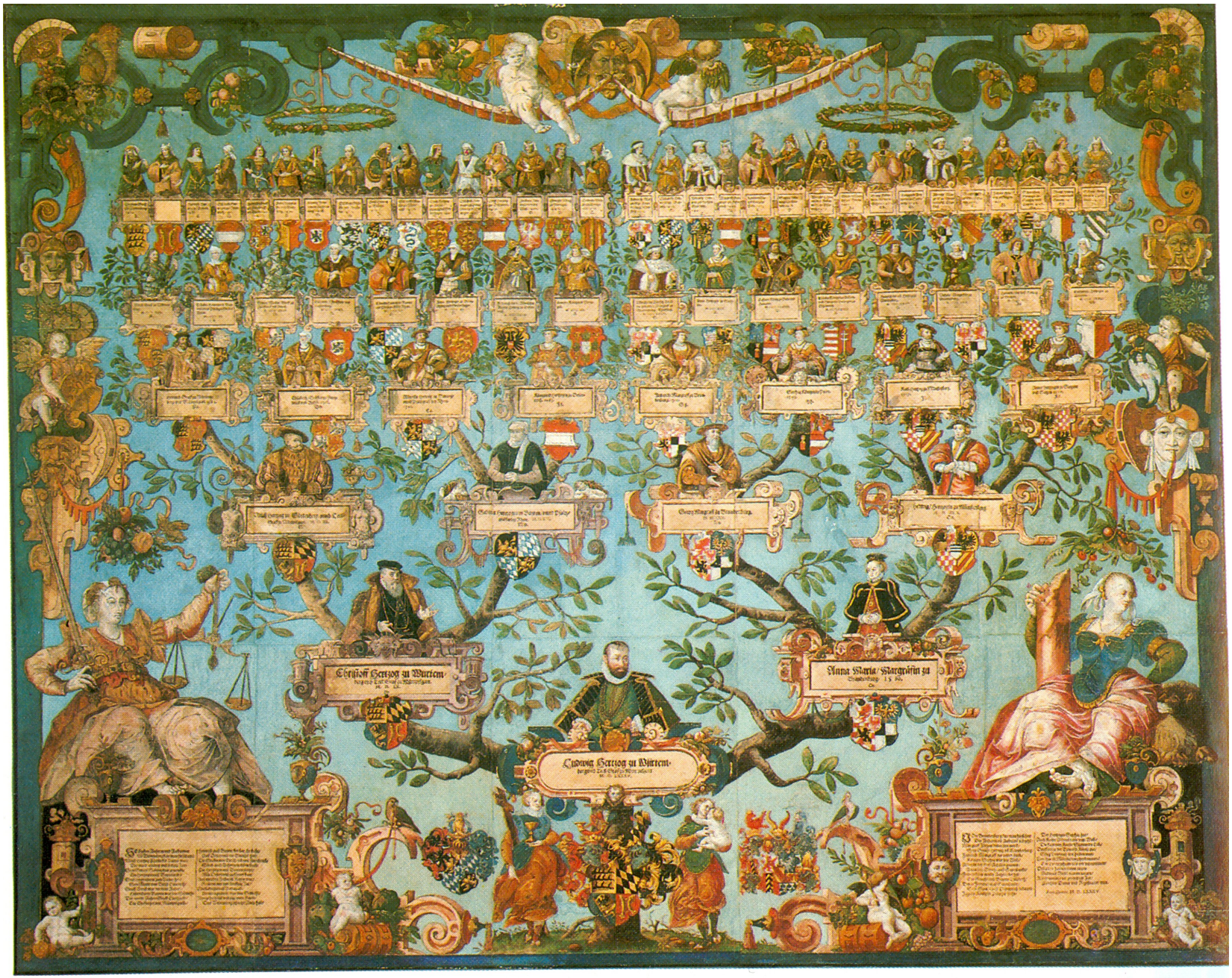
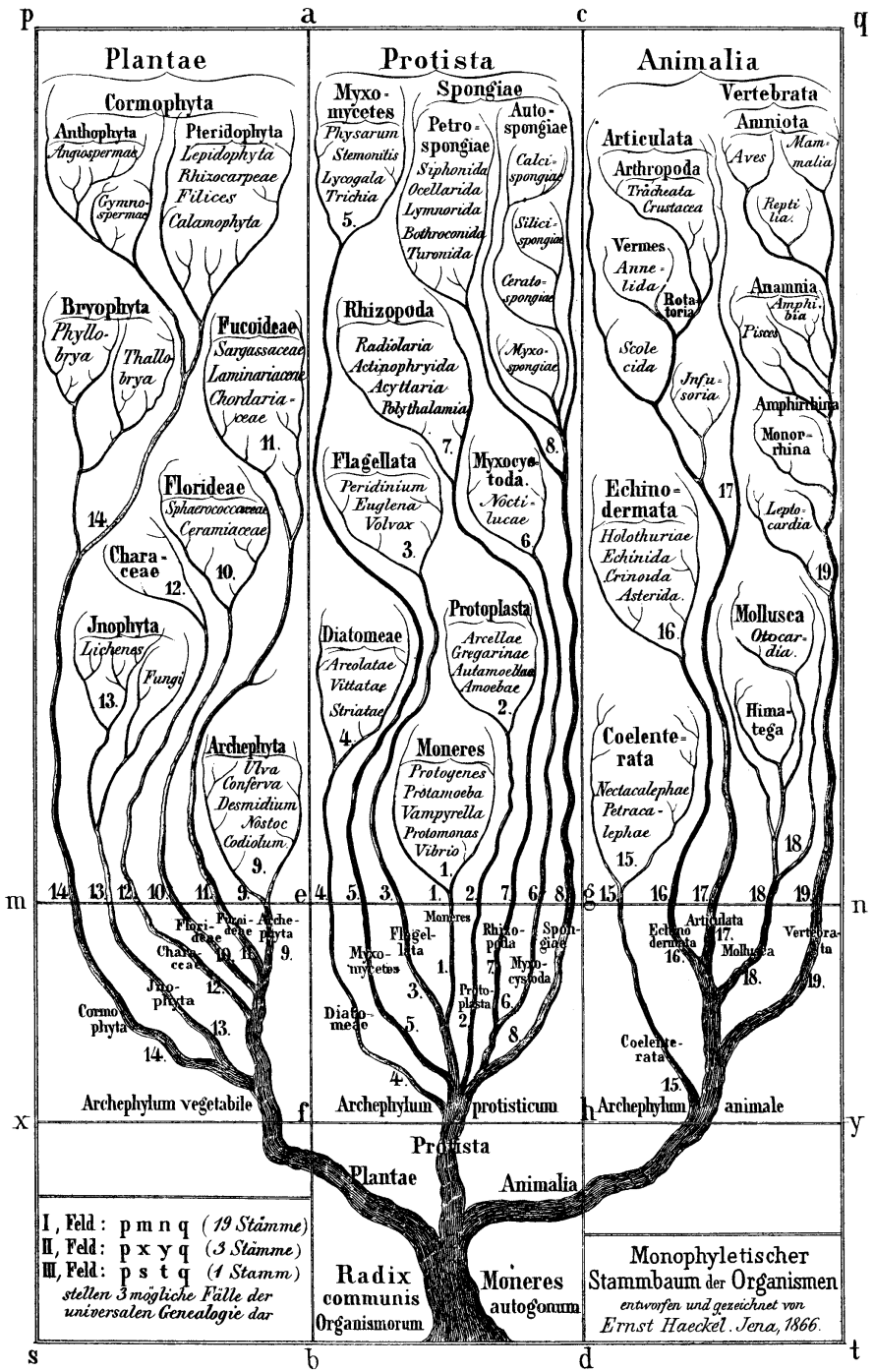
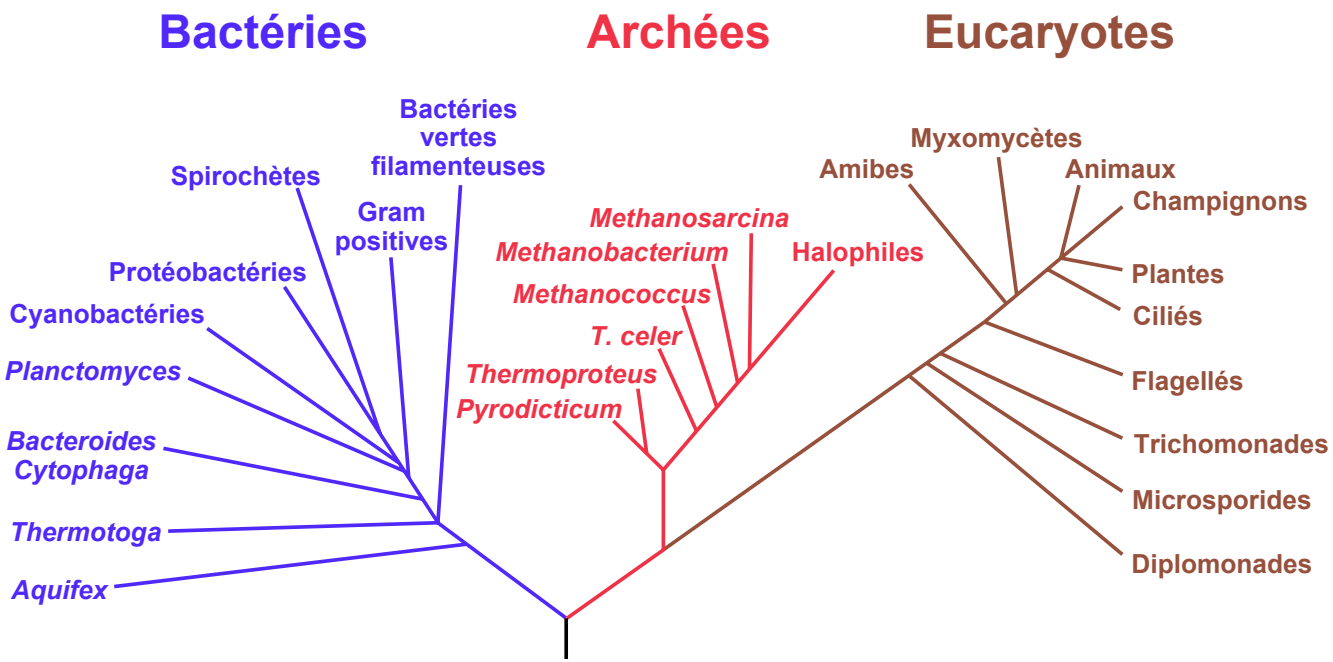
Algorithmique 1
Les arbres

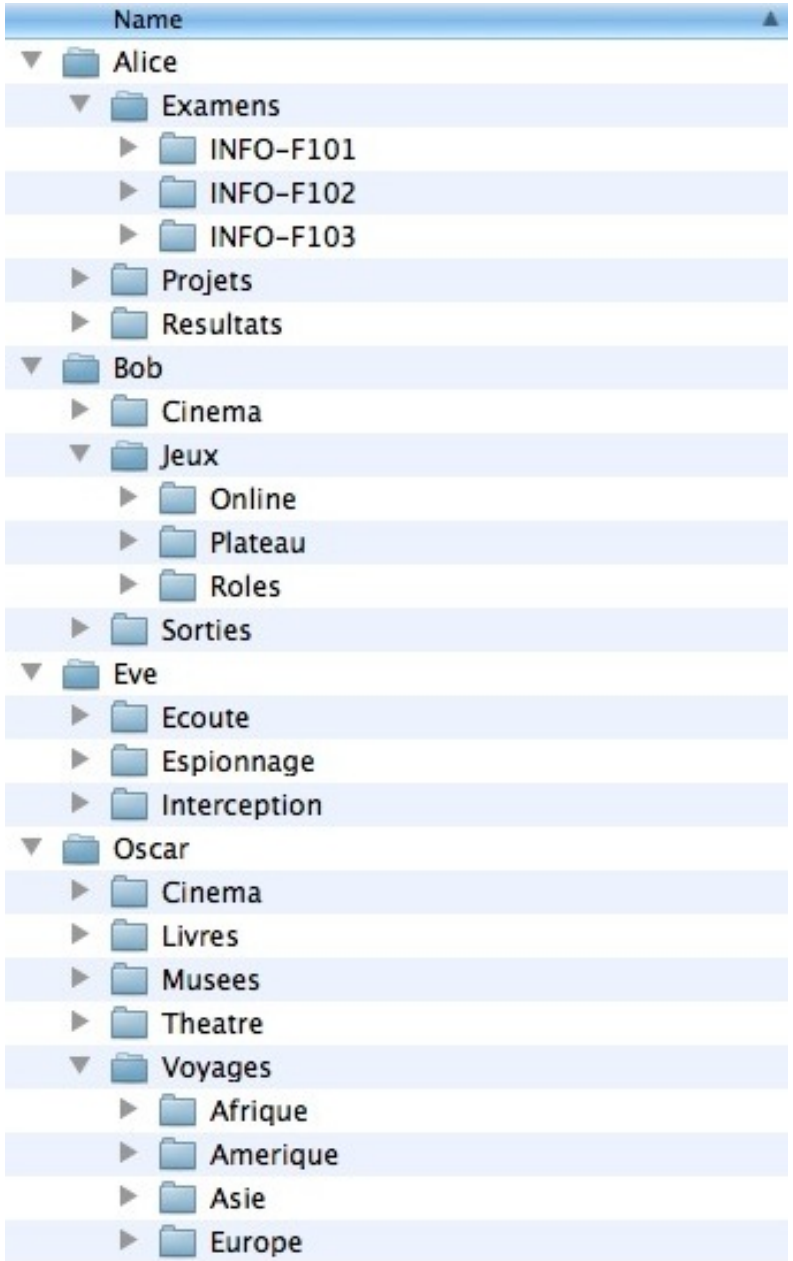
Olivier Markowitch


Christoff Hertzog zu Wirtem-
Ludowig Hertzog zu Wirtem-
Anna Maria/ Marggrävin zu


p
a
c
q
Plantae
Protista
Animalia
Cormophyta
Anthophyta
Angiospermae
Gymnospermae
Pteridophyta
Lepidophyta
Rhizocarpeae
Filices
Calamophyta
Bryophyta
Phyllobrya
Thallobrya
Fucoideae
Sargassaceae
Laminariaceae
Chordariaceae
11.
Florideae
Sphaerococcaceae
Ceramiaceae
14.
Characeae
12.
10.
Jnophyta
Lichenes
Fungi
13.
Archephyta
Ulva
Conferva
Desmidium
Nostoc
Codiolum
9.
Myxomycetes
Physarum
Stemonitis
Lycogala
Trichia
5.
Spongiae
Petrospongiae
Siphonida
Ocellarida
Lymnorida
Bathroconida
Turonida
Autospongiae
Calcispongiae
Silicispongiae
Ceratospongiae
Myxospongiae
Rhizopoda
Radiolaria
Actinophryida
Acyttaria
Polythalamia
7.
8.
Flagellata
Peridinium
Euglena
Volvox
3.
Myxocystoda.
Noctilucae
6.
Protoplasta
Arcellae
Gregarinae
Autamoebae
Amoebae
2.
Diatomeae
Areolatae
Vittatae
Striatae
4.
Moneres
Protogenes
Protamoeba
Vampyrella
Protomonas
Vibrio
1.
Vertebrata
Amniota
Aves
Mammalia
Reptilia.
Anamnia
Amphibia
Pisces
Amphirhina
Monorhina
Leptocardia
19.
Mollusca
Otocardia.
Himatega
18.
Articulata
Arthropoda
Tracheata
Crustacea
Vermes
Annelida
Rotatoria
Scolecida
Infusoria
Echinodermata
Holothuriae
Echinida
Crinoida
Asterida.
16.
17
Coelenterata
Nectacalephae
Petracalephae
15.
m
e
g
n
x
f
h
y
s
b
d
t
Archephylum vegetabile
Archephylum protisticum
Archephylum animale
Protista
Plantae
Animalia
I, Feld: p m n q (19 Stämme)
II, Feld: p x y q (3 Stämme)
III, Feld: p s t q (1 Stamm)
stellen 3 mögliche Fälle der universalen Genealogie dar
Radix communis Organismorum
Moneres autogonum
Monophyletischer Stammbaum der Organismen
entworfen und gezeichnet von
Ernst Haeckel. Jena, 1866.

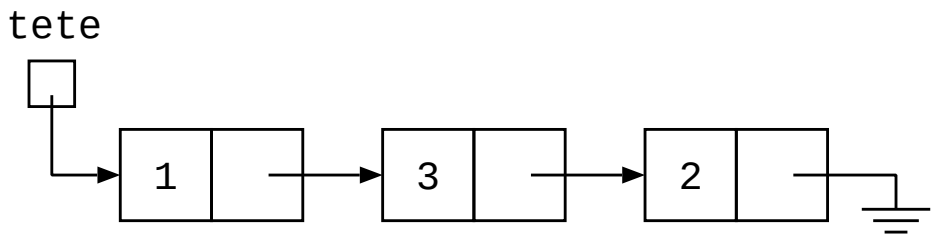
Arbre phylogénétique de la vie




Name
Alice
Examens
INFO-F101
INFO-F102
INFO-F103
Projets
Resultats
Bob
Cinema
Jeux
Online
Plateau
Roles
Sorties
Eve
Ecoute
Espionnage
Interception
Oscar
Cinema
Livres
Musees
Theatre
Voyages
Afrique
Amerique
Asie
Europe

```
> ls -R | grep ":$" | sed -e 's/:$//' -e 's/[^-][^\/]*\//--/g' -e 's/^/   /' -e 's/-/|/'
   |-Alice
   |---Examens
   |-----INFO-F101
   |-----INFO-F102
   |-----INFO-F103
   |---Projets
   |---Resultats
   |-Bob
   |---Cinema
   |---Jeux
   |-----Online
   |-----Plateau
   |-----Roles
   |---Sorties
   |-Eve
   |---Ecoute
   |---Espionnage
   |---Interception
   |-Oscar
   |---Cinema
   |---Livres
   |---Musees
   |---Theatre
   |---Voyages
   |-----Afrique
   |-------Gabon
   |-------Kenya
   |-------Tanzanie
   |-----Amerique
   |-------Bresil
   |-------Canada
   |-------USA
   |-----Asie
   |-----Europe
   |-------Belgique
   |-------France
   |-------Italie
```

Arbres



Les arbres sont une première généralisation des listes linéaires

On peut considérer les arbres comme des structures liées à deux dimensions

Les nœuds et arêtes

Un arbre est une collection de nœuds (ou sommets) et d'arêtes qui relient deux nœuds

La racine d'un arbre est un nœud spécifique qui représente le premier nœud de l'arbre en question

Chemins dans un arbre

Un chemin dans un arbre est une suite de nœuds distincts dans laquelle deux nœuds successifs sont reliés par une arête

Cette collection d'arêtes et de nœuds doit respecter la condition qu'il n'existe qu'un et un seul chemin entre la racine de l'arbre et chacun des autres nœuds de cet arbre

Arbres et sous-arbres

Une définition récursive d'un arbre est la suivante : un arbre est soit un arbre vide, soit un arbre qui comporte un nœud particulier appelé racine ainsi qu'un ensemble de $m \geq 0$ arbres disjoints, sans nœuds communs, attachés à la racine

Les arbres attachés à un nœud sont aussi appelés sous-arbres

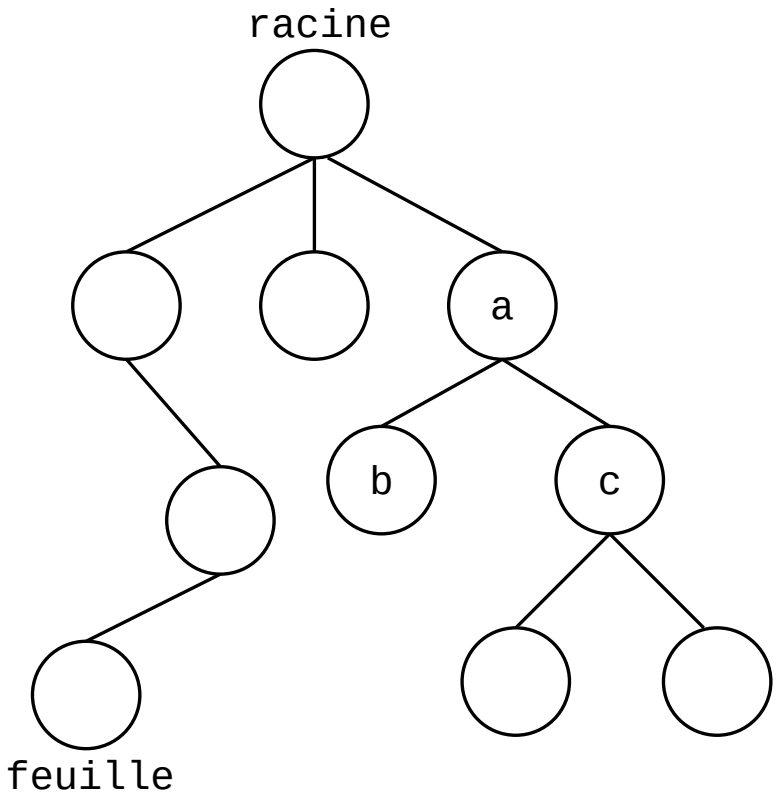
Le degré d'un nœud

Le degré d'un nœud est le nombre de sous-arbres attachés à ce nœud

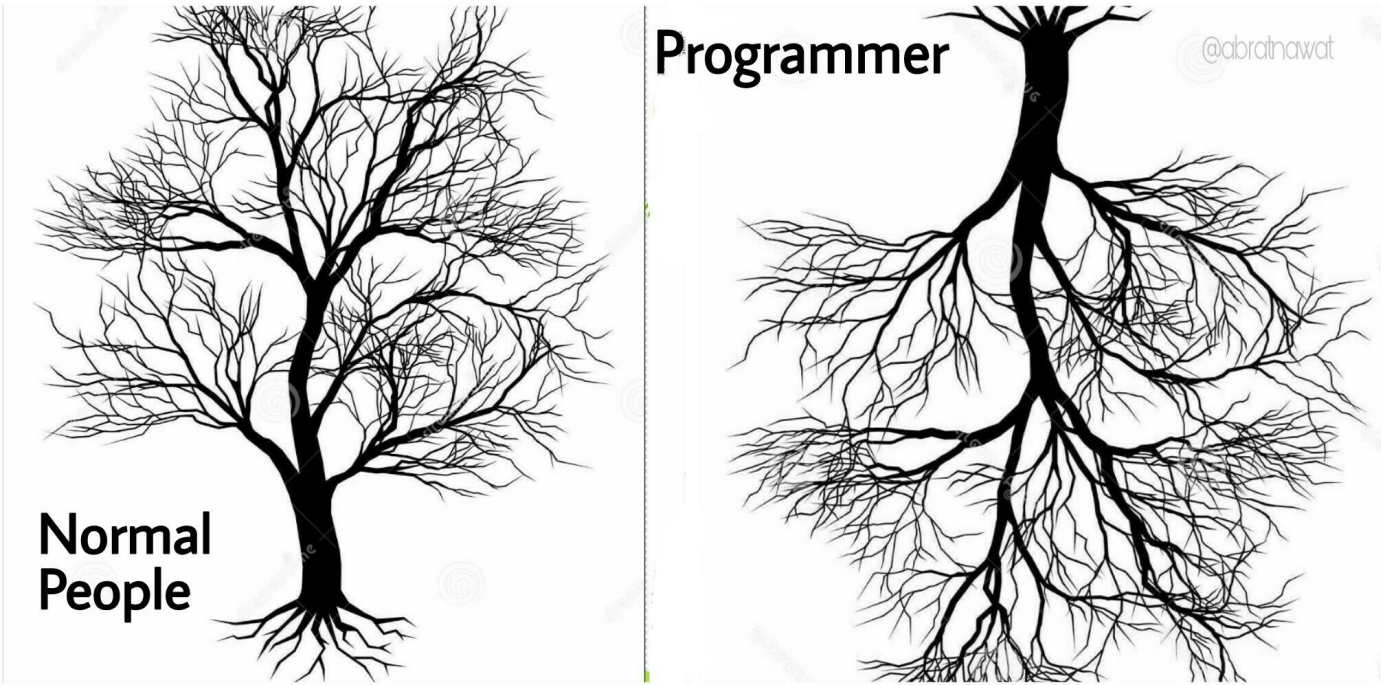
Les feuilles sont des nœuds de degré égal à zéro

Les nœuds qui ne sont pas des feuilles sont aussi appelés nœuds internes

Arbre



Arbre



Le niveaux d'un nœud

Le niveau d'un nœud est calculé de la manière suivante :

- $niveau(racine) = 0$
- pour tout autre nœud s attaché par une arête au nœud t sur le chemin entre la racine et s : $niveau(s) = niveau(t) + 1$

Les parentés dans un arbre

Le nœud s est appelé enfant ou fils du nœud t et le nœud t est appelé père du nœud s

La hauteur d'un arbre

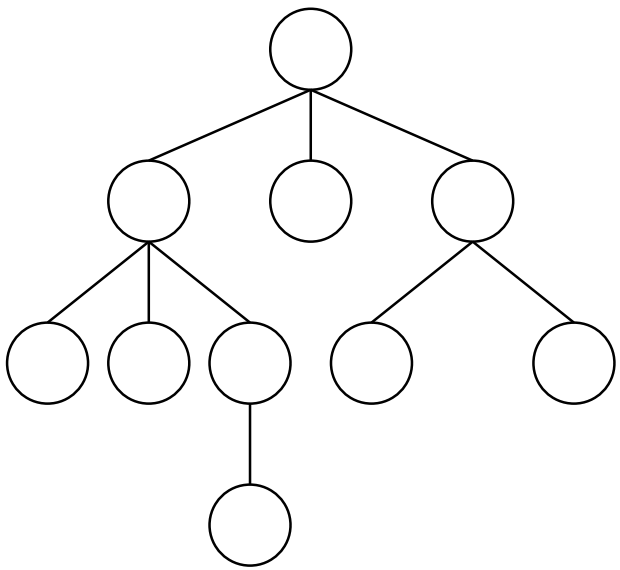
La hauteur d'un arbre est le niveau maximum de l'arbre, et donc la distance la plus grande d'un nœud à la racine

Nombre d'arêtes d'un arbre

Un arbre de n nœuds possède $n - 1$ arêtes, car chaque nœud, à l'exception de la racine, est relié par une arête à son unique père

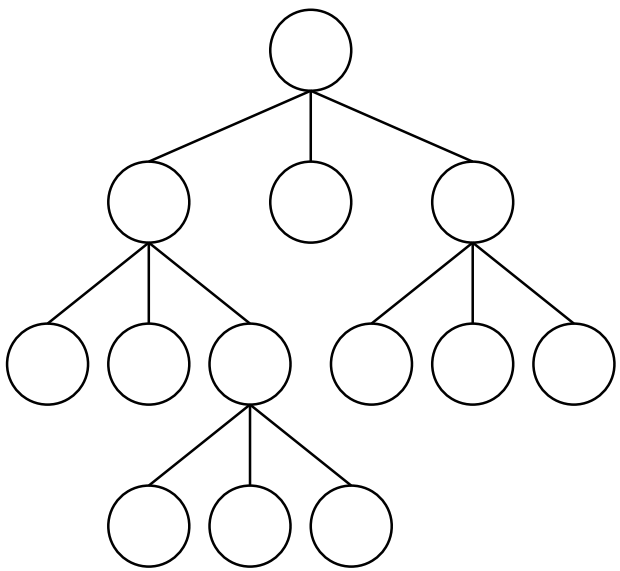
Arbres m-aire

Si un arbre est tel que chaque nœud qui n'est pas une feuille a au plus m fils, l'arbre est dit m-aire



Arbres pleins

Un arbre m-aire est dit plein lorsque chaque nœud a exactement 0 fils ou exactement m fils

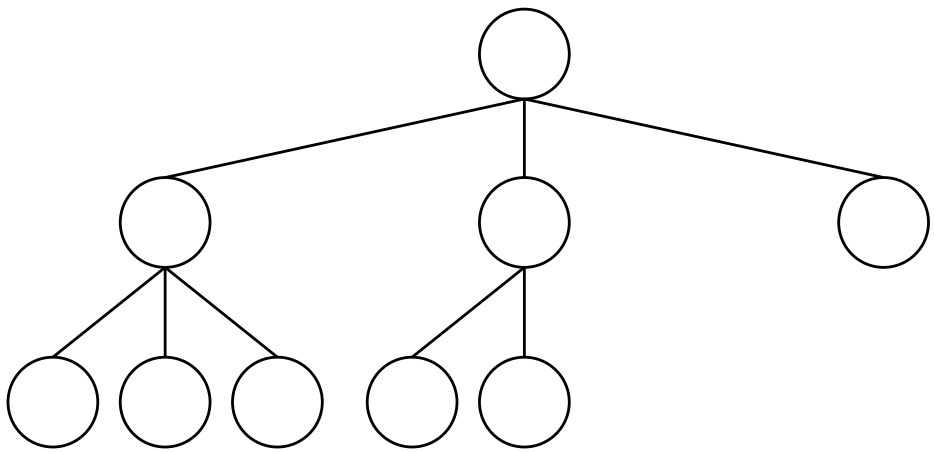


Niveaux remplis d'un arbre

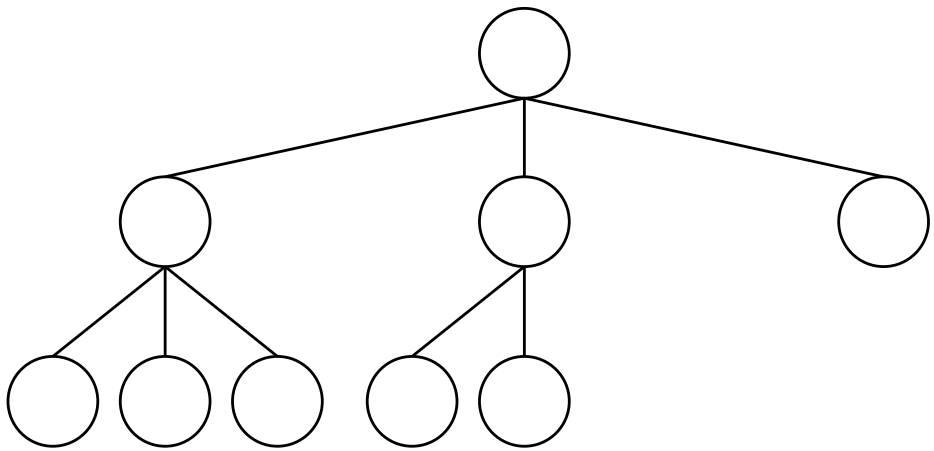
Le niveau i d'un arbre m-aire est dit être rempli lorsque tous les nœuds du niveau $i-1$ ont exactement m fils

Arbres complets

Un arbre m-aire de hauteur n est dit complet lorsque tous les niveaux de l'arbre sont remplis, à l'exception du niveau n qui peut n'être que partiellement rempli à la condition que tous les nœuds de ce niveau se trouvent le plus à gauche de ce niveau



Arbres complets

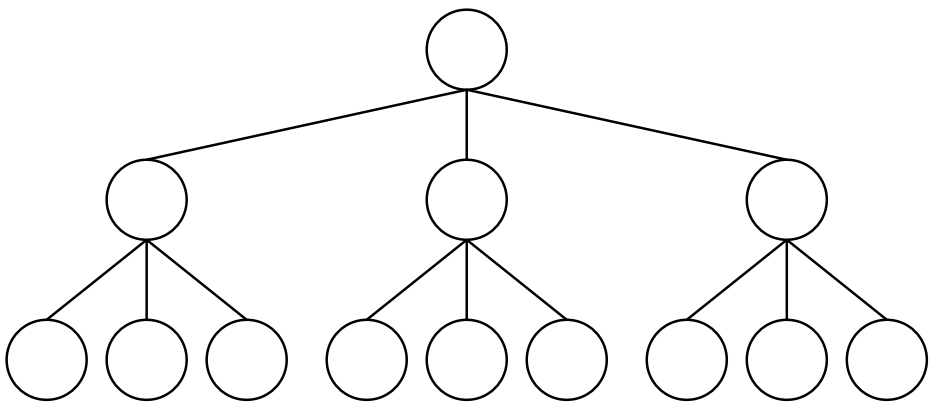


Un arbre m-aire complet de hauteur n possède m^k nœuds au niveau $k < n$ et entre 0 et m^n nœuds rangés à gauche au niveau n

Arbres parfaits

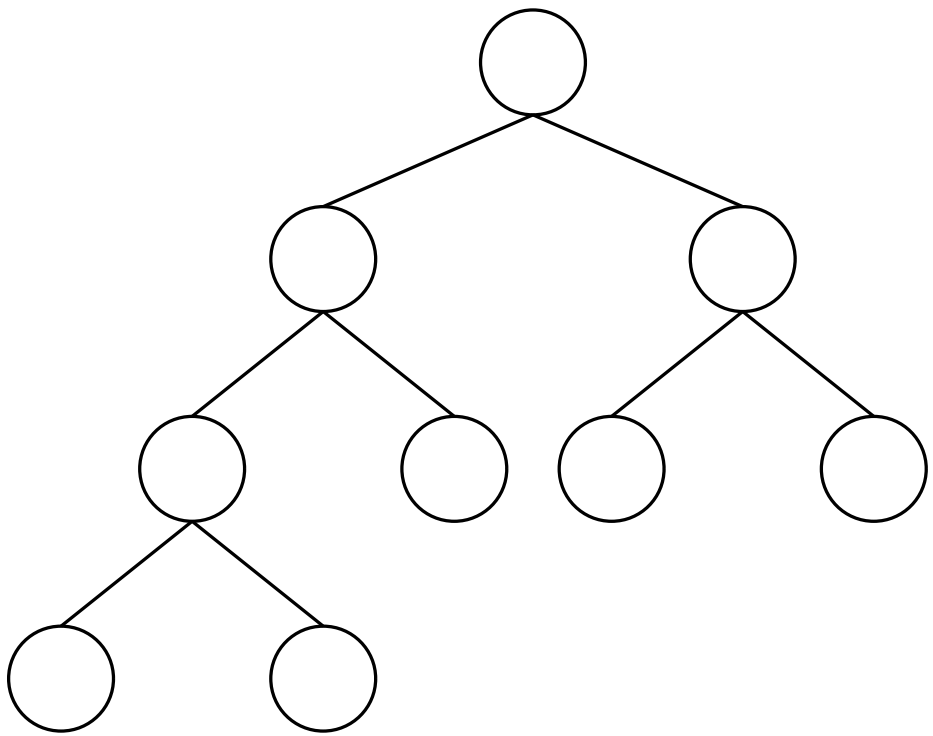
Un arbre m-aire est parfait lorsque toutes les feuilles sont au même niveau et lorsque tous les nœuds internes ont donc un degré égal à m

La figure ci-dessous donne un exemple d'arbre 3-aire parfait



Arbres binaires

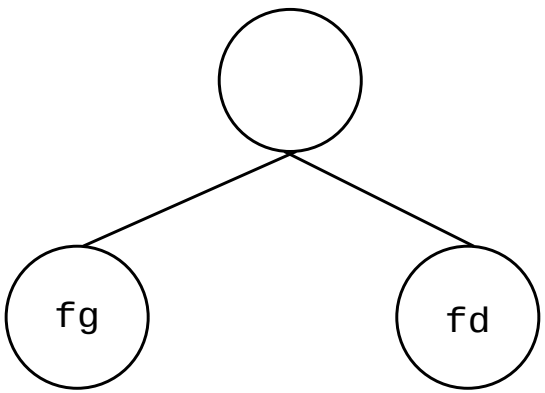
Le type d'arbre m-aire le plus classique apparaît lorsque m vaut 2, on parle alors d'arbres binaires



Arbres binaires

Un arbre binaire peut aussi être plein, complet ou parfait

De plus, chaque nœud ayant des fils a un fils gauche et/ou un fils droit



Arbres binaires : théorème 1

Théorème

Un arbre binaire parfait de hauteur n possède 2^n feuilles

Appelons f_i le nombre de feuilles d'un arbre binaire de hauteur i

Nous avons qu'un arbre parfait de deux feuilles est de hauteur 1, $f_1 = 2^1 = 2$

Supposons la propriété vraie pour 2^n feuilles, $f_n = 2^n$

Un arbre parfait de 2^n feuilles a, par hypothèse d'induction, une hauteur égale à n

Arbres binaires : théorème 1 (suite)

Théorème

Un arbre binaire parfait de hauteur n possède 2^n feuilles

Appelons f_i le nombre de feuilles d'un arbre binaire de hauteur i. Nous avons donc $f_1 = 2^1 = 2$ et $f_n = 2^n$

Considérons 2^{n+1} feuilles. Nous pouvons écrire $2^{n+1} = 2^n 2$

Aussi, $2^n 2$ représente le fait de rajouter deux fils à toutes les feuilles de l'arbre parfait de hauteur n, ce qui consiste bien en l'ajout d'un niveau supplémentaire dans l'arbre parfait

Nous avons donc bien un arbre parfait de hauteur $n+1$ composé de $f_{n+1} = 2^{n+1}$ feuilles

Arbres binaires : théorème 2

Théorème

Un arbre binaire parfait de hauteur n possède $2^{n+1} - 1$ nœuds

Appelons s_i le nombre de nœuds d'un arbre binaire de hauteur i

Si la hauteur de l'arbre parfait vaut $n = 0$, un seul nœud compose l'arbre, et le nombre de nœuds vaut bien $s_0 = 2^1 - 1 = 1$

Supposons que la propriété est vraie pour une hauteur n,
$s_n = 2^{n+1} - 1$

Arbres binaires : théorème 2 (suite)

Théorème

Un arbre binaire parfait de hauteur n possède $2^{n+1} - 1$ nœuds

Appelons s_i le nombre de nœuds d'un arbre binaire de hauteur i.
Supposons que la propriété est vraie pour une hauteur égale 0 et n

Pour un arbre binaire parfait de hauteur $n+1$, nous avons le nombre de nœuds d'un arbre de hauteur n auxquels s'ajoutent les feuilles du niveau $n+1$:

$$s_{n+1} = s_n + f_{n+1} = 2^{n+1} - 1 + 2^{n+1} = 2^{n+2} - 1$$

Arbres binaires : théorème 3

Théorème

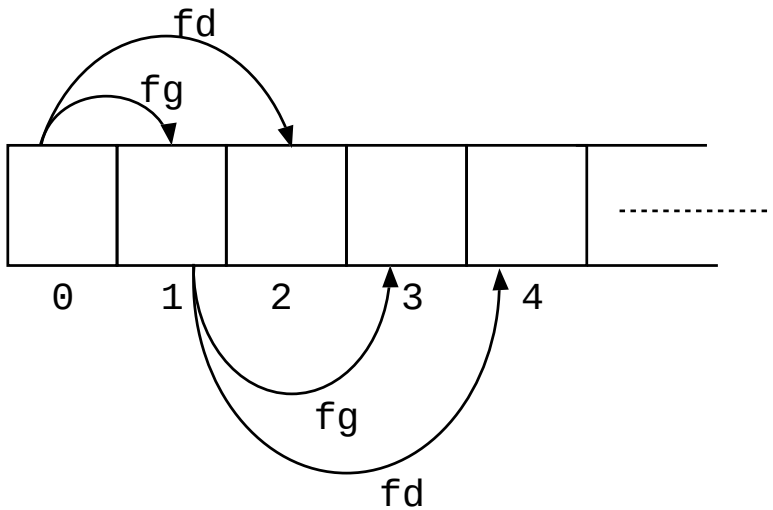
Un arbre binaire qui possède m nœuds internes, possède au plus $m + 1$ feuilles

Puisqu'un arbre binaire parfait de hauteur n possède $2^{n+1} - 1$ nœuds dont $f_n = 2^n$ feuilles ...

... cet arbre contient $m = 2^{n+1} - 1 - 2^n = 2^n - 1 = f_n - 1$ nœuds internes

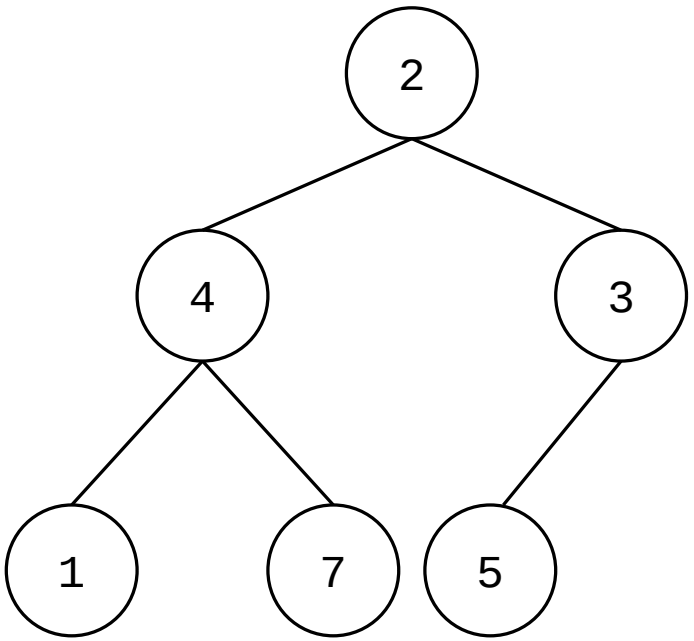
donc $f_n = m + 1$

Arbre binaire au moyen d'un tableau



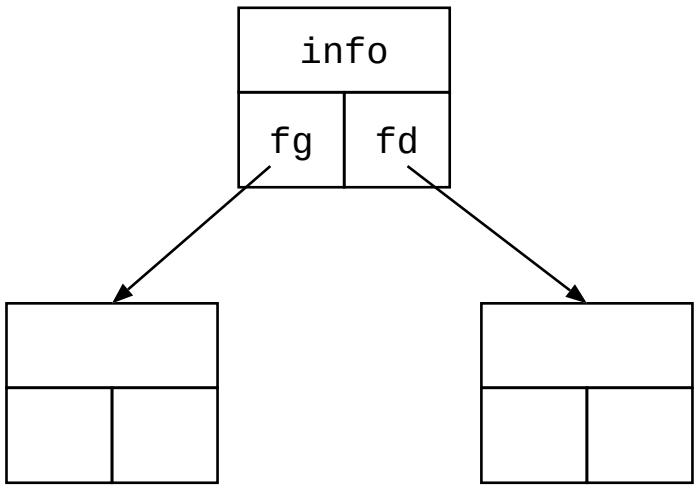
Les fils gauche et droit du nœud d'indice i se trouvent respectivement aux indices $2i + 1$ et $2i + 2$

Arbre binaire au moyen d'un tableau

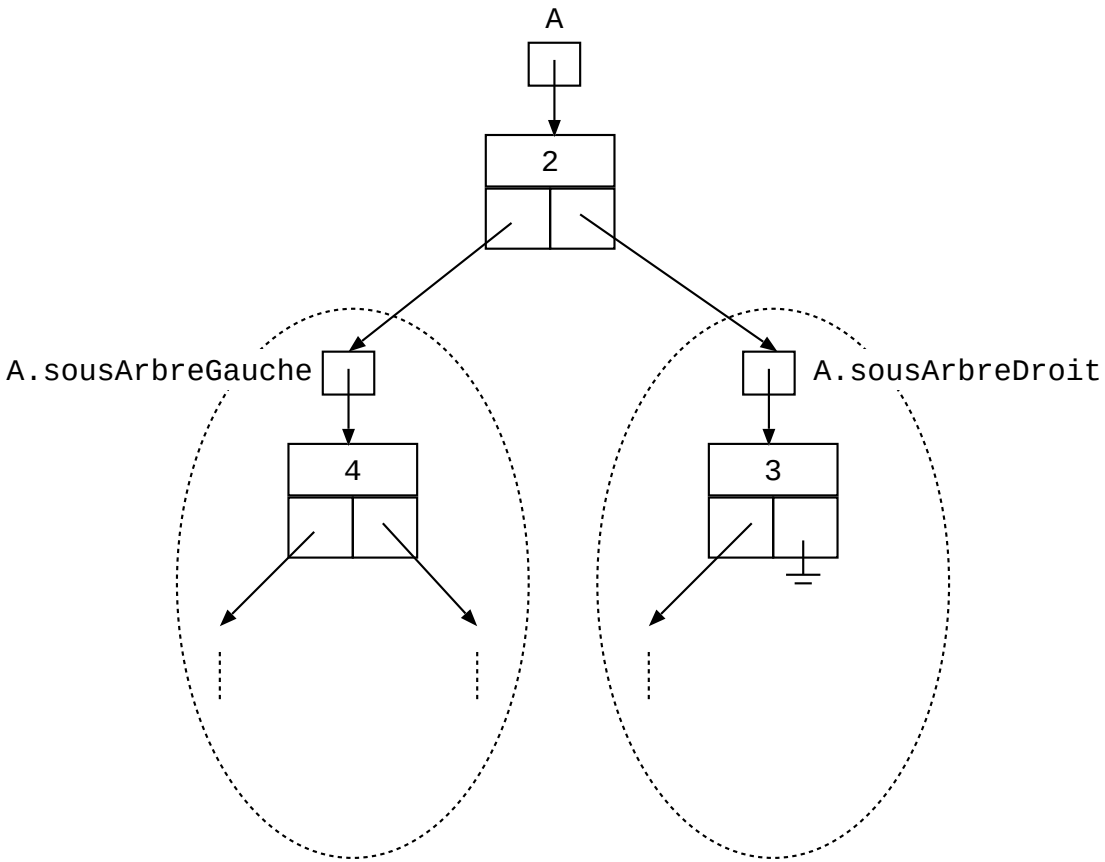


0	1	2	3	4	5
2	4	3	1	7	5

Arbre binaire au moyen de pointeurs



Vision récursive d'un arbre binaire



Interface d'un arbre récursif

L'interface de base d'un arbre binaire récursif consiste en les primitives suivantes :

- getRootVal : indique la valeur contenue dans la racine de l'arbre considéré
- setRootVal : modifie la valeur contenue dans la racine de l'arbre considéré

Interface d'un arbre binaire

Interface de base d'un arbre binaire récursif (suite) :

- getLeftChild : donne accès au sous-arbre gauche de la racine de l'arbre considéré
- getRightChild : donne accès au sous-arbre droit de la racine de l'arbre considéré

Interface d'un arbre binaire

Interface de base d'un arbre binaire récursif (suite) :

- `modifyLeft` : modifie le sous-arbre gauche de la racine de l'arbre considéré
- `modifyRight` : modifie le sous-arbre droit de la racine de l'arbre considéré

Arbre binaire via les listes Python

```
def BinaryTree(r):
    return [r, [], []]

def modifyLeft(root,value):
    t = root.pop(1)
    root.insert(1,[value,[],[]])
    return root

def modifyRight(root,value):
    t = root.pop(2)
    root.insert(2,[value,[],[]])
    return root

def getRootVal(root):
    return root[0]

def setRootVal(root,newVal):
    root[0] = newVal
```

Arbre binaire via les listes Python

```
def getLeftChild(root):
    return root[1]

def getRightChild(root):
    return root[2]
```

Arbre binaire via les listes Python

```
>>> r=BinaryTree(1)
>>> r
[ 1, [ ], [ ] ]
>>> modifyLeft(r,2)
[ 1, [ 2, [ ], [ ] ], [ ] ]
>>> modifyRight(r,3)
[ 1, [ 2, [ ], [ ] ], [ 3, [ ], [ ] ] ]
>>> s=getLeftChild(r)
>>> s
[ 2, [ ], [ ] ]
>>> modifyLeft(s,4)
[ 2, [ 4, [ ], [ ] ], [ ] ]
>>> r
[ 1, [ 2, [ 4, [ ], [ ] ], [ ] ], [ 3, [ ], [ ] ] ]
>>> modifyLeft(r,5)
[ 1, [ 5, [ ], [ ] ], [ 3, [ ], [ ] ] ]
```

Arbre binaire

```
class BinaryTree:
    def __init__(self,item):
        self.info = item
        self.left = None
        self.right = None
```

Arbre binaire

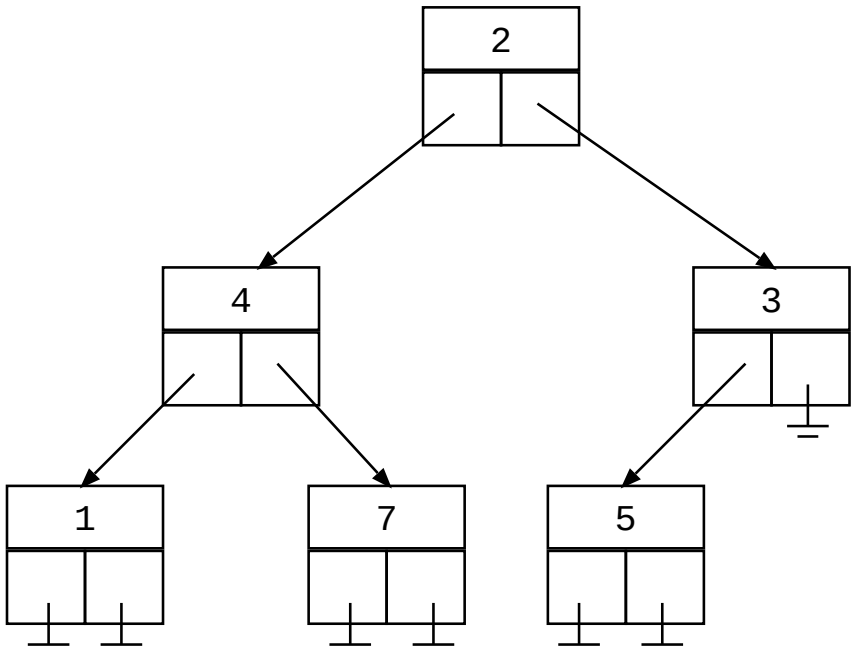
```
    def modifyLeft(self,item):
        self.left = BinaryTree(item)

    def modifyRight(self,item):
        self.right = BinaryTree(item)

    def getRootVal(self):
        return self.info

    def setRootVal(self,item):
        self.info = item

    def getLeftChild(self):
        return self.left

    def getRightChild(self):
        return self.right
```

Arbre binaire : vision non récursive

Un arbre binaire peut aussi être vu comme composé physiquement de nœuds

Un nœud de l'arbre possède donc alors explicitement un fils gauche et un fils droit (qui sont des nœuds et non plus des sous-arbres)

Arbre binaire : vision non récursive



Interface d'un nœud d'un arbre binaire non récursif

L'interface de base d'un nœud d'un arbre binaire non récursif consiste en les primitives suivantes :

- getInfo : indique la valeur contenue dans le nœud
- getLeft : donne accès au nœud fils gauche du nœud courant
- getRight : donne accès au nœud fils droit du nœud courant

Interface d'un nœud d'un arbre binaire non récursif

Interface de base d'un nœud d'un arbre binaire non récursif (suite) :

- setInfo : modifie la valeur contenue dans le nœud
- setLeft : modifie le nœud fils gauche du nœud courant
- setRight : modifie le nœud fils droit du nœud courant

Interface d'un arbre binaire non récursif

L'interface de base d'un arbre binaire non récursif consiste quant à elle en les primitives suivantes :

- getRoot : donne accès au nœud qui est à la racine de l'arbre considéré
- `modifyLeft` : modifie le nœud fils gauche du nœud considéré dans l'arbre
- `modifyRight` : modifie le nœud fils droit du nœud considéré dans l'arbre

Arbre binaire : vision non récursive

```
class Node:
    def __init__(self,item):
        self.info = item
        self.left = None
        self.right = None

    def getInfo(self):
        return self.info

    def getLeft(self):
        return self.left

    def getRight(self):
        return self.right
```

Arbre binaire : vision non récursive

```
    def setInfo(self,newinfo):
        self.info = newinfo

    def setLeft(self,newleft):
        self.left = newleft

    def setRight(self,newright):
        self.right = newright
```

Arbre binaire : vision non récursive

```
class BinaryTree:
    def __init__(self,item):
        self.root = Node(item)

    def treeGetRoot(self):
        return self.root

    def treeSetLeft(self,base,item):
        base.setLeft(Node(item))

    def treeSetRight(self,base,item):
        base.setRight(Node(item))

    def treeGetLeft(self,base):
        return base.getLeft()

    def treeGetRight(self,base):
        return base.getRight()
```

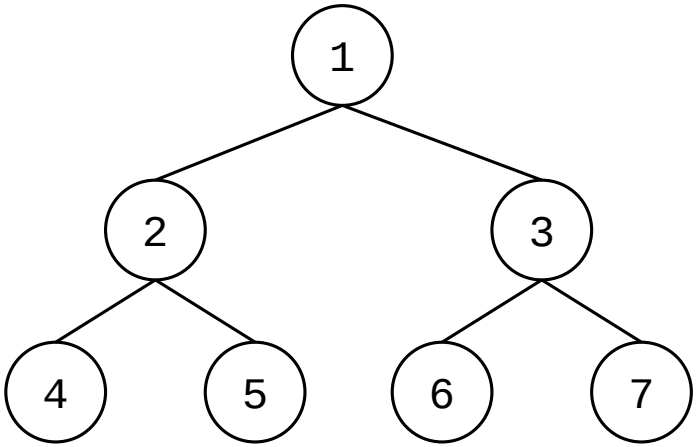
Parcours d'arbres binaires

Un traitement réalisable sur un arbre consiste à en parcourir les nœuds

Il existe différentes manières de parcourir un arbre en fonction de l'ordre dans lequel nous considérons les nœuds de cet arbre

Les algorithmes qui suivent se basent sur un arbre binaire vu de manière récursive

Parcours préfixé

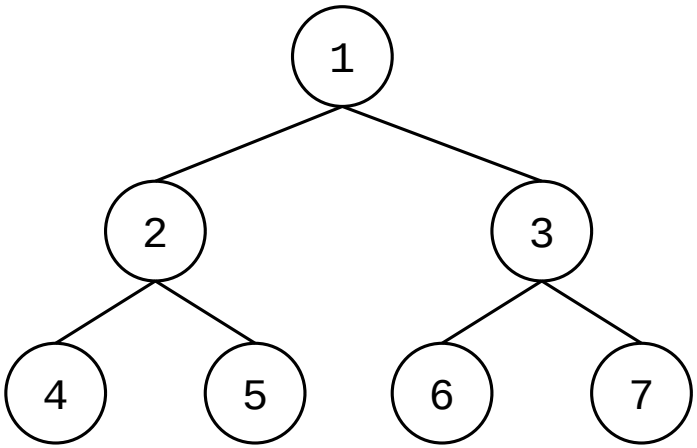


Le parcours préfixé ou parcours en préordre consiste à traiter d'abord la racine, puis le sous-arbre gauche et enfin le sous-arbre droit

Sur notre exemple, cela donne l'ordre de parcours suivant :

$$1 - 2 - 4 - 5 - 3 - 6 - 7$$

Parcours infixé

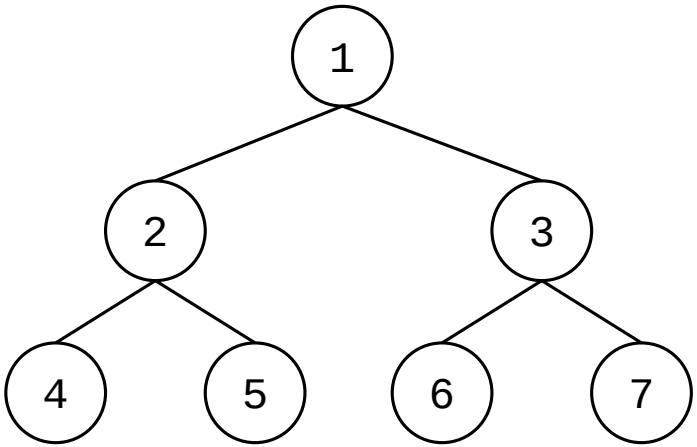


Le parcours infixé ou parcours en inordre, ou encore parcours symétrique, consiste à traiter d'abord le sous-arbre gauche, puis la racine et enfin le sous-arbre droit

Sur notre exemple, cela donne l'ordre de parcours suivant :

$$4 - 2 - 5 - 1 - 6 - 3 - 7$$

Parcours suffixé

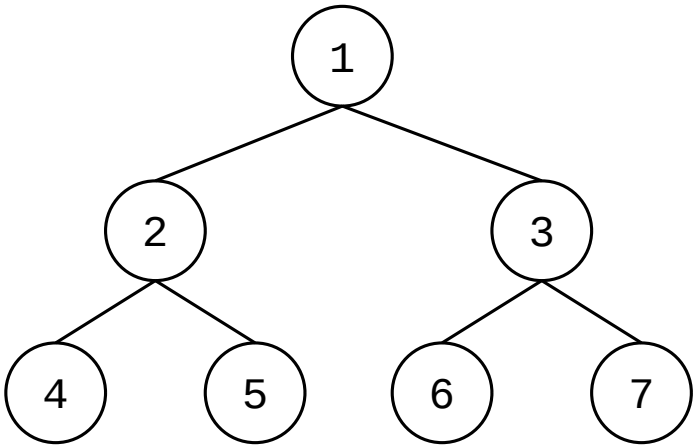


Le parcours suffixé ou parcours en postordre traite d'abord le sous-arbre gauche, puis le sous-arbre droit et enfin la racine

Ce qui donne sur notre exemple :

$$4 - 5 - 2 - 6 - 7 - 3 - 1$$

Parcours par niveau



Le parcours par niveau ou parcours en largeur traite les nœuds niveau par niveau, en partant de la racine

Sur notre exemple un tel parcours donne :

$$1 - 2 - 3 - 4 - 5 - 6 - 7$$

Parcours préfixé d'arbres récursifs

```
def preorder(tree):
    if tree != None:
        print tree.getRootVal()
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

```
def preorder(tree):
    while tree != None:
        print tree.getRootVal()
        preorder(tree.getLeftChild())
        tree = tree.getRightChild()
```

Parcours infixé d'arbres récursifs

```
def inorder(tree):
    if tree != None:
        inorder(tree.getLeftChild())
        print tree.getRootVal()
        inorder(tree.getRightChild())
```

```
def inorder(tree):
    while tree != None:
        inorder(tree.getLeftChild())
        print tree.getRootVal()
        tree = tree.getRightChild()
```

Parcours suffixé d'arbres récursifs

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print tree.getRootVal()
```

Parcours par niveau d'arbres récursifs

```
def niveau(tree):
    f=Queue()
    f.insert(tree)
    while not f.isEmpty():
        n = f.remove()
        if n != None:
            print n.getRootVal()
            f.insert(n.getLeftChild())
            f.insert(n.getRightChild())
```

Parcours préfixé d'arbres non récursifs

```
def preorder(noeud):
    while noeud != None:
        print noeud.getInfo()
        preorder(noeud.getLeft())
        noeud = noeud.getRight()
```

Parcours infixé d'arbres non récursifs

```
def inorder(noeud):
    while noeud != None:
        inorder(noeud.getLeft())
        print noeud.getInfo()
        noeud = noeud.getRight()
```

Parcours suffixé d'arbres non récursifs

```
def postorder(noeud):
    if noeud != None:
        postorder(noeud.getLeft())
        postorder(noeud.getRight())
        print noeud.getInfo()
```

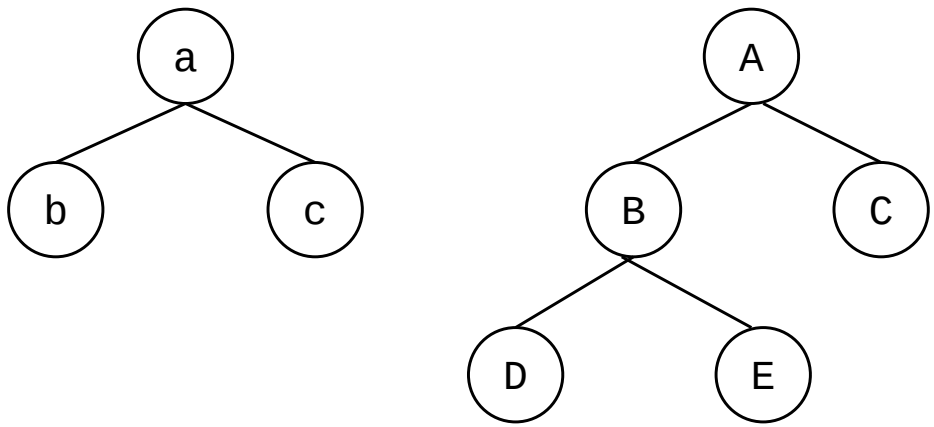
Parcours par niveau d'arbres non récursifs

```
def niveau(noeud):
    f=Queue()
    f.insert(noeud)
    while not f.isEmpty():
        n = f.remove()
        if n != None:
            print n.getInfo()
            f.insert(n.getLeft())
            f.insert(n.getRight())
```

Forêts et arbres m-aires

Une forêt est un ensemble d'arbres

Par exemple, nous pouvons dire que l'ensemble des sous-arbres de la racine d'un arbre *m*-aire est une forêt



Un nœud possède zéro, un ou plusieurs fils, et zéro, un ou plusieurs frères

Interface d'une forêt

L'interface d'un tel type de données abstrait peut être pour les nœuds composant la forêt :

- getRootVal : indique la valeur contenue dans la racine de l'arbre considéré
- setRootVal : modifie la valeur contenue dans la racine de l'arbre considéré

Interface d'une forêt

Interface de base d'une forêt (suite) :

- getChild : donne accès au premier fils de la racine de l'arbre considéré
- getBrother : donne accès au frère de la racine de l'arbre considéré

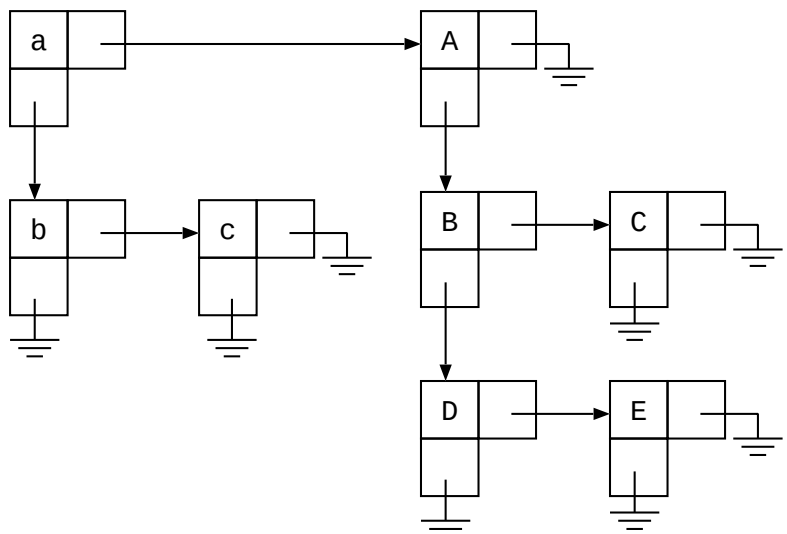
Interface d'une forêt

Interface de base d'une forêt (suite) :

- `modifyChild` : modifie les fils de la racine de l'arbre considéré
- `modifyBrother` : modifie les frères de la racine de l'arbre considéré

Organisation d'une forêt ou d'un arbre m-aire)

Nous accèdons aux différents fils d'un nœud en accédant tout d'abord à son premier fils, puis en accédant aux autres fils via les accès « frère »de ce premier fils



Forêt ou arbre m-aire

```
class Foret:
    def __init__(self,item):
        self.info = item
        self.child = None
        self.brother = None

    def getRootVal(self,):
        return self.info

    def setRootVal(self,item):
        self.info = item
```

Forêt ou arbre m-aire

```
    def getChild(self):
        return self.child

    def getBrother(self):
        return self.brother

    def modifyChild(self,newNode):
        self.child = Foret(newNode)

    def modifyBrother(self,newNode):
        self.brother = Foret(newNode)
```

Parcours en préordre d'une forêt ou d'un arbre m-aire

```
def preordre(forest):
    while forest != None:
        print forest.getRootVal()
        preorder(forest.getChild())
        forest = forest.getBrother()
```

Parcours en postordre d'une forêt ou d'un arbre m-aire

```
def postorder(forest):
    while forest != None:
        postorder(forest.getChild())
        print forest.getRootVal()
        forest = forest.getBrother()
```

Parcours par niveau d'une forêt ou d'un arbre m-aire

```
def niveau(forest):
    f=Queue()
    f.insert(forest)
    while not f.isEmpty():
        n = f.remove()
        while n != None:
            print n.getRootVal()
            f.insert(n.getChild())
            n = n.getBrother()
```