Recursivitat 2

ETSEIB/GIE

17 de novembre de 2023

1. Exercicis
2. Cerca d'un element en una llista
3. Cerca d'un element en una llista ordenada
4. Arbres

Suma llista d'enters

Dissenyeu una funció recursiva suma(llista) que retorni la suma de tots els enters de llista

... En el cas que la llista no sigui mai buida, una solució recursiva no final:

```
def suma(llista):
    if len(llista)==1:
        r= llista[0]
    else:
        r= llista[0]+suma(llista[1:])
    return r
```

Suma llista d'enters. Versió 1

... Si la llista pot ser buida:

```
def suma(llista):
    if len(llista)==0:
        r= 0
    else:
        r= llista[0]+suma(llista[1:])
    return r
```

... Cada crida genera una nova llista. Si una llista té n elements, en la darrera crida recursiva hi hauran n llistes de $n, n-1, n-2, \ldots$ elements respectivament, és a dir $\frac{n(n+1)}{2}$ elements a memòria.

Suma llista d'enters. Versió 2

Estalviant còpies innecessaries de llistes.

. . . Recursivitat lineal no final

```
def suma(llista):
    return sumai(llista,0,len(llista)-1)

def sumai(llista,ini,fi):
    if ini>fi:
        r= 0
    else:
        r = llista[ini]+sumai(llista,ini+1,fi)
    return r
```

... Recursivitat lineal final

```
def suma(llista):
    return sumai(llista,0,0)

def sumai(llista, desde, sumaAc):
    if desde>len(llista)-1:
        r=sumaAc
    else:
        r=sumai(llista,desde+1,sumaAc+llista[desde])
    return r
```

. . . Recursivitat lineal final

Amb llesques

```
def suma(llista):
    return sumai(llista,0)

def sumai(llista,sumaAc):
    if llista==[]:
        s=sumaAc
    else:
        s=sumai(llista[1:], sumaAc+llista[0])
    return s
```

... Recursivitat múltiple no final

```
def suma(llista):
    if len(llista)==0:
        r=0
    elif len(llista)==1:
        r=llista[0]
    else:
        m=len(llista)//2
        r=suma(llista[:m])+suma(llista[m:])
    return r
```

*Dissenyeu una funció recursiva cerca(x, *llista*) que retorni True si x està en llista o retorni False en cas contrari.*

Cerca d'un element en una llista


```
def cerca(x, llista):
    if llista == []:
        r= False
    else:
        if x==llista[0]:
            r= True
        else:
            r= cerca(x, llista[1:])
    return r
```

... evitant còpies de llistes,

```
def cerca(x, llista):
    return cercai(x,llista,0)

def cercai(x ,llista, i):
    if i>len(llista)-1:
        r=False
    else:
        if llista[i]==x:
            r=True
        else:
            r=cercai(x, llista, i+1)
    return r
```

*Dissenyeu una funció recursiva cerca(x, *llista*) que retorni True si x està en llista o retorni False en cas contrari. La llista té els elements ordenats ascendentment.*

Cerca d'un element en una llista ordenada. Versió 1

La solució al problema anterior, serveix però no treu profit de la ordenació:

```
def cerca(x, llista):
    if llista == []:
        r= False
    else:
        if x==llista[0]:
            r= True
        else:
            r= cerca(x, llista[1:])
    return r
```

Cerca d'un element en una llista ordenada. Versió 2

Es pot escriure també com la cerca a dos trossos de la llista: cap de la llista i la resta de la llista (cua de la llista):

```
def cerca(x, llista):
    if len(llista)==0:
        r=False
    elif len(llista)==1:
        r=llista[0]==x
    else:
        r=cerca(x, llista[:1]) or cerca(x, llista[1:])
    return r
```

Cerca d'un element en una llista ordenada. Versió 3

Es pot escriure també com la cerca a dos trossos de la llista: meitat i meitat:

```
def cerca(x, llista):
    if len(llista)==0:
        r=False
    elif len(llista)==1:
        r=llista[0]==x
    else:
        m = len(llista)//2
        r=cerca(x, llista[:m]) or cerca(x, llista[m:])
    return r
```

El cas pitjor (no hi és l'element) cal passar per tots els elements.

Cerca dicotòmica. Versió 4 del problema

La ordenació dels elements imposa que l'element a cercar només es pot trobar a una posició específica de la llista (suposant elements diferents). Cerquem doncs, la posició de l'element.

```
def cercaBinaria(x, llista):
    if llista == []:
        r=False
    else:
        m = len(llista) // 2
        if x < llista[m]:
            r=cercaBinaria(x, llista[:m])
        elif x > llista[m]:
            r=cercaBinaria(x, llista[m+1:])
        else: # x == llista[m]
            r=True
    return r
```

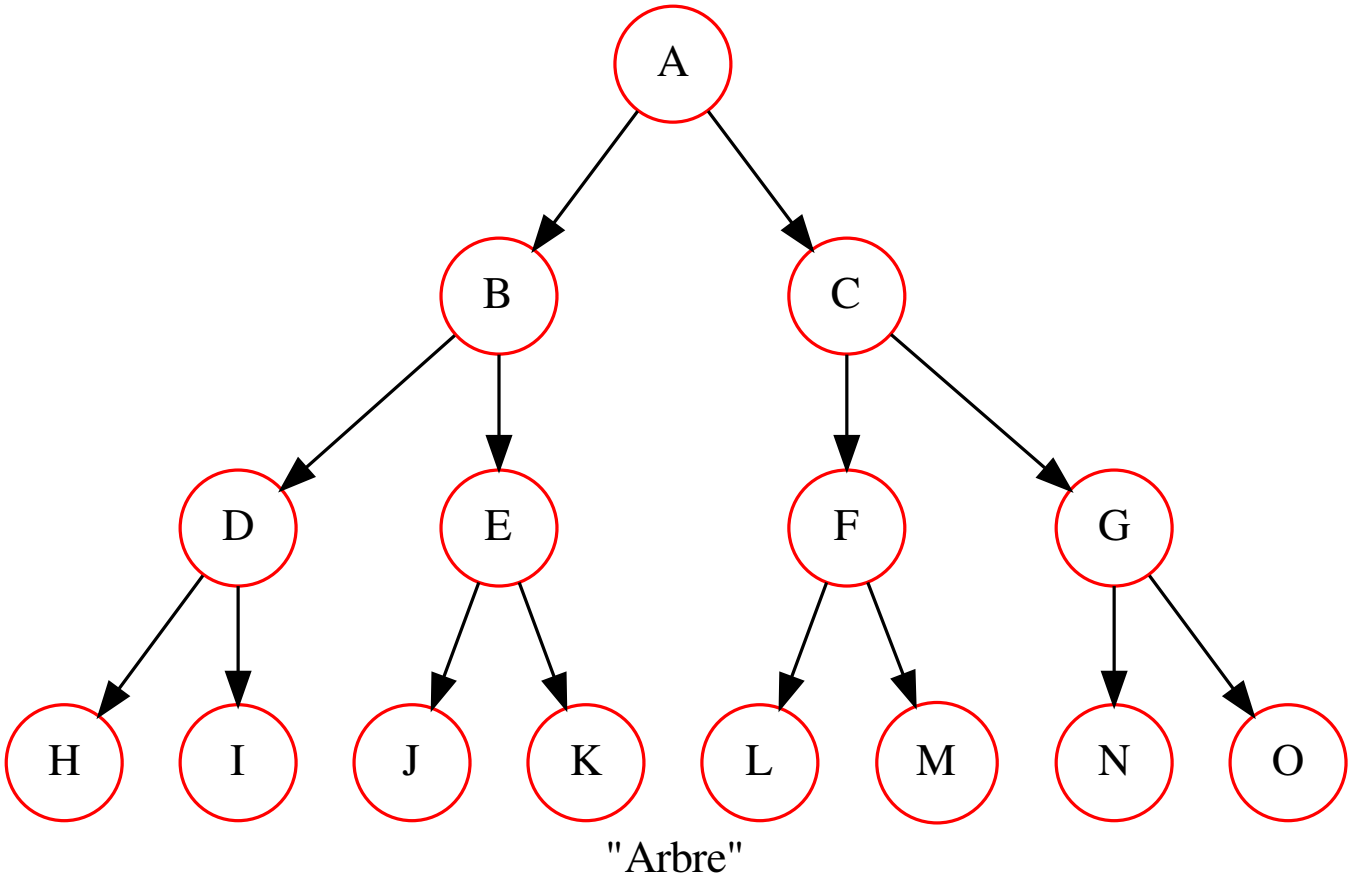
Cerca dicotòmica. Versió 5


```
def cercaBinaria(x, llista):
    return cercaB(x,llista,0,len(llista)-1)

def cercaB(x ,llista, ini, fi):
    if ini>fi:
        return False
    else:
        m = (ini + fi) // 2
        if x < llista[m]:
            r=cercaB(x, llista, ini, m-1)
        elif x > llista[m]:
            r=cercaB(x, llista, m+1, fi)
        else: # x == llista[m]
            r=True
    return r
```

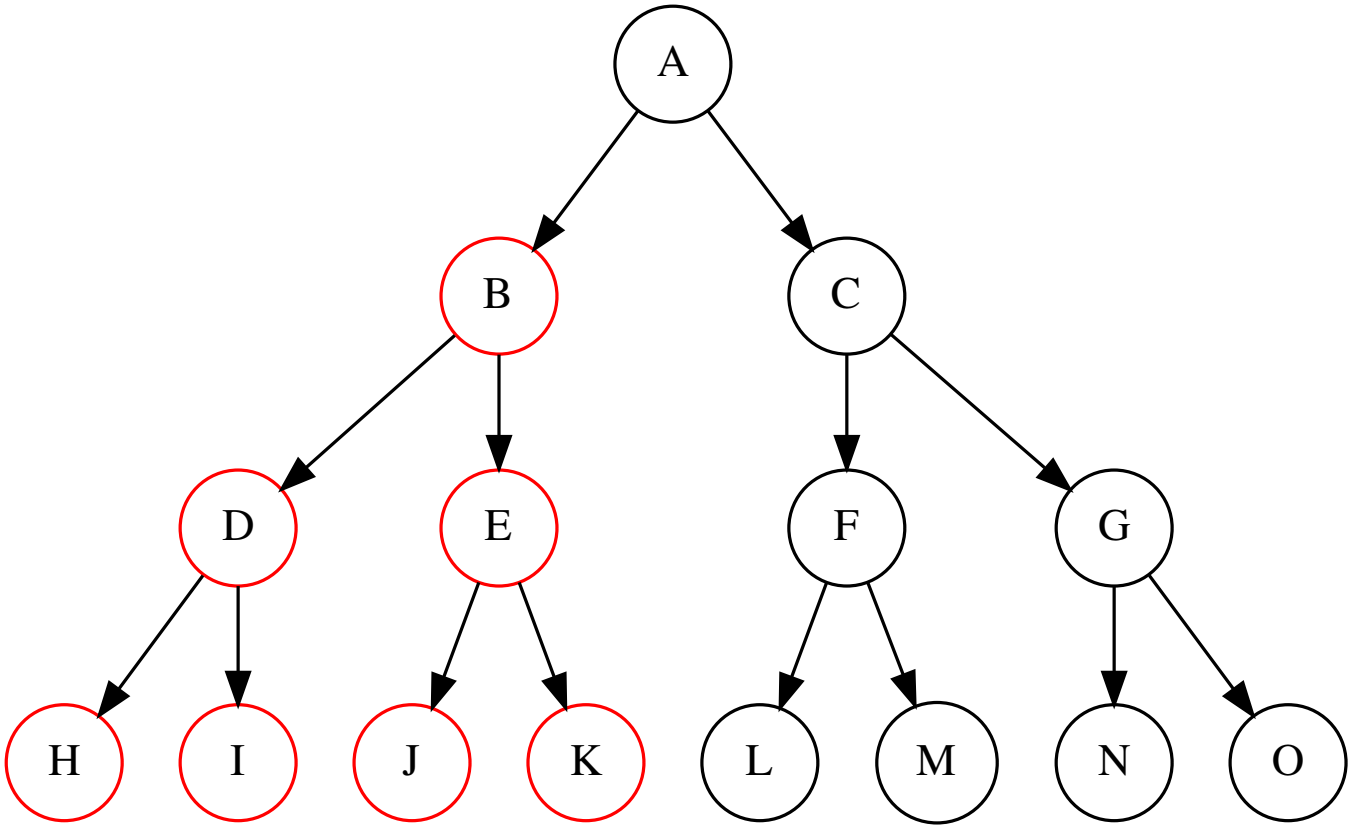
Cerca dicotòmica. Versió 6

Una altra versió que assegura que *ini* acaba apuntant a la posició on li toca estar l'element a cercar.

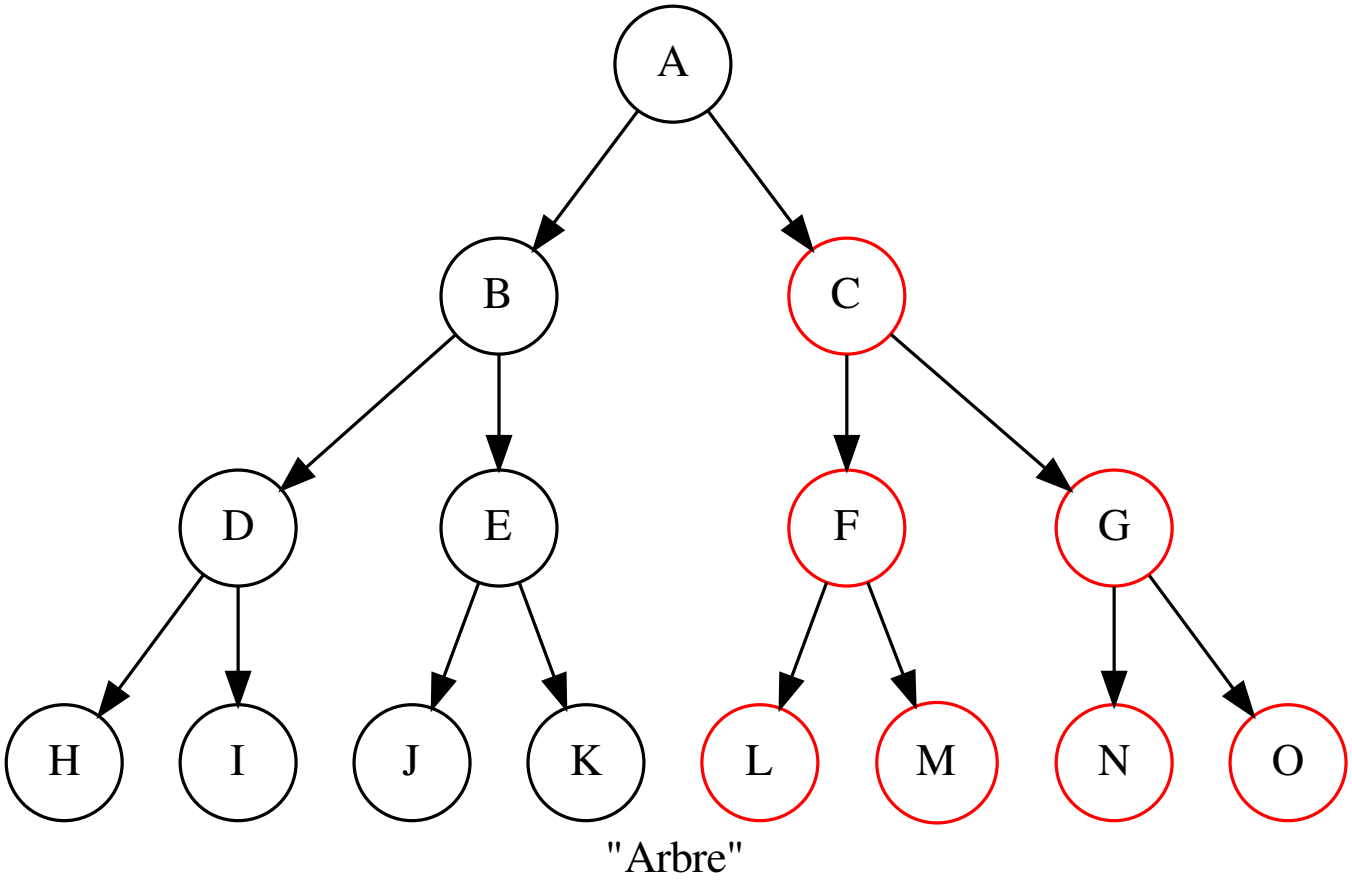
```
def cercaBinaria(x, llista):
    if llista!=[]:
        r=cercaB(x,llista,-1,len(llista))
    else:
        r=False
    return r

def cercaB(x ,llista, ini, fi):
    if ini+1==fi:
        r=llista[ini]==x
    else:
        m = (ini + fi) // 2
        if x < llista[m]:
            r=cercaB(x, llista, ini, m)
        else: # x >= llista[m]:
            r=cercaB(x, llista, m, fi)
    return r
```



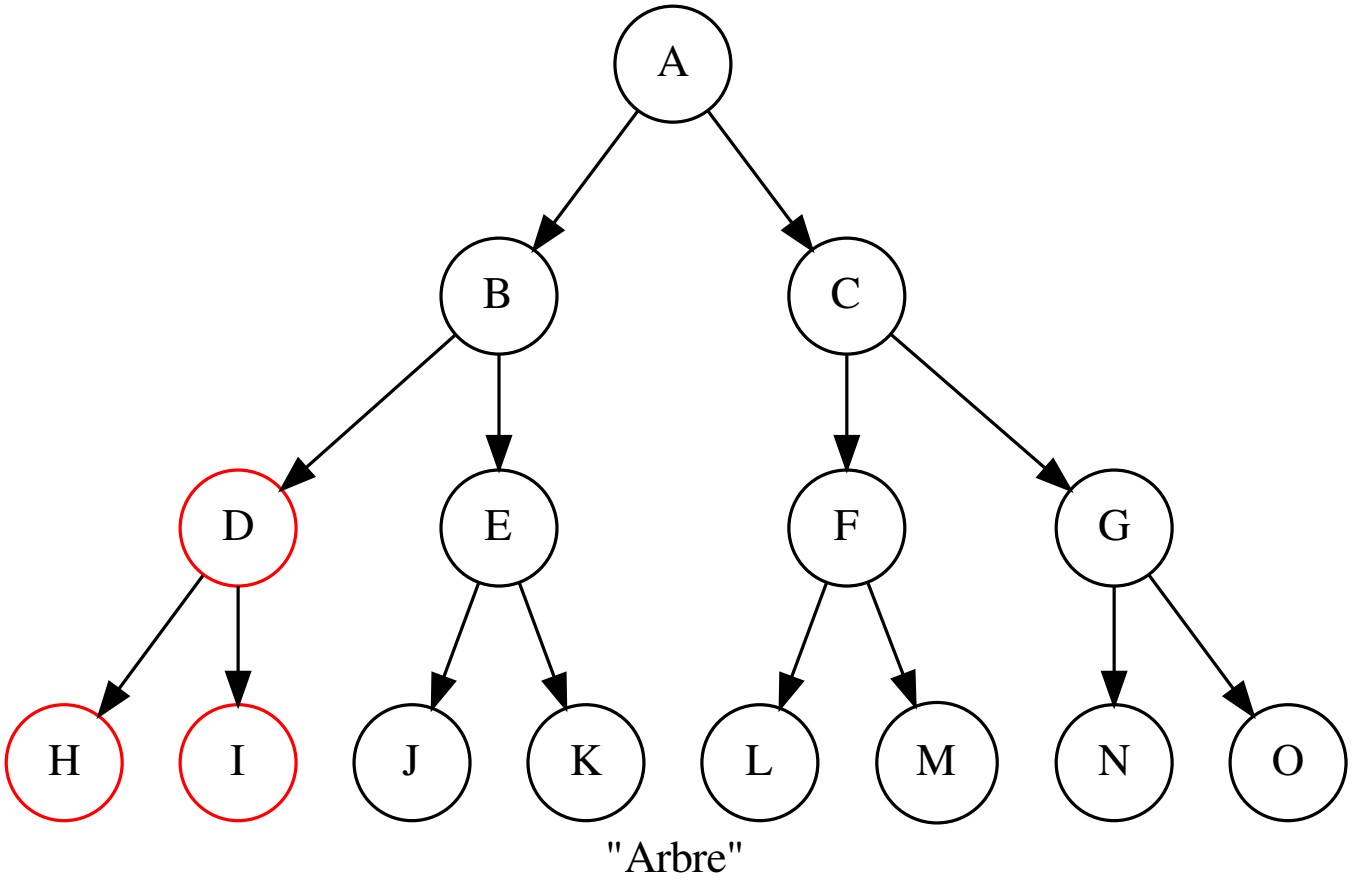
"Arbre"



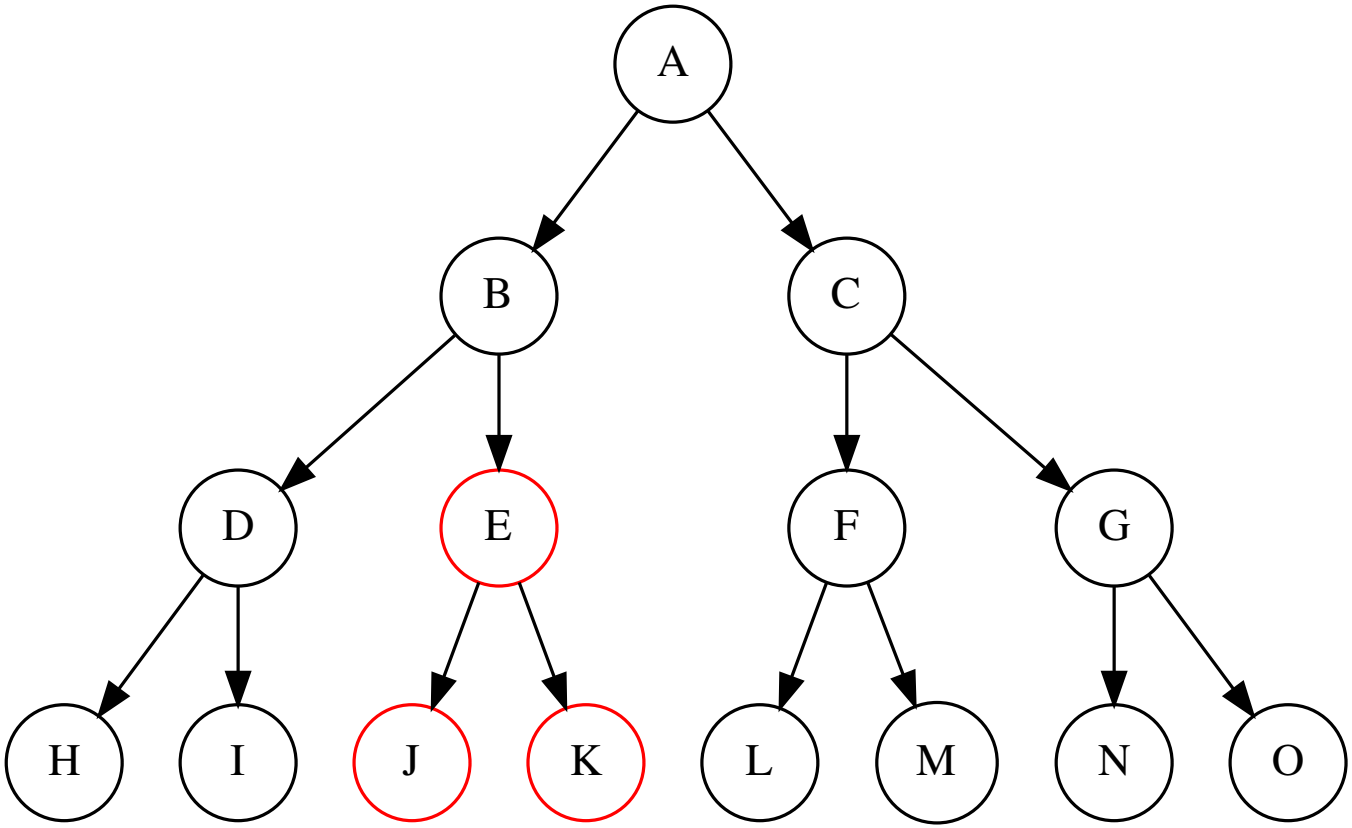
"Arbre"



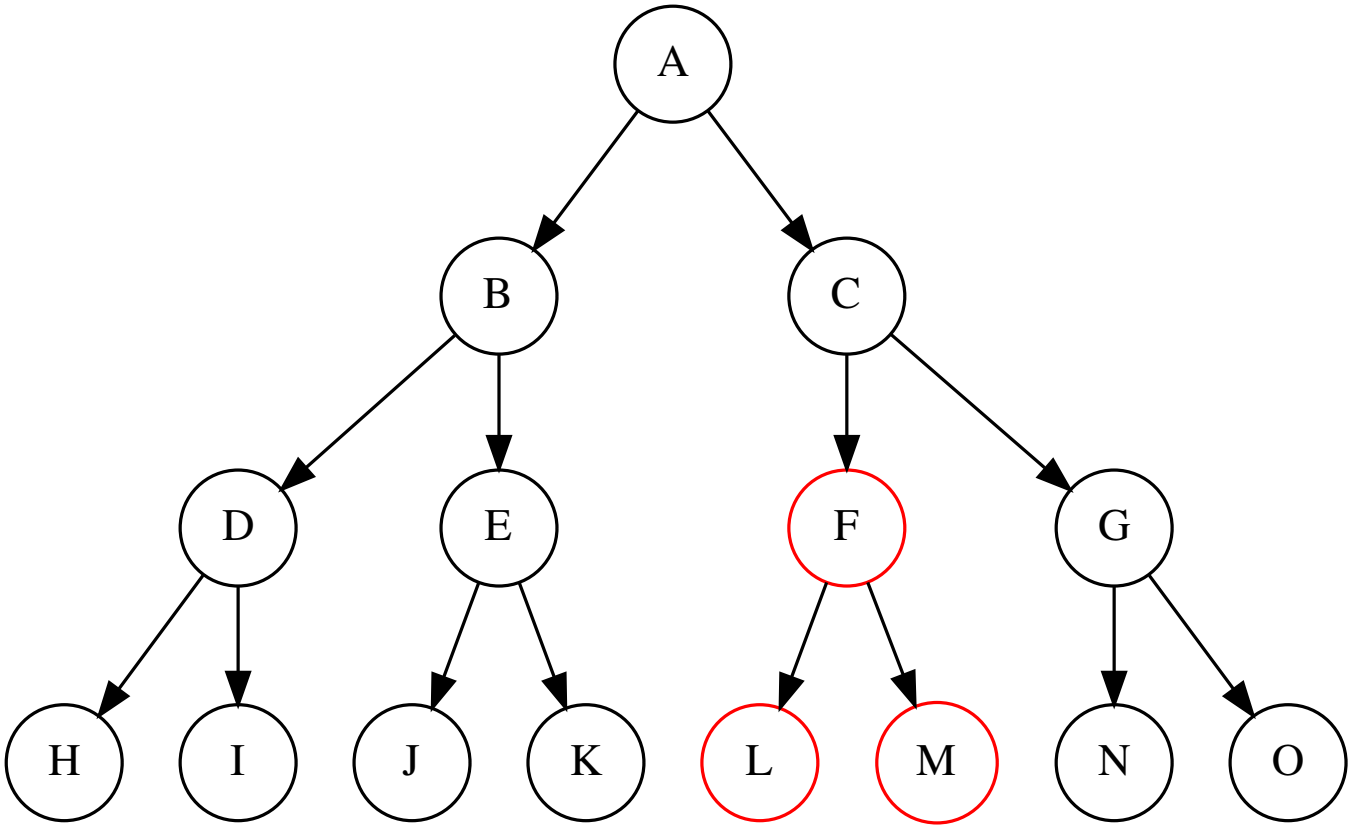
"Arbre"



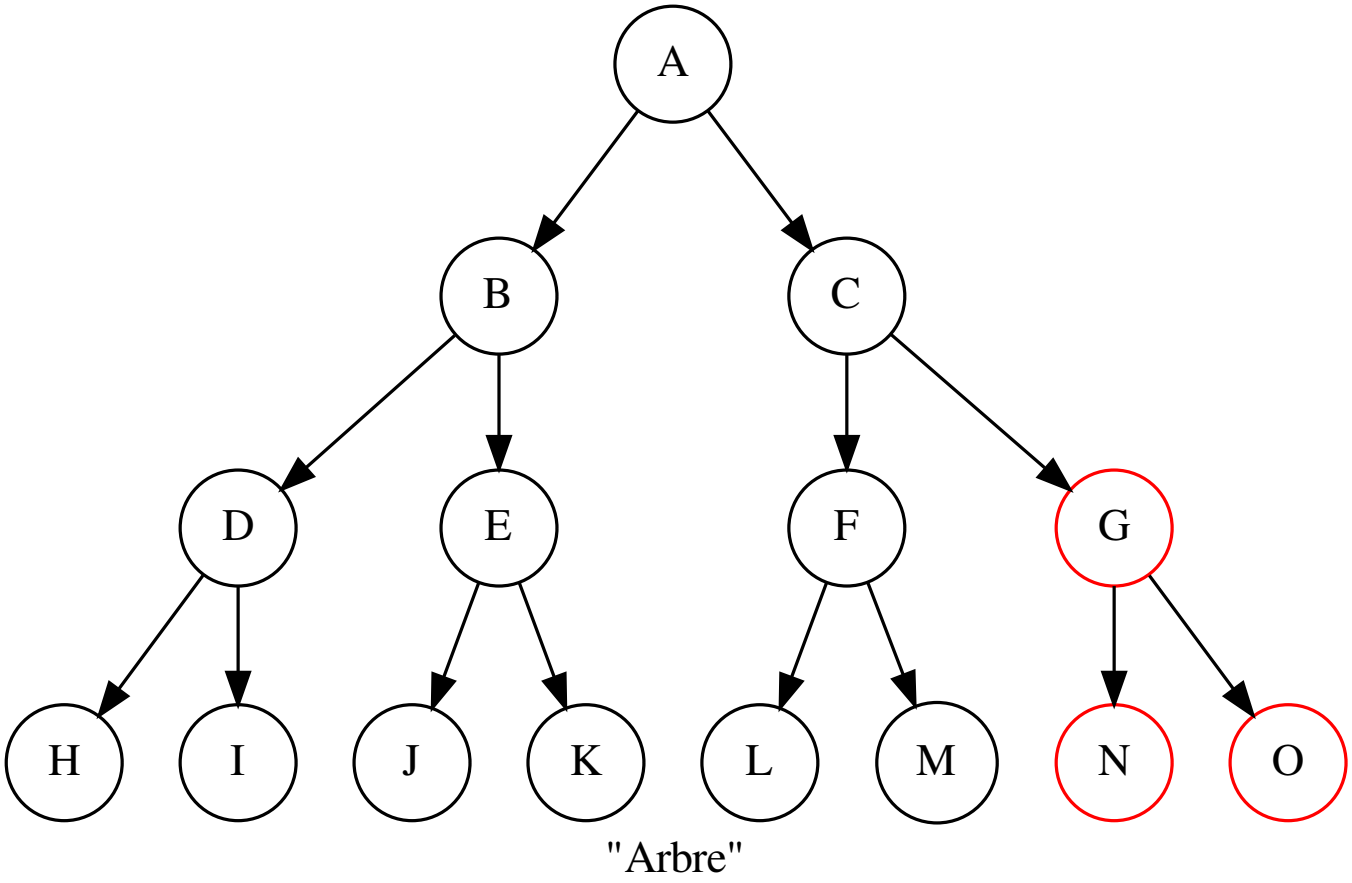
"Arbre"



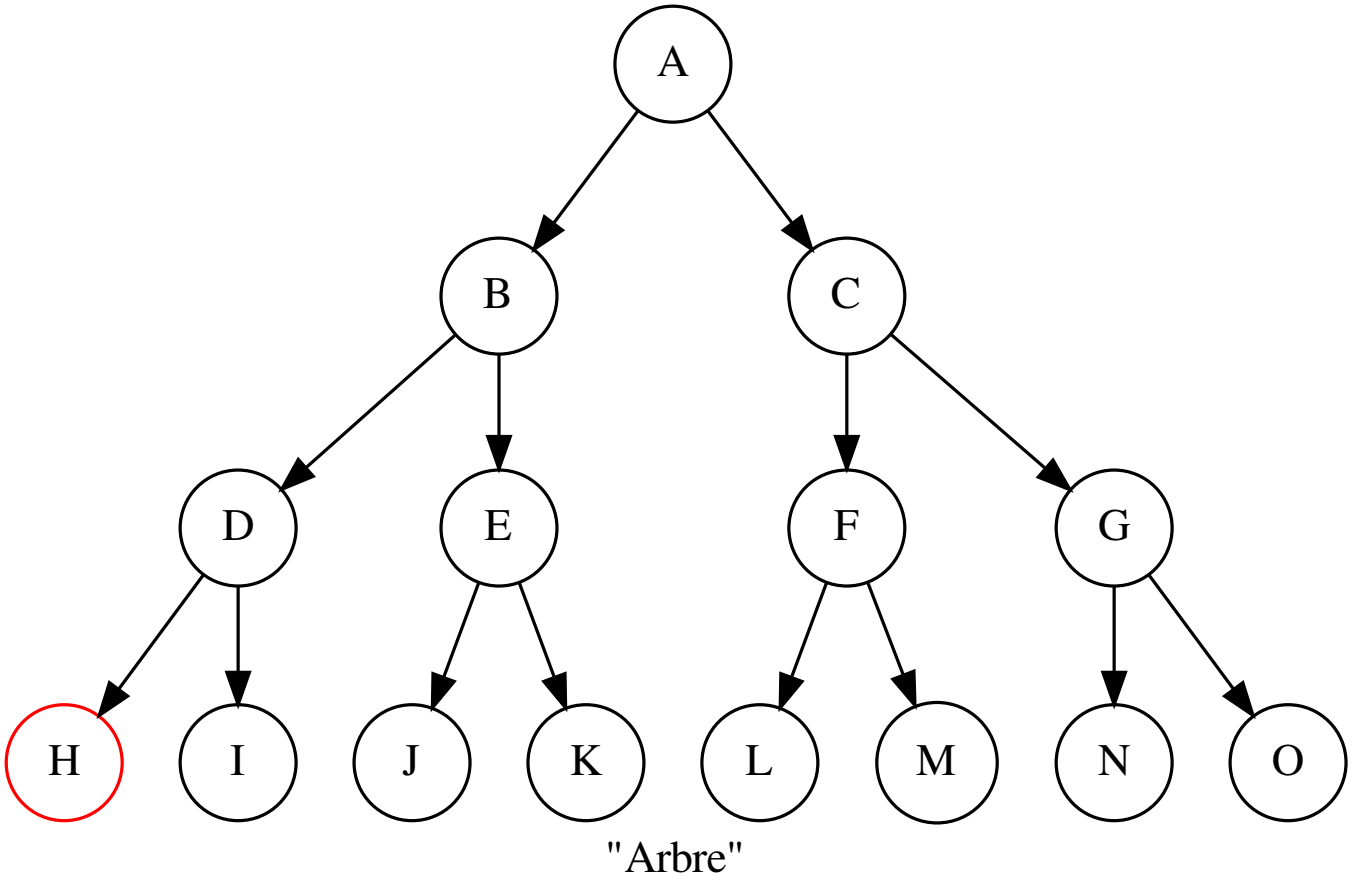
"Arbre"



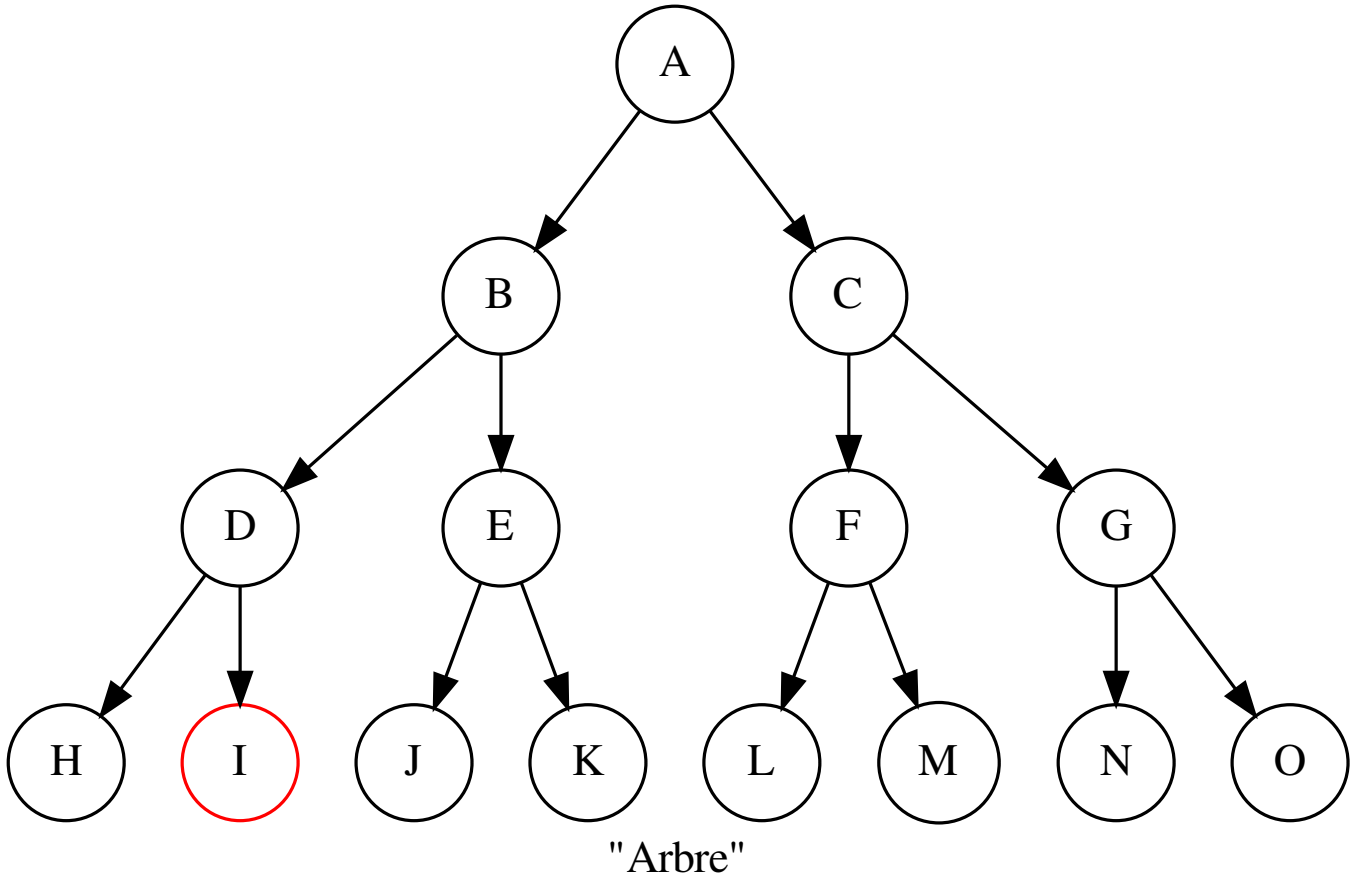
"Arbre"



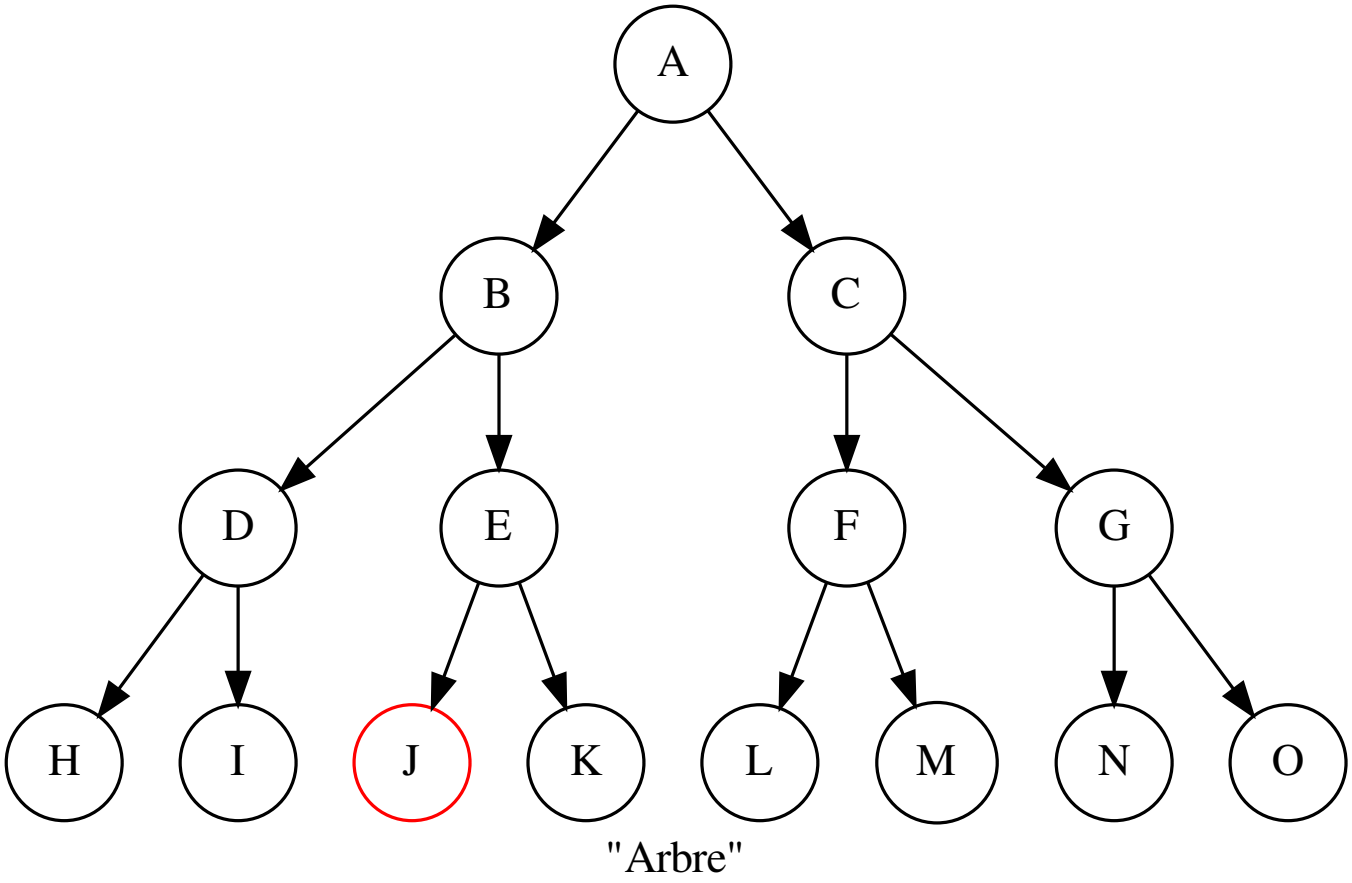
"Arbre"



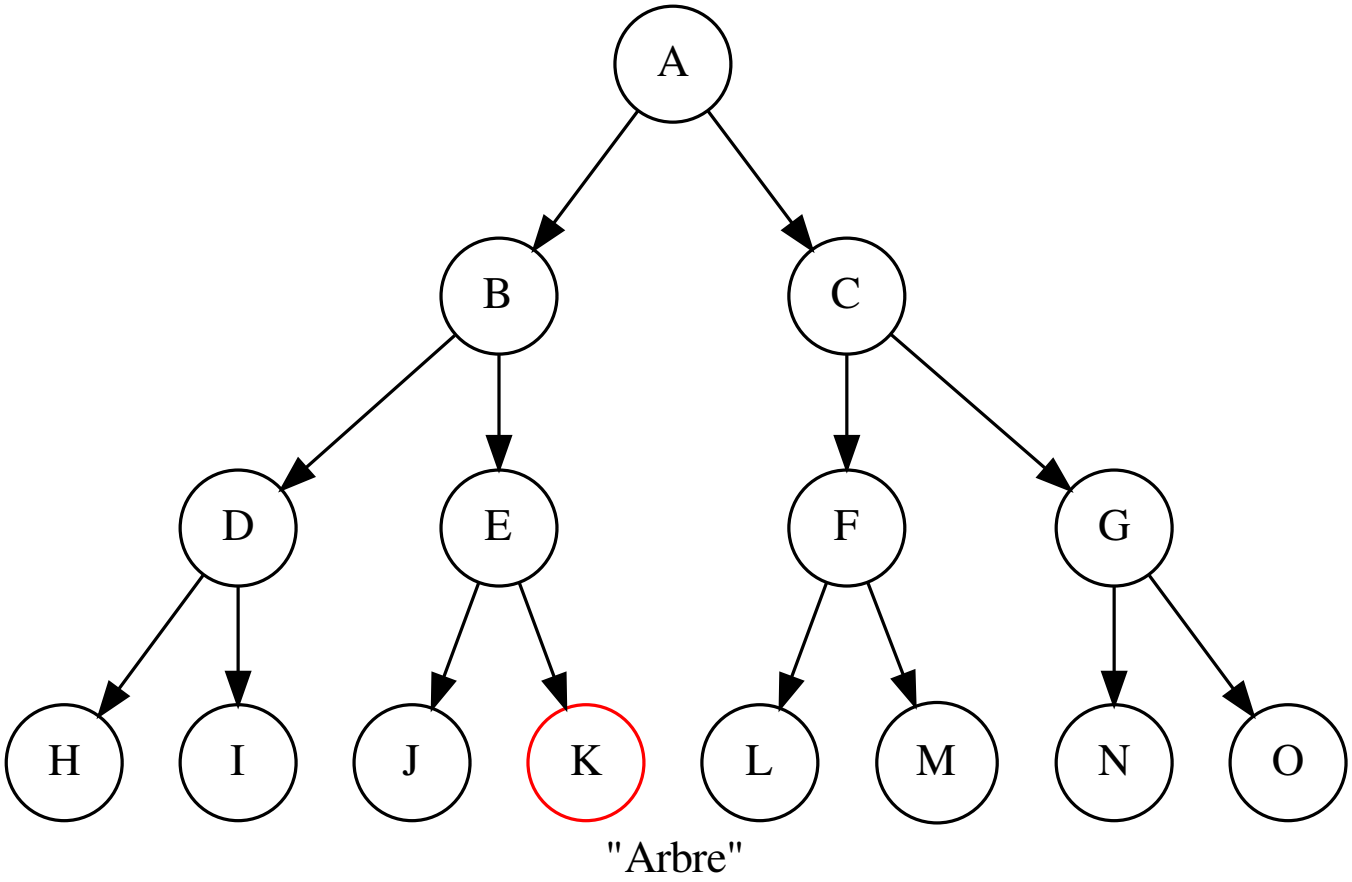
"Arbre"



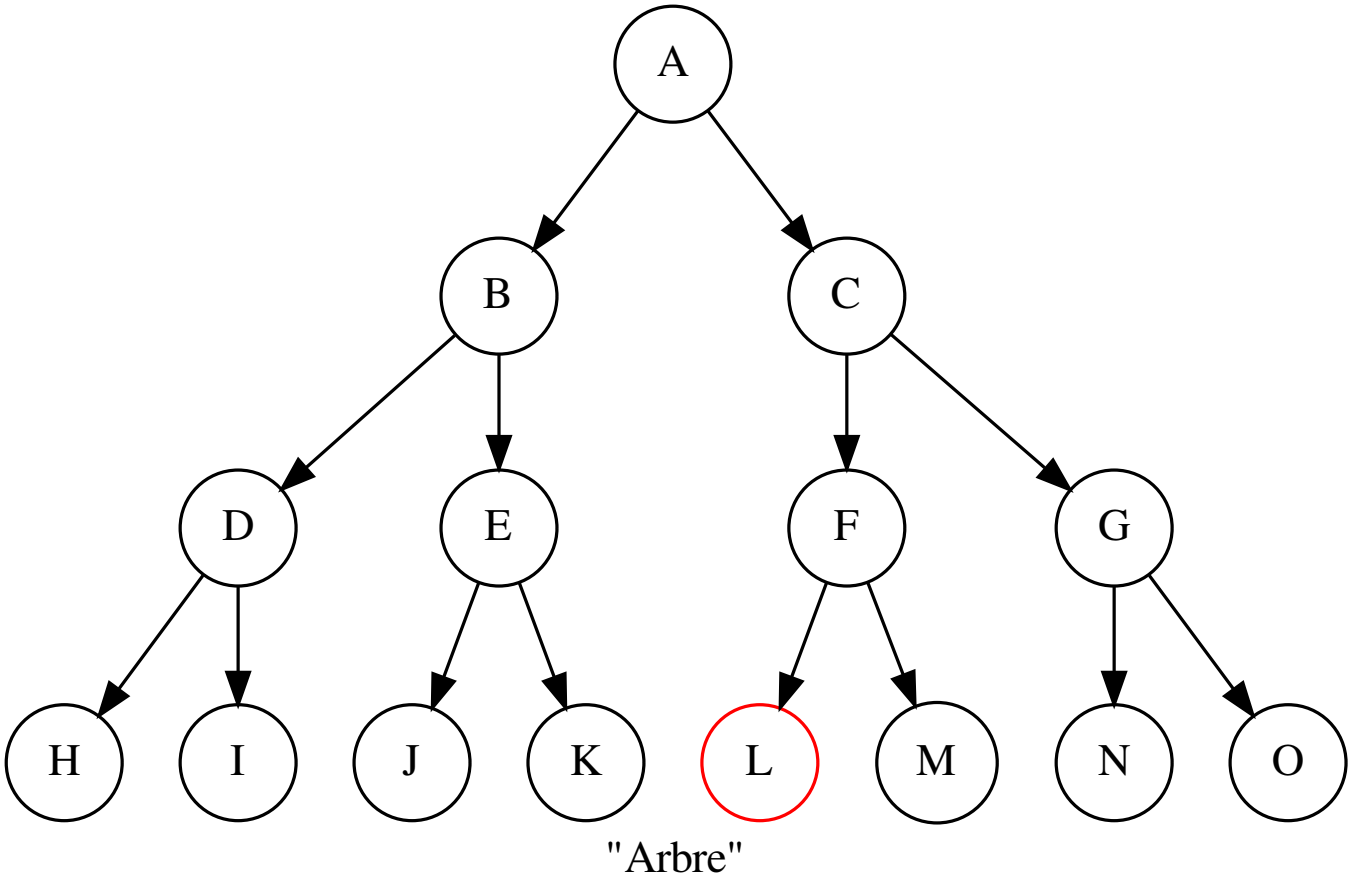
"Arbre"



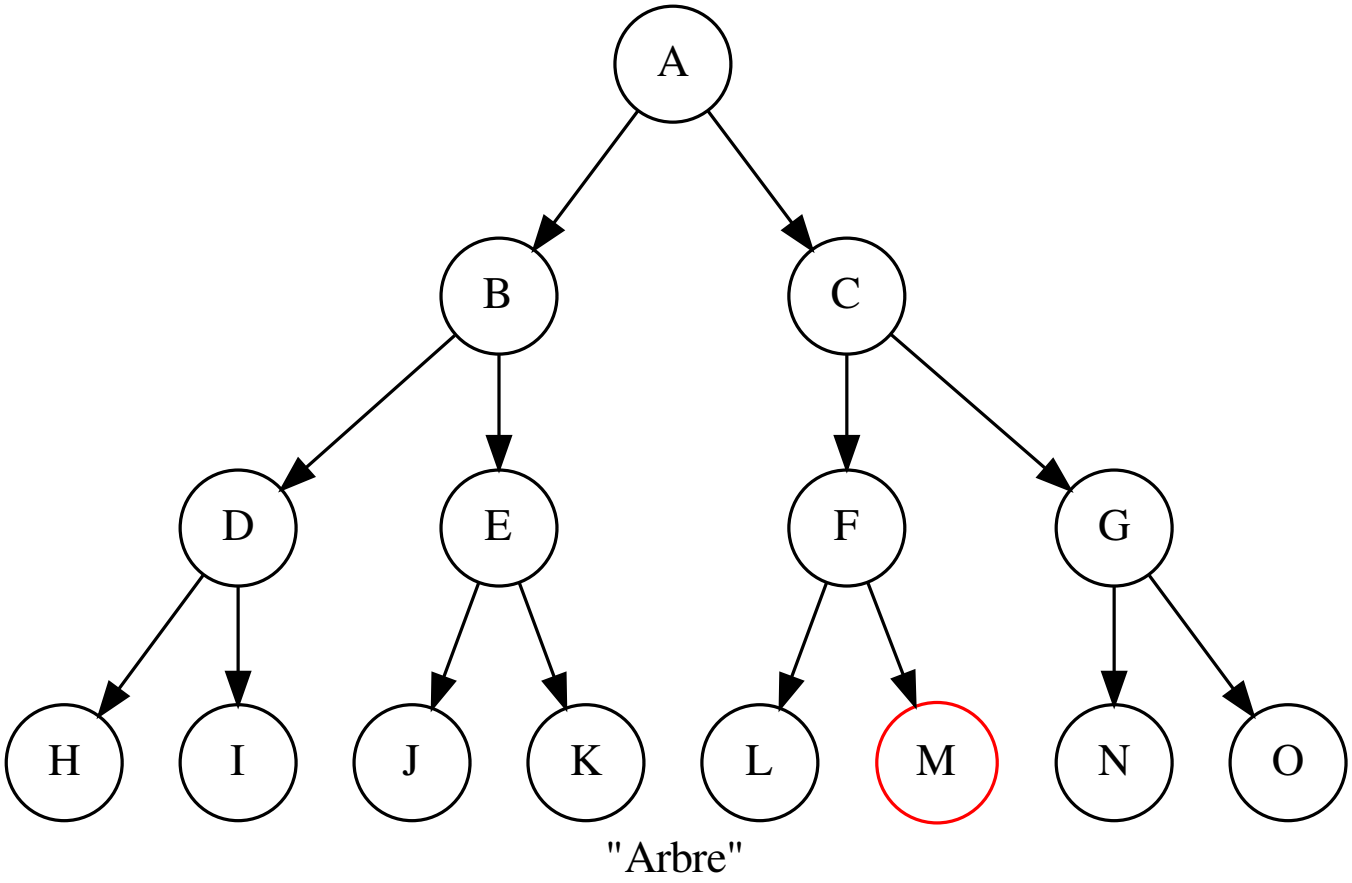
"Arbre"



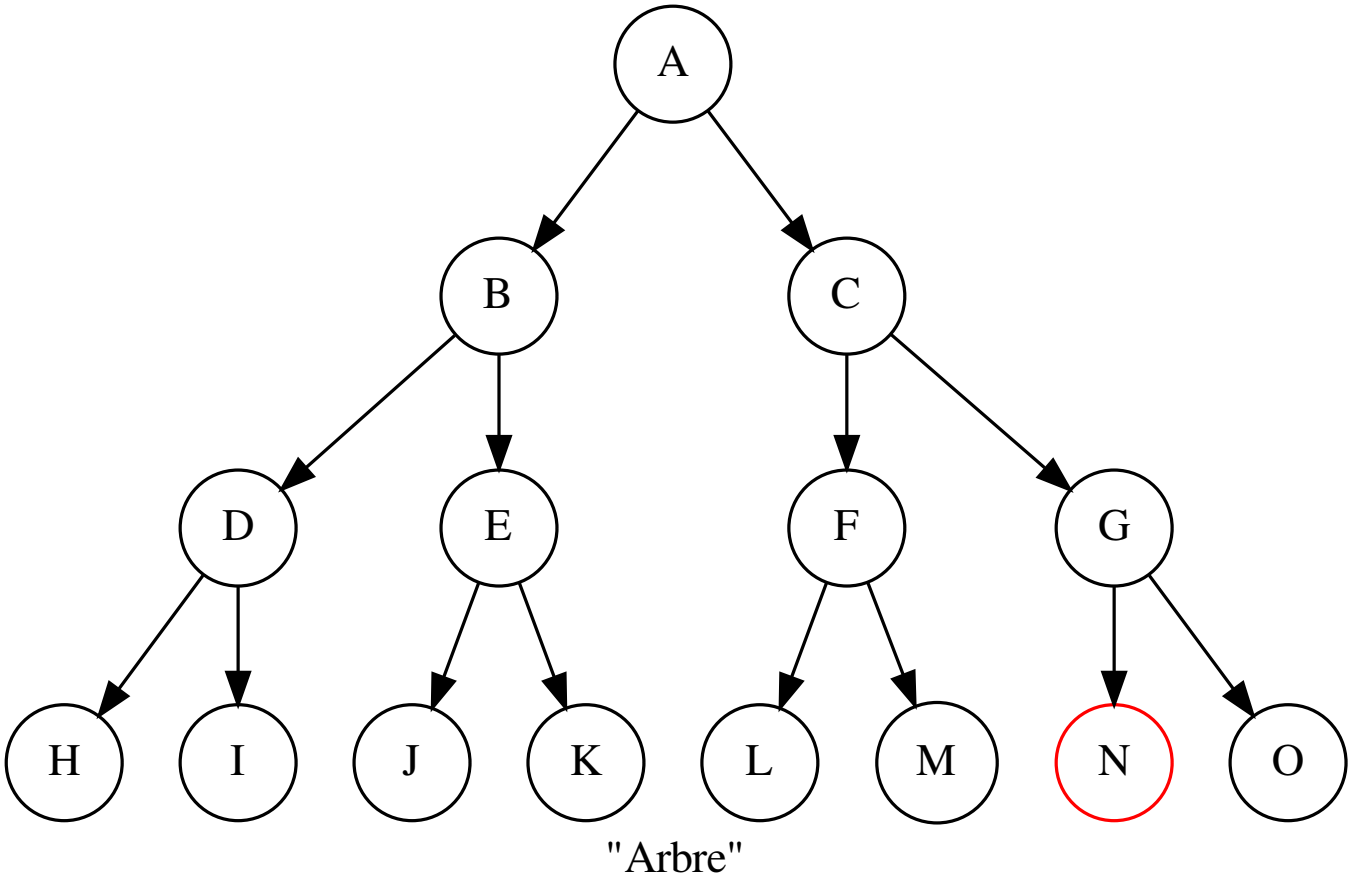
"Arbre"



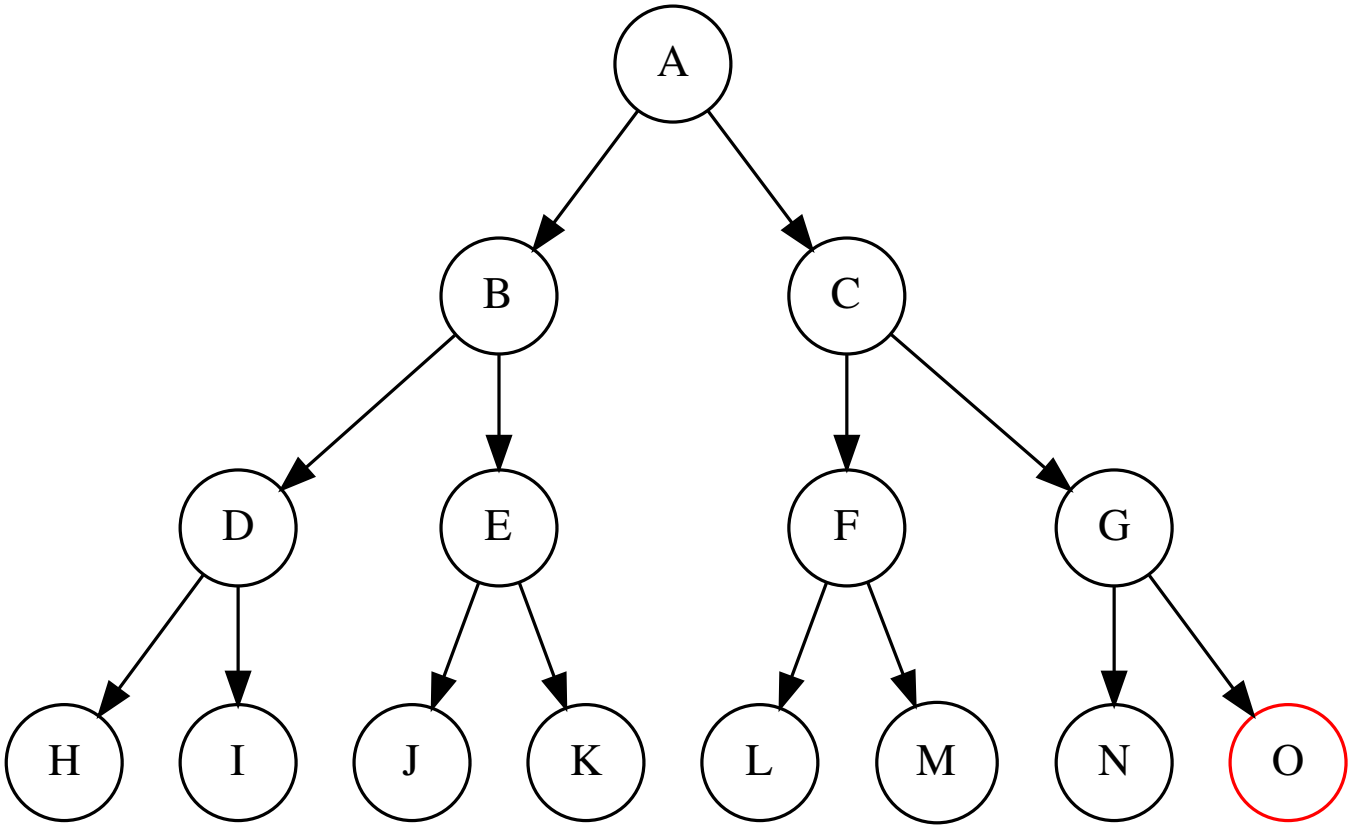
"Arbre"



"Arbre"

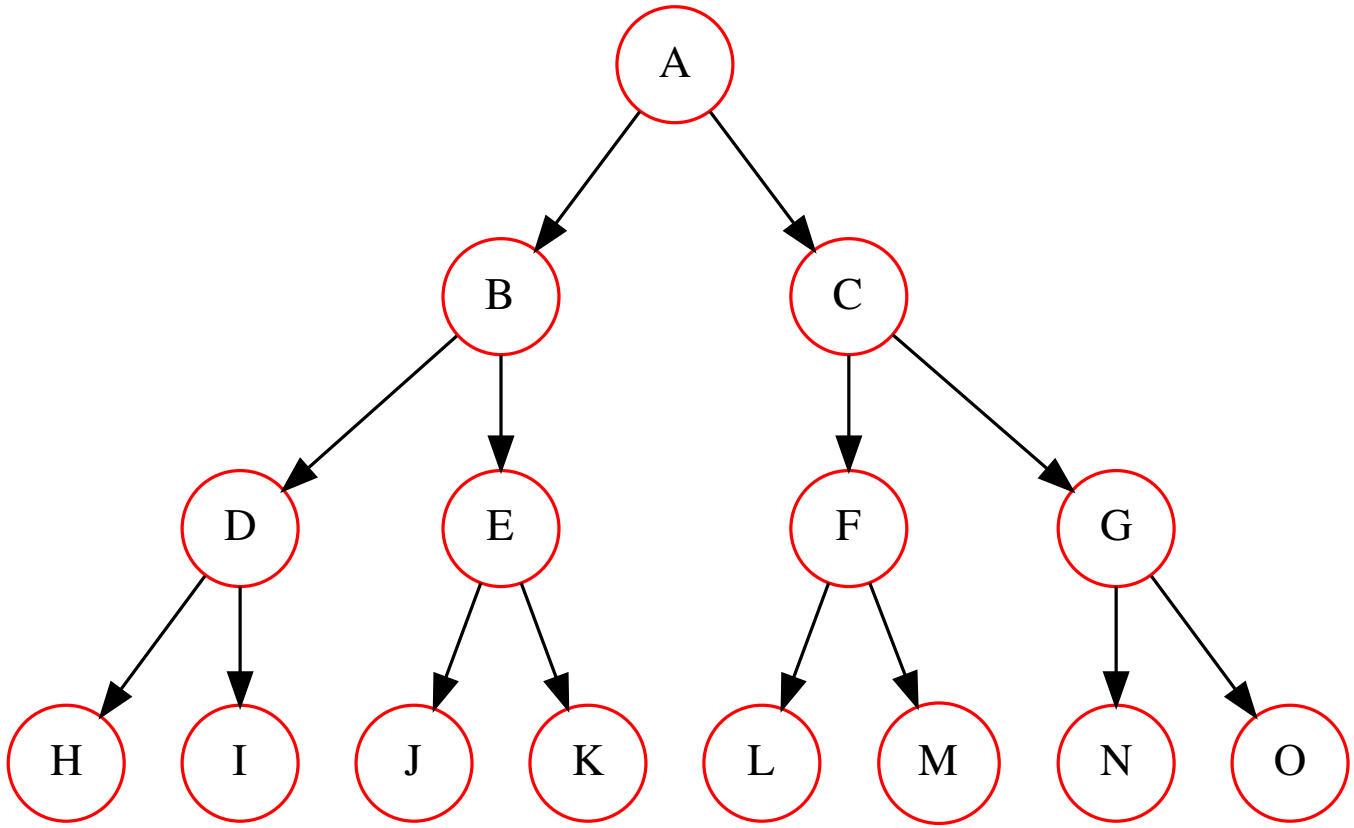


"Arbre"



"Arbre"

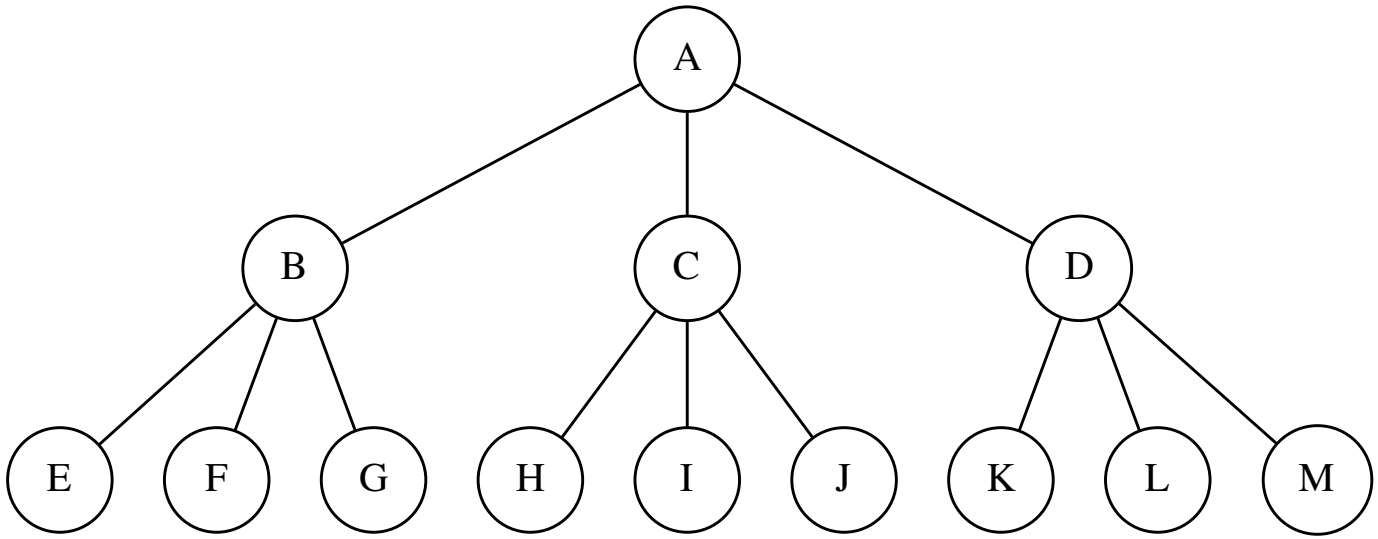
Nombre màxim de nodes arbre



"Arbre"

nombre màxim de nodes *maxNodes* per alçada h: $maxNodes = \frac{(N^{h+1}-1)}{(N-1)}$,
on N és el grau màxim del node.
$N = 2 \rightarrow maxNodes = 2^{h+1} - 1$

Arbre 3-ari

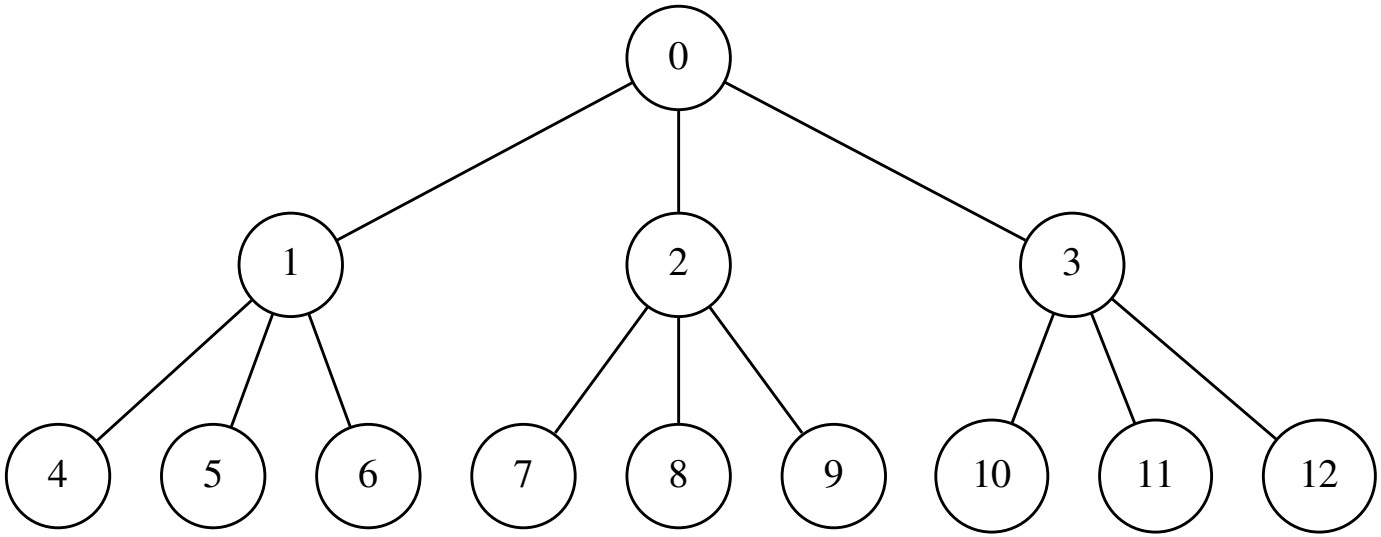


Arbre 3-ari

nombre màxim de nodes *maxNodes* per alçada h: $maxNodes = \frac{(N^{h+1}-1)}{(N-1)}$, on N és el grau màxim del node.

$N = 3 \rightarrow maxNodes = \frac{3^{h+1}-1}{2}$

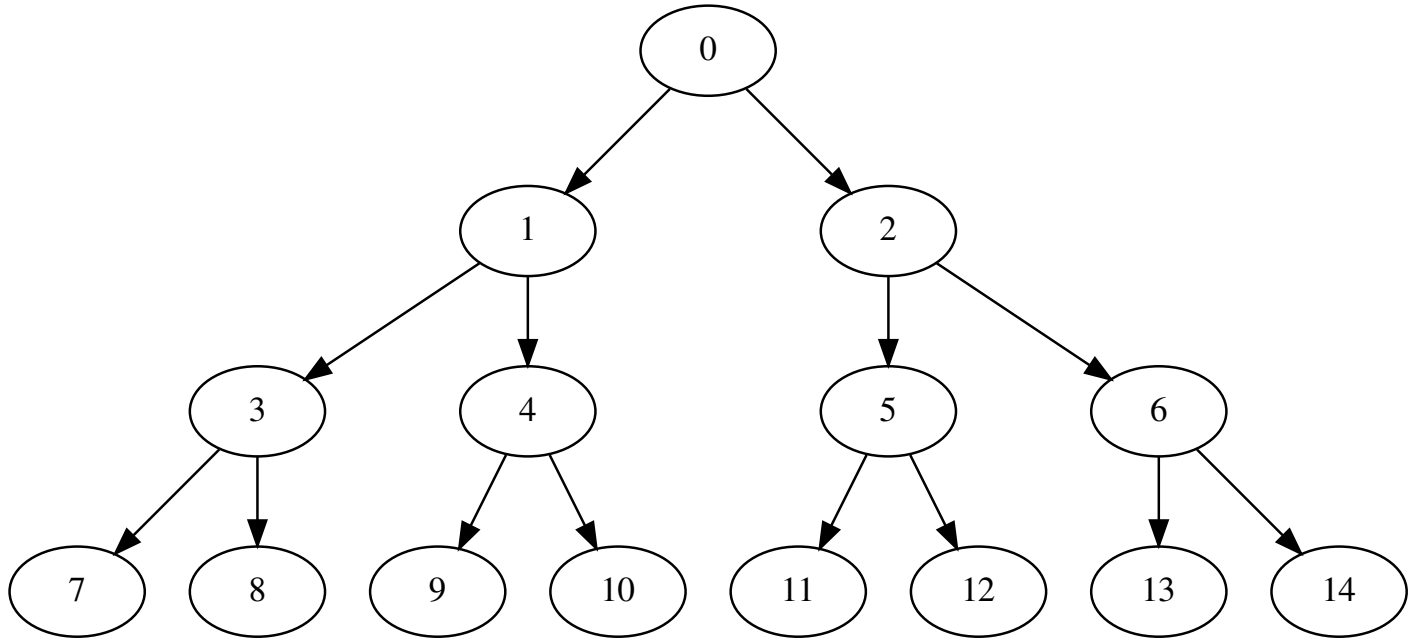
Llista d'arbre 3-ari



Arbre 3-ari

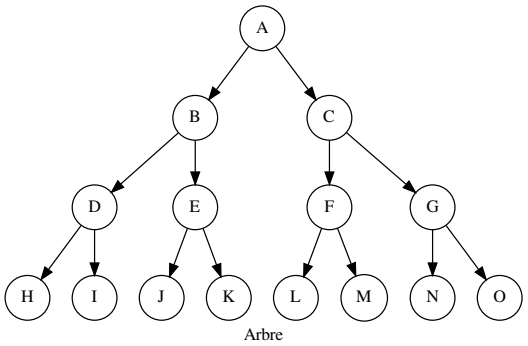
Node en la posició *pos* pot tenir fills a les posicions $3 \times pos + 1$, $3 \times pos + 3$, i $3 \times pos + 3$,

Llista d'arbre binari



Node en la posició *pos* pot tenir fills a les posicions $2 \times pos + 1$, i $2 \times pos + 2$.

Alçada arbre binari



Arbre

Dissenyar la funció alçada(arrel, gd) que donat un arbre binari com una parella de node arrel arrel i gd (Digraph de networkx de l'arbre, calcula l'alçada de l'arbre binari.

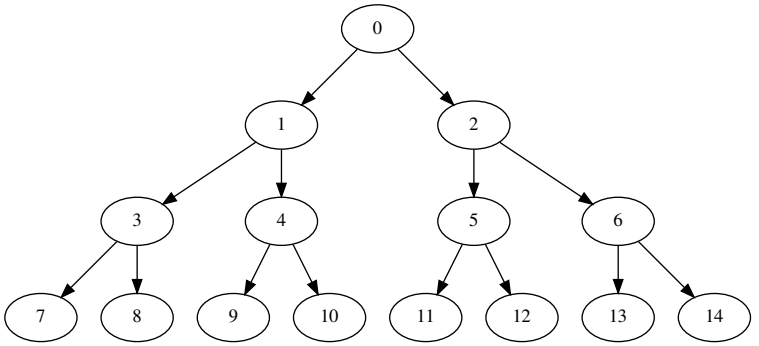
Alçada arbre binari


```
def alçada(node, arbre):
    fills = list(arbre[node])
    if len(fills) == 0:
        r = 0
    if len(fills) == 1:
        r = 1 + alçada(fills[0], arbre)
    elif len(fills) == 2:
        r = 1 + max(alçada(fills[0], arbre),
                    alçada(fills[1], arbre))
    return r
```

Alçada arbre binari


```
>>> from alcada import alçada
>>> import networkx as nx

>>> g = nx.DiGraph()
>>> g.add_edges_from([('A', 'B'), ('A', 'C'),
... ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G'),
... ('D', 'H'), ('D', 'I'), ('E', 'J'), ('E', 'K'),
... ('F', 'L'), ('F', 'M'), ('G', 'N'), ('G', 'O')])
>>> alçada('A', g)
3
```

Alçada arbre binari



Dissenyar la funció alçada(pos, arbre) que donat un arbre binari arbre representat com una llista de nodes, i la posició pos on està l'arrel de l'arbre en la llista, calcula l'alçada de l'arbre binari.

Alçada arbre binari


```
def alçada(pos, arbre):
    if pos >= len(arbre):
        r = -1
    elif arbre[pos] == None:
        r = -1
    else:
        r1 = alçada(2 * pos + 1, arbre)
        r2 = alçada(2 * pos + 2, arbre)
        r = 1 + max(r1, r2)
    return r
```

Alçada arbre binari


```
>>> from alcada2 import alçada

>>> a = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
... 'J', 'K', 'L', 'M', 'N', 'O']
>>> alçada(0, a)
3

>>> a = ['A']
>>> alçada(0, a)
0
```