ÁRBOLES

Joaquín Fernández-Valdivia
Javier Abad
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

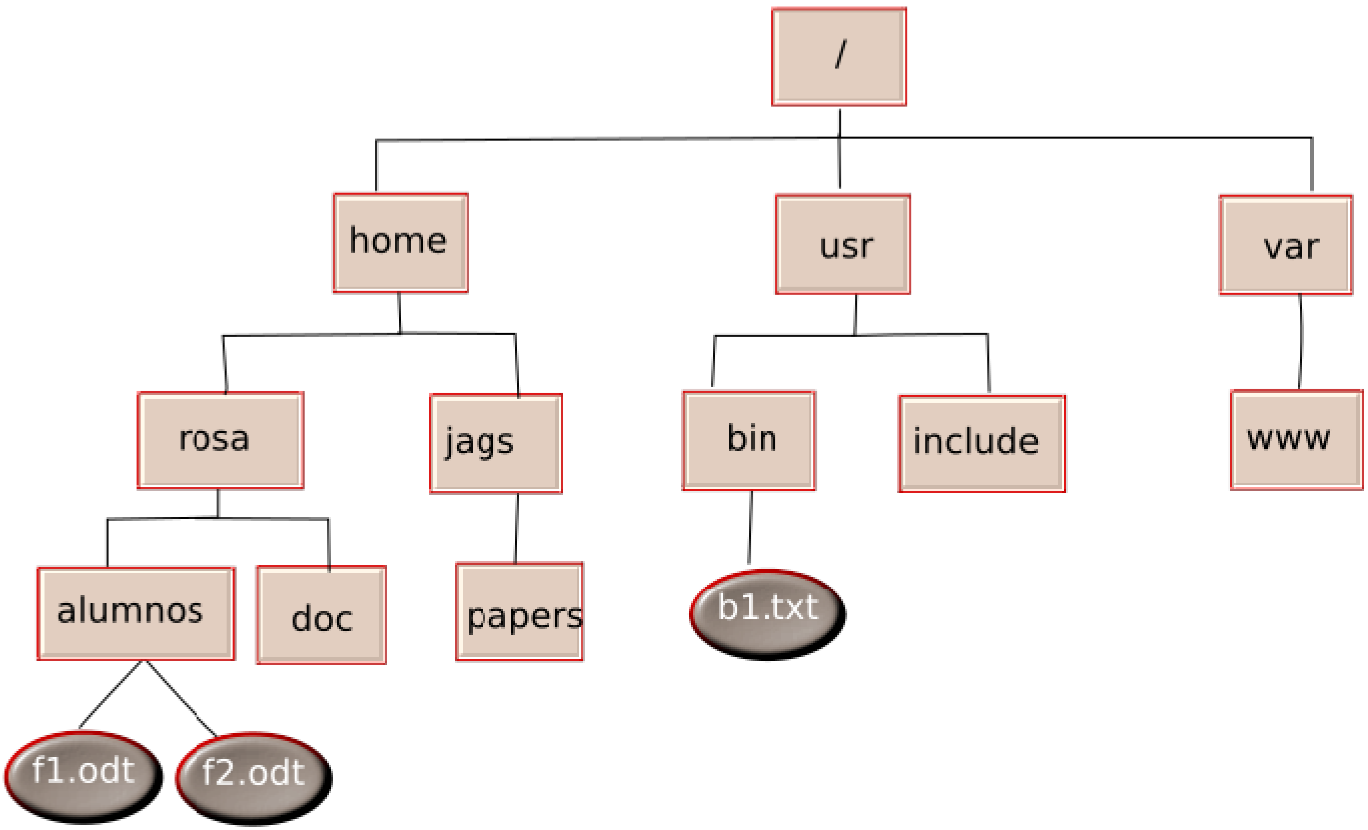
DECSAI

Árboles

- Estructura de datos lineal → existe una relación de anterior y siguiente entre los elementos que la componen

- Estructura de datos no lineal → grafos y árboles

Árboles


/
home
usr
var
rosa
jags
bin
include
www
alumnos
doc
papers
b1.txt
f1.odt
f2.odt

Árboles

- **Teoría de grafos**: un árbol es un grafo acíclico donde cada nodo tiene grado de entrada 1 (excepto el nodo raíz que tiene grado de entrada 0) y el grado de salida 0 o mayor que cero

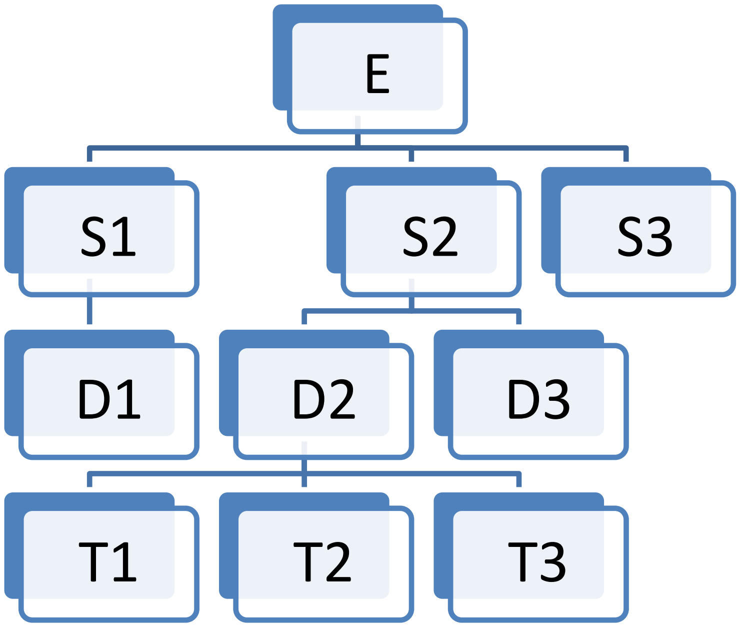
- Un árbol se compone de *nodos*
  - **Raíz**: no tiene padre, está en la parte superior de la jerarquía
  - **Hoja**: no tienen hijos, están en la parte inferior de la jerarquía
  - **Interior**: resto de nodos

Árboles

- Algunas características de los árboles son:

1. Todos los nodos descienden de la raíz
2. Los descendientes directos se llaman hijos
3. Los nodos del mismo nivel y que descienden del mismo padre son hermanos
4. Y los padres de los padres de un nodo son los ancestros de éste

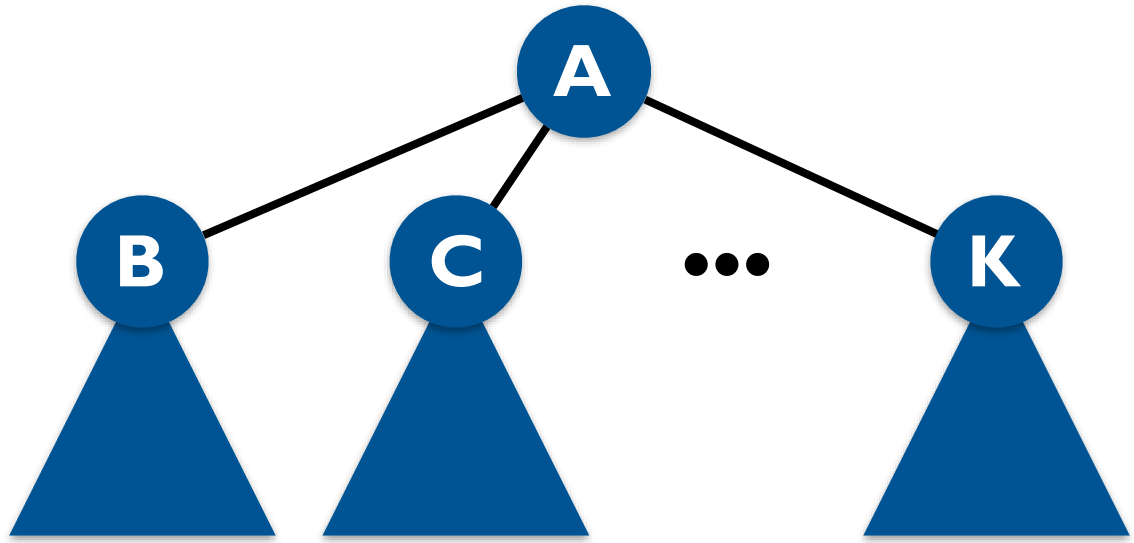
Árboles



- *E* es la raíz del árbol
- *S1*, *S2*, *S3* son los hijos de *E*
- *S1*, *D1* componen un subárbol de la raíz
- D1, T1, T2, T3, D3, S3 son las hojas del árbol
- etc…

Árbol Etiquetado

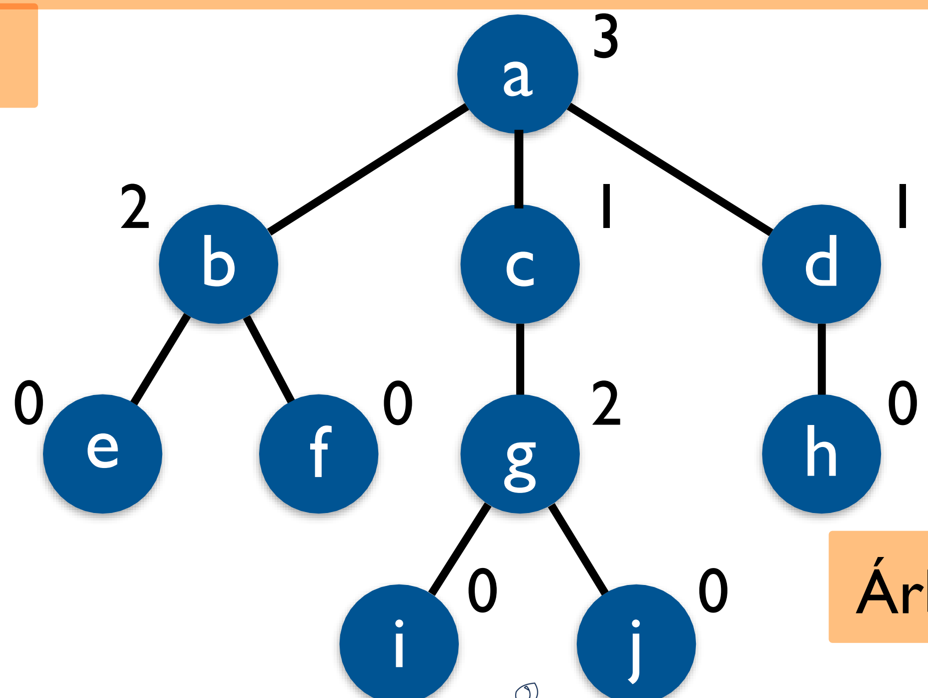
- Se dice que **un árbol está etiquetado** si todos sus nodos contienen una etiqueta



- A los nodos que son hijos de un mismo padre se les denomina **hermanos**

Grado de un Árbol

- Se llama **grado de un nodo** al número de subárboles (de hijos) que tiene dicho nodo. Los nodos de grado 0 se denominan **hojas** o **nodos terminales**. El resto se llaman nodos **no terminales** o **interiores**
- El **grado de un árbol** es el máximo de los grados de sus nodos



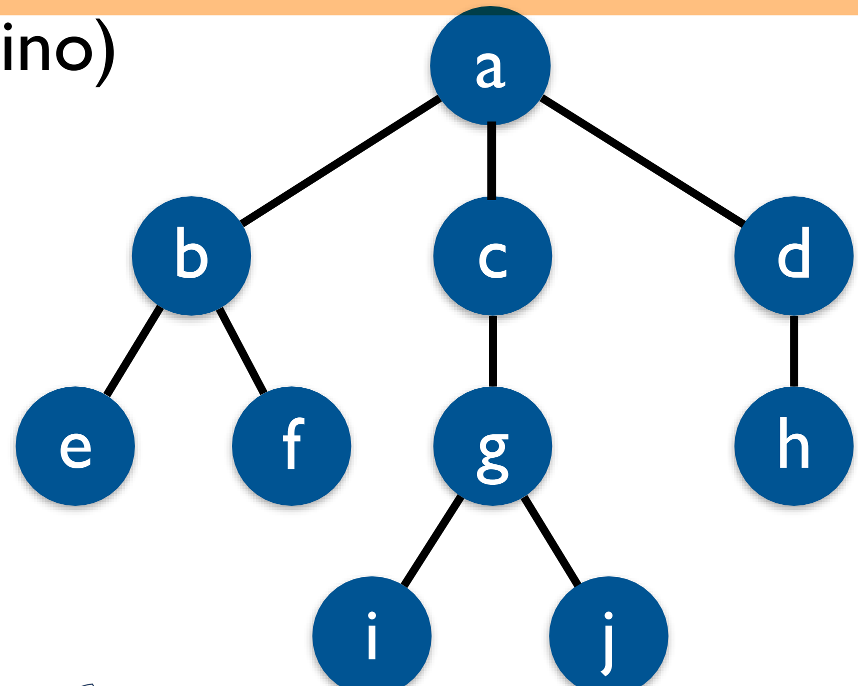
Árbol de grado 3

Caminos

- El **camino entre dos nodos**, n_i y n_j se define como la secuencia de nodos del árbol necesaria para alcanzar el nodo n_j desde el nodo n_i
- La **longitud del camino entre dos nodos** es igual al número de nodos que forman el camino menos 1 (número de ejes del camino)

camino(a,f) = {a,b,f}

Longitud 2

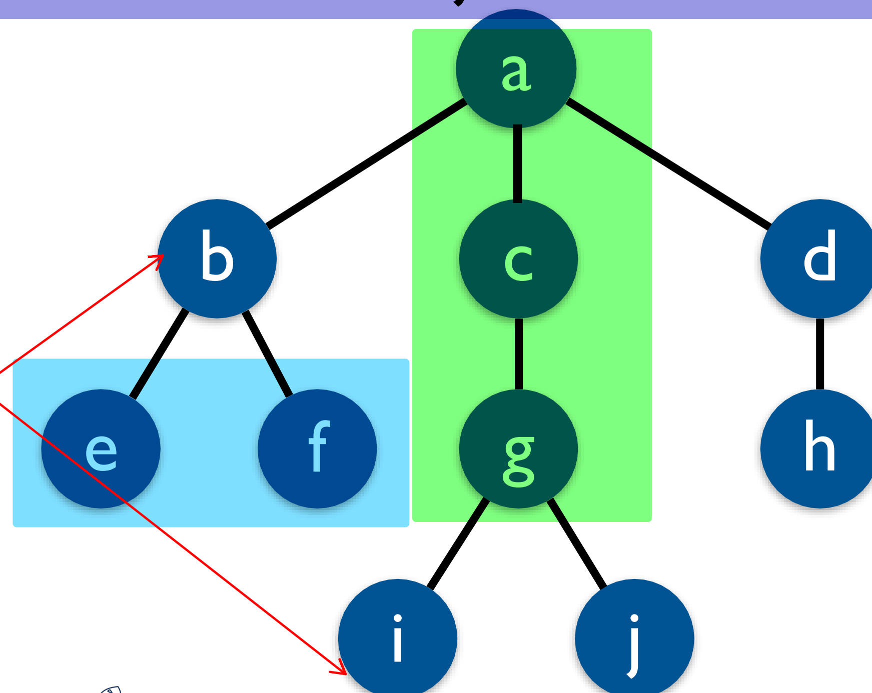


Ancestros y Descendientes

- Si existe un camino del nodo *a* al nodo *b*, entonces *a* es un **ancestro** de *b* y *b* es un **descendiente** de *a*
- La raíz es el único nodo que no tiene ancestros
- Una hoja es un nodo sin descendientes
- Un subárbol de un árbol es un nodo, junto con todos sus descendientes

Ancestros(i): {g, c, a}

Descendientes(b):{e, f}



Altura y Profundidad

Altura de un nodo

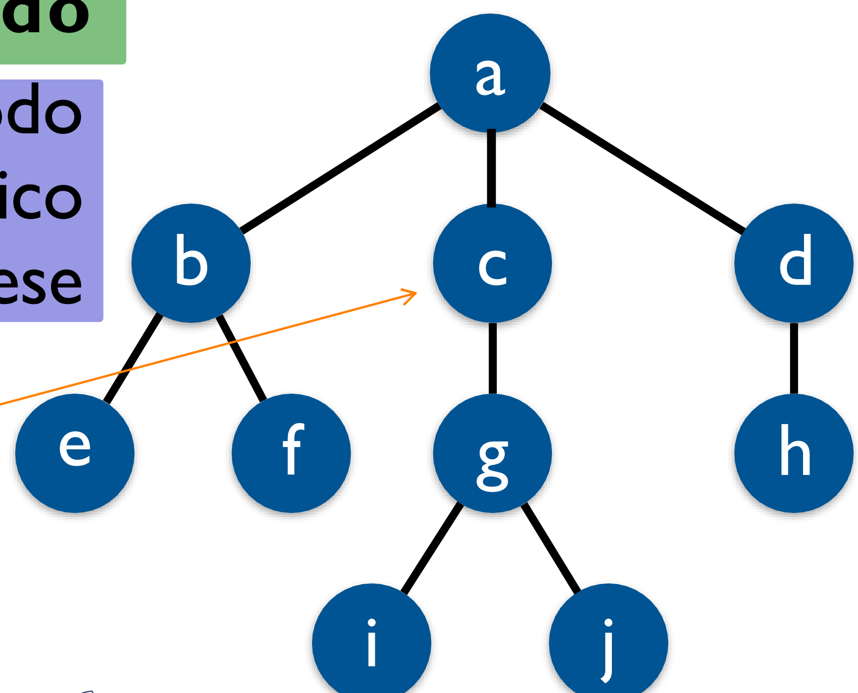
- Longitud del mayor de los caminos del nodo a cada hoja
- La altura de un árbol es la altura de su raíz

Profundidad de un nodo

- La profundidad de un nodo es la longitud del único camino de la raíz a ese nodo

Altura(c): 2

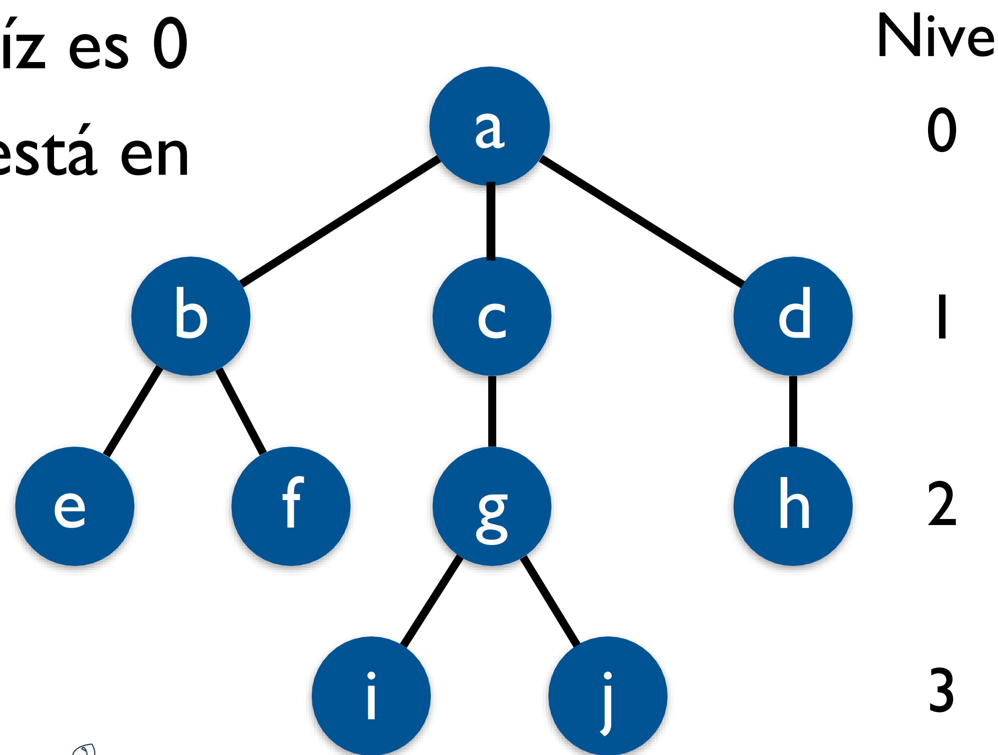
Profundidad(c): 1



Niveles

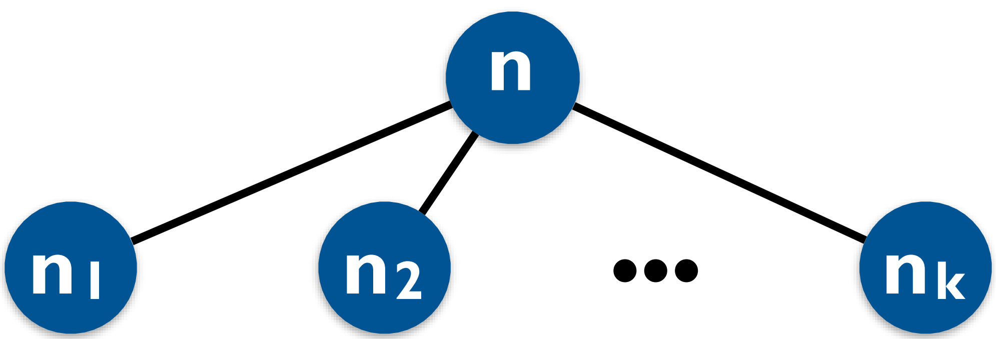
Nivel de un nodo

- Para un árbol de altura *h*, se definen los niveles 0, …, *h* de manera que el nivel *i* está compuesto por todos los nodos de profundidad *i*
- Base: El nivel del nodo raíz es 0
- Recurrencia: si un nodo está en el nivel i, todos sus hijos están en el nivel i+1
- Altura $h \rightarrow h+1$ niveles



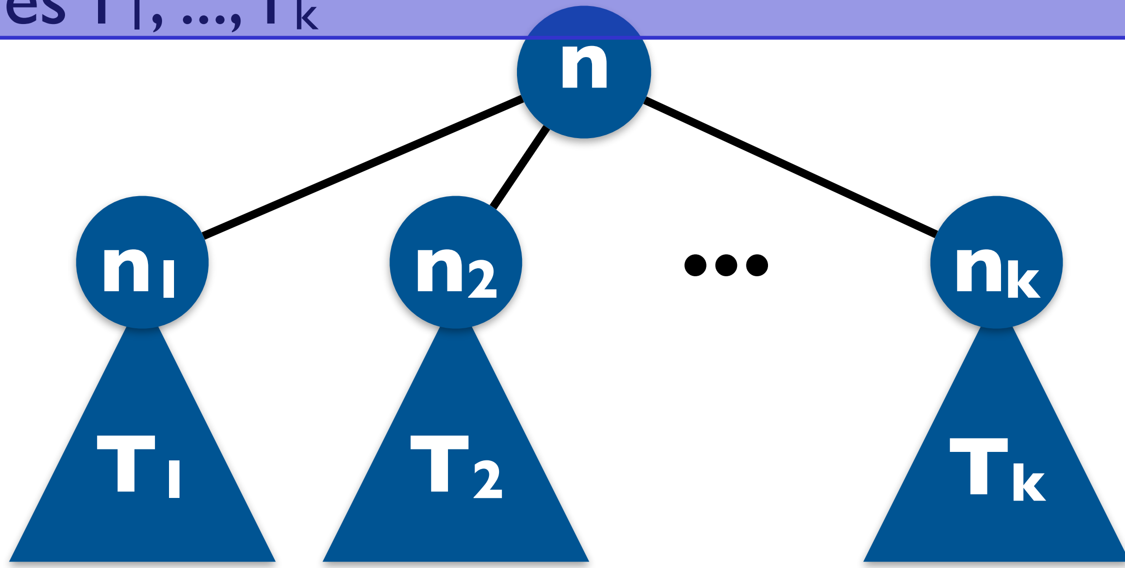
Orden

- Los hijos de un nodo están ordenados de izq. a dcha.
- Si queremos ignorar el orden → árbol no-ordenado
- Si n_1 y n_2 son hermanos, y n_1 está a la izquierda de n_2, todos los descendientes de n_1 están a la izquierda de todos los descendientes de n_2



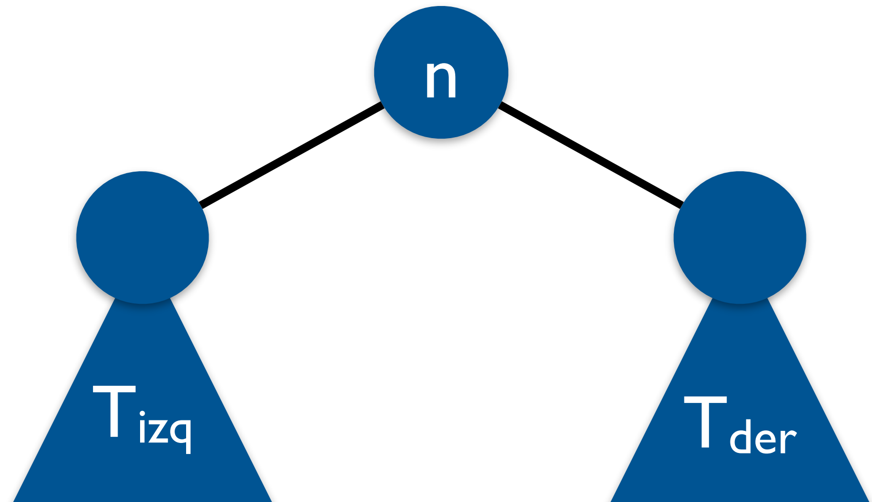
Árbol n-ario

- Base: Un nodo es un árbol n-ario (si el árbol tiene un sólo nodo, éste es el nodo raíz)
- Recurrencia: Si n es un nodo y $T_1, ..., T_k$ son árboles n-arios con raíces $n_1, ..., n_k$, respectivamente, podemos construir un árbol que tenga como raíz el nodo n y subárboles $T_1, ..., T_k$



Árboles binarios

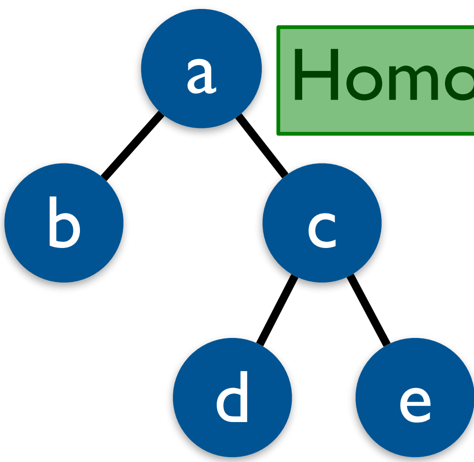
- Base: Un árbol vacío es un árbol binario
- Recurrencia: Si n es un nodo y T_{izq} y T_{der} son árboles binarios, podemos construir un nuevo árbol binario que tenga como raíz el nodo n y como subárboles T_{izq} y T_{der} (subárbol izquierdo y derecho, respectivamente)



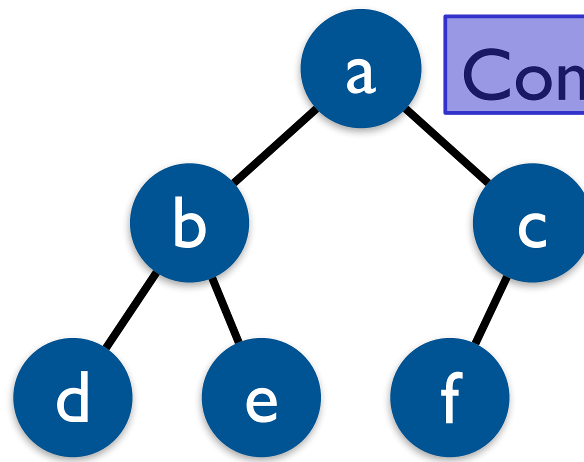
Un árbol binario NO es un árbol n-ario de grado 2

Árboles binarios

- **Árbol binario homogéneo**: aquél cuyos nodos tienen grado 0 ó 2 (no hay ninguno de grado 1)
- **Árbol binario completo**: aquél que tiene todos los niveles llenos excepto, quizá, el último, en cuyo caso los huecos deben quedar a la derecha



Homogéneo

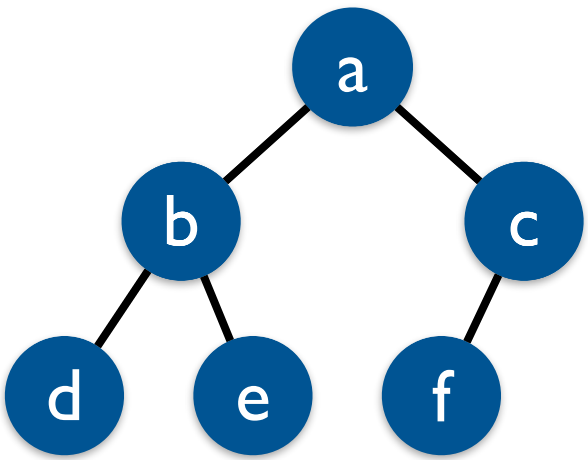


Completo

En un árbol binario completo con n nodos el camino más largo de la raíz a las hojas no atraviesa más de $\log_2 n$ nodos

Árboles binarios

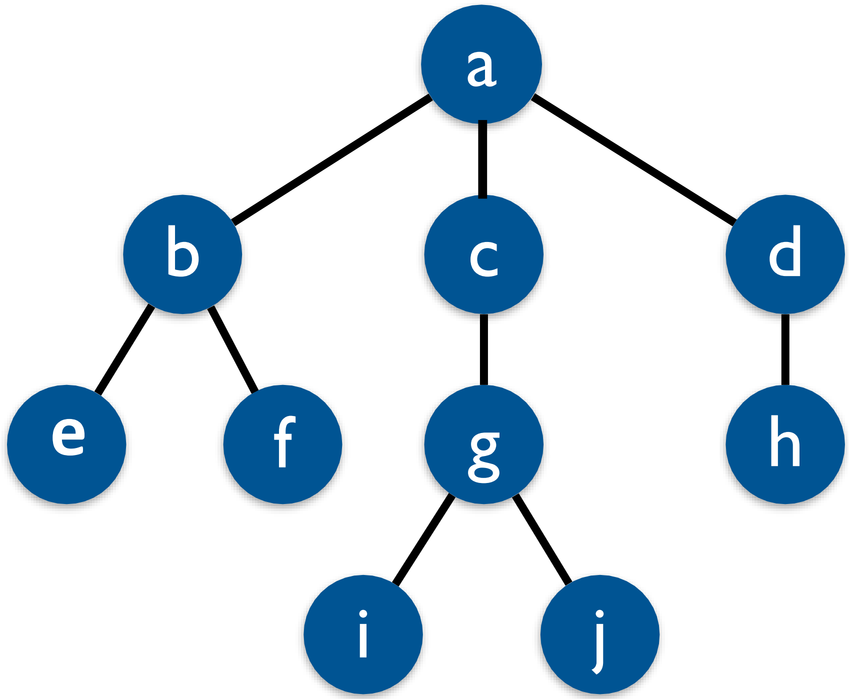
- En un árbol binario, el número máximo de nodos que puede haber en el nivel i es 2^i



Nivel	Nodos
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$

- En un árbol binario completo de altura k, el número máximo de nodos es $2^{k+1}-1$

Recorridos en árboles n-arios



Recorridos en árboles n-arios

- En una estructura lineal resulta trivial establecer un criterio de movimiento
- En un árbol no hay un criterio único
- Cuando hablamos de recorridos en un árbol, nos referimos al orden en el que visitamos sus nodos
- Tipos:
  - Profundidad/Recursivos
  - Anchura/Iterativo o por niveles

Profundidad / Recursivos

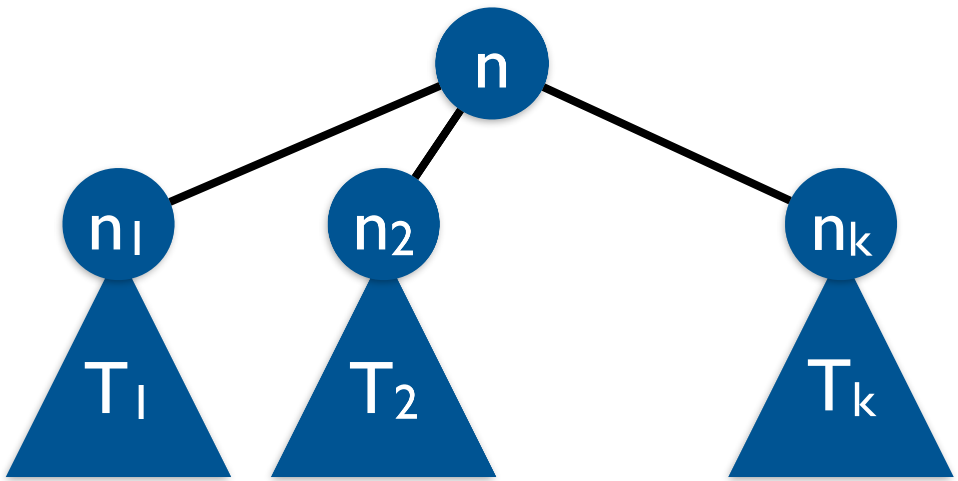
- Visitan los nodos desde la raíz hacia las hojas, dejándose nodos en un mismo nivel sin visitar hasta más adelante
- Se puede realizar de tres formas:
  - **Preorden**: Al visitar un nodo, se procesa en ese momento (bien para imprimir o hacer algo con él)
  - **Inorden**: Al visitar un nodo se procesará cuando se haya procesado su hijo más a la izquierda
  - **Postorden**: Al visitar un nodo se procesará cuando se hayan procesados todos sus hijos

Anchura / Iterativo

- Se visitan y procesan en primer lugar todos los nodos del mismo nivel de izquierda a derecha
- Se parte de igual forma desde la raíz y se avanza hacia las hojas, procesando todos los nodos del nivel 0, luego los del nivel 1, etc.

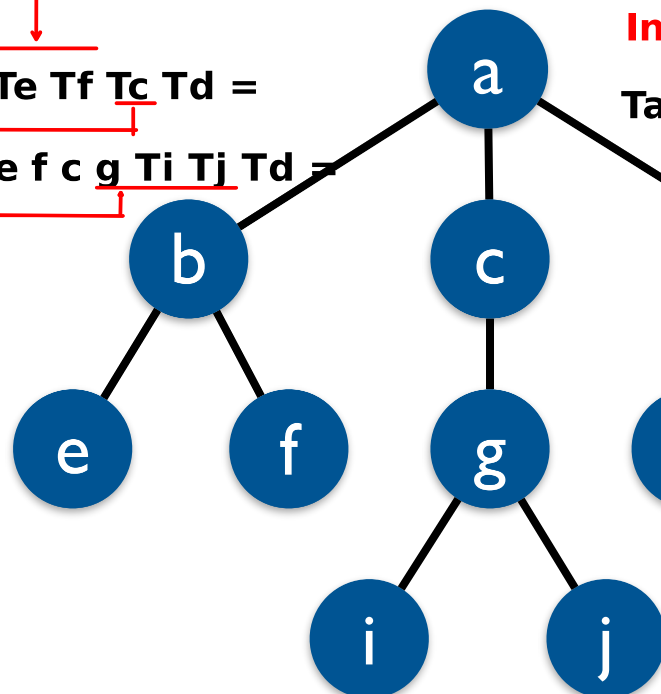
Recorridos en árboles n-arios

- Recorridos en profundidad:
  - **Preorden**: raíz, $Pre(T_1)$, $Pre(T_2)$,..., $Pre(T_k)$
  - **Inorden**: $In(T_1)$, raíz, $In(T_2)$,..., $In(T_k)$
  - **Postorden**: $Pos(T_1)$, $Pos(T_2)$,..., $Pos(T_k)$, raíz



- Recorrido en anchura: por niveles ➢ de arriba a abajo y de izquierda a derecha, empezando por la raíz

Recorridos en árboles n-arios



Pre:

Ta = a Tb Tc Td = a b Te Tf Tc Td =

= a b e f c Tg Td = a b e f c g Ti Tj Td =

= a b e f c g i j Td =

= a b e f c g i j d Th =

= a b e f c g i j d h

Ino:

Ta = Tb a Tc Td = Te b Tf a Tc Td =

= e b f a Tg c Td =

= e b f a Ti g Tj c Td =

= e b f a i g j c Th d =

= e b f a i g j c h d

Pos:

Ta = Tb Tc Td a =

= Te Tf b Tc Td a = e f b Tg c Td a =

= e f b Ti Tj g c Td a =

= e f b i j g c Td a =

= e f b i j g c Th d a =

= e f b i j g c h d a

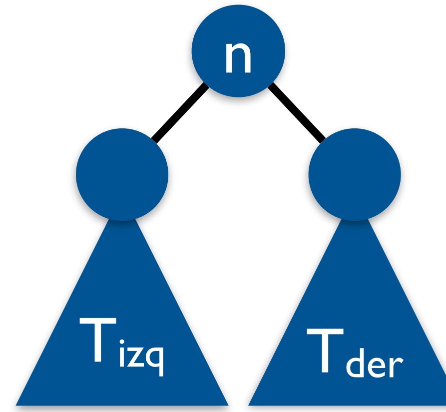
- Preorden: a b e f c g i j d h
- Inorden: e b f a i g j c h d
- Postorden: e f b i j g c h d a
- Por niveles: a b c d e f g h i j

Recursivos

Iterativo

Recorridos en árboles binarios

- Recorridos en profundidad:
  - **Preorden**: raíz, Pre(T_{izq}), Pre(T_{der})
  - **Inorden**: In(T_{izq}), raíz, In(T_{der})
  - **Postorden**: Pos(T_{izq}), Pos(T_{der}), raíz



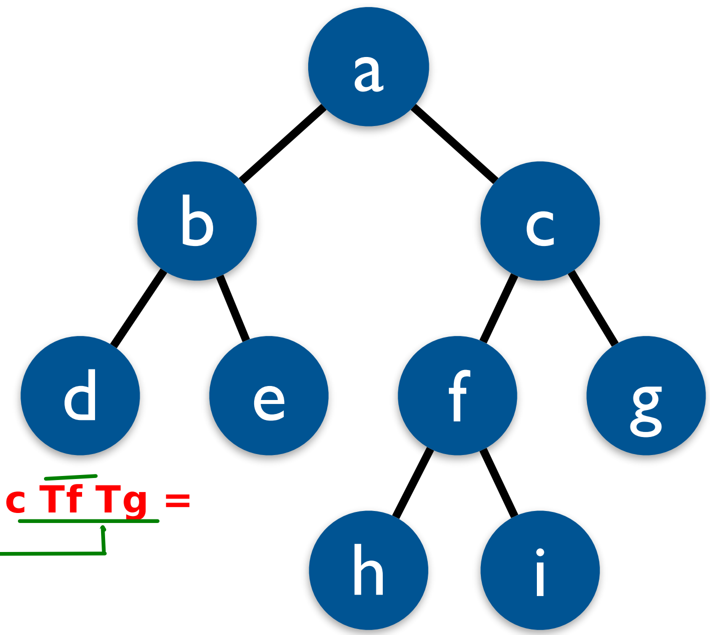
  Se pueden realizar de forma recursiva, siguiendo el esquema de construcción recursivo de árboles binarios

- Recorrido en anchura:
  - **Por niveles**, de izquierda a derecha

  Se realiza de forma iterativa

Recorridos en árboles binarios

- Preorden: a b d e c f h i g
- Inorden: d b e a h f i c g
- Postorden: d e b h i f g c a
- Por niveles: a b c d e f g h i



Pre: Ta= a Tb Tc = a b Td Te Tc = a b d e Tc = a b d e c Tf Tg =

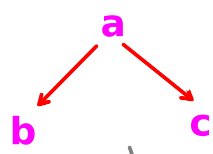
= a b d e c f Th Ti Tg = a b d e c f h i g

Ino: Ta = Tb a Tc = Td b Te a Tc = d b e a Tf c Tg = d b e a Th f Ti c Tg = d b e a h f i c g

Pos: Ta = Tb Tc a = Td Te b Tc a = d e b Tf Tg c a = d e b Th Ti f Tg c a = d e b h i f g c a

Niv: Ta = a b c d e f g h i

Recorridos en árboles binarios



In: b a c

In(b) < In(a) y b es descend. de a

In(a) < In(c) y a es ancestro de c

	Pre(n)<Pre(m)	In(n)<In(m)	Pos(n)<Pos(m)
n a la izquierda de m	✅	✅	✅
n a la derecha de m	❌	❌	❌
n descendiente de m	❌	✅❌	✅
n ancestro de m	✅	✅❌	❌

Uso para información ancestral:

Pre(n)<Pre(m) <=> n a la izqda de m || n ancestro de m

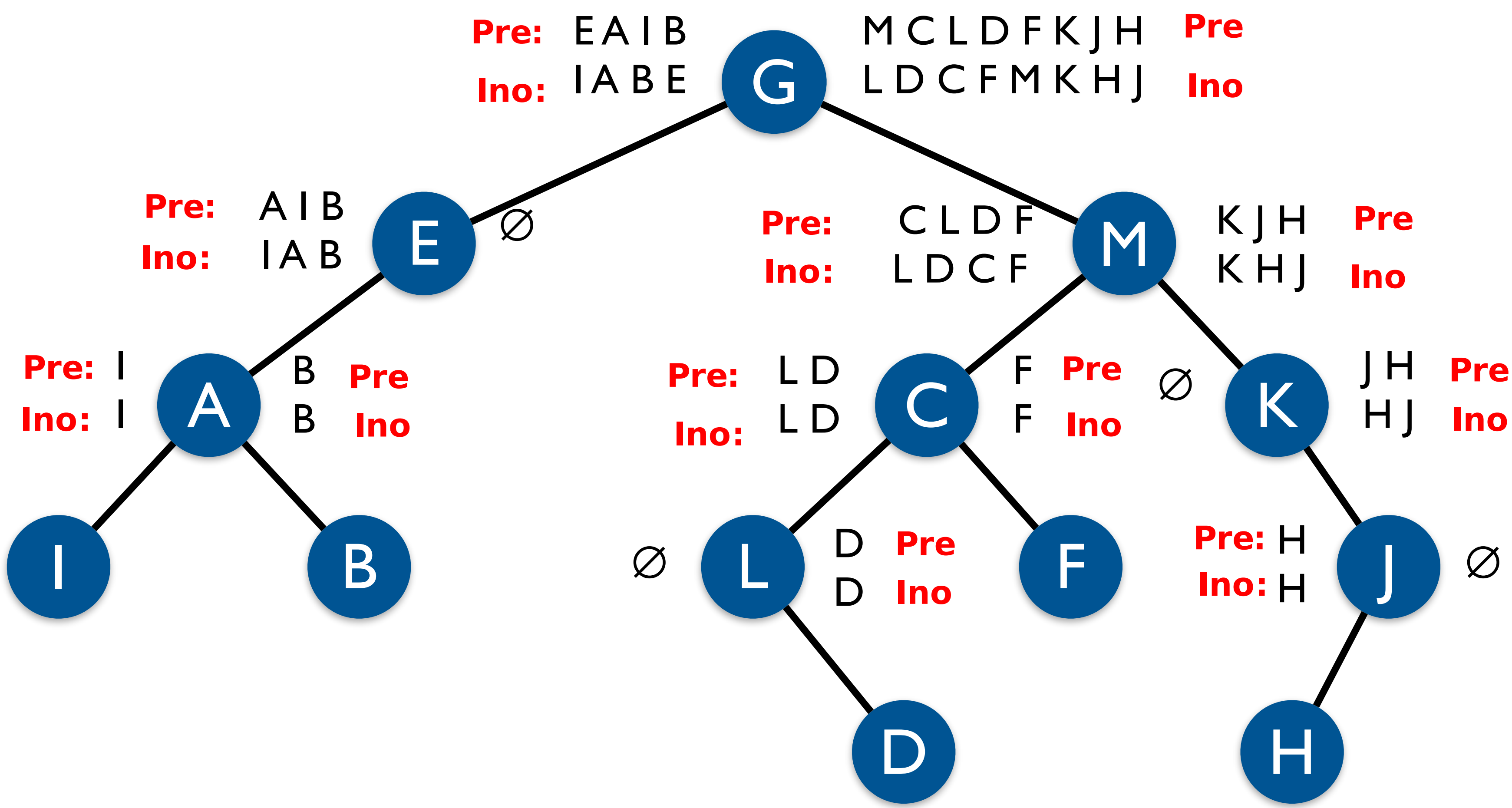
&&

Post(m)<Post(n) <=> m a la izqda de n || m descendiente de n

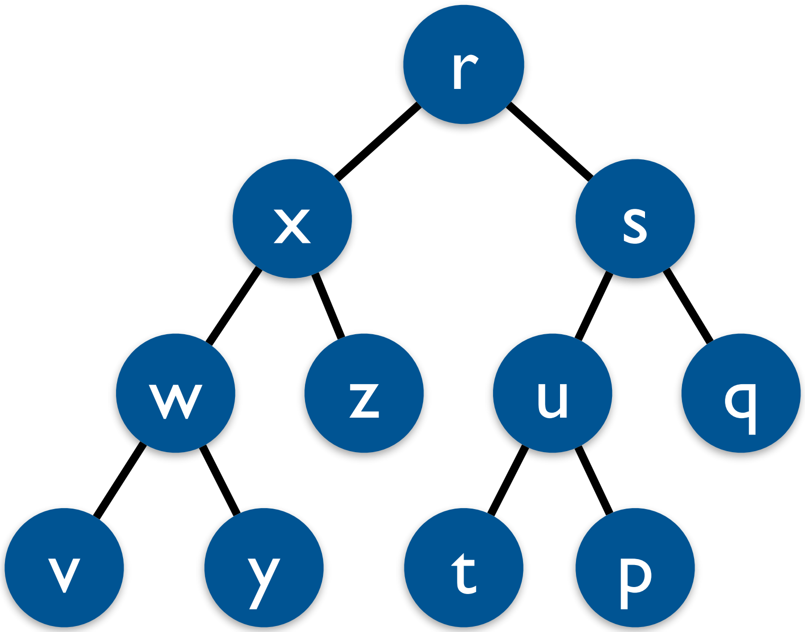
Pre(n) < Pre(m) && Post(m) < Post(n) <=> n ancestro de m

Recorridos en árboles binarios

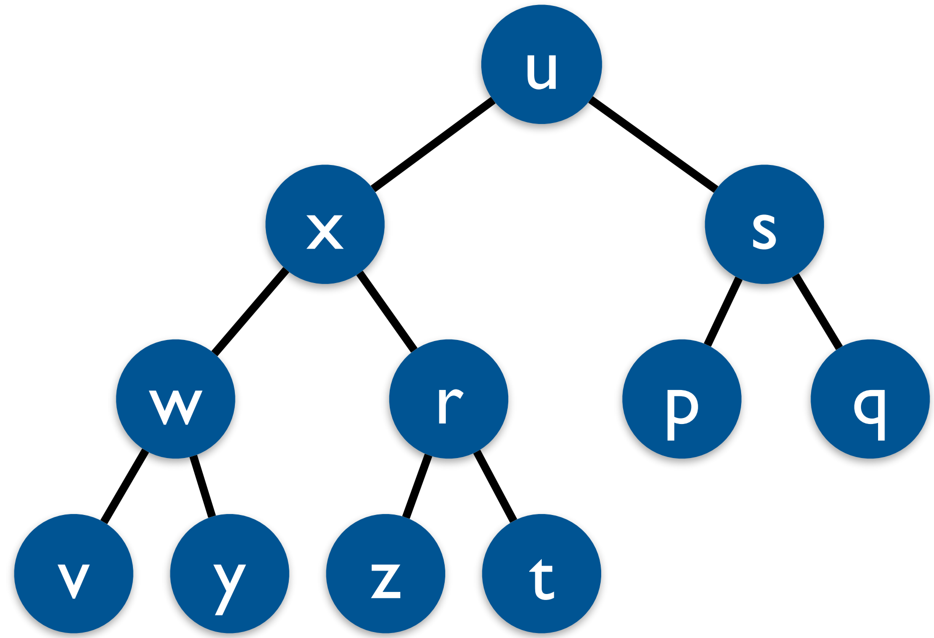
- Preorden: G E A I B M C L D F K J H
- Inorden: I A B E G L D C F M K H J



Recorridos en árboles binarios



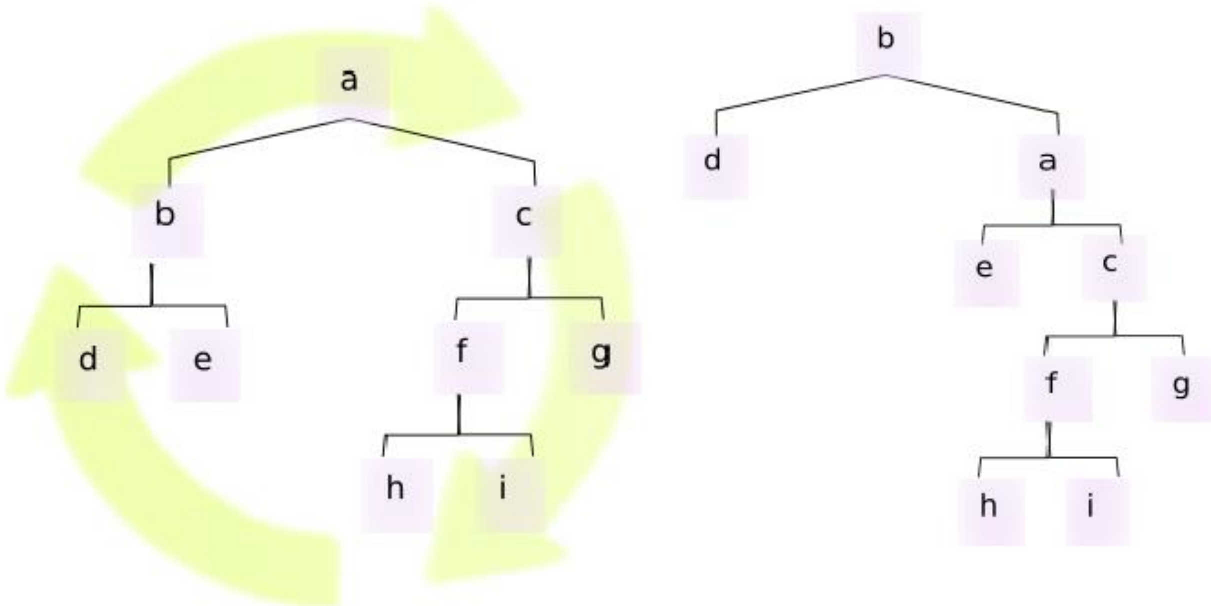
Inorden: v w y x z r t u p s q



Inorden: v w y x z r t u p s q

En general, un árbol no puede recuperarse con sólo uno de sus recorridos

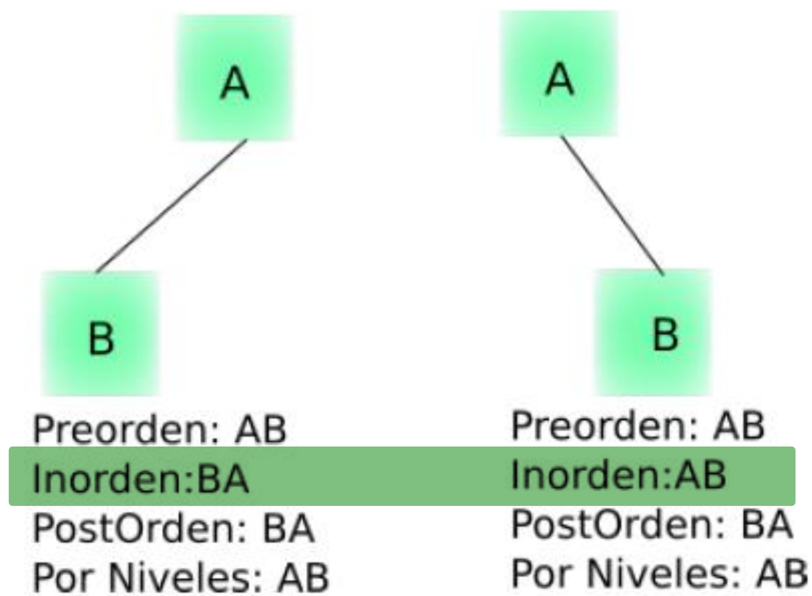
Recorridos en árboles binarios



Inorden: ***d b e a h f i c g***

Por normal general, con sólo uno de los recorridos de un árbol, no puede recuperarse de manera univoca, es decir, dos árboles diferentes pueden tener el mismo recorrido

Recorridos en árboles binarios



Recuperar un Árbol

Podemos recuperar el árbol de forma univoca si los listados que nos dan son:

- Inorden y Preorden
- Inorden y Postorden
- Inorden y Por Niveles

No podremos recuperar si nos dan:

- Postorden y Preorden (hay alguna excepción para los árboles binarios pero en general no se puede)
- Preorden y Por Niveles
- Postorden y Por Niveles

Incluso si nos dieran el Preorden, Postorden y Por niveles no podemos definir el árbol

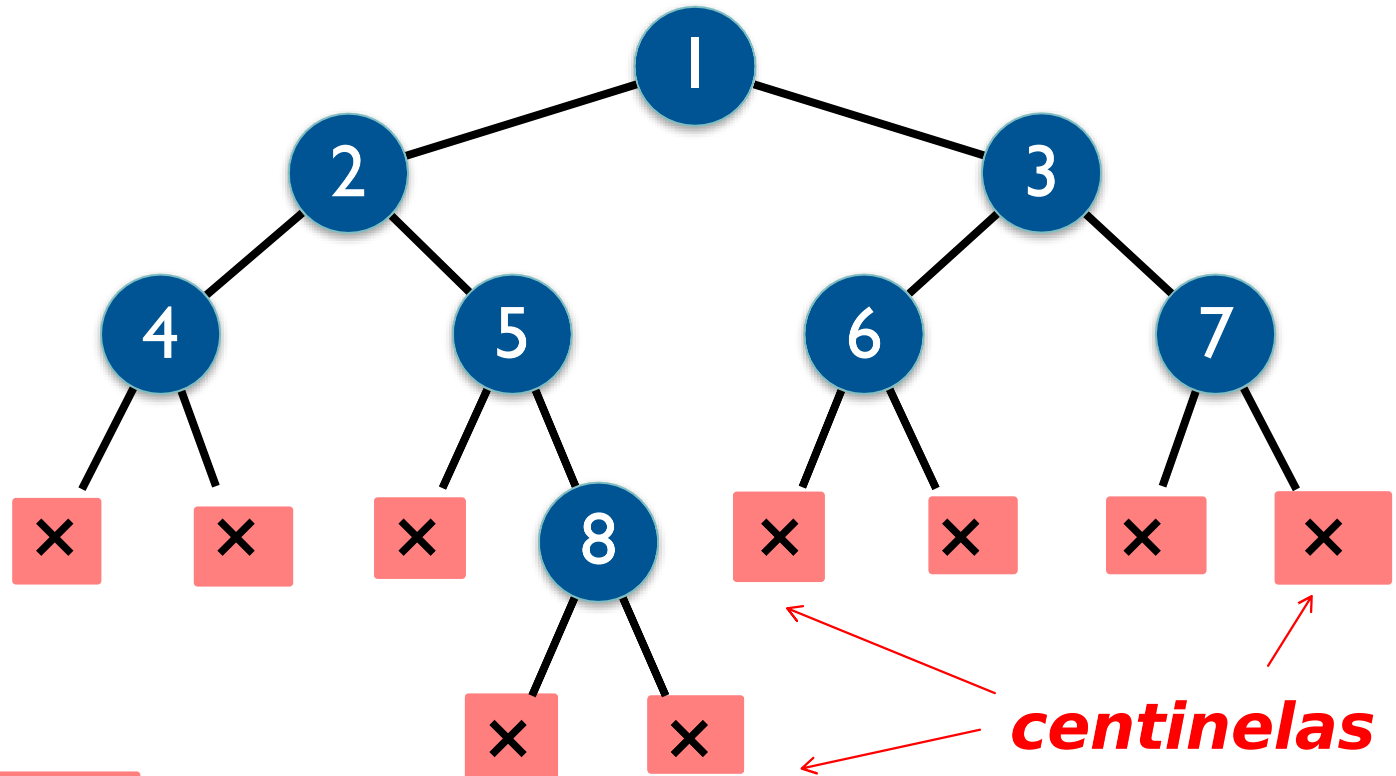
Lectura/escritura de un árbol

- Para guardar un árbol en disco, se realiza un preorden del árbol transformado

- Este árbol transformado consiste en añadirle a los nodos que no tienen los dos hijos (si es un árbol binario) un nuevo nodo ficticio, que tiene como etiqueta x

- Cuando hacemos el listado del árbol si el nodo existe se le antepone a la etiqueta n y si es un nodo ficticio simplemente listamos x

Lectura/escritura de un árbol

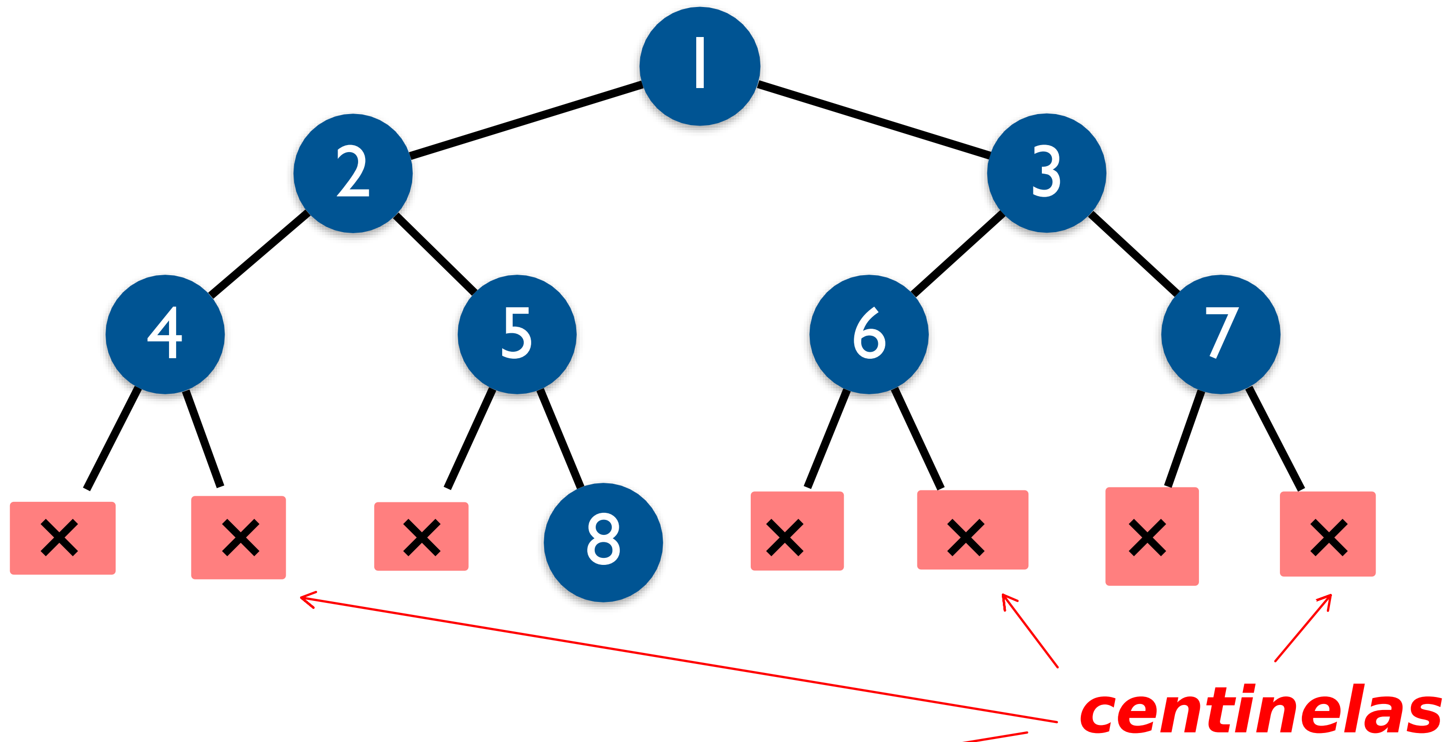


Preorden

n 1 n 2 n 4 x x n 5 x n 8 x x n 3 n 6 x x n 7 x x

1 2 4 5 8 3 6 7

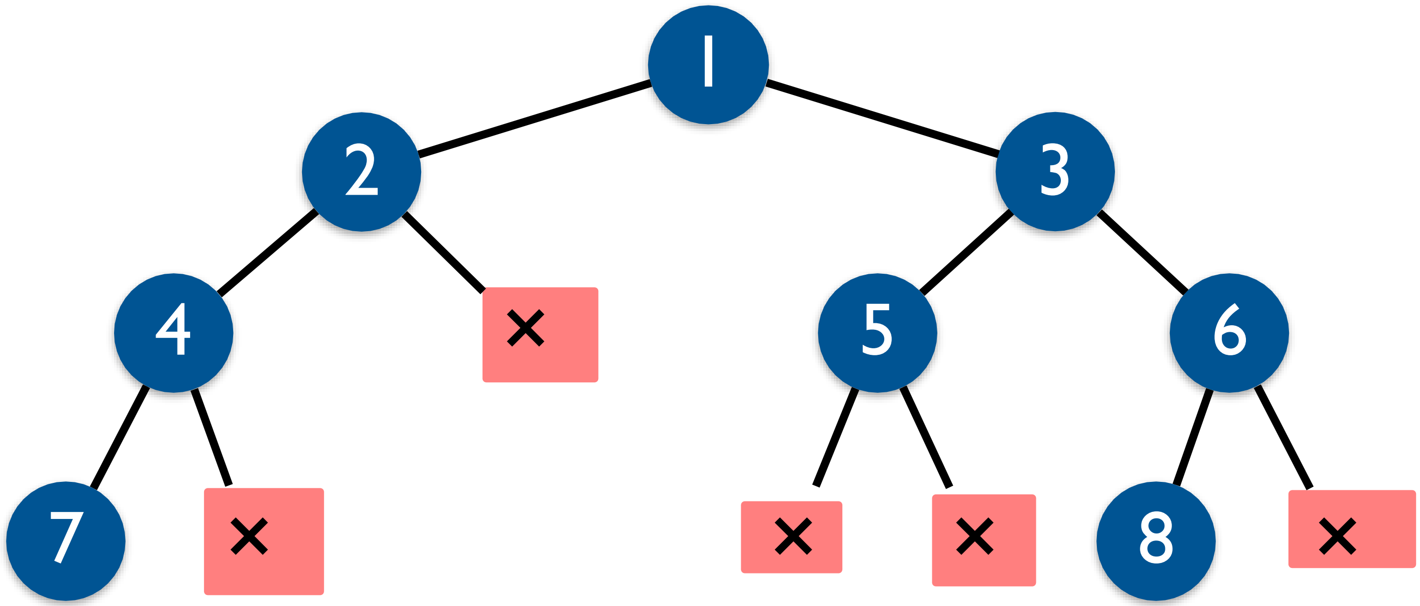
Lectura/escritura de un árbol



Por niveles

1 $\boxed{-1}$ 2 3 $\boxed{-1}$ 4 5 6 7 $\boxed{-1}$ -1 -1 -1 8 -1 -1 -1 -1 $\boxed{-1}$

Lectura/escritura de un árbol

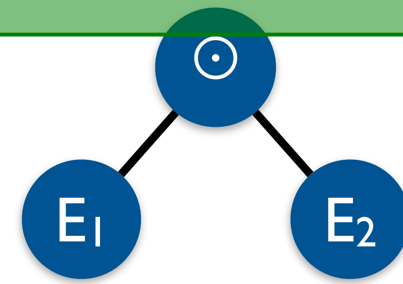


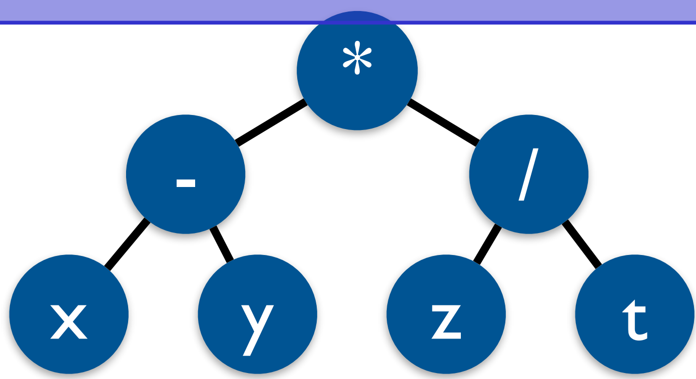
Por niveles

1 $\boxed{-1}$ 2 3 $\boxed{-1}$ 4 -1 5 6 $\boxed{-1}$ 7 -1 -1 -1 -1 -1 8 -1 $\boxed{-1}$

Aplicación: árboles de expresión

- **Árboles sintácticos**: árboles que contienen las derivaciones de una gramática necesarias para obtener una frase del lenguaje
- **Árboles de expresión**: etiquetamos
  - hojas con un operando
  - nodos interiores con un operador





((x)-(y))*((z)/(t))

≡

(x-y) * (z/t) Inorden

Preorden: *-xy/zt ➢ Representación prefija

Postorden: xy-zt/* ➢ Representación postfija

No necesita paréntesis

Aplicación: árboles de expresión

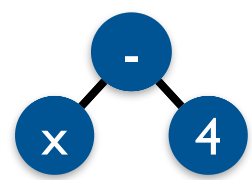
- **Resolución de ambigüedades**: recorridos en preorden o postorden más
  - Nivel de cada nodo, ó
  - Número de hijos de cada nodo

Ejemplo: x4-2^y2+3/* (postfijo)

Los operadores -, ^, +, / y * son binarios

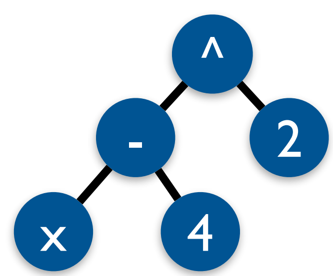
x 4 - 2 ^ y 2 + 3 / *

(x-4)



(x-4) 2 ^ y 2 + 3 / *

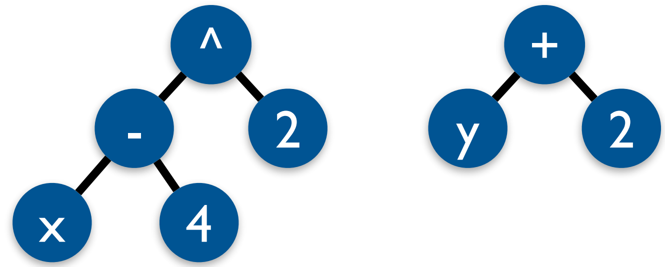
((x-4)^2)



Aplicación: árboles de expresión

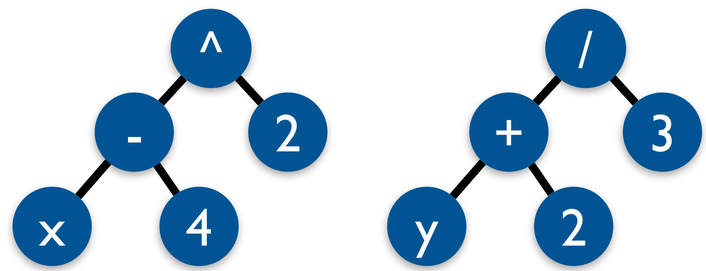
((x-4)^2) y 2 + 3 / *

((x-4)^2) (y+2)



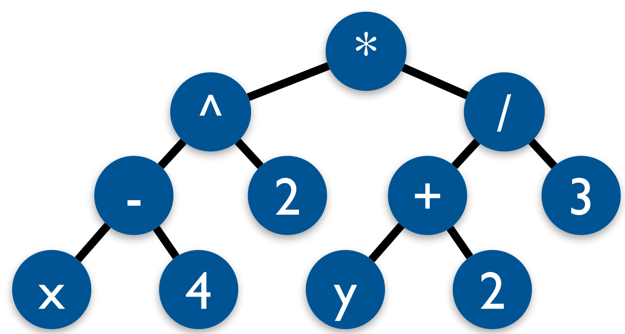
((x-4)^2) (y+2) 3 / *

((x-4)^2) ((y+2)/3)



((x-4)^2) ((y+2)/3) *

((x-4)^2)*((y+2)/3)



Aplicación: árboles de expresión

- Las notaciones prefija y posfija facilitan la evaluación automática de expresiones aritméticas
- Ejemplo: ((15 / (7 - (1 + 1))) * 3) - (2 + (1 + 1))

```
-   *   /   15  -   7   +   1   1   3   +   2   +   1   1   =
-   *   /   15  -   7         2     3   +   2   +   1   1   =
-   *   /   15          5           3   +   2   +   1   1   =
-   *               3               3   +   2   +   1   1   =
-                         9             +   2   +   1   1   =
-                         9             +   2         2     =
-                         9                     4           =
                                        5
```

Aplicación: árboles de expresión

$$\underbrace{[(a+b)+(c*(d+e)+f)]}_{E_1} * \underbrace{(g+h)}_{E_2} \Rightarrow * E_1 E_2$$

$$[\underbrace{(a+b)}_{E_{11}} + \underbrace{(c*(d+e)+f)}_{E_{12}}] \Rightarrow * + E_{11} E_{12} E_2$$

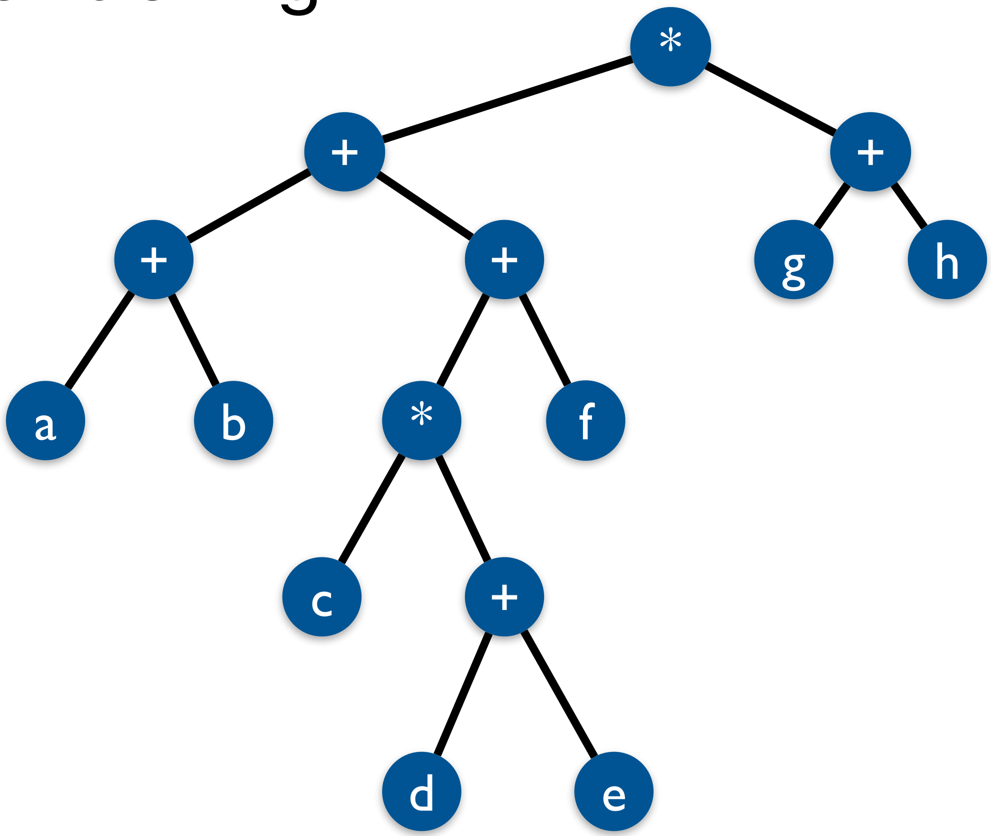
$$\left.\begin{array}{l} E_{11} \equiv +ab \Rightarrow * + + a\, b\, E_{12}\, E_2 \\ E_{12} \equiv [\underbrace{(c*(d+e)}_{E_{121}} + \underbrace{f}_{E_{122}}] \Rightarrow + E_{121}\, E_{122} \end{array}\right\} * + + a\, b + E_{121}\, E_{122}\, E_2$$

$$E_{121} \equiv \underbrace{c}_{E_{1211}} * \underbrace{(d+e)}_{E_{1212}} \Rightarrow * c E_{1212} \equiv * c + de$$

$$\left.\begin{array}{l} * + + a\, b + * c + d\, e\, f\, E_2 \\ E_2 \equiv (g+h) \equiv + g\, h \end{array}\right\} * + + a\, b + * c + d\, e\, f + g\, h$$

Aplicación: árboles de expresión

`*++ab+*c+def+gh`



Árboles binarios

- **Especificación**
  1. Son árboles tal que cada nodo tiene 0, 1 o 2 hijos
  2. Cada nodo tiene un nodo padre, a excepción del nodo raíz
  3. El árbol vacío es un árbol binario

Árboles binarios

Definición de funciones en árboles

- Generalmente la forma más simple de definir una función sobre un árbol es trasladar la definición recurrente (recursiva) del dominio a la definición de ésta

- Esto no quiere decir que toda función definida sobre un árbol deba ser recursiva

- Podemos encontrar problemas cuya solución exija diseñar funciones iterativas, ya que no es posible encontrar una función recursiva (por extensión de la definición recurrente del dominio) que lo resuelva
- Ej: el recorrido por niveles del árbol

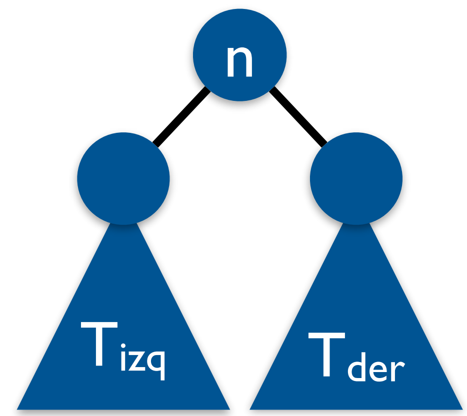
Árboles binarios

- **Función f(t) sobre un árbol binario, t**: definición por extensión de la definición del conjunto de árboles binarios

- **Base**: Valor de la función si t es el árbol vacío

- **Recurrencia**: se supone conocida la función para cada uno de los subárboles T_{izq} y T_{der} de t.

  Se calcula el valor final de la función suponiendo conocidos los valores anteriores



Árboles binarios

Ejemplo: igualdad de árboles binarios

- Base: Si t_1 y t_2 son árboles binarios vacíos, son iguales
- Recurrencia: Hipótesis
  - igual(t_{izq1}, t_{izq2})
  - igual(t_{der1}, t_{der2})

t_{izqi} y t_{deri} son los subárboles izquierdo y derecho de t_i

- Tesis: Los árboles binarios t_1 y t_2 serán iguales si se cumplen las condiciones:
  - t_1.etiqueta() == t_2.etiqueta()
  - igual(t_{izq1}, t_{izq2}), e
  - igual(t_{der1}, t_{der2})

Árboles binarios

Ejemplo: altura de un árbol binario

- Base: Si t es un árbol binario vacío, su altura es 0
- Recurrencia: Hipótesis
  - altura(t_{izq}) = a_{izq}
  - altura(t_{der}) = a_{der}

  t_{izq} y t_{der} son los subárboles izquierdo y derecho de t
- Tesis: la altura se calcula como:

  $$1 + \text{máximo}(a_{izq}, a_{der})$$
- Ejercicios:
  - Contar el número de nodos de un árbol
  - Calcular el grado de un árbol

Árboles binarios

Ejemplo: árboles binarios isomorfos

- Base: Si t_1 y t_2 son árboles binarios vacíos, son isomorfos
- Recurrencia: Hipótesis
  - $iso(t_{izq1}, t_{izq2})$
  - $iso(t_{izq1}, t_{der2})$
  - $iso(t_{der1}, t_{der2})$
  - $iso(t_{der1}, t_{izq2})$

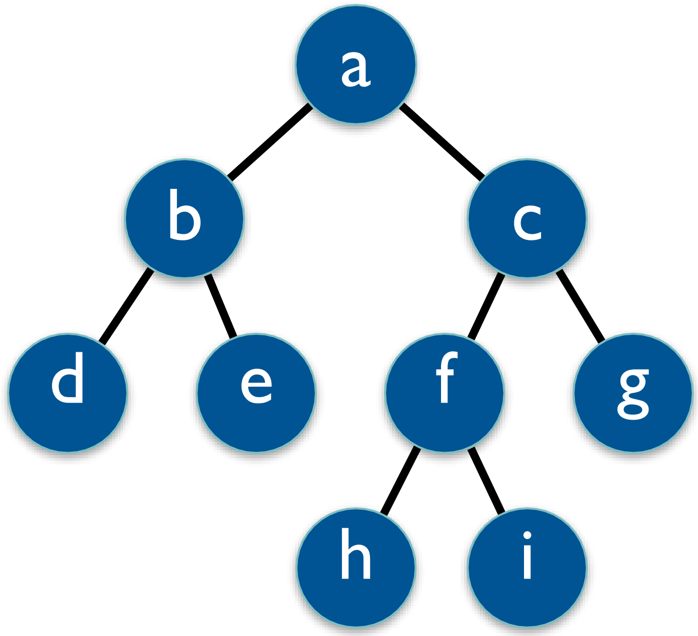
t_{izqi} y t_{deri} son los subárboles izquierdo y derecho de t_i

- Tesis: Los árboles binarios t_1 y t_2 serán isomorfos si se cumplen las condiciones:
  - t_1.etiqueta() == t_2.etiqueta()
  - $iso(t_{izq1}, t_{izq2})$ e $iso(t_{der1}, t_{der2})$, ó
  - $iso(t_{izq1}, t_{der2})$ e $iso(t_{der1}, t_{izq2})$

Representación Estática

1. Las etiquetas de los nodos se almacenan en un vector
2. Los nodos se enumeran de la siguiente forma:
    - A la raíz le corresponde el índice 0
    - Si a un nodo le corresponde el índice k:
        - Su hijo izquierdo, si tiene, está en la posición $2*(k+1)-1 = 2*k+1$
        - Su hijo derecho, si tiene, está en la posición $2*(k+1) = 2*k+2$
        - Su padre, si tiene, está en la posición $(k-1)/2$

Representación Estática



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
a	b	c	d	e	f	g					h	i			...

Representación Estática

- Averiguar la información ancestral de los nodos es muy simple porque mantienen posiciones fijas
- Útil en árboles completos en los que el vector no tendría huecos
- Por lo general, no es eficiente en espacio

Representación Dinámica

- La implementación dinámica la vamos a dividir por capas
- Primero definimos un objeto de tipo nodo (lo llamaremos *info_nodo*) en el que tendremos:
  1. La información o etiqueta que almacena
  2. Enlaces al padre, hijo izquierda e hijo derecha
- Hay que tener en cuenta que dando el nodo raíz, tendremos la información del árbol completo

Representación Dinámica

```
#include <queue> //para hacer el recorrido por niveles
using namespace std;
template <class T>
struct info_nodo {
        info_nodo *padre, //puntero al padre
        *hijoizq, //puntero al hijo izquierda
        *hijodcha; // puntero al hijo derecha
        T et; // etiqueta del nodo

        // Constructor por defecto del struct
        info_nodo() {
                padre = hijoizq = hijodcha = 0;
        }

        info_nodo(const T &e) {
                et = e;
                padre = hijoizq = hijodcha = 0;
        }
};
```

Representación mediante celdas enlazadas