Árboles

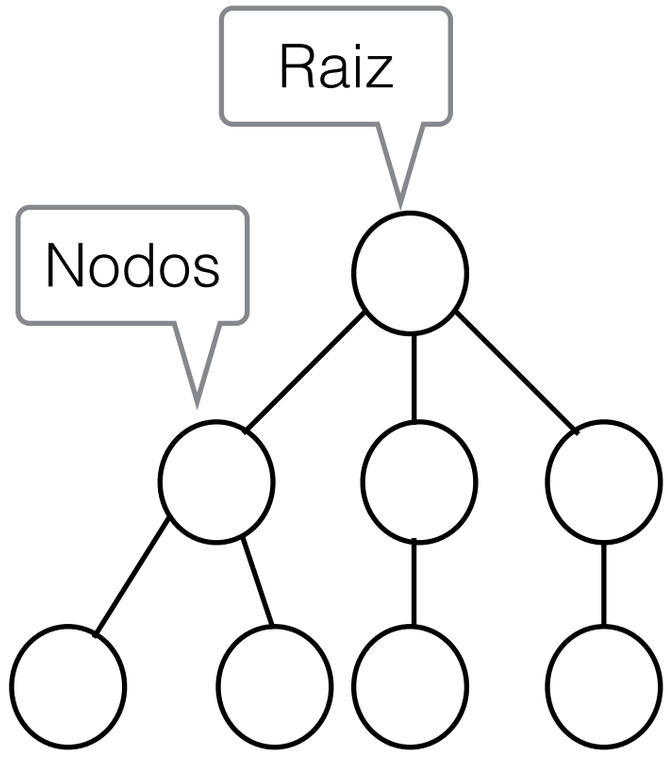
Estructuras de Datos y Algoritmos /
Algoritmos y Estructuras de Datos II
Año 2025

Dr. Pablo Ponzio
Universidad Nacional de Río Cuarto
CONICET

Árboles

Los árboles tienen las siguientes características:

- No poseen una organización lineal,
- Tenemos una raíz,
- Cada nodo tiene un padre, excepto la raíz,
- Un nodo puede tener 0 ó muchos hijos



Aplicaciones

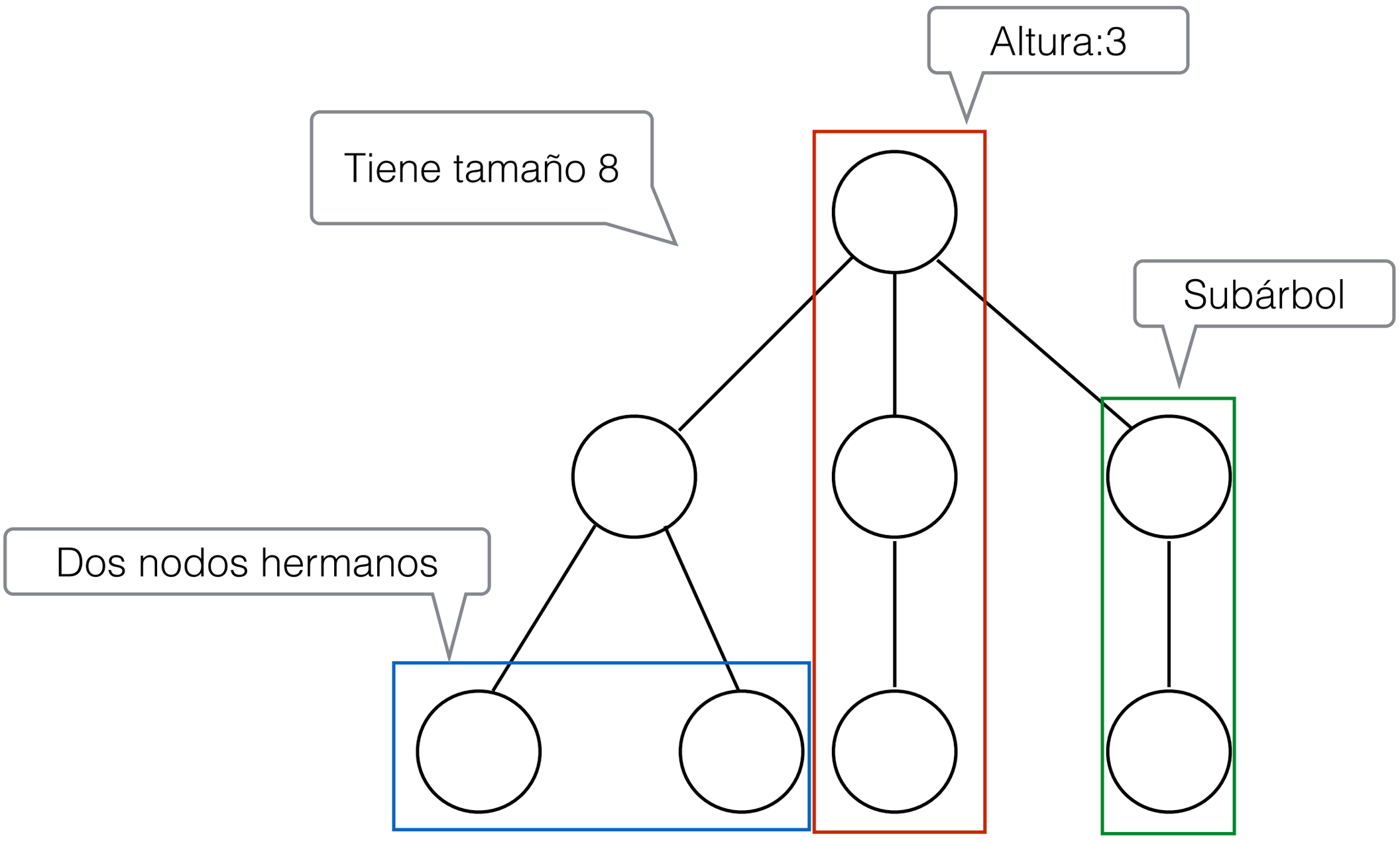
Los árboles tienen diversas aplicaciones, por ejemplo:

- Representación de expresiones (aritméticas, booleanas, etc),
- Implementación eficiente de colecciones de datos (bases de datos),
- Organización de datos (sistema de archivos),
- Varios algoritmos importantes usan árboles para obtener una implementación elegante y eficiente.

Definiciones

- **Hermanos:** Nodos con el mismo padre,
- **Hojas:** Nodos sin hijos,
- **Altura:** Cantidad de nodos en el camino más largo desde la raíz a una hoja,
- **Tamaño:** Cantidad de nodos,
- **Subárbol:** Es un nodo del árbol junto con todos sus descendientes

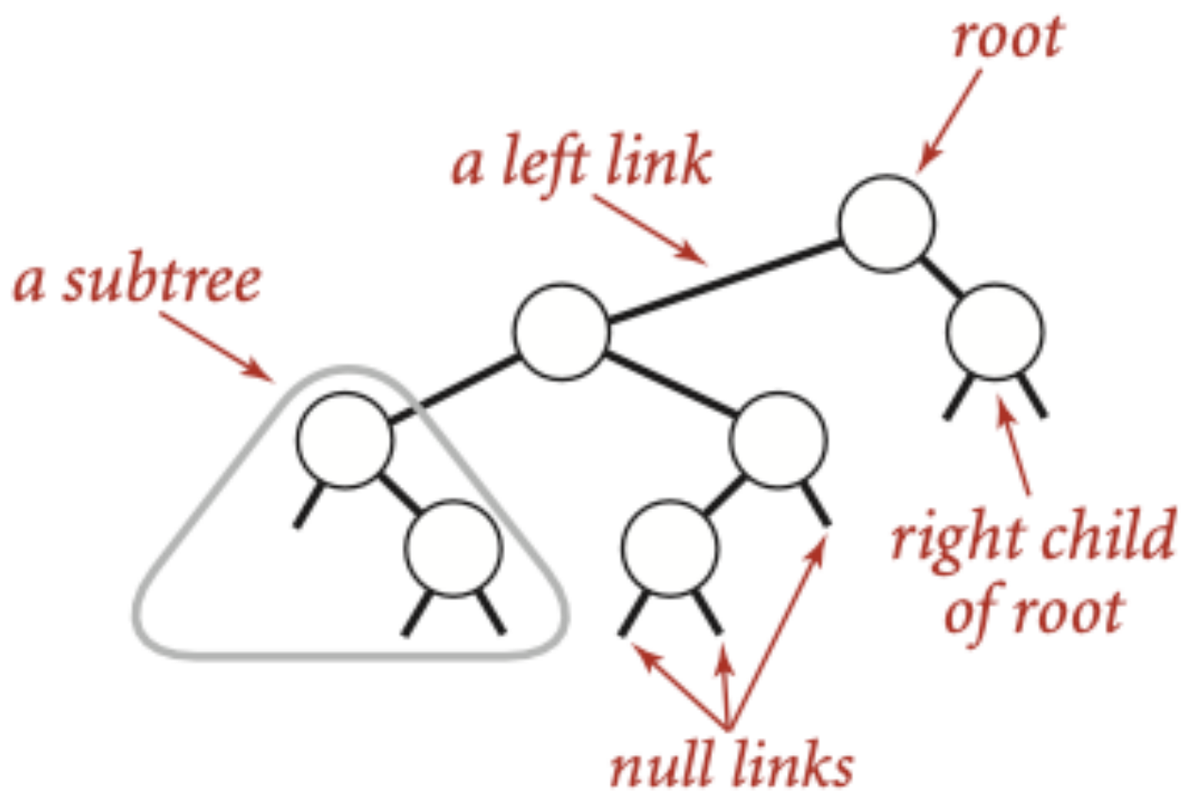
Ejemplo



Árboles Binarios

Los árboles binarios son aquellos que:

- Cada nodo tiene a lo sumo dos hijos,
- Cada hijo de un nodo es llamado hijo izq. o hijo derecho.



Árboles Binarios en Haskell

Los árboles binarios se definen recursivamente de la siguiente manera:

```
data BinTree a = Nil | Node (BinTree a) a (BinTree a)
```

hijo izquierdo

hijo derecho

Definamos algunas funciones sobre árboles:

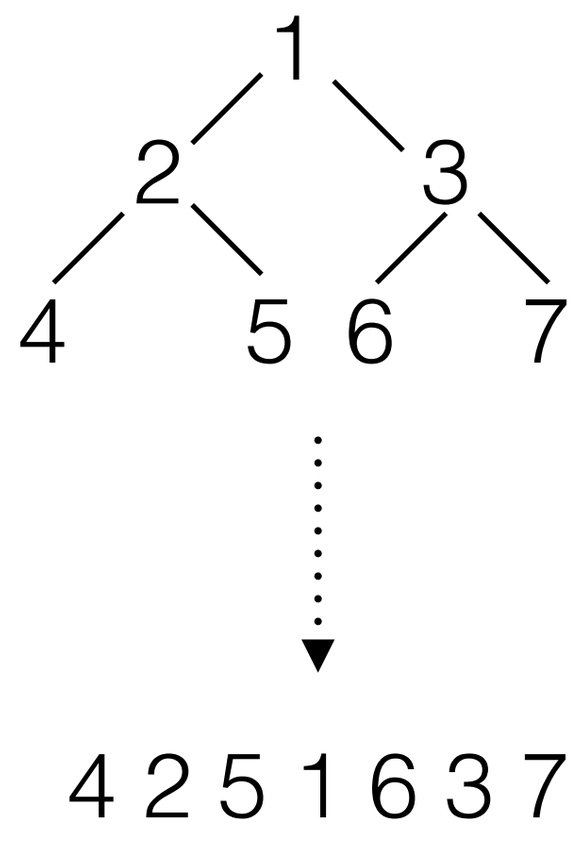
```
size:: BinTree a->Int
size Nil = 0
size (Node hi r hd) = 1 + size hi + size hd

height :: BinTree a->Int
height Nil = 0
height (Node hi _ hd) = 1 + max (height hi)
(height hd)
```

Recorridos: Inorder

Hay 3 formas de recorrer un árbol binario

Inorder: Primero se recorre el hi, después la raíz y finalmente el hd.

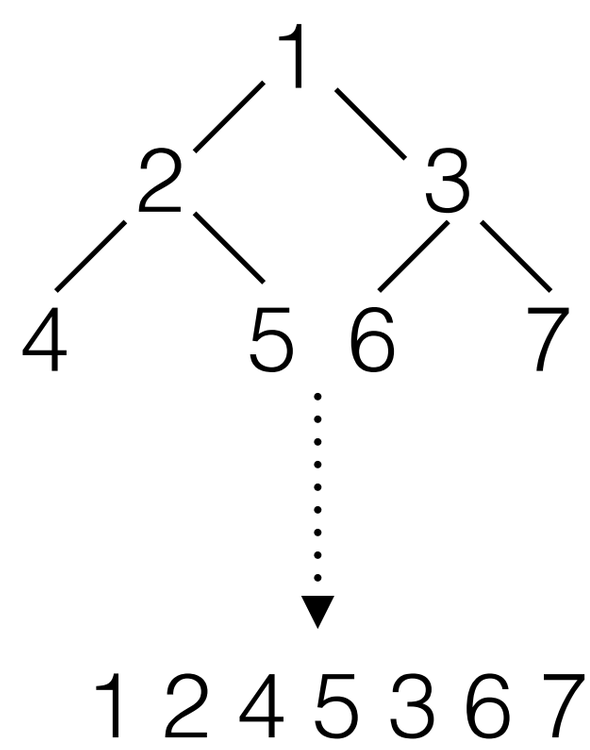
```
inorder :: BinTree a -> [a]
inorder Nil = []
inorder (Node hi r hd) =
    inorder hi ++ [r] ++ inorder hd
```



Recorridos: Preorder

Con preorder se recorre primero la raíz, después el hi y finalmente el hd.

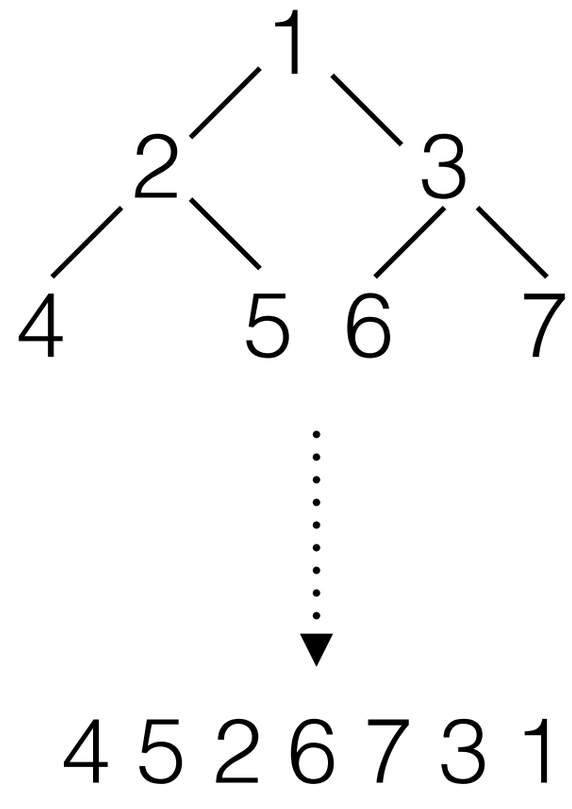
```
preorder :: BinTree a -> [a]
preorder Nil = []
preorder (Node hi r hd) =
    r : (preorder hi) ++ preorder hd
```



Recorridos: Posorder

Con posorder se recorre primero el hi, después el hd, y por último la raíz

```
posorder :: BinTree a -> [a]
posorder Nil = []
posorder (Node hi r hd) =
    posorder hi ++ posorder hd ++ [r]
```



Propiedades

Sea *t* un árbol binario, en donde *size(t)* es su tamaño, *alt(t)* su altura

full(t) dice si el árbol está lleno: tiene todos los nodos en todos los niveles

Propiedad 1: $size(t) \leq 2^{alt(t)} - 1$

Propiedad 2: $log_2(size(t) + 1) \leq alt(t)$

Propiedad 3: $full(t) \Rightarrow size(t) = 2^{alt(t)} - 1$

Propiedad 4: $full(t) \Rightarrow alt(t) = log_2(size(t) + 1)$

Propiedades

Sea *t* un árbol binario, en donde *size(t)* es su tamaño, *alt(t)* su altura

full(t) dice si el árbol está lleno: tiene todos los nodos en todos los niveles

Propiedad 1: $size(t) \leq 2^{alt(t)} - 1$

Propiedad 2: $log_2(size(t) + 1) \leq alt(t)$

Propiedad 3: $full(t) \Rightarrow size(t) = 2^{alt(t)} - 1$

Propiedad 4: $full(t) \Rightarrow alt(t) = log_2(size(t) + 1)$

Ejercicio: Probar estas propiedades usando inducción

Árboles Binarios con valores en las hojas

Otra posibilidad de representación de árboles binarios consiste en almacenar los valores en las hojas del árbol:

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

Ejemplos de funciones sobre estos árboles:

```
size :: Tree a -> Int
size (Leaf x)                = 1
size (Branch left right)  = size left + size right

fringe :: Tree a -> [a]
fringe (Leaf x)                 = [x]
fringe (Branch left right) = fringe left ++ fringe
right
```

Actividades

- Leer los capítulos 1-5 de "A Gentle Introduction to Haskell". P. Hudak, J. Peterson & J. Fasel. 1998. Disponible en: https://www.haskell.org/tutorial/

Bibliografía

- "A gentle introduction to Haskell". P. Hudak, J. Peterson & J. Fasel. 1998. Disponible en: https://www.haskell.org/tutorial/