

Algoritmos y Estructuras de Datos II

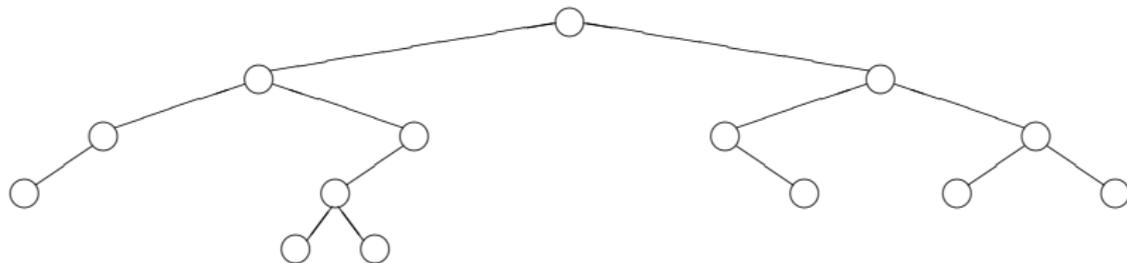
Árboles binarios

18 de abril de 2016

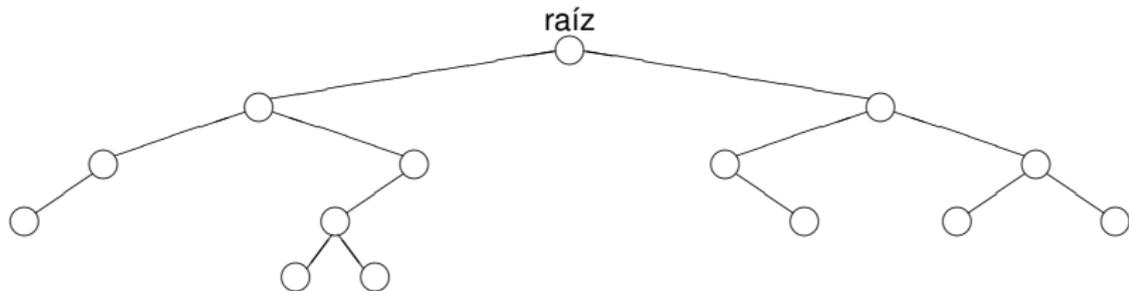
Clase de hoy

- 1 Árboles Binarios
 - Intuición
 - Especificación
 - Terminología habitual
 - Implementación con punteros
 - Posiciones

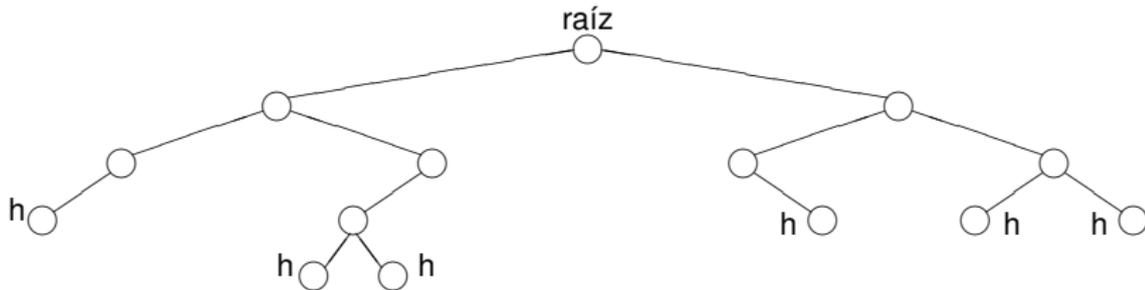
Intuición



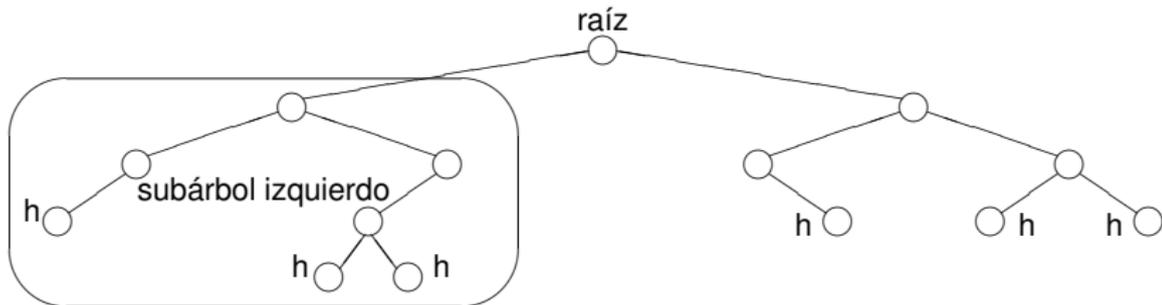
Intuición



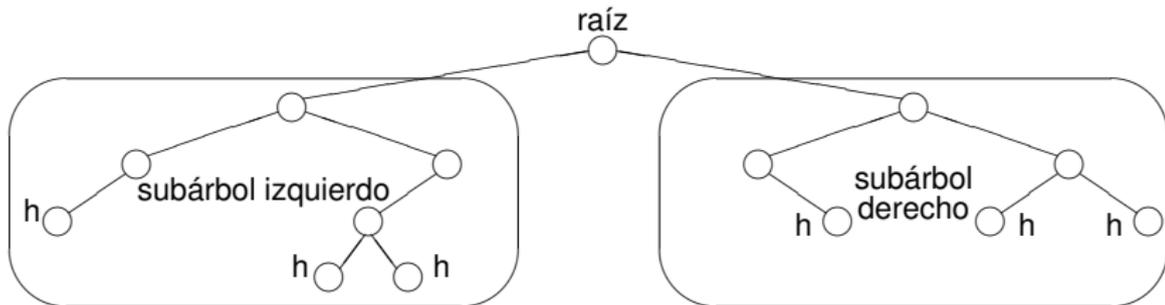
Intuición



Intuición



Intuición



- De cada círculo, cuelgan dos subárboles.
- En el caso de las hojas, los subárboles son vacíos.
- Todos los árboles pueden construirse con los constructores
 - $\langle \rangle$, que construye un árbol vacío
 - $\langle _, _, _ \rangle$, que construye un árbol no vacío a partir de un elemento y dos subárboles

Especificación del TAD árbol binario

module TADÁrbolBinario **where**

data ÁrbolBinario e = <>
| <_,_,_> (ÁrbolBinario e) e (ÁrbolBinario e)

es_vacío :: ÁrbolBinario e → Bool

raíz :: ÁrbolBinario e → e

izquierdo :: ÁrbolBinario e → ÁrbolBinario e

derecho :: ÁrbolBinario e → ÁrbolBinario e

- - las tres últimas se aplican sólo a árbol no vacío

es_vacío <> = True

es_vacío <i,r,d> = False

raíz <i,r,d> = r

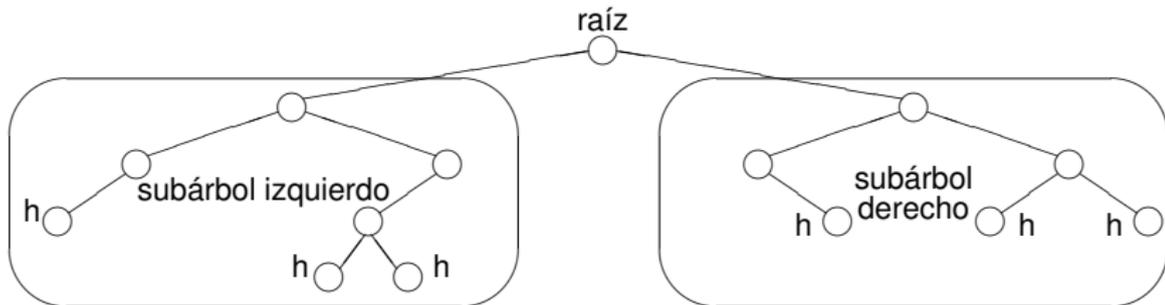
izquierdo <i,r,d> = i

derecho <i,r,d> = d

Notación $\langle \rangle$

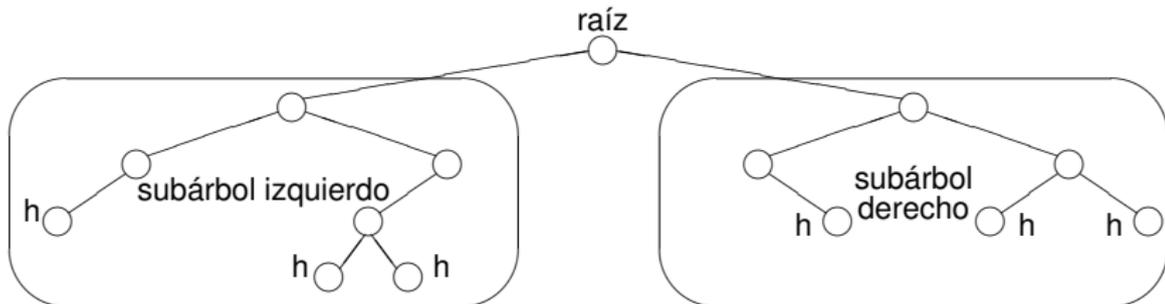
- Notar la sobrecarga de la notación $\langle \rangle$:
- cuando no tiene ningún argumento, $\langle \rangle$ es el árbol vacío,
- cuando tiene tres argumentos, $\langle i,r,d \rangle$ es el árbol no vacío cuya raíz es r , subárbol izquierdo es i y subárbol derecho es d .
- Una hoja es un árbol de la forma $\langle \langle \rangle, r, \langle \rangle \rangle$. Se la abrevia $\langle r \rangle$.
- Conclusión: la notación $\langle \rangle$ puede tener 0, 1 ó 3 argumentos.

Botánica y genealogía



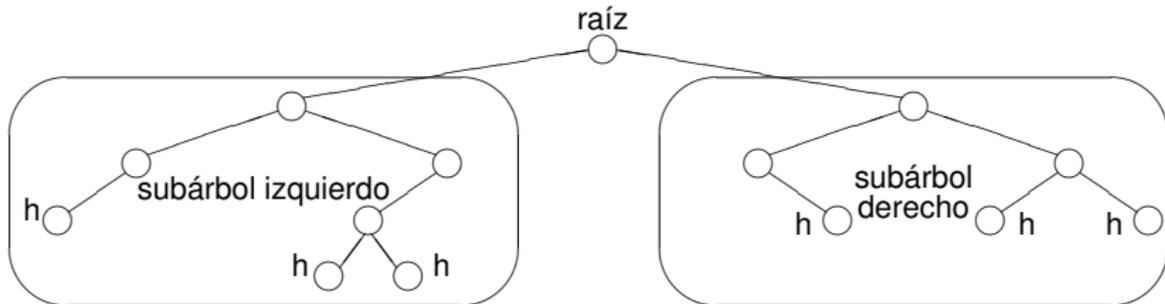
- Un **nodo** es un árbol no vacío.
- Tiene **raíz**, **subárbol izquierdo** y **subárbol derecho**.
- A los subárboles se los llama también **hijos** (izquierdo y derecho).
- Y al nodo se le dice **padre** de sus hijos.
- Una **hoja** es un nodo con los dos hijos vacíos.

Más genealogía



- Dos **hermanos** son hijo izquierdo y derecho del mismo padre.
- Un **camino** es una secuencia A_1, \dots, A_n donde cada árbol es hijo del anterior.
- A es **ancestro** de B si hay un camino (de longitud ≥ 0) de A a B ,
- en ese caso B es **descendiente** o **subárbol** de A
- un camino se puede identificar con un recorrido descendente del árbol.

Altura, profundidad, nivel



- Decimos que siempre hay un camino de longitud 0 de un árbol a sí mismo.
- La **altura** de un árbol es la longitud del camino que va desde él hasta el subárbol vacío más lejano. Según otras definiciones, altura es la longitud del camino que va desde él hasta la hoja más lejana.
- La **profundidad** de un subárbol es la longitud del camino que va desde el árbol hasta dicho subárbol.
- Se llama **nivel** al conjunto de los subárboles de igual profundidad.

Implementación con punteros

```
type node = tuple
    lft: pointer to node
    value: elem
    rgt: pointer to node
end
type bintree = pointer to node

fun empty() ret t:bintree
    t := null
end
{Post: t ~<> }
```

Implementación con punteros

```
{Pre: l ~ L ∧ e ~ E ∧ r ~ R}
fun node(l:bintree,e:elem,r:bintree) ret t:bintree
    alloc(t)
    t→lft:= l
    t→value:= e
    t→rgt:= r
{Post: t ~ <L,E,R>} end

{Pre: t ~ T ∧ ¬ is_empty(t)}
fun root(t:bintree) ret e:elem
    e:= t→value
end
{Post: e ~ raíz(T)}
```

Implementación con punteros

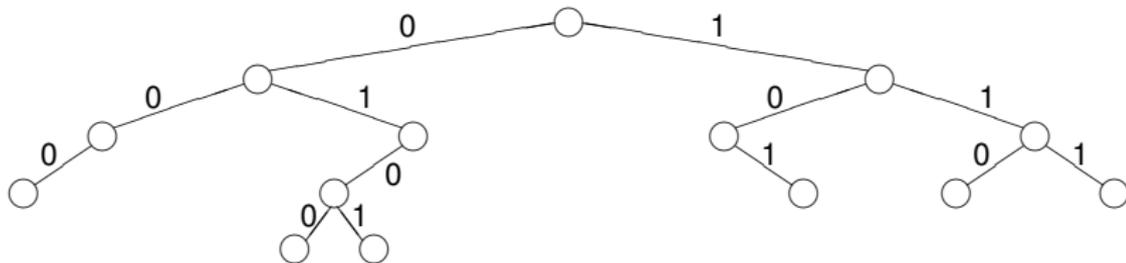
```
{Pre: t ~ T ∧ ¬ is_empty(t)}  
fun left(t:bintree) ret l:bintree  
    l := t → lft  
end  
{Post: l ~ izquierdo(T)}
```

```
{Pre: t ~ T ∧ ¬ is_empty(t)}  
fun right(t:bintree) ret r:bintree  
    r := t → rgt  
end  
{Post: r ~ derecho(T)}
```

Implementación con punteros

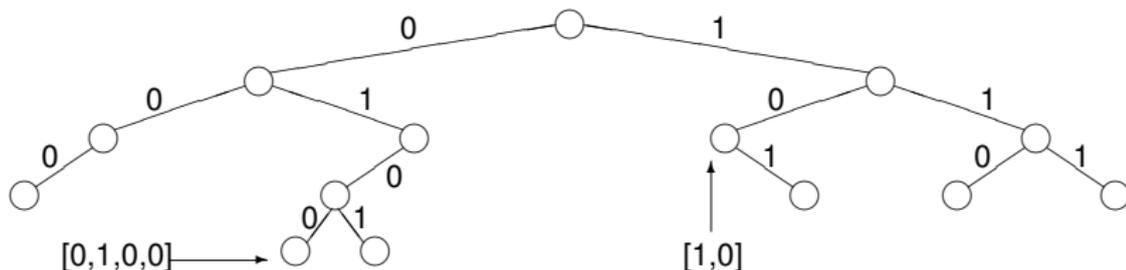
```
{Pre:  $t \sim T$ }  
fun is_empty(t:bintree) ret b:bool  
    b:= (t = null)  
end  
{Post:  $b \sim \text{es\_vacío}(T)$ }  
proc destroy(in/out t:bintree)  
    if  $\neg$  is_empty(t) then destroy(left(t))  
        destroy(right(t))  
        free(t)  
        t:= null  
    fi  
end
```

Indicaciones



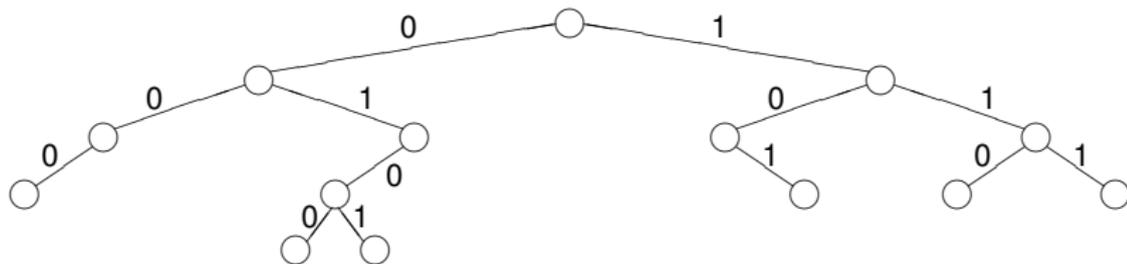
- A cada arista que conecta un padre con su hijo se la rotula 0 si es con el hijo izquierdo y 1 si es el derecho,
- Este 0 ó 1 puede entenderse como dando **indicaciones**
- 0 es ir a la izquierda
- 1 es ir a la derecha

Posiciones



- Una lista de 0's y 1's sirve para desplazarse desde la raíz hacia las hojas.
- Cada subárbol queda señalado por una lista de 0's y 1's.
- Estas listas de 0's y 1's marcan **posiciones** dentro del árbol.
- Definimos $pos = [\{0, 1\}]$.
- Es el conjunto de todas las posiciones.

Selección de subárbol



Dado un árbol t y una posición $p \in pos$, $t \downarrow p$ es el subárbol de t que se encuentra en la posición p :

$$\langle \rangle \downarrow p = \langle \rangle$$

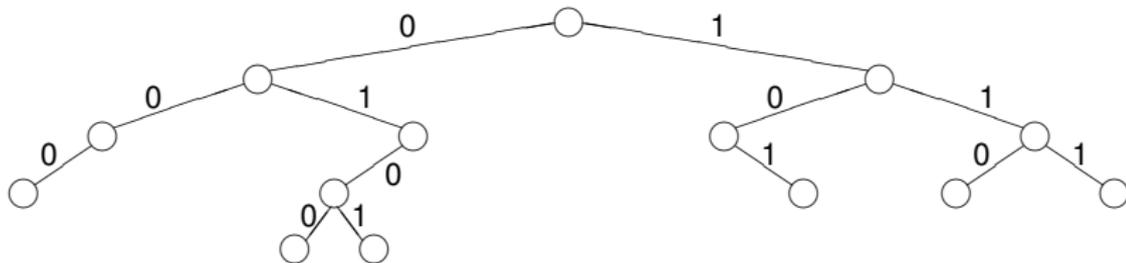
$$\langle i, e, d \rangle \downarrow [] = \langle i, e, d \rangle$$

$$\langle i, e, d \rangle \downarrow (0 \triangleright p) = i \downarrow p$$

$$\langle i, e, d \rangle \downarrow (1 \triangleright p) = d \downarrow p$$

Se define $pos(t) = \{p \in pos \mid t \downarrow p \neq \langle \rangle\}$. Es el conjunto de las posiciones del árbol binario t .

Selección de elemento



Dado un árbol t y una posición $p \in pos(t)$, $t.p$ es el elemento de t que se encuentra en la posición p :

$$\langle i, e, d \rangle . [] = e$$

$$\langle i, e, d \rangle . (0 \triangleright p) = i.p$$

$$\langle i, e, d \rangle . (1 \triangleright p) = d.p$$

o equivalentemente $t.p = raiz(t \downarrow p)$.