

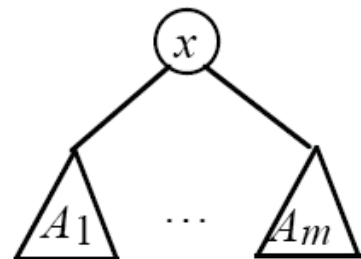
Árboles n -arios

Lección 15

Conceptos, definiciones y terminología básica

- **Árbol:**

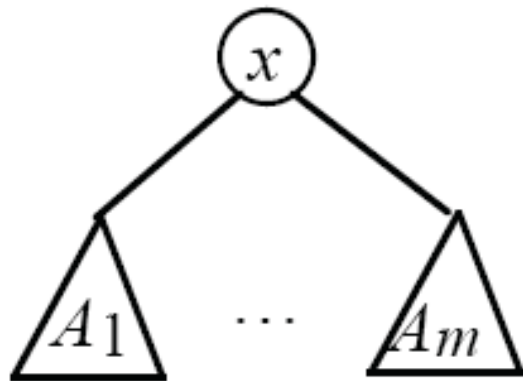
- Conjunto de elementos de un mismo tipo, denominados nodos, que pueden representarse en un grafo no orientado, conexo y acíclico, en el que existe un vértice destacado denominado raíz
 - Por lo general es una estructura jerárquica...
- Definición recursiva:
 - Un **árbol n -ario** (con $n \geq 1$) es un conjunto no vacío de elementos del mismo tipo tal que:
 - Existe un elemento destacado llamado **raíz** del árbol
 - el resto de los elementos se distribuyen en m subconjuntos disjuntos ($0 \leq m \leq n$), llamados **subárboles** del árbol original, cada uno de los cuales es a su vez un árbol n -ario



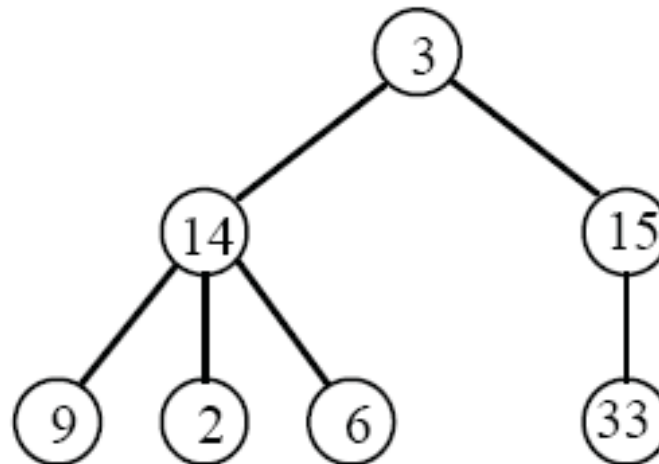
Conceptos, definiciones y terminología básica

- **Árbol ordenado:**

- Si en el conjunto de subárboles de un árbol n -ario se supone definida una relación de orden total, el árbol se denomina ordenado

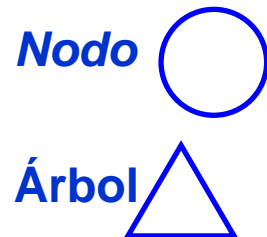


Árbol ordenado con raíz X y subárboles $A_1 \dots A_m$



Árbol 3-ario de números enteros

Leyenda:



Conceptos, definiciones y terminología básica

- **Bosque:**

- Un bosque ordenado de grado n (con $n \geq 1$) es una secuencia A_1, \dots, A_m , con $0 \leq m \leq n$, de árboles n -arios ordenados.
 - Si $m=0$, el bosque se llama vacío
- Un **árbol n -ario** se genera a partir de un elemento y un bosque ordenado de grado n , bastando considerar el elemento como raíz del árbol, y el bosque como sus subárboles
- Los **bosques** se generarán como secuencias de árboles, a partir de un bosque vacío, y una operación de añadir por la derecha un árbol a un bosque

Especificación

espec árbolesOrdenados

usa booleanos, naturales

parámetro formal

género elmto

fpf

géneros bosque, árbol

operaciones

procedimiento crearVacío(**sal** b:bosque)

{devuelve el bosque vacío, es decir, la secuencia vacía de árboles}

procedimiento añadirÚltimo(**e/s** b:bosque; **ent** a:árbol)

{devuelve el bosque resultante de añadir el árbol a como último elemento de b}

función long(b:bosque) **devuelve** natural

{devuelve el nº de árboles del bosque b, es decir, la longitud de la secuencia}

procedimiento observar(**ent** b:bosque; **ent** n:natural; **sal** a:árbol)

{devuelve el n-ésimo árbol de la secuencia de árboles b;

es una operación parcial: si $n > \text{long}(b) \rightarrow \text{error}$ }

procedimiento resto(**ent** b:bosque; **sal** rb:bosque)

{devuelve en rb el bosque resultante de borrar el primer árbol de b;

es una operación parcial: si $\text{long}(b)=0 \rightarrow \text{error}$ }

procedimiento plantar(**sal** a:árbol; **ent** e:elmto; **ent** b:bosque)

{devuelve un árbol a cuya raíz es e y sus subárboles son el bosque b}

...

Especificación

...

función raíz(a:árbol) **devuelve** elmto

{devuelve la raíz del árbol a}

procedimiento elBosque(**ent** a:árbol; **sal** b:bosque)

{devuelve el bosque de los subárboles del árbol a}

función numHijos(a:árbol) **devuelve** natural

{devuelve el nº de subárboles de a, es decir long(elBosque(a) }

procedimiento subÁrbol(**ent** a:árbol; **ent** n:natural; **sal** sa:árbol)

{devuelve en sa el n-ésimo subárbol de a; **es parcial: si $n > \text{numHijos}(a) \rightarrow \text{error}$** }

función esHoja?(a:árbol) **devuelve** booleano

{devuelve verdad si y sólo si a no tiene subárboles, es decir, numHijos(a)=0 }

función alturaBosque(b:bosque) **devuelve** natural

{devuelve la altura del árbol más alto de b}

función alturaÁrbol(a:árbol) **devuelve** natural

{devuelve la altura de a}

fespec

Recordar recorridos

Un recorrido de un árbol consiste en visitar todos los elementos del árbol una sola vez.

- **Recorridos en profundidad:**

- Recorrido en **pre-orden**:

1. se visita la raíz

2. se recorren en pre-orden todos los subárboles, de izquierda a derecha

- Recorrido en **post-orden**:

1. se recorren en post-orden todos los subárboles, de izquierda a derecha

2. se visita la raíz

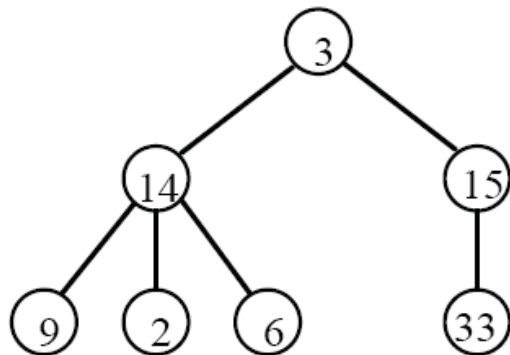
- **El recorrido en anchura:**

- primero se visitan los elementos del nivel 0, luego los del nivel 1, y así sucesivamente,

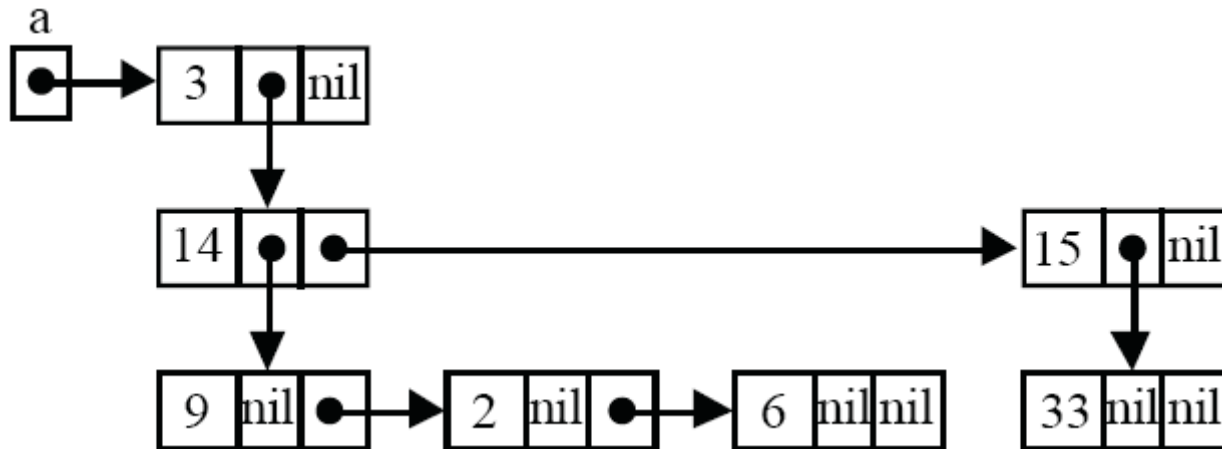
- En cada nivel, se visitan los elementos de izquierda a derecha

Implementación Dinámica

- Representación “primogénito - siguiente hermano” (*child-brother*)



```
tipos árbol = ↑nodo;  
nodo = registro  
      dato:elemento;  
      primogénito,sigHermano:árbol  
freg;  
bosque = árbol
```



Implementación “primogénito - sig. hermano”

modulo árbolesOrdenados

importa defTipoElemento

exporta

tipos árbol,bosque

procedimiento crearVacío(**sal** b:bosque)

{devuelve el bosque vacío, es decir, la secuencia vacía de árboles}

procedimiento añadirÚltimo(**e/s** b:bosque; **ent** a:árbol)

{dev. el bosque resultante de añadir el árbol a como último elemento de b}

función long(b:bosque) **devuelve** natural

{devuelve el nº de árboles del bosque b, es decir, la longitud de la secuencia}

procedimiento observar(**ent** b:bosque; **ent** n:natural;
sal error:booleano; **sal** a:árbol)

{si $1 \leq n \leq \text{long}(b)$: devuelve el n-ésimo árbol de la secuencia de árboles b y error=falso;
en caso contrario devuelve error=verdad}

procedimiento resto(**ent** b:bosque; **sal error:booleano**; **sal** rb:bosque)

{si b no es vacío devuelve en rb el bosque resultante de borrar el primer árbol de b y
error=falso; en caso contrario devuelve error=verdad}

...

Implementación “primogénito - sig. hermano”

...

procedimiento plantar(**sal** a:árbol; **ent** e:elemento; **ent** b:bosque)

{devuelve un árbol a cuya raíz es e y sus subárboles son el bosque b}

función raíz(a:árbol) **devuelve** elemento

{devuelve la raíz del árbol a}

procedimiento elBosque(**ent** a:árbol; **sal** b:bosque)

{devuelve el bosque de los subárboles del árbol a}

función numHijos(a:árbol) **devuelve** natural

{devuelve el nº de subárboles de a, es decir long(elBosque(a) }

procedimiento subÁrbol(**ent** a:árbol; **ent** n:natural;

sal error:booleano; **sal** sa:árbol)

{si $1 \leq n \leq \text{numHijos}(a)$: devuelve el n-ésimo subárbol de a y error=falso;
en caso contrario error=verdad}

función esHoja(a:árbol) **devuelve** booleano

{dev. verdad si y sólo si a no tiene subárboles, es decir, numHijos(a)=0 }

función alturaBosque(b:bosque) **devuelve** natural

{devuelve la altura del árbol más alto de b}

función alturaÁrbol(a:árbol) **devuelve** natural

{devuelve la altura de a}

...

Implementación “primogénito - sig. hermano”

...

procedimiento duplicarBosque(**sal** nuevo:bosque; **ent** viejo:bosque)
{Duplica la representación del bosque viejo guardándolo en nuevo}

procedimiento liberarBosque(**e/s** b:bosque)
{Libera la memoria dinámica accesible desde b, quedando b vacío}

procedimiento duplicarÁrbol(**sal** nuevo:árbol; **ent** viejo:árbol)
{Duplica la representación del árbol viejo guardándolo en nuevo}

procedimiento liberarÁrbol(**e/s** a:árbol)
{Libera la memoria dinámica accesible desde a}

implementación

tipos

árbol = ↑nodo;
nodo = **registro**
 dato:elemento;
 primogénito,sigHermano:árbol
 freg;
bosque = árbol

...

Implementación “primogénito - sig. hermano”

...

procedimiento crearVacío(**sal** b:bosque)

principio

b:=nil

fin

procedimiento añadirÚltimo(**e/s** b:bosque; **ent** a:árbol)

variable aux:bosque

principio

si b=nil **entonces**

b:=a

sino

aux:=b;

mientrasQue aux↑.sigHermano≠nil **hacer**

aux:=aux↑.sigHermano

fmq;

aux↑.sigHermano:=a

fsi

fin

...

Implementación “primogénito - sig. hermano”

...

función long(b:bosque) **devuelve** natural

principio

si b=nil **entonces**

devuelve 0

sino

devuelve 1+long(b↑.sigHermano)

fsi

fin

procedimiento observar(**ent** b:bosque; **ent** n:natural; **sal** error:booleano; **sal** a:árbol)

principio

si n=0 or b=nil **entonces** error:=verdad

sino

si n=1 **entonces**

 a:=b; error:=falso

sino

 observar(b↑.sigHermano,n-1,error,a)

fsi

fsi

fin

...

Implementación “primogénito - sig. hermano”

```
...  
procedimiento resto(ent b:bosque; sal error:booleano; sal rb:bosque)  
principio  
    si b=nil entonces error:=verdad sino error:=falso; rb:=b↑.sigHermano fsi  
fin
```

```
procedimiento plantar(sal a:árbol; ent e:elemento; ent b:bosque)  
principio  
    nuevoDato(a);  
    a↑.dato:=e;  
    a↑.primogénito:=b;  
    a↑.sigHermano:=nil  
fin
```

```
función raíz(a:árbol) devuelve elemento  
principio  
    devuelve a↑.dato  
fin
```

```
...
```

Implementación “primogénito - sig. hermano”

```
...
procedimiento elBosque(ent a:árbol; sal b:bosque)
principio
    b:=a↑.primogénito
fin
procedimiento subárbol(ent a:árbol; ent n:natural; sal error:booleano; sal sa:árbol)
variable b:bosque
principio
    elBosque(a,b); observar(b,n,error,sa)
fin
función numHijos(a:árbol) devuelve natural
variable b:bosque
principio
    elBosque(a,b); devuelve long(b)
fin
función esHoja(a:árbol) devuelve booleano
principio
    devuelve a↑.primogénito=nil
fin
...
```

Implementación “primogénito - sig. hermano”

...

función alturaBosque(b:bosque) **devuelve** natural

principio

si b=nil **entonces**

devuelve 0

sino

devuelve max(alturaÁrbol(b),alturaBosque(b↑.sigHermano))

fsi

fin

función alturaÁrbol(a:árbol) **devuelve** natural

variable b:bosque

principio

si esHoja(a) **entonces**

devuelve 0

sino

 elBosque(a,b); **devuelve** 1+alturaBosque(b)

fsi

fin

...

Implementación “primogénito - sig. hermano”

...

procedimiento duplicarBosque(**sal** nuevo:bosque; **ent** viejo:bosque)

variables primerÁrbol:árbol; error:booleano

principio

si viejo=nil **entonces** nuevo:=nil

sino

 observar(viejo,1,error,primerÁrbol);

 duplicarÁrbol(nuevo,primerÁrbol);

 duplicarBosque(nuevo↑.sigHermano,viejo↑.sigHermano)

fsi

fin

procedimiento duplicarÁrbol (**sal** nuevo:árbol; **ent** viejo:árbol)

variables viejoBosque,nuevoBosque:bosque

principio

 elBosque(viejo,viejoBosq);

 duplicarBosque(nuevoBosque,viejoBosque);

 plantar(raíz(viejo),nuevoBosque,nuevo)

fin

...

Implementación “primogénito - sig. hermano”

...

procedimiento liberarBosque(**e/s** b:bosque)

principio

si $b \neq \text{nil}$ **entonces**

 liberarBosque($b \uparrow . \text{sigHermano}$);

 liberarÁrbol(b)

fsi

fin

procedimiento liberarÁrbol(**e/s** a:árbol)

principio

 liberarBosque($a \uparrow . \text{primogénito}$);

 disponer(a)

fin

fin {del modulo árbolesOrdenados}

Implementación “primogénito - sig. hermano”

Coste en tiempo de las operaciones:

- *crearVacío, resto, plantar, raíz, elBosque y esHoja* $\rightarrow \Theta(1)$
- Modificando la representación, puede lograrse que las operaciones *añadirÚltimo, long, alturaBosque, numHijos* y *alturaArbol* tengan también coste $\rightarrow \Theta(1)$
- *observar y subárbol* $\rightarrow \Theta(n)$ (n es el grado del árbol)
- *duplicarBosque, duplicarÁrbol, liberarBosque, liberarÁrbol* $\rightarrow \Theta(N^\circ \text{ de nodos del bosque/del árbol})$

Implementación “primogénito - sig. hermano”

módulo recorridosÁrboles

importa árbolesOrdenados,listas

exporta

procedimiento preOrden(**ent** a:árbol; **sal** l:listas)

{añade a la lista l los elementos de a recorridos en pre-orden}

procedimiento postOrden(**ent** a:árbol; **sal** l:listas)

{añade a la lista l los elementos de a recorridos en post-orden}

implementación

...

Implementación “primogénito - sig. hermano”

...

procedimiento preBosque(**ent** b:bosque; **sal** l:lista)

{añade a la lista l los elmntos. de todos los árboles del bosque b recorridos en pre-orden}

variable primerÁrbol:árbol; rb:bosque; error:booleano

principio

si long(b)≠0 **entonces**

 observar(b, 1, error, primerÁrbol);

 preOrden(primerÁrbol, l);

 resto(b, error, rb);

 preBosque(rb, l)

fsi

fin

procedimiento preOrden(**ent** a:árbol; **sal** l:lista)

variable b:bosque

principio

 añadirÚltimo(l, raíz(a));

 elBosque(a, b);

 preBosque(b, l)

fin

...

Implementación “primogénito - sig. hermano”

...

procedimiento postBosque(**ent** b:bosque; **sal** l:lista)

{añade a la lista l los elmntos. de todos los árboles del bosque b recorridos en post-orden}

variable primerÁrbol:árbol; rb:bosque; error:booleano

principio

si long(b)≠0 **entonces**

 observar(b,1,error,primerÁrbol);

 postOrden(primerÁrbol,l);

 resto(b,error,rb);

 postBosque(rb,l)

fsi

fin

procedimiento postOrden(**ent** a:árbol; **sal** l:lista)

variable b:bosque

principio

 elBosque(a,b);

 postBosque(b,l);

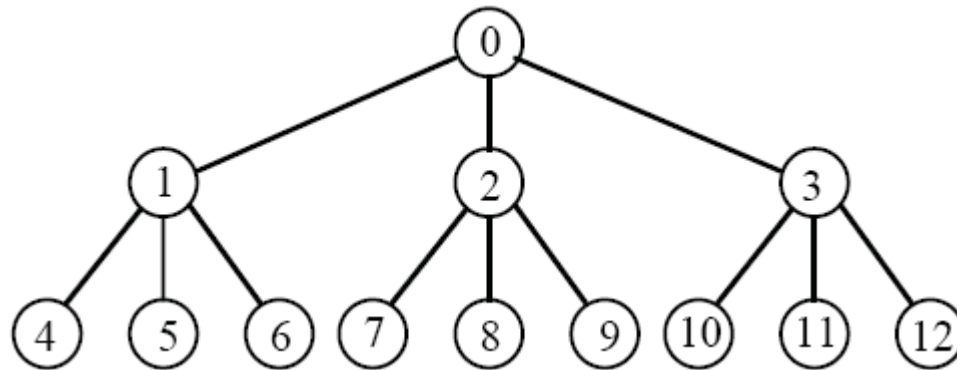
 añadirÚltimo(l,raíz(a))

fin

fin {del modulo}

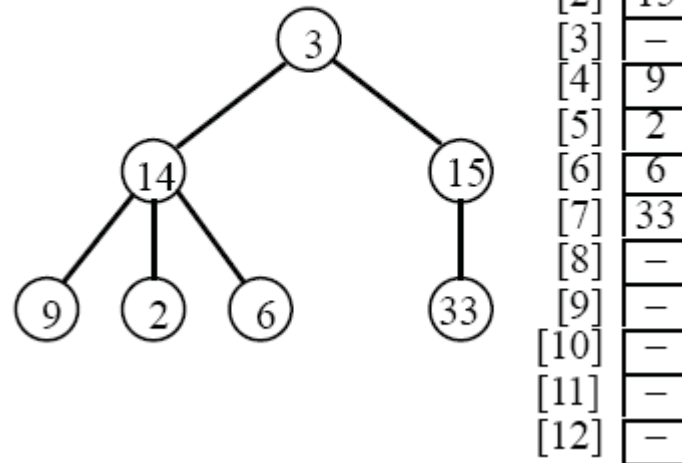
Definiciones

- Un **árbol** n -ario se dice **homogéneo** si todos sus subárboles excepto las hojas tienen n hijos.
- Un **árbol** homogéneo es **completo** cuando todas sus hojas tienen la misma profundidad
- Un **árbol** se dice **casi-completo** cuando se puede obtener a partir de un árbol completo eliminando hojas consecutivas del último nivel, comenzando por la que está más a la derecha



Otras implementaciones

- Implementación estática:



<i>elemento</i>	<i>índice</i>	<i>condición</i>
e	i	
hijo k -ésimo de e	$n*i+k$	si $n*i+k < \max$
padre de e	$(i-1) \text{ div } n$	si $i \neq 0$
hermano siguiente a e	$i+1$	si $i \bmod n \neq 0$
n.º de orden entre sus hermanos	$((i-1) \bmod n)+1$	si $i \neq 0$