ESTRUCTURAS DE DATOS:

ÁRBOLES GENERALES

Profesora: Mª José Domínguez Alda

ÁRBOLES GENERALES

Un árbol A n-ario, con $n \geq 1$, es un conjunto de elementos del mismo tipo tal que:

- existe un elemento llamado raíz;
- el resto de elementos se distribuyen en m subconjuntos disjuntos, con $0 \leq m \leq n$, cada uno de los cuales es un árbol n-ario, llamados subárboles del árbol original.

Cuando el orden importa, se dice que el árbol está ordenado.

¡Importante!: Un árbol general no puede estar vacío

BOSQUES

- Un bosque de grado $n \geq 1$ es una secuencia $A_1, \ldots, A_m$, con $0 \leq m \leq n$, de árboles n-arios
- Cuando $m = 0$, el bosque se denomina vacío.

Gracias a la definición de bosque, *un árbol general puede entenderse como un elemento con un bosque*.

MÁS TERMINOLOGÍA

Nivel: conjunto de nodos que están en la misma profundidad.

Grado de un árbol: número máximo de hijos que puede tener un árbol.

Árbol homogéneo: aquel cuyos subárboles (excepto las hojas) tienen todos n hijos, siendo n el grado del árbol.

Árbol completo: un árbol homogéneo es completo si todas sus hojas tienen la misma profundidad.

Árbol casi completo: un árbol es casi completo cuando se puede obtener a partir de un árbol completo, eliminando cero o más hojas consecutivas del último nivel, comenzando por la hoja más a la derecha.

<u>ESPECIFICACIÓN: ÁRBOLES</u>

espec *ÁRBOLES[ELEMENTO]*

usa *NATURALES2, BOOLEANOS*

parametro formal

  generos *elemento*

fparametro

generos *árbol*, *bosque*

ESPECIFICACIÓN: ÁRBOLES (2)

operaciones

_ • _ : elemento bosque → árbol	*{crea árboles generales}*
[ ] : → bosque	*{bosque vacío}*
_:_ : árbol bosque → bosque	*{crea bosques de árboles}*
raíz : árbol → elemento	*{raíz del árbol; siempre existe}*
hijos : árbol → bosque	*{bosque de hijos del árbol; puede ser vacío}*
vacío? : bosque → bool	*{mira si un bosque está vacío}*
long: bosque → natural	*{tamaño del bosque}*
num_hijos: árbol → natural	*{cantidad de hijos}*

<u>ESPECIFICACIÓN: ÁRBOLES (3)</u>

operaciones

parcial *primero : bosque → árbol*
{devuelve el primer árbol del bosque}

parcial *resto: bosque → bosque*
{devuelve el bosque sin el primer árbol}

parcial *subárbol: árbol natural → árbol*
{acceso al i-ésimo hijo de un árbol}

var

x: elemento

a: árbol; b: bosque

n: natural

ESPECIFICACIÓN: ÁRBOLES (4)

ecuaciones de definitud *{escrita informal por claridad}*

vacio?(b) = F ⇒ Def(primero(b))

vacio?(b) = F ⇒ Def(resto(b))

1 ≤ n ≤ num_hijos(a) ⇒ Def(subárbol(a,n))

ecuaciones

raíz(x•b) = x

hijos(x•b) = b

vacío?([]) = T

vacío?(a:b) = F

vacío?(b) = T ⇒ long(b) = 0

vacío?(b) = F ⇒ long(b) = suc(long(resto(b)))

ESPECIFICACIÓN: ÁRBOLES (y 5)

ecuaciones

$$num\_hijos(x \bullet b) = long(b)$$

$$primero(a:b) = a$$

$$resto(a:b) = b$$

$$subárbol(x \bullet b, 1) = primero(b)$$

$$(1 < n) \wedge (n \leq long(b)) \Rightarrow subárbol(x \bullet b, n) = subárbol(x \bullet resto(b), pred(n))$$

fespec

EJEMPLO 6

Ejemplo: Obtener la suma de todos los nodos de un árbol general de naturales, suponiendo que el bosque vacío tiene valor 0.

- Tenemos que hacer dos operaciones:

  ***suma**: árbol → natural*

  ***sumabosque**: bosque → natural*

- Declaramos las variables...

  x: natural; a: árbol; b: bosque

- Las ecuaciones de las dos operaciones pueden quedar…

  suma(x•b) = x + sumabosque(b)

  sumabosque([]) = 0

  sumabosque(a:b) = suma(a) + sumabosque(b)

EJEMPLO 6. PSEUDOCÓDIGO

Ejemplo: Obtener la suma de todos los nodos de un árbol general de naturales, suponiendo que el bosque vacío tiene valor 0.

```
func suma_árbol(a:árbol) dev s:natural
   s ← raíz(a) + suma_bosque(hijos(a))
finfunc

func suma_bosque(b:bosque) dev s:natural
   si vacío?(b) entonces s ← 0
   si no   s ← suma_árbol(primero(b))
                + suma_bosque(resto(b))
   finsi
finfunc
```

EJEMPLO 7

Ejemplo: Determinar si en un árbol general de naturales hay algún número que sea par.

- Como siempre con árboles generales, dos operaciones:

  hay_par?*: *árbol* → *bool

  hay_par_b?*: *bosque* → *bool

- Las ecuaciones son muy parecidas al ejemplo anterior:

  hay_par?(x•b) = es_par?(x) ∨ hay_par_b?(b)

  hay_par_b?([]) = F

  hay_par_b?(a:b) =

  hay_par?(a) ∨ hay_par_b?(b)

EJEMPLO 7. PSEUDOCÓDIGO

Ejemplo: Determinar si en un árbol general de naturales hay algún número que sea par.

```
func  hay_par?(a:árbol) dev hay:bool
   hay ← es_par?(raíz(a)) ∨ hay_par_b?(hijos(a))
finfunc

func  hay_par _b?(b:bosque) dev hay:bool
var   prim:arbol
   si vacio?(b) entonces hay ← F
   si no
     hay ← hay_par?(primero(b)) ∨
                    hay_par_b?(resto(b))
   fin_si
finfunc
```

EJEMPLO 8

Ejemplo: Contar cuántos pares tiene un árbol general de naturales

- La operación es total:

  cuantos_pares*: *árbol* → *natural

  cuantos_pares_b*: *bosque*→ *natural

- Las ecuaciones deben comprobar si la raíz es par o no:

```
es_par?(x)=F ⇒ cuantos_pares(x•b) =
                              cuantos_pares_b(b)
es_par?(x)=T ⇒ cuantos_pares(x•b) =
                         suc(cuantos_pares_b(b))
cuantos_pares_b([]) = 0
cuantos_pares_b(a:b) =
      cuantos_pares(a) + cuantos_pares_b(b)
```

EJEMPLO 8. PSEUDOCÓDIGO

Ejemplo: Contar cuántos pares tiene un árbol general de naturales

```
func  cuantos_pares(a:árbol) dev s:natural
   si  es_par?(raíz(a)) entonces
          s ← 1 + cuantos_pares_b(bosque(a))
   si no    s ← cuantos_pares_b(bosque(a))
   finsi
finfunc
```

```
func  cuantos_pares_b (b:bosque) dev s:natural
   si vacio?(b) entonces s ← 0
   si no    s ← cuantos_pares(primero(b)) +
                  cuantos_pares_b(resto(b))
   finsi
finfunc
```

EJEMPLO 9

Ejemplo: Comprobar si dos árboles generales tienen la misma forma (no es necesario que los datos tengan el mismo valor).

- Hacemos dos operaciones, una de árbol y otra de bosque:

  igual_forma**: **árbol árbol → bool

  igual_forma_b**: **bosque bosque → bool

- Solo hay que comprobar cómo son los bosques:

$$\textit{igual\_forma}(x \bullet b_1,\ y \bullet b_2) = \textit{igual\_forma\_b}(b_1,\ b_2)$$

$$\textit{igual\_forma\_b}([\,],\ [\,]) = T$$

$$\textit{igual\_forma\_b}(a_1{:}b_1,\ [\,]) = F$$

$$\textit{igual\_forma\_b}([\,],\ a_2{:}b_2) = F$$

$$\textit{igual\_forma\_b}(a_1{:}b_1,\ a_2{:}b_2) = \textit{igual\_forma}(a1,\ a2) \wedge \textit{igual\_forma\_b}(b_1,\ b_2)$$

EJEMPLO 9-PSEUDOCÓDIGO

```
func igual_forma(a1, a2: árbol) dev b:bool
   b ← igual_forma_b(bosque(a1), bosque(a2))
finfunc

func igual_forma_b(b1, b2: bosque)  dev b:bool
   si vacio?(b1) ≠ vacio?(b2) entonces b ← F
   si no
      si vacio?(b1) entonces b ← T
      si no
        b ← igual_forma(primero(b1), primero(b2))
                ∧
             igual_forma_b(resto(b1), resto(b2))
   finsi
finfunc
```

<u>ESPECIFICACIÓN: ÁRBOLES+</u>

espec *ÁRBOLES+[ELEMENTO]*

usa *ÁRBOLES[ELEMENTO], LISTA2[ELEMENTO], BOOLEANOS*

operaciones

preorden : árbol $\rightarrow$ lista	
prebosque : bosque $\rightarrow$ lista	*{auxiliar para preorden}*
postorden: árbol $\rightarrow$ lista	
postbosque: bosque $\rightarrow$ lista	*{auxiliar para postorden}*
hoja? : árbol $\rightarrow$ bool	*{ver si un árbol tiene hijos}*
altura_árbol: árbol $\rightarrow$ natural	*{altura de un árbol}*
altura_bosque: bosque $\rightarrow$ natural	*{altura máxima del bosque}*

<u>ESPECIFICACIÓN: ÁRBOLES+ (2)</u>

var

x: elemento; a: árbol; b: bosque

ecuaciones

```
preorden(x•b) = x:prebosque(b)
prebosque([]) = []
prebosque(a:b) = preorden(a) ++ prebosque(b)

postorden(x•b) = x#postbosque(b)
postbosque([]) = []
postbosque(a:b) = postorden(a)++postbosque(b)
```

ESPECIFICACIÓN: ÁRBOLES+ (y 3)

```
hoja?(a) = (num_hijos(a)==0)

altura_árbol(x•[]) = 0
altura_árbol(x•a:b) = suc(altura_bosque(a:b))

altura_bosque([]) = 0
altura_bosque(a:b) =
        max(altura_árbol(a), altura_bosque(b))
```

fespec

OPERACIÓN PREORDEN. PSEUDOCÓDIGO

```
func preorden (a: árbol) dev l:lista
   l ← raíz(a):prebosque(hijos(a))
finfunc

func prebosque(b:bosque) dev l:lista
   si vacio?(b) entonces l ← []
   si no
     l ← preorden(primero(b)) ++
              prebosque(resto(b))
   finsi
finfunc
```

OPERACIÓN POSTORDEN. PSEUDOCÓDIGO

```
func postorden (a: árbol) dev l:lista
   l ← raíz(a)#postbosque(hijos(a))
finfunc

func postbosque(b: bosque) dev l:lista
   si vacio?(b) entonces l ← []
   si no
     l ← postorden(primero(b)) ++
              postbosque(resto(b))
   finsi
finfunc
```

OPERACIÓN ALTURA. PSEUDOCÓDIGO

```
func altura(a: árbol) dev n:natural
var b:bosque
   b←bosque(a)
   si vacio?(b) entonces n ← 0
   si no n ← 1+altura_b(b)
   finsi
finfunc

func altura_b(b:bosque) dev n:natural
   si vacio?(b) entonces n←0
   si no
      n ← máximo(altura(primero(b)),
                  altura_b(resto(b)))
   finsi
finfunc
```

IMPLEMENTACIÓN DE ÁRBOLES GENERALES

- Una implementación habitual es representar el árbol como un par formado por un elemento y una lista de árboles (bosque) implementada con memoria dinámica.
- Una segunda representación es utilizar un registro con tres campos:
    - Un elemento.
    - Enlace a primer hijo.
    - Enlace a árbol siguiente (siguiente hermano).

En esta representación puede utilizarse el mismo tipo de nodo para árboles que para bosques (listas de árboles) o incorporar el campo longitud en el bosque.

ÁRBOLES GENERALES. TIPOS

```
tipos
   nodo_árbol = reg
      valor: elemento
      primog:bosque
      sig-herm: árbol
   freg
   árbol = puntero a nodo_ árbol
   bosque = reg
         longitud:natural
         inicio: árbol
         freg
ftipos
```

ÁRBOLES GENERALES. CONSTRUCTORAS

{Crear un bosque vacío}

```
func  bosque-vacío () dev b:bosque
   b.inicio←nil
   b.longitud←0
finfunc
```

ÁRBOLES GENERALES. CONSTRUCTORAS

{Crear un árbol}

```
func crear-arbol (e:elemento, b:bosque)
                                          dev a:árbol
   reservar (a)
   a^.valor←e
   a^.primog←b
   a^.sig-herm←nil
finfunc
```

ÁRBOLES GENERALES. CONSTRUCTORAS

{Añadir un árbol al bosque}

```
proc añadir-árbol(a:árbol, b:bosque)
   a^.sig-herm←b.inicio
   b.inicio←a
   b.longitud←b.longitud+1
finproc
```

ÁRBOLES GENERALES. OBSERVADORAS

{Raíz del árbol}

```
func raíz (a: árbol) dev e:elemento
   si a=nil entonces
        Error('El árbol no puede ser vacío')
   sino e←a^.valor
   finsi
finfunc
```

ÁRBOLES GENERALES. OBSERVADORAS

{Hijos del árbol}

```
func hijos (a: árbol) dev b:bosque
   b←a^.primog
finfunc
```

ÁRBOLES GENERALES. OBSERVADORAS

{Longitud del árbol}

```
func longitud (b:bosque) dev n:natural
   n←b.longitud
finfunc
```

ÁRBOLES GENERALES. OBSERVADORAS

{Número de hijos del árbol}

```
func num-hijos (a:árbol) dev n:natural
   n←longitud(a^.primog)
finfunc


func vacio? (b:bosque) dev v:bool
   v←b.inicio=nil
finfunc
```

ÁRBOLES GENERALES. OBSERVADORAS

{subárbol o hijo i-ésimo}

```
func subárbol (a:árbol, i:natural)
                                        dev sa:árbol
   si (i>0)∧ (i<=longitud(bosque(b))
   entonces sa←consultar(a^.primog, i)
   sino Error('El subárbol no existe')
finfunc
```

ÁRBOLES GENERALES. OBSERVADORAS

{árbol i-ésimo de bosque}

```
func consultar (b:bosque, i:natural)
                                          dev sa:árbol
var j:natural

      sa←b.inicio
      j←1
      mientras (sa!=nil )∧ (j!=i) hacer
         j←j+1
         sa←sa^.sig-herm
      finmientras
finfunc
```