

Estructuras de Datos y Algoritmos

Árboles n -arios

LECCIÓN 15

© All wrongs reversed – bajo licencia CC-BY-NC-SA 4.0



Universidad
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2024/2025

Grado en Ingeniería Informática
UNIVERSIDAD DE ZARAGOZA

Aula 0.04, Edificio Agustín de Betancourt



Adaptadas de diapositivas de Javier Campos

Índice

Índice

Árboles n -arios

Conceptos, definiciones y terminología básica

Árbol

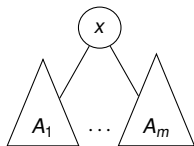
- Conjunto de elementos de un mismo tipo (denominados nodos) que pueden representarse en un **grafo no orientado, conexo y acíclico** en el que existe un vértice destacado denominado raíz
 - Por lo general, es una estructura jerárquica
- *Definición recursiva:*
 - Un árbol n -ario (con $n \geq 1$) es 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

Árboles n -arios

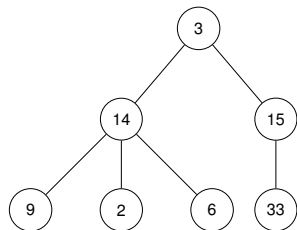
Conceptos, definiciones y terminología básica

Árbol ordenado

- \exists una relación de orden total en el conjunto de subárboles de un árbol n -ario



Árbol ordenado con raíz x y
subárboles A_1, \dots, A_m



Árbol 3-ario de números enteros

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

Índice

Especificación

espec árbolesOrdenados
usa booleanos, naturales
parámetro formal
género elmt0

fpf
géneros

bosque, árbol {Los valores del género bosque representan secuencias de elementos del género árbol.
Los valores del género árbol se componen de una raíz de género elmt0 y un bosque de árboles hijos.}

operaciones

crearVacio: -> bosque
{Devuelve el bosque vacío, es decir, la secuencia vacía de árboles}
añadirÚltimo: bosque b, árbol a -> bosque
{Devuelve un bosque igual al resultante de añadir el árbol a como último elemento de b}
long: bosque b -> natural
{Devuelve el número de árboles del bosque b, es decir, la longitud de la secuencia de árboles}
parcial observar: bosque b, natural n -> árbol
{Devuelve el n-ésimo árbol de la secuencia de árboles b.
Parcial: sólo está definida si $n \leq \text{long}(b)$ }
parcial resto: bosque b -> bosque
{Devuelve un bosque igual al resultante de borrar el primer árbol de b.
Parcial: sólo está definida si $\text{long}(b) > 0$ }
plantar: elmt0 e, bosque b -> árbol
{Devuelve un árbol cuya raíz es e y sus subárboles son un bosque igual que b}
raíz: árbol a -> elmt0
{Devuelve la raíz del árbol a}
elBosque: árbol a -> bosque
{Devuelve un bosque igual al bosque de los subárboles del árbol a}
numHijos: árbol a -> natural
{Devuelve el número de subárboles de a, es decir $\text{long}(\text{elBosque}(a))$ }
parcial subÁrbol: árbol a, natural n -> árbol
{Devuelve un árbol igual al n-ésimo subárbol de a.
Parcial: no está definida si $n > \text{numHijos}(a)$ }
esHoja?: árbol a -> booleano
{Devuelve verdad si y sólo si a no tiene subárboles, es decir, $\text{numHijos}(a) = 0$ }
alturaBosque: bosque b -> natural
{Si $\text{long}(b) > 0$, devuelve la altura del árbol más alto de b. Si $\text{long}(b) = 0$, devuelve 0}
alturaÁrbol: árbol a -> natural
{Devuelve la altura de a}

fespec

Índice

Árboles n -arios

Recorridos en profundidad

- Consiste en visitar todos los elementos del árbol una sola vez

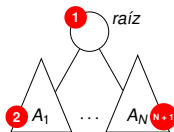
Árboles n -arios

Recorridos en profundidad

- Consiste en visitar todos los elementos del árbol una sola vez

■ Recorrido en pre-orden

- 1 Se visita la raíz
- 2 Se recorren (en pre-orden) todos los subárboles, de izquierda a derecha



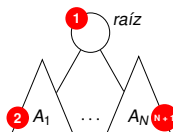
Árboles n -arios

Recorridos en profundidad

- Consiste en visitar todos los elementos del árbol una sola vez

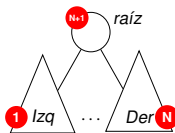
■ Recorrido en pre-orden

- 1 Se visita la raíz
- 2 Se recorren (en pre-orden) todos los subárboles, de izquierda a derecha



■ Recorrido en post-orden

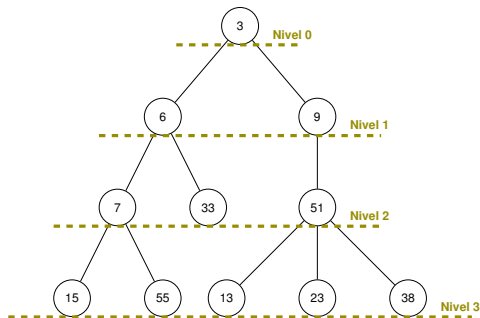
- 1 Se recorren (en post-orden) todos los subárboles, de izquierda a derecha
- 2 Se visita la raíz



Árboles n -arios

Recorrido en anchura

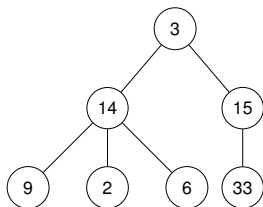
- Visitar todos los elementos del árbol una sola vez, de la forma que:
 - Primero, se visitan los elementos del nivel 0, luego los del nivel 1, y así sucesivamente;
 - En cada nivel, se visitan los elementos de izquierda a derecha



Índice

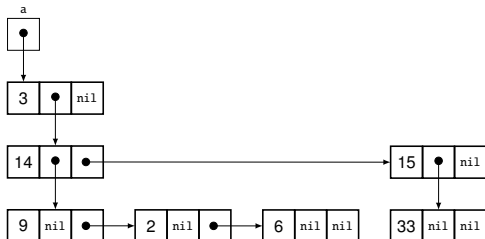
Implementación dinámica

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



tipos

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



Implementación “primogénito - siguiente hermano”

```
módulo árbolesOrdenados
importa defTipoElemento
exporta
  tipos árbol, bosque
  procedimiento crearVacio(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 número 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}
  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 n° de subárboles de a, es decir  $\text{long}(\text{elBosque}(a))$  }
  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
  {Devuelve verdad si y sólo si a no tiene subárboles, es decir,  $\text{numHijos}(a) = 0$  }
  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}
  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

procedimiento crearVacio(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

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

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

```

```

...
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

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

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
...

```

```

...
procedimiento liberarBosque(e/s b: bosque)
principio
  si b ≠ nil entonces
    liberarBosque(b↑.sigHermano);
    liberarÁrbol(b)
  fsi
fin

procedimiento liberarÁrbol(e/s a: árbol)
principio
  liberarBosque(a↑.primogénito);
  disponer(a)
fin
fin {del módulo árbolesOrdenados}

```

Coste en tiempo de las operaciones

- crearVacio, resto, plantar, raíz, elBosque y esHoja: $\Theta(1)$
- Modificando la representación, puede lograrse que añadirÚltimo, long, alturaBosque, numHijos y alturaArbol tengan también coste $\Theta(1)$
- observar y subárbol: $\Theta(n)$ (siendo n el grado del árbol)
- duplicarBosque, duplicarÁrbol, liberarBosque, liberarÁrbol: $\Theta(N)$ (siendo N el número de nodos del bosque/del árbol)

Implementación dinámica

módulo recorridosÁrboles

importa árbolesOrdenados, listas

exporta

```
procedimiento preOrden(ent a: árbol; e/s L: lista)
{añade a la lista l los elementos de a recorridos en pre-orden}
procedimiento postOrden(ent a: árbol; e/s L: lista)
{añade a la lista l los elementos de a recorridos en post-orden}
```

implementación

```
procedimiento preBosque(ent b: bosque; e/s L: lista)
{añade a la lista L los elementos de todos los árboles del bosque b recorridos en pre-orden}
```

principio

```
si b ≠ nil entonces
    preOrden(b, L);
    preBosque(b↑.sigHermano, L)
```

fsi

fin

```
procedimiento preOrden(ent a: árbol; e/s L: lista)
```

principio

```
añadirÚltimo(L, a↑.dato);
preBosque(a↑.primogénito, L)
```

fin

```
procedimiento postBosque(ent b: bosque; e/s L: lista)
```

{añade a la lista L los elementos de todos los árboles del bosque b recorridos en post-orden}

principio

```
si b ≠ nil entonces
    postOrden(b, L);
    postBosque(b↑.sigHermano, L)
```

fsi

fin

```
procedimiento postOrden(ent a: árbol; e/s L: lista)
```

principio

```
postBosque(a↑.primogénito, L);
añadirÚltimo(L, a↑.dato)
```

fin

fin {del módulo}

Índice

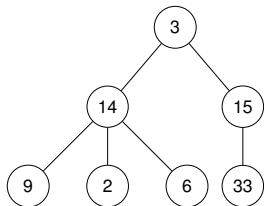
Epílogo

Otras 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

Epílogo

Otras implementaciones – implementación estática



0	3
1	14
2	15
3	-
4	9
5	2
6	6
7	33
8	-
9	-
10	-
11	-
12	-

Elemento	Índice	Condición
e	i	
hijo k -ésimo de e	$n \cdot i + k$	si $n \cdot i + k < \max$
padre de e	$(i - 1) \div n$	si $i \neq 0$
hermano siguiente a e	$i + 1$	si $i \bmod n \neq 0$
número de orden entre sus hermanos	$((i - 1) \bmod n) + 1$	si $i \neq 0$

Epílogo

Alternativa para recorridos de bosques (y árboles) n -arios

```
procedimiento preOrden(ent b: bosque; e/s L: lista)
{añade a la lista L los elementos del bosque  $n$ -ario  $b$  recorridos en pre-orden}
```

```
principio
```

```
  si  $b \neq \text{nil}$  entonces
    añadirÚltimo(L, b↑.dato);
    preOrden(b↑.primogénito, L);
    preOrden(b↑.sigHermano, L)
```

```
  fsi
```

```
fin
```

```
procedimiento postOrden(ent b: bosque; e/s L: lista)
{añade a la lista L los elementos del bosque  $n$ -ario  $b$  recorridos en post-orden}
```

```
principio
```

```
  si  $b \neq \text{nil}$  entonces
    postOrden(b↑.primogénito, L);
    añadirÚltimo(L, b↑.dato);
    postOrden(b↑.sigHermano, L)
```

```
  fsi
```

```
fin
```

Estructuras de Datos y Algoritmos

Árboles n -arios

LECCIÓN 15

© All wrongs reversed – bajo licencia CC-BY-NC-SA 4.0



Universidad
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2024/2025

Grado en Ingeniería Informática

UNIVERSIDAD DE ZARAGOZA

Aula 0.04, Edificio Agustín de Betancourt

