ESTRUCTURAS DE DATOS

ÁRBOLES BINARIOS Y DE BÚSQUEDA

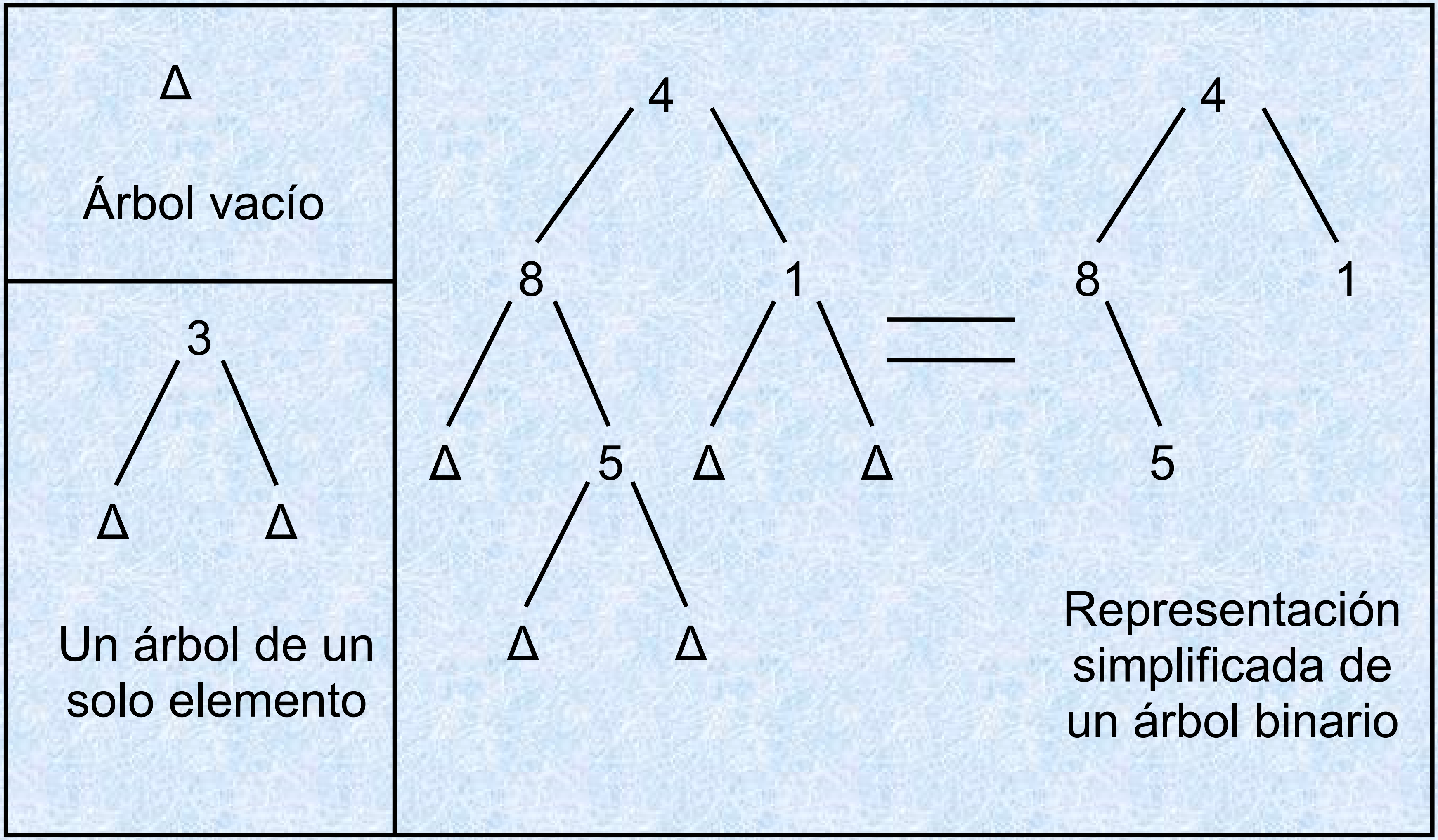
Profesora: Mª José Domínguez Alda

ÁRBOLES BINARIOS

Un árbol binario es un **conjunto de elementos del mismo tipo** (se les suele llamar *nodos*) tal que:

- o bien es el conjunto vacío, en cuyo caso se denomina *árbol vacío* y se denota por Δ;
- o bien es no vacío, y entonces
    - existe un elemento distinguido, llamado *raíz*, y
    - el resto de elementos se distribuyen en dos conjuntos disjuntos, que también forman árboles binarios, y que se denominan *hijo izquierdo* e *hijo derecho*.

EJEMPLOS Y REPRESENTACIÓN HABITUAL



TERMINOLOGÍA

Hoja: árbol con un único elemento.

Camino: secuencia de nodos $N_1, \ldots, N_s$ tal que para cualquier i con $1 \leq i \leq s - 1$, el nodo N_{i+1} es la raíz de un subárbol de N_i.

Longitud de un camino: la longitud del camino $N_1, \ldots, N_s$ es el valor $s - 1$, y es la cantidad de cambios de nodo que hay en el camino.

Ascendiente / Descendiente: Un nodo X es ascendiente de Y (también puede decirse que Y es descendiente de X) si existe un camino de X a Y.

TERMINOLOGÍA (2)

Padre: el primer nodo ascendiente de un nodo. Todos los nodos tienen un único padre, excepto la raíz que no tiene ninguno.

Hijos: son los primeros descendientes de un nodo. Se denomina hijo tanto al subárbol como al nodo de su raíz. Un árbol binario siempre tiene dos hijos (pueden ser vacíos).

Altura: longitud del camino desde la raíz del árbol hasta la hoja más alejada.

Profundidad de un subárbol: longitud del (único) camino desde la raíz hasta dicho árbol.

ESPECIFICACIÓN: ÁRBOLES BINARIOS

espec *ÁRBOLES_BINARIOS[ELEMENTO]*

usa *NATURALES2, BOOLEANOS*

> *{NATURALES2 tiene operaciones para comparar números, como max, min, ≤, etc.}*

parametro formal

> **generos** *elemento*

fparametro

generos *a_bin*   *{árbol_binario}*

ESPECIFICACIÓN: ÁRBOLES BINARIOS (2)

operaciones

Δ *: → a_bin*	*{Generadoras libres}*
_•_•_ : a_bin elemento a_bin → a_bin	

parcial *raíz: a_bin → elemento*

parcial *izq: a_bin → a_bin*

parcial *der: a_bin → a_bin*

vacío?: a_bin → bool

parcial *altura: a_bin → natural*

ESPECIFICACIÓN: ÁRBOLES BINARIOS (3)

var

x: elemento

a, i, d: a_bin

ecuaciones de definitud

vacía?(a) = F ⇒ Def(raíz(a))

vacía?(a) = F ⇒ Def(izq(a))

vacía?(a) = F ⇒ Def(der(a))

vacía?(a) = F ⇒ Def(altura(a))

ESPECIFICACIÓN: ÁRBOLES BINARIOS (4)

ecuaciones

```
raiz(i•x•d) = x

izq(i•x•d) = i

der(i•x•d) = d

vacío?(Δ) = T

vacío(i•x•d) = F
```

ESPECIFICACIÓN: ÁRBOLES BINARIOS (y 5)

$$altura(\Delta \bullet x \bullet \Delta) = 0$$

$$vacío?(i) = F \Rightarrow altura(i \bullet x \bullet \Delta) = suc(altura(i))$$

$$vacío?(d) = F \Rightarrow altura(\Delta \bullet x \bullet d) = suc(altura(d))$$

$$(vacío?(i) = F) \wedge (vacío?(d) = F) \Rightarrow$$
$$altura(i \bullet x \bullet d) = suc(max(altura(i), altura(d)))$$

fespec

OPERACIÓN ALTURA (pseudocódigo)

```
func altura (ab:a_bin) dev natural
 si vacio?(ab) entonces error(Árbol vacío)
 si no
   si (vacío?(izq(a))) entonces
     si (vacío?(der(a)) entonces devolver 0
     si no devolver 1 + altura(der(ab))
     finsi
   si no
     si (vacío?(der(a)) entonces
       devolver 1 + altura(izq(ab))
     si no
       devolver 1 + max(altura(izq(ab)),altura(der(ab)))
     finsi
   finsi
 finsi
finfunc
```

<u>EJEMPLO 1</u>

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

- La operación recibe un árbol binario y devuelve un natural:

  ***suma**: a_bin → natural*

- Declaramos las variables...

  x: natural

  i, d: a_bin

- Como podemos usar el árbol vacío, las ecuaciones son:

  suma(Δ) = 0

  suma(i•x•d) = x + suma(i) + suma(d)

- Si decidimos usar las operaciones de árbol binario,

  vacío?(a) = T ⇒ suma(a) = 0

  vacío?(a) = F ⇒ suma(a) =
  raíz(a) + suma(izq(a)) + suma(der(a))

Profesora: Mª José Domínguez Alda

EJEMPLO 1. PSEUDOCÓDIGO

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

```
func suma_nodos (ab:a_bin) dev natural
       si vacio?(ab) entonces devolver 0
       si no devolver
                     raíz(ab)
                         +
                    suma_nodos(izq(ab))
                         +
                    suma_nodos(der(ab))
       finsi
finfunc
```

EJEMPLO 2

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

- La operación es total:

  hay_par?: a_bin $\rightarrow$ ***bool***

- Las ecuaciones son muy parecidas al ejemplo anterior…

  hay_par?(Δ) = F

  hay_par?(i•x•d) =

  es_par?(x) $\vee$ ***hay_par?(i)*** $\vee$ ***hay_par?(d)***

- …y como veremos con el pseudocódigo, podrían hacerse usando directamente las operaciones del TAD árbol (y no tener que usar las constructoras algebraicas)

EJEMPLO 2. PSEUDOCÓDIGO

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

```
func hay_par(ab:a_bin) dev bool
   si vacio?(ab) entonces devolver F
   sino
      devolver es_par?(raíz(ab))
                 ∨ hay_par(izq(ab))
                 ∨ hay_par(der(ab))
   finsi
finfunc
```

EJEMPLO 3

Ejemplo: Contar cuántos elementos pares hay en un árbol binario de naturales.

- La operación es total:

  cuantos_pares: a_bin → natural

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

  cuantos_pares(Δ) = 0

  es_par?(x)=F ⇒ cuantos_pares(i•x•d) =
  cuantos_pares(i) + cuantos_pares(d)

  es_par?(x)=T ⇒ cuantos_pares(i•x•d) =
  suc(cuantos_pares(i) + cuantos_pares(d))

EJEMPLO 3. PSEUDOCÓDIGO

Ejemplo: Contar cuántos elementos pares hay en un árbol binario de naturales.

```
func cuantos_pares (ab:a_bin) dev natural
var par_en_raíz: natural
    si vacio?(ab) entonces devolver 0
    si no
      si es_par?(raiz(ab)) entonces par_en_raíz ← 1
      si no par_en_raíz ← 0
      finsi
      devolver par_en_raíz + cuantos_pares(izq(ab))
                           + cuantos_pares(der(ab))
    finsi
finfunc
```

EJEMPLO 4

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

- La operación es total:

  igual_forma**: **a_bin a_bin $\rightarrow$ ***bool***

- Es observadora, hay que comprobar todos los casos:

  $igual\_forma(\Delta, \Delta) = T$

  $igual\_forma(i_1 \bullet x \bullet d_1, \Delta) = F$

  $igual\_forma(\Delta, i_2 \bullet y \bullet d_2) = F$

  $igual\_forma(i_1 \bullet x \bullet d_1, i_2 \bullet y \bullet d_2) =$

  $\quad igual\_forma(i_1, i_2) \wedge igual\_forma(d_1, d_2)$

EJEMPLO 4. PSEUDOCÓDIGO

Ejemplo: Comprobar si dos árboles binarios tienen la misma forma

```
func igual_forma(ab1: a_bin, ab2: a_bin) dev bool
   si vacio?(ab1)≠vacío?(ab2) entonces devolver F
   si no
     si vacio?(ab1) entonces devolver T
     si no
     devolver igual_forma(izq(ab1),izq(ab2))
                    ∧
              igual_forma(der(ab1),der(ab2))
     finsi
   finsi
finfunc
```

RECORRIDO DE UN ÁRBOL

Preorden: se visita en primer lugar la raíz del árbol, y a continuación se recorren en preorden todos los subárboles de izquierda a derecha.

Postorden: se recorren en postorden todos los subárboles, de izquierda a derecha, y finalmente se visita la raíz.

Inorden (árboles binarios): se recorre en inorden la rama izquierda, luego se visita la raíz, y después se recorre en inorden la rama derecha.

ESPECIFICACIÓN: ÁRBOLES BINARIOS+

espec *ÁRBOLES_BINARIOS+[ELEMENTO]*

usa *ÁRBOLES_BINARIOS[ELEMENTO], LISTAS[ELEMENTO]*

operaciones

preorden: a_bin → lista *{recorre en preorden}*

inorden: a_bin → lista *{recorre en inorden}*

postorden: a_bin → lista *{recorre en postorden}*

var

x: elemento

i, d: a_bin

Profesora: Mª José Domínguez Alda

ESPECIFICACIÓN: ÁRBOLES BINARIOS+ (y 2)

ecuaciones

```
    preorden(Δ) = []
    preorden(i•x•d) =
                    [x] ++ preorden(i) ++ preorden(d)

    inorden(Δ) = []
    inorden(i•x•d) =
                    inorden(i) ++ [x] ++ inorden(d)

    postorden(Δ) = []
    postorden(i•x•d) =
                    postorden(i) ++ postorden(d) ++ [x]
```

fespec

PSEUDOCÓDIGO PREORDEN

{recorre en *preorden* el árbol binario}

```
func preorden (ab:a_bin) dev lista
var l: lista
   si vacio(ab) entonces l←[]
   si no
     l←raiz(ab):[]
     l←l++preorden(izq(ab))
     l←l++preorden(der(ab))
   finsi
   devolver l
finfunc
```

PSEUDOCÓDIGO PREORDEN

También podríamos hacerlo con un procedimiento, que recibe el árbol y una lista donde va insertando por la derecha los elementos según se los encuentra al recorrer en preorden.

```
proc gpreorden (ab:a_bin, E/S l:lista)
   si !vacio?(ab) entonces
      l ← raiz(ab)#l
      gpreorden(izq(ab), l)
      gpreorden(der(ab), l)
   finsi
finproc

func preorden (ab:a_bin) dev l:lista
   l←[]
   gpreorden(ab,l)
   devolver l
finfunc
```

PSEUDOCÓDIGO INORDEN

{recorre en *inorden* el árbol binario}

```
func inorden(ab:a_bin) dev lista
var l: lista
   si vacio(ab) entonces l←[]
   si no
     l←inorden(izq(ab))
     l←raiz(ab)#l
     l←l++inorden(der(ab))
   finsi
   devolver l
finfunc
```

PSEUDOCÓDIGO INORDEN

Como antes, también podríamos hacerlo con un procedimiento de manera parecida.

```
proc ginorden (ab:a_bin, E/S l:lista)
    si !vacio?(ab) entonces
        ginorden(izq(ab), l)
        l ← raiz(ab)#l
        ginorden(der(ab), l)
    finsi
finproc

func inorden (ab:a_bin) dev l:lista
var l: lista
    l←[]
    ginorden(ab,l)
    devolver l
finfunc
```

PSEUDOCÓDIGO POSTORDEN

{recorre en *postorden* el árbol binario}

```
func postorden(ab:a_bin) dev lista
var l: lista
   si vacio(ab) entonces l←[]
   si no
     l←postorden(izq(ab))
     l←l++postorden(der(ab))
     l←raiz(ab)#l
   finsi
   devolver l
finfunc
```

PSEUDOCÓDIGO POSTORDEN

Con un procedimiento que modifica la lista de entrada / salida

```
proc gpostorden (ab:a_bin, E/S l:lista)
   si !vacio?(ab) entonces
      gpostorden(izq(ab), l)
      gpostorden(der(ab), l)
      l ← raiz(ab)#l
   finsi
finproc

func postorden (ab:a_bin) dev l:lista
var l: lista
   l←[]
   gpostorden(ab,l)
   devolver l
finfunc
```

IMPLEMENTACIÓN DE ÁRBOLES BINARIOS

La implementación más habitual es la de **celdas enlazadas**:

- El tipo árbol es un puntero a una celda
  - Si es vacío, el puntero es "NIL"
  - Si no, apunta a una celda que contiene la raíz del árbol y dos punteros a los subárboles izquierdo y derecho.

ÁRBOLES BINARIOS. TIPOS

```
tipos
   nodo_a_bin =reg                          {esto es lo mínimo}
      valor: elemento
      izq: a_bin
      der: a_bin
   freg
   a_bin = puntero a nodo_a_bin
   {se le pueden añadir otros atributos, como la altura, p.e.}
ftipos
```

ÁRBOLES BINARIOS. CONSTRUCTORAS

*{Crear un árbol binario vacío **Δ**}*

```
func árbol_vacío() dev a:a_bin
   a←nil
finfunc
```

ÁRBOLES BINARIOS. CONSTRUCTORAS (2)

{Crear un árbol binario a partir de un elemento y dos árboles binarios _·_·_}

```
func crea_árbol(e:elemento,hi,hd:a_bin) dev a:a_bin
var aux: nodo_a_bin
   reservar(aux)
   aux^.valor←e
   aux^.izq←hi
   aux^.der←hd
   a ← aux
finfunc
```

ÁRBOLES BINARIOS. OBSERVADORAS

{*Comprobar si es vacío*}

```
func vacío? (ab:a_bin) dev b:bool
   si ab=nil entonces b←T
   si no b←F
   finsi
finfunc
```

ÁRBOLES BINARIOS. OBSERVADORAS (2)

{Devolver la raíz del árbol}

```
func raíz (ab:a_bin) dev r:elemento
   si vacio?(ab) entonces error(Árbol vacío)
   si no r←ab^.valor
   finsi
finfunc
```

{Devolver el subárbol izquierdo}

```
func izquierdo (ab:a_bin) dev i:a_bin
   si vacio?(ab) entonces error(Árbol vacío)
   si no i←a^.izq
   finsi
finfunc
```

{ Devolver el subárbol derecho}

```
func derecho (ab:a_bin) dev d:a_bin
   si vacio?(ab) entonces error(Árbol vacío)
   si no d←a^.der
   finsi
finfunc
```

ÁRBOLES BINARIOS. OBSERVADORAS (3)

{Calcular la altura del árbol binario}

```
func altura (ab:a_bin) dev natural
 si vacio?(ab) entonces error(Árbol vacío)
 si no
   si vacío?(ab^.izq) entonces
     si vacío?(ab^.der) entonces devolver 0
     si no devolver 1 + altura(ab^.der)
     finsi
   si no
     si vacío?(ab^.der) entonces
       devolver 1 + altura(ab^.izq)
     si no
       devolver 1 + max(altura(ab^.izq),altura(ab^.der))
     finsi
   finsi
 finsi
finfunc
```

ÁRBOLES DE BÚSQUEDA

Un árbol de búsqueda es un tipo especial de árbol binario, en el que los elementos están ordenados de la siguiente manera:

- los elementos del hijo izquierdo son todos menores o iguales que la raíz;
- los elementos del hijo derecho son todos mayores que la raíz.

Además de las operaciones típicas de árboles binarios, se añaden operaciones para insertar datos y para comprobar si un dato ya se encuentra en el árbol de búsqueda. Una especificación ampliada posterior permite borrar elementos del árbol.

ESPECIFICACIÓN: ÁRBOLES DE BÚSQUEDA

espec *ÁRBOLES_BÚSQUEDA[ELEMENTO≤]*   *{elem. con orden}*
usa *ÁRBOLES_BINARIOS[ELEMENTO]*
parametro formal
  generos *elemento*
  operaciones *{ todas son "op: elemento elemento → bool" }*
    _ ≤ _   _ < _   _ ≥ _   _ > _   _ = _
  var *x, y: elemento*
  ecuaciones
    { "_ ≤ _" es una relación de orden total en el TAD elemento}
    $x = y = (x \leq y) \wedge (y \leq x)$
    { "_ < _" y "_ > _" se definen usando "_ ≤ _" y "_ = _" }
fparametro

ESPECIFICACIÓN: ÁRBOLES DE BÚSQUEDA (2)

operaciones

insert : elemento a_bin → a_bin *{inserta ordenadamente}*

está? : elemento a_bin → bool *{¿está el dato en el árbol?}*

var

x, y: elemento; i, d: a_bin

ecuaciones

$insert(x,\Delta) = \Delta \bullet x \bullet \Delta$

$(y \leq x) \Rightarrow insert(y,\ i \bullet x \bullet d) = insert(y,\ i) \bullet x \bullet d$

$(y > x) \Rightarrow insert(y,\ i \bullet x \bullet d) = i \bullet x \bullet insert(y,\ d)$

$está?(x,\Delta) = F$

$(y < x) \Rightarrow está?(y,\ i \bullet x \bullet d) = está?(y,\ i)$

$(y = x) \Rightarrow está?(y,\ i \bullet x \bullet d) = T$

$(y > x) \Rightarrow está?(y,\ i \bullet x \bullet d) = está?(y,\ d)$

fespec

ÁRBOLES DE BÚSQUEDA. MODIFICADORAS

{Insertar en orden en un árbol binario de búsqueda}

```
proc insertar (e:elemento, abb:a_bin)
   si vacio?(abb) entonces abb←crea_árbol(e, nil,nil)
   si no
     si e ≤ (abb^.valor) entonces
       si vacío?(abb^.izq) entonces
         abb^.izq←crea_árbol(e, nil,nil)
       si no insertar (e, abb^.izq)
       finsi
     si no
       si vacío?(abb^.der) entonces
         abb^.der←crea_árbol(e, nil,nil)
       si no insertar (e, abb^.der)
       finsi
   finsi
finproc
```

ÁRBOLES DE BÚSQUEDA. OBSERVADORAS

{Buscar un elemento en un árbol binario de búsqueda}

```
func buscar (e.elemento, abb:a_bin) dev bool
   si vacio?(ab) entonces devolver F
   si no
      si e = (abb^.valor) entonces devolver T
      si no
         si e < (abb^.valor) entonces
            devolver buscar (e, abb^.izq)
         si no
            devolver buscar (e, abb^.der)
         finsi
      finsi
   finsi
finfunc
```

ESPECIFICACIÓN: ÁRBOLES DE BÚSQUEDA+

espec *ÁRBOLES_BÚSQUEDA+[ELEMENTO≤]*

usa *ÁRBOLES_BÚSQUEDA[ELEMENTO≤]*

operaciones

borrar : elemento a_bin → a_bin *{quita un elemento}*

{Consideraremos "borrar" total para simplificar las ecuaciones}

operaciones auxiliares

parcial *máximo : a_bin → elemento* *{busca el dato mayor}*

var

x, y: elemento; a, i, d: a_bin

ecuaciones de definitud

vacío?(a) = F ⇒ Def(máximo(a))

ESPECIFICACIÓN: ÁRBOLES DE BÚSQUEDA+ (2)

ecuaciones

$máximo(i \bullet x \bullet \Delta) = x$

$vacío?(d)=F \Rightarrow máximo(i \bullet x \bullet d) = máximo(d)$

$borrar(x,\Delta) = \Delta$

$(y < x) \Rightarrow borrar(y,\ i \bullet x \bullet d) = borrar(y,\ i) \bullet x \bullet d$

$(y > x) \Rightarrow borrar(y,\ i \bullet x \bullet d) = i \bullet x \bullet borrar(y,\ d)$

$(y = x) \Rightarrow borrar(y,\ \Delta \bullet x \bullet d) = d$

$(y = x) \wedge vacío?(i)=F \Rightarrow borrar(y,\ i \bullet x \bullet \Delta) = i$

$(y = x) \wedge vacío?(i)=F \wedge vacío?(d)=F \Rightarrow$

$borrar(y,\ i \bullet x \bullet d) =$

$borrar(máximo(i),\ i) \bullet máximo(i) \bullet d$

fespec

ÁRBOLES DE BÚSQUEDA+. OBSERVADORAS

{Buscar el elemento máximo en el árbol binario de búsqueda}

```
func máximo (abb:a_bin)dev e:elemento
   si (abb=nil ) entonces error (Árbol vacío)
   sino si (abb^.der=nil ) entonces
   e←abb^.valor
   sino e←maximo(abb^.der)
   finsi
finfunc
```

ÁRBOLES DE BÚSQUEDA+. MODIFICADORAS

{Borrar un elemento del árbol}

```
proc borrar (e:elemento, abb:a_bin)
    si !vacio?(abb) entonces
        si (e=valor)entonces                                    {encontrado}
            si (abb^.izq=nil)∧ (abb^.der=nil)entonces  {es hoja}
                        liberar (abb)
                        abb=nil
            sino si (abb^.izq=nil) entonces                     {no hay hi}
                                                    aux←abb
                                                    abb←abb^.der
                                                    liberar(aux)
                    sino si (abb^.der=nil) entonces  {no hay hd}
                                                    aux←abb
                                                    abb←abb^.izq
                                                    liberar(aux)

                            sino                            {tiene hd e hi}
                                  max← máximo(abb^.izq)
                                        {repite la búsqueda para borrarlo}
```

```
                                    borrar(max, abb^.izq)

                                    abb^.valor←max
        finsi

        sino  {sigue buscando e}
            si (e<abb^.valor) entonces borrar(e, abb^.izq)
            sino si (e>abb^.valor)entonces borrar(e, abb^.der)
    finsi
finproc
```

ÁRBOLES DE BÚSQUEDA+. MODIFICADORAS

{Borrar un elemento del árbol}

O también

```
proc borrar (e:elemento, abb:a_bin)
    si !vacio?(abb) entonces
        si (e=valor)entonces                                 {encontrado e}
            si (abb^.izq=nil)∧ (abb^.der=nil)entonces  {es hoja}
                        liberar (abb)
                        abb=nil

            sino si (abb^.izq=nil) entonces  aux←abb
                                             abb←abb^.der
                                             liberar(aux)

                    sino si (abb^.der=nil) entonces aux←abb
                                                 abb←abb^.izq
                                                 liberar(aux)
                            sino borrar_aux(abb,abb^.izq)
            finsi
        sino                          {sigue buscando e}
```

```
            si (e<abb^.valor) entonces borrar(e, abb^.izq)
            sino si (e>abb^.valor)entonces borrar(e, abb^.der)
    finsi
finproc
```

ÁRBOLES DE BÚSQUEDA+. MODIFICADORAS

{Operación auxiliar: Busca en b el nodo con valor máximo, lo elimina y pone el valor máximo en el nodo a. ***No se repite la búsqueda***}

```
proc borrar_aux(a,b:a_bin)
    paux:puntero a nodo_a_bin
    si b^.der!=nil entonces borrar_aux(a, b^.der)
        sino   a^.valor←b^.valor
               paux←b
               b←b^.izq
               liberar(paux)
    finsi
finproc
```

EJEMPLO 5

Ejemplo: Quitar las repeticiones de elementos (es decir, cada elemento solo debe aparecer una vez) en un árbol de búsqueda

- La operación es total:

  quita_copias: a_bin → a_bin

- Las ecuaciones son muy parecidas al ejemplo anterior:

  quita_copias(Δ) = Δ

  está?(x,i)=F ⇒ quita_copias(i•x•d) =
  quita_copias(i) • x • quita_copias(d)

  está?(x,i)=T ⇒ quita_copias(i•x•d) =
  quita_copias(borrar(x,i)•x•d)

EJEMPLO 5-PSEUDOCÓDIGO

Ejemplo: Quitar las repeticiones de elementos (es decir, cada elemento solo debe aparecer una vez) en un árbol de búsqueda.

```
proc quitar_copias (abb:a_bin)

    si !vacio?(abb) entonces

        si esta?(raíz(abb), izquierdo(abb))
            entonces    borrar (raíz (abb), izquierdo(abb))
                        quitar_copias(abb)
            sino        quitar_copias(abb^.izq)
                        quitar_copias(abb^.der)
        finsi
finproc
```