

#### Estructuras de Datos



Soluciones Ejercicios de Árboles binarios y generales

**Ejercicio 1.-** Se dice que un árbol binario es "zurdo" en uno de estos tres casos:

- si es el árbol vacío; o
- si es una hoja; o
- si sus hijos izquierdo y derecho son los dos "zurdos" y el hijo izquierdo tiene más elementos que el hijo derecho.

Crear las operaciones necesarias para determinar si un árbol binario es "zurdo".

#### Solución:

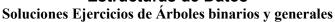
```
Operaciones
```

**Ejercicio 2.-** Extender el TAD árboles binarios de naturales, añadiendo operaciones para:

- a) Obtener la suma de todos los elementos que sean números pares del árbol,
- b) Obtener la imagen especular de un árbol (reflejo respecto al eje vertical),
- c) Crear tres operaciones que generen una lista con los elementos del árbol recorrido en preorden, inorden y postorden,
- d) Comprobar si el árbol está ordenado en inorden, usando para ello únicamente operaciones de árboles (en concreto, no puede utilizarse el apartado anterior).



#### **Estructuras de Datos**





```
Solución:
Los apartados a) y c) están resueltos en clase.
Operaciones
Apartado b)
especular: a bin \rightarrow a bin
func especular (a:a bin) dev aie:a bin
       si vacio?(a) entonces aie \leftarrow \Delta
       sino aie \leftarrow especular(der(a))•raiz(a)•especular(izq(a))
       finsi
finfunc
Apartado d)
mayor_igual: nat a_bin → bool
func mayor igual(n:natural,a:a bin) dev b:bool
       si vacio?(a) entonces b←T
       sino
               b←
                       (n \ge (raiz(a)))
                       \Lambdamayor_igual(n, izq(a))
                       ∧ mayor_igual(n, der(a))
       finsi
finfunc
menor_igual: nat a_bin → bool
func menor igual(n:natural,a:a bin) dev b:bool
       si vacio?(a) entonces b←T
       sino
               b←
                       (n < (raiz(a)))
                       Amenor igual(n, izq(a))
                       \land menor igual(n, der(a))
       finsi
finfunc
```



### Estructuras de Datos





```
esta_inorden?: a_bin → bool

func esta_inorden? (a:a_bin) dev b:bool

si vacio?(a) entonces b←T

sino b← esta_inorden?(izq(a))

Λ (mayor_igual(raíz(a), izq(a)))

Λ esta_inorden?(der(a))

Λ (menor_igual(raíz(a), der(a)))

finsi

finfunc
```

**Ejercicio 3.-** Se quiere hacer un recorrido de un árbol por niveles (el nivel k son todos los nodos que están a distancia k de la raíz del árbol). Se pide:

- a) nivel\_n: a\_bin natural → lista, que crea una lista con todos los nodos que se encuentren en el nivel indicado por el *natural* del segundo parámetro;
- b) niveles\_entre: a\_bin natural natural → lista, que crea una lista con todos los nodos que se encuentren entre los niveles indicados por los dos números naturales; y
- c) recorrer\_niveles: a\_bin → lista, que crea una lista formada por todos los niveles del árbol binario.

#### Solución Operaciones

```
Apartado a)

nivel_n: árbol→lista

func nivel-n(a:a_bin, n:natural) dev l:lista

si vacio?(a) entonces l←[]

sino si n=0 entonces l←[raíz(a)]

sino l←nivel-n (izquierdo(a),n-1)

++ nivel-n(derecho(a),n-1)

finsi
```

finfunc



# **Estructuras de Datos** Soluciones Ejercicios de Árboles binarios y generales



```
Apartado b)
Niveles entre: árbol→lista
func niveles-entre (a:árbol, n, m:natural)
dev 1:lista
        si (n<0) V (m<0) V (n>m)
        entonces Error ('recorrido incorrecto')
        sino si (n=m) entonces 1 \leftarrow \text{nivel } n(a,n)
                sino 1 \leftarrow niveles-entre(a, n, m-1)
                         ++nivel-n(a,m)
        finsi
finfunc
Apartado c)
recorrer niveles: árbol→lista
func recorrer niveles (a:árbol)dev l:lista
        1 \leftarrow \text{niveles-entre}(a, 0, \text{altura}(a))
```

finfunc