



Ejercicio 4.- Extender la especificación de los árboles generales vista en clase con las siguientes operaciones:

- num_nodos: árbol \rightarrow natural, para calcular cuántos nodos hay en un árbol general;
- num_hojas: árbol \rightarrow natural, para ver la cantidad total de hojas que tiene un árbol general;
- max_hijos: árbol \rightarrow natural, que obtiene cuál es la mayor cantidad de hijos en un mismo nodo que hay en un árbol general.
- reflejar: árbol \rightarrow árbol, que obtiene la imagen especular de un árbol;
- frontera: árbol \rightarrow lista, que genera una lista formada por los elementos almacenados en las hojas del árbol, tomados de izquierda a derecha.

Solución Operaciones :

Apartado a)

num_nodos: árbol \rightarrow natural

```
func num_nodos (a:árbol) dev n:natural
    n  $\leftarrow$  1 + num_nodos_b (hijos(a))
```

finfunc

num_nodos_b: bosque \rightarrow natural

```
func num_nodos_b (b:bosque) dev n:natural
```

```
var prim:arbol
```

```
    si vacio?(b) entonces n  $\leftarrow$  0
```

```
    sino    prim  $\leftarrow$  primero(b)
```

```
        n  $\leftarrow$     num_nodos (prim)
            + num_nodos _b(resto(b))
```

```
    fin_si
```

finfunc



Apartado b)

num_hojas: árbol \rightarrow natural

func num_hojas (a:árbol) **dev** n:natural

 si vacio?(bosque(a)) entonces $n \leftarrow 1$

 sino $n \leftarrow \text{num_hojas_b}(\text{hijos}(a))$

finfunc

num_hojas_b: bosque \rightarrow natural

func num_hojas_b (b:bosque) **dev** n:natural

var prim:arbol

si vacio?(b) **entonces** $n \leftarrow 0$

sino

 prim \leftarrow primero(b)

$n \leftarrow \text{num_hojas}(\text{prim}) + \text{num_hojas_b}(\text{resto}(b))$

fin_si

finfunc

Apartado c)

máx_hijos: árbol \rightarrow natural

func máx_hijos (a:árbol) **dev** n:natural

var num_hijos, max_hijos_b:natural

 num_hijos \leftarrow num_hijos(a)

 max_hijos_b \leftarrow max_hijos_b(bosque(a))

si (num_hijos > max_hijos_b)

entonces $n \leftarrow$ num_hijos

sino $n \leftarrow$ máx_hijos_b



finsi

finfunc

máx_hijos_b: bosque \rightarrow natural

func máx_hijos_b (b: bosque) **dev** n: natural

var prim: árbol num_hijos_p, max_hijos_r: natural

si vacío?(b) **entonces** max_hijos_b \leftarrow 0

sino prim \leftarrow primero(b)

num_hijos_p \leftarrow num_hijos(prim)

max_hijos_r \leftarrow max_hijos_b(resto(b))

finsi

si (num_hijos_p > max_hijos_r

entonces n \leftarrow num_hijos_p

sino n \leftarrow max_hijos_r

finsi

finfunc

Apartado d)

reflejar: árbol \rightarrow árbol

func reflejar (a: árbol) **dev** ar: árbol

ar \leftarrow raiz(a) • reflejar_b(bosque(a))

finfunc



bosque_imagen:bosque→bosque

proc bosque_imagen (b, bimag:bosque)

{procedimiento auxiliar que va creando el bosque imagen de b en bimag}

var prim:árbol

mientras !vacío(b) **hacer** prim←primero(b)

 b←resto(b)

 bimag←reflejar(prim):bimag

finmientras

finsi

reflejar _b:bosque→bosque

func reflejar_b(b:bosque) **dev** br:bosque

var bimagen:bosque

 bimagen←[]

 bosque_imagen(b, bimagen)

{procedimiento auxiliar que va creando el bosque imagen de b}

 br←bimag

finfunc

Apartado d)

frontera: árbol→lista

func frontera(a:árbol) **dev** l:lista

si num_hijos(a) =0 **entonces** l←[raíz(a)]

sino l←frontera_b(hijos(a))

finfunc



```
func frontera_b (b:bosque) dev l:lista
    si vacio?(b) entonces l←[]
    sino l←frontera(primer(b))++frontera_b(resto(b))
    finsi
```

finfunc

Ejercicio 5.-Llamaremos a un árbol general de naturales “creciente” en cada nivel del árbol, la cantidad de nodos que hay en ese nivel es igual al valor del nivel más uno; es decir, el nivel 0 tiene exactamente un nodo, el nivel 1 tiene exactamente dos nodos, el nivel k tiene exactamente $k + 1$ nodos. Se pide:

- Especificar completamente el TAD árbol general,
- Comprobar si un árbol general es “creciente”,
- Buscar el nodo con mayor cantidad de hijos de un árbol creciente.

Necesitaremos una función auxiliar que cuente el total de nodos de un nivel dado k :

```
func nodos_nivel_k (a:árbol, k:natural) dev n:natural
    si k=0 entonces n←1
    sino n←nodos_nivel_k_b(hijos(a), k-1)
    finsi

finfunc

func nodos_nivel_k_b(b:bosque, k:natural) dev n:natural
    si vacio?(b) entonces n←0
    sino si k=0 entonces n←long(b)           {tamaño del bosque}
    sino
        n←nodos_nivel_k(primer(b), k) + nodos_nivel_k_b(resto(b), k)
    finsi

finfunc
```



func creciente (a:árbol) **dev** b:boolean

 creciente_desde_k(a,1)

finfunc

func creciente_desde_k (a: árbol,k:natural) **dev** b:boolean

si nodos_nivel_k(a, k) =0 **entonces** b \leftarrow T

sino si nodos_nivel_k(a, k) !=k+1 **entonces** b \leftarrow F

sino b \leftarrow creciente_desde_k(a, k+1)

finsi

finfunc

o tb. {versión iterativa}

func creciente (a:árbol) **dev** b:boolean

 k \leftarrow 0 es_creciente \leftarrow T

mientras nodos_nivel_k(a, k) !=0 \wedge es_creciente **hacer**

si nodos_nivel_k(a, k) !=k+1 **entonces** es_creciente \leftarrow F

sino k \leftarrow k+1

finsi

finmientras

finfunc