



## 3. ESTRUCTURAS DE DATOS NO LINEALES

### 3.1 Conjuntos

- ▲ Diseño basado en abstracciones: abstracción procedimental y de datos

### 3.2 Árboles binarios

- ▲ Diseño basado en abstracciones: abstracción procedimental y de datos

### 3.3 Grafos

- ▲ Diseño basado en abstracciones: abstracción procedimental y de datos

### 3.3 Tablas

- ▲ Diseño basado en abstracciones: abstracción procedimental y de dato

EDI 2002/03

1

---

---

---

---

---

---

---

---



## 3.2 Árboles Binarios

### 3.2.1 Árboles: definiciones

### 3.2.2 Concepto de árbol binario

### 3.2.3 Especificación algebraica del TAD

TipoArbolBinario

### 3.2.4 Implementaciones del TAD TipoArbolBinario

### 3.2.5 Programación con árboles binarios

### 3.2.6 Ejercicios

### 3.2.7 Árboles binarios de búsqueda

EDI 2002/03

2

---

---

---

---

---

---

---

---



## 3.2.1 Árboles: definiciones

### ● Arbol A (definición recursiva)

- ▲ El conjunto vacío es un árbol (árbol vacío).
- ▲ Si  $A_1, A_2, \dots, A_n$  son árboles ( $n \geq 0$ ) y  $R$  es un elemento de información (nodo), entonces el conjunto formado por  $R$  y los árboles  $A_1, A_2, \dots, A_n$  es un árbol.
- $R$  se denomina raíz de  $A$ . Se dice que  $A_1, A_2, \dots, A_n$  son **subárboles** de  $A$  o del nodo raíz.
- Cuando el orden de los subárboles importa, se dice que el árbol está **ordenado**.
- **Nodo**: Elementos de información. Está relacionado con otros nodos por medio de **ramas**.

EDI 2002/03

3

---

---

---

---

---

---

---

---



### 3.2.1 Árboles: definiciones

- **Grado de un nodo:** Número de subárboles del nodo no vacíos.
- **Grado de un árbol:** Máximo grado de los nodos de ese árbol.
- **Hoja o nodo terminal:** Nodo de grado 0
- **Nodo no terminal:** Nodo de grado  $> 0$
- Los nodos raíz de los subárboles de un nodo X se llaman **hijos del nodo X**. Se dice que X es el **nodo** padre.
- Se dice que los nodos hijos del mismo padre son **nodos hermanos**.

EDI 2002/03

4

---

---

---

---

---

---

---



### 3.2.1 Árboles: definiciones

- Un árbol B está incluido en un árbol A si:
  - ⚡ B es un subárbol de A
  - o
  - ⚡ B está incluido en algún subárbol de A
- **Antecedentes de un nodo X en un árbol A:** conjunto formado por la raíz de A y todas las raíces de los árboles incluidos en A que incluyen a su vez al subárbol de raíz X, exceptuando el propio X.

EDI 2002/03

5

---

---

---

---

---

---

---



### 3.2.1 Árboles: definiciones

- **Nivel de un nodo** (definición inductiva):
  - ⚡ El nodo raíz tiene nivel 1
  - ⚡ Si un nodo tiene nivel  $n$ , sus hijos tendrán nivel  $n+1$
- **Altura o profundidad de un árbol:** Máximo nivel de sus nodos
- **Peso de un árbol:** Es el número de hojas
- **Grado de un árbol:** Máximo grado de los nodos de ese árbol.
- **Árbol homogéneo de grado  $n$ :** El grado de todos sus nodos es  $n$ , exceptuando a las hojas.

EDI 2002/03

6

---

---

---

---

---

---

---



### 3.2.1 Árboles: definiciones

- **Árbol completo de grado  $n$  y profundidad  $K$ :** Todas las hojas son de nivel  $k$  y es homogéneo de grado  $n$
- **Árbol casi completo:** puede obtenerse de un árbol completo eliminando 0 o más hojas consecutivas del último nivel, comenzando por la derecha.
- **Bosque:** Conjunto formado por  $n$  árboles disjuntos ( $n \geq 0$ )

EDI 2002/03

7

---

---

---

---

---

---

---



### 3.2.1 Árboles: definiciones

- **Representaciones de un árbol:**
  - ▲ Representación jerárquica (nodos y ramas)
  - ▲ Conjuntos
  - ▲ Lista de nodos  
(raíz, (subárbol1), ..., (subárbol  $n$ ))  
raíz ((subárbol1).....(subárbol  $n$ ))
- **Ejemplos:** Árbol genealógico, diagramas de organización, ...

EDI 2002/03

8

---

---

---

---

---

---

---



### 3.2.2 Concepto de árbol binario

- **Árbol binario:** Es un árbol ordenado donde cada nodo tiene como máximo grado 2.
  - ▲ Es decir, un árbol binario o es vacío o es un nodo con información (raíz) y dos subárboles llamados hijo izquierdo e hijo derecho, que son disjuntos y son a su vez árboles binarios.
  - ▲ Árbol binario linealizado (a la izquierda o derecha): Todo nodo tiene grado 0 o 1
  - ▲ Árbol binario homogéneo. Todo nodo tiene grado 0 o 2.

EDI 2002/03

9

---

---

---

---

---

---

---



## 3.2.2 Concepto de árbol binario

- ▲ Árbol binario equilibrado. La diferencia de altura entre los subárboles de cualquier nodo es como máximo la unidad. Cuando los subárboles de todos los nodos tienen todos la misma altura se dice que está perfectamente equilibrado.
- ▲ Árbol binario compensado. El número de nodos del subárbol izquierdo coincide con el número de nodos del subárbol derecho.

---

---

---

---

---

---

---

---



## 3.2.2 Concepto de árbol binario

- Propiedades:
  - ▲ El máximo número de nodos en el nivel  $i$ -ésimo de un árbol binario es  $2^{i-1}$ ,  $i \geq 1$
  - ▲ El número máximo de nodos de un árbol binario de profundidad  $K$  es  $2^k - 1$ ,  $k \geq 1$
  - ▲ Para todo árbol binario no vacío
$$n_0 = n_2 + 1$$
$$n_0$$
 es el número de nodos hoja  
 $n_2$  es el número de nodos de grado 2

---

---

---

---

---

---

---

---



## 3.2.3 Especificación algebraica del TAD TipoArbolBinario

- Especificación formal:
  - ▲ Un árbol binario, definido como TAD, es una colección jerarquizada de elementos (todos del mismo tipo) y organizada como un nodo raíz y dos subárboles disjuntos (izquierdo y derecho).
- Operaciones:
  - ▲ CrearArbolBinVacio, ConstruirArbolBin, EsArbolBinVacio, Raíz, HijoIzdo, HijoDcho

---

---

---

---

---

---

---

---



### 3.2.3 Especificación algebraica del TAD TipoArbolBinario

ESPECIFICACION ArbolesBinarios

PARAMETROS GENERICOS

TIPOS TipoElemento

FIN PARAMETROS GENERICOS

TIPOS TipoArbolBin

OPERACIONES

(\* CONSTRUCTORAS GENERADORAS \*)

CrearArbolBinVacio:  $\rightarrow$  TipoArbolBin

ConstruirArbolBin:

TipoArbolBin x TipoElemento x TipoArbolBin  $\rightarrow$  TipoArbolBin

(\* OBSERVADORAS SELECTORAS \*)

PARCIAL Raiz: TipoArbolBin  $\rightarrow$  TipoElemento

PARCIAL HijoIzdo: TipoArbolBin  $\rightarrow$  TipoArbolBin

PARCIAL HijoDcho: TipoArbolBin  $\rightarrow$  TipoArbolBin

(\* OBSERVADORAS NO SELECTORAS \*)

EsArbolBinVacio: TipoArbolBin  $\rightarrow$  Booleano

13



### 3.2.3 Especificación algebraica del TAD TipoArbolBinario

(\* CONSTRUCTORAS NO GENERADORAS \*)

VARIABLES

r: TipoElemento;

i, d : TipoArbolBin;

ECUACIONES DE DEFINITUD

DEF(Raiz(ConstruirArbolBin(i,r,d)))

DEF(HijoIzdo(ConstruirArbolBin(i,r,d)))

DEF(HijoDcho(ConstruirArbolBin(i,r,d)))

EDI 2002/03

14



### 3.2.3 Especificación algebraica del TAD TipoArbolBinario

ECUACIONES

(\* Observadoras selectoras \*)

Raiz(ConstruirArbolBin(i,r,d)) = r

HijoIzdo(ConstruirArbolBin(i,r,d)) = i

HijoDcho(ConstruirArbolBin(i,r,d)) = d

(\* Observadoras no selectoras \*)

EsArbolBinVacio(CrearArbolBinVacio) = CIERTO

EsArbolBinVacio(ConstruirArbolBin(i,r,d)) = FALSO

FIN ESPECIFICACIÓN

EDI 2002/03

15



### 3.2.4 Implementaciones del TAD TipoArbolBinario

- Implementación dinámica (punteros)
- Implementación estática (vectores)
  - ▲ Simulando memoria dinámica
  - ▲ Montículos (HEAP's)

EDI 2002/03

16

---

---

---

---

---

---

---



### 3.2.4 Implementaciones del TAD TipoArbolBinario

- Implementación dinámica (punteros)  
TYPE  
TipoArbolBin = ^TipoNodo;  
TipoNodo= RECORD  
    hijoizq, hijodrch: TipoArbolBin;  
    raiz: TipoElemento;  
END;

EDI 2002/03

17

---

---

---

---

---

---

---

```
UNIT ArBinTAD;
INTERFACE
    USES ElemTAD;

    (* DECLARACION TIPO. PRIVADO *)
    TYPE
        TipoArbolBin = ^TipoNodo;
        TipoNodo = RECORD
            hijoizq, hijodrch: TipoArbolBin;
            raiz: TipoElemento;
        END;

    (*****
    (* OPERACIONES PUBLICAS *)

    (* OPERACIONES CONSTRUCTORAS GENERADORAS *)

    PROCEDURE CrearArbolBinVacio (VAR arbol:TipoArbolBin);
    {PRE: Cierto}
    {POST: Inicializa el arbol}
```

---

---

---

---

---

---

---

```
PROCEDURE ConstruirArbolBin(izqdo:TipoArbolBin; e: TipoElemento;
                             drcho:TipoArbolBin; VAR arbol:TipoArbolBin);
{PRE: Hay espacio para almacenar el nuevo elemento en el arbol}
{POST: Construye un arbol con "e" como raíz del "arbol" e hijos "izqdo" y
"drcho"
      La memoria reservada por "drcho" e "izqdo" pasan a ser propiedad de
"arbol"}
{EXCEPCION: "Memoria Agotada" si no queda espacio de almacenamiento}
```

(\* OPERACIONES OBSERVADORAS SELECTORAS \*)

```
FUNCTION Raiz(arbol: TipoArbolBin): TipoElemento;
{PRE: "arbol" no esta vacio}
{POST: devuelve la raíz del "arbol"}
```

```
FUNCTION HijoIzdo(arbol: TipoArbolBin): TipoArbolBin;
{PRE: "arbol" no esta vacio}
{POST: devuelve el hijo izquierdo del arbol}
```

```
FUNCTION HijoDcho(arbol: TipoArbolBin): TipoArbolBin;
{PRE: "arbol" no esta vacio}
{POST: devuelve el hijo derecho del arbol}
```

---

---

---

---

---

---

---

---

(\* OPERACIONES OBSERVADORAS NO SELECTORAS \*)

```
FUNCTION EsArbolBinVacio(arbol:TipoArbolBin):Boolean;
{PRE: Cierto}
{POST: devuelve TRUE si "arbol" esta vacio}
```

```
FUNCTION Igual(arbin1,arbin2:TipoArbolBin):Boolean;
{PRE: Cierto}
{POST: devuelve TRUE si "arbin1" y "arbin2" son iguales}
```

(\* OPERACIONES CONSTRUCTORAS NO GENERADORAS \*)

```
PROCEDURE Copiar( VAR destino :TipoArbolBin; origen:TipoArbolBin);
{PRE: Hay espacio para copiar todos los elementos de "origen"}
{POST: Devuelve en "destino" una copia completa de "origen"}
{EXCEPCION: "Memoria Agotada" si no queda espacio de almacenamiento}
```

(\* OPERACION DESTRUCTORA \*)

```
PROCEDURE DestruirArbolBin(VAR arbol:TipoArbolBin);
{PRE: Cierto}
{POST: destruye todos los nodos de "arbol" y deja el arbol vacio}
```

---

---

---

---

---

---

---

---

```
(*****)
(* OPERACIONES PROTEGIDAS *)
```

```
FUNCTION CreaNodo (izqdo: TipoArbolBin; e: TipoElemento; dcho:
TipoArbolBin):TipoArbolBin;
{PRE: Cierto}
{POST: Devuelve un arbol binario con raíz "e" e hijos "izqdo",
"dcho"}
```

```
PROCEDURE LiberarNodo (VAR nodo: TipoArbolBin);
{PRE: Cierto}
{POST: Libera el espacio reservado por el nodo}
```

---

---

---

---

---

---

---

---

```

IMPLEMENTATION
(*****
(*  OPERACIONES PROTEGIDAS                                *)
*)

FUNCTION CreaNodo (izqdo: TipoArbolBin; e: TipoElemento; dcho:
TipoArbolBin):TipoArbolBin;
{Complejidad: O(1)}
VAR
    nuevo:TipoArbolBin;
BEGIN
    New(nuevo);
    nuevo^.raiz:=e;
    nuevo^.hijoizq:=izqdo;
    nuevo^.hijodrch:=dcho;

    CreaNodo:= nuevo;
END;

PROCEDURE LiberarNodo (VAR nodo: TipoArbolBin);
{Complejidad: O(1)}
BEGIN
    Dispose(nodo);
    nodo:=NIL;
END;

```

---

---

---

---

---

---

---

---

```

(*****
(*  OPERACIONES PUBLICAS                                *)
*)

(* OPERACIONES CONSTRUCTORAS GENERADORAS *)

PROCEDURE CrearArbolBinVacio (VAR arbol:TipoArbolBin);
{Complejidad: O(1)}
BEGIN
    arbol := NIL;
END;

PROCEDURE ConstruirArbolBin(izqdo:TipoArbolBin; e: TipoElemento;
drcho:TipoArbolBin; VAR arbol:TipoArbolBin);
{Complejidad: O(1)}
BEGIN
    IF (arbol <> izqdo) AND (arbol<> drcho) THEN
        DestruirArbolBin(arbol);

    arbol:= CreaNodo(izqdo,e,drcho);
END;

```

---

---

---

---

---

---

---

---

```

(* OPERACIONES OBSERVADORAS SELECTORAS *)

FUNCTION Raiz(arbol: TipoArbolBin): TipoElemento;
{Complejidad: O(1)}
BEGIN
    IF NOT EsArbolBinVacio(arbol) THEN
        Raiz:=arbol^.raiz
    END;

FUNCTION HijoIzdo(arbol: TipoArbolBin): TipoArbolBin;
{Complejidad: O(1)}
BEGIN
    IF NOT EsArbolBinVacio(arbol) THEN
        HijoIzdo:=arbol^.hijoizq
    END;

FUNCTION HijoDcho(arbol: TipoArbolBin): TipoArbolBin;
{Complejidad: O(1)}
BEGIN
    IF NOT EsArbolBinVacio(arbol) THEN
        HijoDcho:=arbol^.hijodrch
    END;

```

---

---

---

---

---

---

---

---



(\* OPERACIONES OBSERVADORAS NO SELECTORAS \*)

```
FUNCTION EsArbolBinVacio(arbol:TipoArbolBin):Boolean;
{Complejidad: O(1)}
BEGIN
    EsArbolBinVacio:= arbol = NIL;
END;

FUNCTION Igual(arbin1,arbin2:TipoArbolBin):Boolean;
{Complejidad: O(min(num_nodos(arbin1), num_nodos(arbin2))}
BEGIN
    IF EsArbolBinVacio (arbin1) AND EsArbolBinVacio (arbin2) THEN
        Igual:=TRUE
    ELSE IF EsArbolBinVacio(arbin1) OR EsArbolBinVacio(arbin2) THEN
        Igual:= FALSE
    ELSE (* ninguno de los dos es vacio *)
        Igual := (Raiz(arbin1) = Raiz(arbin2)) AND
        Igual(HijoIzdo(arbin1), HijoIzdo(arbin2)) AND
        Igual(HijoDcho(arbin1), HijoDcho (arbin2))
    END;
END;
```

---

---

---

---

---

---

---

---

```
PROCEDURE Copiar( VAR destino :TipoArbolBin; origen:TipoArbolBin);
{COMPLEJIDAD: O(max(num_nodos(origen), num_nodos(destino)) }
VAR
    hi, hd: TipoArbolBin;
BEGIN
    IF EsArbolBinVacio(origen) THEN
        DestruirArbolBin(destino)
    ELSE IF EsArbolBinVacio(destino) THEN
        BEGIN
            CrearArbolBinVacio(hi);
            CrearArbolBinVacio(hd);
            Copiar (hi, HijoIzdo(origen));
            Copiar (hd, HijoDcho(origen));
            ConstruirArbolBin(hi, Raiz(origen), hd, destino);
        END
    ELSE (* reutiliza la memoria de destino *)
        BEGIN
            destino^.raiz := Raiz(origen);
            Copiar (destino^.hijoizq, HijoIzdo(origen));
            Copiar (destino^.hijodrch, HijoDcho(origen));
        END;
    END;
END;
```

---

---

---

---

---

---

---

---

(\* OPERACION DESTRUCTORA \*)

```
PROCEDURE DestruirArbolBin(VAR arbol:TipoArbolBin);
{Complejidad: O(n), siendo n el número de nodos de "arbol"}
BEGIN

    IF NOT EsArbolBinVacio(arbol) THEN
        BEGIN
            DestruirArbolBin(arbol^.hijoizq);
            DestruirArbolBin(arbol^.hijodrch);
            LiberarNodo(arbol);
        END
    END;

END.
```

---

---

---

---

---

---

---

---

#### EJEMPLO PROGRAMA BEGIN

(\* [A,[B,[C],[D]],[E,[F]] <--> A(B(C(D)))(E (F())) \*)

```
CrearArbolBinVacio(vacio);
CrearArbolBinVacio(arbol);
CrearArbolBinVacio(izqdo);
CrearArbolBinVacio(drcho);
```

```
ConstruirArbolBin(vacio, 'C', vacio, izqdo);
ConstruirArbolBin(vacio, 'D', vacio, drcho);
ConstruirArbolBin(izqdo, 'B', drcho, arbol);
```

```
CrearArbolBinVacio(izqdo);
CrearArbolBinVacio(drcho);
ConstruirArbolBin(vacio, 'F', vacio, izqdo);
ConstruirArbolBin(izqdo, 'E', vacio, drcho);
```

```
ConstruirArbolBin(arbol, 'A', drcho, arbol);
```

```
Writeln;
DibujarArbolBin(arbol);
```

```
CrearArbolBinVacio(arbol1);
Copiar(arbol1,arbol);
```

```
Writeln;
DibujarArbolBin(arbol1);
Writeln;
Writeln('IGUALES?',Igual(arbol,arbol1));
```

```
DestruirArbolBin(arbol);
DestruirArbolBin(arbol1);
```

END.



### 3.2.4 Implementaciones del TAD TipoArbolBinario

- Implementación estática (Simulando memoria dinámica)

CONST

MAXELEM = 100;

TYPE

TipoArbolBinPos = 0..MAXELEM;

TipoNodo = RECORD

    raiz: TipoElemento;

    hijoizqd,hjodrhc: TipoArbolBinPos;

END;

TipoArbolBin = RECORD

    memoria: ARRAY [1..MAXELEM] OF TipoNodo;

    numNodos: 0..MAXELEM;

END;

EDI 2002/03

```

(* DECLARACION TIPO. PRIVADO *)
CONST
MAXELEM = 100;
TYPE
  TipoArbolBinPos = 0..MAXELEM;

  TipoNodo = RECORD
    raiz: TipoElemento;
    hijoizqd,hjodrhc: TipoArbolBinPos;
  END;
  TipoArbolBin = RECORD
    memoria: ARRAY [1..MAXELEM] OF TipoNodo;
    numNodos: 0..MAXELEM;
  END;

```

---

---

---

---

---

---

---

---

```

(*****
(* OPERACIONES PUBLICAS *)
(* OPERACIONES CONSTRUCTORA GENERADORAS *)

PROCEDURE CrearArbolBinVacio (VAR arbol:TipoArbolBin);
{PRE: Cierto}
{POST: Inicializa el arbol}

PROCEDURE ConstruirArbolBin(izqdo:TipoArbolBin;
  e: TipoElemento;
  drcho:TipoArbolBin;
  VAR arbol:TipoArbolBin);
{PRE: Hay espacio para almacenar elemento en el arbol}
{POST: Construye un arbol con "e" como raíz del "arbol" e hijos "izqdo" y
"drcho"}
{EXCEPCION: "Memoria Agotada" si no queda espacio de almacenamiento}

(* OPERACIONES OBSERVADORAS SELECTORAS *)

FUNCTION Raiz(arbol: TipoArbolBin; pos:TipoArbolBinPos): TipoElemento;
{PRE: "arbol" no esta vacio}
{POST: devuelve el elemento raíz de la posicion}

```

---

---

---

---

---

---

---

---

```

FUNCTION HijoIzdo(arbol: TipoArbolBin; pos:TipoArbolBinPos):
  TipoArbolBinPos;
{PRE: "arbol" no esta vacio}
{POST: devuelve el hijo izquierdo del nodo pos del arbol}

FUNCTION HijoDcho(arbol: TipoArbolBin; pos:TipoArbolBinPos):
  TipoArbolBinPos;
{PRE: "arbol" no esta vacio}
{POST: devuelve el hijo derecho del nodo pos del arbol}

FUNCTION NumNodos(arbol: TipoArbolBin): TipoArbolBinPos;
{POST: devuelve el numero de nodos del arbol}

(* OPERACIONES OBSERVADORAS NO SELECTORAS *)
FUNCTION EsArbolBinVacio(arbol:TipoArbolBin):Boolean;
{PRE: Cierto}
{POST: devuelve TRUE si "arbol" esta vacio}
FUNCTION EsArbolBinPosVacía(arbol:TipoArbolBin;
  pos:TipoArbolBinPos):Boolean;
{PRE: Cierto}
{POST: devuelve TRUE si "arbol" esta vacio o posicion vacia}

```

---

---

---

---

---

---

---

---

#### IMPLEMENTATION

(\* OPERACIONES AUXILIARES \*)

```
PROCEDURE InsertarNodo(posOrigenI,posOrigenD,posDest:TipoArbolBinPos;
    elem:TipoElemento;
    arbolOrigenI,arbolOrigenD:TipoArbolBin;
```

```
    VAR arbolDestino:TipoArbolBin);
```

```
BEGIN
```

```
    arbolDestino.numNodos:=arbolDestino.numNodos+1;
```

```
    arbolDestino.Memoria[posDest].raiz:=elem;
```

```
    IF posOrigenI = 0 THEN
```

```
        arbolDestino.Memoria[posDest].hijoizqd:=0
```

```
    ELSE
```

```
        BEGIN
```

```
            arbolDestino.Memoria[posDest].hijoizqd:=arbolDestino.numNodos+1;
```

```
            InsertarNodo(arbolOrigenI.Memoria[posOrigenI].hijoizqd,
```

```
                arbolOrigenI.Memoria[posOrigenI].hijodrch,
```

```
                arbolDestino.numNodos+1,
```

```
                arbolOrigenI.Memoria[posOrigenI].raiz,
```

```
            arbolOrigenI,arbolOrigenI,arbolDestino);
```

```
        END;
```

```
IF posOrigenD = 0 THEN
```

```
    arbolDestino.Memoria[posDest].hijodrch:=0
```

```
ELSE
```

```
    BEGIN
```

```
        arbolDestino.Memoria[posDest].hijodrch:=arbolDestino.numNodos+1;
```

```
        InsertarNodo(arbolOrigenD.Memoria[posOrigenD].hijoizqd,
```

```
            arbolOrigenD.Memoria[posOrigenD].hijodrch,
```

```
            arbolDestino.numNodos+1 ,
```

```
            arbolOrigenD.Memoria[posOrigenD].raiz,
```

```
        arbolOrigenD,arbolOrigenD,arbolDestino);
```

```
    END
```

```
END;
```

```
(*****)
```

```
(* OPERACIONES PUBLICAS *)
```

```
(* OPERACIONES CONSTRUCTORAS GENERADORAS *)
```

```
PROCEDURE CrearArbolBinVacio (VAR arbol:TipoArbolBin);
```

```
{Complejidad: O(1)}
```

```
BEGIN
```

```
    arbol.numNodos := 0;
```

```
END;
```

```
PROCEDURE ConstruirArbolBin(izqdo:TipoArbolBin;
```

```
    e: TipoElemento;
```

```
    drcho:TipoArbolBin;
```

```
    VAR arbol:TipoArbolBin);
```

```
{Complejidad: O(n)}
```

```
VAR
```

```
    hizq,hdrcho:TipoArbolBinPos;
```

```
    arbolAux:TipoArbolBin;
```

```
BEGIN
```

```
    CrearArbolBinVacio(arbolAux);
```

```
    hizq:=1;
```

```
    hdrcho:=1;
```

```
    IF EsArbolBinVacio(izqdo) THEN
```

```
        hizq:=0;
```

```
    IF EsArbolBinVacio(drcho) THEN
```

```
        hdrcho:=0;
```

```
    InsertarNodo(hizq,hdrcho,1,e, izqdo,drcho,arbolAux);
```

```
    arbol:=arbolAux;
```

```
END;
```

(\* OPERACIONES OBSERVADORAS SELECTORAS \*)

```
FUNCTION Raiz(arbol: TipoArbolBin; pos:TipoArbolBinPos): TipoElemento;
{Complejidad: O(1)}
BEGIN
  IF NOT EsArbolBinVacio(arbol) THEN
    Raiz:=arbol.Memoria[pos].raiz
  END;
```

```
FUNCTION HijoIzdo(arbol: TipoArbolBin; pos:TipoArbolBinPos): TipoArbolBinPos;
{Complejidad: O(1)}
BEGIN
  IF NOT EsArbolBinVacio(arbol) THEN
    HijoIzdo:=arbol.Memoria[pos].hijoizqd
  END;
```

```
FUNCTION HijoDcho(arbol: TipoArbolBin; pos:TipoArbolBinPos): TipoArbolBinPos;
{Complejidad: O(1)}
BEGIN
  IF NOT EsArbolBinVacio(arbol) THEN
    HijoDcho:=arbol.Memoria[pos].hijodrch
  END;
```

---

---

---

---

---

---

---

---

```
FUNCTION NumNodos(arbol: TipoArbolBin): TipoArbolBinPos;
{Complejidad: O(1)}
BEGIN
  NumNodos:=arbol.numNodos
END;
```

(\* OPERACIONES OBSERVADORAS NO SELECTORAS \*)

```
FUNCTION EsArbolBinVacio(arbol:TipoArbolBin):Boolean;
{Complejidad: O(1)}
BEGIN
  EsArbolBinVacio:= arbol.numNodos = 0;
END;
```

```
FUNCTION EsArbolBinPosVacia(arbol:TipoArbolBin;
pos:TipoArbolBinPos):Boolean;
{Complejidad: O(1)}
BEGIN
  EsArbolBinPosVacia:= (arbol.numNodos = 0) or (pos = 0);
END;
```

---

---

---

---

---

---

---

---

```
PROGRAM usaArbolS;
USES
  CRT,ArBiSTAD,ElemTAD;
```

```
PROCEDURE DibujarArbolBin(arbol:TipoArbolBin;i:TipoArbolBinPos);
BEGIN
  IF NOT EsArbolBinPosVacia(arbol,i) THEN
    BEGIN
      Write('[' , Raiz(arbol,i),',',HijoIzdo(arbol,i),',',HijoDcho(arbol,i),']');
      DibujarArbolBin(arbol, HijoIzdo(arbol,i));
      DibujarArbolBin(arbol,HijoDcho(arbol,i));
    END
  END;
```

---

---

---

---

---

---

---

---

```

VAR
    vacio, izqdo, drcho, arbol, arbol1: TipoArbolBin;
    elem: TipoElemento;
BEGIN
    ClrScr;
    (* [A,[B,[C],[D]],,[E,[F]] <--> A(B(C)(D))(E (F)()) *)
    CrearArbolBinVacio(vacio);
    CrearArbolBinVacio(arbol);
    CrearArbolBinVacio(izqdo);
    CrearArbolBinVacio(drcho);

    ConstruirArbolBin(vacio, 'C', vacio, izqdo);
    ConstruirArbolBin(vacio, 'D', vacio, drcho);
    ConstruirArbolBin(izqdo, 'B', drcho, arbol);

    CrearArbolBinVacio(izqdo);
    CrearArbolBinVacio(drcho);
    ConstruirArbolBin(vacio, 'F', vacio, izqdo);
    ConstruirArbolBin(izqdo, 'E', vacio, drcho);

    ConstruirArbolBin(arbol, 'A', drcho, arbol);
    Writeln;
    DibujarArbolBin(arbol, 1);
END.

```

---

---

---

---

---

---

---

---



### 3.2.4 Implementaciones del TAD TipoArbolBinario

- Implementación estática (Heap's)
  - ▲ Árbol binario casi completo
  - ▲ Se usa para implementar Colas de Prioridad. (con la relación de orden "el nodo padre es menor(mayor) que los nodos hijos").

```

CONST
    MAXELEM= ;
TYPE
    TipoArbolBinPos = 0..MAXELEM;
    TipoArbolBin = RECORD
        nodos: ARRAY [0..MAXELEM] OF TipoElemento;
        numNodos: 0..MAXELEM;
    END;

```

EDI 2002/03

41

---

---

---

---

---

---

---

---

```

UNIT ElemTAD;
INTERFACE
TYPE
    TipoElemento = Char;

FUNCTION EsElementoVacio(e1:TipoElemento):Boolean;
FUNCTION CrearElementoVacio:TipoElemento;

IMPLEMENTATION
FUNCTION EsElementoVacio(e1:TipoElemento):Boolean;
BEGIN
    EsElementoVacio:=e1=chr(0);
END;

FUNCTION CrearElementoVacio:TipoElemento;
BEGIN
    CrearElementoVacio:=chr(0)
END;

END.

```

---

---

---

---

---

---

---

---

UNIT ArBiHTAD;

INTERFACE

USES ElemTAD;

(\* DECLARACION TIPO. PRIVADO \*)

CONST

MAXELEM = 100;

TYPE

TipoArbolBinPos = 0..MAXELEM;

TipoArbolBin = RECORD

memoria: ARRAY [0..MAXELEM] OF TipoElemento;

numNodos: 0..MAXELEM;

END;

---

---

---

---

---

---

---

---

IMPLEMENTATION

(\* OPERACIONES AUXILIARES \*)

PROCEDURE InsertarElemento(VAR arbol:TipoArbolBin;

pos:TipoArbolBinPos;e:TipoElemento);

BEGIN

IF pos>MAXELEM THEN

Writeln('ERROR: No hay espacio para almacenar el nodo ',e)

ELSE

arbol.memoria[pos]:=e

END;

---

---

---

---

---

---

---

---

(\* OPERACIONES AUXILIARES \*)

PROCEDURE InsertarNodo(posOrigenI,posOrigenD,posDest:TipoArbolBinPos;

elem:TipoElemento; arbolOrigenI,arbolOrigenD:TipoArbolBin;

VAR arbolDestino:TipoArbolBin);

BEGIN

arbolDestino.numNodos:=arbolDestino.numNodos+1;

InsertarElemento(arbolDestino, posDest, elem);

IF EsElementoVacio(arbolOrigenI.Memoria[posOrigenI]) THEN

InsertarElemento(arbolDestino,posDest\*2,CrearElementoVacio)

ELSE

InsertarNodo( posOrigenI\*2, posOrigenI\*2+1, posDest\*2,

arbolOrigenI.Memoria[posOrigenI],arbolOrigenI,arbolOrigenI,arbolDestino);

IF EsElementoVacio(arbolOrigenD.Memoria[posOrigenD]) THEN

InsertarElemento(arbolDestino,posDest\*2+1,CrearElementoVacio)

ELSE

InsertarNodo(posOrigenD\*2, posOrigenD\*2+1, posDest\*2+1,

arbolOrigenD.Memoria[posOrigenD],arbolOrigenD,arbolOrigenD,arbolDestino);

END;

---

---

---

---

---

---

---

---

(\*\*\*\*\*)

(\* OPERACIONES CONSTRUCTORAS GENERADORAS \*)

```
PROCEDURE CrearArbolBinVacio (VAR arbol:TipoArbolBin);
{Complejidad: O(1)}
BEGIN
    arbol.numNodos := 0;
    arbol.memoria[0]:=CrearElementoVacio;
END;
```

---

---

---

---

---

---

---

```
PROCEDURE ConstruirArbolBin(izqdo:TipoArbolBin;
    e: TipoElemento;
    drcho:TipoArbolBin;
    VAR arbol:TipoArbolBin);
{Complejidad: O(n)}
VAR
    hizq,hdrcho:TipoArbolBinPos;
    arbolAux:TipoArbolBin;

BEGIN
    CrearArbolBinVacio(arbolAux);
    hizq:=1;
    hdrcho:=1;

    IF EsArbolBinVacio(izqdo) THEN
        hizq:=0;

    IF EsArbolBinVacio(drcho) THEN
        hdrcho:=0;

    InsertarNodo(hizq,hdrcho,1,e, izqdo,drcho,arbolAux);

    arbol:=arbolAux;
END;
```

---

---

---

---

---

---

---

(\* OPERACIONES OBSERVADORAS SELECTORAS \*)

```
FUNCTION Raiz(arbol: TipoArbolBin; pos:TipoArbolBinPos): TipoElemento;
{Complejidad: O(1)}
BEGIN
    IF NOT EsArbolBinVacio(arbol) THEN
        IF pos>MAXELEM THEN
            Raiz:=arbol.Memoria[0]
        ELSE
            Raiz:=arbol.Memoria[pos]
    END;

FUNCTION HijoIzdo(arbol: TipoArbolBin; pos:TipoArbolBinPos):
    TipoArbolBinPos;
{Complejidad: O(1)}
BEGIN
    IF NOT EsArbolBinVacio(arbol) THEN
        IF (pos*2)>MAXELEM THEN
            HijoIzdo:=0
        ELSE
            HijoIzdo:=pos*2
    END;
```

---

---

---

---

---

---

---



```

FUNCTION HijoDcho(arbol: TipoArbolBin; pos:TipoArbolBinPos):
TipoArbolBinPos;
{Complejidad: O(1)}
BEGIN
  IF NOT EsArbolBinVacio(arbol) THEN
    IF (pos*2+1)>MAXELEM THEN
      HijoDcho:=0
    ELSE
      HijoDcho:=pos*2+1
    END;

  FUNCTION NumNodos(arbol: TipoArbolBin): TipoArbolBinPos;
  {Complejidad: O(1)}
  BEGIN
    NumNodos:=arbol.numNodos
  END;

```

---

---

---

---

---

---

---

---

```

(* OPERACIONES OBSERVADORAS NO SELECTORAS *)

FUNCTION EsArbolBinVacio(arbol:TipoArbolBin):Boolean;
{Complejidad: O(1)}
BEGIN
  EsArbolBinVacio:= arbol.numNodos = 0;
END;

FUNCTION EsArbolBinPosVacia(arbol:TipoArbolBin;
pos:TipoArbolBinPos):Boolean;
{Complejidad: O(1)}
BEGIN
  EsArbolBinPosVacia:= (arbol.numNodos = 0) or
  EsElementoVacio(arbol.memoria[pos]);
END;

END.

```

---

---

---

---

---

---

---

---



### 3.2.5 Programación con árboles binarios

#### ● Recorridos

- ▲ Se recorre los nodos del árbol en un cierto orden:
  - ↓ Preorden: raíz, hijo izqdo, hijo drcho
  - ↓ Inorden: hijo izqdo, raíz, hijo drcho
  - ↓ Postorden: hijo izqdo, hijo drcho, raíz
- ▲ Determinar los nodos hoja del árbol
  - ↓ Frontera

---

---

---

---

---

---

---

---



### 3.2.5 Programación con árboles binarios

ESPECIFICACION RecorridosArbolesBinarios

USA ArbolesBinarios, Listas

PARAMETROS GENERICOS

TIPOS TipoElemento

FIN PARAMETROS GENERICOS

OPERACIONES

(\* OBSERVADORAS NO SELECTORAS \*)

Preorden: TipoArbolBin -> TipoLista(TipoElemento)

Inorden: TipoArbolBin -> TipoLista(TipoElemento)

PostOrden: TipoArbolBin -> TipoLista(TipoElemento)

Frontera: TipoArbolBin -> TipoLista(TipoElemento)

EDI 2002/03

52

---

---

---

---

---

---

---

---



### 3.2.5 Programación con árboles binarios

VARIABLES

r: TipoElemento;

i, d : TipoArbolBin;

ECUACIONES

(\* Observadoras no selectoras \*)

Preorden(CrearArbolBinVacio)= CrearVacia

Preorden(ConstruirArbolBin(i,r,d))=

Construir(r, Concatenar(Preorden(i), Preorden(d)))

Inorden(CrearArbolBinVacio)= CrearVacia

Inorden(ConstruirArbolBin(i,r,d))=

Concatenar(InOrden(i), Construir(r, Inorden(d)))

EDI 2002/03

53

---

---

---

---

---

---

---

---



### 3.2.5 Programación con árboles binarios

(\* Cont. Observadoras no selectoras \*)

Postorden(CrearArbolBinVacio)= CrearVacia

Postorden(ConstruirArbolBin(i,r,d))=

InsertarFinal(r, Concatenar(Postorden(i), Postorden(d)))

Frontera(CrearArbolBinVacio)= CrearVacia

Frontera(ConstruirArbolBin(i,r,d))=

SI EsArbolBinVacio(i) Y EsArbolBinVacio(d)

Construir(r, CrearVacia)

| Concatenar( Frontera(i), Frontera(d))

FIN ESPECIFICACIÓN

EDI 2002/03

54

---

---

---

---

---

---

---

---

UNIT ArbRcTAD;

INTERFACE

USES ArBinTAD, ListaTAD, ElemTAD;

(\* OPERACIONES OBSERVADORAS NO SELECTORAS \*)

PROCEDURE Preorden(arbol:TipoArbolBin; VAR lista:TipoLista);

{PRE: Cierto}

{POST: devuelve una lista de nodos. (raiz + izqdo + drcho)}

{EXCEPCION: No hay espacio para crear la lista}

PROCEDURE Inorden(arbol:TipoArbolBin; VAR lista:TipoLista);

{PRE: Cierto}

{POST: devuelve una lista de nodos. (izqdo + raiz + drcho)}

{EXCEPCION: No hay espacio para crear la lista}

PROCEDURE Postorden(arbol:TipoArbolBin; VAR lista:TipoLista);

{PRE: Cierto}

{POST: devuelve una lista de nodos. (izqdo + drcho + raiz)}

{EXCEPCION: No hay espacio para crear la lista}

PROCEDURE Frontera(arbol:TipoArbolBin; VAR lista:TipoLista);

{PRE: Cierto}

{POST: devuelve una lista con las hojas del arbol}

{EXCEPCION: No hay espacio para crear la lista}

IMPLEMENTATION

(\* OPERACIONES OBSERVADORAS NO SELECTORAS \*)

PROCEDURE Preorden(arbol:TipoArbolBin; VAR lista:TipoLista);

{Complejidad:  $O(n)$ , siendo  $n$  el número de nodos de "arbol"}

VAR

preordenHI, preordenHD: TipoLista;

BEGIN

ListaTAD.CrearVacia(lista);

IF NOT EsArbolBinVacio(arbol) THEN

BEGIN

ListaTAD.CrearVacia(preordenHI);

ListaTAD.CrearVacia(preordenHD);

Preorden(HijoIzdo(arbol), preordenHI);

Preorden(HijoDcho(arbol), preordenHD);

ListaTAD.Concatenar(preordenHI, preordenHD, lista);

ListaTAD.Construir(Raiz(arbol), lista);

(\* lista no reutiliza la memoria de "preordenHI" y  
"preordenHD". Hay que liberarla \*)

ListaTAD.DestruirLista(preordenHI);

ListaTAD.DestruirLista(preordenHD);

END

END;

PROCEDURE Inorden(arbol:TipoArbolBin; VAR lista:TipoLista);

{Complejidad:  $O(n)$ , siendo  $n$  el número de nodos de "arbol"}

VAR

listaAux: TipoLista;

BEGIN

ListaTAD.CrearVacia(lista);

IF NOT EsArbolBinVacio(arbol) THEN

BEGIN

ListaTAD.CrearVacia(listaAux);

Inorden(HijoIzdo(arbol), lista);

Inorden(HijoDcho(arbol), listaAux);

ListaTAD.Construir(Raiz(arbol), listaAux);

ListaTAD.Concatenar(lista, listaAux, lista);

(\* lista no reutiliza la memoria de "inorden". Hay que liberarla \*)

ListaTAD.DestruirLista(listaAux);

END

END;

```

PROCEDURE Postorden(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad: O(n), siendo n el número de nodos de "arbol"}
VAR
    postOrdenHI,postOrdenHD, aux: TipoLista;
BEGIN
    ListaTAD.CrearVacía(lista);
    IF NOT EsArbolBinVacio(arbol) THEN
        BEGIN
            ListaTAD.CrearVacía(postOrdenHI); ListaTAD.CrearVacía(postOrdenHD);
            ListaTAD.CrearVacía(aux);

            Postorden(HijoIzdo(arbol),postOrdenHI);
            Postorden(HijoDcho(arbol),postOrdenHD);
            ListaTAD.Concatenar(postOrdenHI,postOrdenHD,lista);
            ListaTAD.Construir(Raiz(arbol),aux);
            ListaTAD.Concatenar(lista,aux,lista);

            (* lista no reutiliza la memoria de "postOrdenHI", postOrdenHD, aux.
            Hay que liberarla *)
            ListaTAD.DestruirLista(postOrdenHI);
            ListaTAD.DestruirLista(postOrdenHD);
            ListaTAD.DestruirLista(aux);
        END
    END;
END;

```

---

---

---

---

---

---

---

---

```

PROCEDURE Frontera(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad: O(n), siendo n el número de nodos de "arbol"}
VAR
    fronteraHI,fronteraHD: TipoLista;
BEGIN
    ListaTAD.CrearVacía(lista);
    IF NOT EsArbolBinVacio(arbol) THEN
        BEGIN
            IF EsArbolBinVacio(HijoIzdo(arbol)) AND EsArbolBinVacio(HijoDcho(arbol)) THEN
                ListaTAD.Construir(Raiz(arbol),lista)
            ELSE
                BEGIN
                    ListaTAD.CrearVacía(fronteraHI); ListaTAD.CrearVacía(fronteraHD);

                    Frontera(HijoIzdo(arbol),fronteraHI);
                    Frontera(HijoDcho(arbol),fronteraHD);
                    ListaTAD.Concatenar(fronteraHI,fronteraHD,lista);
                END
            END;
        END
    END;
END;
END.

```

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

- Extensión Árboles binarios
  - ▲ Propiedades respecto al número de nodos:
    - ↓ NumNodos
    - ↓ NumHojas
    - ↓ Profundidad
    - ↓ Compensado
    - ↓ Nivel Elemento (elem)
  - ▲ Transformación árbol
    - ↓ Espejo

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

ESPECIFICACION ArbolesBinariosExtension

USA ArbolesBinarios

PARAMETROS GENERICOS

TIPOS TipoElemento

FIN PARAMETROS GENERICOS

OPERACIONES

(\* OBSERVADORAS NO SELECTORAS \*)

NumNodos: TipoArbolBin  $\rightarrow$  Natural

NumHojas: TipoArbolBin  $\rightarrow$  Natural

Profundidad: TipoArbolBin  $\rightarrow$  Natural

Compensado: TipoArbolBin  $\rightarrow$  Booleano

NivelElemento: TipoArbolBin x TipoElemento  $\rightarrow$  Natural

(\* CONSTRUCTORAS NO GENERADORAS \*)

Espejo: TipoArbolBin  $\rightarrow$  TipoArbolBin

EDI 2002/03

61

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

VARIABLES

e, r: TipoElemento;

i, d: TipoArbolBin;

ECUACIONES

(\* Observadoras no selectoras \*)

NumNodos(CrearArbolBinVacio) = 0

NumNodos(ConstruirArbolBin(i, r, d)) =

1 + NumNodos(i) + NumNodos(d)

NumHojas(CrearArbolBinVacio) = 0

NumHojas(ConstruirArbolBin(i, r, d)) =

SI EsArbolBinVacio(i) Y EsArbolBinVacio(d)  $\rightarrow$

1

| NumHojas(i) + NumHojas(d)

EDI 2002/03

62

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

(\* Cont. Observadoras no selectoras \*)

Profundidad(CrearArbolBinVacio) = 0

Profundidad(ConstruirArbolBin(i, r, d)) = 1 + Maximo(Profundidad(i), Profundidad(d))

Compensado(CrearArbolBinVacio) = CIERTO

Compensado(ConstruirArbolBin(i, r, d)) = NumNodos(i) = NumNodos(d)

NivelElemento(CrearArbolBinVacio, e) = 0

NivelElemento(ConstruirArbolBin(i, r, d), e) =

SI  $e=r \rightarrow$

1

| SI NivelElemento(i, e) > 0 Y NivelElemento(d, e) > 0  $\rightarrow$

1 + Minimo(NivelElemento(i, e), NivelElemento(d, e))

| 1 + Maximo(NivelElemento(i, e), NivelElemento(d, e))

EDI 2002/03

63

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

(\* CONSTRUCTORAS NO GENERADORAS \*)

```
Espejo(CrearArbolBinVacio)= CrearArbolBinVacio
Espejo(ConstruirArbolBin(i,r,d))=
  ConstruirArbolBin(Espejo(d) , r , Espejo(i) )
```

FIN ESPECIFICACIÓN

EDI 2002/03

64

---

---

---

---

---

---

---

---

```
UNIT NaturTAD;
INTERFACE
  TYPE Natural= 0..MAXINT;
  Function Maximo(a,b:Natural):Natural;
  Function Minimo(a,b:Natural):Natural;

IMPLEMENTATION
  Function Maximo(a,b:Natural):Natural;
  BEGIN
    IF (a>=b) THEN
      Maximo:= a
    ELSE
      Maximo:= b
    END;

  Function Minimo(a,b:Natural):Natural;
  BEGIN
    IF (a<=b) THEN
      Minimo:= a
    ELSE
      Minimo:= b
    END;
  END.
```

---

---

---

---

---

---

---

---

```
UNIT ArBExTAD;
INTERFACE
  USES ArBinTAD, ElemTAD , NaturTAD;

  (* OPERACIONES OBSERVADORAS NO SELECTORAS *)
  FUNCTION NumNodos(arbin: TipoArbolBin):Natural;
  {PRE: Cierto}
  {POST:
    resultado = 0 si EsArbolBinVacio(arbin)
    resultado = 1 + NumNodos(HijoIzdo(arbin)) +
    NumNodos(HijoDcho(arbin)) e.o.c.}

  FUNCTION NumHojas(arbin: TipoArbolBin):Natural;
  {PRE: Cierto}
  {POST:
    resultado = 0 si EsArbolBinVacio(arbin)
    resultado = 1 si EsArbolBinVacio(HijoIzdo(arbin)) and
    EsArbolBinVacio(HijoDcho(arbin))
    resultado = NumHojas(HijoIzdo(arbin)) +
    NumHojas(HijoDcho(arbin)) e.o.c. }
```

---

---

---

---

---

---

---

---

```

FUNCTION Profundidad(arbin: TipoArbolBin):Natural;
{PRE: Cierto}
{POST:
    resultado = 0    si EsArbolBinVacio(arbin)
    resultado = 1 +
    Maximo(Profundidad(HijoIzdo(arbin)), Profundidad(HijoDcho(arbin))
    e.o.c. }

FUNCTION Compensado(arbin: TipoArbolBin): Boolean;
{PRE: Cierto}
{POST:
    resultado = CIERTO    si EsArbolBinVacio(arbin)
    resultado = NumNodos(HijoIzdo(arbin))=NumNodos(HijoDcho(arbin))
    e.o.c. }

```

---

---

---

---

---

---

---

---

```

FUNCTION NivelElemento(arbin: TipoArbolBin;
e:TipoElemento):Natural;
{PRE: Cierto}
{POST:
    resultado = 0    si EsArbolBinVacio(arbin)
    resultado = 1    si Raiz(arbin)=e
    resultado = 1 + minimo (NivelElemento(e, HijoIzdo(arbin)),
        NivelElemento(e, HijoDcho(arbin)))

        si NivelElemento(e, HijoIzdo(arbin)) > 0 and
        NivelElemento(e, HijoDcho(arbin)) > 0

    resultado = 0    si (NivelElemento(e, HijoIzdo(arbin)) = 0 and
        NivelElemento(e, HijoDcho(arbin)) = 0)

    resultado = 1 + maximo(NivelElemento(e, HijoIzdo(arbin)),
        NivelElemento(e, HijoDcho(arbin))) e.o.c }

```

---

---

---

---

---

---

---

---

```

(* OPERACIONES CONSTRUCTORA NO GENERADORAS *)
PROCEDURE Espejo(arbin: TipoArbolBin; VAR imagen: TipoArbolBin);
{PRE: Cierto}
{POST:
    CrearArbolVacio(imagen) si EsArbolBinVacio(arbin)
    Espejo(HijoIzdo(arbin), imagen_hi) and
    Espejo(HijoDcho(arbin), imagen_hd) and
    ConstruirArbolBin(imagen_hd, Raiz(arbin), imagen_hi, imagen) e.o.c. }
{ EXCEPCION: 'MemoriaAgotada' si no queda memoria dinámica }

```

#### IMPLEMENTATION

```

(* OPERACIONES OBSERVADORAS NO SELECTORAS *)

FUNCTION NumNodos(arbin: TipoArbolBin):Natural;
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
BEGIN
    IF EsArbolBinVacio (arbin) THEN
        NumNodos:= 0
    ELSE
        NumNodos:= 1 + NumNodos (HijoIzdo (arbin)) + NumNodos (HijoDcho(arbin))
    END;

```

---

---

---

---

---

---

---

---

```

FUNCTION NumHojas(arbin: TipoArbolBin):Natural;
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
BEGIN
  IF EsArbolBinVacio(arbin) THEN
    NumHojas:= 0
  ELSE IF EsArbolBinVacio(HijoIzdo(arbin)) AND
    EsArbolBinVacio(HijoDcho(arbin)) THEN
    NumHojas:= 1
  ELSE
    NumHojas:= NumHojas(HijoIzdo(arbin)) + NumHojas(HijoDcho(arbin));
END;

```

```

FUNCTION Profundidad(arbin: TipoArbolBin):Natural;
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
BEGIN
  IF EsArbolBinVacio (arbin) THEN
    Profundidad:= 0
  ELSE
    Profundidad:= 1 + NaturTAD.Maximo(Profundidad(HijoIzdo(arbin)),
      Profundidad(HijoDcho(arbin)));
END;

```

```

FUNCTION Compensado(arbin: TipoArbolBin): Boolean;
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
BEGIN
  IF EsArbolBinVacio (arbin) THEN
    Compensado:= True
  ELSE
    Compensado:= NumNodos(HijoIzdo(arbin)) =
      NumNodos(HijoDcho(arbin))
END;

```

```

FUNCTION NivelElemento(arbin: TipoArbolBin; e:TipoElemento):Natural;
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
VAR
  nivel_hi,nivel_hd:Natural;
BEGIN
  IF EsArbolBinVacio (arbin) THEN
    NivelElemento:= 0
  ELSE IF Raiz(arbin) = e THEN
    NivelElemento:= 1
  ELSE
    BEGIN
      nivel_hi := NivelElemento(HijoIzdo (arbin),e);
      nivel_hd := NivelElemento(HijoDcho (arbin), e);
      IF (nivel_hi=0) AND (nivel_hd=0) THEN
        NivelElemento:= 0
      ELSE IF (nivel_hi > 0) AND (nivel_hd > 0) THEN
        NivelElemento:= 1 + NaturTAD.Minimo(nivel_hi,nivel_hd)
      ELSE
        NivelElemento:= 1 + NaturTAD.Maximo(nivel_hi,nivel_hd)
      END
    END
  END;

```



(\* OPERACIONES CONSTRUCTORAS NO GENERADORAS \*)

```
PROCEDURE Espejo(arbin: TipoArbolBin; VAR imagen: TipoArbolBin);
{Complejidad: O(n), siendo n el número de nodos de 'arbin'}
VAR
    imagen_hi, imagen_hd: TipoArbolBin;
BEGIN
    IF EsArbolBinVacio(arbin) THEN
        CrearArbolBinVacio(imagen)
    ELSE
        BEGIN
            CrearArbolBinVacio(imagen_hi);
            CrearArbolBinVacio(imagen_hd);
            Espejo (HijoIzdo (arbin), imagen_hi);
            Espejo (HijoDcho (arbin), imagen_hd);
            ConstruirArbolBin(imagen_hd, Raiz(arbin), imagen_hi, imagen);

            (* La memoria de 'imagen_hi' y 'imagen_hd' pasa a ser
            propiedad de 'imagen', por lo que no se debe liberar *)
        END
    END;
END.
```

---

---

---

---

---

---

---

---

```
PROGRAM usaArbol;
USES
    CRT, ArbExTAD, ArBinTAD, ElemTAD, NaturTAD;
VAR
    vacio, izqdo, drcho, arbol, imagen: TipoArbolBin;
    elem: TipoElemento;
BEGIN
    ClrScr;
    (* [A,[B,[C],[D]], [E,[F]] <--> A(B(C(D))(E (F()) *)

    CrearArbolBinVacio(vacio);
    CrearArbolBinVacio(arbol);  CrearArbolBinVacio(imagen);
    CrearArbolBinVacio(izqdo);  CrearArbolBinVacio(drcho);

    ConstruirArbolBin(vacio, 'C', vacio, izqdo);  ConstruirArbolBin(vacio, 'D', vacio,
    drcho);  ConstruirArbolBin(izqdo, 'B', drcho, arbol);

    CrearArbolBinVacio(izqdo);  CrearArbolBinVacio(drcho);
    ConstruirArbolBin(vacio, 'F', vacio, izqdo);  ConstruirArbolBin(izqdo, 'E', vacio,
    drcho);
    ConstruirArbolBin(arbol, 'A', drcho, arbol);
    Writeln;
    DibujarArbolBin(arbol);
```

---

---

---

---

---

---

---

---

```
Writeln;
Writeln('Numero de nodos:', NumNodos(arbol));

Writeln('Numero de hojas:', NumHojas(arbol));

Writeln('Profundidad:', Profundidad(arbol));

Writeln('Compensado? ', Compensado(arbol));

Writeln;
Writeln('Nivel Elemento A: ', NivelElemento(arbol, 'A'));
Writeln('Nivel Elemento F: ', NivelElemento(arbol, 'F'));
Writeln('Nivel Elemento Z: ', NivelElemento(arbol, 'Z'));

Writeln;
Espejo(arbol, imagen);
DibujarArbolBin(imagen);

DestruirArbolBin(arbol);
DestruirArbolBin(imagen);

END.
```

---

---

---

---

---

---

---

---



### 3.2.6 Ejercicios

- EVALUACION DE EXPRESIONES ARITMETICAS TOTALMENTE PARENTIZADAS.

Arbol binario de Expresiones (árbol sintáctico) :

- Arbol binario cuyas hojas contienen operandos y los nodos no terminales contienen operadores.

EDI 2002/03

76

---

---

---

---

---

---

---



### 3.2.6 Ejercicios. Implementación No recursiva Recorridos

#### ● Recorridos:

- ▲ En profundidad:
  - ↓ Preorden: raíz, hijo izqdo, hijo drcho
  - ↓ Inorden: hijo izqdo, raíz, hijo drcho
  - ↓ Postorden: hijo izqdo, hijo drcho, raíz
- ▲ En amplitud

EDI 2002/03

77

---

---

---

---

---

---

---

```
PROCEDURE Preorden(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad: O(n), siendo n el número de nodos de "arbol"}
VAR
  listaAux: TipoLista;  p:TipoPila;  aux:TipoArbolBin;
BEGIN
  CrearVacia(lista);
  CrearPilaVacia(p);
  aux:=arbol;
  REPEAT
    IF NOT EsArbolBinVacio(aux) THEN
      BEGIN
        InsertaFinal(Raiz(aux),lista);
        Apilar(HijoDcho(aux),p);
        Apilar(HijoIzdo(aux),p);
      END;
    IF NOT EsPilaVacia(p) THEN
      BEGIN
        Cima(aux,p);
        Desapilar(p);
      END
    UNTIL EsPilaVacia(p) AND EsArbolBinVacio(aux);
  Destruir(p);
END;
```

---

---

---

---

---

---

---

```

PROCEDURE Inorden(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad:  $O(n)$ , siendo n el número de nodos de "arbol"}
VAR
  listaAux: TipoLista;  p:TipoPila;  aux:TipoArbolBin;
BEGIN
  CrearVacia(lista);
  CrearPilaVacia(p);
  aux:=arbol;
  WHILE NOT EsArbolBinVacio(aux) OR NOT EsPilaVacia(p) DO
    IF EsArbolBinVacio(aux) THEN
      BEGIN
        Cima(aux,p);
        Desapilar(p);
        InsertaFinal(Raiz(aux),lista);
        aux:=HijoDcho(aux)
      END
    ELSE
      BEGIN
        Apilar(aux,p);
        aux:=HijoIzdo(aux)
      END;
    Destruir(p);
  END;

```

---

---

---

---

---

---

---

---

```

PROCEDURE Postorden(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad:  $O(n)$ , siendo n el número de nodos de "arbol"}
VAR
  listaAux: TipoLista;  p:TipoPila;  aux:TipoArbolBin;
BEGIN
  CrearVacia(lista);
  CrearPilaVacia(p);
  aux:=arbol;
  REPEAT
    IF NOT EsArbolBinVacio(aux) THEN
      BEGIN
        Construir(Raiz(aux),lista);
        Apilar(HijoDcho(aux),p);
        Apilar(HijoIzdo(aux),p);
      END;

    IF NOT EsPilaVacia(p) THEN
      BEGIN
        Cima(aux,p);
        Desapilar(p);
      END
    UNTIL EsPilaVacia(p) AND EsArbolBinVacio(aux);
  Destruir(p);
END;

```

---

---

---

---

---

---

---

---

```

PROCEDURE Anchura(arbol:TipoArbolBin; VAR lista:TipoLista);
{Complejidad:  $O(n)$ , siendo n el número de nodos de "arbol"}
VAR
  listaAux: TipoLista;  c:TipoCola;  aux:TipoArbolBin;
BEGIN
  CrearVacia(lista);
  IF NOT EsArbolBinVacio(arbol) THEN
    BEGIN
      CrearColaVacia(c);
      aux:=arbol;
      Insertar(arbol,c);
      WHILE NOT EsColaVacia(c) DO
        BEGIN
          PrimeroCola(aux,c); Eliminar(c);
          InsertaFinal(Raiz(aux),lista);
          IF NOT EsArbolBinVacio(HijoIzdo(aux)) THEN
            Insertar(HijoIzdo(aux),c);
          IF NOT EsArbolBinVacio(HijoDcho(aux)) THEN
            Insertar(HijoDcho(aux),c);
        END;
      DestruirCola(c);
    END;
  END;

```

---

---

---

---

---

---

---

---



### 3.2.7 Árboles binarios de búsqueda

#### ● Árboles Binarios de Búsqueda

- ▲ Un árbol binario de búsqueda es un árbol binario donde sus elementos están ordenados en un cierto orden:
  - ↓ Para cada nodo x en el árbol, los valores(claves) de todos los nodos de su subárbol izquierdo son menores que el valor de x, y los valores(claves) de todos los nodos de su subárbol derecho son mayores que el valor del nodo x.
  - ↓ Orden lexicográfico
  - ↓ Orden numérico
- ▲ Una aplicación importante de los árboles binarios es su uso en la búsqueda.

EDI 2002/03

82

---

---

---

---

---

---

---

---



### 3.2.7 Árboles binarios de búsqueda

#### ● Árboles Binarios de Búsqueda

- ▲ Si un árbol binario de búsqueda se recorre en inorden, la lista de las claves visitadas aparece ordenada. (Método de ordenación Tree-Sort)
- ▲ Operaciones:
  - ↓ Pertenece
  - ↓ Mínimo elemento del árbol
  - ↓ Insertar
  - ↓ Eliminar

EDI 2002/03

83

---

---

---

---

---

---

---

---



### 3.2.7 Árboles binarios de búsqueda

#### ESPECIFICACION ArbolesBinariosBusqueda

USA ArbolesBinarios

PARAMETROS GENERICOS

#### OPERACIONES

Mayor: TipoElemento x TipoElemento → Booleano

Igual: TipoElemento x TipoElemento → Booleano

FIN PARAMETROS GENERICOS

#### OPERACIONES

(\* OBSERVADORAS NO SELECTORAS \*)

Pertenece: TipoArbolBin x TipoElemento → Booleano

PARCIAL Mínimo: TipoArbolBin → TipoElemento

(\* CONSTRUCTORAS NO GENERADORAS \*)

Insertar: TipoArbolBin x TipoElemento → TipoArbolBin

Eliminar: TipoArbolBin x TipoElemento → TipoArbolBin

(\* Para trabajar con árboles binarios de búsqueda sólo pueden usarse las constructoras

'CrearArbolBinVacio', 'Insertar' y 'Eliminar' \*)

84

---

---

---

---

---

---

---

---



## 3.2.7 Árboles binarios de búsqueda

### VARIABLES

r,e: TipoElemento;  
i , d : TipoArbolBin;

### ECUACIONES DE DEFINITUD

DEF(Mínimo(ConstruirArbolBin(i,r,d)))

### ECUACIONES

(\* Observadoras no selectoras \*)

Pertenece(CrearArbolBinVacio, e) = FALSO  
Pertenece(ConstruirArbolBin(i,r,d), e) =  
SI Igual(e,r) → CIERTO  
| SI Mayor(e,r) → Pertence(d,e)  
| Pertenece(i, e)

EDI 2002/03

85

---

---

---

---

---

---

---

---



## 3.2.7 Árboles binarios de búsqueda

(\* Observadoras no selectoras \*)

Mínimo(ConstruirArbolBin(i,r,d) ) =  
SI EsArbolBinVacio(i) → r  
| Mínimo(i)

(\* Constructoras no generadoras \*)

Insertar(CrearArbolBinVacio, e) =  
ConstruirArbolBin(CrearArbolBinVacio, e, CrearArbolBinVacio)  
Insertar(ConstruirArbolBin(i,r,d), e) =  
SI Igual(e,r) →  
ConstruirArbolBin(i, r, d)  
| SI Mayor(e,r) →  
ConstruirArbolBin(i, r, Insertar(d,e))  
| ConstruirArbolBin(Insertar(i,e),r,d)

EDI 2002/03

86

---

---

---

---

---

---

---

---



## 3.2.7 Árboles binarios de búsqueda

(\* Cont...Constructoras no generadoras \*)

Eliminar(CrearArbolBinVacio, e) = CrearArbolBinVacio  
Eliminar(ConstruirArbolBin(i,r,d), e) =  
SI Igual(e,r) →  
SI EsArbolBinVacio(d) → i  
| ConstruirArbolBin(i, Mínimo(d), Eliminar(d,Mínimo(d)))  
| SI Mayor(e,r) →  
ConstruirArbolBin(i, r, Eliminar(d,e))  
| ConstruirArbolBin(Eliminar(i,e),r,d)

FIN ESPECIFICACIÓN

EDI 2002/03

87

---

---

---

---

---

---

---

---

```

UNIT ElemTAD;
INTERFACE
TYPE
  TipoClave= Integer;  TipoInfo= Char;

  TipoElemento = RECORD
    clave: TipoClave;
    info: TipoInfo;
  END;

FUNCTION Mayor(e1,e2:TipoElemento):Boolean;
FUNCTION IgualElemento(e1,e2:TipoElemento):Boolean;

IMPLEMENTATION
FUNCTION Mayor(e1,e2:TipoElemento):Boolean;
BEGIN
  Mayor:= e1.clave>e2.clave;
END;

FUNCTION IgualElemento(e1,e2:TipoElemento):Boolean;
BEGIN
  IgualElemento:= e1.clave=e2.clave;
END;
END.

```

---

---

---

---

---

---

---

---

```

(* DECLARACION TIPO. PRIVADO *)
TYPE
  TipoArbolBinBusq = TipoArbolBin;

(* OPERACIONES OBSERVADORAS NO SELECTORAS *)

FUNCTION Pertenece(arbol:TipoArbolBinBusq; ele:TipoElemento): Boolean;
{PRE: Cierto}
{POST: devuelve TRUE si "elemento" esta en el "arbol"}

PROCEDURE Minimo( arbol:TipoArbolBinBusq; VAR ele:TipoElemento);
{PRE: "arbol" no esta vacio}
{POST: devuelve el nodo con valor minimo en la clave}

(* OPERACIONES CONSTRUCTORAS NO GENERADORAS *)
PROCEDURE Insertar(VAR arbol: TipoArbolBinBusq; ele: TipoElemento);
{PRE: Cierto}
{POST: inserta "ele" en "arbol" respetando relación de orden}
{EXCEPCION: "Memoria Agotada" si no queda espacio de almacenamiento}

PROCEDURE Eliminar( VAR arbol: TipoArbolBinBusq; ele: TipoElemento);
{PRE: Cierto}
{POST: elimina "ele" del "arbol"}

```

---

---

---

---

---

---

---

---

```

IMPLEMENTATION
(* OPERACIONES OBSERVADORAS NO SELECTORAS *)

FUNCTION Pertenece(arbol:TipoArbolBinBusq; ele:TipoElemento): Boolean;
{Complejidad:  $O(\log(n))$  -  $O(n)$ }
VAR
  e:TipoElemento;
BEGIN
  IF EsArbolBinVacio(arbol) THEN
    Pertenece:=FALSE
  ELSE
    BEGIN
      Raiz(arbol,e);
      IF IgualElemento(ele,e) THEN
        Pertenece:=TRUE
      ELSE IF Mayor(ele,e) THEN
        Pertenece:=Pertenece(HijoDcho(arbol),ele)
      ELSE
        Pertenece:=Pertenece(HijoIzdo(arbol),ele)
    END;
  END;
END;

```

---

---

---

---

---

---

---

---

```

PROCEDURE Minimo( arbol:TipoArbolBinBusq; VAR
ele:TipoElemento);
{Complejidad:  $O(\log(n))$  -  $O(n)$ }
BEGIN
  IF NOT EsArbolBinVacio(arbol) THEN
    BEGIN
      IF EsArbolBinVacio(HijoIzdo(arbol)) THEN
        Raiz(arbol,ele)
      ELSE
        Minimo(HijoIzdo(arbol), ele)
      END
    END
  END;

```

---

---

---

---

---

---

---

---

```

(* OPERACIONES CONSTRUCTORAS NO GENERADORAS *)
PROCEDURE Insertar( VAR arbol: TipoArbolBinBusq; ele: TipoElemento);
{Complejidad:  $O(\log(n))$  -  $O(n)$ }
VAR
  auxVacio,aux,aux2:TipoArbolBinBusq;  e:TipoElemento;
BEGIN
  CrearArbolBinVacio(auxVacio);  CrearArbolBinVacio(aux);CrearArbolBinVacio(aux2);
  IF EsArbolBinVacio(arbol) THEN
    ConstruirArbolBin(auxVacio, ele, auxVacio, arbol)
  ELSE
    BEGIN
      Raiz(arbol,e);
      IF IgualElemento(e,ele) THEN
        (* No duplicamos elementos *)
      ELSE IF Mayor(ele,e) THEN  BEGIN
        aux:=HijoDcho(arbol); Insertar(aux,ele);
        ConstruirArbolBin(HijoIzdo(arbol), e, aux, aux2)
      END
      ELSE  BEGIN
        aux:=HijoIzdo(arbol); Insertar(aux,ele);
        ConstruirArbolBin(aux, e, HijoDcho(arbol), aux2)
      END;
      arbol:=aux2;
    END  END;

```

---

---

---

---

---

---

---

---

```

PROCEDURE Eliminar( VAR arbol: TipoArbolBinBusq; ele: TipoElemento);
{Complejidad:  $O(\log(n))$  -  $O(n)$ }
VAR
  auxVacio, aux, aux2:TipoArbolBinBusq;  e,e1:TipoElemento;
BEGIN
  CrearArbolBinVacio(auxVacio);  CrearArbolBinVacio(aux);
  IF EsArbolBinVacio(arbol) THEN  (* No se hace nada *)
  ELSE
    BEGIN
      Raiz(arbol,e);
      IF IgualElemento(e,ele) THEN
        IF EsArbolBinVacio(HijoDcho(arbol)) THEN
          BEGIN
            aux2:=arbol; arbol:=HijoIzdo(arbol);
            LiberarNodo(aux2);
          END
        ELSE
          BEGIN
            Minimo(HijoDcho(arbol),e1);
            aux2:=HijoDcho(arbol);
            Eliminar(aux2,e1);
            ConstruirArbolBin(HijoIzdo(arbol), e1,aux2 , aux);
            arbol:=aux;
          END
        END
      END
    END

```

---

---

---

---

---

---

---

---

```

ELSE IF Mayor(ele,e) THEN
  BEGIN
    aux2:=HijoDcho(arbol);
    Eliminar(aux2,ele);
    ConstruirArbolBin(HijoIzdo(arbol), e,aux2 , aux);
    arbol:=aux;
  END
ELSE (* el elemento es menor que la raiz *)
  BEGIN
    aux2:=HijoIzdo(arbol);
    Eliminar(aux2,ele);
    ConstruirArbolBin(aux2, e, HijoDcho(arbol) , aux);
    arbol:=aux;
  END
END (*fin arbol no vacío *)
END;

```

---

---

---

---

---

---

---

---



### 3.2.7 Árboles binarios de búsqueda equilibrados (AVL)

- Adelson-Velskii y Landis

- ▲ Operaciones:

- ↓ Insertar

- ↓ Eliminar

- ▲ Factor de equilibrio de un nodo:

- Altura(subarbol derecho) – Altura(subarbol Izquierdo)

- ↓ Su valor oscila entre -1 , 0 , 1

---

---

---

---

---

---

---

---



### 3.2.7 Árboles binarios de búsqueda equilibrados (AVL)

- Árboles de FIBONACCI

- ▲ Son los árboles AVL menos densos

- ▲ El número de nodos para un árbol de altura h es:

- ↓  $N_0 = 0$

- ↓  $N_1 = 1$

- ↓  $N_h = N_{h-1} + N_{h-2} + 1$  para todo  $h \geq 2$

---

---

---

---

---

---

---

---