

Tema 7: Árboles

ESTRUCTURAS DE DATOS

Contenidos

- Definiciones
- Conceptos de Árboles Binarios
- Especificación algebraica
- Implementaciones
- Programación con Árboles Binarios
- Árboles Binarios de Búsqueda
 - Introducción
 - Especificación e Implementación
 - Árboles equilibrados

Definiciones

- Un árbol es una **estructura** que organiza sus elementos **formando jerarquías entre sus elementos** (nodos)
- Pueden representar relaciones
- Relación clave: padre/hijo (ascendiente/descendiente)
- Nodos unidos por aristas o ramas (*edges*)
- Nodos sin descendientes: Hojas o nodos terminales
- Nodos: raíz, padre, hijo, hermano, hoja

Definiciones

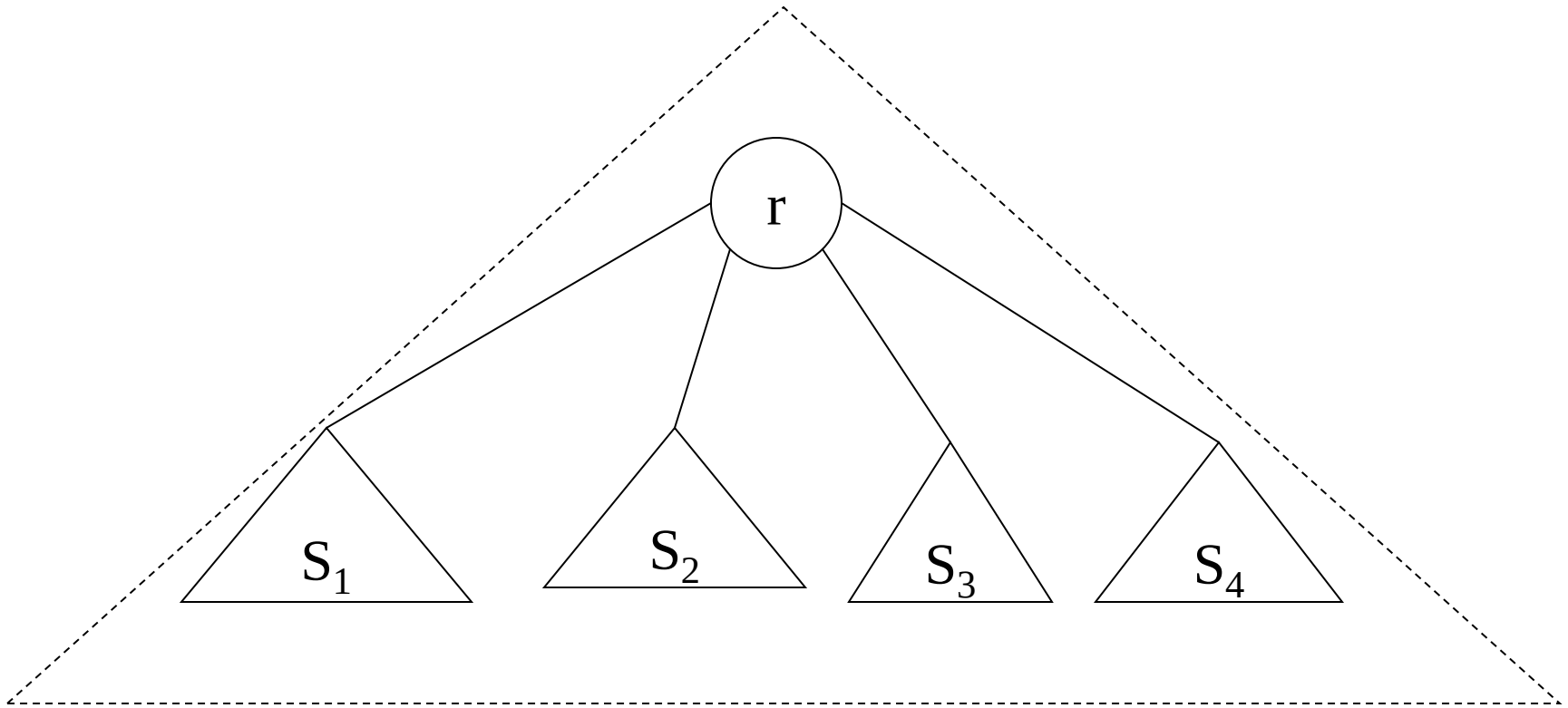
- Un árbol enraizado tiene un nodo raíz (*root*) y otros nodos conectados en parejas por ramas
- Cada nodo, excepto el raíz, es conectado por una única rama a su nodo padre (antecesor).
- Cada nodo, excepto las hojas (*leaf node*), tiene uno o más hijos

Definiciones

- Definición recursiva:
 - Un árbol se compone de una colección de nodos
 - Esta colección puede estar vacía (árbol vacío)
 - Nodo raíz y cero o más subárboles
 - Cada raíz de los subárboles es hijo del nodo raíz inicial y está conectado mediante una rama

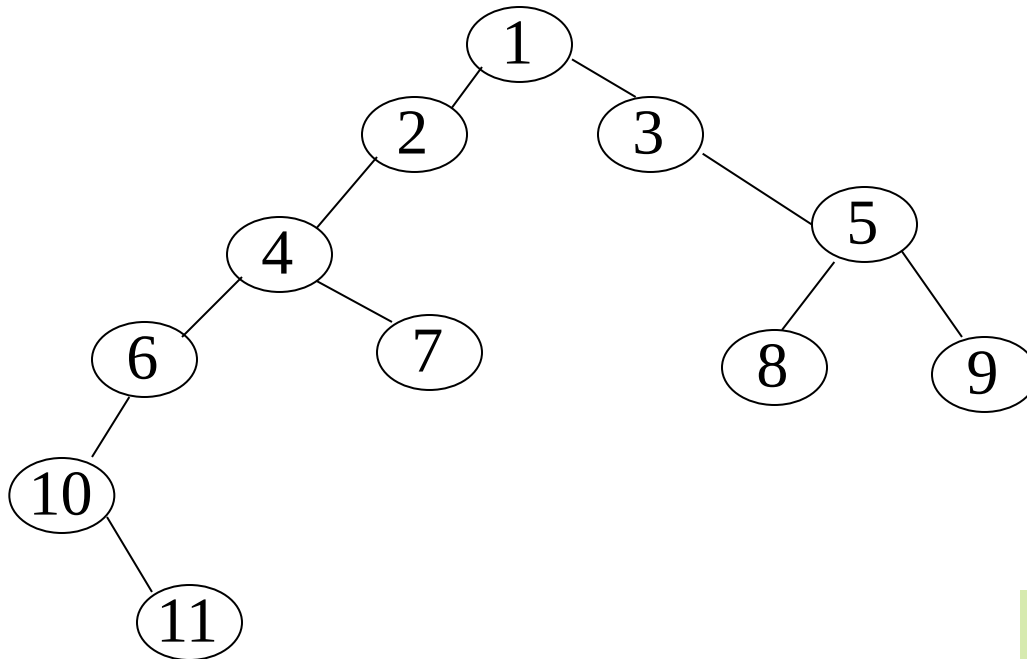
Definiciones

Definición recursiva:



Definiciones

Camino: Secuencia de nodos $n_1, n_2, n_3, \dots, n_k$ siendo n_i padre de n_{i+1} , conectados dentro de un árbol



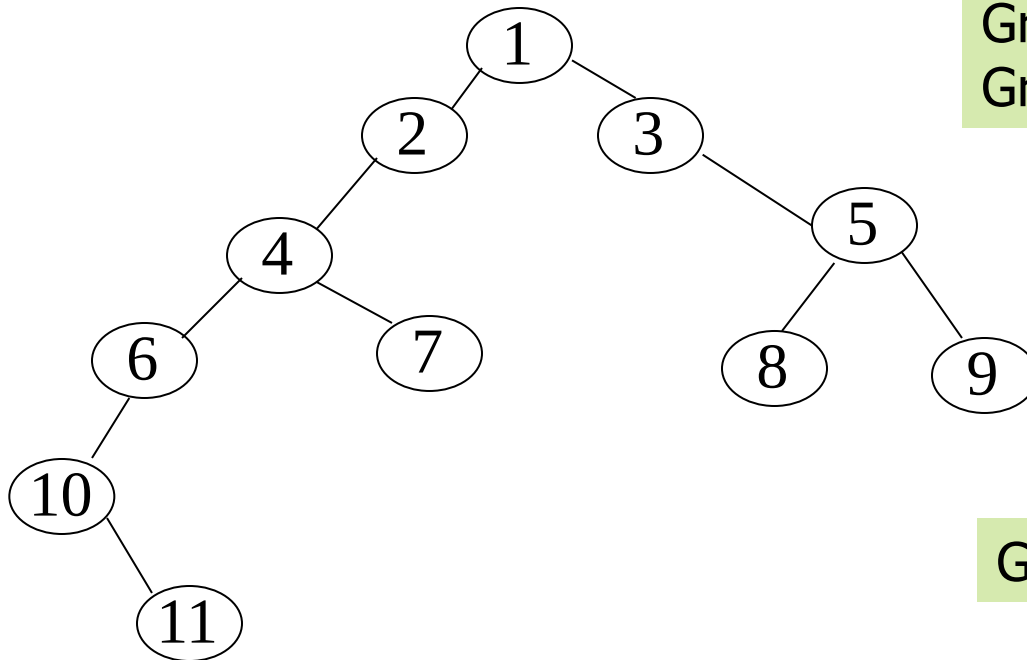
Camino entre el nodo 2 y el 7: **2, 4 y 7**

Longitud del camino entre el nodo 2 y el 7: **2**

Longitud de un camino: número de aristas que unen el camino

Definiciones

Grado (*aridad*) de un **nodo**: número de hijos del nodo



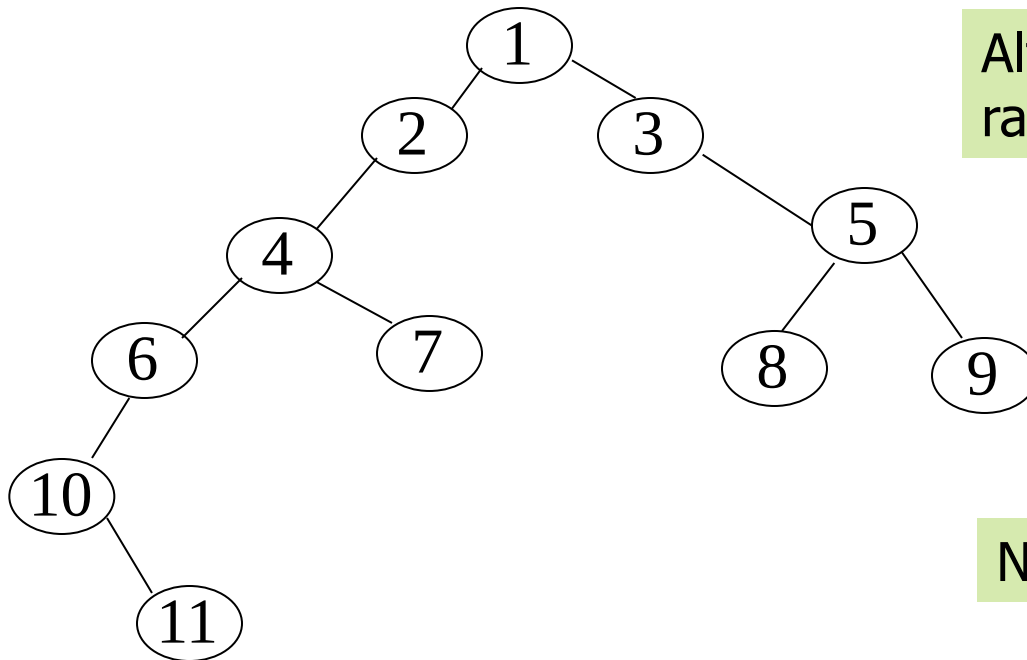
Grado del nodo 4: **2**
Grado del nodo 7: **0**

Grado del árbol: **2**

Grado (*aridad*) de un **árbol**: Máximo grado de sus nodos

Definiciones

Altura o profundidad de un **árbol**: Longitud del camino desde el nodo raíz a su hoja más lejana (o # nodos)



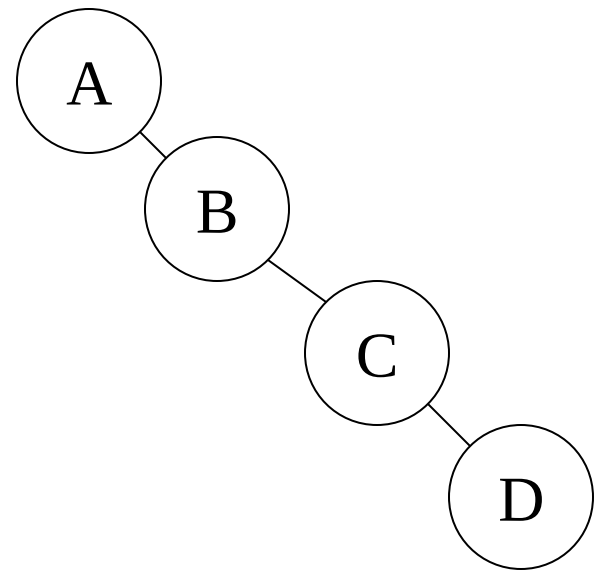
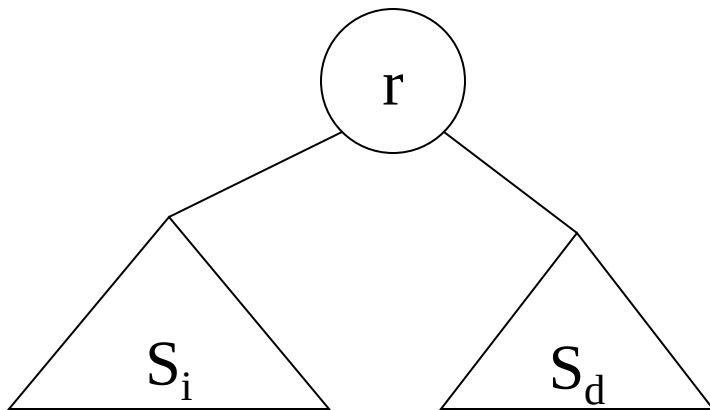
Altura del árbol: **6** (si la raíz tiene altura 1)

Nivel del nodo 6: **4**

Nivel o profundidad de un **nodo**: longitud entre la raíz del árbol y dicho nodo (0 ó 1 para el nodo raíz)

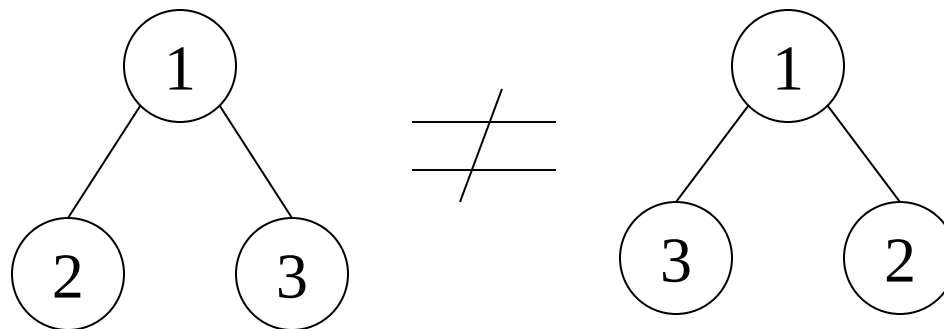
Conceptos de Árboles Binarios

- Árboles Binarios
 - Particularización de un árbol cuyos padres no tienen más de 2 hijos



Conceptos de Árboles Binarios

- Hijos de un nodo son: hijo izquierdo, hijo derecho



- Profundidad del árbol binario promedio es mucho menor que $n \rightarrow$ árboles binarios de búsqueda

Especificación algebraica

ESPECIFICACIÓN ArbolesBinarios

PARÁMETROS GENÉRICOS

TIPOS TipoElemento

FIN PARAMETROS

TIPOS TipoArbolBin

OPERACIONES

(* constructoras generadoras *)

CrearArbolBinVacio: $\rightarrow$ TipoArbolBin

ConstruirArbolBin: TipoArbolBin x TipoElemento x TipoArbolBin $\rightarrow$ TipoArbolBin

Especificación algebraica

(* observadoras selectoras *)

PARCIAL Raiz: TipoArbolBin $\rightarrow$ TipoElemento

PARCIAL HijoIzdo: TipoArbolBin $\rightarrow$ TipoArbolBin

PARCIAL HijoDcho: TipoArbolBin $\rightarrow$ TipoArbolBin

(* observadoras no selectoras *)

EsArbolBinVacio: TipoArbolBin $\rightarrow$ Booleano

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)))

Especificación algebraica

ECUACIONES

(* observadoras selectoras *)

$\text{Raiz}(\text{ConstruirArbolBin}(i, r, d)) = r$

$\text{Hijolq}(\text{ConstruirArbolBin}(i, r, d)) = i$

$\text{HijoDer}(\text{ConstruirArbolBin}(i, r, d)) = d$

(* observadoras no selectoras *)

$\text{EsArbolBinVacio}(\text{CrearArbolBinVacio}) = \text{CIERTO}$

$\text{EsArbolBinVacio}(\text{ConstruirArbolBin}(i, r, d)) = \text{FALSO}$

FIN ESPECIFICACIÓN

Especificación algebraica

ESPECIFICACIÓN ArbolesBinariosExtension

USA ArbolesBinarios

OPERACIONES

(* observadoras no selectoras *)

NumNodos: TipoArbolBin $\rightarrow$ Natural

NumHojas: TipoArbolBin $\rightarrow$ Natural

Altura: TipoArbolBin $\rightarrow$ Natural

Compensado: TipoArbolBin $\rightarrow$ Booleano

NivelElemento: TipoElemento x TipoArbolBin $\rightarrow$ Natural

Especificación algebraica

(* constructoras no generadoras *)

Espejo: TipoArbolBin $\rightarrow$ TipoArbolBin

VARIABLES

e, r: TipoElemento;

i, d: TipoArbolBin;



ECUACIONES

(* observadoras no selectoras *)

NumNodos(CrearArbolBinVacio) = 0

NumNodos(ConstruirArbolBin(i, r, d)) = 1 + NumNodos(i) + NumNodos(d)

Especificación algebraica

$\text{NumHojas}(\text{CrearArbolBinVacio}) = 0$

$\text{NumHojas}(\text{ConstruirArbolBin}(i, r, d)) = \text{SI EsArbolBinarioVacio}(i) \text{ Y EsArbolBinarioVacio}(d) \rightarrow$
1

| $\text{NumHojas}(i) + \text{NumHojas}(d)$

$\text{Altura}(\text{CrearArbolBinVacio}) = 0$

$\text{Altura}(\text{ConstruirArbolBin}(i, r, d)) = 1 + \text{Maximo}(\text{Altura}(i), \text{Altura}(d))$

$\text{Compensado}(\text{CrearArbolBinVacio}) = \text{CIERTO}$



$\text{Compensado}(\text{ConstruirArbolBin}(i, r, d)) = (\text{NumNodos}(i) = \text{NumNodos}(d))$

Especificación algebraica

$\text{NivelElemento}(e, \text{CrearArbolBinVacio}) = 0$

$\text{NivelElemento}(e, \text{ConstruirArbolBin}(i, r, d)) = \text{SI } e=r \rightarrow$

1

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

1 + $\text{Minimo}(\text{NivelElemento}(e, i), \text{NivelElemento}(e, d))$

| SI $\text{NivelElemento}(e, i) = 0$ Y $\text{NivelElemento}(e, d) = 0 \rightarrow$

0

| 1 + $\text{Maximo}(\text{NivelElemento}(e, i), \text{NivelElemento}(e, d))$

(* constructoras no generadoras *)

$\text{Espejo}(\text{CrearArbolBinVacio}) = \text{CrearArbolBinVacio}$

$\text{Espejo}(\text{ConstruirArbolBin}(i, r, d)) = \text{ConstruirArbolBin}(\text{Espejo}(d), r, \text{Espejo}(i))$

FIN ESPECIFICACIÓN

Implementaciones

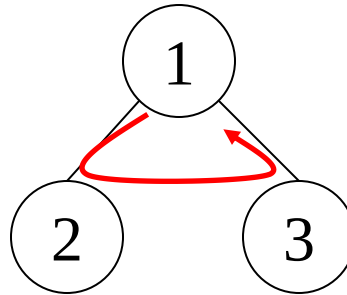
- Mediante punteros y variables dinámicas

```
TApuntdadorArbol = ^TNodo;  
TNodo = RECORD  
    elemento: TipoElemento;  
    izquierdo, derecho: TApuntdadorArbol;  
END;  
TipoArbol = TApuntdadorArbol;
```

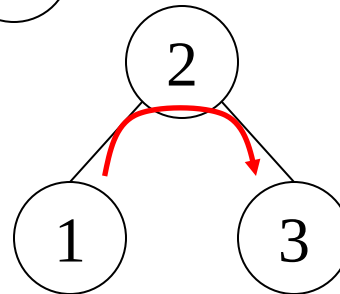
- Mediante vectores
 - Simulando memoria dinámica

Recorridos

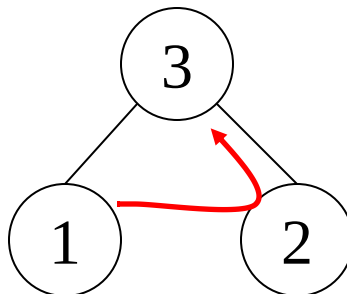
Preorden (RID)



Inorden (IRD)



Postorden (IDR)



Especificación algebraica

ESPECIFICACIÓN RecorridosArbolesBinarios

USA ArbolesBinarios, Listas

OPERACIONES

(* observadoras no selectoras *)

Preorden : TipoArbolBin $\rightarrow$ TipoLista

Inorden : TipoArbolBin $\rightarrow$ TipoLista

Postorden : TipoArbolBin $\rightarrow$ TipoLista

Frontera : TipoArbolBin $\rightarrow$ TipoLista

VARIABLES

r: TipoElemento;

i, d: TipoArbolBin;

Especificación algebraica

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)))

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

Implementación

```
UNIT ABinP;

INTERFACE

USES ELEMTAD;

TYPE

  TipoArbolBin = ^TipoNodo;
  TipoNodo = RECORD
    r: TipoElemento;
    izq, der: TipoArbolBin;
  END;

{ CONSTRUCTORAS GENERADORAS }

PROCEDURE CrearArbolVacio(VAR a: TipoArbolBin);
(* POST: CrearArbolVacio= () *)

PROCEDURE ConstruirArbolBin(VAR a: TipoArbolBin; izq: TipoArbolBin;
  r: TipoElemento; der: TipoArbolBin);
(* POST: a= (izq) r (der) *)
```

Implementación

```
{ OBSERVADORAS NO SELECTORAS }

FUNCTION EsArbolBinVacio(a: TipoArbolBin): Boolean;
{ POST: a = () }

{ OBSERVADORAS SELECTORAS }

PROCEDURE Raiz(a: TipoArbolBin; VAR e: TipoElemento);
{ PRE: a = (izq) r (der)
  POST: e <- r }

PROCEDURE HijoIzq(a: TipoArbolBin; VAR hIzq: TipoArbolBin);
{ PRE: a = (izq) r (der)
  POST: hIzq <- izq }

PROCEDURE HijoDer(a: TipoArbolBin; VAR hDer: TipoArbolBin);
{ PRE: a = (izq) r (der)
  POST: hDer <- der }
```

Implementación

IMPLEMENTATION

```
{ CONSTRUCTORAS GENERADORAS }
```

```
PROCEDURE CrearArbolVacio(VAR a: TipoArbolBin);
```

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

```
BEGIN
```

```
  a:=NIL
```

```
END;
```

```
PROCEDURE ConstruirArbolBin(VAR a: TipoArbolBin; izq: TipoArbolBin; r:TipoElemento; der:  
  TipoArbolBin);
```

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

```
BEGIN
```

```
  New(a);
```

```
  Asignar(a^.r,r);
```

```
  a^.izq:=izq;
```

```
  a^.der:=der
```

```
END;
```

Implementación

```
{ OBSERVADORAS NO SELECTORAS }

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

{ OBSERVADORAS SELECTORAS }

PROCEDURE Raiz(a: TipoArbolBin; VAR e: TipoElemento);
{ Complejidad: O(1)}
BEGIN
    Asignar(e,a^.r)
END;
```

Implementación

```
PROCEDURE HijoIzq(a: TipoArbolBin; VAR hIzq: TipoArbolBin);  
{ Complejidad: O(1)}  
BEGIN  
    hIzq := a^.izq  
END;  
  
PROCEDURE HijoDer(a: TipoArbolBin; VAR hDer: TipoArbolBin);  
{ Complejidad: O(1)}  
BEGIN  
    hDer := a^.der  
END;  
  
END.
```

Programación con Árboles Binarios

- Contar número de nodos 
- Contar el número de hojas
- Decir si dos árboles son iguales
- Pertenencia de un elemento a un árbol
- Construir imagen especular
- Calcular la profundidad de un árbol
- Ver si un árbol está equilibrado
- Calcular el nivel en el que aparece un nodo en un árbol

Árboles Binarios de Búsqueda

- Introducción
- Especificación Algebraica
- Implementación
- Árboles equilibrados: Árboles AVL

Introducción



- Particularización de un árbol binario donde sus nodos están ordenados en función del elemento que contienen
- Para cada nodo X , los valores de los elementos de su subárbol izquierdo son menores que el de X y los del derecho son mayores que el de X
- Se consigue una ordenación consistente

Especificación algebraica

ESPECIFICACIÓN ArbolesBinariosBusqueda

USA ArbolesBinarios

PARAMETROS GENERICOS

OPERACIONES

Mayor: TipoElemento x TipoElemento $\rightarrow$ Booleano

FIN PARAMETROS

OPERACIONES

(* observadoras no selectoras *)

Pertenece : TipoArbolBin x TipoElemento $\rightarrow$ Booleano

PARCIAL Minimo: TipoArbolBin $\rightarrow$ TipoElemento

(* constructoras no generadoras *)

Insertar: TipoElemento x TipoArbolBin $\rightarrow$ TipoArbolBin

Eliminar: TipoElemento x TipoArbolBin $\rightarrow$ TipoArbolBin

Especificación algebraica

(* Para trabajar con árboles binarios de búsqueda el usuario deberá utilizar exclusivamente las constructoras 'CrearArbolBinVacio', 'Insertar' y 'Eliminar' *)

VARIABLES

r, e: TipoElemento;

i, d: TipoArbolBin;

ECUACIONES DE DEFINITUD

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

ECUACIONES

(* observadoras no selectoras *)

Pertenece(e, CrearArbolBinVacio) = **FALSO**

Pertenece(e, ConstruirArbolBin(i, r, d)) = SI e=r →

CIERTO

| SI Mayor(e, r) →

Pertenece(e, d)

| Pertenece(e, i)

Especificación algebraica

$\text{Minimo}(\text{ConstruirArbolBin}(i, r, d)) = \text{SI EsArbolBinVacio}(i) \rightarrow$

r

| $\text{Minimo}(i)$

(* constructoras no generadoras *)

$\text{Insertar}(e, \text{CrearArbolBinVacio}) = \text{ConstruirArbolBin}(\text{CrearArbolBinVacio}, e, \text{CrearArbolBinVacio})$

$\text{Insertar}(e, \text{ConstruirArbolBin}(i, r, d)) = \text{SI } e=r \rightarrow$

$\text{ConstruirArbolBin}(i, r, d)$

| $\text{SI Mayor}(e, r) \rightarrow$

$\text{ConstruirArbolBin}(i, r, \text{Insertar}(e, d))$

| $\text{ConstruirArbolBin}(\text{Insertar}(e, i), r, d)$

Especificación algebraica

$\text{Eliminar}(e, \text{CrearArbolBinVacio}) = \text{CrearArbolBinVacio}$

$\text{Eliminar}(e, \text{ConstruirArbolBin}(i, r, d)) = \text{SI } e=r \rightarrow$

$\text{SI } \text{EsArbolBinVacio}(d) \rightarrow$

i

$| \text{ConstruirArbolBin}(i, \text{Minimo}(d), \text{Eliminar}(\text{Minimo}(d), d))$

$| \text{SI } \text{Mayor}(e, r) \rightarrow$

$\text{ConstruirArbolBin}(i, r, \text{Eliminar}(e, d))$

$| \text{ConstruirArbolBin}(\text{Eliminar}(e, i), r, d)$

FIN ESPECIFICACIÓN

Implementación

```
UNIT ABINBUS;  
  
INTERFACE  
  
USES ELEMTAD, ABIN;  
  
    (* CONSTRUCTORAS NO GENERADORAS *)  
  
    PROCEDURE Insertar(VAR a: TipoArbolBin; e: TipoElemento);  
    {PRE: Cierto}  
    {POST: Insertamos el elemento e si no está ya en el árbol}  
  
    PROCEDURE Eliminar(VAR a: TipoArbolBin; e: TipoElemento);  
    {PRE: No definido para Árbol vacío}  
    {POST: Eliminamos el elemento e del árbol binario}
```

Implementación

(* OBSERVADORAS NO SELECTORAS *)

```
FUNCTION Pertenece(a: TipoArbolBin; e: TipoElemento): Boolean;  
{PRE: Cierto}  
{POST: Determina si e está en el árbol}
```

```
PROCEDURE Minimo(a: TipoArbolBin; VAR e: TipoElemento);  
{PRE: Cierto}  
{POST: Devolvemos en "e" el menor elemento del árbol}
```

```
PROCEDURE Maximo(a: TipoArbolBin; VAR e: TipoElemento);  
{PRE: Cierto}  
{POST: Devolvemos en "e" el mayor elemento del árbol}
```

Implementación

IMPLEMENTATION

```
(* CONSTRUCTORAS NO GENERADORAS *)

PROCEDURE Insertar(VAR a: TipoArbolBin; e: TipoElemento);

{ Complejidad: O(n)}

VAR

    r: TipoElemento;
aVacio: TipoArbolBin;

BEGIN

    CrearArbolVacio(aVacio);

    IF EsArbolBinVacio(a) THEN

        ConstruirArbolBin(a, aVacio, e, aVacio)

    ELSE

        BEGIN

            Raiz(a, r);

            IF Menor(e, r) THEN

                Insertar(a^.Izq, e)

            ELSE

                Insertar(a^.Der, e)

        END;

    END;
```

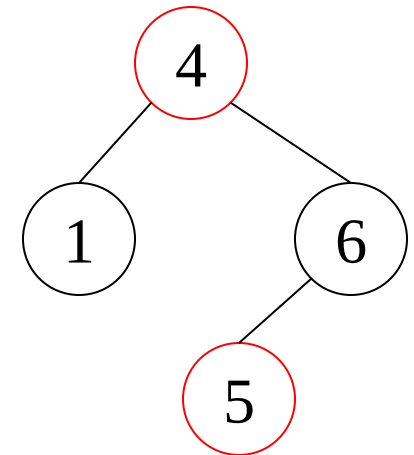
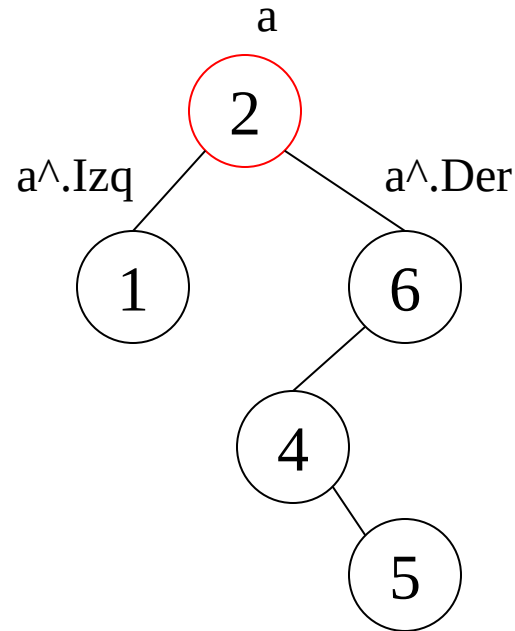
```
END;
```

Implementación

```
PROCEDURE Eliminar(VAR a: TipoArbolBin; e: TipoElemento);  
{ Complejidad: O(n)}  
  
VAR  
    m, r: TipoElemento;  
    sDer, sIzq: TipoArbolBin;  
  
BEGIN  
    IF NOT EsArbolBinVacio(a) THEN  
        BEGIN  
            Raiz(a, r);  
            HijoIzq(a, sIzq);  
            HijoDer(a, sDer);  
  
            IF IgualElem(r, e) THEN {elemento encontrado}  
                IF EsArbolBinVacio(a^.Der) THEN BEGIN {vacía rama derecha}  
                    Dispose(a);  
                    a:= sIzq;  
                END  
            ELSE IF EsArbolBinVacio(a^.izq) THEN {vacía rama izquierda}  
                BEGIN
```

Implementación

```
BEGIN
  Dispose(a);
  a:= sDer;
END
ELSE BEGIN {no vacía ninguna rama}
  Minimo(a^.Der, m);
  Asignar(a^.r, m);
  Eliminar(a^.Der, m);
END
ELSE {casos recursivos: no encontrado}
  IF Menor(e,r) THEN
    Eliminar(a^.Izq, e)
  ELSE
    Eliminar(a^.Der, e)
  END
END;
END;
```



Implementación

```
(* OBSERVADORAS NO SELECTORAS *)  
  
FUNCTION Pertenece(a: TipoArbolBin; e: TipoElemento): Boolean;  
{ Complejidad: O(n)}  
  
VAR  
    r: TipoElemento; sIzq, sDer: TipoArbolBin;  
  
BEGIN  
    IF EsArbolBinVacio(a) THEN  
        Pertenece := FALSE  
    ELSE BEGIN  
        Raiz(a, r);  
        HijoIzq(a, sIzq);  
        HijoDer(a, sDer);  
        IF IgualElem(e, r) THEN  
            Pertenece:= TRUE;  
        IF Menor(e,r) THEN  
            Pertenece:= Pertenece(sIzq, e);  
        IF Mayor(e,r) THEN  
            Pertenece:= Pertenece(sDer, e)  
        END;  
    END;  
END;
```

Implementación

```
PROCEDURE Minimo(a: TipoArbolBin; VAR e: TipoElemento);  
{ Complejidad: O(n)}  
{ Versión Iterativa}  
BEGIN  
  IF NOT EsArbolBinVacio(a) THEN BEGIN  
    (* El elemento más a la izquierda es el menor *)  
    WHILE NOT EsArbolBinVacio(a^.izq) DO  
      HijoIzq(a, a);  
    Raiz(a, e);  
  END  
END;
```

Implementación

```
PROCEDURE Maximo(a: TipoArbolBin; VAR e: TipoElemento);
{ Complejidad: O(n) }
{ Versión recursiva }
VAR
    sDer: TipoArbolBin;
BEGIN
    IF NOT EsArbolBinVacio(a) THEN BEGIN
        (* El elemento más a la derecha es el mayor *)
        HijoDer(a, sDer);
        IF EsArbolBinVacio(sDer) THEN
            Raiz(a, e)
        ELSE
            Maximo(sDer, e);
        END;
    END;
END.
```