

Tema 5

TIPOS DE

DATOS ABSTRACTOS

DINÁMICOS NO LINEALES:

ÁRBOLES

<p>Tutor: Enrique Carmona Asignatura: Estructuras de Datos y Algoritmos</p>

TEMA V : TIPOS DE DATOS ABSTRACTOS NO LINEALES: ÁRBOLES

5.1 Conceptos y definiciones

5.2 Árboles perfectamente balanceados

5.3 Árboles de expresión

5.4 Árboles de búsqueda binarios

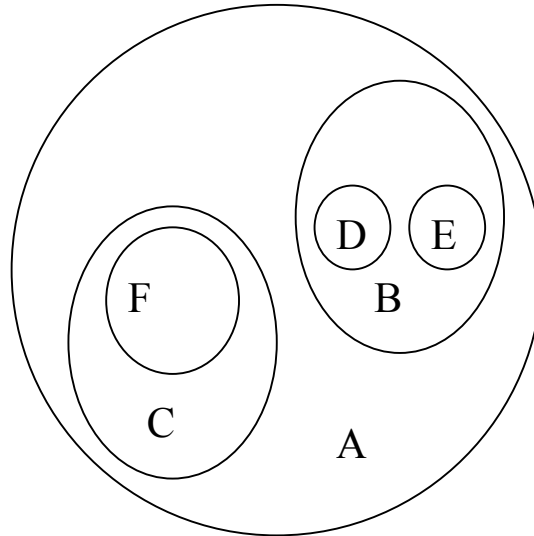
5.5 Árboles de búsqueda balanceados (AVL)

TDA's DINÁMICOS NO LINEALES: ÁRBOLES

Estructura de árbol Estructura utilizada para representar relaciones jerárquicas entre objetos. Cada objeto se representa en un nodo y las relaciones entre ellos están marcadas por las ramas.

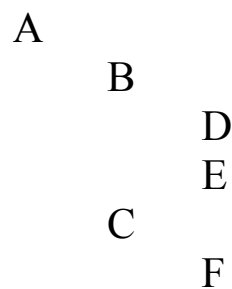
Representaciones de una estructura de árbol

a) Conjuntos anidados.

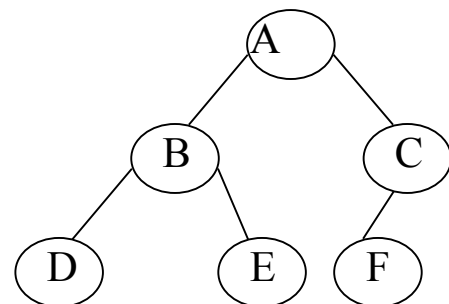


b) Paréntesis anidados : (A(B(D,E),C(F)))

c) Sangría (escalonamiento).



d) Gráfica



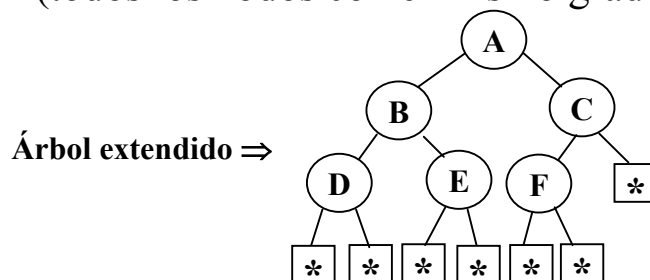
TDA Árbol está formado por nodos con uno o más apuntadores, cada uno de ellos es apuntado por un único nodo (salvo uno, el raíz) y, a su vez, cada uno apunta a uno o más árboles (subárboles). Las operaciones básicas asociadas son la de inserción, búsqueda y eliminación de nodos.

Definiciones Básicas

- **Descendiente (directo) o hijo de un nodo:** sucesor inmediato.
- **Ancestro (directo) de un nodo:** predecesor inmediato.
- **Raíz del árbol:** nodo superior del árbol.
- **Nodo terminal o nodo hoja:** aquel que no tiene descendientes.
- **Nivel de un nodo:** número de descendientes que deben recorrerse desde la raíz al nodo (nodo raíz \Rightarrow Nivel 0)
- **Profundidad o altura de un árbol:** nivel máximo de cualquier nodo del árbol.
- **Grado de un nodo:** n° de descendientes directos del nodo.
- **Grado del árbol:** máximo grado de entre los nodos que pertenecen al árbol (árboles binarios, ternarios, etc.)
- **Longitud de trayectoria de un nodo:** n° de ramas que se tienen que recorrerse para ir desde la raíz al nodo.
- **Longitud de trayectoria interna o longitud de trayectoria del árbol:** Suma de las long. de trayectoria de todos sus nodos.
- **Longitud de trayectoria media:** siendo $n_i =$ n° nodos en nivel i ,

$$L_{med} = \sum_{i=1}^n \frac{(n_i \cdot i)}{n}$$

- **Longitud de trayectoria externa:**
 1. Se extiende el árbol por medio de un nodo especial (todos los nodos con el mismo grado, el grado del árbol).



2. La suma de longitudes de trayectoria en todos los nodos especiales es la longitud de trayectoria externa, L_E .

- **Longitud de trayectoria externa media:** siendo $m_i = n^\circ$ de nodos especiales en el nivel i y m el n° de nodos especiales totales

$$L_{Emed} = \frac{\sum_{i=1}^m m_i \cdot i}{m}$$

- **Propiedad 1:** $m = f(g, n)$

Siendo

$n = n^\circ$ de nodos originales

$m = n^\circ$ de nodos especiales

$g =$ grado del árbol

$$g * n + 1 = m + n \Rightarrow m = (g - 1) n + 1$$

- **Propiedad 2:** N° Máximo de nodos para un árbol de altura h y grado g :

$$N_g(h) = \sum_{i=0}^{i=h} g^i = \frac{1 - g^{h+1}}{1 - g}$$

Si $g = 2$,

$$N_2(h) = 2^{h+1} - 1$$

- **Propiedad 3:** En los árboles binarios, la longitud de trayectoria interna, L_I , y la longitud de trayectoria externa, L_E , están relacionadas mediante la siguiente expresión:

$$L_E = 2n + L_I$$

Representación de la estructura árbol (Modula2):

```
TYPE
  Ptr_Nodo = POINTER TO Nodo;
  Nodo = RECORD
    Dato: Tipo_Datos
    izq, der : Ptr_Nodo
  END
```

Árbol binario perfectamente balanceado (ABPB)

Definición: Un árbol binario es perfectamente balanceado si, para cada nodo, el número de nodos de su subárbol izquierdo y derecho difieren como mucho en 1.

- **Propiedad:** Todo *ABPB* tiene altura mínima.
- **Algoritmo 1:** Construir un *ABPB* conocido el número de nodos:

```
PROCEDURE Arbol_perf_bal (n:INTEGER):Ptr_Nodo;
  (*n: numero de nodos a almacenar en el arbol*)
  VAR   Nuevo_nodo    : Ptr_Nodo;
        n_izq, n_dch, valor : INTEGER;
  BEGIN
    IF n=0 THEN Nuevo_nodo := NIL
    ELSE n_izq := n DIV 2;
        n_dch := n-n_izq-1;
        ReadInt(valor); (*Lee valor a almacenar*)
        ALLOCATE(Nuevo_nodo,SIZE(Nodo));
        WITH Nuevo_nodo^ DO
          Dato := valor;
          izq := Arbol_perf_bal(n_izq);
          dch := Arbol_perf_bal(n_dch);
        END;
    END;
    RETURN Nuevo_nodo
  END Arbol_perf_bal;
```

Recursividad

- **Algoritmo 2:** Construir un *APB* con n° de nodos desconocido:

```

PROCEDURE Perfec_balanceado(f:File;VAR k:Ptr_Nodo);
  (*f: fichero con los datos a almacenar en el arbol*)
  VAR w: INTEGER;
  PROCEDURE Insertar_Arbol_perf(w: INTEGER;VAR k:Ptr_Nodo);
    (*w:elemento a introducir en el arbol; k:raiz del arbol*)
    VAR numdch,numizq : INTEGER;

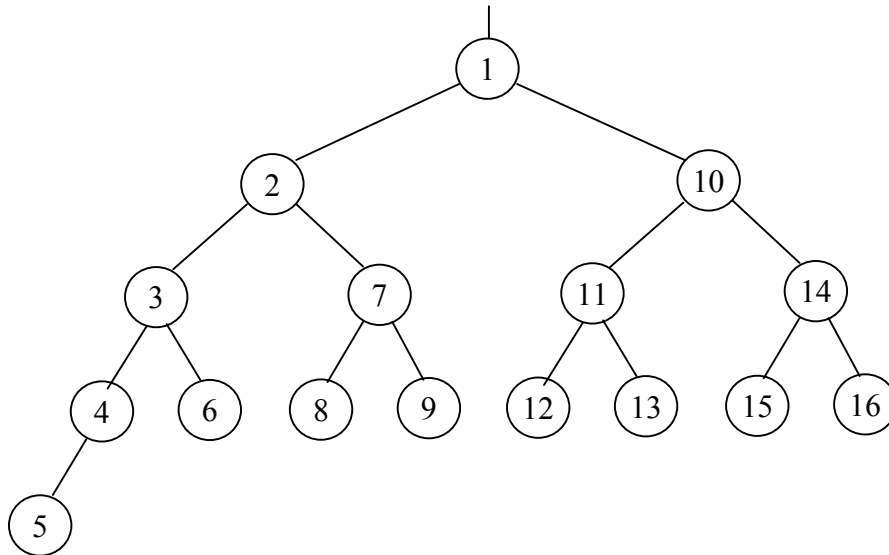
    PROCEDURE NumNodos(a: Ptr_Nodo):INTEGER;
      (*Devuelve el numero de nodos del arbol apuntado por a*)
      BEGIN
        IF a=NIL THEN RETURN 0
        ELSE RETURN 1+NumNodos(a^.izq)+NumNodos(a^.dch);
      END
      END NumNodos;

    BEGIN
      (*lectura hasta que termine la entrada de datos*)
      IF k#NIL THEN
        numdch:=NumNodos(k^.dch);
        numizq:=NumNodos(k^.izq);
      END;
      IF k=NIL THEN (*insertar*)
        ALLOCATE(k,SIZE(Nodo));
        k^.Dato:=w;
        k^.dch:=NIL;
        k^.izq:=NIL;
        ELSIF numdch=numizq THEN Insertar_Arbol_perf(w,k^.izq)
        ELSIF numizq>numdch THEN Insertar_Arbol_perf(w,k^.dch)
      END;
    END Insertar_Arbol_perf;

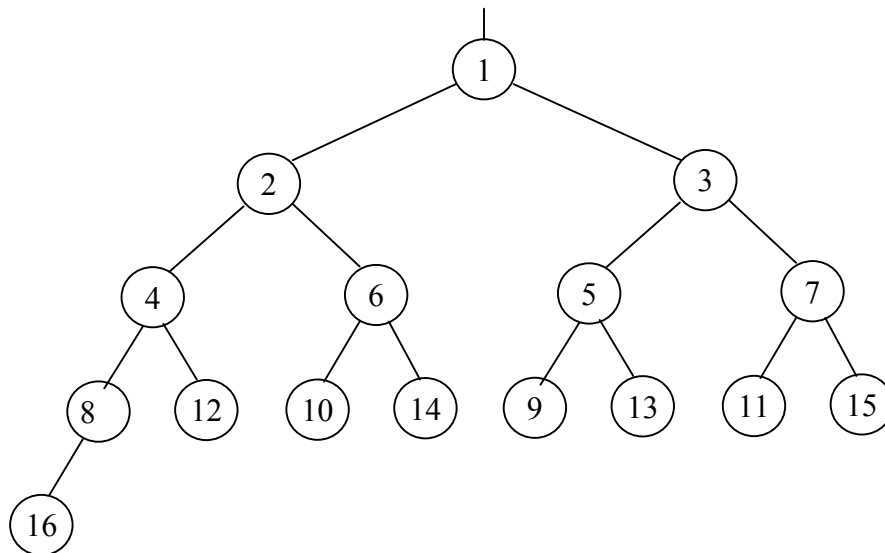
  BEGIN
    Reset(f);
    WHILE ~f.eof DO
      ReadWord(f,w);
      Insertar_Arbol_perf(w,k);
    END;
    Close(f);
  END Perfec_balanceado;

```

Datos: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

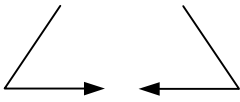
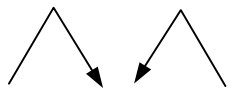
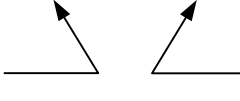


Distribución del árbol binario perfectamente balanceado cuando **el número de datos se conoce a priori**



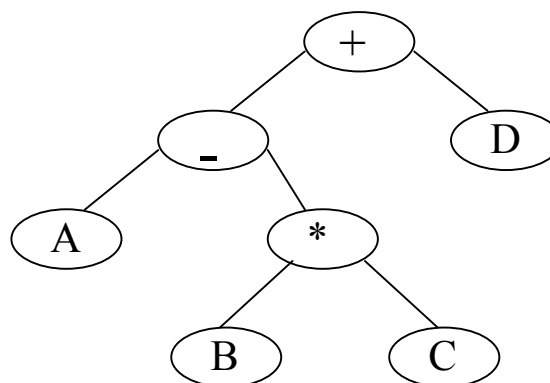
Distribución del árbol binario perfectamente balanceado cuando **el número de datos no se conoce a priori**

Recorrido de un árbol binario

Recorrido			Notación
Preorden		1. Raíz 2. Un Subárbol 3. Otro Subárbol	Prefija
En orden		1. Un Subárbol 2. Raíz 3. Otro Subárbol	Infija
Postorden		1. Un Subárbol 2. Otro Subárbol 3. Raíz	Postfija

El ordenamiento se establece en función del orden en que se visitan el nodo y sus descendientes izquierdos y derecho

• **Árboles de expresión:** Son árboles binarios que permiten tratar expresiones diádicas. Para ello los nodos contienen operadores y éstos actúan sobre los operandos que se almacenan en los hijos del nodo.



La expresión aritmética en **notación infija**: $((A-(B*C))+D)$

Se expresa en **notación prefija**: $+-(A*B)CD$

Y en **notación postfija**: $ABC*-D+$

Es decir, basta crear el *árbol de expresión* correspondiente y recorrerlo de la forma adecuada para obtener una u otra notación

Recorrido preorden recursivo

```
PROCEDURE preorden (t : Ptr);
BEGIN
    IF t # NIL THEN
        P(t); (*Representa la acción a realizar en cada nodo*)
        preorden(t^.izq);
        preorden(t^.der)
    END
END preorden
```

Recorrido en orden recursivo

```
PROCEDURE inorden (t : Ptr);
BEGIN
    IF t # NIL THEN
        inorden(t^.izq);
        P(t); (*Representa la acción a realizar en cada nodo*)
        inorden(t^.der)
    END
END inorden
```

Recorrido postorden recursivo

```
PROCEDURE preorden (t : Ptr);
BEGIN
    IF t # NIL THEN
        postorden(t^.izq);
        postorden(t^.der);
        P(t) (*Representa la acción a realizar en cada nodo*)
    END
END postorden
```

Recorrido en orden no recursivo

```
PROCEDURE inorden (t : Ptr);
VAR
  Pila : Tipo_Pila; (*sus elementos son del tipo Ptr*)
  p : Ptr;
BEGIN
  Inicia_Pila(pila);
  p := t;
  REPEAT
    WHILE p # NIL DO
      Meter_pila (pila, p);
      p := p^.izq
    END;
    IF NOT Pila_Vacia(pila) THEN
      Sacar_Pila (pila, p);
      WRITELN (p^.datos);
      p := p^.der
    END
  UNTIL (p = NIL) AND (Pila_Vacia (pila))
END inorden;
```

Problema: los árboles binarios perfectamente balanceados son eficaces en el sentido de altura mínima pero son ineficaces en cuanto a operaciones de búsqueda (están desordenados)

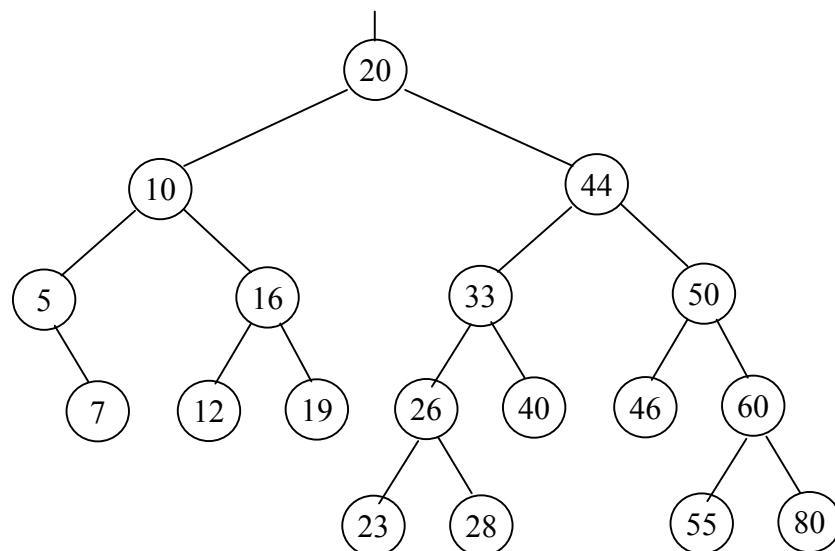
Árboles de Búsqueda Binarios

Definición: son aquellos árboles que cumplen que para todo nodo del árbol, todas las llaves de su subárbol izquierdo son menores que la llave que él contiene y todas las del subárbol derecho son mayores.

Definición generalizada: Un árbol de búsqueda es un TDA árbol en el que para cada nodo todas las llaves de cada subárbol satisfacen una y sólo una condición de un conjunto de n_C condiciones mutuamente excluyentes (cada nodo tiene n_C enlaces)

Ejemplo

Datos: 20, 10, 5, 7, 44, 33, 26, 23, 16, 50, 40, 60, 12, 19, 46, 80, 28, 55



Insertión en un árbol de búsqueda

```
PROCEDURE Insertar_busqueda(x:Tipo_datos;VAR puntero:Ptr_Nodo);
BEGIN
  IF puntero=NIL THEN (* insertar *)
    ALLOCATE(puntero,SIZE(Nodo));
    WITH puntero^ DO
      llave:=x; izq:=NIL; dch:= NIL
    END
  ELSIF x<puntero^.Dato THEN
    Insertar_busqueda(x,puntero^.izq)
  ELSIF x>puntero^.Dato THEN
    Insertar_busqueda(x,puntero^.dch)
  ELSE (*No se satisface ninguna de las dos condiciones*)
  END;
END Insertar_busqueda;
```

Búsqueda de un nodo en un árbol de búsqueda

```
PROCEDURE Busqueda(x:Tipo_datos; VAR puntero:Ptr_Nodo; VAR
encontrado:BOOLEAN);
BEGIN
  IF puntero=NIL THEN (*la llave no está en el árbol; insertar*)
    encontrado:=FALSE;
    ALLOCATE(puntero, SIZE(Nodo));
    WITH puntero^ DO
      llave:=x; izq:=NIL; dch:= NIL
    END
  ELSIF (x<puntero^.Dato) THEN
    Busqueda(x,puntero^.izq,encontrado)
  ELSIF (x>puntero^.Dato) THEN
    Busqueda(x,puntero^.dch,encontrado)
  ELSE encontrado:=TRUE;
  END
END Busqueda;
```

Eliminación de un nodo en un árbol de búsqueda

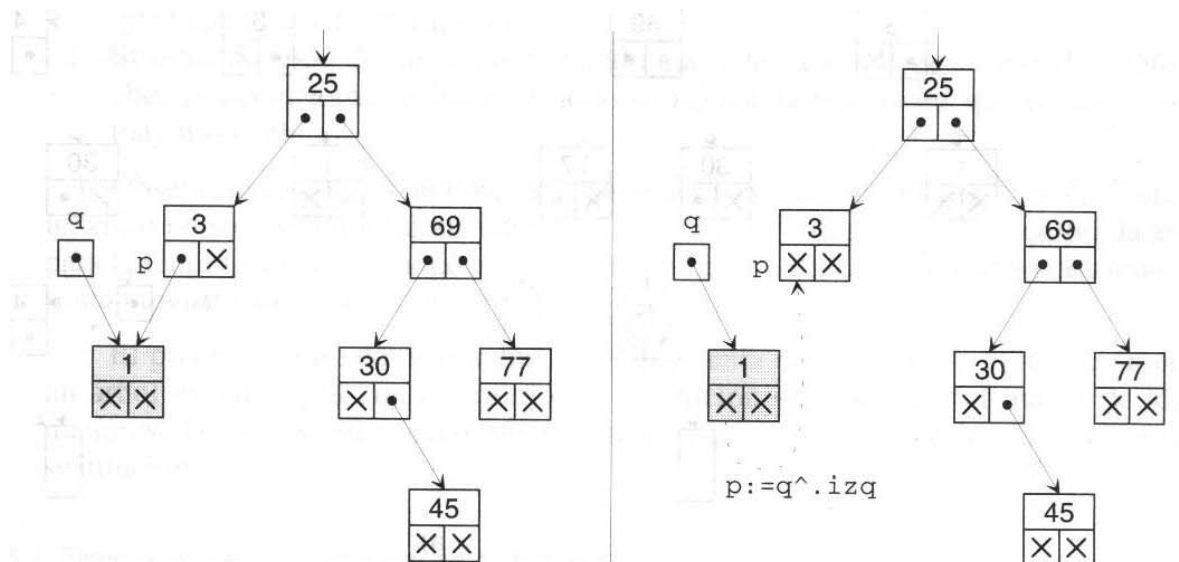
Se pueden dar tres situaciones distintas:

- No existe el nodo que se quiere eliminar (trivial)
- El nodo a eliminar tiene como máximo un descendiente:
 - Si ningún descendiente: Asignar NIL al puntero que apunta al nodo a eliminar y liberar nodo (ptro. auxiliar)
 - Si un descendiente: el puntero del nodo que lo apunta se modifica por el descendiente del nodo a borrar y se libera el nodo a borrar (ptro. auxiliar)
- El nodo a eliminar tiene dos descendientes:

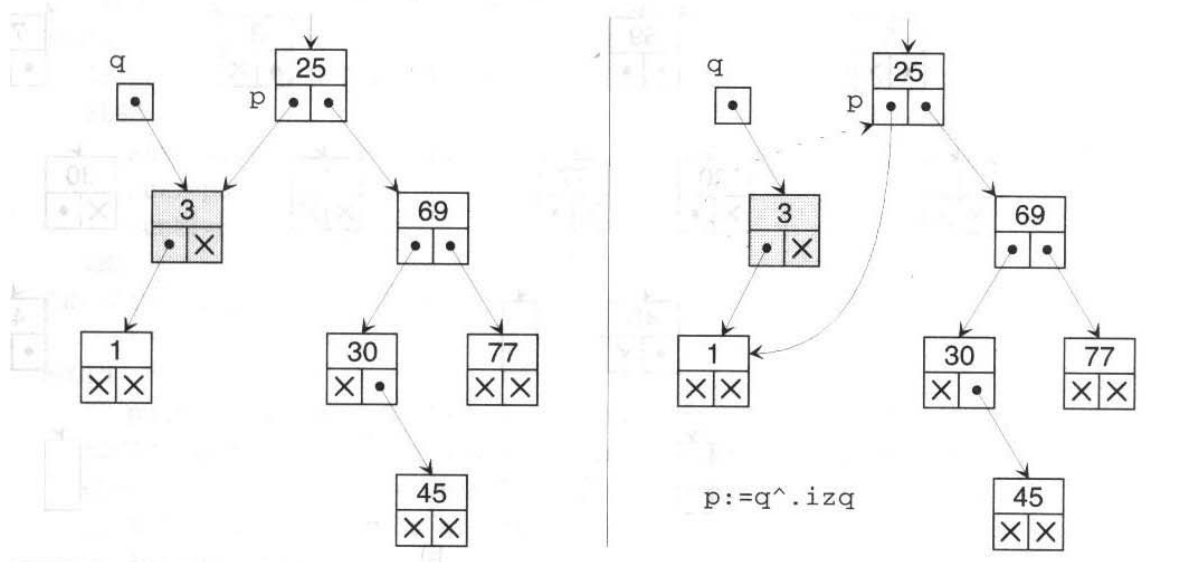
Hay dos soluciones:

- (1) Sustituir el nodo eliminado por el nodo mas a la derecha de su subárbol izquierdo, es decir, sustituirlo por el nodo de llave mayor de todas las menores que él.
- (2) Sustituir el nodo eliminado por el nodo mas a la izquierda de su subárbol derecho, es decir, sustituirlo por el nodo de llave menor de todas las mayores que él.

Eliminación en un árbol de búsqueda

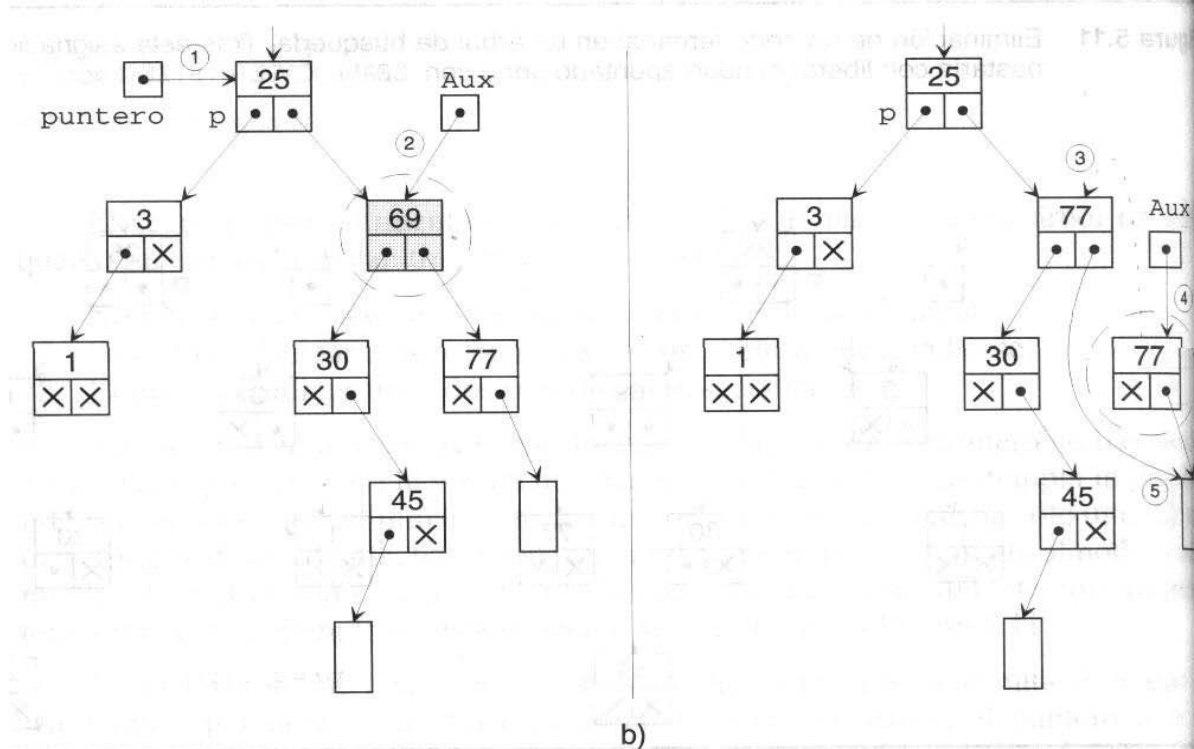
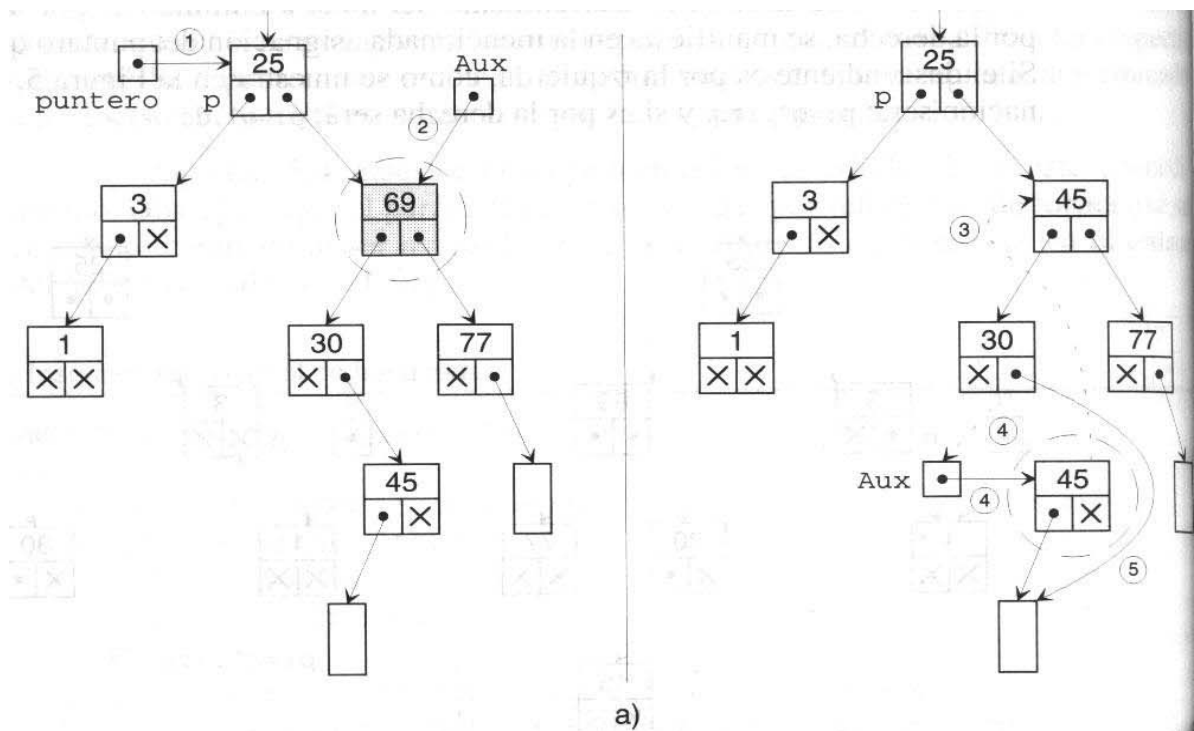


Eliminación de un nodo terminal en un árbol de búsqueda. Tras esta asignación de puntero bastaría con liberar el nodo apuntado por q con `DEALLOCATE(q , SIZE(Nodo))`;



Eliminación de un nodo con un único descendiente, donde puede observarse una situación similar a la mostrada en la Figura 5.11 para un nodo terminal. Si el nodo de llave 3 hubiese tenido el descendiente por la derecha la asignación sería $p := q^{\wedge}.dch$;

Eliminación en un árbol de búsqueda



Eliminación de un nodo con dos descendientes en un árbol binario de búsqueda.

- a) Búsqueda recursiva y sustitución por el mayor de los menores.
 b) Búsqueda recursiva y sustitución por el menor de los mayores.


```

PROCEDURE Eliminar_busqueda(x:Tipo_datos;VAR puntero:Ptr_Nodo);
VAR
  Aux : Ptr_Nodo;
PROCEDURE Eliminar_dch(VAR punt:Ptr_Nodo);
BEGIN
  IF punt^.dch#NIL THEN Eliminar_dch(punt^.dch)
  ELSE Aux^.Dato:=punt^.Dato;
    Aux:=punt;
    punt:=punt^.izq;
  END
END Eliminar_dch;

BEGIN
  IF puntero=NIL THEN
    WriteString ('No hay nodo a eliminar')
  ELSIF x<puntero^.Dato THEN
    Eliminar_busqueda(x,puntero^.izq)
  ELSIF x>puntero^.Dato THEN
    Eliminar_busqueda(x,puntero^.dch)
  ELSE (* encontrado nodo a eliminar*)
    Aux:=puntero;
    IF Aux^.dch=NIL THEN puntero:=Aux^.izq (*menos de dos...*)
    ELSIF Aux^.izq=NIL THEN puntero:=Aux^.dch (*...hijos*)
    ELSE
      (* dos hijos: sustituir por el de más a la dcha del subárbol izqdo *)
      Eliminar_dch(Aux^.izq)
    END;
    DEALLOCATE(Aux,SIZE(Nodo));
  END
END Eliminar_busqueda;

```

Análisis de la inserción y la búsqueda en árboles de búsqueda

- **Mejor caso:** árbol de n nodos perfectamente balanceado

$$C_n = \log n$$

- **Peor caso:** árbol de n nodos completamente degenerado (lista)

$$C_n = O(n)$$

- **Caso promedio:** para ello se deberá considerar la búsqueda entre los n nodos y los $n!$ árboles posibles que se pueden generar.

Si consideramos cualquier árbol binario, el número de comparaciones promedio, si la llave buscada existe en el árbol:

$$C_n = 1 + \frac{L_I}{n}$$

si la llave buscada no existiera:

$$C'_n = \frac{L_E}{m}$$

teniendo en cuenta las propiedades P.1 y P.3 anteriores:

$$m = (g-1)n + 1 \quad L_E = 2n + L_I$$

se obtiene:

$$C_n = \left(1 + \frac{1}{n}\right) C'_n - 1$$

formula de Hibbard que se satisface para cualquier árbol binario.

Se supone que las $n!$ secuencias de las n llaves se insertan en el árbol de búsqueda con igual probabilidad $1/n$. Entonces el número de comparaciones promedio para insertar un nodo en el árbol de búsqueda será:

$$\frac{C'_0 + C'_1 + \dots + C'_{n-1}}{n}$$

siendo C'_i , el número de comparaciones promedio cuando aún no se encuentra en el árbol la llave i .

El número de comparaciones promedio para buscar un nodo será el número de comparaciones promedio para insertarlo más uno:

$$C_n = 1 + \frac{C'_0 + C'_1 + \dots + C'_{n-1}}{n}$$

Teniendo en cuenta la fórmula de Hibbard:

$$(n+1)C'_n = 2n + C'_0 + C'_1 + \dots + C'_{n-1}$$

Desarrollando la expresión anterior para $(n-1)$ y restando ambas expresiones:

$$C'_n = C'_{n-1} + \frac{2}{n+1}$$

Desarrollando en sus sucesivos términos y aplicando la notación del número armónico:

$$C'_n = 2H_{n+1} - 2$$

Aplicando la fórmula de Hibbard y teniendo en cuenta que $H_{n+1} = 1/(n+1) + H_n$

$$C_n = 2\left(1 + \frac{1}{n}\right)H_n - 3$$

Finalmente, si se aplica la fórmula de Euler para n grandes ($H_n \approx \ln(n) + \gamma$):

$$C_n \approx 2 \ln n = 1.386 \log n$$

“el número de comparaciones promedio en un árbol de búsqueda es del 39% mayor que la correspondiente a un árbol perfectamente balanceado”

Conclusión: no se justifica el costo necesario para convertir un árbol de búsqueda en un árbol perfectamente balanceado en cada inserción.

Árboles Balanceados o árboles AVL

- Los árboles AVL surgen al tratar de encontrar un cierto equilibrio entre la eficacia de búsqueda que presentan los *árboles de búsqueda* y el crecimiento uniforme que presentan los *árboles perfectamente balanceados*.
- **Criterio de equilibrio:** Un árbol está balanceado si y sólo si para cada uno de sus nodos se cumple que las alturas de sus dos subárboles, izquierdo y derecho, difieren como mucho en 1.
- **Definición:** Un árbol AVL es un árbol de búsqueda al que se le impone el criterio de equilibrio mencionado anteriormente.

Inserción en árboles balanceados

- El proceso de inserción consta de dos fases:
 - 1ª) Insertar el nuevo nodo como si de un árbol de búsqueda se tratara (se supone que el nodo no existe).
 - 2ª) Retroceder a lo largo de la trayectoria seguida para insertar el nodo, chequeando si cada nodo, perteneciente a dicha trayectoria, cumple el criterio de equilibrio. Si existe alguno para el que no se satisface el criterio de equilibrio, entonces rebalancear.

Nota: En cada nodo habrá que transmitir información sobre si ha variado la altura sobre el subárbol en el que se realizó la inserción; para ello se utiliza *h*: **BOOLEAN**.

- 3ª) El rebalanceo se realiza de acuerdo a uno de los cuatro patrones de rebalanceo: LL, LR, RR ó RL.

- Supóngase un nodo N, con subárboles L y R y que el nuevo nodo se inserta en L, haciendo que su altura aumente en 1. Pueden presentarse tres casos (y sus otros 3 casos simétricos):
 1. Si inicialmente $h_L = h_R$: L y R tendrán altura desigual, $h_L = h_R + 1$, pero sin violar el criterio de equilibrio.
 2. Si inicialmente $h_L < h_R$: L y R tendrán igual altura, esto es, el equilibrio ha mejorado, $h_L = h_R$.
 3. Si inicialmente $h_L > h_R$: el criterio de equilibrio se viola, $h_L = h_R + 2$, hay que reestructurar el árbol (**REBALANCEO**).
- Para cada nodo, se define el llamado **factor de balance o equilibrio, bal**, como: $bal = h_R - h_L$
 - Todos los nodos de un AVL tienen un factor de equilibrio 0,+1,-1 (en particular, el de los nodos terminales es siempre 0)
- Se amplía la **estructura de datos** utilizada hasta ahora para incluir el **factor de balance o de equilibrio**:

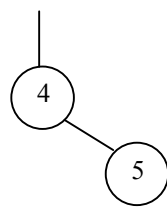
```

TYPE Ptr_Nodo = POINTER TO Nodo;
TYPE Balance = [-1..+1];
TYPE Nodo = RECORD
    llave: INTEGER;
    izq, der: Ptr_Nodo;
    bal: Balance
END;
```

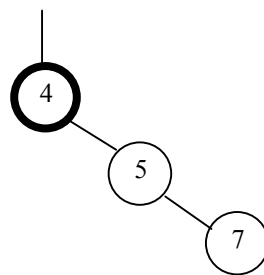
Rebalanceo: Tipo de Rotaciones

- Si al añadir un nodo al árbol, el factor de equilibrio de algún nodo pasara a valer +2 ó -2 => reestructurar el árbol.
- **Notación:**
 - H : nodo que acabamos de insertar.
 - P : predecesor más próximo a H cuyo factor de equilibrio pasa a valer +2 ó -2.

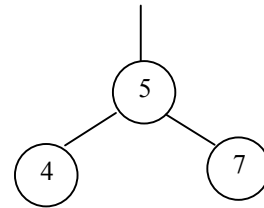
- Si es necesario el rebalanceo, las rotaciones pueden ser:
 - Rotación simple LL:** H insertado en el SI del SI de P.
 - Rotación doble LR:** H insertado en el SD del SI de P.
 - Rotación simple RR:** H insertado en el SD del SD de P.
 - Rotación doble RL:** H insertado en el SI del SD de P.
- Las rotaciones son sólo cambios de enlaces => reasignaciones de apuntadores, además de la actualización del factor de equilibrio.



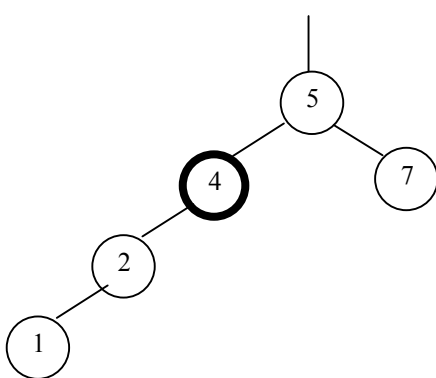
Inicio



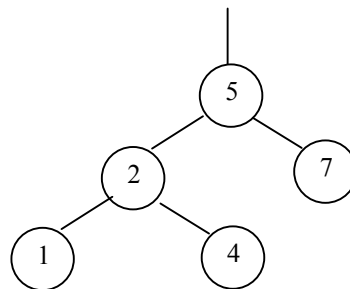
Inserta 7



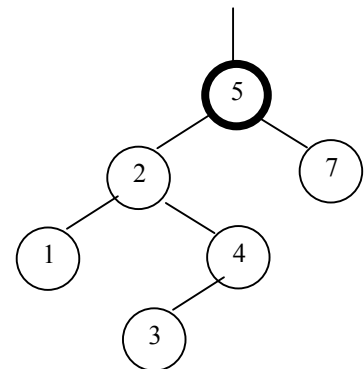
Rebalanceo RR



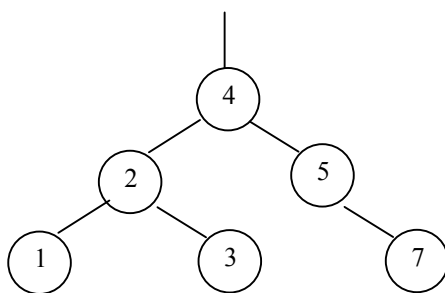
Inserta 2 y 1



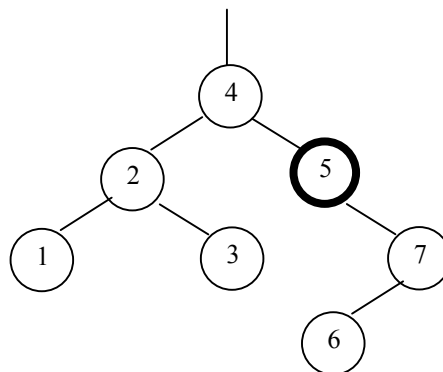
Rebalanceo LL



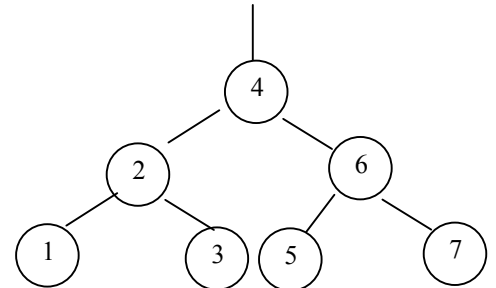
Inserta 3



Rebalanceo LR



Inserta 6



Rebalanceo RL

Procedimiento para insertar un nodo en un árbol AVL

```
PROCEDURE Insertar_AVL(x:Tipo_datos;VAR p:Ptr_Nodo2;
VAR h:BOOLEAN);
VAR p1, p2 : Ptr_Nodo2;
BEGIN
  IF p=NIL THEN (* insertar *)
    ALLOCATE(p,SIZE(Nodo)); h:=TRUE;
    WITH p^ DO llave:=x; izq:=NIL; dch:= NIL; bal:=0;END
  ELSIF p^.llave>x THEN Insertar_AVL(x,p^.izq,h);
    IF h THEN
      CASE p^.bal OF
        1 : p^.bal:=0; h:=FALSE |
        0 : p^.bal:=-1 |
        -1: (* REBALANCEO IZQUIERDO*)
          IF p1^.bal=-1 THEN “ROTACIÓN LL”
          ELSE “ROTACIÓN LR”
          END;
          h:=FALSE
        END
      END
    END
  ELSIF p^.llave<x THEN Insertar_AVL(x,p^.dch,h);
    IF h THEN
      CASE p^.bal OF
        -1: p^.bal:=0; h:=FALSE |
        0 : p^.bal:=1 |
        1 : (* REBALANCEO DERECHO *)
          IF p1^.bal=1 THEN “ROTACIÓN RR”
          ELSE “ROTACIÓN RL”
          END;
          h:=FALSE
        END
      END
    END
  ELSE (*No se satisfacen las condiciones excluyentes*)
  END
END Insertar_AVL;
```


Rebalanceo izquierdo

```
p1:=p^.izq;  
IF p1^.bal=-1 THEN  (* ROTACIÓN LL *)  
    p^.izq:=p1^.dch; p1^.dch:=p;  
    p^.bal:=0; p:=p1  
ELSE                (* ROTACIÓN LR *)  
    p2:=p1^.dch;  
    p1^.dch:=p2^.izq; p2^.izq:=p1;  
    p^.izq:=p2^.dch; p2^.dch:=p;  
    IF p2^.bal=-1 THEN p^.bal:=1 ELSE p^.bal:=0 END;  
    IF p2^.bal=1 THEN p1^.bal:=-1 ELSE p1^.bal:=0 END;  
    p:=p2;  
END;  
p^.bal:=0; h:=FALSE
```

Rebalanceo derecho

```
p1:=p^.dch;  
IF p1^.bal=1 THEN  (* ROTACIÓN RR *)  
    p^.dch:=p1^.izq; p1^.izq:=p;  
    p^.bal:=0; p:=p1  
ELSE                (* ROTACIÓN RL *)  
    p2:=p1^.izq;  
    p1^.izq:=p2^.dch; p2^.dch:=p1;  
    p^.dch:=p2^.izq; p2^.izq:=p;  
    IF p2^.bal=1 THEN p^.bal:=-1 ELSE p^.bal:=0 END;  
    IF p2^.bal=-1 THEN p1^.bal:=1 ELSE p1^.bal:=0 END;  
    p:=p2;  
END;  
p^.bal:=0; h:=FALSE
```

Eliminación en árboles balanceados.

- Implica mayor costo que la inserción.
- Pasa por dos fases:
 - 1ª) Eliminar el nodo que se quiere borrar de acuerdo a las mismas reglas de eliminación que se uso en árboles de búsqueda.
 - 2ª) Comprobar si es necesario el rebalanceo después de la eliminación y, si es así, hacerlo
- $h : \text{BOOLEAN} \Rightarrow$ Controla si, respecto a un nodo, la altura de uno de sus subárboles ha sido reducida:
 - Es condición necesaria que $h = \text{TRUE}$ para que haya que plantearse el rebalanceo, pero no es condición suficiente.
 - La suficiencia del rebalanceo se obtiene chequeando el valor del campo **bal** de cada uno de los nodos implicados en la trayectoria que va desde la raíz hasta el nodo que se quería eliminar.

Procedimiento para eliminar un nodo en un árbol AVL

```
PROCEDURE Eliminar_AVL(x:Tipo_datos; VAR p:Ptr_Nodo2;  
VAR h:BOOLEAN);  
  VAR    q : Ptr_Nodo2;  
PROCEDURE Eliminar_dch(VAR pl:Ptr_Nodo2; VAR h:BOOLEAN);  
  BEGIN  
    IF pl^.dch#NIL THEN  
      Eliminar_dch(pl^.dch,h);  
    IF h THEN Balance_R(pl,h) END  
    ELSE q^.llave:=pl^.llave; q:=pl; pl:=pl^.izq;  
    h:=TRUE;  
  END  
END Eliminar_dch;  
  
BEGIN  (*Eliminar_AVL*)  
  IF p=NIL THEN (* La llave no está en el árbol *)  
  ELSIF p^.llave>x THEN  
    Eliminar_AVL(x,p^.izq,h);  
    IF h THEN Balance_L(p,h) END  
  ELSIF p^.llave<x THEN  
    Eliminar_AVL(x,p^.dch,h);  
    IF h THEN Balance_R(p,h) END  
  ELSE (* Eliminación *)  
    q:=p;  
    IF q^.dch=NIL THEN p:=q^.izq; h:=TRUE;  
    ELSIF q^.izq=NIL THEN p:=q^.dch; h:=TRUE  
    ELSE  
      Eliminar_dch(q^.izq,h);  
      IF h THEN Balance_L(p,h) END  
    END;  
    DEALLOCATE(q,SIZE(Nodo));  
  END  
END Eliminar_AVL;
```

PROCEDURE Balance_R(VAR p:Ptr_Nodo2; VAR h: BOOLEAN);

VAR p1,p2: Ptr_Nodo2; b1,b2:Balance;

BEGIN (**h="true", la rama de la derecha decreció**)

CASE p^.bal OF

1 :p^.bal:=0|

0 :p^.bal:=-1;h:=FALSE|

-1: (**rebalanceo**) p1:=p^.izq;b1:=p1^.bal;

IF b1<=0 THEN (**rotación simple LL**)

p^.izq:=p1^.dch;p1^.dch:=p;

IF b1=0 THEN p^.bal:=-1;p1^.bal:=1;h:=FALSE

ELSE p^.bal:=0;p1^.bal:=0

END;

p:=p1

ELSE (**rotación doble LR**)

p2:=p1^.dch;b2:=p2^.bal;

p1^.dch:=p2^.izq;p2^.izq:=p1;

p^.izq:=p2^.dch;p2^.dch:=p;

IF b2=-1 THEN p^.bal:=1 ELSE p^.bal:=0 END;

IF b2=+1 THEN p1^.bal:=-1 ELSE p1^.bal:=0 END;

p:=p2;p2^.bal:=0

END

END

END Balance_R;

PROCEDURE Balance_L(VAR p:Ptr_Nodo2; VAR h: BOOLEAN);

VAR p1,p2: Ptr_Nodo2; b1,b2:Balance;

BEGIN (**h="true", la rama de la izquierda decreció**)

CASE p^.bal **OF**

-1:p^.bal:=0|

0 :p^.bal:=1;h:=FALSE|

1 : (**rebalanceo**) p1:=p^.dch;b1:=p1^.bal;

IF b1>=0 **THEN** (**rotación simple RR**)

p^.dch:=p1^.izq;p1^.izq:=p;

IF b1=0 **THEN** p^.bal:=1;p1^.bal:=-1;h:=FALSE

ELSE p^.bal:=0;p1^.bal:=0

END;

p:=p1

ELSE (**rotación doble RL**)

p2:=p1^.izq;b2:=p2^.bal;

p1^.izq:=p2^.dch;p2^.dch:=p1;

p^.dch:=p2^.izq;p2^.izq:=p;

IF b2=+1 **THEN** p^.bal:=-1 **ELSE** p^.bal:=0 **END;**

IF b2=-1 **THEN** p1^.bal:=1 **ELSE** p1^.bal:=0 **END;**

p:=p2;p2^.bal:=0

END

END

END Balance_L;

Análisis

Teorema: La altura, h , de un árbol balanceado con n nodos cumple que:

$$\lg_2(n+1) \leq h_n < 1.4404 \lg_2(n+2) - 0.328$$

- Teniendo en cuenta que la altura de un árbol perfectamente balanceado es

$$h = \lfloor \lg_2 n \rfloor$$

el resultado del teorema indica que la altura de un árbol AVL nunca será mayor que el 45% respecto a su árbol perfectamente balanceado.

➤ **Inserciones**

Resultados experimentales permiten establecer que:

- En promedio, el rebalanceo es necesario cada dos inserciones.
- Las rotaciones simples y dobles son igual de probables.
- La altura esperada es $h_{\text{esp}} = \lg_2(n+0.25)$.

➤ **Eliminaciones**

Resultados experimentales permiten establecer que:

- En promedio, sólo es necesario un rebalanceo en una de cada cinco eliminaciones.

➤ **Conclusión**

- Los árboles AVL presentan procedimientos de rebalanceo manejables.
- Las operaciones de búsqueda, inserción y eliminación tienen un coste del orden $\lg_2 n$.