Estructura de datos y algoritmos

Tema V

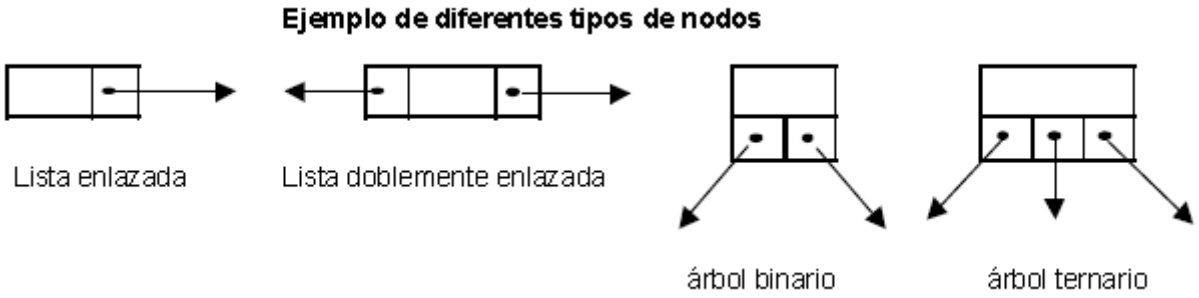
TDA DINÁMICOS NO LINEALES:
Árboles: árboles binarios

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)

5.1 Introducción y Definiciones

- Se denomina nodo
  - A cualquier tipo cuyos elementos son registros formados por un campo Datos y un número dado de apuntadores o enlaces.

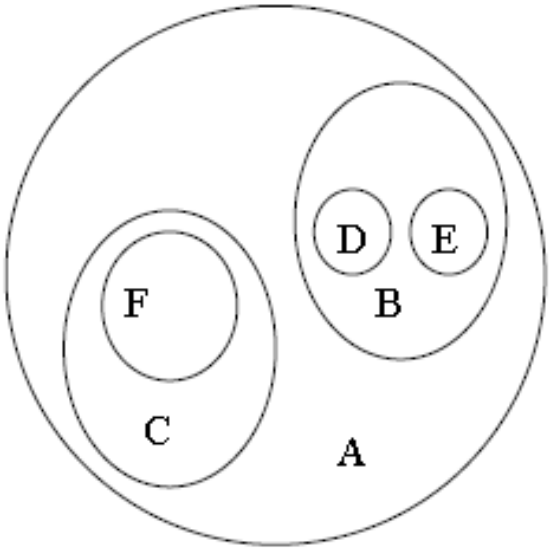


El TDA árbol de grado n

- 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.

Representaciones de una estruc

a) Conjuntos anidados.

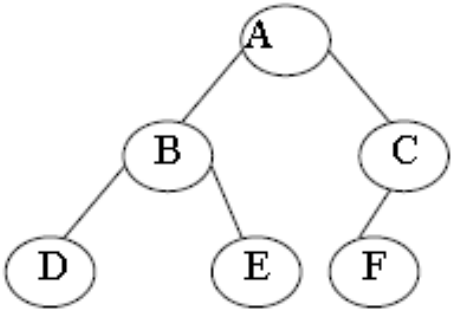


b) Paréntesis anidados : (A(B(D,E),C(F)))

c) Sangría (escalonamiento).

```
A
    B
        D
        E
    C
        F
```

d) Gráfica



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.

Definiciones

- Nivel de un nodo:
  - Número de descendientes que deben recorrerse desde la raíz al nodo (nodo raíz ⇒ Nivel 0)
- Profundidad o altura de un árbol:
  - Nivel máximo de cualquier nodo del árbol.
- Grado de un nodo:
  - nº 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:
  - nº de ramas que se tienen que recorrerse para ir desde la raíz al nodo.
- Longitud de trayectoria interna o longitud de trayectoria del árbol (L_I) :
  - Suma de las longitudes de trayectoria de todos sus nodos.
- Longitud de trayectoria media:
  - siendo n_i = nº nodos en nivel i

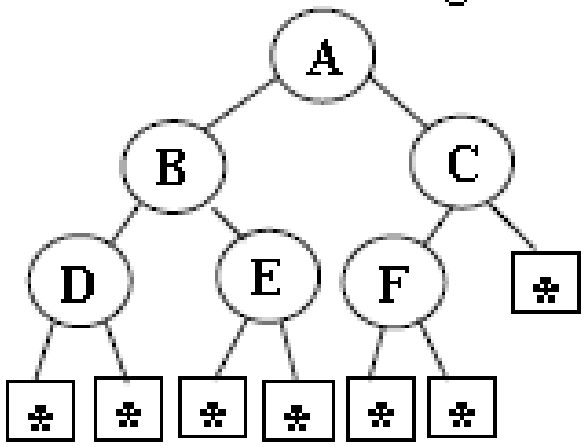
$$L_I^m = \frac{\sum_{i=1}^{n} n_i \cdot i}{n} \quad \text{donde } n_i \text{ es el número de nodos en el nivel } i.$$

Árbol Extendido

- Dado un árbol, su ***árbol extendido*** *es el árbol ampliado con nodos especial es tal que todos los* subárboles son completos, de manera que todos los apuntadores sin nodos descendientes apuntan a un nodo especial.
  - Los nodos especiales no tienen descendientes.

1. Se extiende el árbol por medio de un nodo especial (todos los nodos con el mismo grado, el grado del árbol).

Árbol extendido ⇒



Trayectoria externa de un árbol LE,

- Se define la *longitud de trayectoria externa de un árbol L_E, como la suma de las longitudes* de trayectoria de todos sus nodos especiales.
- La *longitud de trayectoria externa media es:*

$$L_E^m = \frac{\sum_{i=1}^{m} m_i \cdot i}{m}, \text{ donde } m_i \text{ es el número de nodos especiales en el nivel } i.$$

Propiedades

- **Propiedad 1:** el número de nodos especiales *m*
  - **m = f (g, n)** :
  - Siendo n = nº de nodos originales y g = grado del árbol, entonces el número de nodos especiales *m, que debe añadirse para calcular la longitud de la trayectoria externa ES:*

$$g * n + 1 = m + n \Rightarrow \boxed{m = (g - 1)\, n + 1}$$

- **Propiedad 2: NºMáximo de nodos para un árbol de altura h y grado g es:**

$$N_g = \sum_{i=0}^{h-1} g^i$$

- **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:

$$L_E = 2n + L_I$$

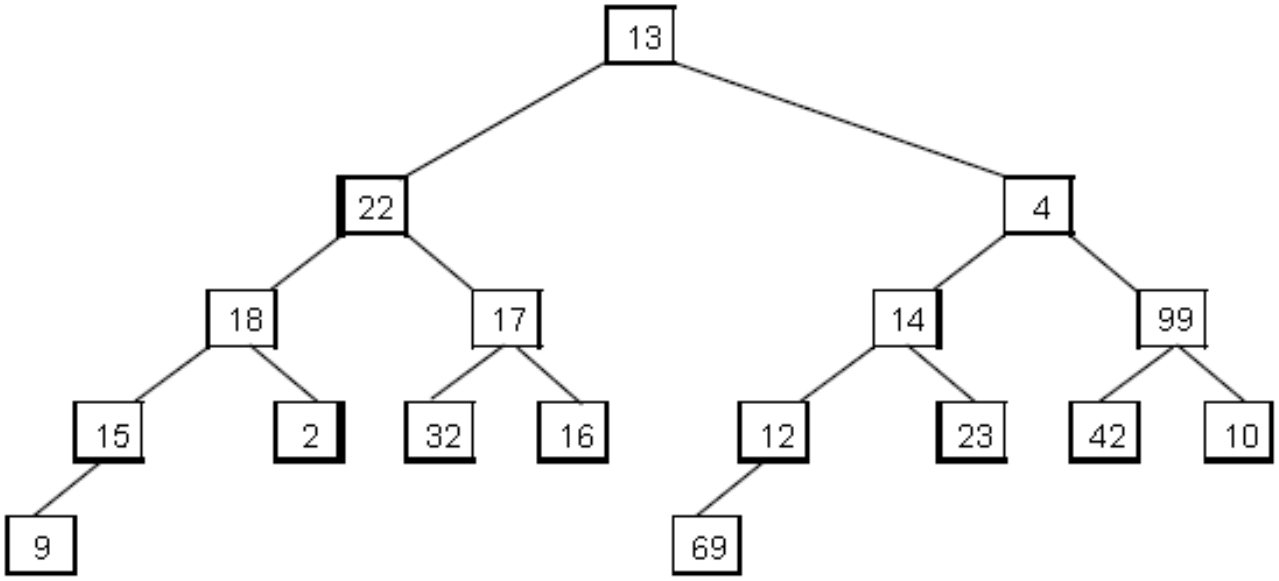
Árbol binario perfectamente balanceado

- 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.

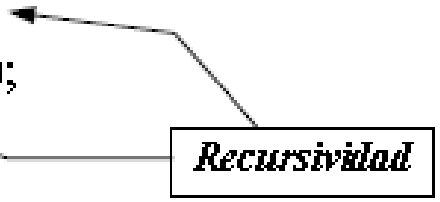
Construir un ABPB conocido el número de nodos:

- Si se conoce el número n de nodos del árbol binario perfectamente balanceado, para cada nodo se construye un subárbol izquierdo perfectamente balanceado de n_izq = n DIV 2 y otro derecho de n_dch = n – n_izq – 1 nodos.

Ejemplo: *Nodos:* *13, 22, 18, 15, 9, 2, 17, 32, 16, 4, 14, 12, 69, 23, 99, 42, 10*

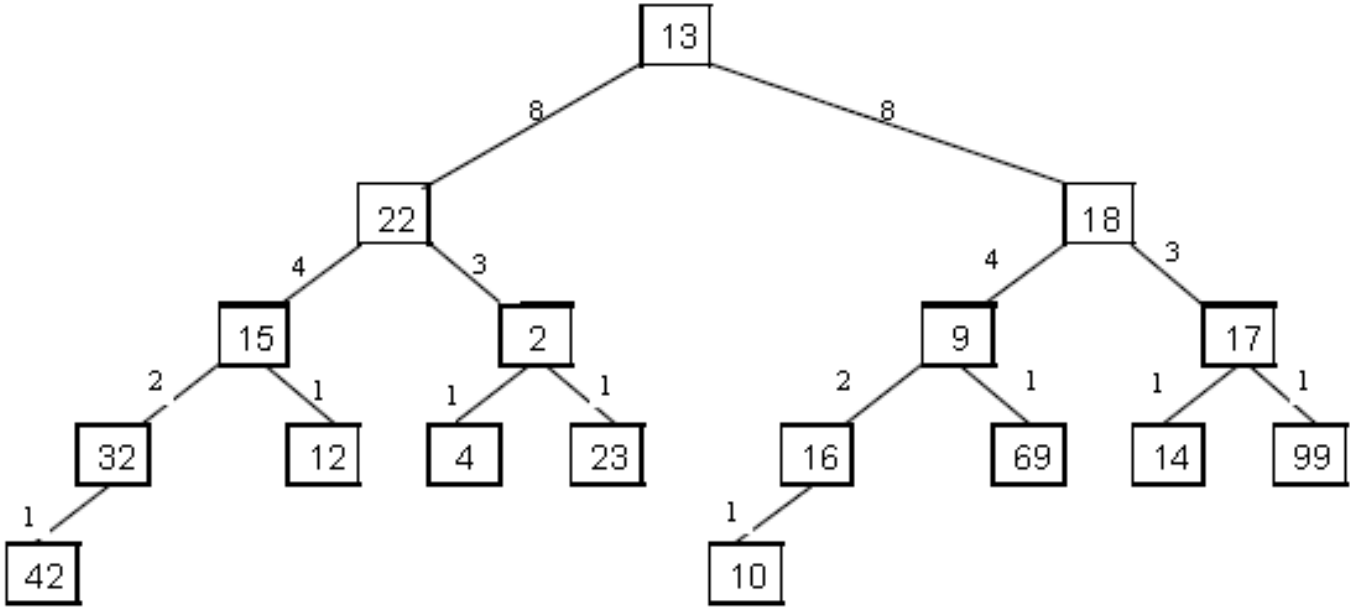


```
PROCEDURE Arbol_perf_bal (n:INTEGER):Ptr_Nodo;
    (*n: numero de nodos a almacenar en el arbol*)
VAR     Nuevo_nodo     : Ptr_Nodo;
        n_izq, n_dch,valor : INTEGER;
BEGIN
   IF n=0 THEN Nuevo_nodo := NIL
   ELSE  n_izq := n DIV 2;
        n_dch := n-n_izq-1;
        ReadInt(valor); (*Lee valor 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;
```



Construir un ABPB sinconocer el número de nodos

- Lo normal será que no conozcamos el número de nodos sino que se vaya construyendo el árbol a medida que se van creando los nuevos nodos.
- En la construcción tendremos que tener en cuenta a la hora de añadir el nuevo nodo la condición de que sea perfectamente balanceado



```
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;
```

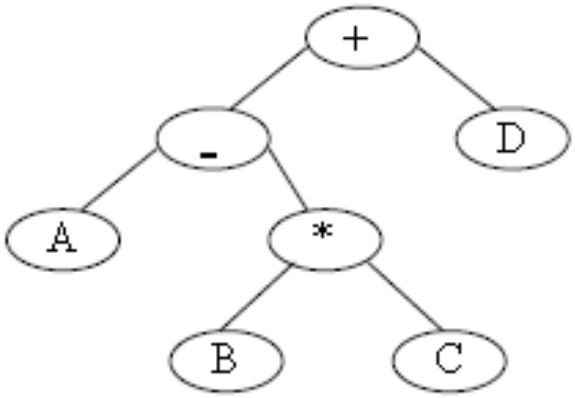
5.3 Árboles binarios ordenados según el recorrido

- Un *árbol binario ordenado según el recorrido es aquél que para cada nodo se visita el nodo,* su subárbol izquierdo y su subárbol derecho en un orden establecido:
  - *Preorden:*
    - *Visitar el nodo antes que los subárboles (NDI, NID)*
  - *En orden:*
    - *Visitar el nodo después de un subárbol y antes que el otro (DNI, IND)*
  - *Postorden:*
    - *Visitar el nodo después de los subárboles (DIN, IDN)*

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

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*BCD**
- 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

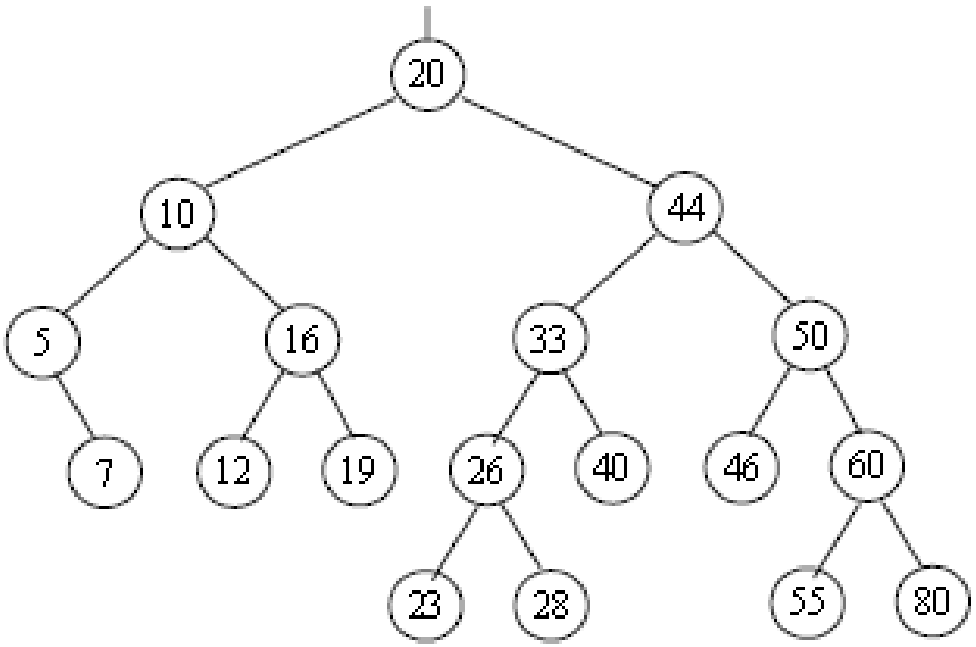


5.4 Árboles de Búsqueda Binarios

- 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)
- Un árbol de búsqueda
  - Es un TDA árbol en el que para cada nodo todos 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)
- Un árbol binario de búsqueda
  - Es un árbol binario en el que dadas dos condiciones mutuamente excluyentes (por ejemplo > y <), para cada nodo, todas las llaves de su subárbol izquierdo satisfacen una condición y todas las de su subárbol derecho la otra

Árboles de búsqueda binarios

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



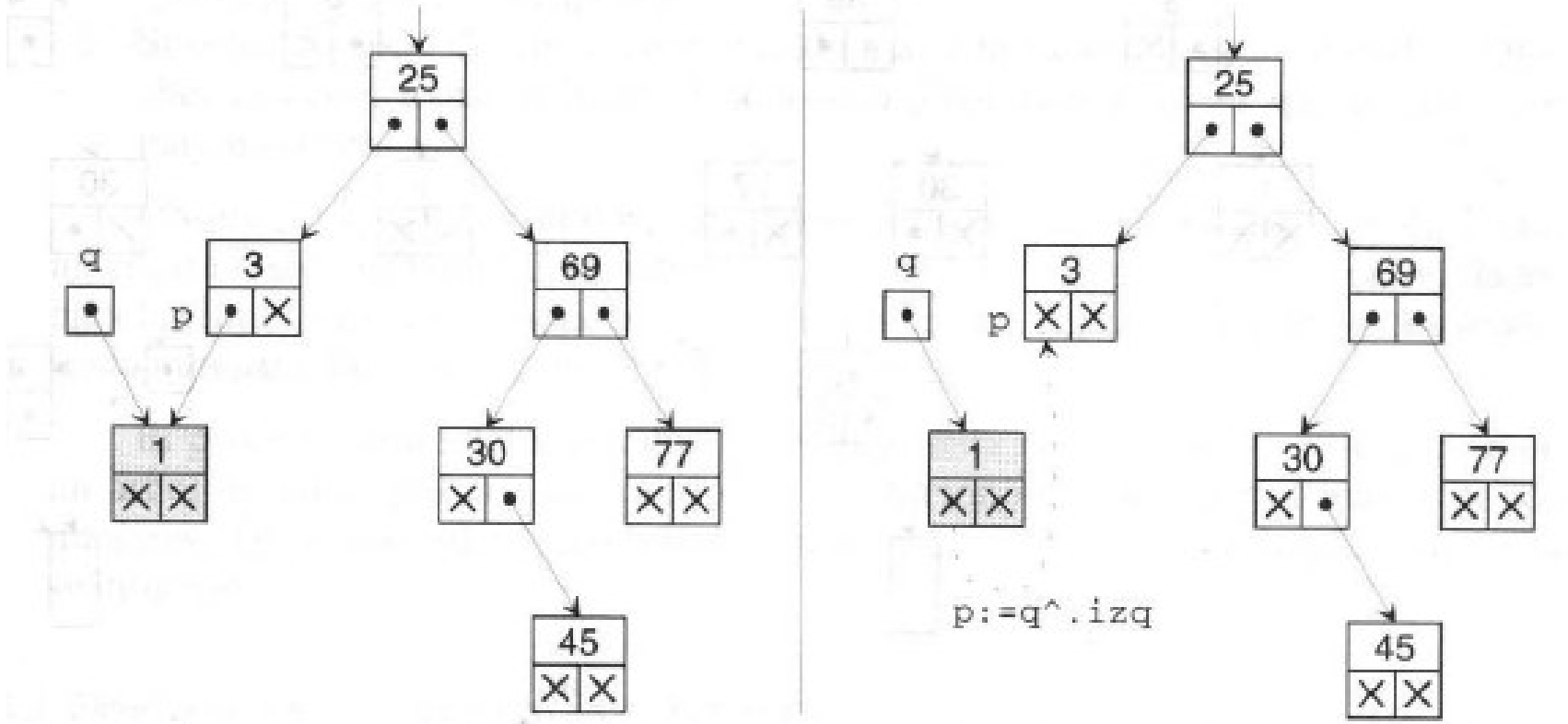
la operación de inserción

- Para insertar un elemento en el árbol de búsqueda, para cada nodo se consulta si el dato es menor o mayor que la llave, decidiendo así si se prosigue la búsqueda por la izquierda o por la derecha respectivamente.
- El procedimiento termina cuando se alcanza el puntero NIL, ya que esto querrá decir que el elemento no se ha encontrado y hay que insertarlo.

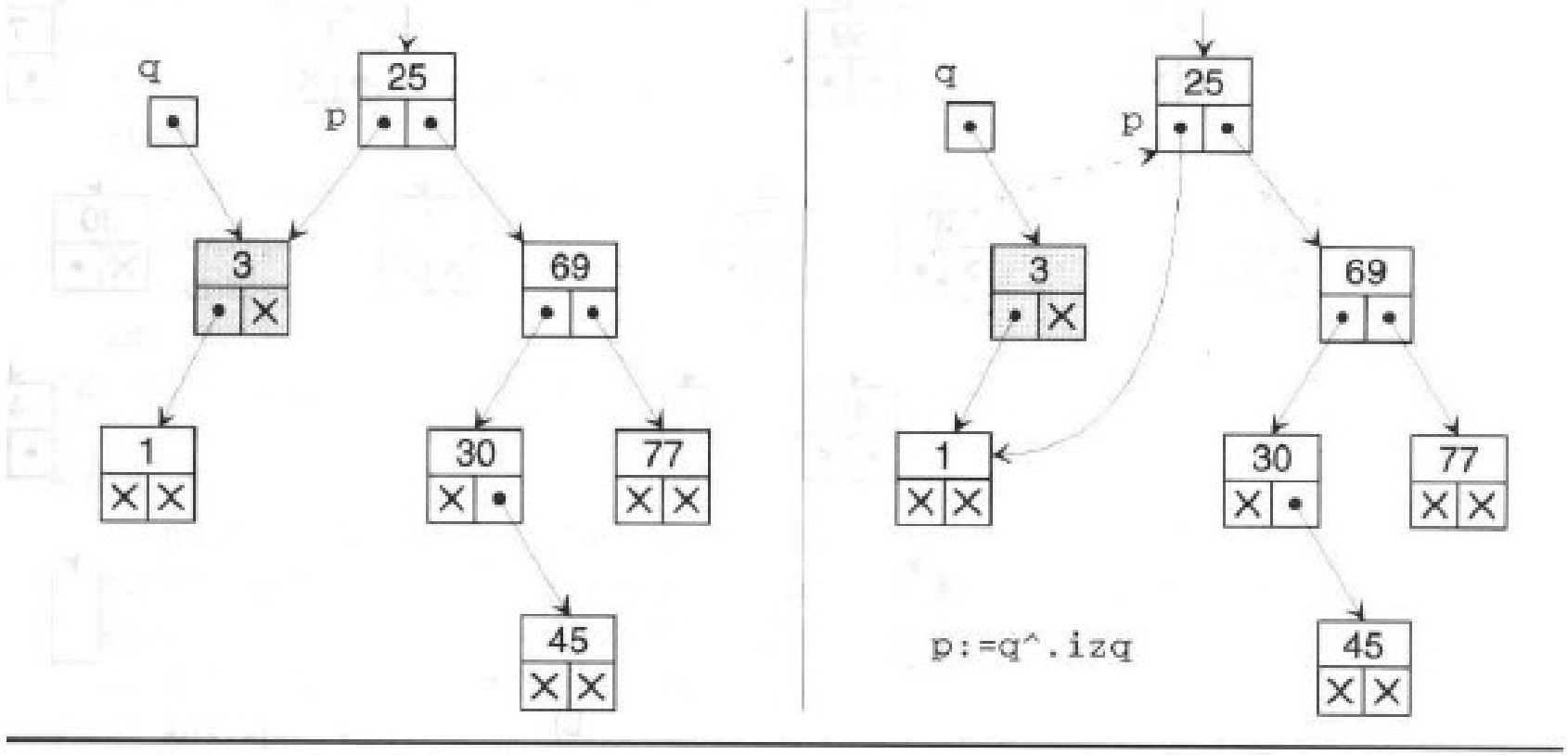
```
PROCEDURE Insertar_busqueda(x:Tipo_datos;VAR puntero:Ptr_Nodo);
  BEGIN
    IF puntero=NIL THEN (* insertar *)
      ALLOCATE(puntero,SIZE(Nodo));
      WITH puntero^ DO
        llave:=x; izq:=NIL; dch:= NIL
      END
    ELSIF x<puntero^.Dato THEN
      Insertar_busqueda(x,puntero^.izq)
    ELSIF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

Eliminación de un nodo

- 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)



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));`

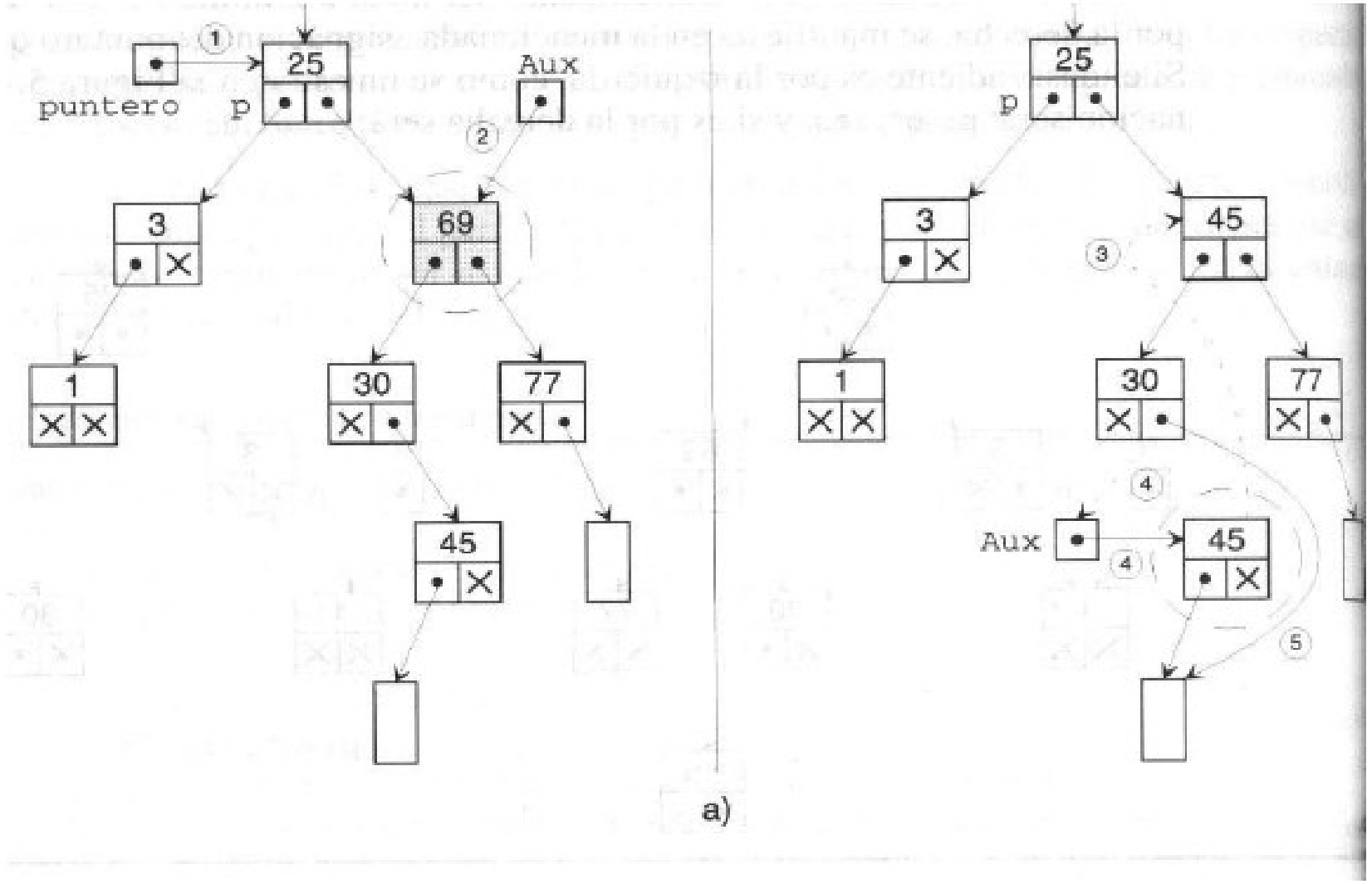


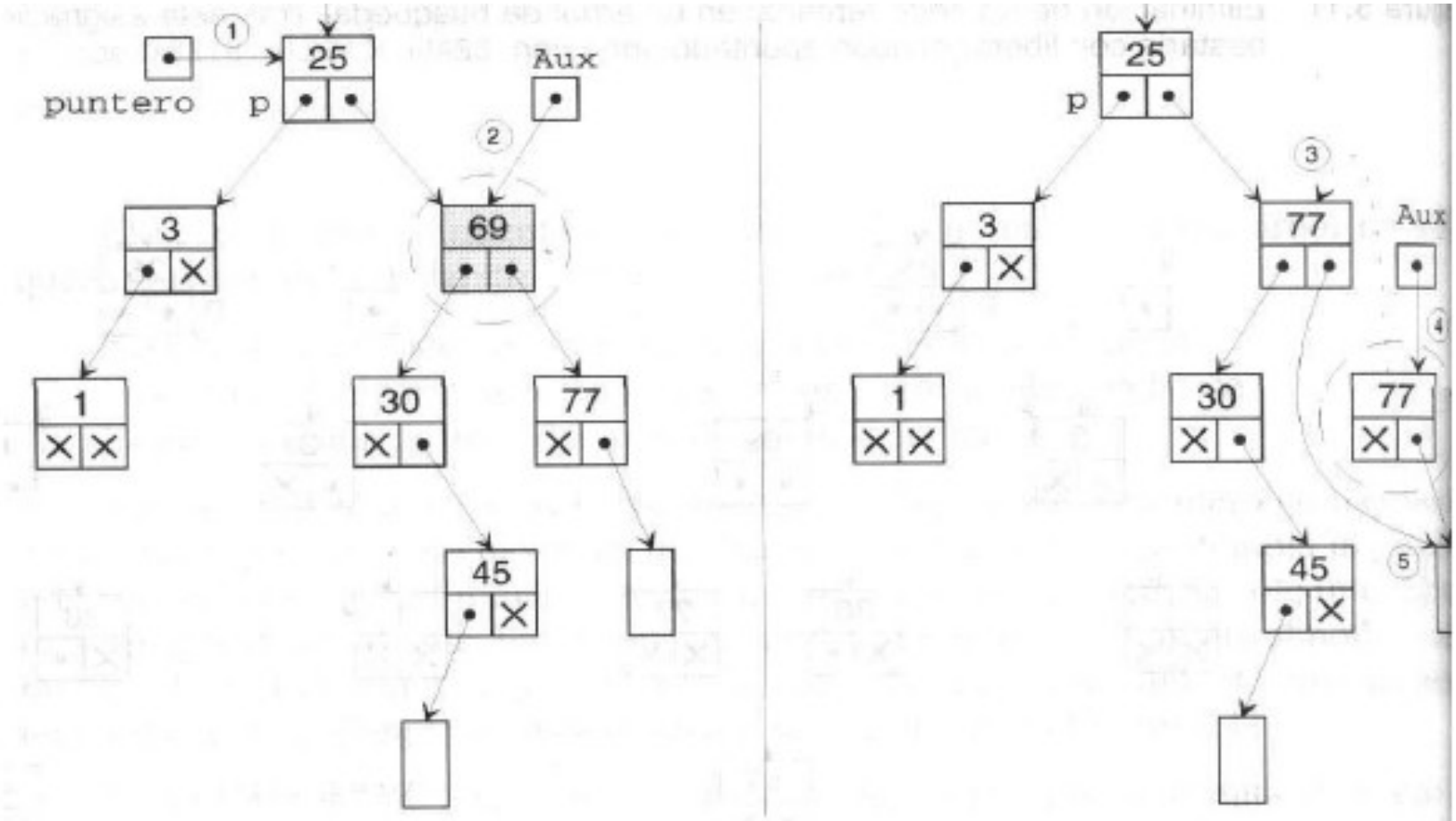
Eliminación de un nodo con un único descendiente, donde puede observarse una situación similar a la mostrada en la Figura 5.11 para un nodo terminal. Si el nodo de llave 3 hubiese tenido el descendiente por la derecha la asignación sería `p:=q^.dch;`

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, substituirlo 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, substituirlo por el nodo de llave **menor de todas las mayores** que él.

Eliminación en un árbol de búsqueda





b)

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.

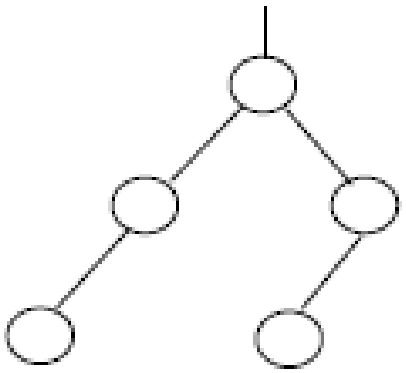
Análisis

- Para buscar en el árbol, el número de comparaciones a realizar dependerá del número de nodos que debe consultarse, o lo que es lo mismo, del recorrido a realizar.
  - En **el peor caso** será *n/2* y se da cuando se genera una lista enlazada.
  - En **el mejor caso** es log *n* cuando el árbol está perfectamente balanceado.
  - El número de comparaciones **promedio** para encontrar una llave en un árbol de búsqueda con *n* nodos es del orden del 39% mayor que las correspondientes a un árbol perfectamente balanceado (aprox. log *n*).
- Por tanto no se justifica el costo necesario para convertir el árbol de búsqueda en un árbol perfectamente balanceado en cada inserción

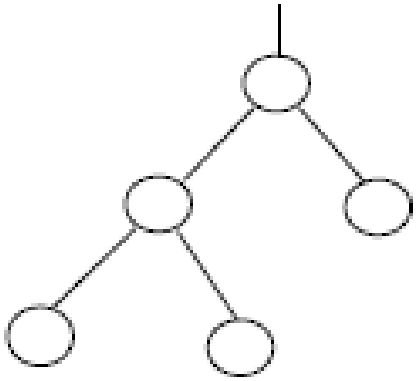
5.5 Árboles de búsqueda balanceados (AVL)

- 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.

todos los árboles perfectamente balanceados son AVL pero no al revés:



Árbol perfectamente balanceado



AVL (no perfectamente balanceado

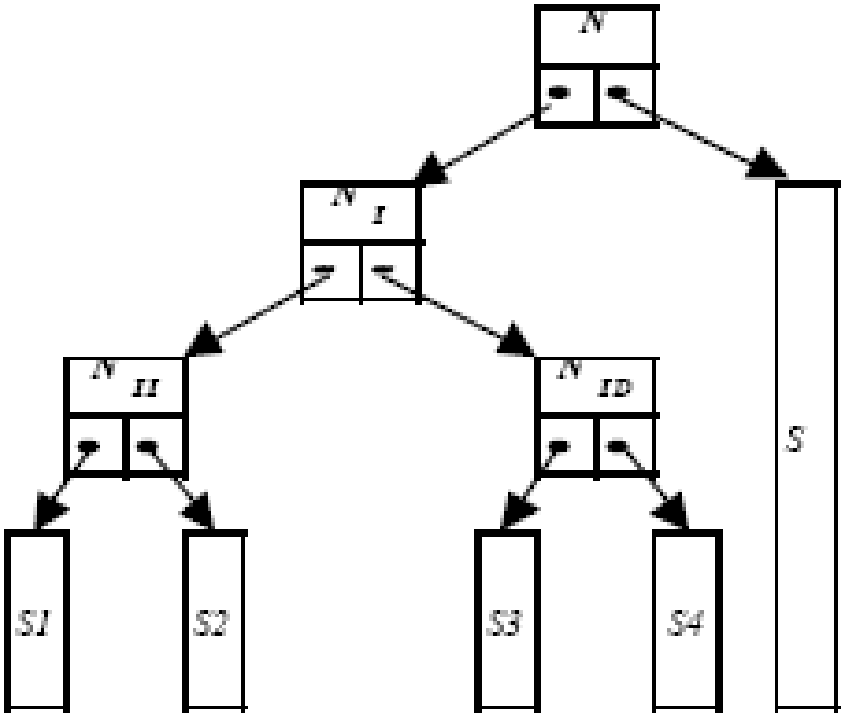
Inserción en árboles balanceados

- Supongamos que se va a insertar un elemento en un subárbol con N como nodo padre y con subárboles I y D terminales de alturas h_I y h_D. Antes de insertar un elemento, N puede encontrarse de tres formas diferentes: $h_I = h_D$, $h_I < h_D$ o $h_I > h_D$
- Consideremos que el nuevo nodo se inserta en I, entonces:
  - 1.- Si N tenía $h_I = h_D$, entonces el árbol seguirá siendo AVL
  - 2.- Si N tenía $h_I < h_D$, entonces el árbol seguirá siendo AVL
  - 3.- Si N tenía $h_I > h_D$ entonces el árbol no será AVL
- En el tercer caso será necesario manipular el árbol para que siga siendo AVL (rebalanceo).
- Para realizar el rebalanceo se deberá guardar información sobre el equilibrio en cada nodo.
  - La siguiente definición de los nodos introduce un campo balance que se calcula como $bal(N)=h_D - h_I$

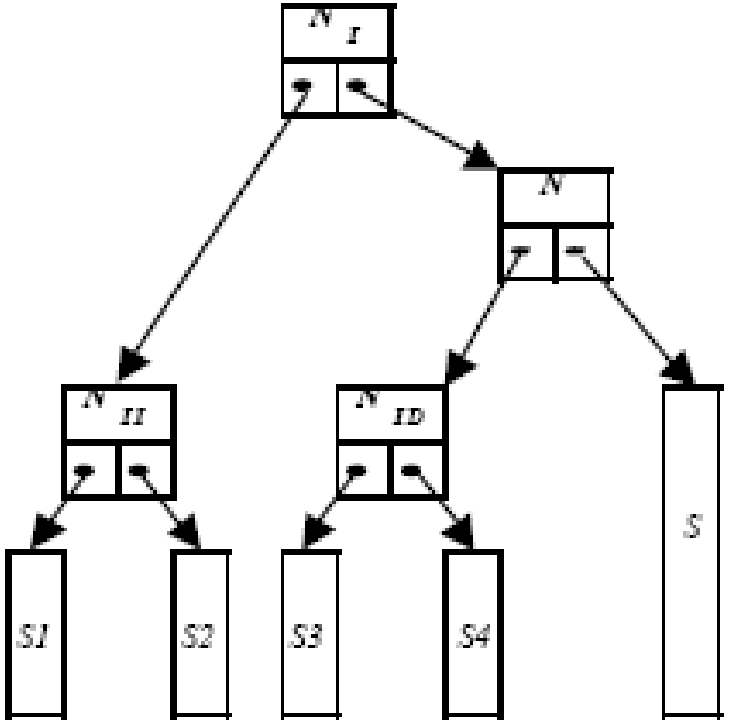
5.5.1 Inserción de un nuevo nodo por la izquierda de un subárbol

- La situación inicial en la que deberá realizarse rebalanceo es:
  - *bal(N) = -1 y bal(N_l) = 0*
- a) El nuevo nodo se inserta *en el subárbol izquierdo de N_l*: **rebalanceo LL - rotación simple**
  - El nodo N_l toma el lugar de *N* y se reasigna el subárbol derecho de N_l al subárbol izquierdo de *N*

Situación inicial



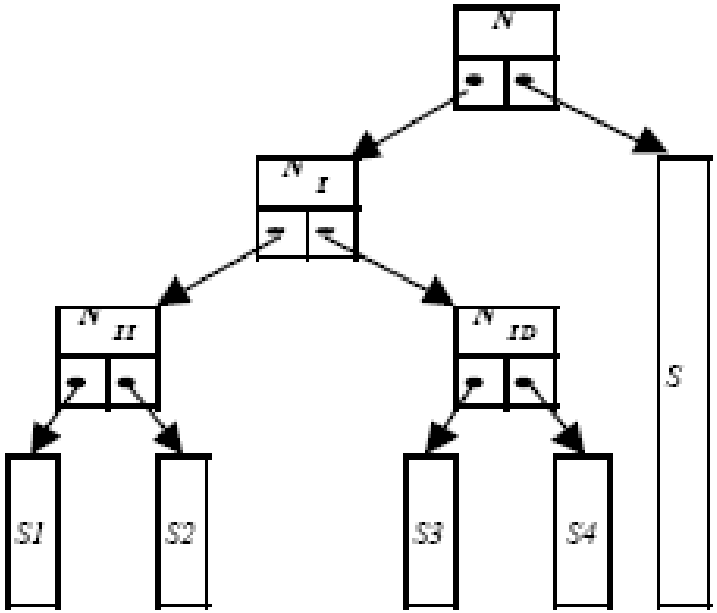
rebalanceo LL



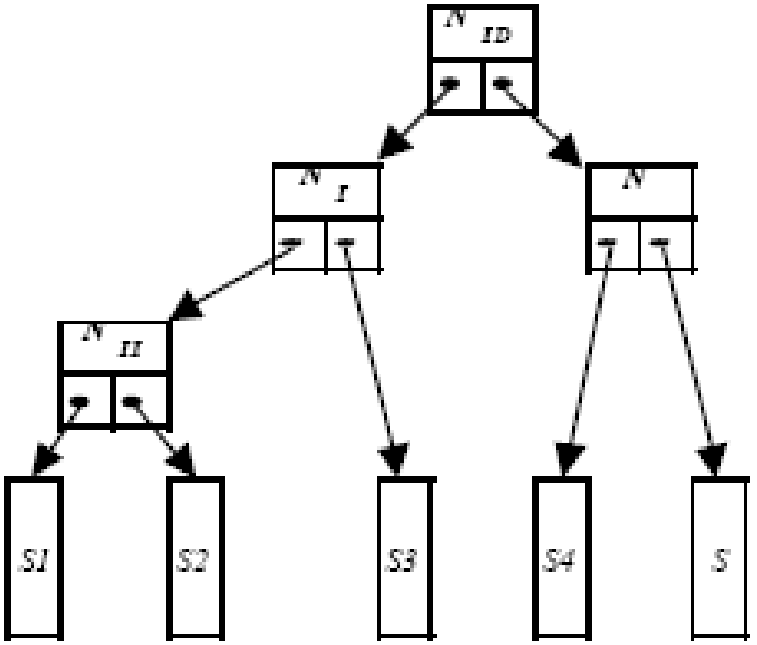
b) El nuevo nodo se inserta en el subárbol derecho de NI: rebalanceo LR - rotación doble

- N_{ID} se coloca entre N y N_I, colocando el nodo N_I como su hijo izquierdo y N como su hijo derecho.
- El subárbol izquierdo de N_{ID} pasa a ser el subárbol derecho de N_I
- y el subárbol derecho de N_{ID} pasa a ser subárbol izquierdo de N

Situación inicial



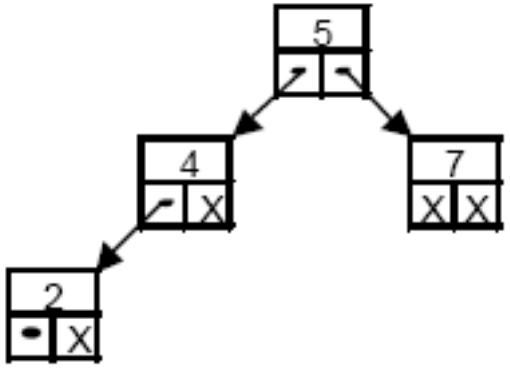
rebalanceo LR

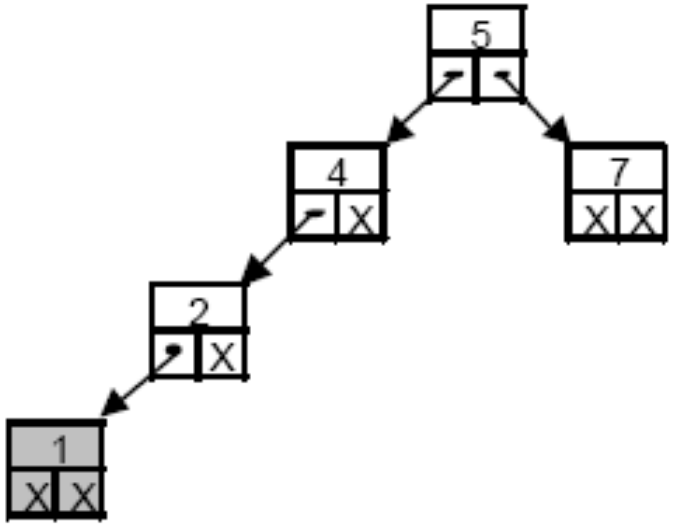


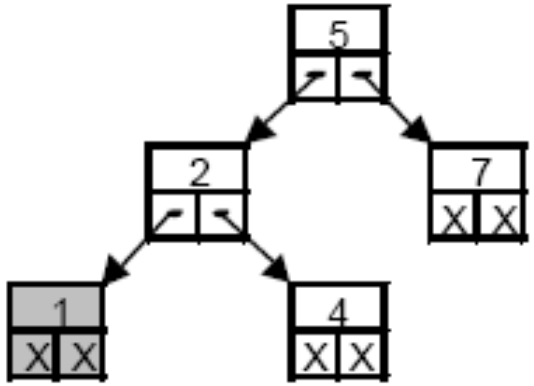
Proceso de Inserción

- El proceso de inserción está formado por tres partes:
  - 1. Buscar siguiendo la trayectoria de búsqueda, con lo que se distinguirá la inserción por la izquierda o por la derecha
  - 2. Insertar el nodo y determinar su balance
  - 3. Retroceder y verificar el factor de balance en cada nodo, realizando el rebalanceo en caso necesario.

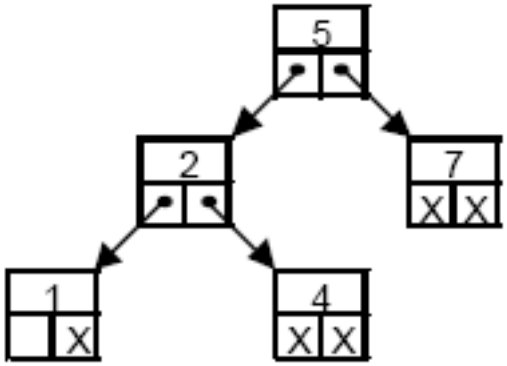
Ejemplos de inserción: Partiendo del árbol de la figura insertar los datos 1, 3 y 6

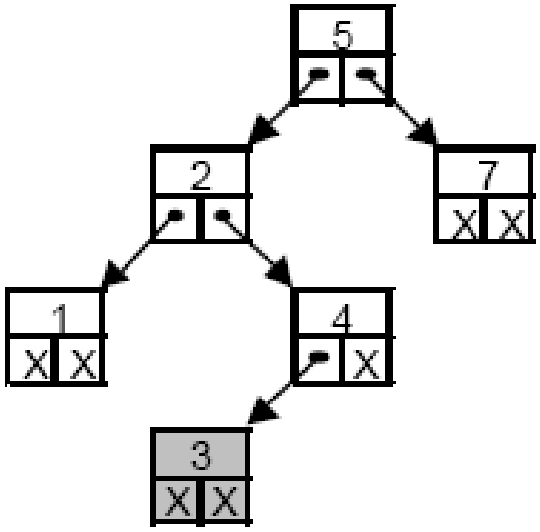


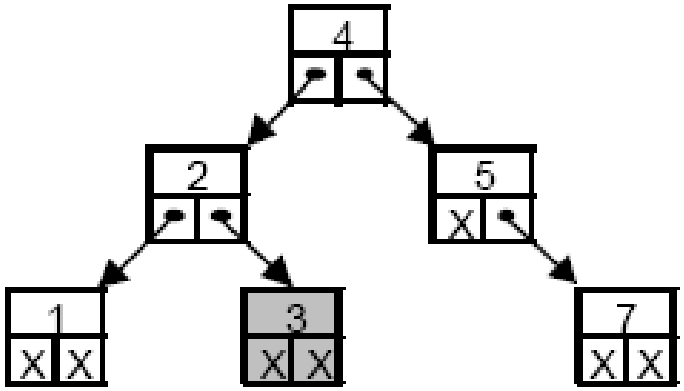




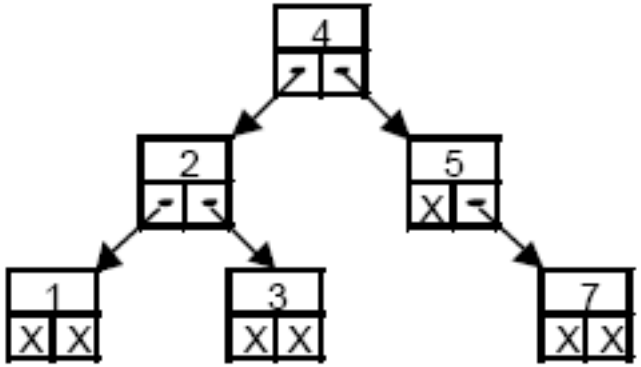
Al insertar el dato 1 se necesita rebalanceo LL para N=4

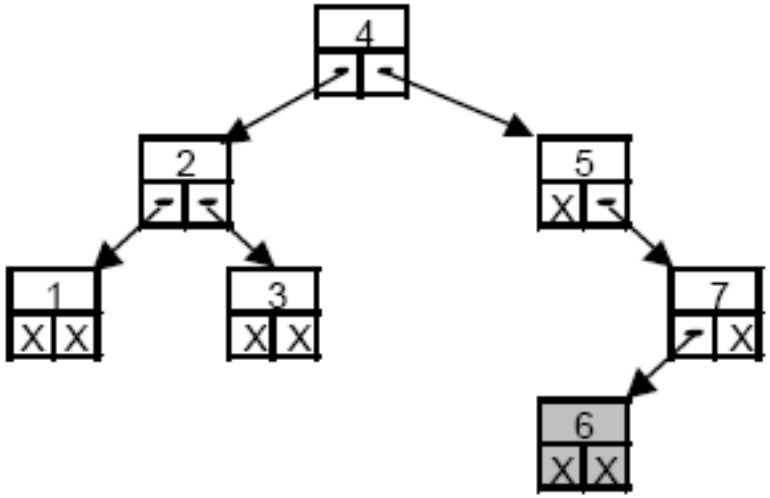


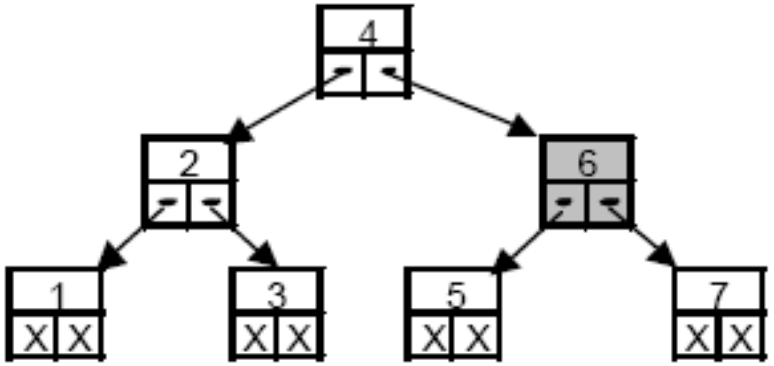




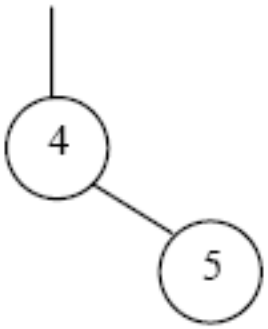
nsertar el dato 3 se necesita rebalanceo LR para N=5



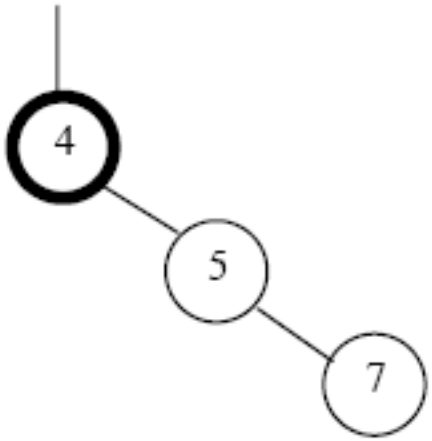




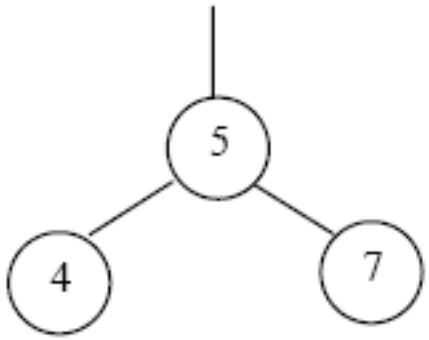
Al insertar el dato 6 se necesita rebalanceo RL para N=5



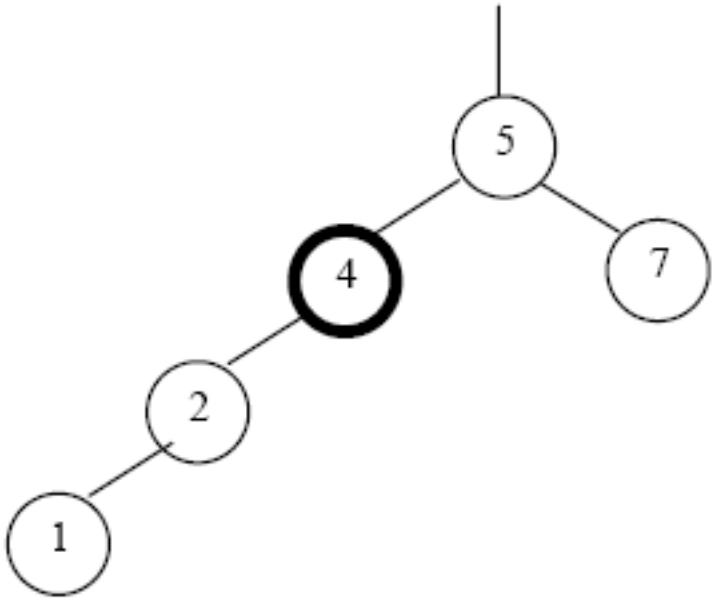
Inicio



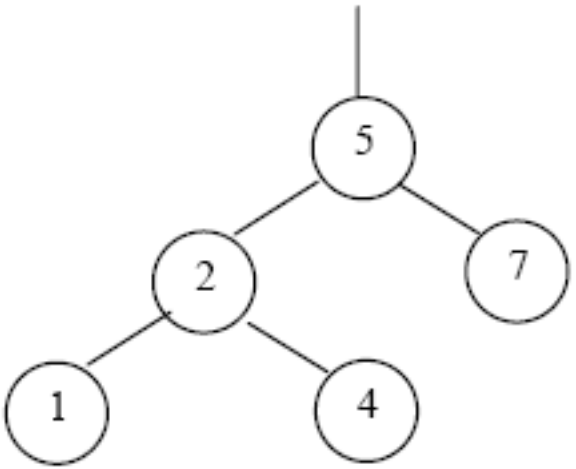
Inserta 7



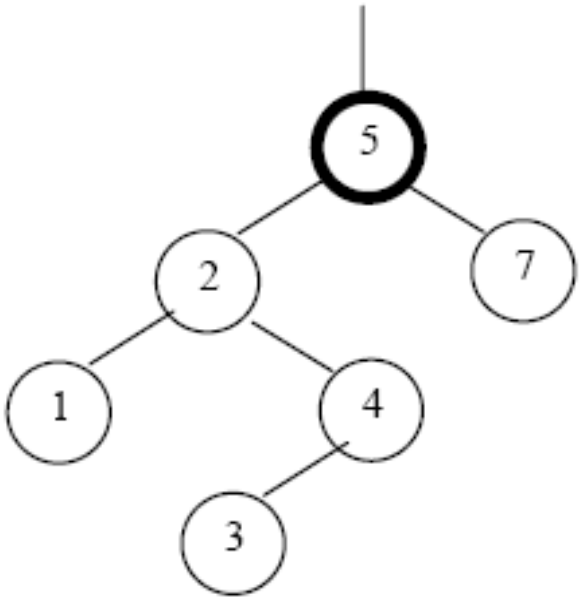
Rebalanceo RR



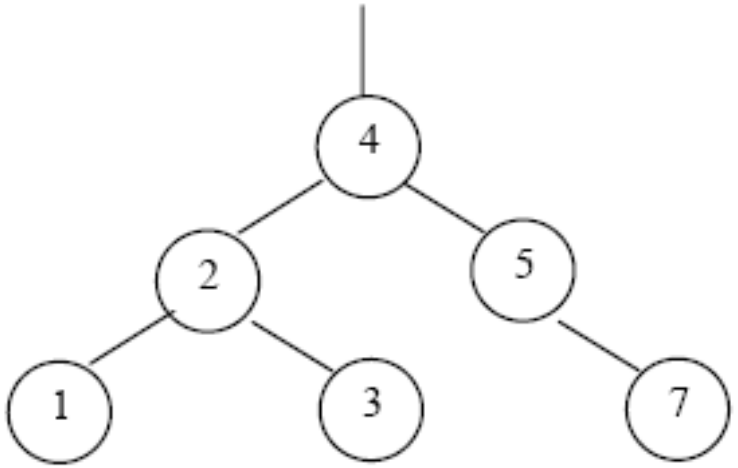
Inserta 2 y 1



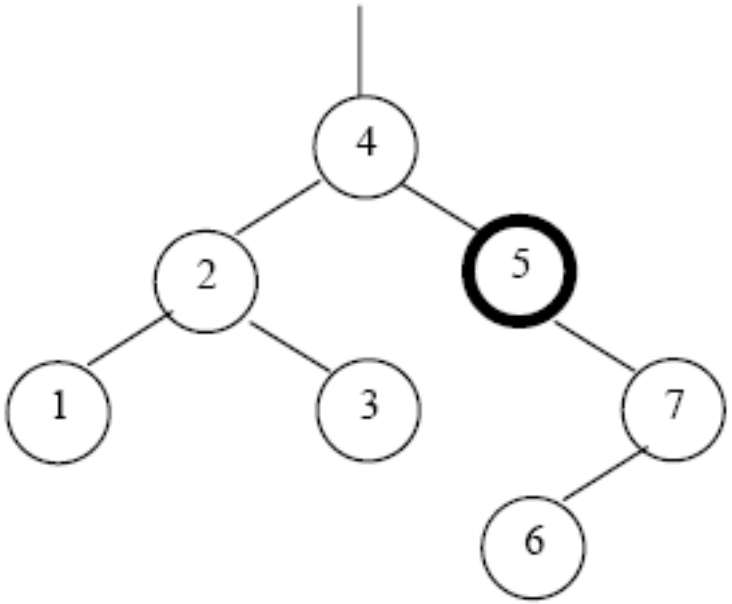
Rebalanceo LL



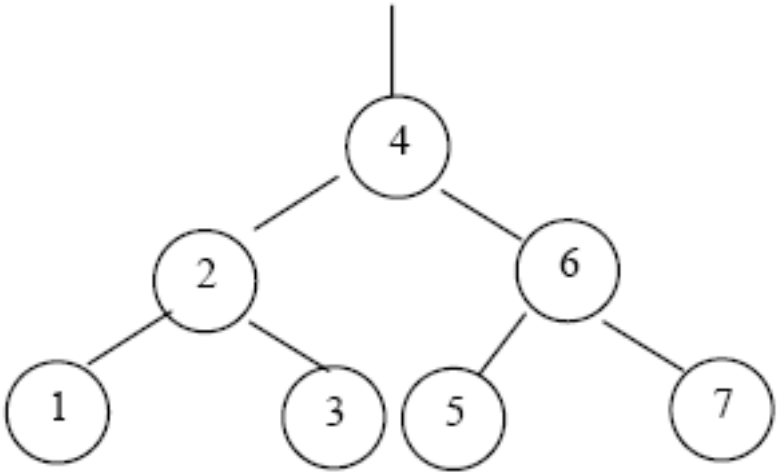
Inserta 3



Rebalanceo LR



Inserta 6



Rebalanceo RL

5.5.2 Eliminación en árboles AVL

- Debe tenerse en cuenta las mismas consideraciones que para la eliminación en los árboles de búsqueda más los rebalanceos necesarios.
- La supresión de los nodos terminales y la de los nodos con un único descendiente es directa.
- Si el nodo que debe suprimirse tiene dos subárboles deberá mantenerse el árbol de búsqueda, sustituyéndolo tras su eliminación por el nodo más a la izquierda de su subárbol derecho o el más a la derecha de su subárbol izquierdo.
- Tras la sustitución del nodo a eliminar la altura habrá cambiado y tendremos que inspeccionar el árbol y hacer los rebalanceos necesarios.

- 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

- Cuando se elimina un nodo por la *izquierda*
  - será necesario rebalanceo cuando el balance del nodo *N* sea 1, puesto que al eliminar el nodo por la izquierda el subárbol quedaría *cargado* en 2 por la derecha.
  - El rebalanceo necesario es *RR*.
- Cuando el balance de *N* es 0, al eliminar por la izquierda aumenta a 1 pero su altura no ha disminuido.
- Cuando el balance de *N* es –1, pasa a 0 y ha variado la altura del árbol.
  - Será una rotación *simple* o *doble* dependiendo del balance del descendiente derecho de *N, ND*.
  - Si es 1 la rotación es *RR*, si es 0 también es *RR* pero los balances son diferentes puesto que *ND* tiene subárboles derecho e izquierdo; y si es –1, entonces la rotación es *RL*.

- Cuando se elimina un nodo por la *derecha* será necesario rebalanceo cuando el balance del nodo *N* sea –1, puesto que al eliminar el nodo por la derecha el subárbol quedaría *cargado* en 2 por la derecha.
  - El rebalanceo necesario es *LL*.
- Si el balance de *N* es 0, al eliminar por la derecha pasa a –1 pero su altura no ha disminuido.
- Cuando el balance de *N* es 1, pasa a 0 y ha variado la altura del árbol. Será una rotación *simple* o *doble* dependiendo del balance del descendiente izquierdo de *N, NI*.
  - Si es 1 la rotación es *LL*, si es 0 también es *LL* pero los balances son diferentes puesto que *NI* tiene subárboles derecho e izquierdo; y si es 1, entonces la rotación es *LR*.

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 $h_{esp}=lg_2(n+0.25)$.

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 $lg_2\ n$.