Algoritmos
y
Estructuras de Datos

Cursada 2015

Prof. Alejandra Schiavoni
Prof. Catalina Mostaccio

Facultad de Informática – UNLP

Árboles Binarios de Búsqueda: AVL

Agenda

- Árboles AVL
  - Definición
  - Características
  - Desbalanceo
  - Rotaciones simples y dobles

Árbol AVL: Definición

Un árbol AVL (Adelson–Velskii–Landis) es un árbol binario de búsqueda que cumple con la condición de estar balanceado

La propiedad de balanceo que cumple dice:

Para cada nodo del árbol, la diferencia de altura entre el subárbol izquierdo y el subárbol derecho es a lo sumo 1

Características

- *La propiedad de balanceo es fácil de mantener y garantiza que la altura del árbol es de O(log n)*
- *En cada nodo del árbol se guarda información de la altura*
- *La altura del árbol vacío es -1*

Operaciones en un AVL

- *Búsqueda/Recuperación*
- *Inserción*
- *Eliminación*

*al insertar o eliminar un dato del AVL se **puede** perder la propiedad de **balanceo***

*Se debe **preservar** el **balanceo** al realizar estas operaciones sobre el árbol.*

Problemas: Desbalanceo

- *La inserción se realiza igual que en un árbol binario de búsqueda*

- *Puede destruirse la propiedad de balanceo*

↓

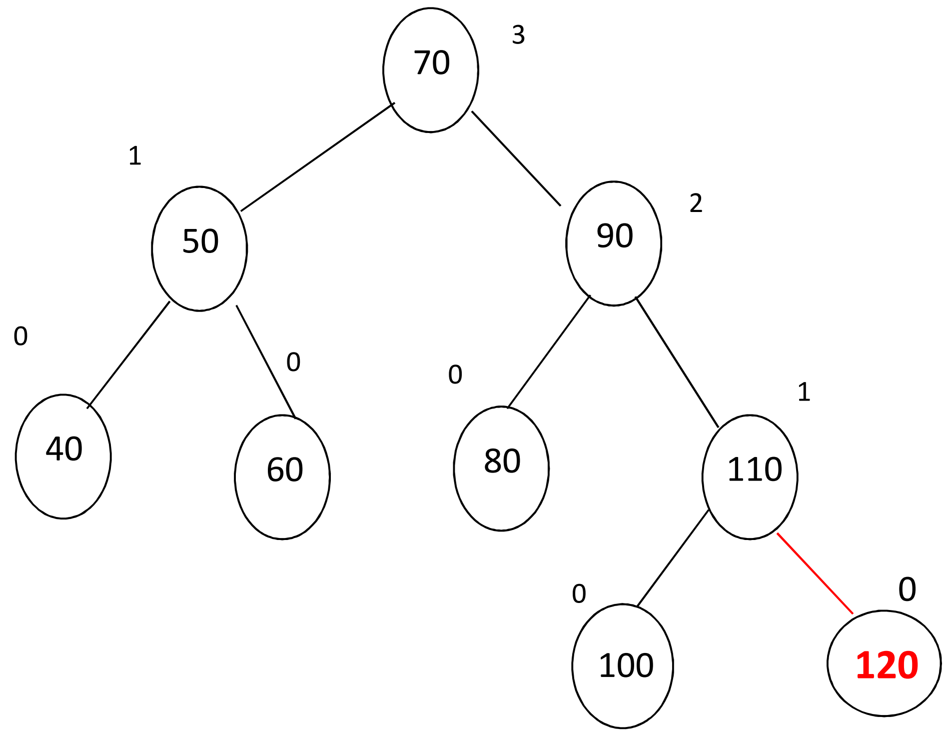
Rebalancear el árbol

Problemas: Desbalanceo

- *Al insertar un elemento se actualiza la información de la altura de los nodos que están en el camino desde el nodo insertado a la raíz*

Desbalanceo del árbol

***Ejemplo** que no produce desbalanceo al Insertar la clave 120*

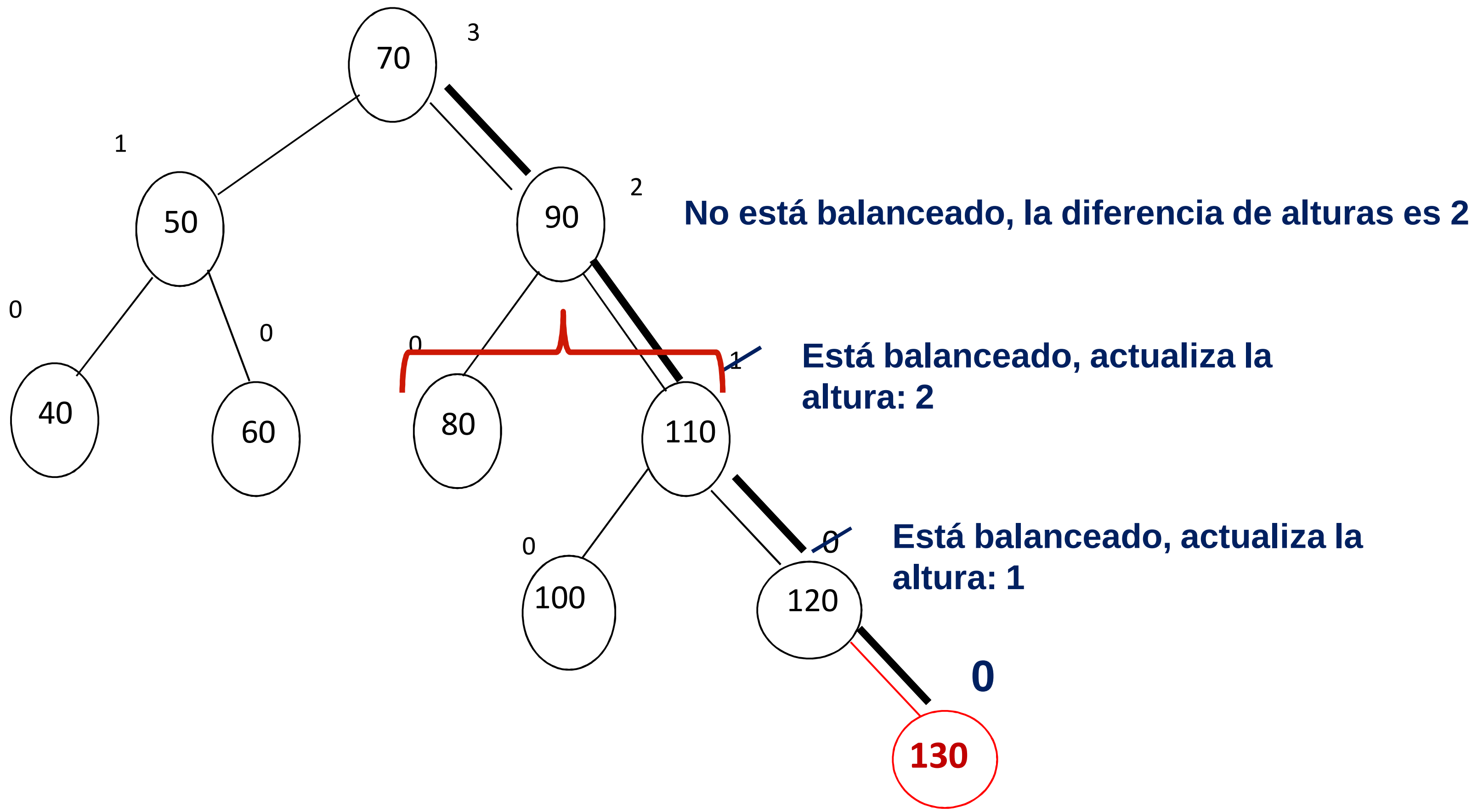


Problemas: Desbalanceo

- *El desbalanceo sólo se produce en el camino desde el nodo insertado a la raíz, ya que sólo esos nodos tienen sus subárboles modificados*

Desbalanceo del árbol

Ejemplo *de desbalanceo en el camino desde la clave 130 insertada hasta la raíz.*



Rebalanceo del árbol

Para restaurar el balanceo del árbol:

- *se recorre el camino de búsqueda en orden inverso*
- *se controla el equilibrio de cada nodo*
- *Si está desbalanceado se realiza una modificación simple:* ***rotación***
- *después de rebalancear el nodo, la inserción termina*
- *se debe actualizar la altura del resto de los nodos en el camino de vuelta hasta llegar a la raíz*

Rebalanceo del árbol

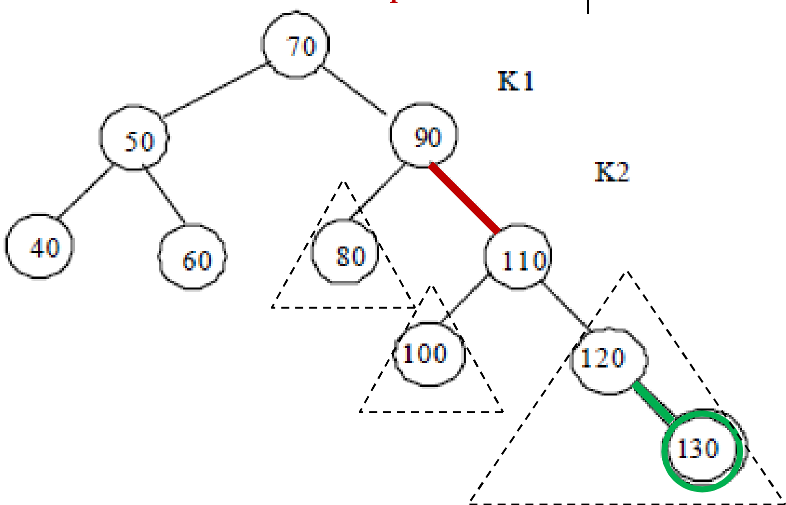
- *La solución para restaurar el balanceo es la **ROTACION***

  *La **rotación** es una modificación simple de la estructura del árbol, que restaura la propiedad de balanceo, preservando el orden de los elementos*

Rebalanceo del árbol

***Ejemplo** de rotación simple :* *<u>Rotación Simple</u> con Derecho al Insertar clave 130*

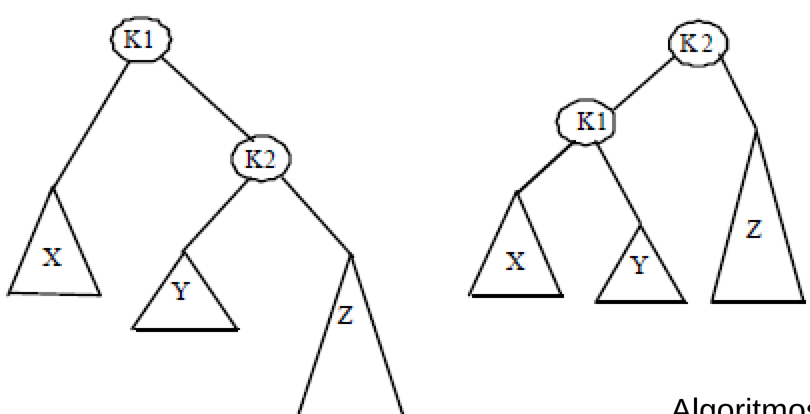
la altura del subárbol der. de K2 > la altura del subárbol izq. de K2



K1 : raíz del árbol desbalanceado
K2 := k1.der;
K1.der := k2.izq;
K2.izq := k1;

K1.altura:=max (altura (k1.izq),altura (k1.der))+1;
K2.altura :=max (altura (k2.der),k1.altura) + 1;

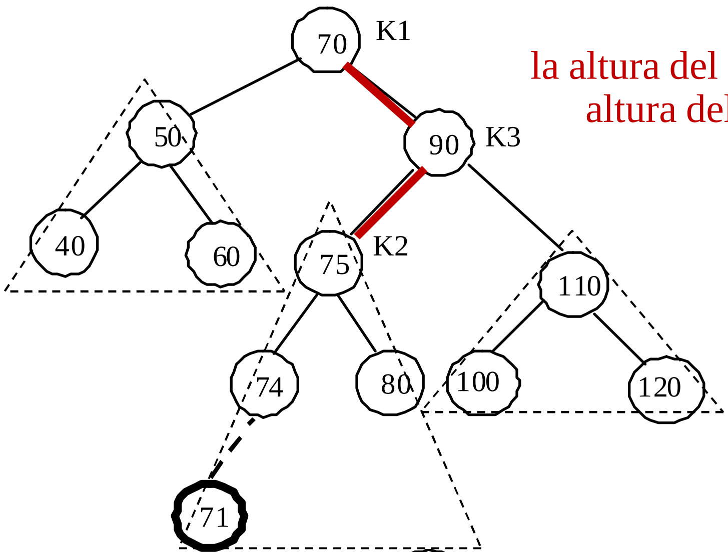
K1 := k2 ; { nueva raíz }



K1 < K2
todos los datos en X < K1 (<K2)
todos los datos en Z > K2 (> K1)
K1 < todos los datos en Y < K2

Rebalanceo del árbol

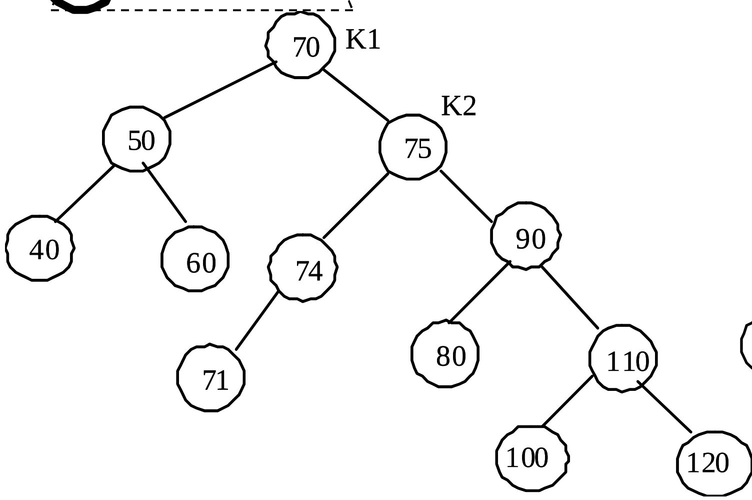
***Ejemplo** de rotación doble :* *<u>Rotación doble</u> con Derecho al Insertar la clave 71*

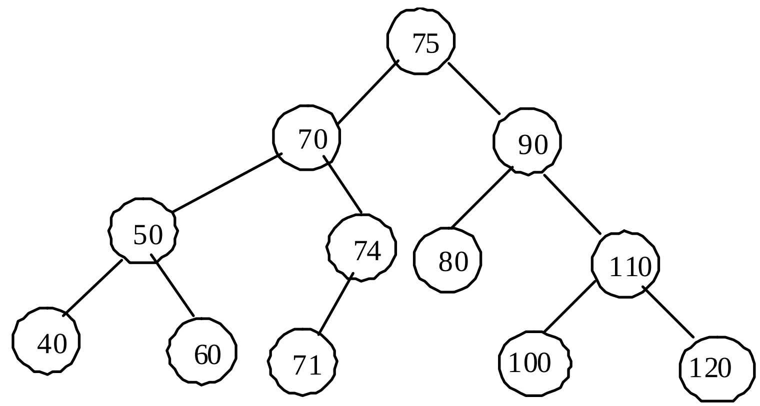


la altura del subárbol izq. de K3 > la altura del subárbol der. de K3

i) Rotación Simple Izq. (90) : RSI (K1.der)

ii) Rotación Simple Der. (70) : RSD (K1)

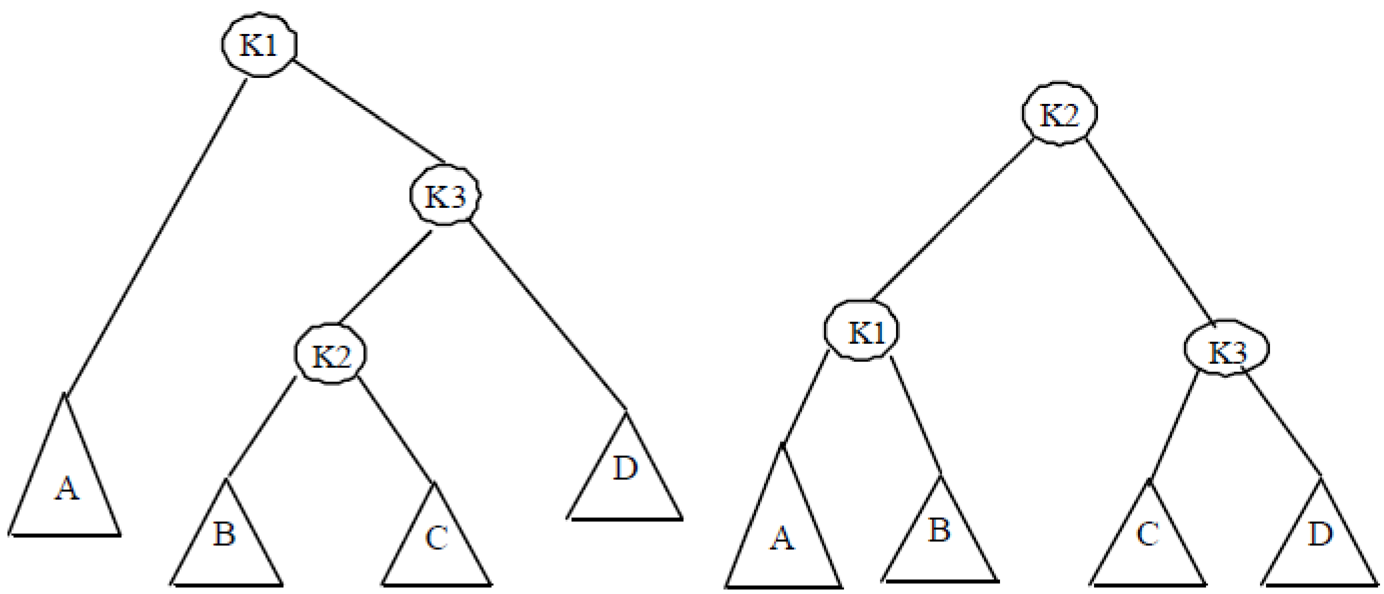




Rebalanceo del árbol

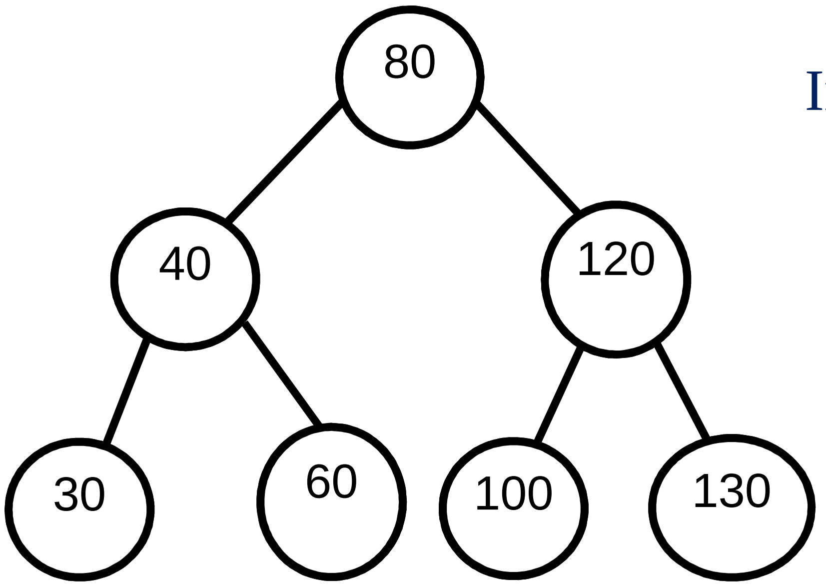
Ejemplo de rotación doble : *Rotación doble con Derecho al Insertar la clave 71*

Rotación Doble Derecho (R-S Izquierdo (hijo derecho(K1)) , R-S Derecho (K1))



Árbol AVL – Insert (cont.)

Ejemplo y ejercitación



Insertar claves :

140,125, 150 (RSD 120)

20, 15 (RSI 30)

35 (RDI 40)

126,124, 122 (RDD120)

Tarea :

78,47,23,14,99,5,60,3,45,44,110,130,4

Árbol AVL - Insert

```
private Avl_nodo insert ( Comparable x , Avl_nodo t ) {
    if (t == null)
            // Insertar una hoja como en el ABB más el campo de “altura =0”
            t = new ……
    else
          if (x < t.dato)
                insert ( x, t.izq ) ;
                              /*  Balancear si es necesario   */
            /*  Actualizar altura  del  nodo  raíz del árbol  T  - siempre   */
           else
                if   (x >  t .dato)
                      insert ( x, t .der ) ;
                              /*  Balancear si es necesario   */
            /*  Actualizar altura  del  nodo  raíz del árbol  T  - siempre   */
                else;  //  x  está en el árbol y no hay nada que hacer
  return t;
  }
```

Árbol AVL – Insert (cont.)

/* Balancear si es necesario   */

Si   **abs (altura (t.izq) - altura (t.der)) = 2**

entonces se desbalanceó el subárbol con raíz en el nodo t

/* Actualizar altura del nodo raíz del árbol t - siempre   */

t.altura := max (altura (t.izq) , altura (t.der)) + 1;

Árbol AVL – Insert (cont.)

i
z
q
u
i
e
r
d
o

```
                                   /*  Balancear si es necesario   */
if (altura(t.izq) - altura(t.der)  ==  2 )

             balancear !!!!! ????

  /*  Actualizar altura  del  nodo  raíz del árbol  t  - siempre
else
   t.altura := max(altura (t.izq),altura (t.der)) + 1;
```

```
                                   /*  Balancear si es necesario   */
if (altura(t.der) – altura(t.izq)  ==  2)

             balancear !!!!! ????

/*  Actualizar altura  del  nodo  raíz del árbol  t  - siempre
else
   t.altura := max(altura (t.der),altura (t.izq) ) + 1;
```

d
e
r
e
c
h
o

Árbol AVL - altura

```
private int altura (avl_nodo p )  {
   /* retorna el valor de la altura de p */
        if  p == nil {
                    return -1;
                    } else {
                    return  p.getAltura();
         }
}
```

Árbol AVL – Insert (cont.)

i
z
q
u
i
e
r
d
o

```
                                          /*  Balancear si es necesario   */
if (  altura (t.izq) - altura (t.der) = =  2 )
     if    (x  <  t.izq.dato)
                         rotación_s_izq(t) {subárb. extremo izquierdo }
     else
                         rotación_d_izq(t) {subárb. medio }
     /*  Actualizar altura  del  nodo  raíz del árbol  t  - siempre   */
else
     t.altura := max (altura (t.izq),altura (t.der) ) + 1;
```

```
                                          /*  Balancear si es necesario   */
if (  altura (t.der) - altura (t.izq)  ==  2)
     if    (x  >  t.der.dato)
                         rotación_s_der(t) {subárb. extremo derecho }
     else
                         rotación_d_der(t) {subárb. medio }
     /*  Actualizar altura  del  nodo  raíz del árbol  t  - siempre   */
else
     t.altura := max (altura (t.der),altura (t.izq) ) + 1;
```

d
e
r
e
c
h
o

Árbol AVL – Insert (cont.)

```
private Avl_nodo balancear ( Avl_nodo t ) {
    if (altura (t.izq ) - altura (t.der )  ==  2 )
      if   (x  <  t.izq.dato)
               rotación_s_izq (t) {subárb. extremo izquierdo }
       else
               rotación_d_izq(t) {subárb. medio }

                                else
if (altura (t.der ) - altura (t.izq )  ==  2)
      if   (x  >  t.der.dato)
               rotación_s_der (t) {subárb. extremo derecho }
      else
               rotación_d_der (t) {subárb. medio }

}
```

/* Actualizar altura del nodo raíz del árbol T - siempre   */

```
t.altura := max (altura(t.der),altura(t.izq) ) + 1;
```

Árbol AVL – Insert (cont.)

```
private Avl_nodo insert ( Comparable x , Avl_nodo t ) {
    if (t == null)
            // Insertar una hoja como en el ABB más el campo de “altura =0”
            t = new ……
   else
            if (x < t.dato)
                insert ( x, t.izq ) ;
            else
                if (x > t.dato)
                        insert ( x, t.der ) ;
                else; // x está en el árbol y no hay nada que hacer
   // Balancear si es necesario
   t = balancear ( T );
   // Actualizar altura del nodo raíz de árbol t - siempre
   t.altura = max (altura (t.der) , altura (t.izq) ) + 1;
   return t;
}
```

Ejercitación

Árbol AVL

i.- Construir un AVL con las siguientes claves:

77,46,22,13,98,5,60,3,45,44,110,130,4

Árbol AVL

Delete

Árbol AVL - Delete

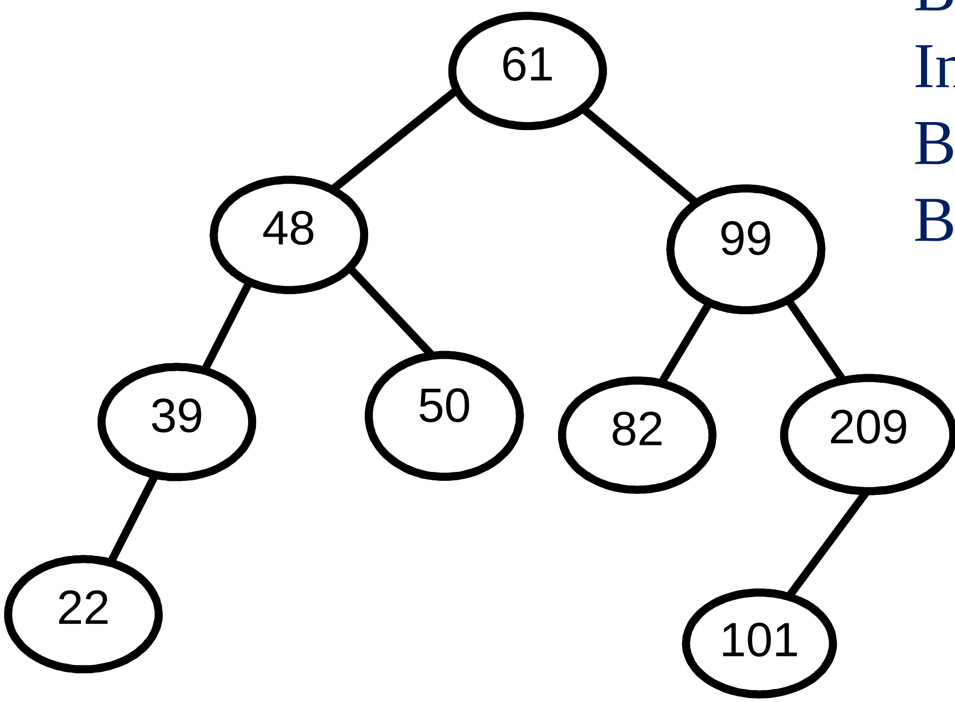
```
private Avl_nodo delete ( Comparable x , Avl_nodo t ) {
    if  t == null
            return t;
    else
            if  (t.dato == x)

                        /* Borrar según los 3 casos posibles */

            else
               if  (x <  t.dato)
                        delete (x, t.izq ) ;
               else
                        delete (x, t.der ) ;
return t;
}
```

Árbol AVL – Delete (cont.)

Ejemplo



Borrar claves : 50 (RSI 48)
Insertar 17,46,3 (RSI 22) // 20, 25
Borrar 82 (RDD 99 y RSI 61)
Borrar 17, 39

Árbol AVL – Delete (cont.)

```
private Avl_nodo balancear ( Avl_nodo t ) {
    if (altura (t.izq ) - altura (t.der )  ==  2 )
     if   (x  <  t.izq.dato)
            rotación_s_izq(t) {subárb. extremo izquierdo }
      else
            rotación_d_izq(t) {subárb. medio }

                         else
if (altura (t.der ) - altura (t.izq )  ==  2)
     if   (x  >  t.der.dato)
            rotación_s_der(t) {subárb. extremo derecho }
     else
            rotación_d_der(t) {subárb. medio }
}
```

Este procedimiento YA NO NOS SIRVE pues no existe un "x" insertado para determinar si es rotación Simple o Doble ! ! !

Árbol AVL – Balancear (c/alturas)

```
/* Método Balancear considerando "alturas" de los
subárboles
                                              */
private Avl_nodo balancear ( Avl_nodo t ) {
    if ( altura(t.izq) - altura(t.der) == 2 )
        if (altura(t.izq.izq) >= altura(t.izq.der))
                rotación_s_izq (t);
        else
                rotación_d_izq(t);
    if ( altura(t.der) - altura(t.izq) == 2)
        if (altura (t.der.der) >= altura(t.der.izq))
                rotación_s_der ( t );
        else
                rotación_d_der ( t );
return t;
}
```

Árbol AVL – Delete (cont.)

```
private Avl_nodo delete ( Comparable x , Avl_nodo t ) {
      if (t== null)
              return t;
      else
              if (x < t.dato)
                     delete (x, t.izq);
              else
                     if (x > t.dato)
                            delete(x, t.der);
                     else
                            if (t.izq != null & t.der != nil) // Dos hijos
                                       t.dato = delete_min(t.der);
                            else
                                       t = (t.izq != null) ? t.izq : t.der;
              t = balancear(t);
              t.altura = max (altura(t.der),altura(t.izq)) + 1;
              return t;
}
```

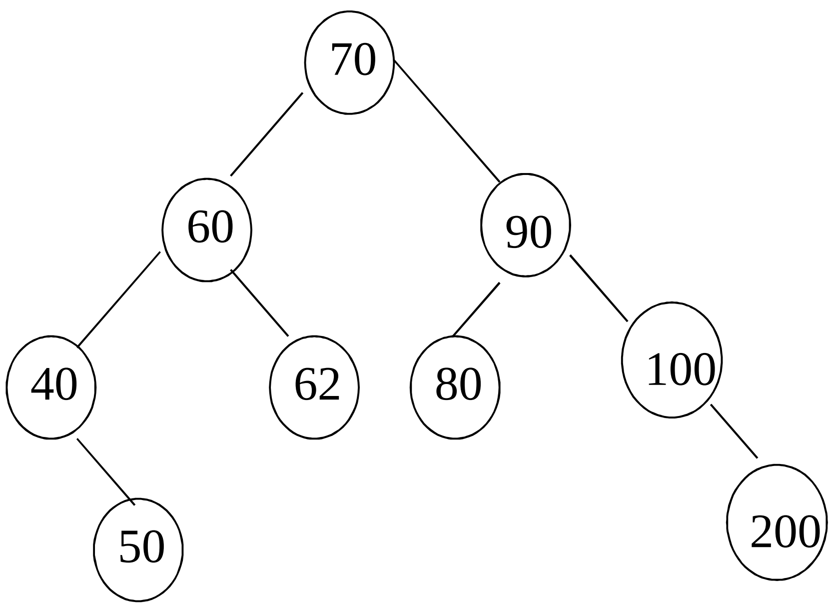
Árbol AVL – Delete (cont.)

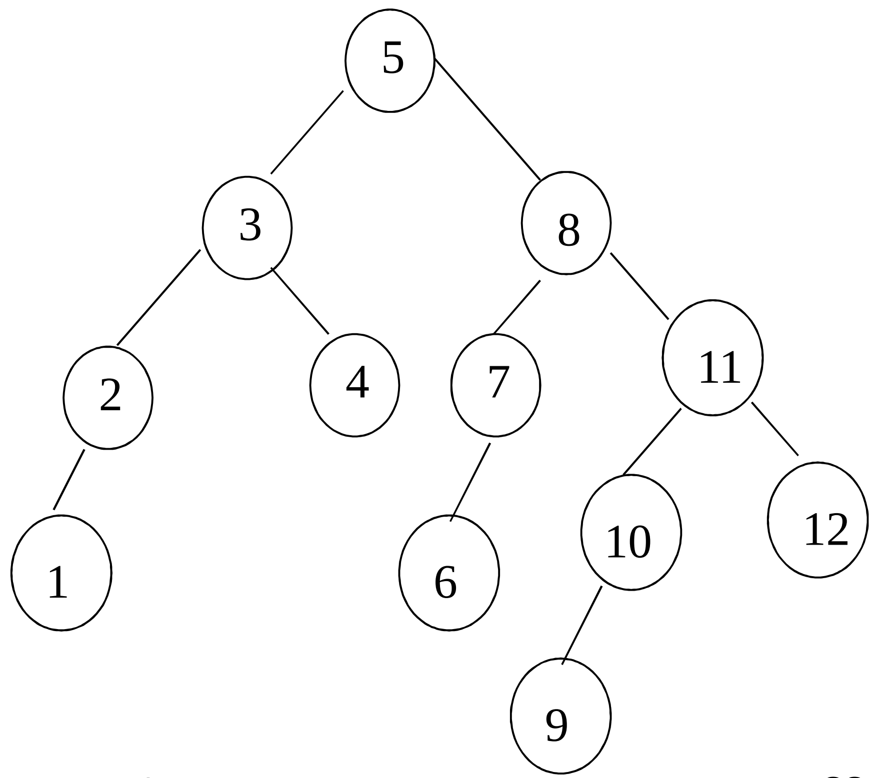
```
private Comparable delete_min ( Avl_nodo t ) {
            // retorna el mínimo del árbol t y lo borra
    if (t != null)
            if (t.izq != null) {
                    // Sigo por la rama izquierda
                    delete_min (t.izq) ;
                    t =balancear (t);
                    t.altura := max (altura (t.der), altura (t.izq) ) + 1; }
            else
                    min = t.dato;
                    t = t.der;
return min;
}
```

Ejercitación

Árbol AVL

ii.- Borrar: 80,60,100 *iii.- Borrar: 4*





Árbol AVL

Tiempo de ejecución de las operaciones

Árbol AVL – Tiempo de ejecución

Suponemos que el árbol AVL está balanceado.

Tomamos N_h como el número de nodos en un árbol AVL de altura h

$$
\begin{aligned}
N_h &\geq N_{h-1} + N_{h-2} + 1 \\
&\geq 2\,N_{h-2} + 1 \\
&\geq 1 + 2(1 + 2\,N_{h-4}) = 1 + 2 + 2^2\,N_{h-4} \\
&\geq 1 + 2 + 2^2 + 2^3\,N_{h-6} \\
&\ \ \ldots \\
&\geq 1 + 2 + 2^2 + 2^3 + \ldots + 2^{h/2} = 2^{h/2+1} - 1
\end{aligned}
$$

continúa …

Árbol AVL – Tiempo de ejecución

Entonces,

$2^{h/2+1} - 1 \leq n$

$h/2 \leq \log_2(n + 1) - 1$

$h \leq 2 \log_2(n + 1) - 2$

Un análisis cuidadoso basado en la teoría de los números de Fibonacci, nos da un valor más ajustado de $1.44 \log_2(n + 2)$.

Final de la clase