Apuntes de Teórico

PROGRAMACIÓN 3

Árboles AVL

Versión 1.0

Índice

Introducción

En el curso de Programación 2, se vio que los Árboles Binarios de Búsqueda son representaciones adecuadas para diccionarios u otros TAD's con las siguientes características:

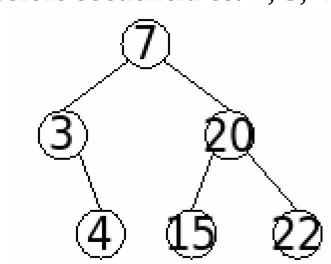
1- Hay definido una relación de orden entre los elementos.
2- Las operaciones a realizar son:
   2.1 - Insert
   2.2 - Delete
   2.3 - Find
   2.4 – Member
   notar que se pueden ver como búsquedas en el TAD. Siendo
      2.1 Una búsqueda no exitosa. (Una búsqueda que siempre falla)
      2.2 – 2.4 búsquedas exitosas.

En general en el curso de Programación 2 se manejó que usando ABB (Árbol binario de Búsqueda), la complejidad de las operaciones anteriores es O(log n), siendo n la cantidad de elementos.

Lo anterior no es válido para todos los casos; De hecho la complejidad de las operaciones depende la altura del ABB, y la altura depende de cual fue la secuencia (el orden) de ingreso a la estructura de los n elementos. Observar que para n elementos hay n! posibles secuencias de ingreso.

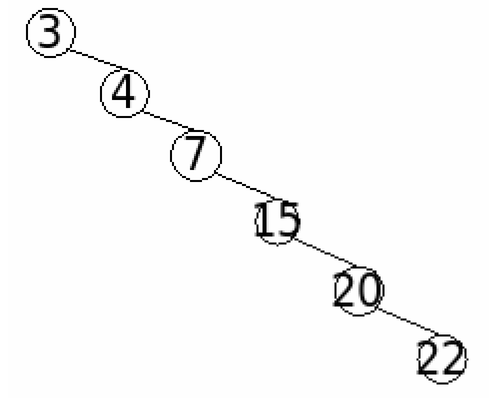
Por ejemplo para los siguientes elementos: 3, 20, 22, 7, 15, 4

Una posible secuencia es: 7, 3, 4, 20, 15, 22, la cual genera el siguiente ABB:

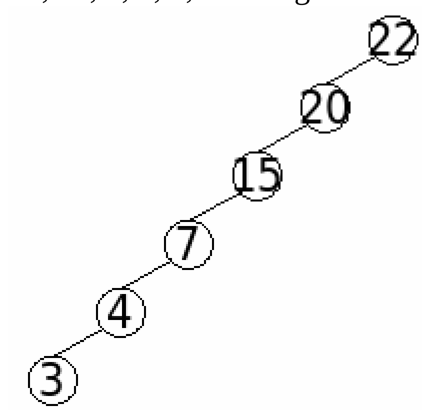


Otras posibles secuencias de ingreso son:

A)   3, 4, 7, 15, 20, 22, la cual general el ABB:



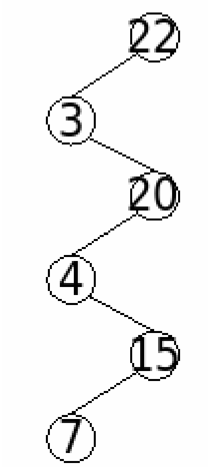
B) 22, 20, 15, 7, 4, 3, la cual general el ABB:



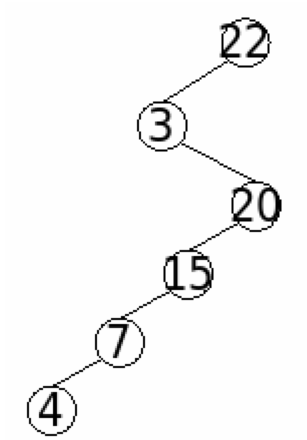
Se observa que las 2 ultimas secuencias de ingreso tienen la particularidad de estar ordenadas de forma creciente y decreciente respectivamente, en cuyo caso el ABB degenera en una lista cuya altura es n, con lo cual las operaciones tienen una complejidad de O(n).

Las secuencias anteriores no son las únicas para las cuales el ABB degenera en una lista, modo de ejemplo y sin ser exhaustivos se pueden mencionar:

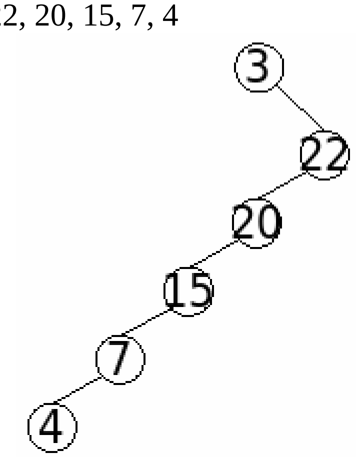
1) 22, 3, 20, 4, 15, 7



2) 22, 3, 20, 15, 7, 4



3) 3, 22, 20, 15, 7, 4



Casos de ordenamientos en la secuencia de ingreso similares a los anteriores plantean la siguiente interrogante: **¿Con que frecuencia se producen estos casos?**

Complejidad Media

Para responder lo anterior, es necesario calcular la complejidad MEDIA de las operaciones (T_A), tal como se vio en Análisis de Algoritmos.

Si se quiere calcular la complejidad media, es necesario determinar cual es la medida a utilizar.

La primera opción es usar la CANTIDAD DE COMPARACIONES como medida. Analizando la conveniencia de utilizar esta medida, se llega a la conclusión que no sirve dado que:
a- Se está tratando de calcular la complejidad media para distintas operaciones a la vez y por lo tanto no se conoce la cantidad de comparaciones.
b- Se estaría calculando la complejidad de manera dependiente de una implementación particular, por ejemplo: en la función Member, podría preguntar:

```
if(x < a->dato)
{
     ...
}
else
     if(x > a->dato)
     {
          …
     }
     else
          if(x == a->dato)
          {
               ...
          }
```

O bien cambiar el orden de las preguntas, lo cual daría una cantidad de comparaciones diferente.

En definitiva, si se quiere calcular la complejidad media de las operaciones sin tener en cuenta las implementaciones particulares de cada una de ellas, se debe buscar otra medida.

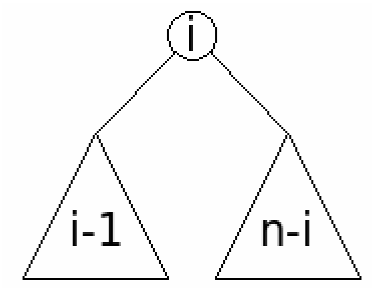
Si se piensa en lo que hacen los algoritmos que implementan las operaciones anteriores se observa que cualquiera de ellos establece un camino desde el nodo raíz hasta un nodo interno o una hoja del ABB. Por lo cual una posible medida es hallar la LONGITUD MEDIA DE CAMINO (cantidad de nodos en el camino promedio) y usarlo para deducir el orden de las operaciones.

En base a lo anterior se desea calcular: **La longitud de camino promedio entre los n elementos y todos los n! árboles que pueden ser generados.**

Para realizar el cálculo anterior se supondrá lo siguiente: sean n elementos distintos con valores 1,2, ...,n. que llegan de manera aleatoria (equiprobable).

En estas condiciones la probabilidad de que el primer elemento que llega tenga el valor i (y sea la raíz del ABB) es 1/n.

En este caso el subárbol izquierdo tendrá **i-1** nodos y el derecho tendrá **n – i** nodos.



a_{i-1} longitud de camino medio en el sub. izq.

a_{n-i} longitud de camino medio en el sub. der.

La longitud de camino media en un árbol de n nodos es la suma de los productos de la longitud de camino de cada nodo por la probabilidad de acceso al mismo.
Suponiendo que la probabilidad de acceso para cada nodo es la misma (equiprobable):

$$a_n = \sum_{j=i}^{n} P_j C_j = \frac{1}{n}\sum_{j=1}^{n} C_j$$

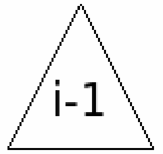
donde:

P_j es equiprobable => $P_j = \frac{1}{n} \forall j$

C_j longitud de camino para el j-esimo nodo.

Descomponiendo a_n para el árbol anterior**:**

$$a_n = \frac{1}{n}\left(\sum_{j=1}^{i-1} C_j + 1 + \sum_{j=i+1}^{n} C_j\right)$$

En el subárbol 

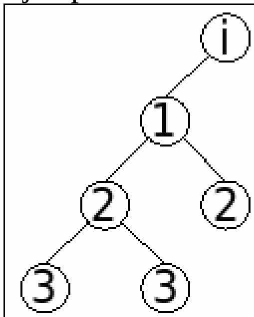
aplicando la misma idea, se tiene:

$$a_{i-1} = \frac{1}{i-1}\sum_{j=1}^{i-1} C'_j$$

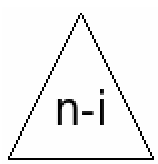
Pero $C'_j = C_j - 1$ dado que la longitud de camino de los nodos del subárbol es menor en una unidad a la longitud de camino, de los mismos nodos, que en el árbol de raíz *i*.

$$\Rightarrow a_{i-1} = \frac{1}{i-1}\sum_{j=1}^{i-1}(C_j - 1) = (\frac{1}{i-1}\sum_{j=1}^{i-1} C_j) - 1 \Rightarrow \sum_{j=1}^{i-1} C_j = (a_{i-1}+1)(i-1)$$

Ejemplo:



- $a_{i-1}=\frac{1}{i-1}\sum_{j=1}^{i-1}C'_j=\frac{1}{5}(1+2+2+3+3)=\frac{11}{5}$
- $\sum_{j=1}^{n}C_j=2+3+3+4+4=16$
- $16=(a_{i-1}+1)(i-1)=(\frac{11}{5}+1)5$

En el subárbol 

se puede hacer lo mismo, por lo tanto:

$$\sum_{j=i+1}^{n} C_j = (a_{n-i}+1)(n-1)$$

Sustituyendo en a_n : $a_n = (a_{i-1}+1)\frac{i-1}{n} + \frac{1}{n} + (a_{n-i}+1)\frac{n-i}{n}$

La expresión anterior es la longitud media de camino para el árbol que tiene al elemento i en la raíz o sea en realidad es $a_n^{(i)}$.

Para calcular a_n es necesario promediar $a_n^{(i)}$ para todos los árboles con elementos 1, .., n en la raíz.

Suponiendo equiprobabilidad,

$$a_n = \frac{1}{n}\sum_{i=1}^{n}((a_{i-1}+1)\frac{i-1}{n} + \frac{1}{n} + (a_{n-i}+1)\frac{n-i}{n})$$

Resolviendo la expresión anterior se llega a

$$a_n = 2L(n) = 2L(2)\log_2(n) \cong 1.4\log_2(n) \ \ (\log_2(n) = \frac{L(n)}{L(2)})$$

=> se puede deducir que a_n es O($\log_2(n)$) => $T_A \in$ O($\log(n)$)

lo cual implica que las operaciones INSERT, DELETE, MEMBER y FIND son O($\log(n)$) en el caso medio.

A pesar de esto en el peor caso, las operaciones son O(n), por lo cual es importante diseñar algoritmos que generen árboles cuya altura máxima sea O($\log(n)$) (independientemente de cual fue la secuencia de ingreso)

Los árboles cuya altura máxima es O($\log(n)$) se llaman EQUILIBRADOS o BALANCEADOS.

Árboles perfectamente equilibrados

Basan el equilibrio en la cantidad de nodos. Exigiendo que en todo momento que: $|cant\_nodos(izq) - cant\_nodos(der)| \leq 1$

Tiene la virtud de que la altura es $\log(n)$ siempre y la desventaja que los algoritmos son sumamente complejos ya que frecuentemente deben reorganizar el Árbol.

Árboles AVL

Un criterio de equilibrio no tan estricto como el anterior, fue propuesto por Adelson, Velsky y Landis en 1962, los cuales introdujeron el árbol equilibrado en altura o AVL (por la primera letra del apellido de cada uno de ellos).

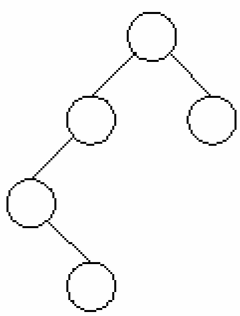
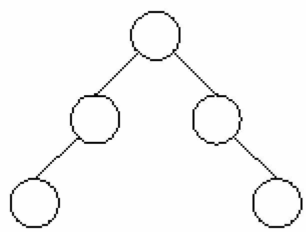
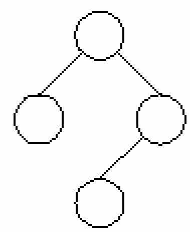
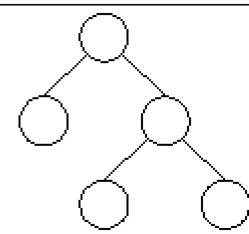
Definición:

a) Un árbol vacío es un árbol AVL

b) si T es un árbol no vacío y T_L y T_R son sus subárboles, T es un AVL <=>

1) T_L es un AVL
2) T_R es un AVL
3) $|Altura(T_L) - Altura(T_R)| \leq 1$

Ejemplos:

			
NO es AVL	AVL	AVL	AVL. Notar que no es perfectamente equilibrado

La diferencia de altura de los subárboles se denomina FACTOR DE BALANCEO (FB) y para todo nodo de un AVL se cumple que FB = 1, 0, -1

en este curso se usará la siguiente convención:

- FB = 1 si $ALTURA(T_L) > ALTURA(T_R)$
- FB = -1 si $ALTURA(T_L) < ALTURA(T_R)$
- FB = 0 si $ALTURA(T_L) = ALTURA(T_R)$

Adelson, Velsky y Landis demostraron el siguiente teorema, que indica cual es la altura máxima y mínima de un AVL:

Teorema:

Sea *h* la altura de un árbol AVL con n nodos
$\Rightarrow \log(n+1) \leq h \leq 1.44\log(n+2) - 0.328$

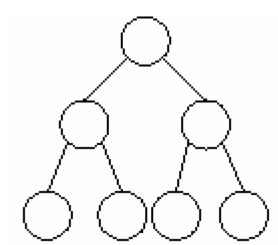
Demostración:

a) $\log(n-1) \leq h$

Esta desigualdad es válida para cualquier árbol binario de altura *h*, el cual tiene como máximo $\sum_{i=1}^{h-1} 2^i$ nodos

O sea, $n \leq \sum_{i=1}^{h-1} 2^i = 2^h - 1 \Rightarrow n+1 \leq 2^h \Rightarrow \log(n+1) < h$

Ejemplo, si se considera el siguiente árbol completo con altura *h*=3:



la cantidad de nodos es $\sum_{i=0}^{2} 2^i = 2^0 + 2^1 + 2^2 = 1+2+4 = 7$

b) $h \leq 1.4404\log(n+2) - 0.328$

Sea T_h un árbol AVL de altura h con mínima cantidad de nodos. Por ser T_h un AVL se cumple que uno de sus subárboles tiene altura *h-1* y el otro subárbol tiene altura *h-1* ó *h-2*.

Como T_h es de mínima cantidad de nodos, un subárbol tiene altura *h-1* y el otro tendrá altura *h-2*.

Sea N_h la cantidad de nodos del árbol de altura $h \Rightarrow N_h = N_{h-1} + N_{h-2} + 1$, además $N_1 = 1, N_0 = 0$

Es posible observar la similitud con los números de Fibonacci:
$F_n = F_{n-1} + F_{n-2}$ con $F_1 = 1; F_0 = 0$.

Se demostrará por inducción completa en h que $N_h = F_{h+2} - 1$.

Paso base:

$h = 0$: $N_0 = 0$ $F_{h+2} = F_2 = F_1 + F_0 - 1 = 0$
$h = 1$: $N_1 = 1$ $F_{h+2} - 1 = F_3 - 1 = F_2 + F_1 - 1 = 1$

Paso inductivo:

HI) Se cumple para $h \leq k$ que $N_h = F_{h+2} - 1$

TI) Se cumple para $h = k + 1$ que $N_h = F_{h+2} - 1$

$N_{k+1} = N_k + N_{k-1} + 1 = (F_{k+2} - 1) + (F_{k+1} - 1) + 1 = F_{k+2} + F_{k+1} - 1 = F_{k+3} - 1$ LQQD.

Como T_h es un AVL con mínima cantidad de nodos => la cantidad de nodos *n* de cualquier otro AVL de altura *h* es *n* >= N_h

O sea: $n \geq N_h \Rightarrow n \geq F_{n+2} - 1$

Por la teoría de números de Fibonacci:

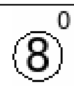
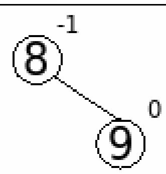
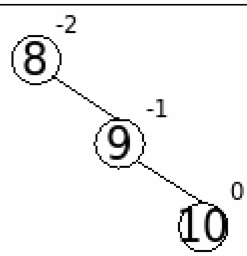
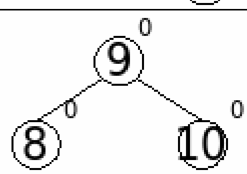
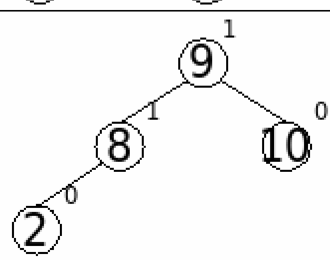
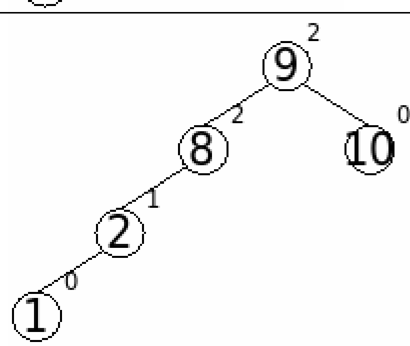
$$F_n = \frac{\phi^h - 1}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2} \Rightarrow \frac{\phi^{h+2} - 1}{\sqrt{5}} - 1 = \frac{(\frac{1+\sqrt{5}}{2})^{h+2} - 1}{\sqrt{5}} - 1$$

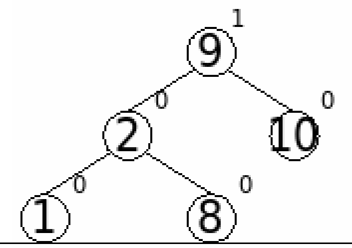
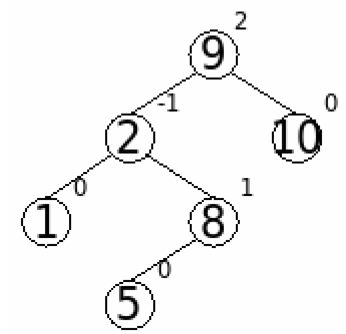
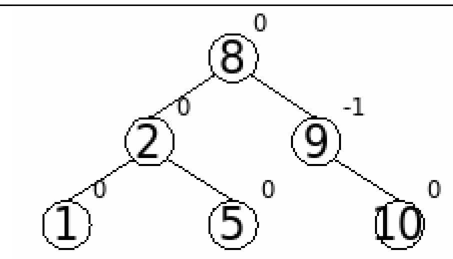
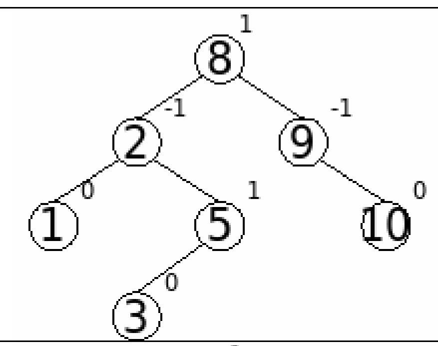
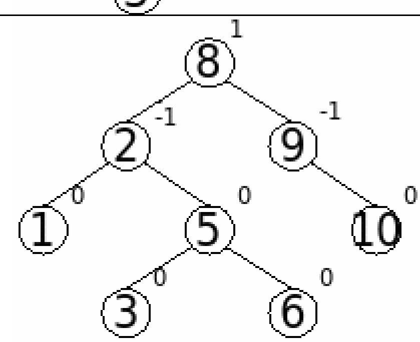
y despejando $h \Rightarrow h \leq 1.404 \log(n+2) - 0.328$

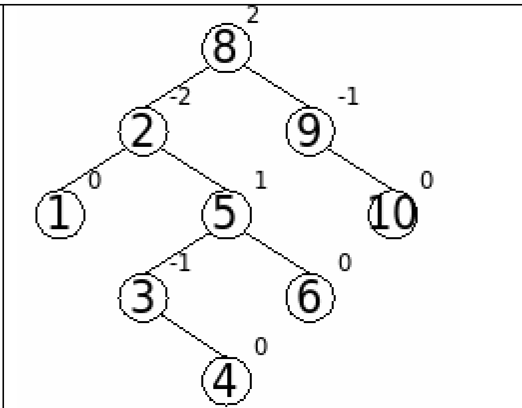
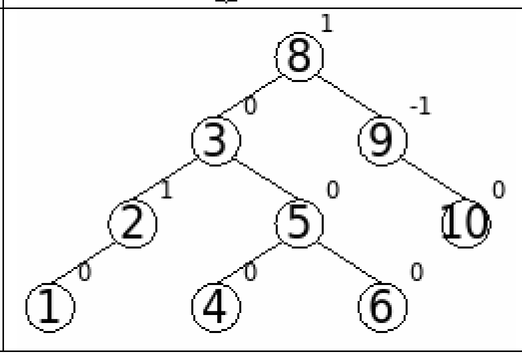
Inserción en AVL

Los algoritmos mas interesantes sobre árboles AVL son los algoritmos de inserción y borrado, que son los encargados de mantener el equilibrio, rebalanceando el árbol en caso de ser necesario. En este teórico es estudiará en detalle la inserción.

Se verá primero como es el proceso de inserción mediante un ejemplo, usando la siguiente secuencia de ingreso: 8, 9, 10, 2, 1, 5, 3, 6, 4

Inserción 8		
Inserción 9		
Inserción 10		En este caso se debe rebalancear. Observar que la altura previa a la inserción era h=2 y luego de la inserción es h=3.
Rebalanceo		El rebalanceo la vuelve la altura a h=2, tal como era previo la inserción.
Inserción 2		
Inserción 1		Es necesario rebalancear. La pregunta es cual nodo se debe rebalancear, el 8 o el 9? Teniendo en cuenta lo dicho anteriormente sobre que el rebalanceo vuelve la altura al valor previo a la inserción, se observa que en el subárbol de raíz 8 la altura era h=2. Luego de la inserción se incrementó a

		h=3, si se rebalancea la altura vuelve a ser h=2.
Rebalanceo		Si se rebalancea el subárbol de raíz 8 queda rebalanceado también el subárbol de raíz 9 (porque el subárbol izquierdo de raíz 8 vuelve a tener altura h=2 y el derecho tiene h=1)
Inserción 5		Observar que se debe rebalancear. La idea es la misma que en los casos anteriores: volver a tener un árbol con la misma altura que previo a la inserción.
Rebalanceo		
Inserción 3		
Inserción 6		

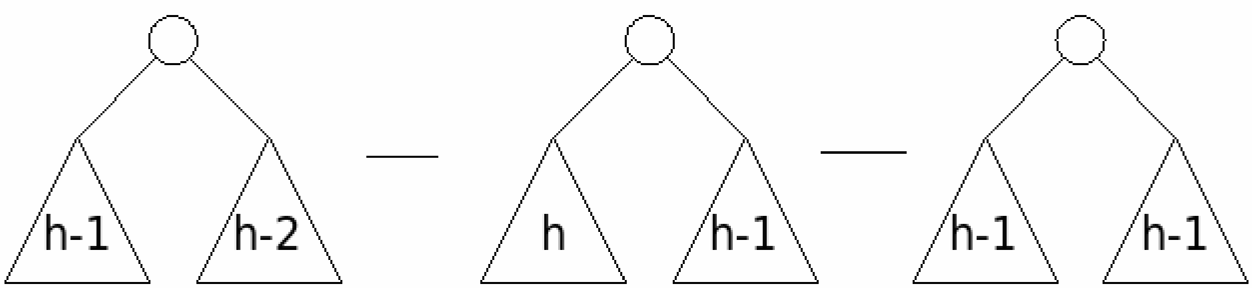
Inserción 4		Es necesario rebalancear. Aplicando la misma idea que cuando se insertó el 1, si se rebalancea el subárbol que tiene raíz 2 no es necesario seguir rebalanceando hacia arriba porque se vuelve a disminuir la altura del subárbol.
Rebalanceo		

Conclusiones sobre la restauración del equilibrio

En base al ejemplo anterior pueden extraerse las siguientes conclusiones:

- El rebalanceo se realiza en el subárbol cuya raíz sea el ancestro mas cercano al nodo insertado y cuyo factor de balance se transforme, a consecuencia de la inserción, en 2 o -2.

- Luego de rebalanceado el subárbol anterior, no es necesario rebalancear nada más, ya que si la altura de este subárbol antes de la inserción era *h* y luego de la inserción la altura se incrementa a *h+1*, luego del rebalanceo vuelve a ser *h*. Como se observa en la siguiente figura:



.

Inserción de un nodo

Sea una raíz con subárbol izquierdo T_L y subárbol derecho T_R.

1) El nodo a insertar se inserta en T_L haciendo que su altura se incremente (si no se incrementa, no se produce desbalanceo). Pueden suceder los siguientes casos:

Previo a la inserción	**Luego de la inserción**
ALTURA(T_L) = ALTURA(T_R), FB = 0	FB = 1. Puede vulnerarse el equilibrio en un nodo ancestro.
ALTURA(T_L) < ALTURA(T_R), FB = -1	FB = 0 no se debe rebalancear.
ALTURA(T_L) > ALTURA(T_R), FB = 1	Se debe rebalancear

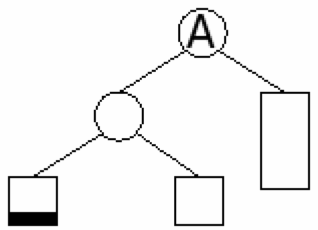
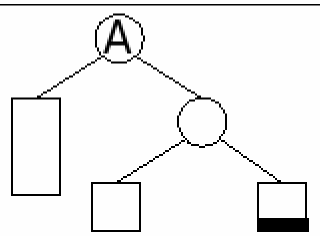
Supongamos que el nodo a insertar se inserta en T_R haciendo que su altura se incremente (si no se incrementa, no se produce desbalanceo). Pueden suceder los siguientes casos:

Previo a la inserción	**Luego de la inserción**
ALTURA(T_R) = ALTURA(T_L), FB = 0	FB = -1. Puede vulnerarse el equilibrio en un nodo ancestro.
ALTURA(T_R) < ALTURA(T_L), FB = 1	FB = 0 no se debe rebalancear.
ALTURA(T_R) > ALTURA(T_L), FB = -1	Se debe rebalancear

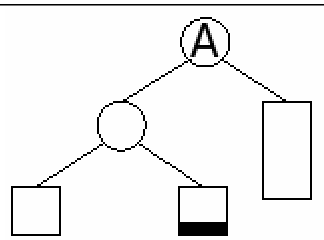
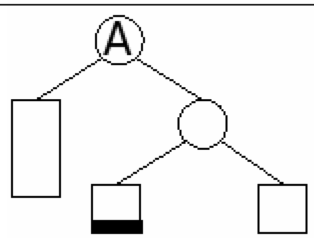
Tipos de rebalanceos o rotaciones

Sea A el ancestro más cercano del nodo insertado, cuyo FB luego de la inserción es 2 ó -2. Los rebalanceos se clasifican en:

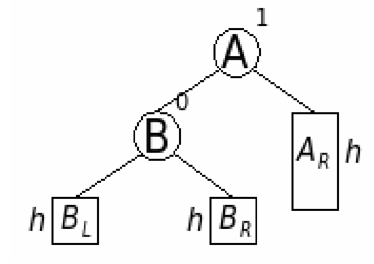
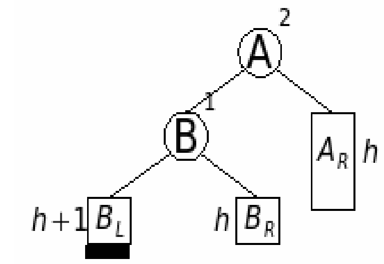
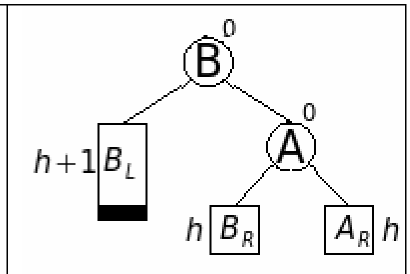
Rotaciones Simples: Involucran 3 subárboles

LL		El nodo fue insertado en el subárbol izquierdo del subárbol izquierdo de A.
RR		El nodo fue insertado en el subárbol derecho del subárbol derecho de A.

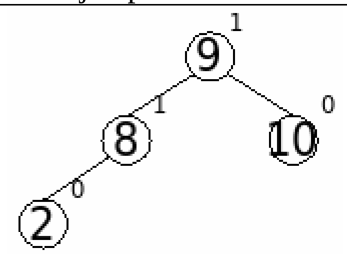
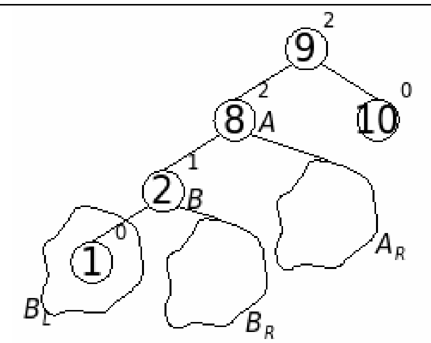
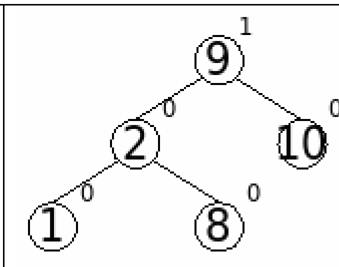
Rotaciones Dobles: Involucran 4 subárboles. Pueden resolverse en función de 2 rotaciones simples.

LR		El nodo fue insertado en el subárbol derecho del subárbol izquierdo de A
RL		El nodo fue insertado en el subárbol izquierdo del subárbol derecho de A.

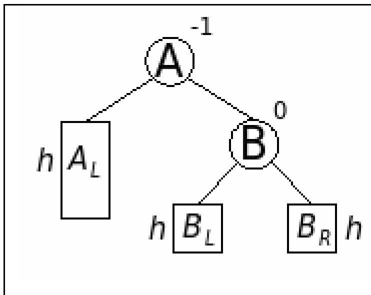
Tipo LL

<table>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción</td>
<td>Rebalanceo</td>
</tr>
</table>

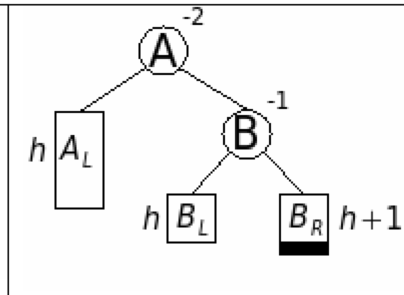
en el ejemplo:

<table>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción del 1</td>
<td>Rebalanceo</td>
</tr>
</table>

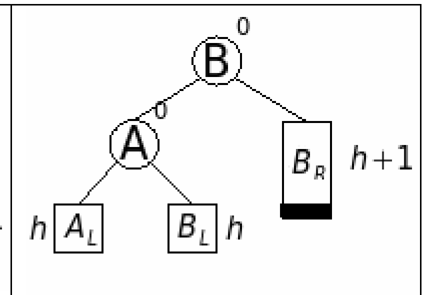
Tipo RR



Previo a la inserción

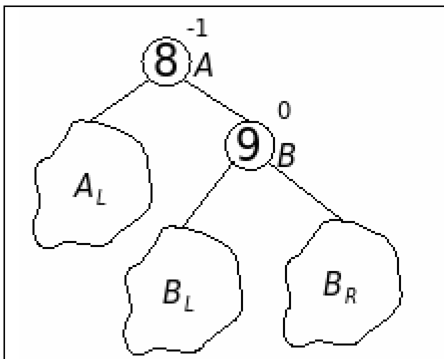


Luego de la inserción

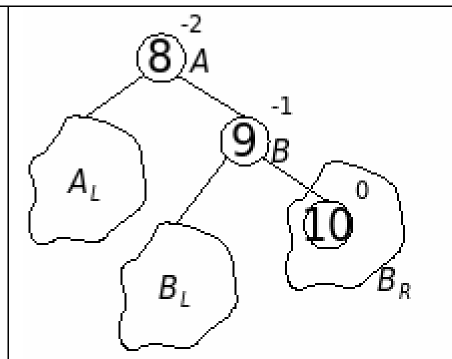


Rebalanceo

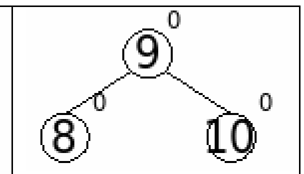
en el ejemplo:



Previo a la inserción



Luego de la inserción del 10

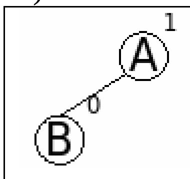
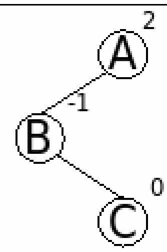
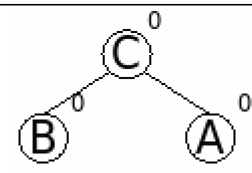


Rebalanceo

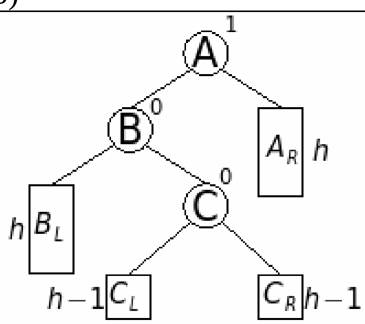
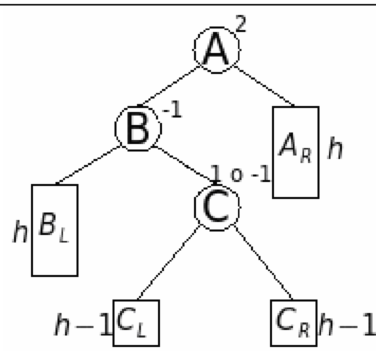
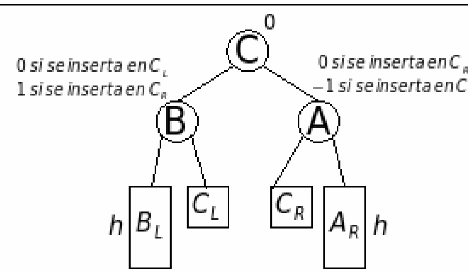
Tipo LR

Se presenta esta rotación en dos casos, aunque en realidad se trata del mismo, a los efectos de ver primeramente el caso sencillo y después el caso completo.

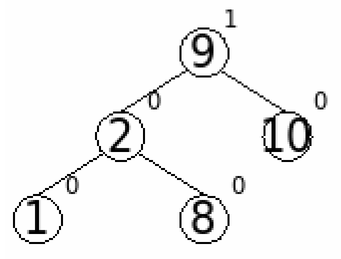
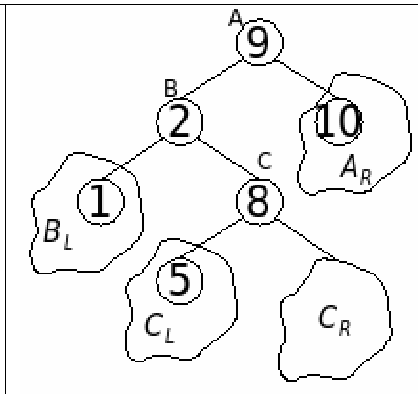
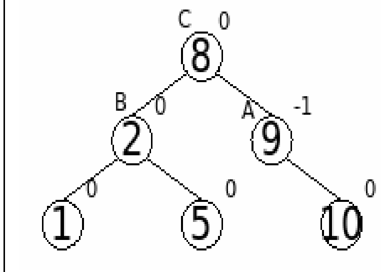
a)

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Previo a la inserción</td><td>Luego de la inserción</td><td>Rebalanceo</td></tr>
</table>

b)

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>Previo a la inserción</td><td>Luego de la inserción</td><td>Rebalanceo</td></tr>
<tr><td></td><td>En el nodo puede ser insertado en C_L o C_R. Si va a C_L el FB de C pasa a ser 1, sino va a C_R el FB de C pasa a ser –1. En cualquiera de los 2 casos, el FB de B pasa a ser -1 y el de A pasa a ser 2.</td><td></td></tr>
</table>

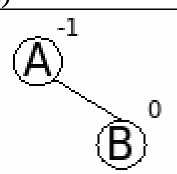
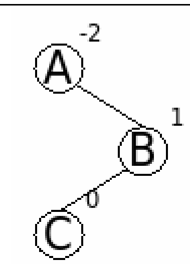
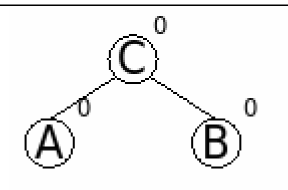
en el ejemplo:

<table>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción del 5</td>
<td>Rebalanceo</td>
</tr>
</table>

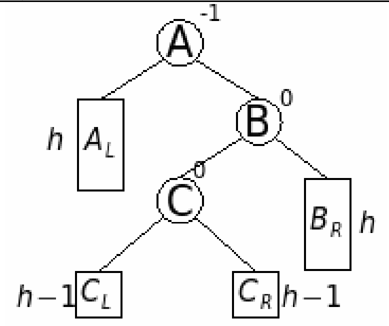
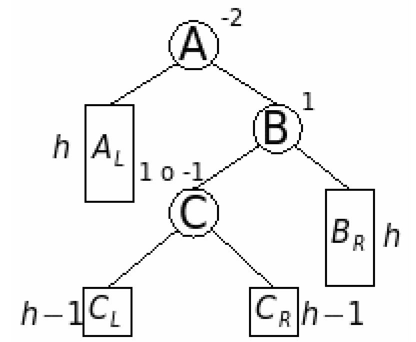
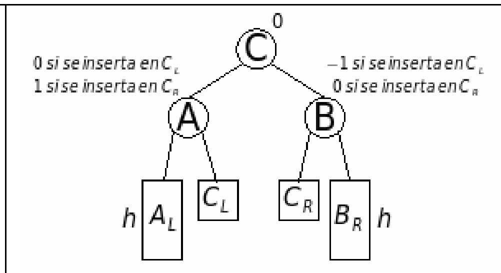
Tipo RL

Se presenta esta rotación en dos casos, aunque en realidad se trata del mismo, a los efectos de ver primeramente el caso sencillo y después el caso completo.

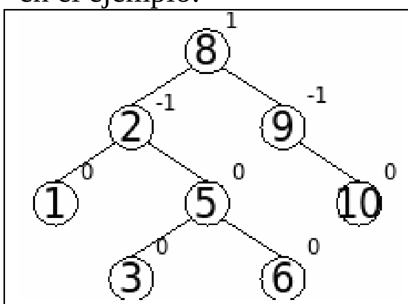
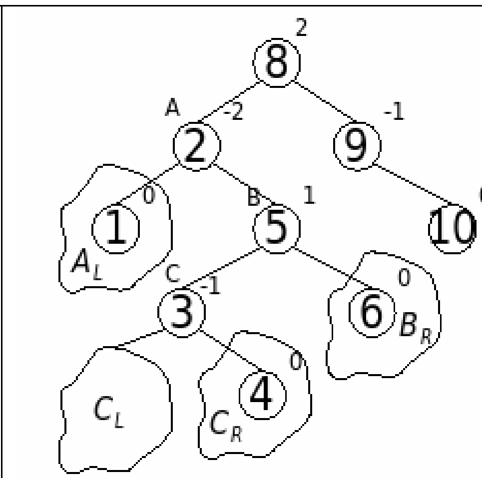
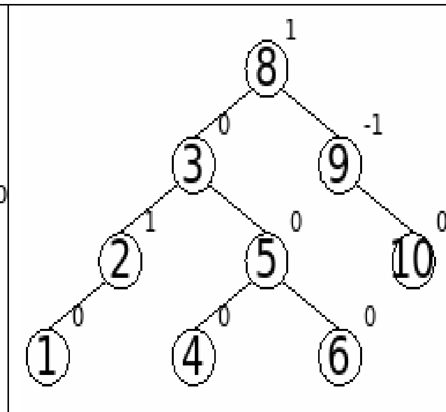
a)

<table>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción</td>
<td>Rebalanceo</td>
</tr>
</table>

b)

<table>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción</td>
<td>Rebalanceo</td>
</tr>
<tr>
<td></td>
<td>El nodo a insertar puede ir a C_L o C_R, si va a C_L el FB de C pasa a ser 1, si va a C_R el FB de C pasa a ser -1. En cualquiera de los 2 casos, el FB de B pasa a ser 1 y el de A pasa a ser -2</td>
<td></td>
</tr>
</table>

en el ejemplo:

<table>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>Previo a la inserción</td>
<td>Luego de la inserción del 4</td>
<td>Rebalanceo</td>
</tr>
</table>

Como se observa en los esquemas de los tipos de rebalanceo, en todos los casos se trata de operaciones que solamente involucran intercambios de punteros, por lo cual los algoritmos que implementan los distintos tipos de rebalanceos serán de O(1).

Para poder saber que rebalanceo aplicar es necesario conocer los factores de balance de los nodos involucrados. Estos factores de balance son calculables a partir de las alturas de los subárboles, sin embargo resulta sumamente ineficiente calcularlos cada vez que sea necesario. Es mejor almacenar dicha información en el propio nodo del árbol. Esta información deberá ser actualizada por los algoritmos de inserción y borrado.

```
struct nodeAVL
{
     Key clave;
     int FB;
     struct nodeAVL * izq;
     struct nodeAVL * der;
}*AVL;
```

Algoritmo de inserción

Se divide en 2 etapas:

a) Insertar el nuevo nodo en el lugar correspondiente
b) recorrer el camino realizado para insertar el nodo en “vuelta atrás”, chequeando los factores de balanceo.

Para saber si se produjo desbalanceo se necesita, además de los factores de balanceo, saber si se incrementó o no la altura del subárbol donde se insertó el nodo, para lo cual se utilizará un parámetro booleano.

```
Insert(Key X, boolean & aumento, AVL &a)
{
    if (Vacio(a))
    {
        a = crearNodo(X); //crea un nodo con FB = 0
        aumento = true;
    }
    else
        if (X < a->clave)
        {
            insert(X, aumento, a->izq);
            if (aumento)
            {
                switch(a->FB)
                {
                    case -1: //antes de la ins. ALT(TL) < ALT(TR)
                            a->FB = 0;//No se produce desbalanceo
                            aumento = false;
                            break;
                    case 0: //antes de la ins. ALT(TL) = ALT(TR)
                            a->FB=1;//mirar los ancestros
                            break;
                    case 1: //antes de la ins. ALT(TL) > ALT(TR)
                              //rebalanceo, el tipo es LL o LR
                            if (a->izq->FB == 1) // es LL
                                a = RebalancearLL(a);
                            else //es LR
                                a = RebalancearLR(a);
                            aumento = false;
                            break;
                }
            }
        }
```

```
        else
        {
            insert(X, aumento, a->der);
            if(aumento)
            {
                switch(a->FB)
                {
                    case 1: //antes de la ins. ALT(TR) < ALT(TL)
                            a->FB = 0;//No se produce desbalanceo
                            aumento = false;
                            break;
                    case 0: //antes de la ins. ALT(TR) = ALT(TL)
                            a->FB = -1; // mirar los ancestros
                            break;
                    case -1: //antes de la ins. ALT(TR) > ALT(TL)
                            //rebalanceo, el tipo es RR o RL
                            if (a->der->FB == -1) // es RR
                                a = RebalancearRR(a);
                            else //es RL
                                a = RebalancearRL(a);
                            aumento = false;
                            break;
                }
            }
        }
     }
}
```