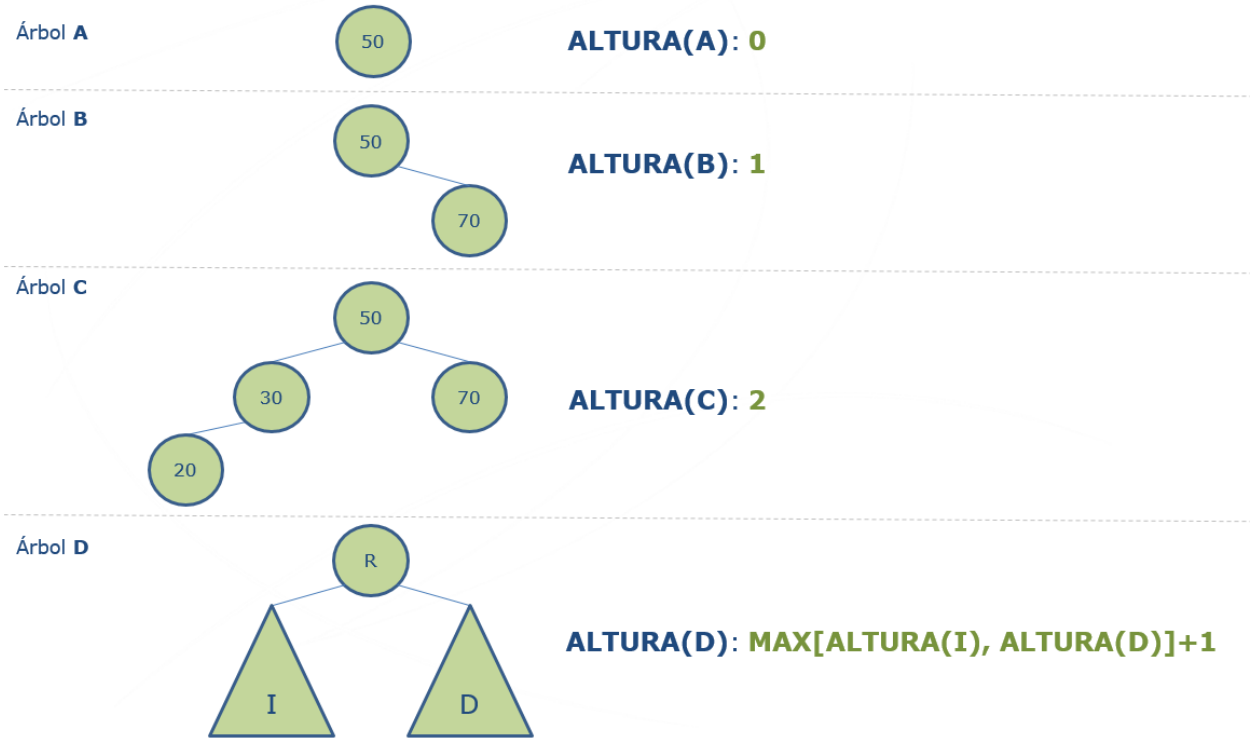
Apuntes

Tema ***Arboles Binarios de Búsqueda***

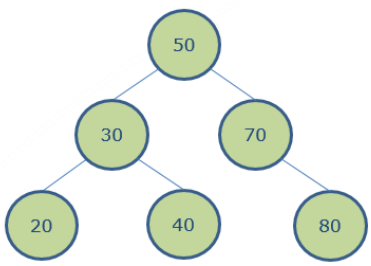
"La ciencia, muchacho, está hecha de errores, pero de errores útiles de cometer, pues poco a poco, conducen a la verdad."
Julio Verne

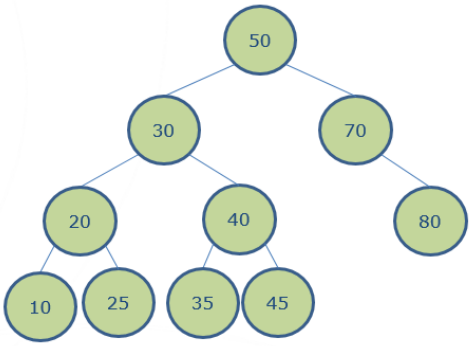
Arboles auto-balanceados

Recordemos conceptos de altura



Criterios de equilibrio (o de balanceo)





El número de nodos del subárbol izquierdo difiere como máximo en 1 con el número de nodos del subárbol derecho	La altura del subárbol izquierdo difiere como máximo en 1 con la altura del subárbol derecho
Árbol Perfectamente Equilibrado	Árbol Equilibrado

Arboles AVL

Un árbol AVL es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-**Velskii** y **Landis**. Fue el primer árbol de búsqueda binario auto balanceable que se ideó.
Lo dieron a conocer en la publicación de un artículo en 1962, "**An algorithm for the organization of information**" (Un algoritmo para la organización de la información).
Suponiendo que deseamos construir un **ABB** para un grupo de datos planteados en cierto orden, tenemos como resultado un árbol muy poco balanceado y con características muy pobres para la búsqueda. Los ABB trabajan muy bien para una amplia variedad de aplicaciones, pero tienen el problema de que la eficiencia en el peor caso es **O(n)**. Los árboles que estudiaremos a continuación nos darán una idea de cómo podría resolverse el problema garantizando en el peor caso un tiempo **O($\log_2$ n)**.


Adelson-Velskii


Yevgueni Landis

Diremos que un árbol binario está equilibrado (en el sentido de Addelson-Velskii y Landis) de tal modo que para todos los nodos, **la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa**.
Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad **O(log n)**. El ***factor de equilibrio*** puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.
Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se *rompe la condición de equilibrio*, hay que realizar una serie de **rotaciones** de los nodos.

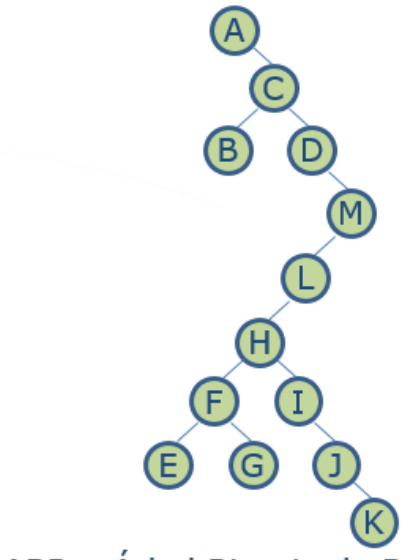
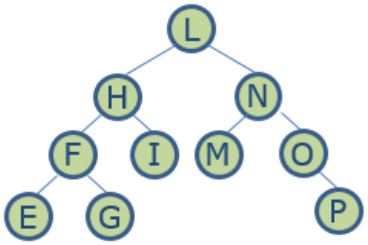


ABB – Árbol Binario de Búsqueda

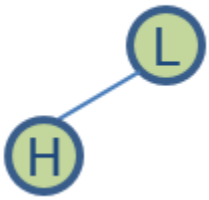


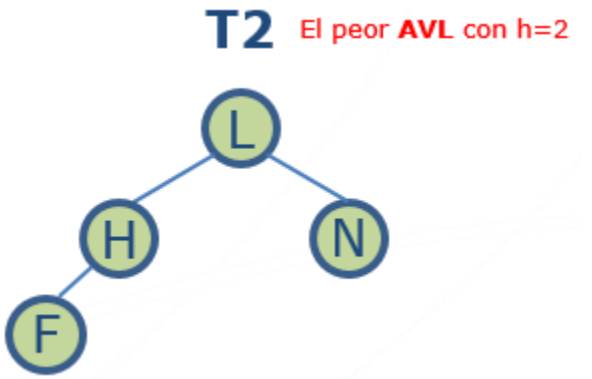
Árbol AVL

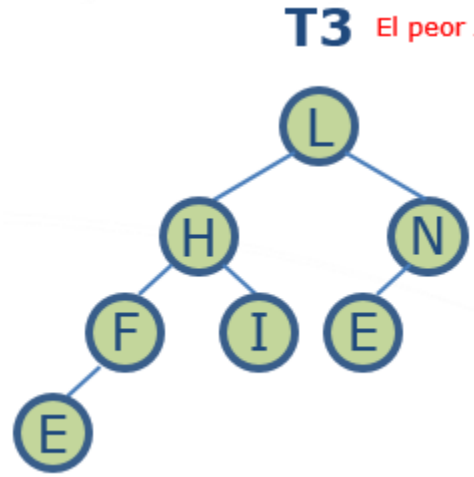
A través de los árboles AVL llegaremos a un procedimiento de búsqueda análogo al de los ABB pero con la ventaja de que garantizaremos un caso peor de **$O(\log_2 n)$**, manteniendo el árbol en todo momento equilibrado.

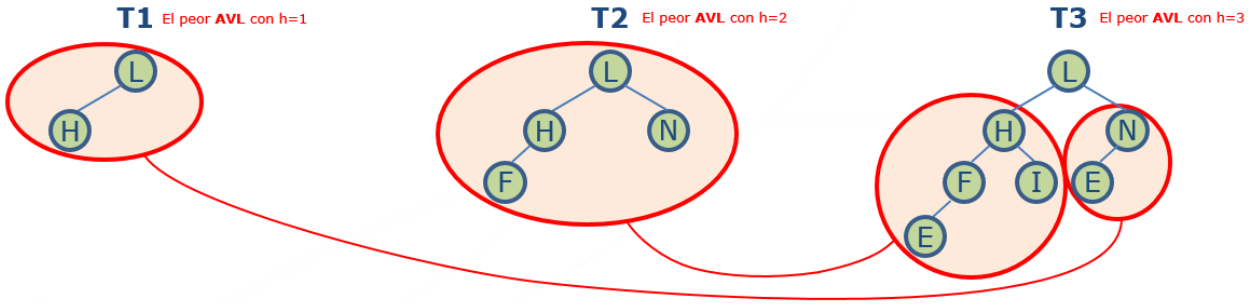
Para llegar a este resultado, podríamos preguntarnos cuál podría ser el peor **AVL** que podríamos construir con **n** nodos, o dicho de otra forma cuanto podríamos permitir que un árbol binario se desequilibrara manteniendo la propiedad de **AVL**.
Para responder a la pregunta podemos construir para una altura **h** el **AVL T_h**, con mínimo número de nodos. Cada uno de estos árboles mínimos debe constar de una raíz, un subárbol **AVL** mínimo de altura **h-1** y otro subárbol AVL también mínimo de altura **h-2**. Los primeros **Ti** pueden verse en la siguiente figura

T1 El peor **AVL** con h=1









Es fácil ver que el número de nodos $n(T_h)$ está dado por la siguiente relación de recurrencia:

$n(T_h) = 1 + n(T_{h-1}) + n(T_{h-2})$

$$n(T_h) = 1 + n(T_{h-1}) + n(T_{h-2}) \cong F_{i-1} + F_{i-2} = F_i$$

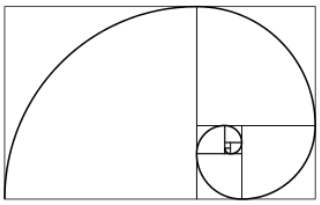
Demostración de complejidad de Arboles AVL

Partiendo de

$$n(T_h) = 1 + n(T_{h-1}) + n(T_{h-2})$$

o

$$N_i = 1 + N_{i-2} + N_{i-1}$$



Podemos establecer una relación **similar** a la que aparece en los números de **Fibonacci**

$$F_n = F_{n-1} + F_{n-2}$$

de forma que la secuencia de valores para $n(T_h)$ está relacionada con los valores de la secuencia de Fibonacci:

AVL -> -, -, 1, 2, 4, 7, 12, ...
FIB -> 1, 1, 2, 3, 5, 8, 13, ...

Partimos entonces de que para Fibonacci establecemos que:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-2} + F_{n-1}$$

Y que

$$\frac{F_n}{F_{n-1}} \cong \Phi = \frac{1+\sqrt{5}}{2}$$

*Φ es llamado número de áureo o **proporción** aurea el que aproximadamente es ~1,618033988*

Despejando tenemos que

$$F_n = \frac{\Phi^n}{\sqrt{5}}$$

Pudiéndose usar para obtener cualquier número de la serie

En Fibonacci también podemos establecer que la suma de los n primeros términos está dado por:

$$N_n = \sum_{i=1}^{n} F_i = F_{n+2} - 1$$

Es igual al termino n+2 menos 1

Entonces podemos decir que

$$N_i \geq F_{i+2} - 1$$

$$N_i \geq \frac{\Phi^{i+2}}{\sqrt{5}} - 1$$

Reemplazando F_{i+2} por la formula general F_n

$$\log(N_i + 1) \geq (i + 2)\log\Phi - \frac{1}{2}\log 5$$

Haciendo pasaje de términos en la desigualdad y tomando logaritmos en ambos términos.

$$i \leq \frac{\log(N_i + 1) - 2\log\Phi + \frac{1}{2}\log 5}{log\Phi}$$

Luego despejando i

Y asignando a los valores contantes los nombres c1 y c2

$$i \leq 1{,}44\log(N + c1) + c2$$

Llegamos a la siguiente formula

$$\mathbf{h=O(1,44\log n)}$$

o dicho de otra forma, la longitud de los caminos de búsqueda (o la altura) para un AVL de **n** nodos, **nunca excede al 44% de la longitud de los caminos (o la altura)** de un árbol completamente equilibrado con esos **n** nodos. En consecuencia, aún en el peor de los casos llevaría un tiempo **O(log$_2$ n)** al encontrar un nodo con una clave dada.

Parece, pues, que el único problema es el mantener siempre tras cada inserción la condición de equilibrio, pero esto puede hacerse muy fácilmente sin más que hacer algunos reajustes locales, cambiando punteros *(o datos referenciales).*

Definición de Árbol AVL

Para poder definirlo, primero debemos definir la altura H de un árbol binario

* Sea T un árbol binario de búsqueda y sean T_i y T_d sus subárboles, su altura $\mathrm{H}(T)$, es:
  - **0** si el árbol T contiene solo la raíz
  - $\mathbf{1 + \max(H(T_i), H(T_d))}$ si contiene mas nodos

Definición de AVL:

* Un árbol vacío es un árbol AVL
* Si T es un árbol no vacío y T_i y T_d sus subárboles, entonces T es AVL si y solo si:
  - T_i es AVL
  - T_d es AVL
  - $|\boldsymbol{H(T_i) - H(T_d)}| \leq \mathbf{1}$

Factor de equilibrio

Cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecho e izquierdo, igual que los árboles binarios de búsqueda (ABB), y además el dato que controla el factor de equilibrio. El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo: **FE = altura subárbol izquierdo- altura subárbol derecho;**

Por definición, para un árbol AVL, este valor debe ser **-1**, **0** ó **1**.
Si el factor de equilibrio de un nodo es:

0 -> el nodo está equilibrado y sus subárboles tienen exactamente la misma altura.
1 -> el nodo está equilibrado y su subárbol izquierdo es un nivel más alto.
-1 -> el nodo está equilibrado y su subárbol derecho es un nivel más alto.

Si el factor de equilibrio **|Fe| >= 2** es necesario reequilibrar.

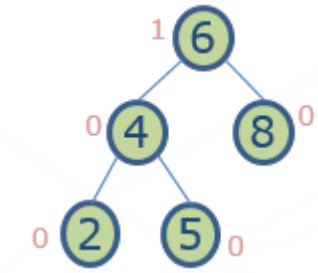
Rotaciones

El reequilibrado se produce de abajo hacia arriba sobre los nodos en los que se produce el desequilibrio. Pueden darse dos casos: **rotación simple** o **rotación doble**; a su vez ambos casos pueden ser hacia la derecha o hacia la izquierda.

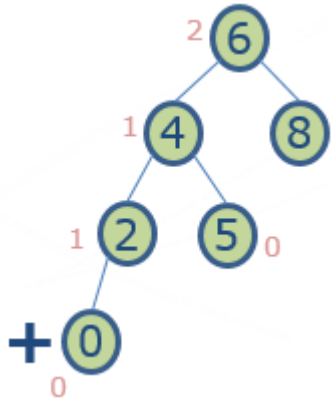
Rotación simple a derecha

De un árbol de raíz (r) y de hijos izquierdo (i) y derecho (d), lo que haremos será formar un nuevo árbol cuya raíz sea la raíz del hijo izquierdo, como hijo izquierdo colocamos el hijo izquierdo de i (nuestro i') y **como hijo derecho construimos un nuevo árbol que tendrá como raíz, la raíz del árbol (r)**, el hijo derecho de i (d') será el hijo izquierdo y el hijo derecho será el hijo derecho del árbol (d).

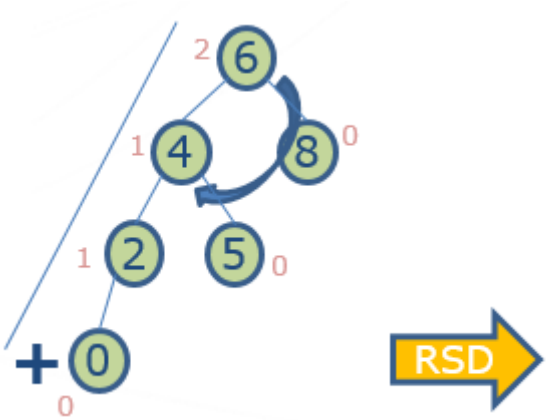
Ej: Partiendo de un árbol AVL agregamos un nodo y nos queda un árbol desequilibrado

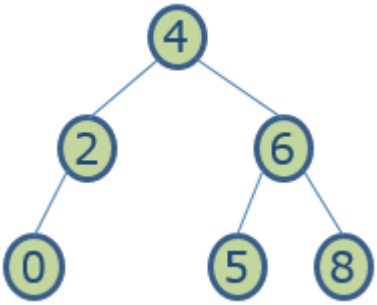


Árbol avl



Árbol desbalanceado





Árbol balanceado

Viéndolo en un ejemplo el algoritmo a aplicar es el siguiente:
Tengamos siempre en cuenta que ***Fe=NivSubArbIzquierdo-NivSubArbDerecho***

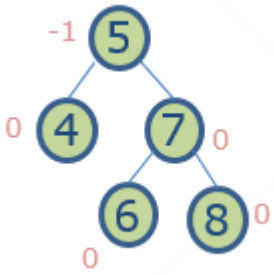
P: Es el nodo con Fe>1 y PadreP el nodo padre de P
Q=P.izquierda
A=Q.izquierda
B=Q.derecha
C=P.derecha
// realizar la rotación
Q.derecha=P
P.izquierda=B

Debemos contemplar el que PadreP exista o que P sea raíz
Si P es Raiz la nueva Raiz será Q
Si PadreP es distinto de null y P no es Raiz cambiar el hijo izquierdo o derecho de PadreP (según corresponda) que apuntaba a P apuntando a Q

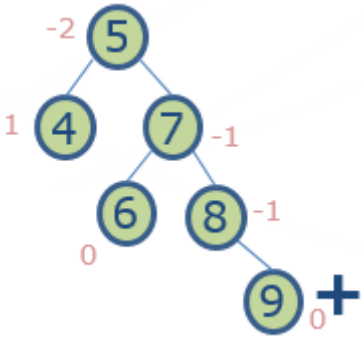
Rotación simple a izquierda

De un árbol de raíz (r) y de hijos izquierdo (i) y derecho (d), consiste en formar un nuevo árbol cuya raíz sea la raíz del hijo derecho, como hijo derecho colocamos el hijo derecho de d (nuestro d') y **como hijo izquierdo construimos un nuevo árbol que tendrá como raíz la raíz del árbol (r)**, el hijo izquierdo de d será el hijo derecho (i') y el hijo izquierdo será el hijo izquierdo del árbol (i).
Precondición : Tiene que tener hijo derecho no vacío.

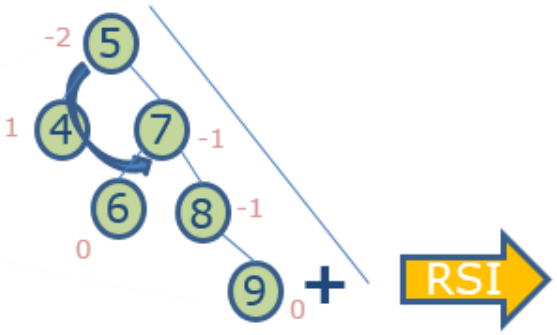
Ej: Partiendo de un árbol AVL agregamos un nodo y nos queda un árbol desequilibrado

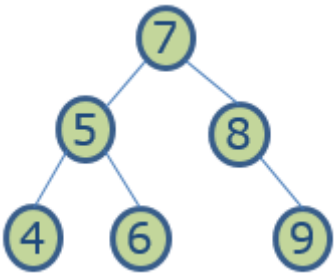


Árbol avl



Árbol desbalanceado





Viéndolo en un ejemplo el algoritmo a aplicar es el siguiente:
Tengamos siempre en cuenta que ***Fe=NivSubArbIzquierdo-NivSubArbDerecho***

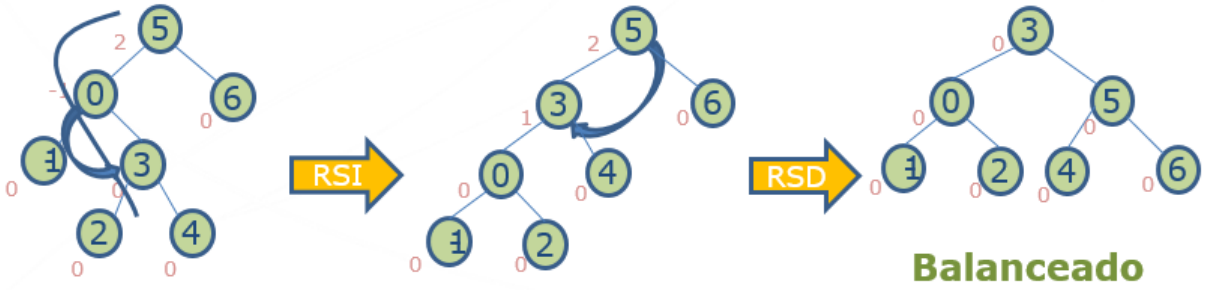
P: Es el nodo con Fe<-1 y PadreP el nodo padre de P
Q=P.derecha
A=P.izquierda
B=Q.izauierda
C=Q.derecha
// realizar la rotación
Q.izquierda=P
P.derecha=B

Debemos contemplar el que PadreP exista o que P sea raíz
Si P es Raiz la nueva Raiz será Q
Si PadreP es distinto de null y P no es Raiz cambiar el hijo izquierdo o derecho de PadreP (según corresponda) que apuntaba a P apuntando a Q

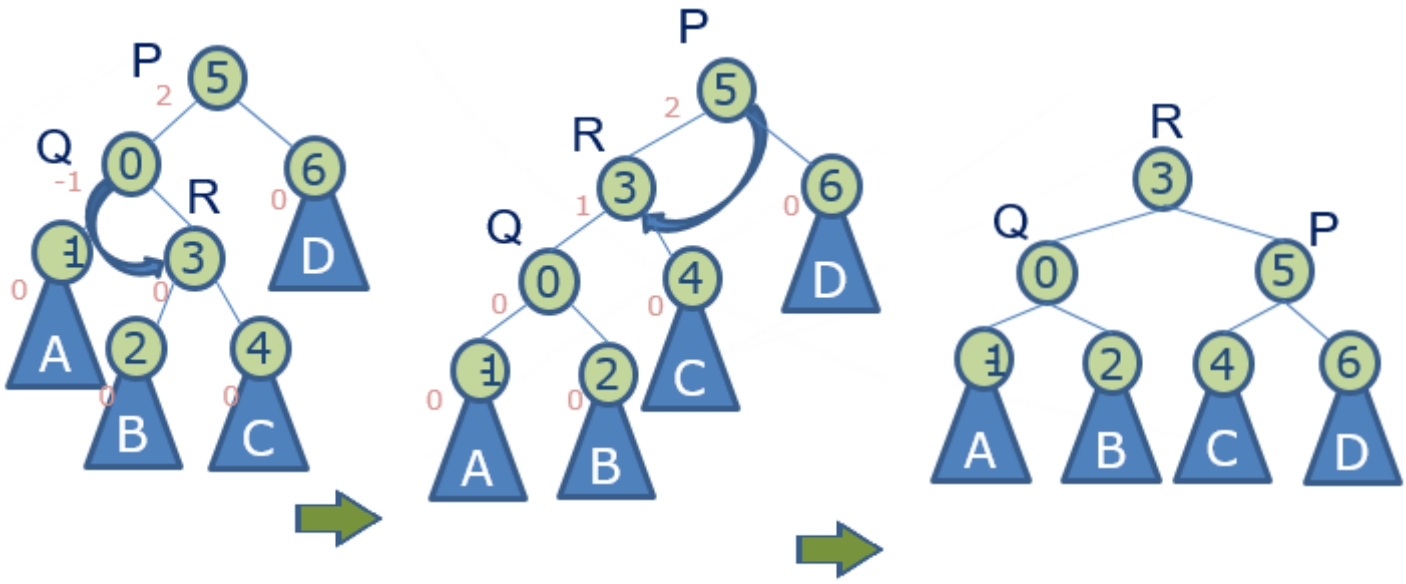
Rotación doble a derecha

La Rotación **Doble a la Derecha** son dos rotaciones simples, primero rotación simple izquierda y luego rotación simple derecha.

Un ejemplo seria



Viéndolo en un ejemplo el algoritmo a aplicar es el siguiente:
Tengamos siempre en cuenta que ***Fe=NivSubArbIzquierdo-NivSubArbDerecho***



P: Es el nodo con Fe>1 y Q con Fe<0 y PadreP el nodo padre de P
Q=P.izquierda
R=Q.derecha
A=Q.izquierda
B=R.izquierda
C=R.derecha
D=P.derecha

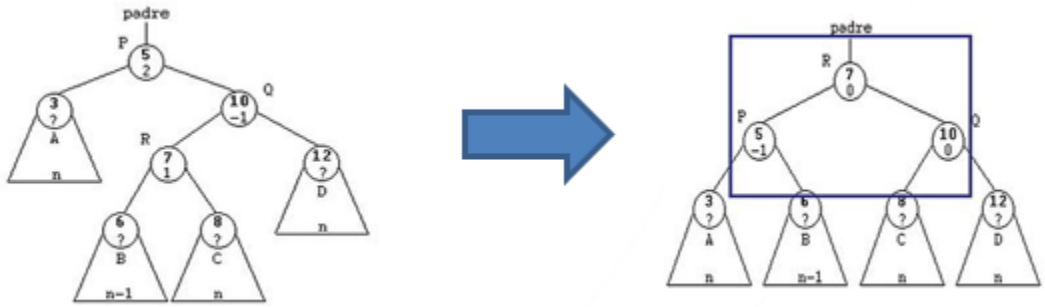
// realizar la rotación 1
Q.derecha=B
P.izquierda=R
R.izquierda=Q

// realizar la rotación 2
P.izquierda=C
R.derecha=P

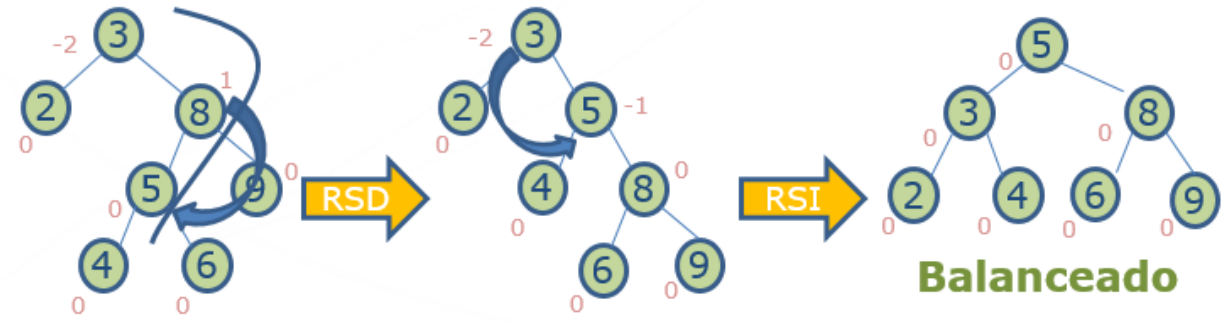
Debemos contemplar el que PadreP exista o que P sea raíz
Si P es Raiz la nueva Raiz será R
Si PadreP es distinto de null y P no es Raiz cambiar el hijo izquierdo o derecho de PadreP (según corresponda) que apuntaba a P apuntando a R

Rotación doble a izquierda

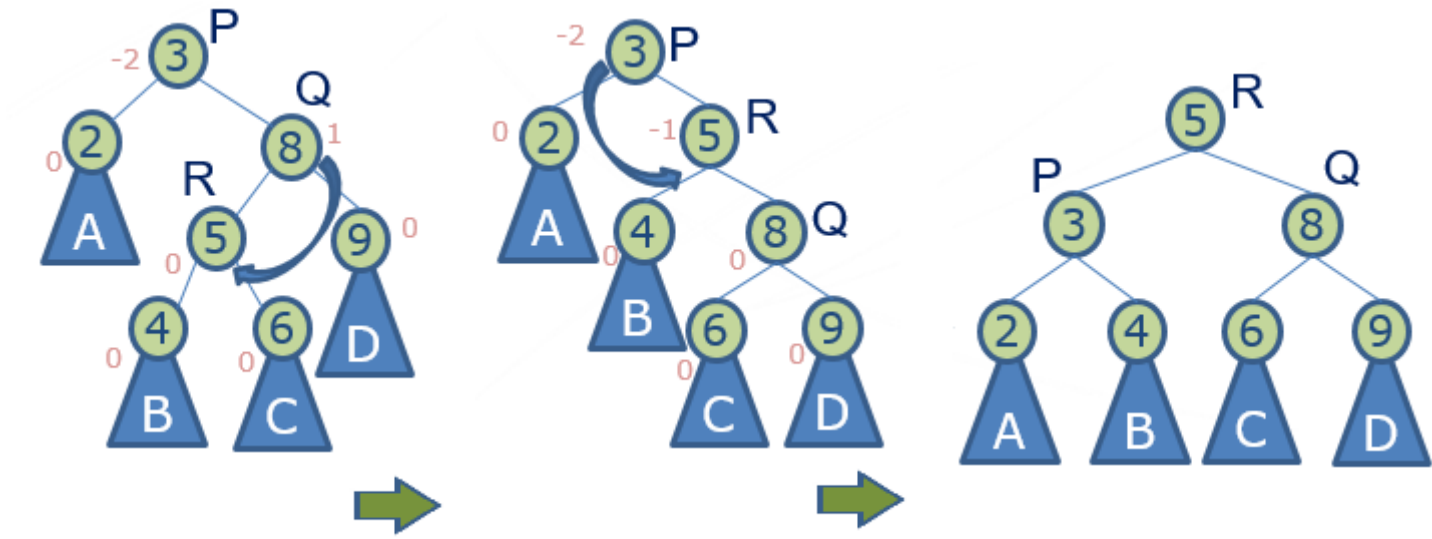
La Rotación **Doble a la Izquierda** son dos rotaciones simples, primero rotación simple derecha y luego rotación simple izquierda.



Un ejemplo seria



Viéndolo en un ejemplo el algoritmo a aplicar es el siguiente:
Tengamos siempre en cuenta que ***Fe=NivSubArbIzquierdo-NivSubArbDerecho***



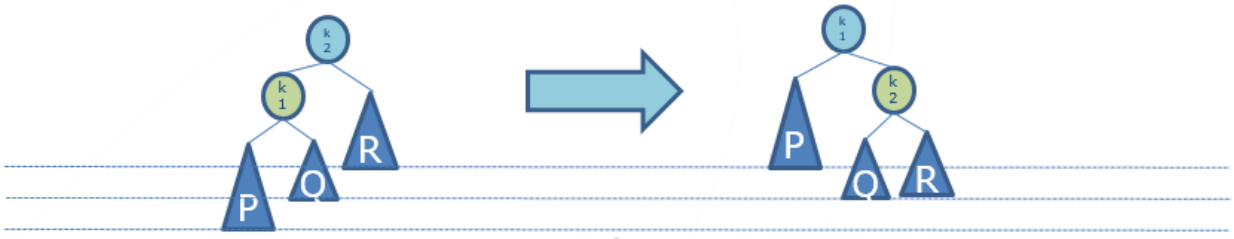
P: Es el nodo con Fe<-1 y Q con Fe>0 y PadreP el nodo padre de P
Q=P.derecha
R=Q.izquierda
A=P.izquierda
B=R.izquierda
C=R.derecha
D=Q.derecha

// realizar la rotación 1
Q.izquierda=C
P.derecha=R
R.derecha=Q

// realizar la rotación 2
P.derecha=B
R.izquierda=P

Debemos contemplar el que PadreP exista o que P sea raíz
Si P es Raiz la nueva Raiz será R
Si PadreP es distinto de null y P no es Raiz cambiar el hijo izquierdo o derecho de PadreP (según corresponda) que apuntaba a P apuntando a R

Balance
Una vez definidas las rotaciones, necesitamos un método que realice el balance en cada vez que insertemos un elemento en el árbol.



El método debe realizar una comparación de las alturas del sub-árbol izquierdo con el sub-árbol derecho. Si la diferencia de alturas es igual a 2, entonces el árbol necesita ser balanceado por medio de rotaciones. Si la altura del sub-árbol izquierdo es mayor que la del sub-árbol derecho, entonces se sostiene que el desequilibrio está hacia la izquierda, caso contrario, se afirma que el desequilibrio está hacia la derecha.
Una vez que sabemos de qué lado se encuentra el desequilibrio, debemos analizar si necesitamos una rotación simple o doble por medio de las alturas. Si es un desequilibrio hacia la izquierda, debemos conocer cuál de los sub-árboles del sub-árbol izquierda tiene mayor altura. Si es el izquierdo, entonces se trata de una rotación simple y si es el derecho, entonces es una rotación doble. El mismo razonamiento se aplica cuando es un desequilibrio a derecha.

Inserción

La inserción en un árbol de AVL puede ser realizada insertando el valor dado en el árbol como si fuera un árbol de búsqueda binario desequilibrado y después retrocediendo hacia la raíz, rotando sobre cualquier nodo que pueda haberse desequilibrado durante la inserción.
Proceso de inserción:

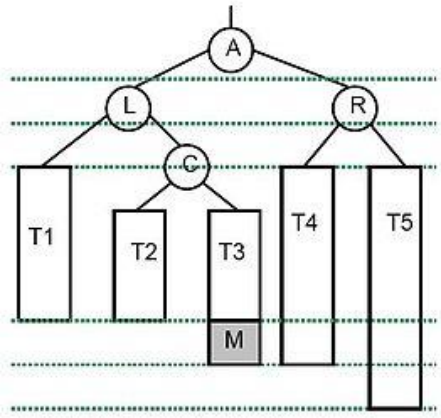
1.buscar hasta encontrar la posición de inserción o modificación (proceso idéntico a inserción en árbol binario de búsqueda)

2.insertar el nuevo nodo con factor de equilibrio “equilibrado”

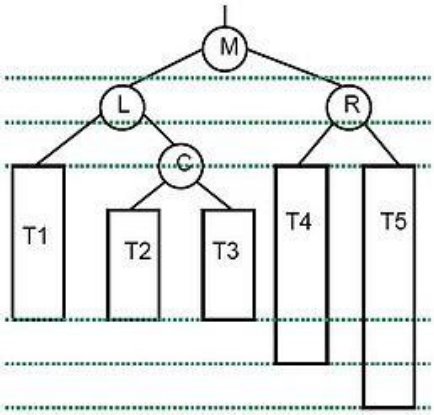
3.desandar el camino de búsqueda, verificando el equilibrio de los nodos, y re-equilibrando si es necesario

Eliminar

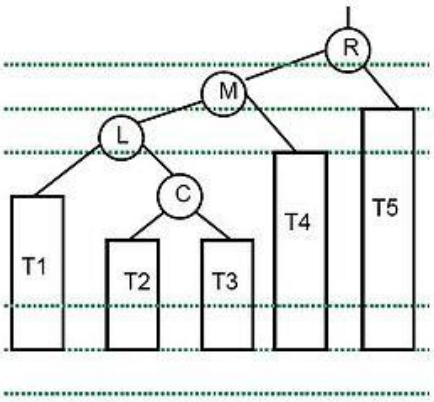
El procedimiento de **borrado es el mismo que en el caso de árbol binario de búsqueda**. La diferencia se encuentra en el proceso de reequilibrado posterior. El problema de la extracción puede resolverse en O (log n) pasos. Una extracción trae consigo una disminución de la altura de la rama donde se extrajo y tendrá como efecto un cambio en el factor de equilibrio del nodo padre de la rama en cuestión, pudiendo necesitarse una rotación.
Esta disminución de la altura y la corrección de los factores de equilibrio con sus posibles rotaciones asociadas pueden propagarse hasta la raíz.



Borrar A, y la nueva raíz será M.



Borrado A, la nueva raíz es M. Aplicamos la rotación a la izquierda.



El árbol resultante ha perdido altura.

En borrado pueden ser necesarias varias operaciones de restauración del equilibrio, y hay que seguir comprobando hasta llegar a la raíz.

Operaciones de balanceo según la operación previa

*Balanceado o equilibrado de un árbol AVL por **inserción de un nodo**.*

Fe de nodo actual	**Fe** nodo derecho	**Fe** nodo izquierdo	**Rotación**
2	No importa	>0	RSD
2	No importa	<0	RDD
-2	>0	No importa	RDI
-2	<0	No importa	RSI

*Balanceado o equilibrado de un árbol AVL por **borrado de un nodo**.*

Fe de nodo actual	**Fe** nodo derecho	**Fe** nodo izquierdo	**Rotación**
2	No importa	>=0	RSD
2	No importa	<0	RDD
-2	>=0	No importa	RSI
-2	<0	No importa	RDI

Consideraciones de la altura

Como vimos, se necesitará tener ***acceso a la altura de cada nodo del árbol***.
"Dado que la función para hallar la altura de un nodo dado en un árbol tiene un tiempo de ejecución de **O(log(n))** en el peor caso…"
Esto nos impacta cada vez que pretendemos insertar o eliminar un nodo.
Para solucionar este problema, podemos almacenar una variable **altura** en cada nodo e irla actualizando en las inserciones y eliminaciones que se efectúen sobre el árbol y así, conseguir que el método de la altura sea constante **O(1)**. De esta forma, podemos redefinir la función de altura aprovechando el campo altura que ahora tiene cada nodo del árbol.