UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

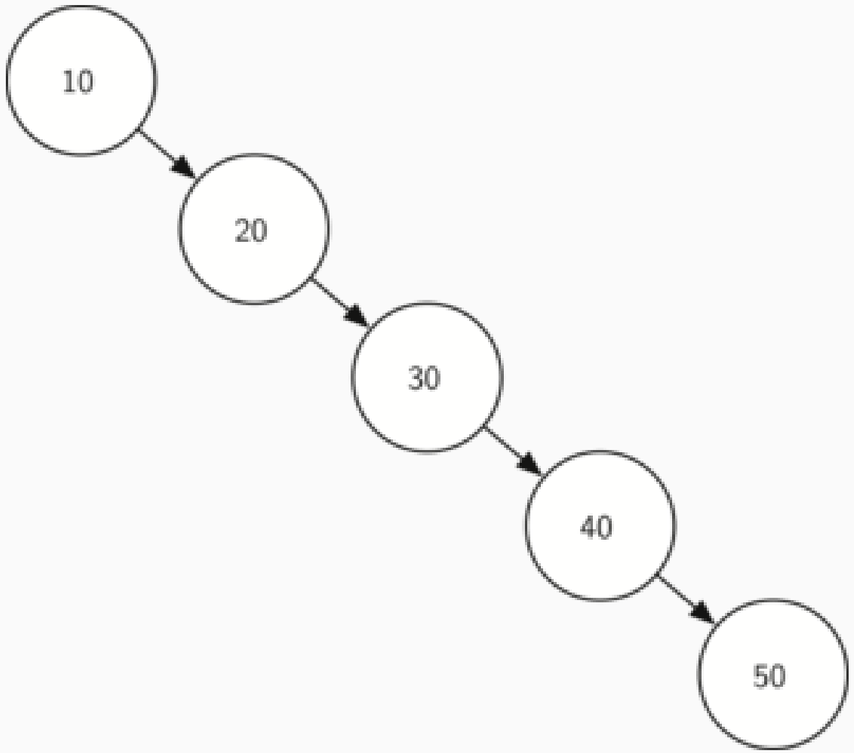
Árbol Binario de Búsqueda

Los árboles binarios de búsqueda (BST) permiten tiempos muy eficientes en sus funciones como: búsqueda, inserción y remover.

En todos los casos promedios O(log n). Y ocupan un espacio de O(n)

¿Cuál es el problema de los BST?
El problema está en que deben estar balanceados, sino podrían caer en los peores casos O(n)

¿Cómo solucionarían su desbalance?



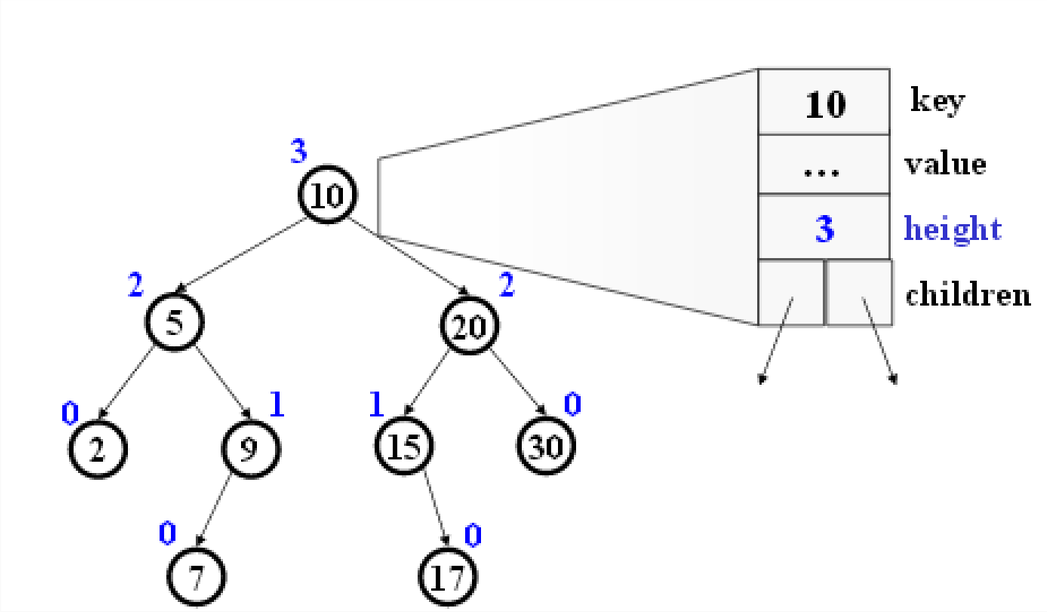
Árbol AVL: BST Balanceado

Es el primer árbol binario auto balanceado que se inventó

Ocupa un espacio de O(n), y tanto su búsqueda, inserción y borrado toman O(log n) en su caso promedio y peor

Son árboles similares a los *red-black*, ya que soportan el mismo conjunto de operaciones

Las búsquedas en un árbol AVL se realizan de la misma manera que en un BST



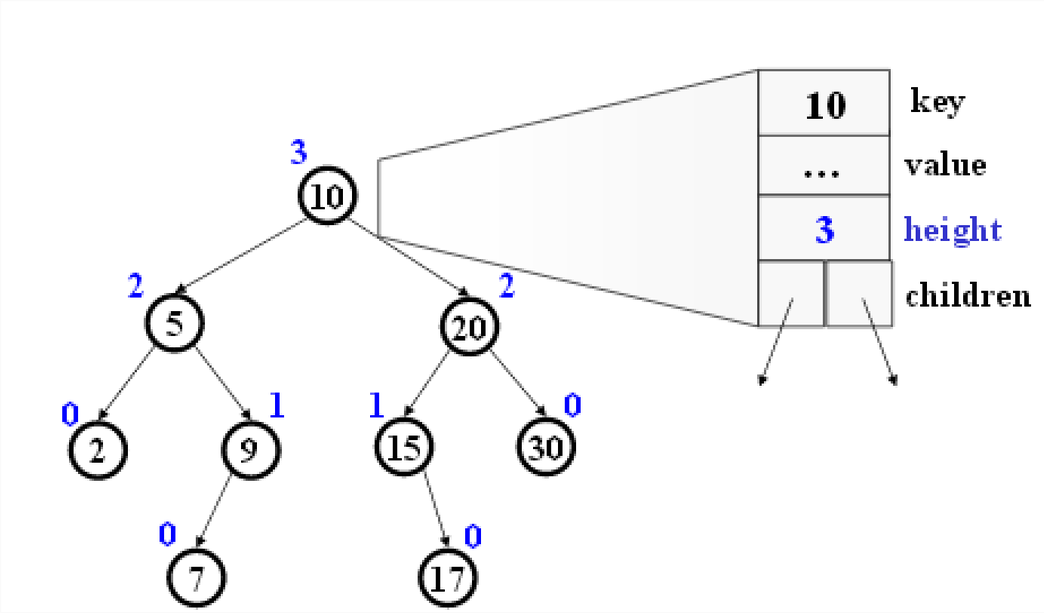
Árbol AVL

En el árbol AVL las alturas de los sub-árboles de cada nodo difieren por un máximo de uno.

Altura(H): Para un árbol T con v como su raíz. La altura es la longitud de la ruta más larga de v a una hoja

Entonces, para todo nodo ***u*** su **factor de balanceo (hb)** se define:

hb(u) = H(Left(u)) - H(Right(u))



donde los resultados deben ser {**0:** balanceado, **+1**: cargado a la izquierda, **-1**: cargado a la derecha}

Árbol AVL (calcule el factor de equilibrio)

Inserta los siguientes elementos y calcular el balanceo de cada nodo:

57, 35, 92, 64, 44, 77, 13, 68, 87, 13, 88, 90, 7, 23, 24, 14

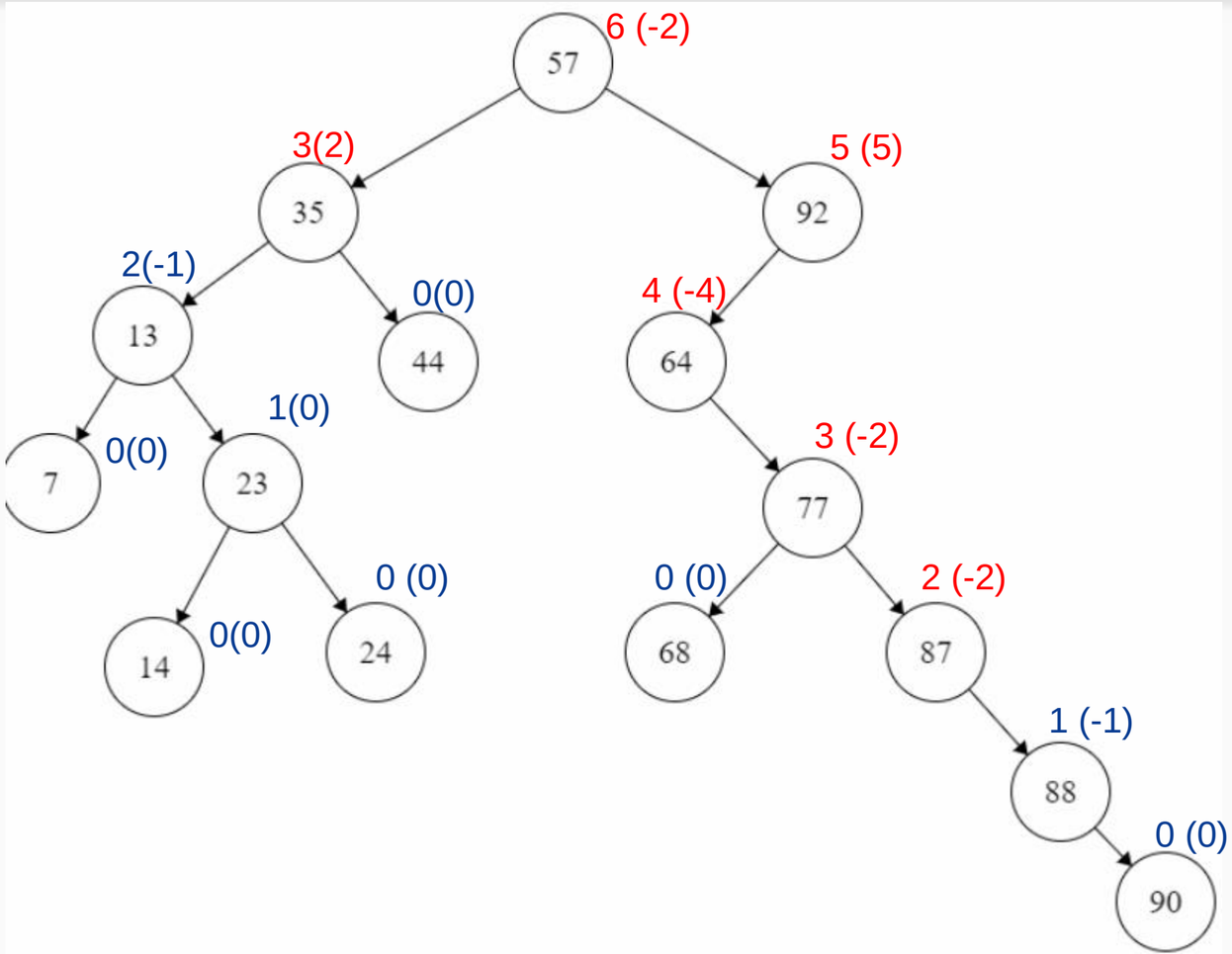
hb(u) = H(Left(u)) - H(Right(u))

Árbol AVL (calcule el factor de equilibrio)

Inserta los siguientes elementos y calcular el balanceo de cada nodo:

57, 35, 92, 64, 44, 77, 13, 68, 87, 13, 88, 90, 7, 23, 24, 14

hb(u) = H(Left(u)) - H(Right(u))



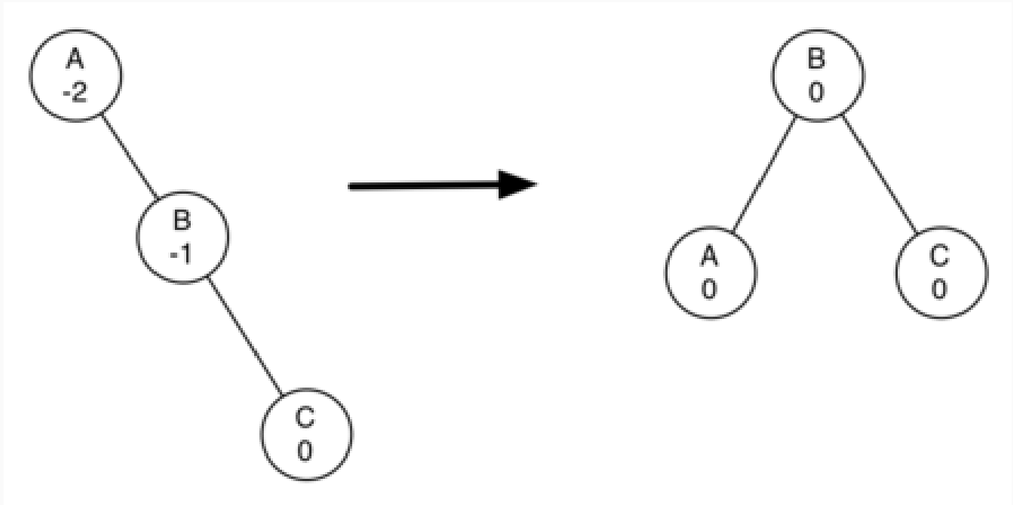
Árbol AVL

Dado que el árbol AVL es un árbol binario, los nuevos elementos también se agregan como hojas

El nuevo nodo hoja actualizará los factores de equilibrio de los nodos en la ruta que tomó para su ingreso

Si el equilibrio cae fuera del rango aceptado, entonces es necesario volver a equilibrar el árbol

Existen tres casos posibles cuando se inserta un nuevo elemento



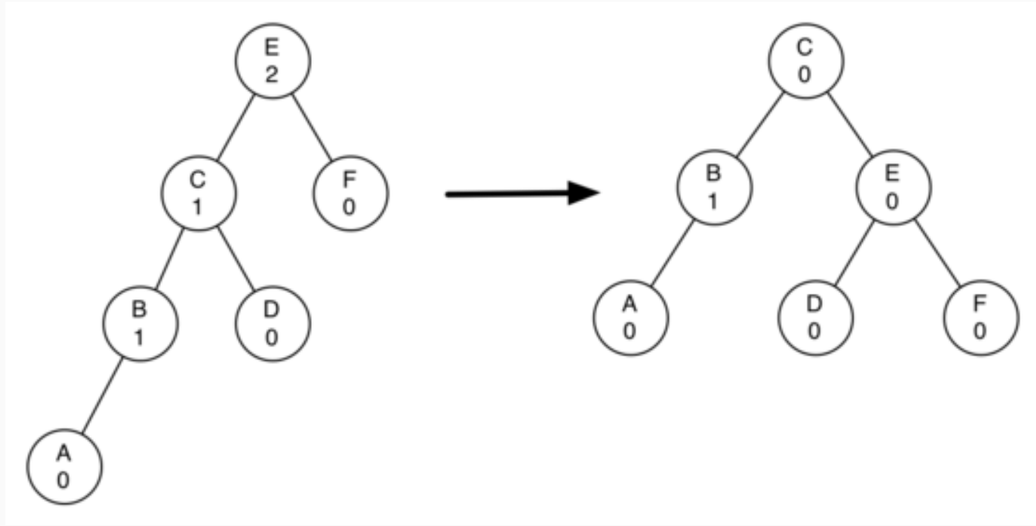
Inserción

Caso 1: La altura del árbol se mantiene y basta con agregar el nuevo nodo

Caso 2: Se necesita volver a equilibrar, por ello se realizan rotaciones simples o dobles

Caso 3: La altura del árbol crece, pero no hay necesidad de volver a equilibrar. Solo se actualizan las alturas

El peor caso del AVL tree, es cuando se requiere hacer rotaciones en todos los niveles hasta llegar al padre



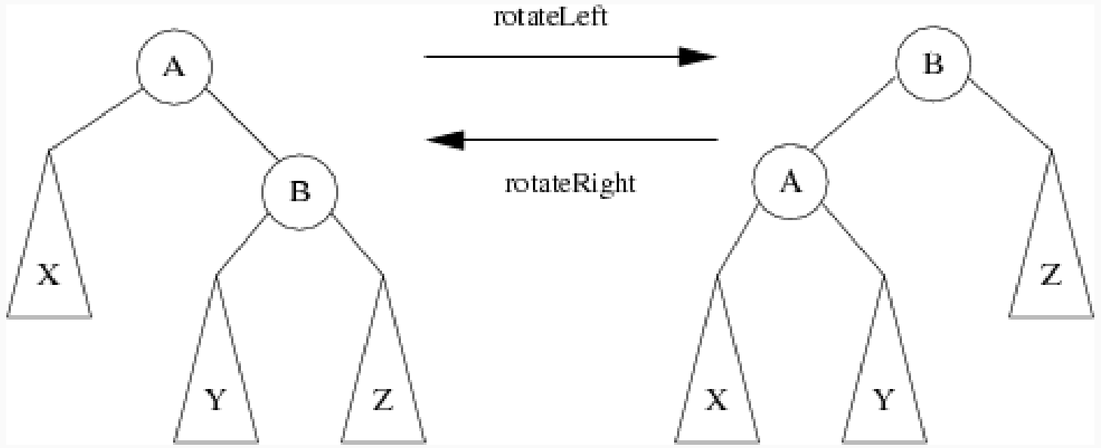
Rotaciones simples

Nos podemos dar cuenta que es una rotación simple si los valores de equilibrio del nuevo nodo con su padre tienen el mismo signo (e.g. +2, +1)

La rotación se hace a partir del nodo crítico, el cual es el que tiene el factor de equilibrio fuera del rango

Ejemplo, insertar: 75, 24, 10

Qué pasaría en el siguiente caso?
Insertar: 75, 24, 32



xAyBz = xAyBz

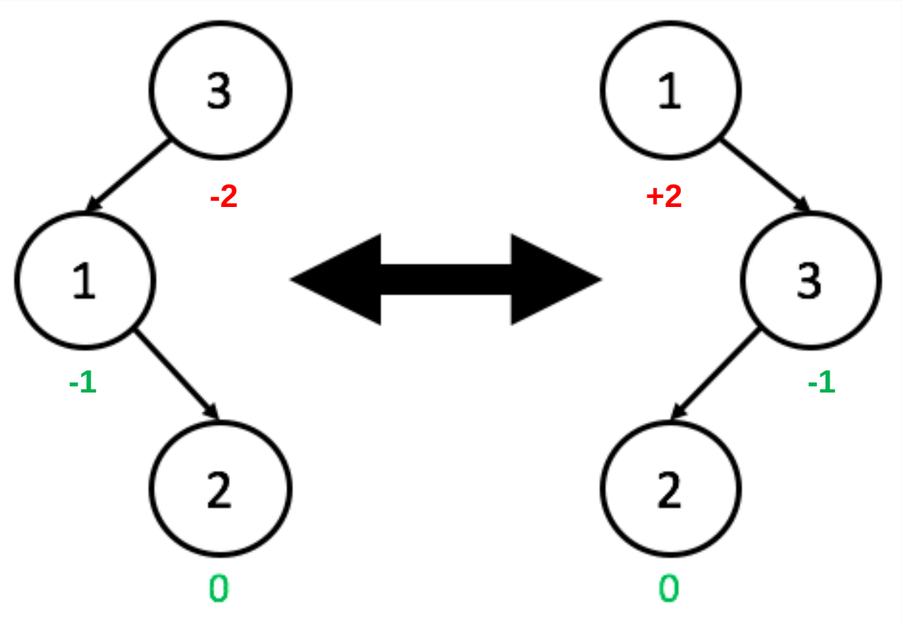
Rotaciones dobles

Las rotaciones dobles son importantes en situaciones donde una rotación simple nos dejaría en el mismo problema (ver imagen)

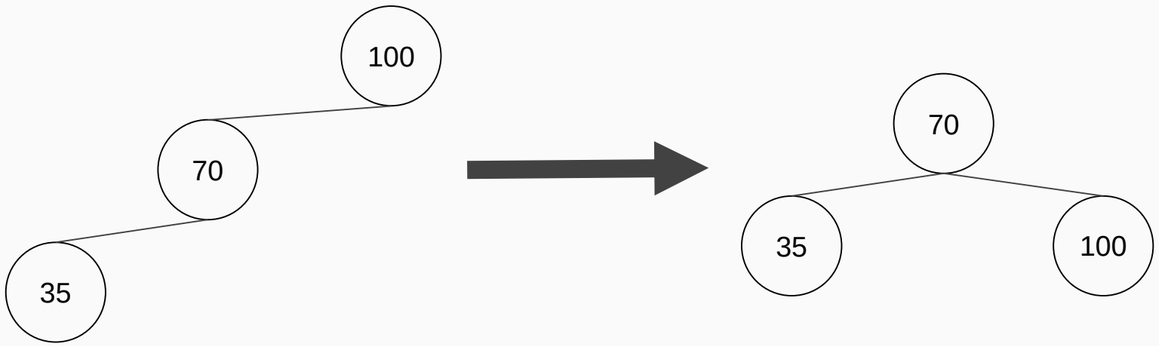
En dicho caso se debe realizar primero una rotación contraria sobre el nodo hijo

En la imagen se debería hacer una rotación a la izquierda sobre el nodo 1 primero y después una rotación a la derecha

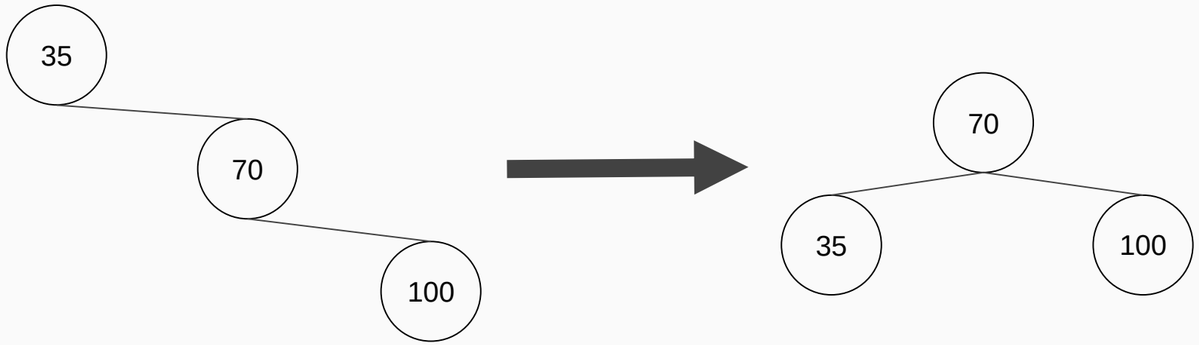
Esto sería una rotación izquierda derecha



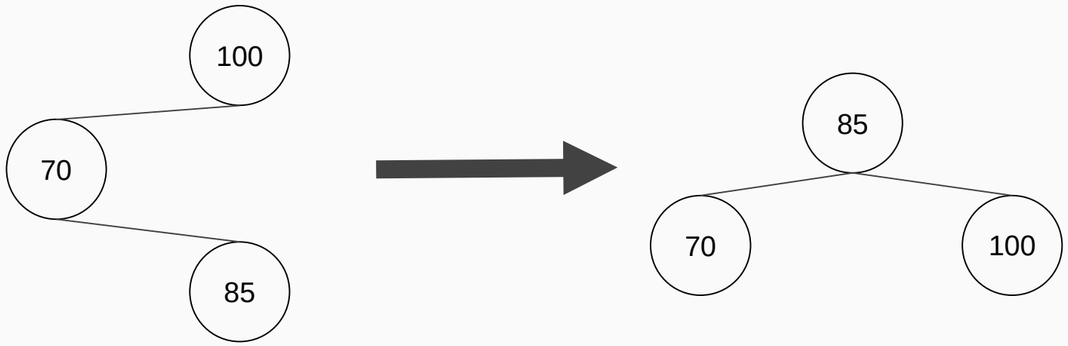
Rotaciones



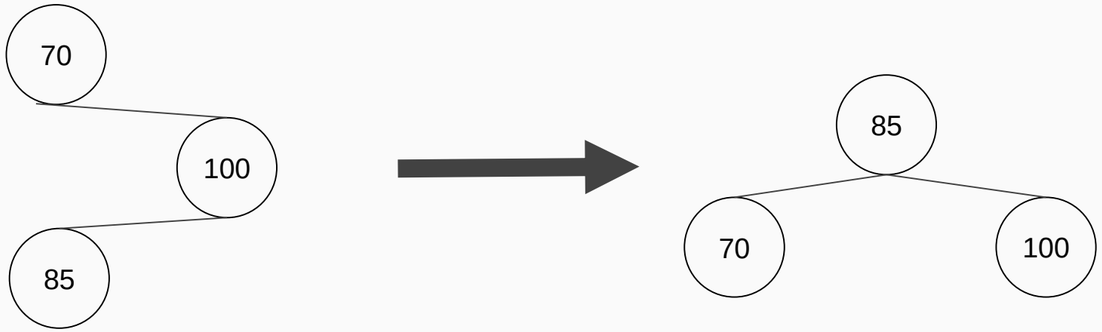
Rotación a la derecha



Rotación a la izquierda



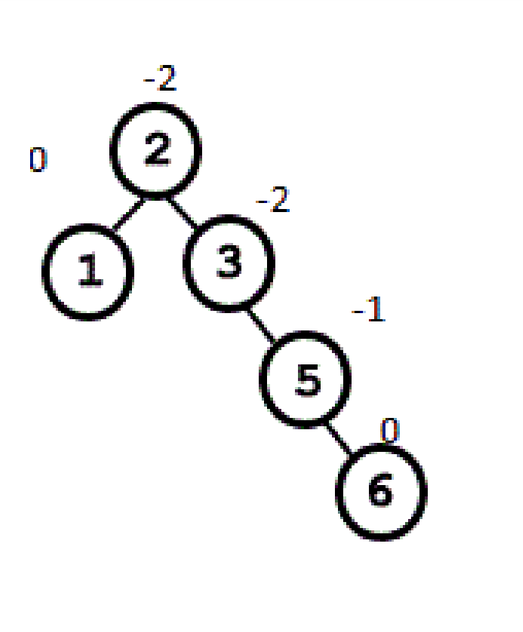
Rotación a la izquierda derecha

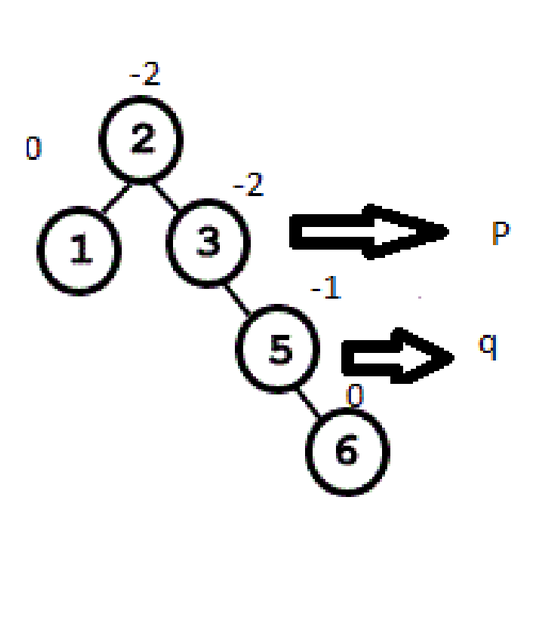


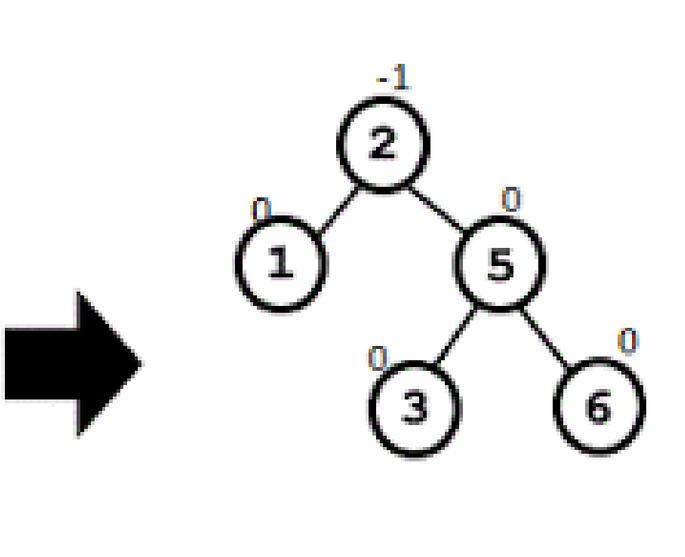
Rotación a la derecha izquierda

Rotaciones simples

- Ejemplo: rotación simple izquierda desde 3

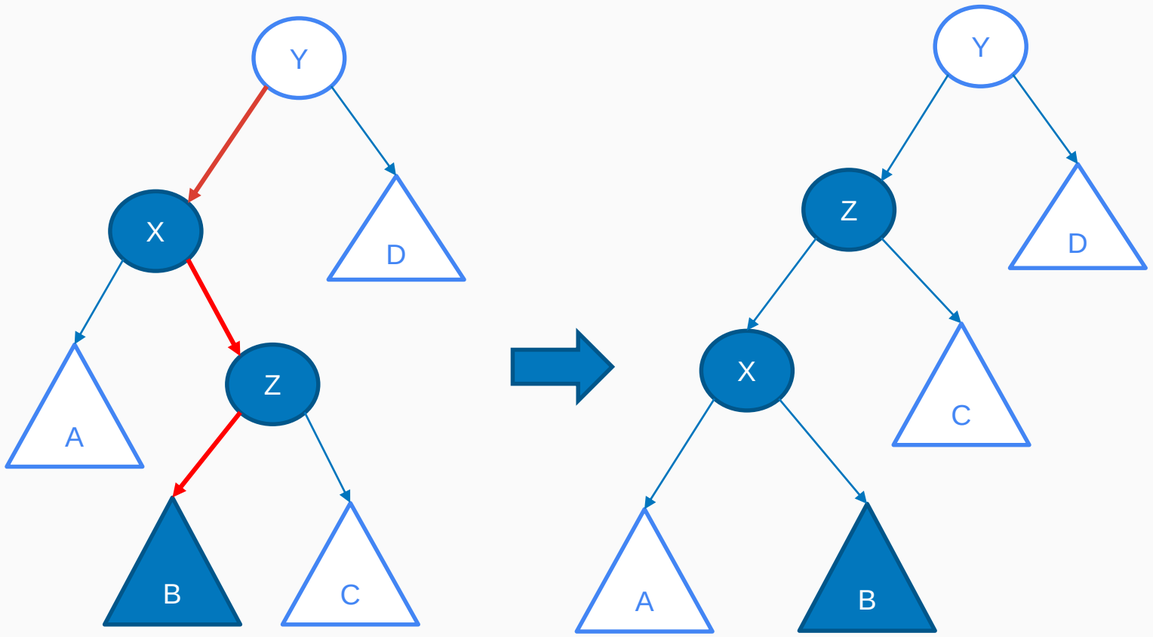






Rotaciones (algoritmo)

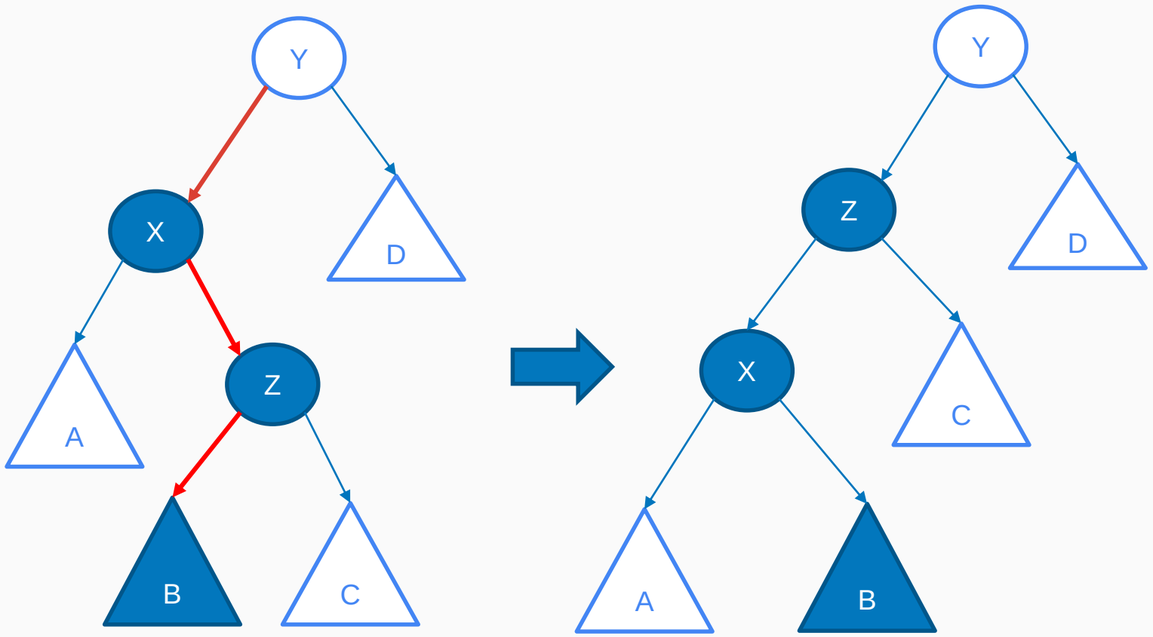
Rotación a la izquierda en X



L_Rota(&X):

Rotaciones (algoritmo)

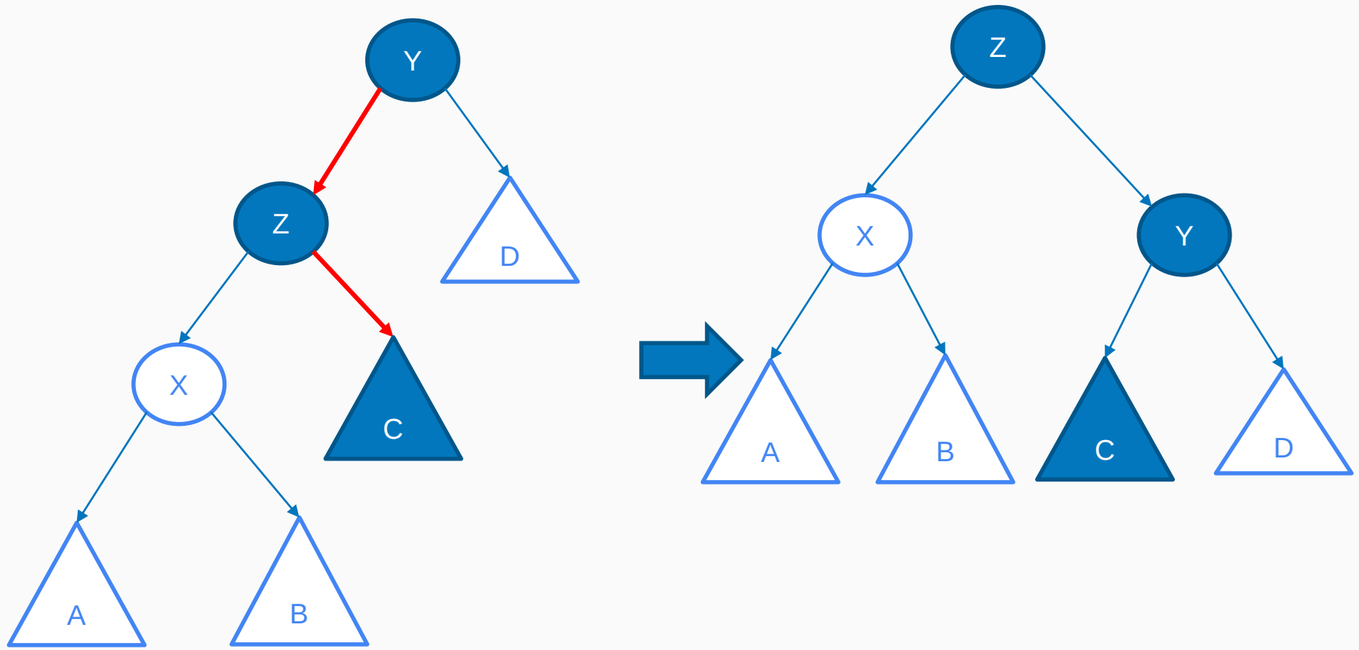
Rotación a la izquierda en X




```
L_Rota(&X):
    Z = X->right
    X->right = Z->left
    Z->left = X
    X = Z
```

Rotaciones (algoritmo)

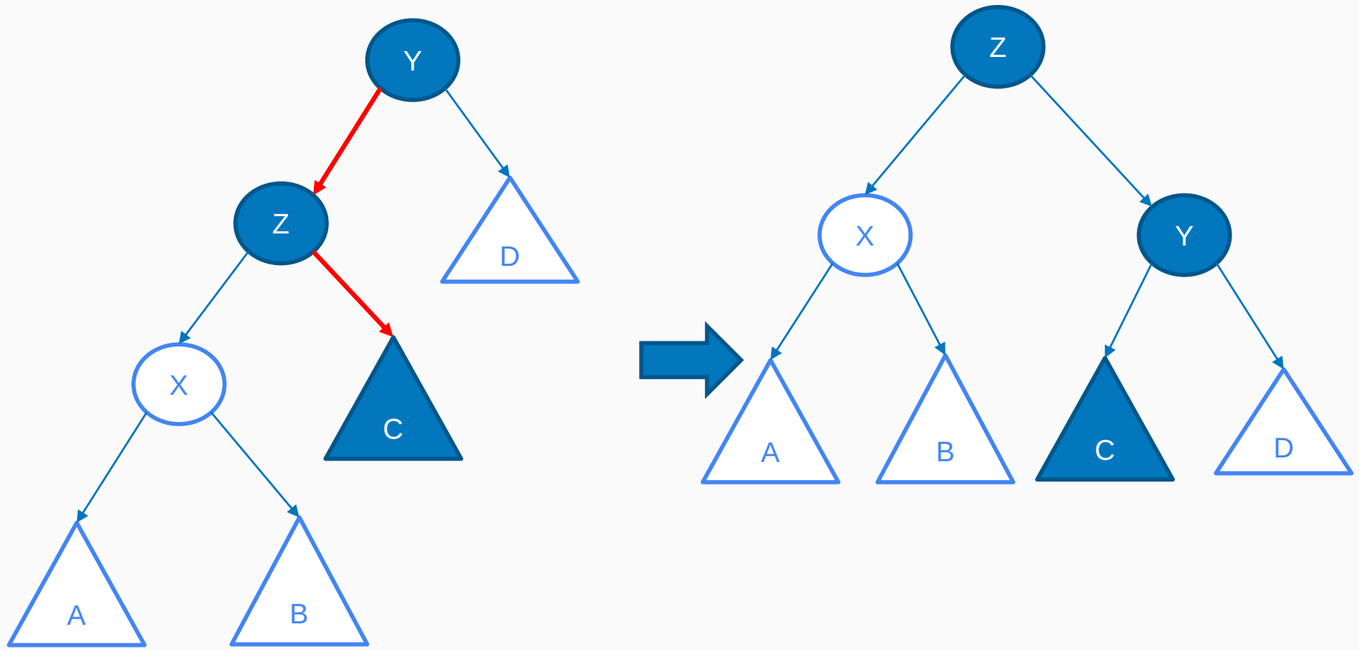
Rotación a la derecha en Y



R_Rota(&Y):

Rotaciones (algoritmo)

Rotación a la derecha en Y




```
R_Rota(&Y):
    Z = Y->left
    Y->left = Z->right
    Z->right = Y
    Y = Z
```

Rotaciones (algoritmo)

Rotación doble izquierda derecha

```
LR_Rota(&Y):
    L_Rota(Y->left)
    R_Rota(Y)
```

Rotación doble derecha izquierda

```
RL_Rota(&Y):
    R_Rota(Y->right)
    L_Rota(Y)
```

Rotaciones (algoritmo balancear)

```
Balancear(node)
----------------------
si hb(node) > 1:
        si hb(node->left) > 0
                R_Rota(node)
        caso contrario
                 LR_Rota(node)
si hb(node) < -1:
        si hb(node->right) < 0:
                L_Rota(node)
        caso contrario
                RL_Rota(node)
```

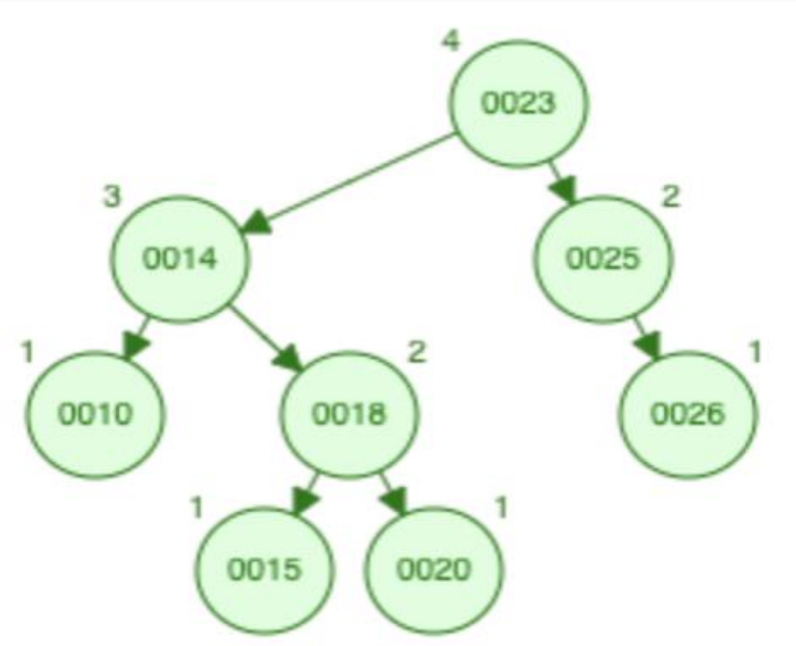
Borrado

Para el borrado hay ciertos casos a tener en cuenta:

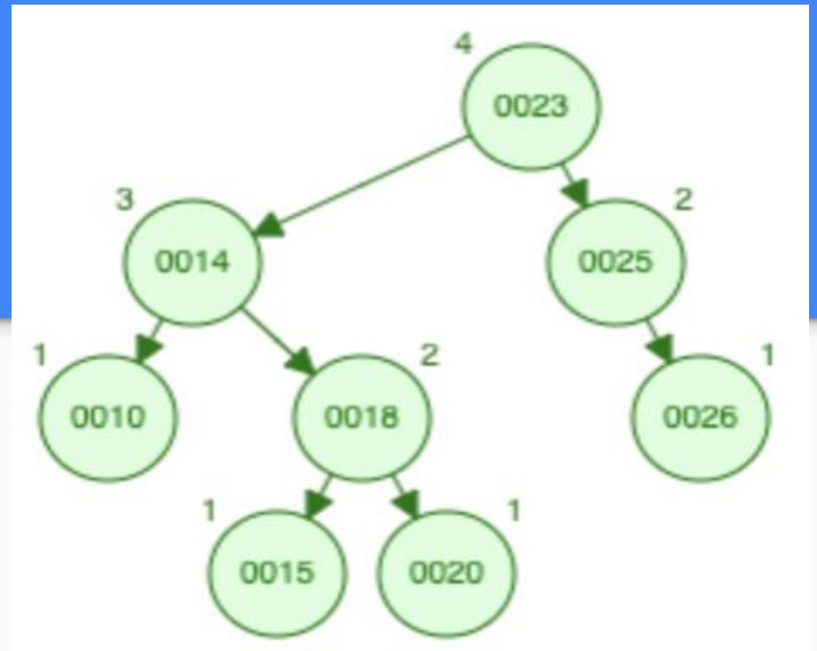
1. Buscar el desbalance verificando los hijos de mayor altura.
2. Este proceso se repite hasta encontrar dos casos e identificar el tipo de rotación.

Por ejemplo, borrar el 26:

- El nodo en desbalance sería el 23, su hijo con mayor altura 14, y el el siguiente 18.
- Entonces sería una rotación LR



Borrado

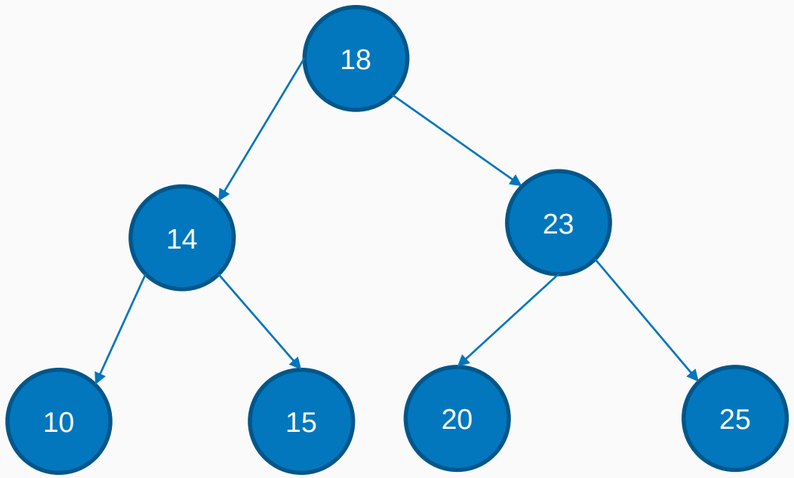


Para el borrado hay ciertos casos a tener en cuenta:

1. Buscar el desbalance verificando los hijos de mayor altura.
2. Este proceso se repite hasta encontrar dos casos e identificar el tipo de rotación.

Por ejemplo, borrar el 26:

- El nodo en desbalance sería el 23, su hijo con mayor altura 14, y el el siguiente 18.
- Entonces sería una rotación LR



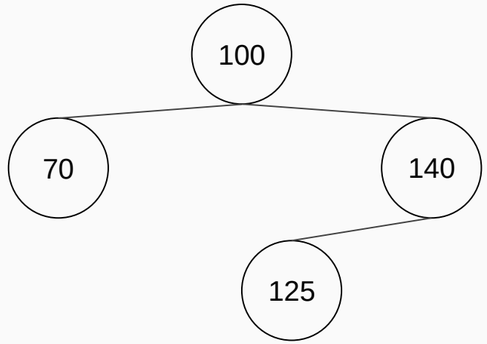
Borrado

El caso del borrado se trata similar al del borrado en un BST:

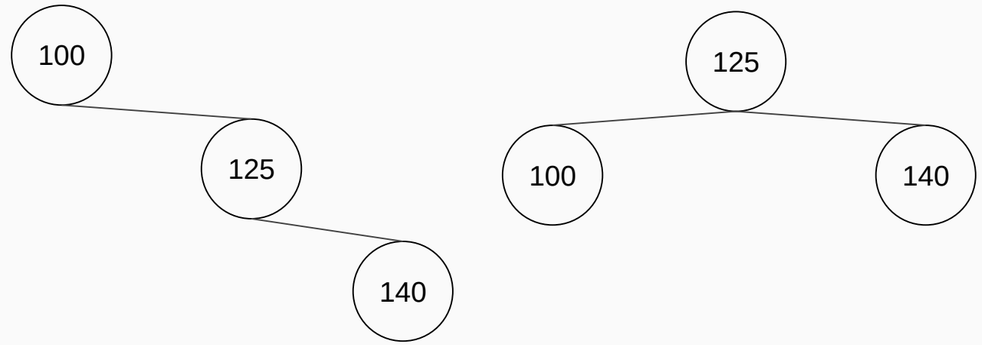
1. Primero debemos cambiar el nodo con el nodo siguiente (derecha, más a la izquierda) o anterior (izquierda, más a la derecha)

2. Se procede a eliminar el nodo en la hoja

3. **Se verifica el equilibrio y se aplican rotaciones si es necesario**



Borrar el 70



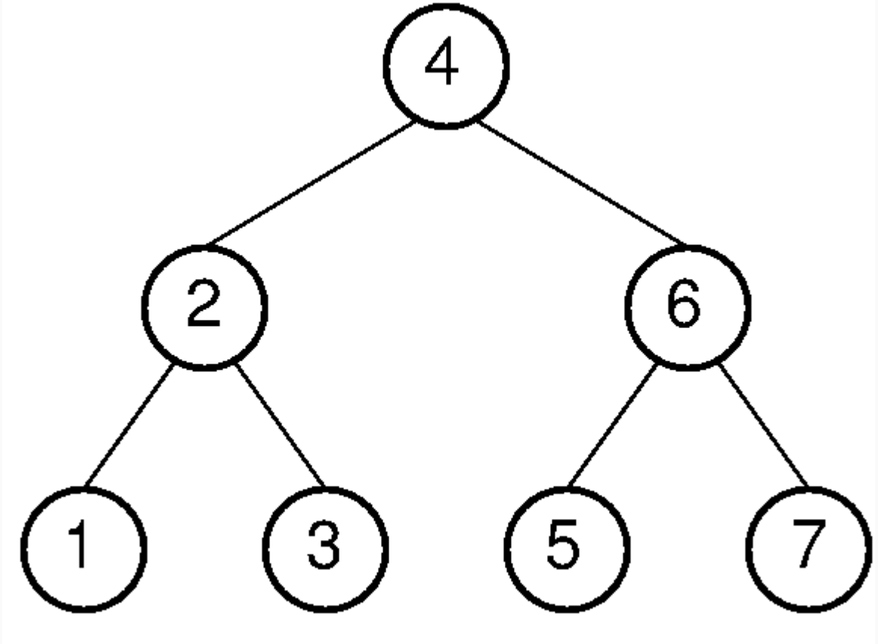
AVL Sort

Un árbol AVL, debido al tiempo de ejecución en sus funciones puede ser utilizado similar a los heaps (e.g. cola de prioridad)

Insertar O(n log n) y el recorrido (traversal) de un árbol AVL se realiza en tiempo O(n)

A pesar que los árboles AVL son bastante eficientes como para reemplazar a los heaps, **por qué se preferiría usar heaps?**

Porque los árboles AVL ocupan más espacio que un heap



Ejemplo

Insertar en un árbol AVL:

35
30
27
11
16
100
50
91
73
5

Remover:

30
35
100

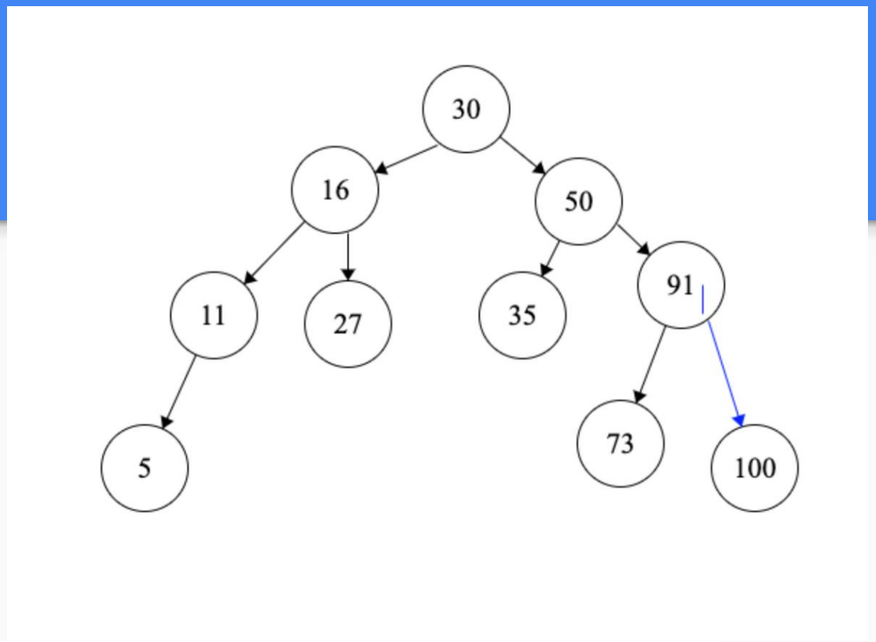
Ejemplo

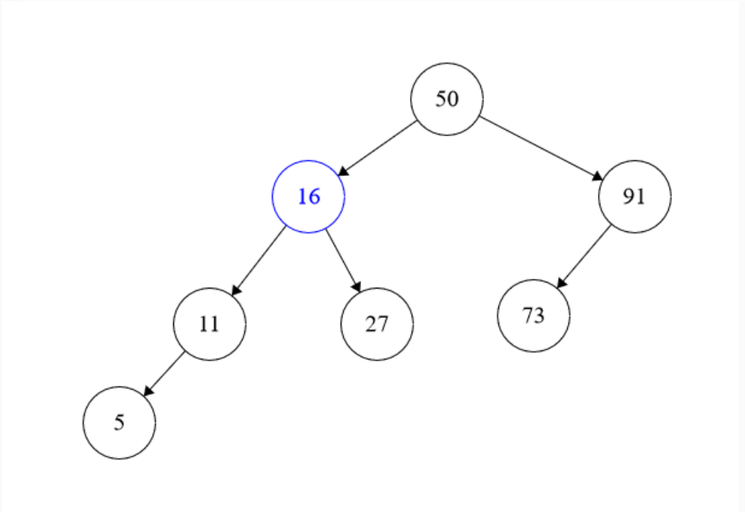
Insertar en un árbol AVL:

35
30
27
11
16
100
50
91
73
5

Remover:

30
35
100





UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100