AVL Trees

> ❗ Razonar con first principles en temas de estructuras de datos es crucial.

Rotations are always performed just by looking at 3 nodes.

Definimos el balance de un nodo n como:

$$B(n) = H(node.left) - H(node.right)$$

$$H(n) = 1 + max\{H(n.left), H(n.right)\}$$

$$H(leaf) = 0$$

Donde $H(n)$ es la altura, es decir el número de aristas del camino más largo desde n hasta un nodo hoja, de un nodo n

La idea de un AVL es que el balance de todo nodo sea o 0, -1 o 1. Hay una prueba matemática que demuestre que si se mantiene esta propiedad, entonces la altura del árbol será logarítmica y por lo tanto todas sus operaciones correrán en $O(\lg\ n)$

Como la altura se define así:

$$H(n) = 1 + max\{H(n.left), H(n.right)\}$$

$$H(leaf) = 0$$

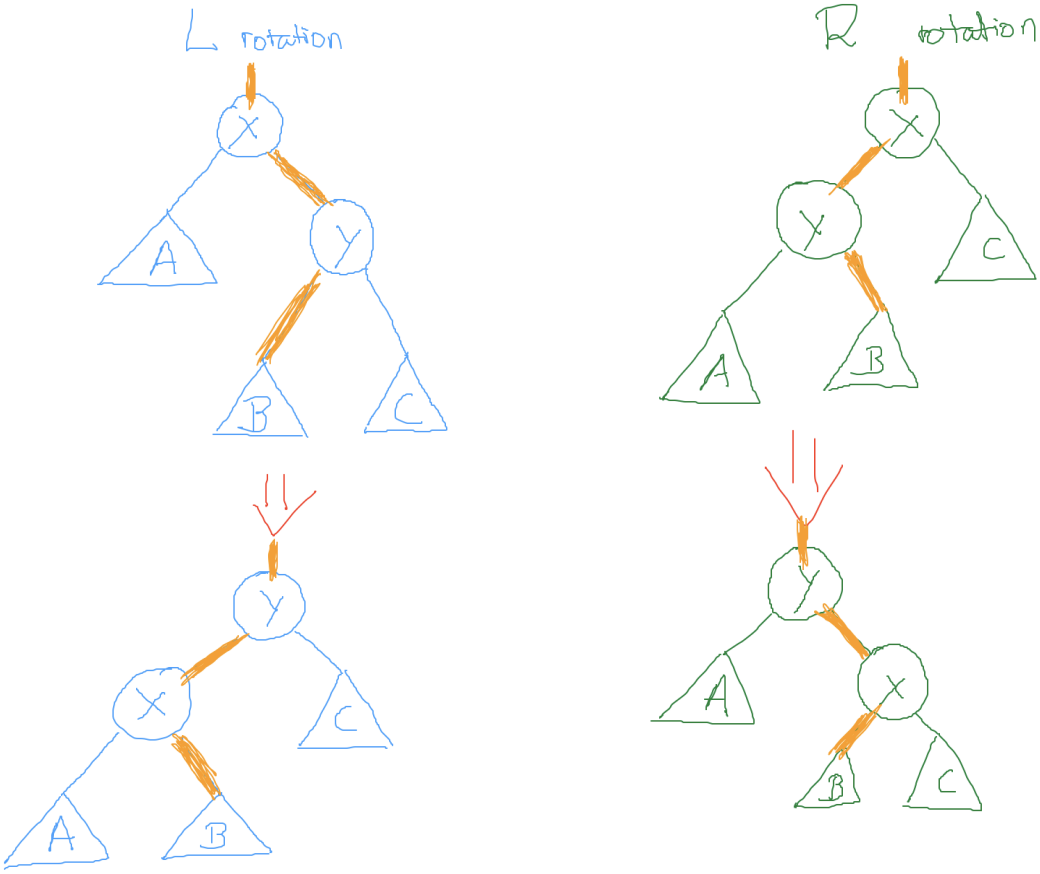
Podemos aprovechar la propiedad de la subestructura óptima para calcular la altura de cada nodo en $O(\lg\ n)$, ya que al insertar un nodo, solo tenemos que actualizar la altura de todos los nodos que están en el camino desde ese nodo hasta la raíz

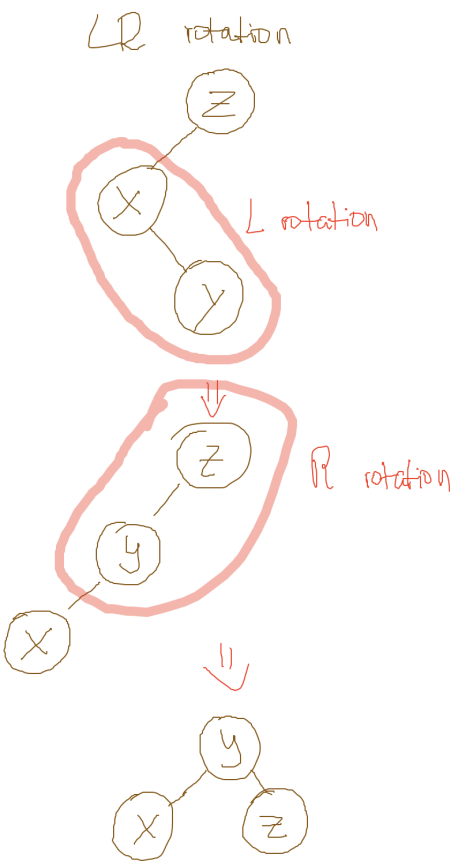
```
# O(h) para actualizar las alturas de cada nodo al insertar
while node.parent not None:
  node.height = max(node.left.height, node.right.height) + 1
  # acá uno checaría el B(node) para ver si toca hacer una rotación
  node = node.parent
```

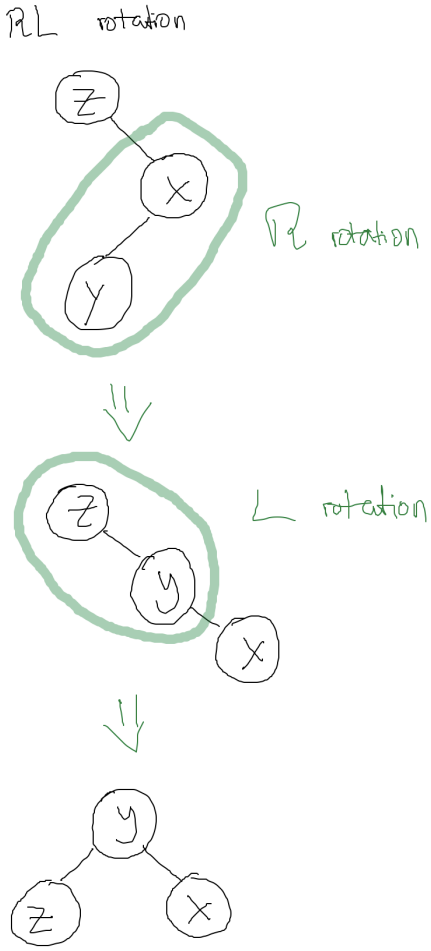
Rotations

La rotación es la técnica que un árbol AVL usa para mantener una altura logarítimica del árbol. Dependiendo del estado del árbol, hay cuatro distintas rotaciones que se pueden aplicar, cada una toma tiempo $O(1)$.

L-rotation	R-rotation	LR-rotation	RL-rotation
Right-heavy: $B(x) < -1$	Left-heavy: $B(x) > 1$	Left-right-heavy: $B(z) > 1$ $B(z.left) \leq 0$	Right-left-heavy: $B(z) < -1$ $B(z.right) \geq 0$
Focus on 2 nodes	Focus on 2 nodes	Focus on 3 nodes	Focus on 3 nodes







> ❗ Aun así luego de hacer una rotación, tenemos que actualizar la altura de todos los nodos que están por encima del *pivote* aplicando el pseudocódigo de antes. ¿Se dan cuenta porque siempre tenemos que actualizar la altura de los nodos **de arriba** pero no los de abajo?

Search

Igual al del BST.

Insert

El insert es el mismo del BST. Sin embargo, al insertar un nodo, hemos alterado la altura de todos los nodos en el camino desde ese nodo borrado hasta la raíz, por lo que tenemos que aplicar el pseudocódigo que presentamos anteriormente para actualizar las alturas de cada nodo y dependiendo del desbalance que descubramos en ese camino, aplicamos las rotaciones pertinentes.

Delete

El delete es el mismo del BST, tres casos:

1. Eliminar una hoja
    a. Eliminarla nomás
2. Eliminar un nodo con un solo hijo
    a. Hacerle un *swap* al nodo con su hijo y el nodo en su nueva posición
3. Eliminar un nodo con dos hijos
    a. Hacerle un *swap* con el predecesor o sucesor y borrar el nodo en su nueva posición

Sin embargo, al borrar el nodo, hemos alterado la altura de todos los nodos en el camino desde ese nodo borrado hasta la raíz, por lo que tenemos que aplicar el pseudocódigo que presentamos anteriormente para actualizar las alturas de cada nodo y dependiendo del desbalance que descubramos en ese camino, aplicamos las rotaciones pertinentes.