Weight-Balanced Trees

1. Background

- A weight-balanced BST is another way to achieve O(log n) tree height.

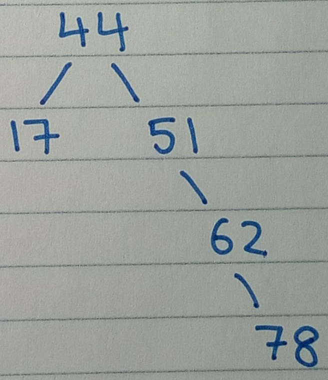
- All weight-balanced BSTs have this property:

  At every node v,
  1. weight(v.left) $\leq$ weight(v.right) x 3
  2. weight(v.right) $\leq$ weight(v.left) x 3

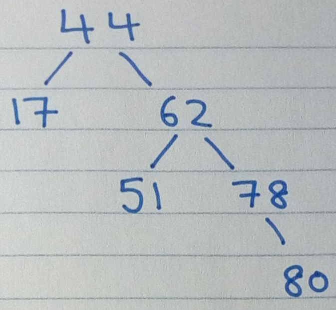
  where weight(v) = size(v)+1

- Examples:

  1.



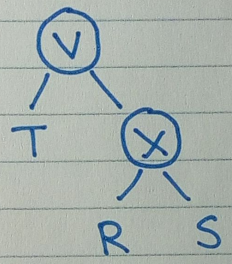
2.



2. Balancing and Rotation:

1. Single Rotation CCW:

Consider the diagram below.

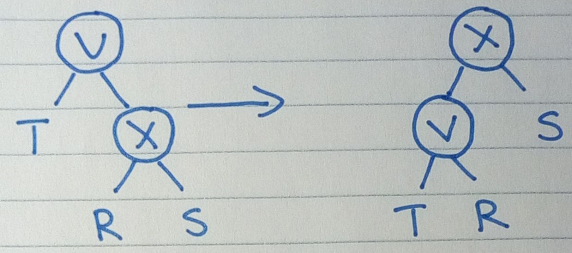


V is right-heavy when
$weight(v.left) \times 3 < weight(v.right)$.

If we need a single rotation, then the tree rooted at v.right must not be left heavier.

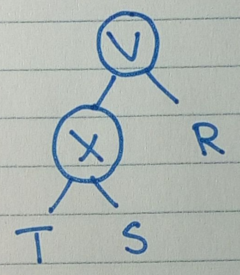
If $weight(x.left) < 2 \times weight(x.right)$ then we rotate around V.

The idea is that T+R should have a combined weight less than 3x weight(S).

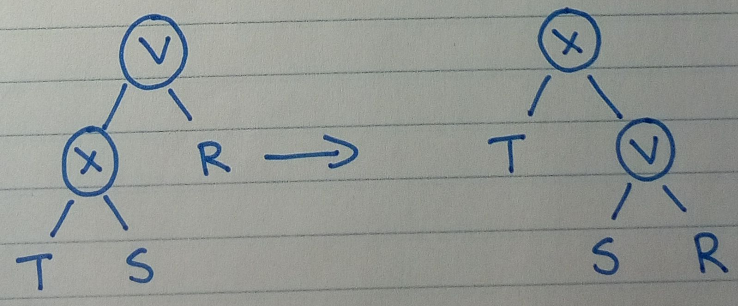


2. Single Rotation CW:

Consider the diagram below.

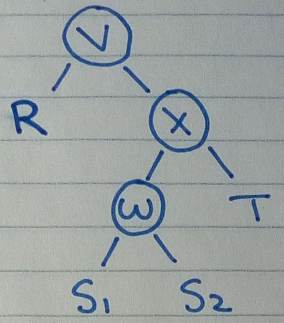


If weight(v.left) > 3x weight(v.right):
    Let x = v.left
    If w(x.left) x2 > weight(x.r):
        rotate CW about V



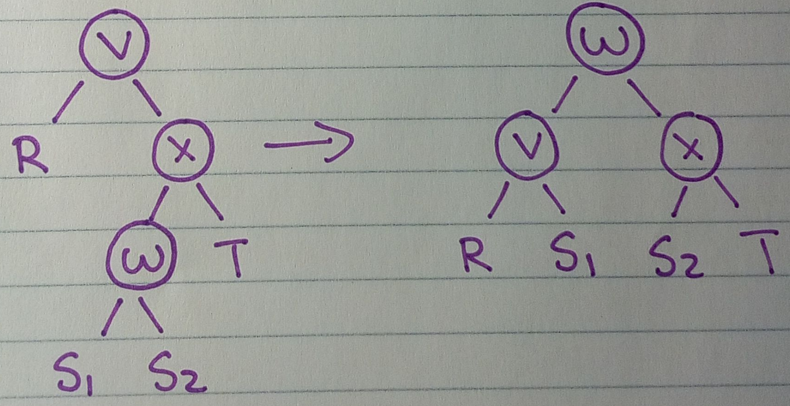
3. Double Rotation CW, CCW

Consider the diagram below.



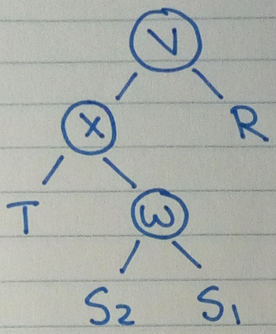
Here, the idea is that w is too heavy.

If w(v.l) x 3 < w(v.r):
    Let x = v.right
    If w(x.l) ≥ w(x.r) x 2:
        Let w = x.left
        Rotate CW about x, then CCW about v.

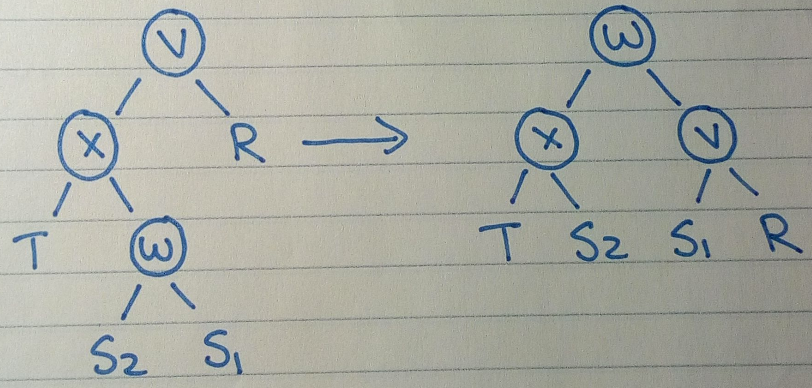


4. Double Rotation CCW, CW

Consider the diagram below.



If $w(v.\ell) > w(v.r) \times 3$:
  Let $x = v.left$
  If $w(x.\ell) \times 2 \leq w(x.r)$:
    Let $w = x.right$
    Rotate CCW about x
    Rotate CW about v



5. Summary of Rebalancing

For each node v on the path from the new/deleted node back to the root:

```
If w(v.l) x 3 < w(v.r):
    let x = v.right
    if w(x.l) < w(x.r) x 2:
        single rotation ccw
    else
        double rotation cw then ccw
else if w(v.l) > w(v.r) x 3:
    let x = v.left
    if w(x.l) x 2 > w(x.r):
        single rotation cw
    else
        double rotation ccw then cw
else
    No rotation
```

3. Insertion:

1. Insert as an AVL Tree.
2. Check and fix balance. $\Theta(\lg n)$ nodes
   Then, update the size from the parent of the new node to the root. $\Theta(1)$ time per node

In total, it takes $\Theta(\lg n)$ time.

4. Deletion:

1. Find the node that has the key.
   $\Theta(\lg n)$ time

2. If the node is a leaf, remove and update the balance starting at the node's parent and up to the root.
   $\Theta(1)$ time

3. Else:
   a) Replace the key of the node with the successor key. $\Theta(\lg n)$ time
   b) Successor's parent adopts successor's right child. $\Theta(1)$ time
   c) From the adopter to root, check and fix balance and update size.
   $\Theta(\lg n)$ time

5. WBT Height

Claim: $h(T) \leq c \times \lg(\text{size}(T)+1)$
where $c = \frac{1}{\lg(4/3)}$ and
$h(T)$ is the height of T.

Proof:
We can do a proof by induction.

Base Case:
T is empty.
$h(T) = h(\text{empty})$
$= 0$

$c \times \lg(\text{size}(T)+1)$
$= c \times \lg(\text{size}(0)+1)$
$= c \times \lg(1)$
$= 0$

$h(T) \leq c \times \lg(\text{size}(T)+1)$, as wanted

Induction Hypothesis:
Suppose for all $k \in \mathbb{N}$ where $0 \leq k < n$, the height of every weight-balanced BST of size k is at most $c \times \lg(k+1)$.

Induction Step:

Let $n_\ell = \text{size}(T.\text{left})$

Let $n_r = \text{size}(T.\text{right})$

$$n = n_\ell + n_r + 1$$

$$\begin{aligned} n+1 &= n_\ell + n_r + 1 + 1 \\ &\geq \frac{(n_r+1)}{3} + n_r + 1 \quad \text{T is balanced} \\ &= \frac{4n_r+4}{3} \\ &= \frac{4}{3}(n_r+1) \end{aligned}$$

$$n_r + 1 \leq \frac{3}{4}(n+1)$$

Without loss of generality, assume $h(T.\text{left}) \leq h(T.\text{right})$.

$$\begin{aligned} h(T) &= 1 + h(T.\text{right}) \\ &\leq 1 + c \cdot \lg(n_r+1) \quad \text{By I.H.} \\ &\leq 1 + c \cdot \lg\left(\tfrac{3}{4}(n+1)\right) \\ &= 1 + c \cdot \lg\left(\tfrac{3}{4}\right) + c \cdot \lg(n+1) \\ &= 1 + (-1) + c \cdot \lg(n+1) \\ &= c \cdot \lg(n+1) \\ &= c \cdot \lg(\text{size}(T)+1), \text{ as wanted} \end{aligned}$$