CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

[1] based on notes by Anna Bretscher and Albert Lai

Weight-balanced Binary Search Trees

Another way to keep a BST balanced: a weight-balanced BST.

Idea: at every node n:

$$\frac{1}{3} \leq \frac{size(n.left) + 1}{size(n.right) + 1} \leq 3$$

or

$$\frac{1}{3} \leq \frac{weight(n.left)}{weight(n.right)} \leq 3$$

where $weight(n) = size(n) + 1$

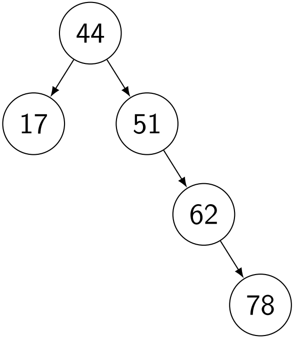
Equivalently,

$$weight(n.left) \leq weight(n.right) \times 3$$
$$weight(n.right) \leq weight(n.left) \times 3$$

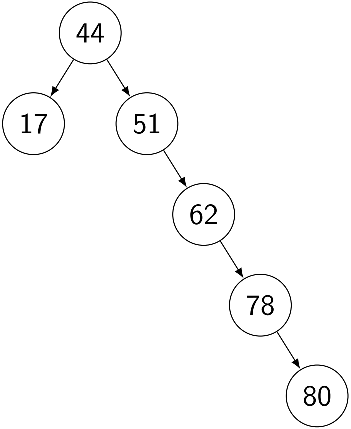
Q. How should we augment the tree?

A. Add a `size` field to each node.

WBT example

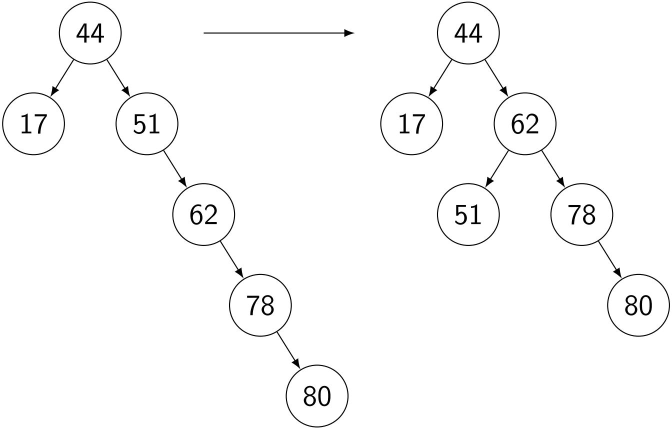


balanced



unbalanced: node 51

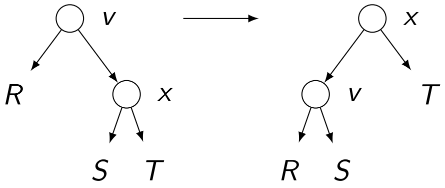
WBT rebalance



Rotations again!

WBT rebalance

Case 1: v is right-heavy; single counter-clockwise rotation works

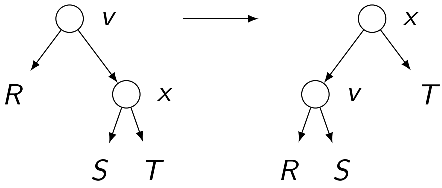


Q. When exactly is v right heavy?

A. $weight(x) > weight(R) \times 3$, i.e.
$weight(v.right) > weight(v.left) \times 3$

WBT rebalance

Case 1: v is right-heavy; single counter-clockwise rotation works



Q. For a single rotation to work, what should be true about x?

A. $weight(S) < weight(T) \times 2$, i.e.
$weight(v.right.left) < weight(v.right.right) \times 2$

WBT rebalance

Show why $weight(x.left) < weight(x.right) \times 2$ is a sufficient condition.

Let $r = size(R)$, $s = size(S)$, $t = size(T)$ at the time of the rotation.
v is right-heavy, so either

- a node was added to x to cause imbalance, or
- a node was removed from R to cause imbalance.

Assumptions:

$$s + 1 < 2(t + 1) \qquad \text{assumption } (1)$$

$$3(r + 1) < s + t + 2 \qquad v \text{ is right-heavy } (2)$$

Before addition, we had a WBT:

$$r + 1 \leq 3(s + t + 1) \text{ and } s + t + 1 \leq 3(r + 1) \qquad v \text{ was balanced } (3)$$

$$t \leq 3(s + 1) \text{ and } s \leq 3(t + 1) \qquad x \text{ was balanced } (4)$$

Show that after addition + rotation, we have a WBT:

$$r + s + 2 \leq 3(t + 1) \text{ and } t + 1 \leq 3(r + s + 2) \qquad x \text{ is balabced}$$

$$r + 1 \leq 3(s + 1) \text{ and } s + 1 \leq 3(r + 1) \qquad v \text{ is balabced}$$

WBT rebalance

Show why $weight(x.left) < weight(x.right) \times 2$ is a sufficient condition.

Let $r = size(R)$, $s = size(S)$, $t = size(T)$ at the time of the rotation.
v is right-heavy, so either

- a node was added to x to cause imbalance, or
- a node was removed from R to cause imbalance.

Assumptions:

$$s + 1 < 2(t + 1) \qquad \text{assumption } (1)$$

$$3(r + 1) < s + t + 2 \qquad v \text{ is right-heavy } (2)$$

Before removal, we had a WBT:

$$r + 2 \leq 3(s + t + 2) \text{ and } s + t + 2 \leq 3(r + 2) \qquad v \text{ was balanced } (3)$$

$$s + 1 \leq 3(t + 1) \text{ and } t + 1 \leq 3(s + 1) \qquad x \text{ was balanced } (4)$$

Show that after removal + rotation, we have a WBT:

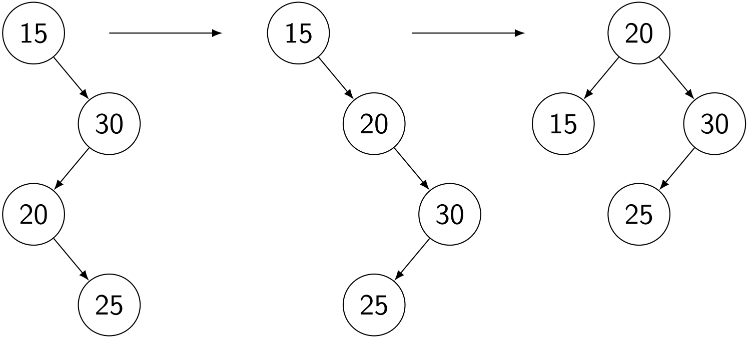
$$r + s + 2 \leq 3(t + 1) \text{ and } t + 1 \leq 3(r + s + 2) \qquad x \text{ is balabced}$$

$$r + 1 \leq 3(s + 1) \text{ and } s + 1 \leq 3(r + 1) \qquad v \text{ is balabced}$$

WBT rebalance

What if

- $weight(v.right) > weight(v.left) \times 3$ and
- $weight(v.right.left) \geq weight(v.right.right) \times 2$?

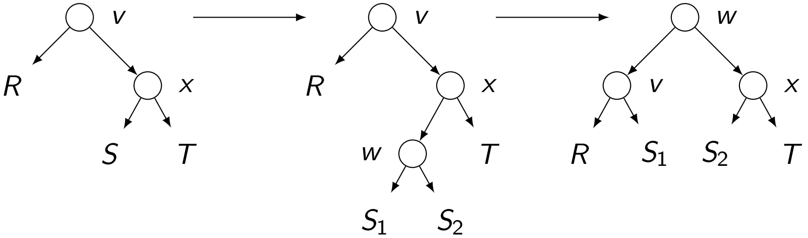


Double rotation.

WBT rebalance

Case 2: v is right-heavy; need a double rotation: clockwise then counter-clockwise

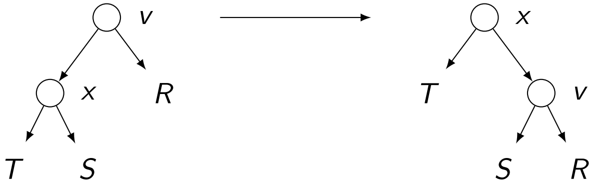
- $weight(x) > weight(R) \times 3$
- $weight(S) \geq weight(T) \times 2$



- S was too big: we split it
- convince yourself that v, x, and w are balanced (even longer, but not more complex, proof)

WBT rebalance

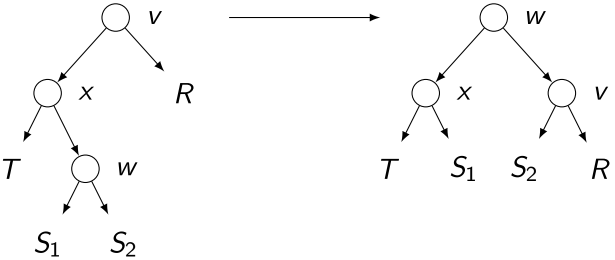
Case 3: v is left-heavy; single clockwise rotation works



- $weight(v.left) > weight(v.right) \times 3$ and
- $weight(x.right) < weight(x.left) \times 2$
- argument is symmetric to Case 1

WBT rebalance

Case 4: v is left-heavy; need a double rotation: counter-clockwise then clockwise



- $weight(v.left) > weight(v.right) \times 3$ and
- $weight(x.right) \geq weight(x.left) \times 2$
- argument is symmetric to Case 2

WBT rebalance

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

```
if weight(v.right) > weight(v.left) * 3:
  let x = v.right
  if weight(x.left) < weight(x.right) * 2:
    single rotation: counter-clockwise
  else:
    double rotation: clockwise then counter-clockwise
else if weight(v.left) > weight(v.right) * 3:
  let x = v.left
  if weight(x.right) < weight(x.left) * 2:
    single rotation: clockwise
  else:
    double rotation: counter-clockwise then clockwise
else:
  no rotation
```

WBT insert

Assuming the height of the weight-balanced tree is $\mathcal{O}(\log n)$,

1. insert as in BST
2. check and fix balance, update size from parent of new node up to root

- complexity: $\Theta(\log n)$

WBT delete

Assuming the height of the weight-balanced tree is $\mathcal{O}(\log n)$,

1. find which node has the key, call it w
   - complexity: $\Theta(\log n)$ time
2. if w is a leaf, remove it
   - complexity: $\Theta(1)$ time
3. if w has one child, w's parent adopts that child
   - complexity: $\Theta(1)$ time
4. else:
   - 4.1 go to successor node (complexity: $\Theta(\log n)$ time)
   - 4.2 replace key of node with successor key
     - complexity: $\Theta(1)$ time
   - 4.3 successor's parent adopts successor's right child
     - complexity: $\Theta(1)$ time
5. from parent node to root: check and fix balance, update size
   - complexity: $\Theta(\log n)$ time

WBT union

Recall the algorithm to compute union of AVL trees T_1 and T_2:

```
if T_1 == nil:
  return T_2
if T_2 == nil:
  return T_1

k = T_2.key
(L, R) = split(T_1, k)
L' = union(L, T_2.left)
R' = union(R, T_2.right)
return join(L', k, R')
```

What needs to change for WBTs?

WBT union

Need to change the algorithm for *join*(L, k, G):

```
if height(L) - height(G) > 1:
  p = L
  while height(p.right) - height(G) > 1:
    p = p.right
  q = new node(key=k, left=p.right, right=G)
  p.right = q
  rebalance and update heights at p up to the root
  return L
elif height(G) - height(L) > 1:
  ... symmetrical ...
else:
  return new node(key=k, left=L, right=G)
```

WBT union

New algorithm for *join*(L, k, G):

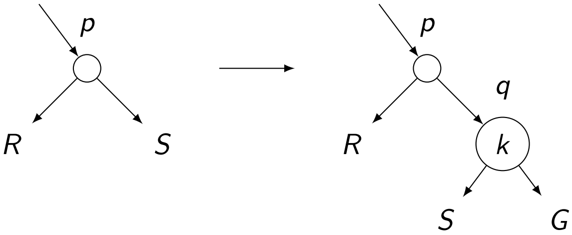
```
if weight(L) > weight(G) * 3:
  p = L
  while weight(p.right) > weight(G) * 3:
    p = p.right
  q = new node(key=k, left=p.right, right=G)
  p.right = q
  rebalance and update sizes at p up to the root
  return L
elif weight(G) > weight(L) * 3:
  ... symmetrical ...
else:
  return new node(key=k, left=L, right=G)
```

WBT union — $join(L, k, G)$

In L, keep going to the right until find node p:

- $weight(p) > weight(G) \times 3$
- $weight(p.right) \leq weight(G) \times 3$

Create new node q with key k, left child $p.right$, right child G.
This node is balanced. (Why?)



p and ancestors may need rebalancing.