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?

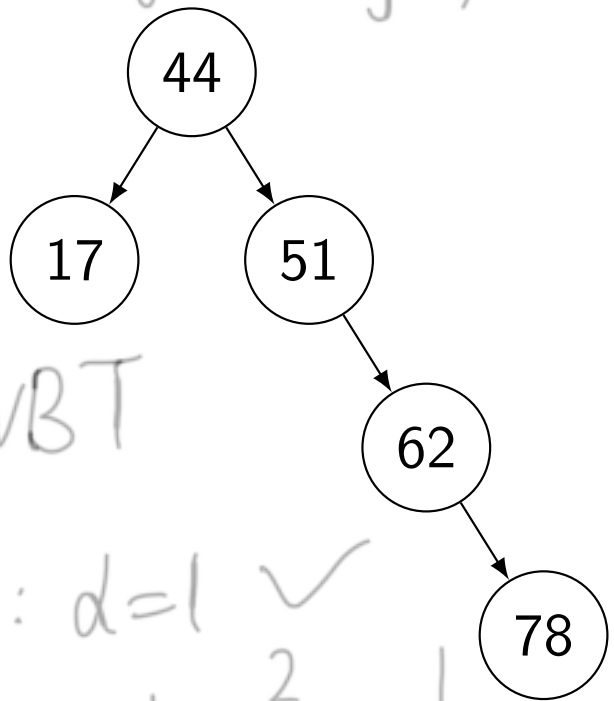
A. size(n) = number of nodes

Note:
weight(n)
= weight(n.left) + weight(n.right)
= size(n.left) + 1
+ size(n.right) + 1
= size(n.left) + size(n.right) + 2

weight:
number of nodes
(including itself)

WBT example

$\alpha = \frac{\text{weight}(n.\text{left})}{\text{weight}(n.\text{right})}$



WBT

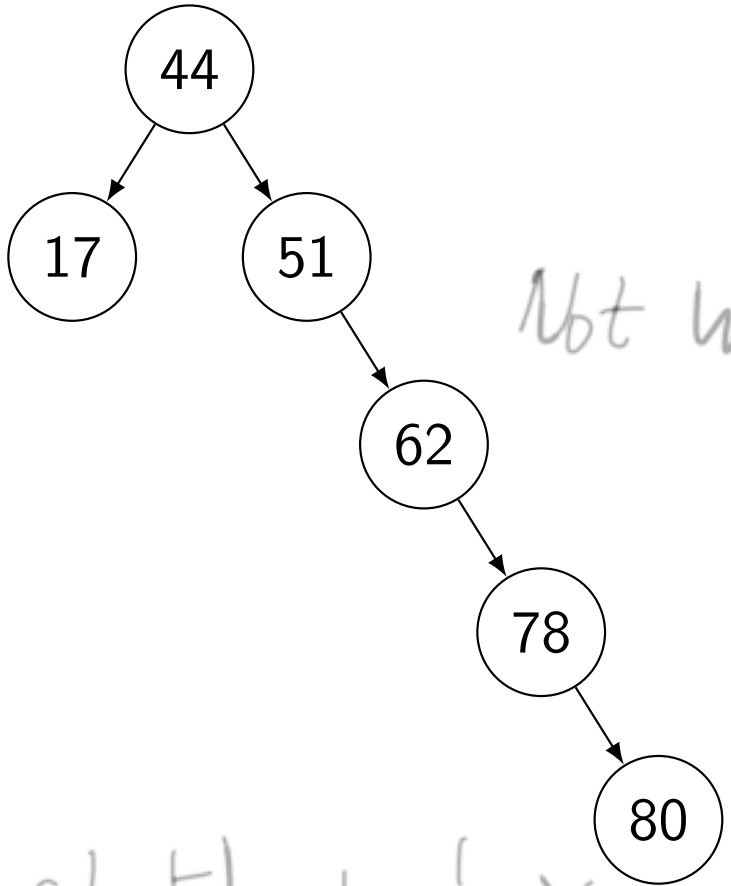
at 17: $\alpha = 1$ ✓

at 44: $\alpha = \frac{2}{4} = \frac{1}{2}$ ✓

at 51: $\alpha = \frac{1}{3}$ ✓

at 62: $\alpha = \frac{1}{2}$ ✓

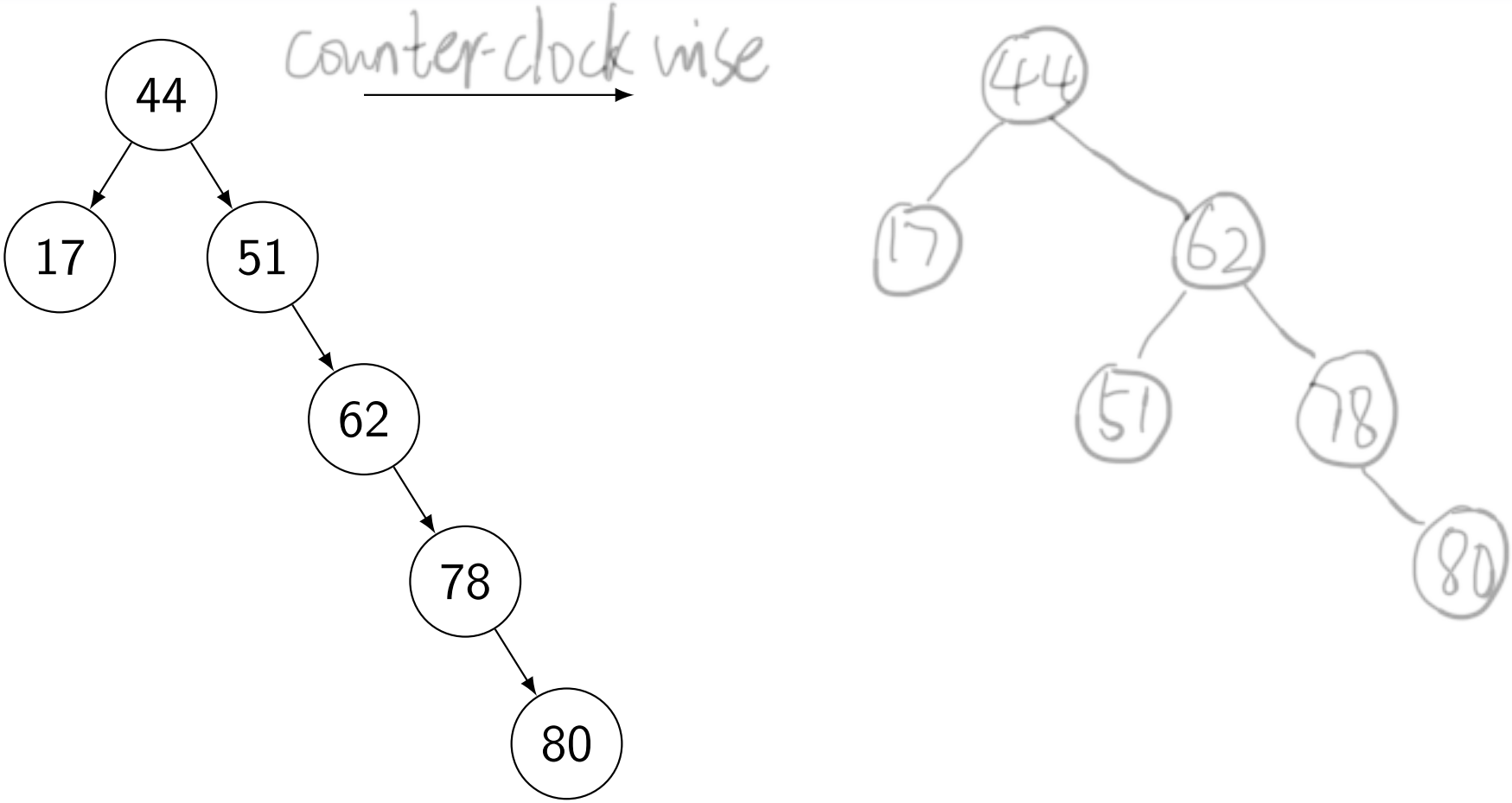
at 78: $\alpha = 1$ ✓



Not WBT

at 51: $\alpha = \frac{1}{4}$ X

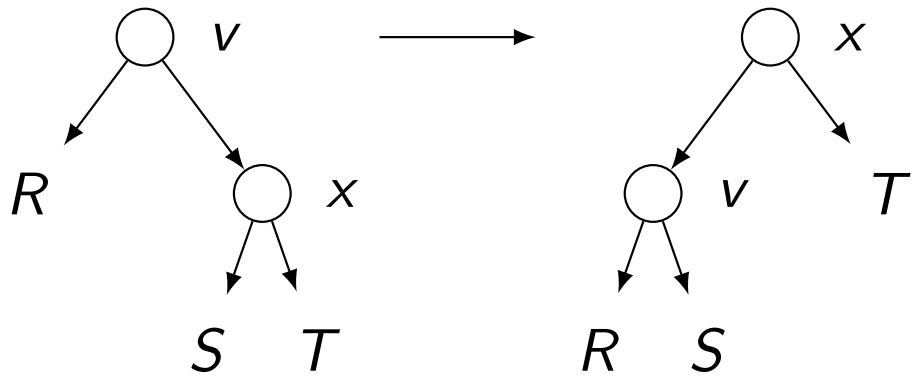
WBT rebalance



Rotations again!

WBT rebalance

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



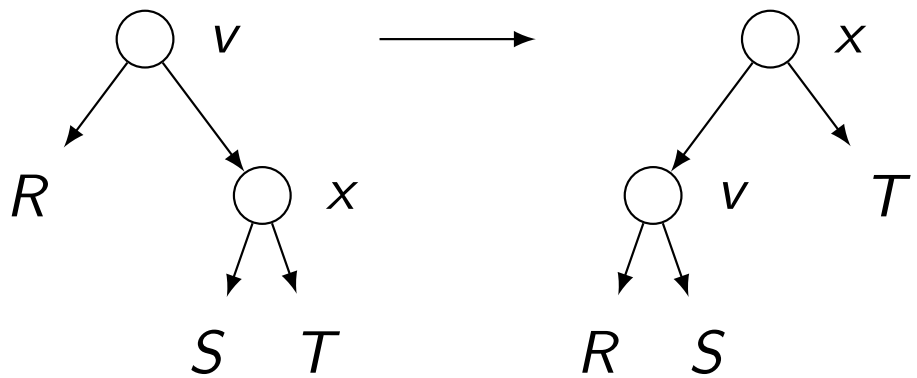
Q. When exactly is v right heavy?

A.

$$\frac{\text{weight}(v.\text{left})}{\text{weight}(v.\text{right})} \simeq \frac{\text{weight}(R)}{\text{weight}(x)} < \frac{1}{3}$$

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.

$$\frac{\text{weight}(x.\text{left})}{\text{weight}(x.\text{right})} = \frac{\text{weight}(S)}{\text{weight}(T)} < 2$$

WBT rebalance

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

Given $s = size(S)$, $r = size(R)$, $t = size(T)$

$$\frac{weight(R)}{weight(x)} = \frac{size(R)+1}{size(x.left)+size(x.right)+2} = \frac{r+1}{s+t+2} < \frac{1}{3} \quad (1)$$

$$\frac{weight(S)}{weight(T)} = \frac{size(S)+1}{size(t)+1} = \frac{s+1}{t+1} < 2 \quad (2)$$

WTS:

(a) $\frac{1}{3} \leq \frac{weight(R)}{weight(S)} = \frac{r+1}{s+1} \leq 3$

(b) $\frac{1}{3} \leq \frac{weight(v)}{weight(T)} = \frac{r+s+2}{t+1} \leq 3$

[after rebalance satisfy WBT property]

To make v right-heavy (unbalanced):
1. added a node to X
2. removed a node in R

WBT rebalance

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

Before 1:

$$\frac{1}{3} \le \frac{weight(v.left)}{weight(v.right)} = \frac{r+1}{s+t+1} \le 3 \quad (3)$$

$$\frac{1}{3} \le \frac{weight(x.left)}{weight(x.right)} = \frac{s+1}{t} \le 3 \quad (4)$$

insert node in T

Intuition:

$(s+t+2) = (s+t+1) + 1$ ← insert

or $\frac{1}{3} \le \frac{s}{t+1} \le 3$

insert node in S

Prove: (1) (2) (3) (4) → (a) (b)

Before 2:

$$\frac{1}{3} \leq \frac{\text{weight}(v.\text{left})}{\text{weight}(v.\text{right})} = \frac{r+2}{s+t+2} \leq 3 \quad (5)$$

$$\frac{1}{3} \leq \frac{\text{weight}(x.\text{left})}{\text{weight}(x.\text{right})} = \frac{s+1}{t+1} \leq 3 \quad (6)$$

Intuition:

$(r+1) - (r+2) = -1$

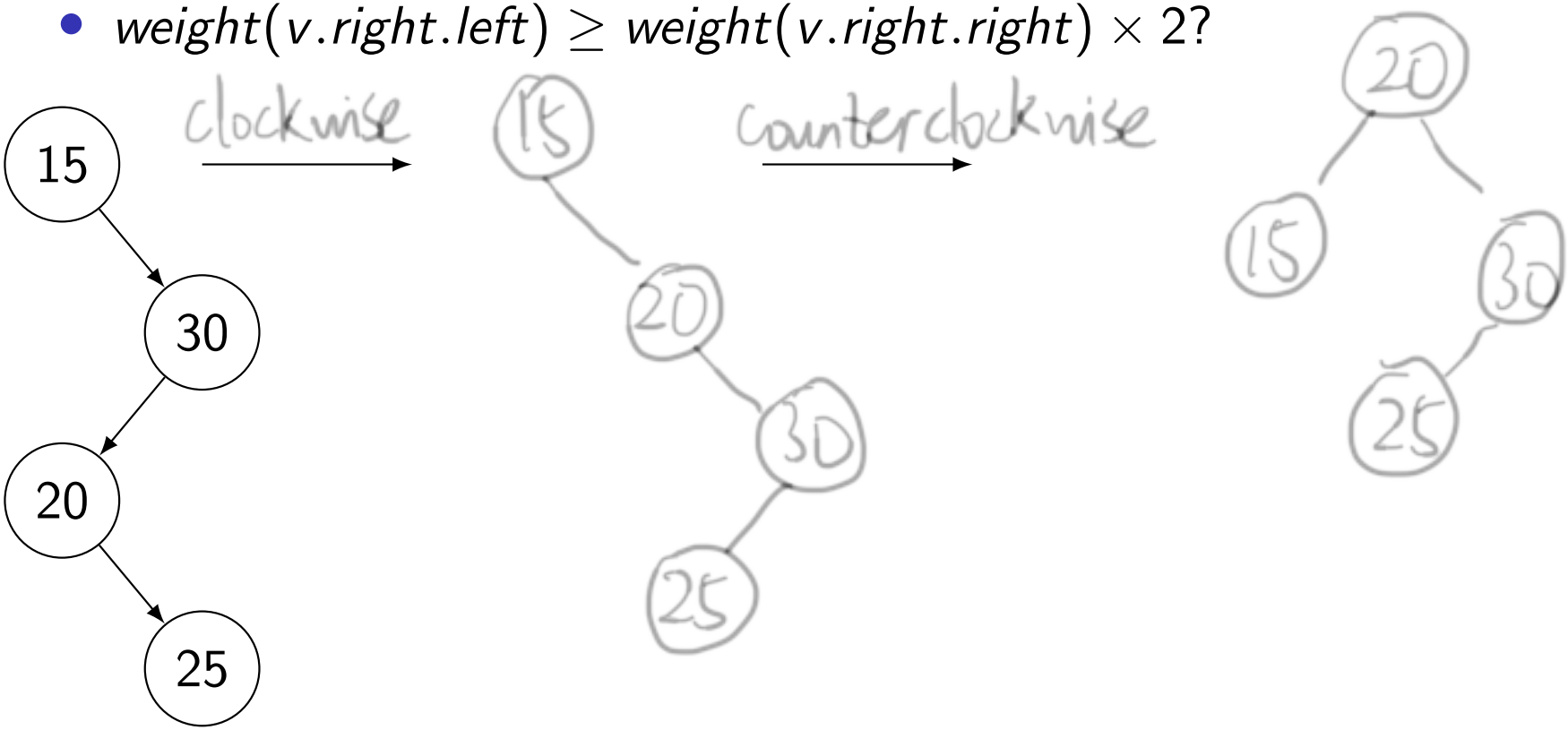
↑ removal

Prove: (1)(2)(5)(6) → (a)(b)

WBT rebalance

What if

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

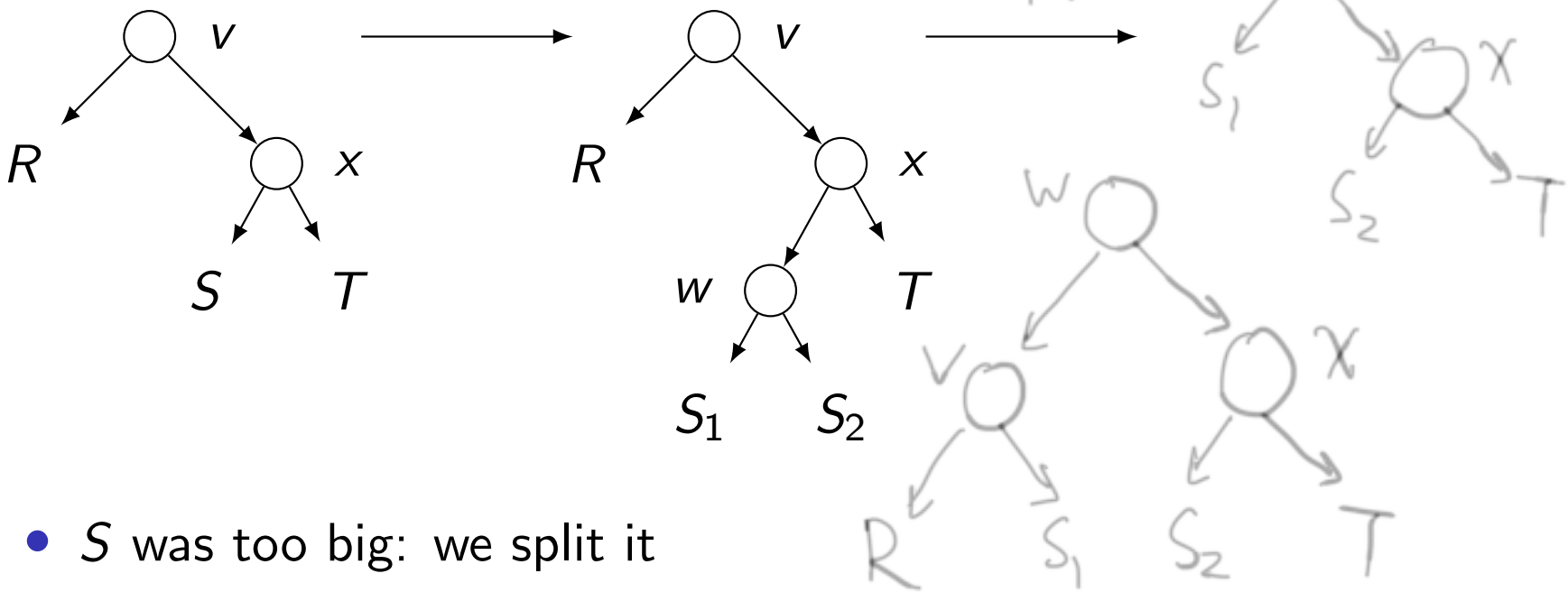


Double rotation.

WBT rebalance

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

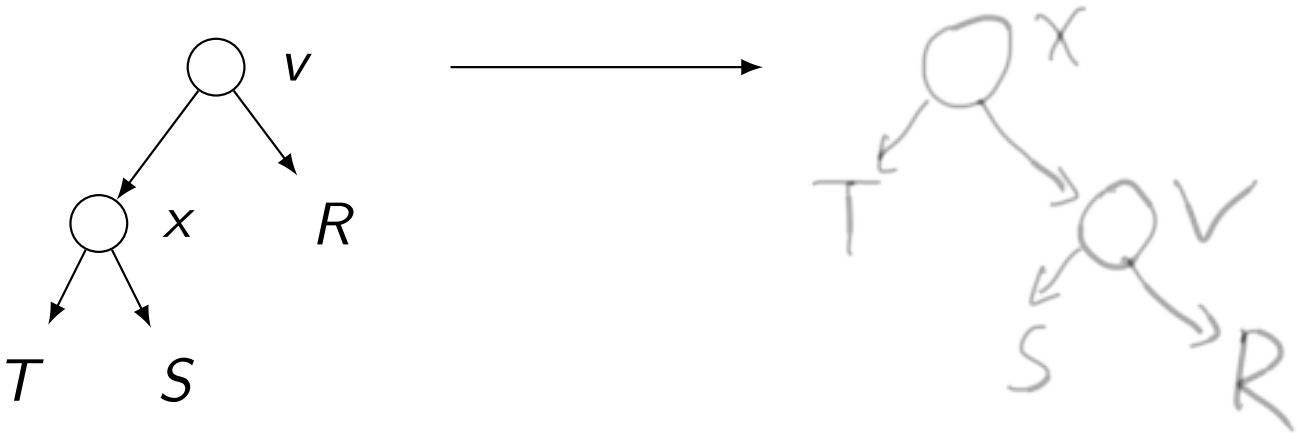
- $weight(x) > weight(R) \times 3$
- $weight(S) \geq weight(T) \times 2$



- S was too big: we split it
- convince yourself that v, x, and w are balanced (even longer proof)

WBT rebalance

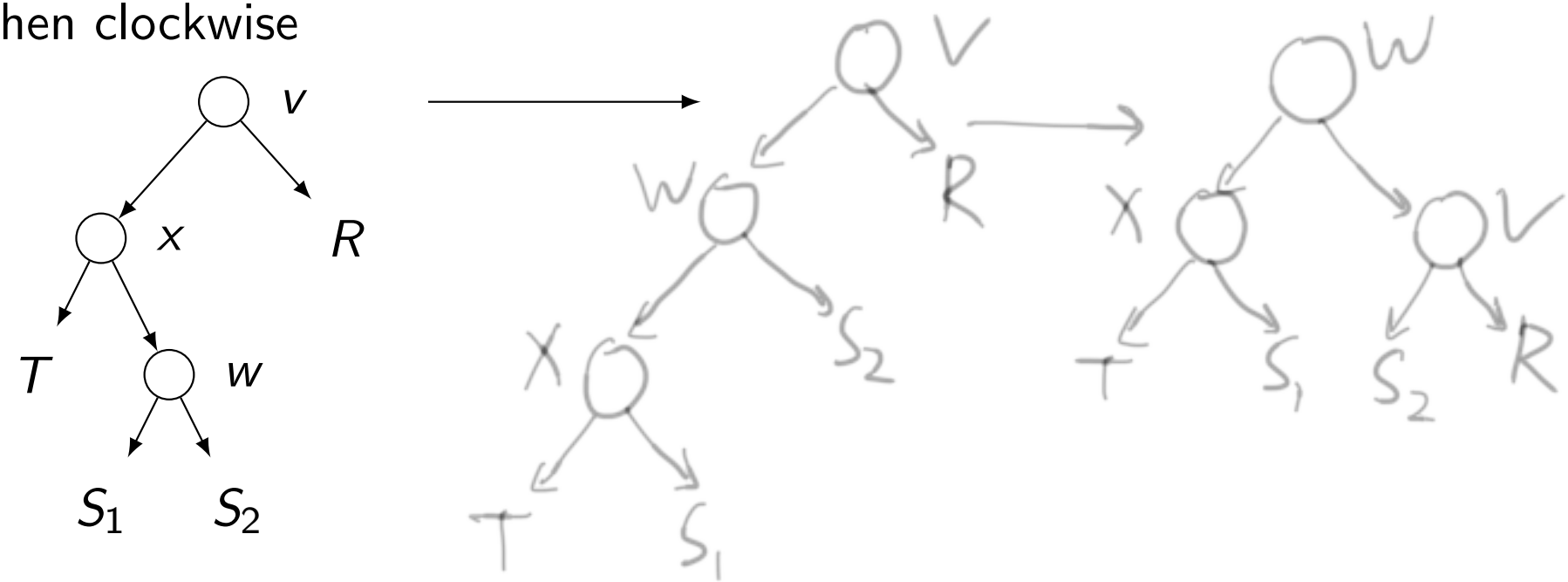
Case 3: v is left-heavy; single clockwise rotation works



- $weight(v.left) > weight(v.right) \times 3$ 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: $O(\log_2 n)$

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: $\Theta(\log_2 n)$
2. if w is a leaf, remove it
   - complexity: $\Theta(1)$
3. if w has one child, w's parent adopts that child
   - complexity: $\Theta(1)$
4. else:
   - 4.1 go to successor node (complexity: $\Theta(\log_2 n)$)
   - 4.2 replace key of node with successor key
     - complexity: $\Theta(1)$
   - 4.3 successor's parent adopts successor's right child
     - complexity: $\Theta(1)$
5. from parent node to root: check and fix balance, update size
   - complexity: $\Theta(\log_2 n)$

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)
```

use weight instead of height

WBT union

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

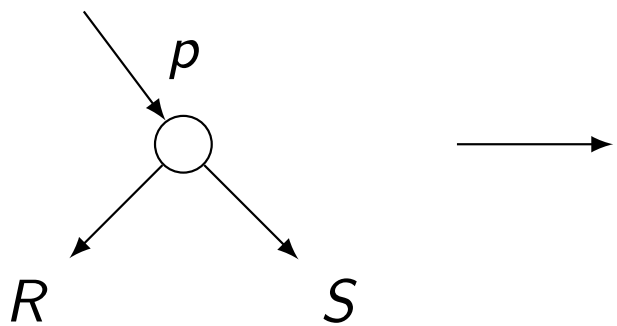
```
if weight(L) > weight(G) * 3:
  p = L
  while weight(p.right) > weight(G) * 3:
    p = p.right
  q = new node(key=k, left=p.right, right=G)
  p.right = q
  rebalance and update sizes at p up to the root
  return L
elif weight(G) > weight(L) * 3:
  ... symmetrical ...
else:
  return new node(key=k,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

Height of the WBT

Claim:

$$height(T) \in \Theta[\log_2(size(T))]$$

$$height(T) \leq \log(size(T) + 1)/\log(4/3)$$

for all weight-balanced trees T.

Proof. By induction on size of the tree. [Complete induction]

Base. $n=0=size(T)$, $height(T)=0$ ✓

IH. Suppose $\forall k \in \mathbb{N}, 0 \leq k < n, height(T') \leq \log(k+1)/\log(4/3)$ where $size(T') = k$.

Show. $height(T) \leq \log(n+1)/\log(4/3)$ where $size(T) = n$.

$h(T)$

Height of the WBT

Show. $height(T) \leq \log(n+1)/\log(4/3)$ where $size(T) = n$.

Consider $l = size(T.left)$, $r = size(T.right)$

By IH, $0 \leq l < n$, $0 \leq r < n$

$$\Rightarrow \begin{cases} h(L) \leq \dfrac{\log(l+1)}{\log(\frac{4}{3})} \\ h(R) \leq \dfrac{\log(r+1)}{\log(\frac{4}{3})} \end{cases}$$

$h(T) = \max(h(T.left), h(T.right)) + 1$

$\xrightarrow[h(T.left) < h(T.right)]{\text{WLOG}}$ $h(T) = h(T.right) + 1$

$\Rightarrow$ WTS:

$$h(T) \leq \frac{\log(r+1)}{\log(\frac{4}{3})} + 1$$

$$= \frac{\log[\frac{4}{3}r + \frac{4}{3}]}{\log(\frac{4}{3})}$$

Height of the WBT

Show. $height(T) \leq \log(n+1)/\log(4/3)$ where $size(T) = n$.

By WBT def:

$$\frac{1}{3} \leq \frac{weight(T.left)}{weight(T.right)} = \frac{l+1}{r+1} \leq 3 \Rightarrow l+1 \geq \frac{1}{3}(r+1)$$

$$n = size(T) = l+r+1 \geq \frac{4}{3}r + \frac{1}{3}$$

$$\Rightarrow h(T) \leq \frac{\log(\frac{4}{3}r + \frac{4}{3})}{\log(\frac{4}{3})} = \frac{\log(n+1)}{\log(\frac{4}{3})}$$