Ordered Dictionary & Binary Search Tree

Finite map from keys to values, assuming keys are comparable ($<$, $=$, $>$).

- insert(k, v)
- lookup(k) aka find: the associated value if any
- delete(k)
- some more later

(Ordered set: No values; just keys, call them elements).

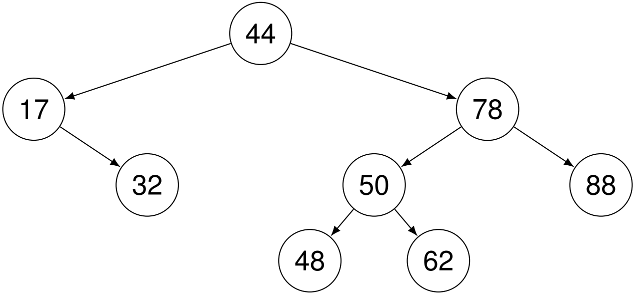
We will use a special kind of binary search trees called “AVL trees”. It prevents pathological trees, ensures $O(\lg n)$ worst-case time.

(Binary tree height in $\Omega(\lg n)$, so actually $\Theta(\lg n)$ time.)

AVL Tree: Definition

AVL trees are one way to ensure $\Theta(\lg n)$ tree height.
(Georgy Adelson-Velsky and Evgenii Landis)

- is a binary search tree
- at every node: subtree heights differ by at most 1.



(Keys shown, values omitted.)

Binary Search Tree: lookup

```
lookup($k$):
$n$ := $root$
while $n \neq null$:
    if $k < n.key$ then $n := n.left$
    else if $k > n.key$ then $n := n.right$
    else return $n.value$
return $null$.
```

AVL Tree: insert

insert(k, v) begins like lookup,
expects to **not** find k (if found, change value to v),
but now it knows where to put the new node.

Add the new node there. The AVL property may break. Now fix it:

There are “rotations” we do along the path from insertion point to root, restoring the AVL property.

And want to manipulate only that path. Why?

AVL Tree: insert

insert(k, v) begins like lookup,
expects to **not** find k (if found, change value to v),
but now it knows where to put the new node.

Add the new node there. The AVL property may break. Now fix it:

There are “rotations” we do along the path from insertion point to root, restoring the AVL property.

And want to manipulate only that path. Why?
Because time budget is $O(\lg n)$.

Rebalancing: Overview

To check and fix a node v:

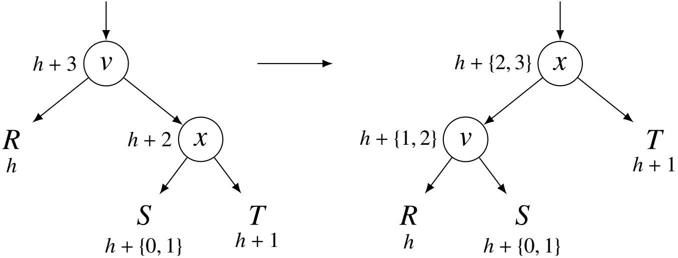
```
if $height(v.left) + 1 < height(v.right)$
    (the right is taller by 2)
    re-balance at $v$ (two further subcases)
else if $height(v.left) > height(v.right) + 1$
    (the left is taller by 2)
    re-balance at $v$ (two further subcases)
else
    nothing to fix for $v$
```

Do this for each node on the path from new node back to root.
⇒ When processing v, descendents are already fixed.

Rebalance (Right Side) Subcase 1 of 2

Single-Rotation Counterclockwise

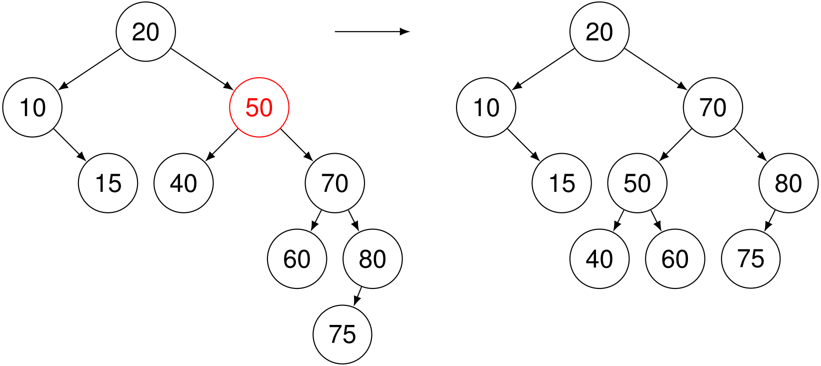
If $height(v.left) + 1 < height(v.right)$:
  Let $x = v.right$ (Why does it exist?)
  If $height(x.left) \leq height(x.right)$:



Why can we assume x is balanced? Answer on last slide.

Exercise: Why is the outcome a binary search tree?

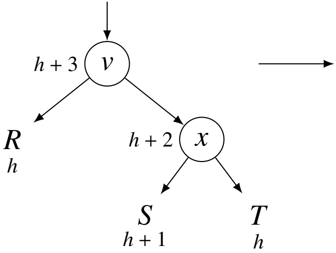
Rebalance (Right Side) Subcase 1 of 2 Example



Rebalance (Right Side) Subcase 2 of 2

Why Single-Rotation No Workie

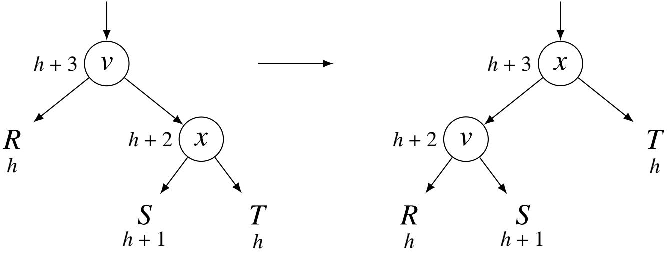
```
If $height(v.left) + 1 < height(v.right)$:
    Let $x = v.right$
    If $height(x.left) > height(x.right)$:
```



Rebalance (Right Side) Subcase 2 of 2

Why Single-Rotation No Workie

```
If $height(v.left) + 1 < height(v.right)$:
    Let $x = v.right$
    If $height(x.left) > height(x.right)$:
```

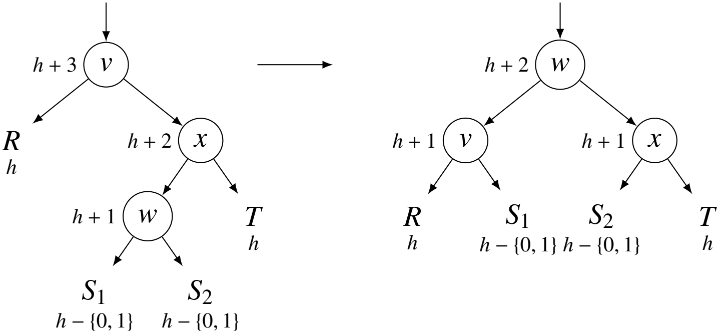


Result still unbalanced. Solution on next slide.

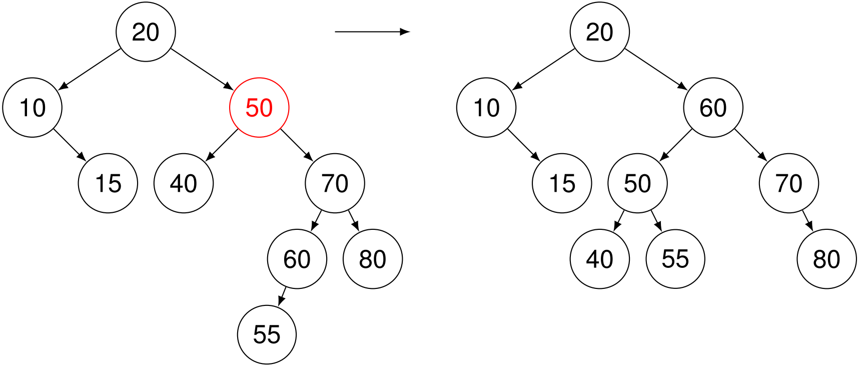
Rebalance (Right Side) Subcase 2 of 2

Double-Rotation Clockwise Then Counterclockwise

```
If $height(v.left) + 1 < height(v.right)$:
  Let $x = v.right$
  If $height(x.left) > height(x.right)$:
    Let $w = x.left$:
```



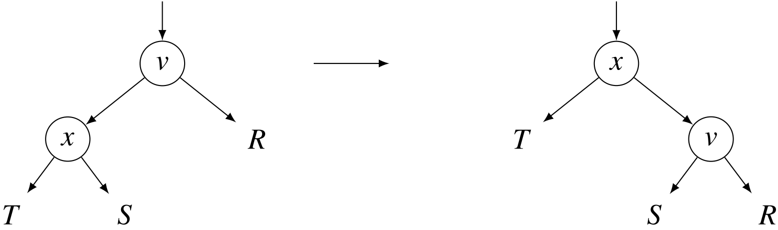
Rebalance (Right Side) Subcase 2 of 2 Example



Rebalance (Left Side) Subcase 1 of 2

Single-Rotation Clockwise

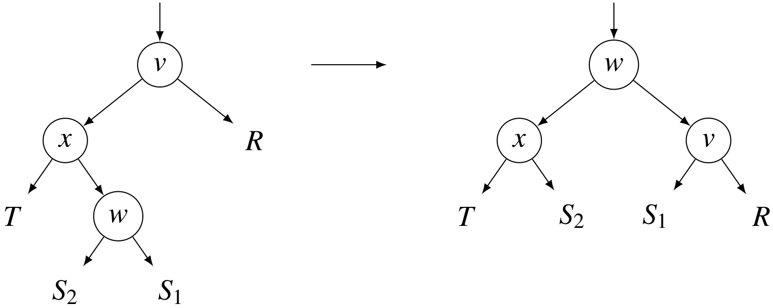
```
If $height(v.left) > height(v.right) + 1$:
    Let $x = v.left$
    If $height(x.left) \geq height(x.right)$:
```



Rebalance (Left Side) Subcase 2 of 2

Double-Rotation Counterclockwise Then Clockwise

```
If $height(v.left) > height(v.right) + 1$:
    Let $x = v.left$
    If $height(x.left) < height(x.right)$:
        Let $w = x.right$:
```



Rebalancing: Summary

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

```
if $height(v.left) + 1 < height(v.right)$
   let $x = v.right$
   if $height(x.left) \leq height(x.right)$
      single-rotation ccw
   else
      double-rotation cw then ccw
else if $height(v.left) > height(v.right) + 1$
   let $x = v.left$
   if $height(x.left) \geq height(x.right)$
      single-rotation cw
   else
      double-rotation ccw then cw
else
   no rotation
```

Height Comparison And Update

Two alternatives: Cache height or cache difference.

Cache height (more bits):

- Each node has field h for known height of *self*.
- Query: $height(v) = (v = null \ ? \ -1 : v.h)$
- Update: Set children's before parent's, so simply: $v.h := 1 + \max(height(v.left), height(v.right))$

Cache difference (a.k.a. balance factor, fewer bits):

- Each node has field BF for known difference of *children*.
- Update: See Hadzilacos's notes.

Either way, update at: New node and ancestors (later ancestors of deleted node), nodes affected by rotations.

Only Path of Ancestors Needs Fixing

Why is it enough to just fix the path from new node back to root?

Only Path of Ancestors Needs Fixing

Why is it enough to just fix the path from new node back to root?

Say a node v is visited when finding where to add new node.
Say it is decided to be v's right subtree.

⇒ v's left subtree won't change, doesn't even need to check.

(v itself needs checking later because right subtree will change, but it *is* on the path.)

Only Path of Ancestors Needs Fixing

Why is it enough to just fix the path from new node back to root?

Say a node v is visited when finding where to add new node.
Say it is decided to be v's right subtree.

$\Rightarrow$ v's left subtree won't change, doesn't even need to check.

(v itself needs checking later because right subtree will change, but it *is* on the path.)

Similar story for updating heights.

Similar story for deleting a node in later slides.

Summary of AVL Tree Insertion

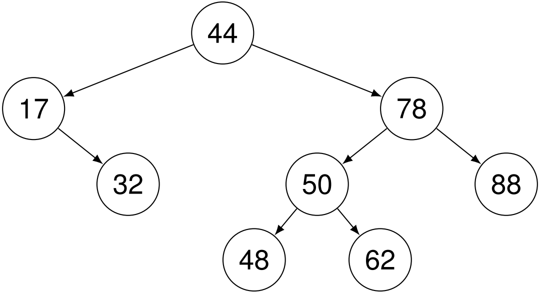
Later we will see why tree height is in $O(\lg n)$.

With that in mind, AVL tree insertion:

1. find which node to become parent of new node [$\Theta(\lg n)$ time]
2. put new node there [$\Theta(1)$ time]
3. from that parent to root (bottom-up): check and fix balance, update height [$\Theta(\lg n)$ nodes, $\Theta(1)$ time per node]

Total $\Theta(\lg n)$ time.

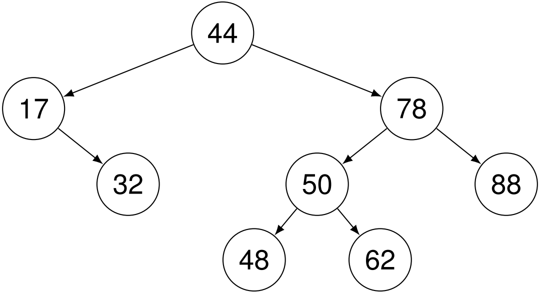
Delete: Easy Case



Delete 32, or 48, or 62, or 88.

If the node has no children, just unlink from parent.
(Then update heights of ancestors, rebalance. . .)

Delete: Slightly Harder Case



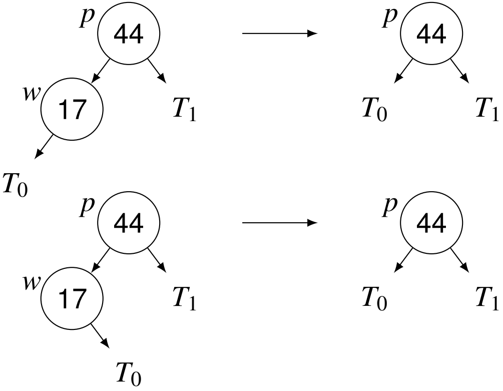
Delete 17. (Note that 32 is a good replacement.)

If the node has at most one child, just link parent to that child.
(Then update heights of ancestors, rebalance. . .)

This generalizes the easy case.

Delete: Slightly Harder Case, Generally

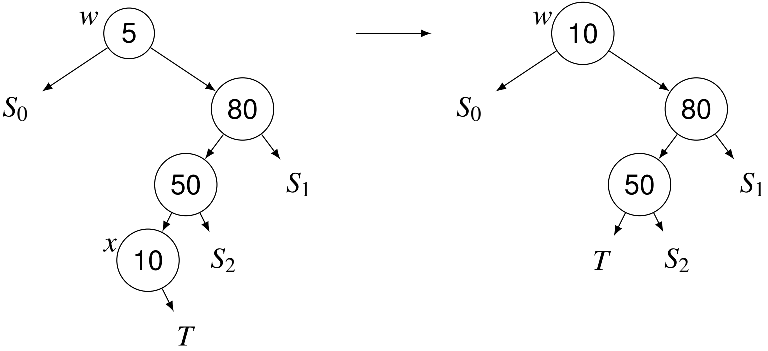
Prune w. T_0 may be empty. p and ancestors need height updates and rebalancing.



There are two more mirror images.

Delete: Hard Case

Delete 5. Call the node w. Both children non-empty.

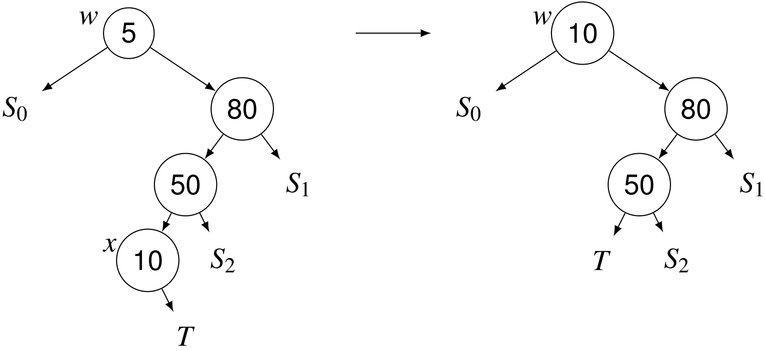


Find successor: Go right once, go left all the way, call it x. Replace $w.key$ by $x.key$. x's parent adopts x's right child T.

Rebalancing and height updates start from:

Delete: Hard Case

Delete 5. Call the node w. Both children non-empty.

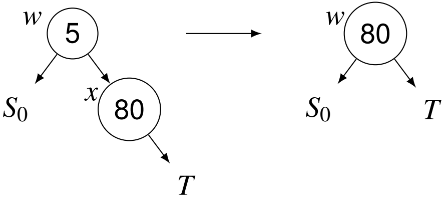


Find successor: Go right once, go left all the way, call it x. Replace $w.key$ by $x.key$. x's parent adopts x's right child T.

Rebalancing and height updates start from: x's old parent.

Delete: Hard Case, Degenerate

Delete 5. Call the node w. Both children non-empty.



Go right. Go left all the way—can't! That's already x. x's parent is w.

Still, replace $w.key$ by $x.key$. w also adopts x's right child T.

Summary of AVL Tree Deletion

Next we will see why tree height is $O(\lg n)$.

1. find which node has the key, call it w [$\Theta(\lg n)$ time]
2. if at most one child, $w.parent$ adopts that child [$\Theta(1)$ time]
3. else:
    - 3.1 go to successor x [$\Theta(\lg n)$ time]
    - 3.2 $w.key := x.key$ [$\Theta(1)$ time]
    - 3.3 $x.parent$ adopts $x.right$ [$\Theta(1)$ time]
4. from adopter to root (bottom-up): check and fix balance, update heights [$\Theta(\lg n)$ time]

Total $\Theta(\lg n)$ time.

AVL Tree Height

If there are n nodes, what is the maximum possible height?

$\Leftrightarrow$

If the height is h, what is the minimum possible number of nodes?

$$\begin{aligned} minsize(0) &= 1 \\ minsize(1) &= 2 \\ minsize(h+2) &= 1 + minsize(h+1) + minsize(h) \end{aligned}$$

Can prove by induction:

$$minsize(h) = fib(h+3) - 1$$

Golden ratio: $\phi = (\sqrt{5} + 1)/2 = 1.618\ldots$

$$minsize(h) = \frac{\phi^{h+3} - (1-\phi)^{h+3}}{\sqrt{5}} - 1$$

AVL Tree Height

$$
\begin{aligned}
n \geq \mathit{minsize}(h) &= \frac{\phi^{h+3}}{\sqrt{5}} - \frac{(1-\phi)^{h+3}}{\sqrt{5}} - 1 \\
&> \frac{\phi^{h+3}}{\sqrt{5}} - 1 - 1
\end{aligned}
$$

$$
\frac{\phi^{h+3}}{\sqrt{5}} - 2 < n
$$

$$
\begin{aligned}
h &< \frac{\lg(n+2)}{\lg \phi} + \frac{\sqrt{5}}{\lg \phi} + 3 \\
&= 1.44 \lg(n+2) + \text{constant} \\
&\in O(\lg n)
\end{aligned}
$$