

CSE 332

Data Structures & Parallelism

AVL Trees

Melissa Winstanley
Spring 2024

Class Updates

- **P1 due tomorrow night!**
 - No more than 2 late days can be used on this project
 - Optional: submit to the hidden test runner to get faster feedback on your P1 (before the TAs finish grading) - linked from the course website
- **Ex1 grades released**
 - Regrades: open after 48 hours, until 1 week from today
- **P2 released tomorrow**
 - You already know everything for the first checkpoint!

Balanced BST

Observation:

- BST: the shallower the better!
- For a BST with n nodes inserted in arbitrary order
 - Average height is $O(\log n)$ – see text for proof
 - Worst case height is $O(n)$
- Simple cases such as inserting in key order lead to the worst-case scenario

Solution: Require a Balance Condition that

1. ensures depth is always $O(\log n)$ – strong enough!
2. is easy to maintain – not too strong!

Potential Balance Conditions

1. Left and right subtrees of the root have equal number of nodes



2. Left and right subtrees of the root have equal height

Potential Balance Conditions

3. Left and right subtrees of every node have equal number of nodes



4. Left and right subtrees of every node have equal height



The AVL Balance Condition

Left and right subtrees of *every node* have heights differing by at most 1

Definition: $\text{balance}(\text{node}) = \text{height}(\text{node.left}) - \text{height}(\text{node.right})$

AVL property: **for every node x , $-1 \leq \text{balance}(x) \leq 1$**

- Ensures small depth
 - We will prove this by showing that an AVL tree of height h must have a number of nodes *exponential* in h
- Easy (well, efficient) to maintain
 - Using single and double rotations

The AVL Tree Data Structure

Structural properties

1. Binary tree property (0,1, or 2 children)
2. Heights of left and right subtrees of every node differ by at most 1

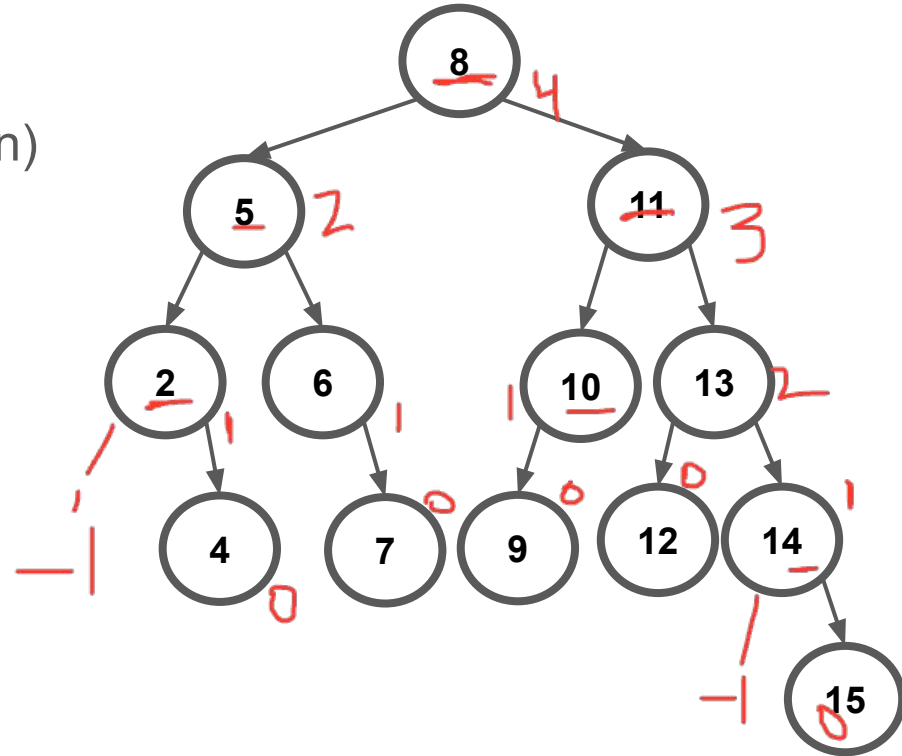
Result:

Worst case depth of any node is:

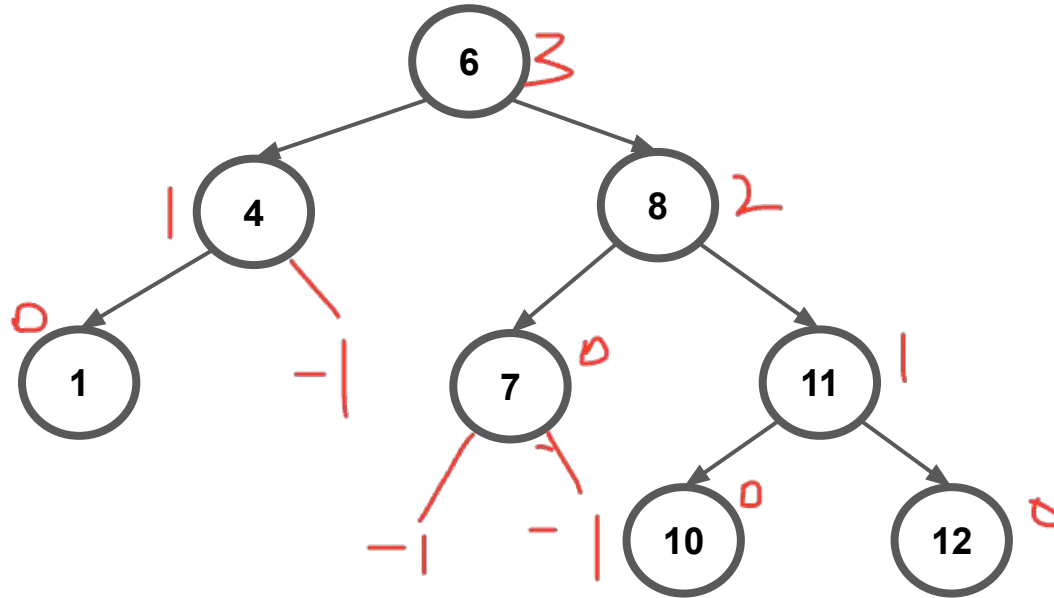
$$O(\log n)$$

Ordering property:

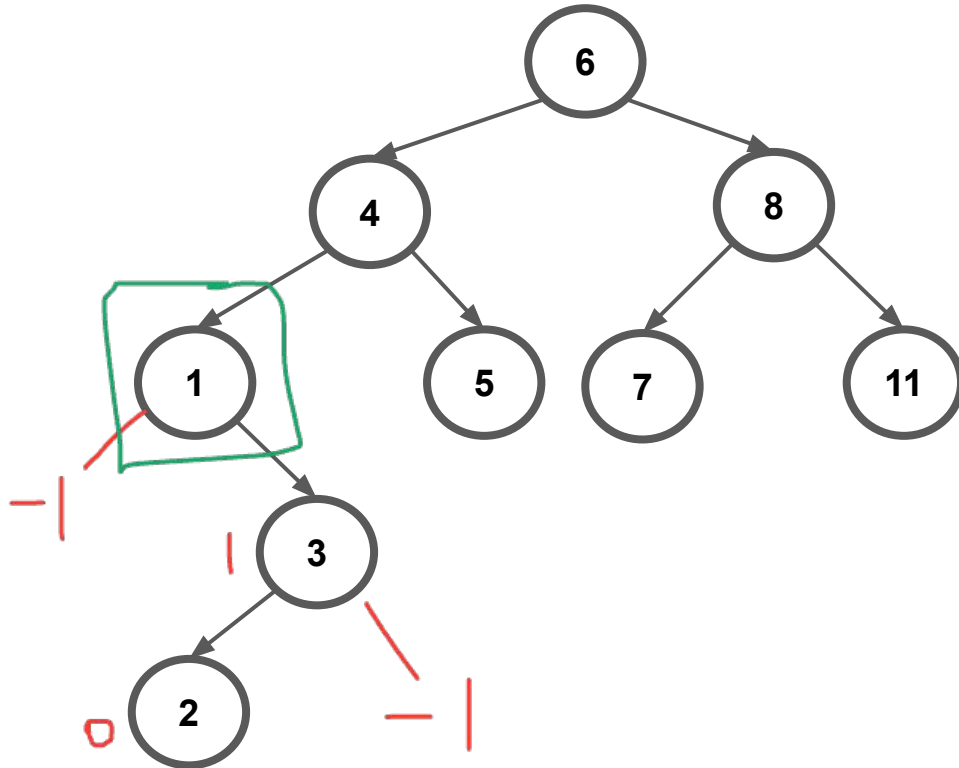
- Same as for BST



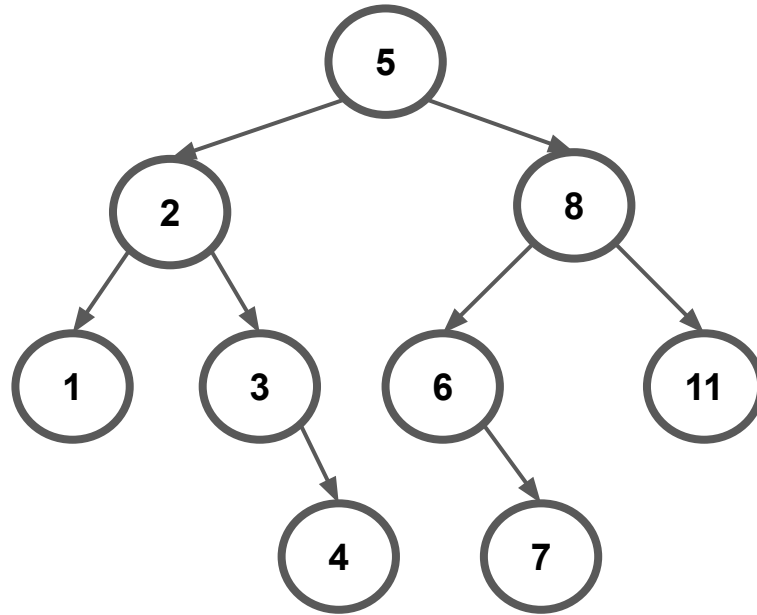
Ex1: An AVL tree?



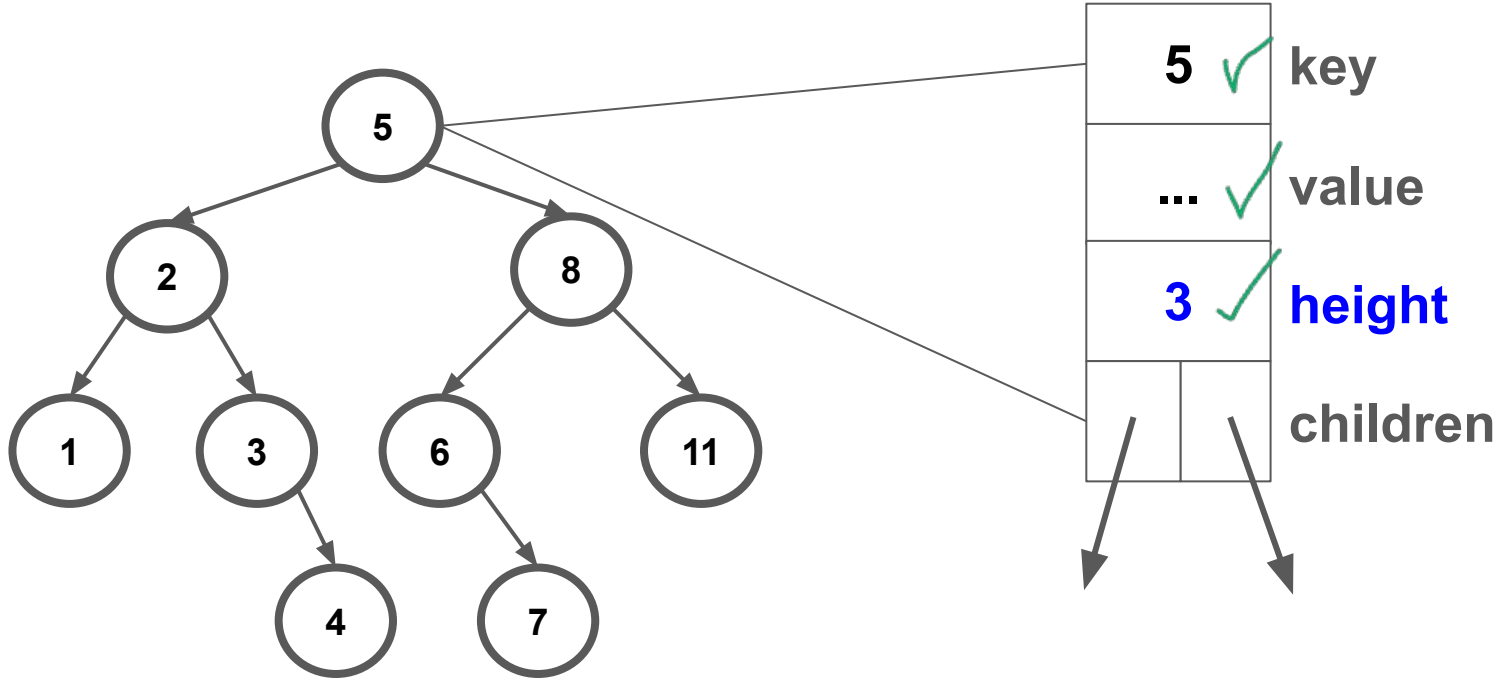
Ex2: An AVL tree?



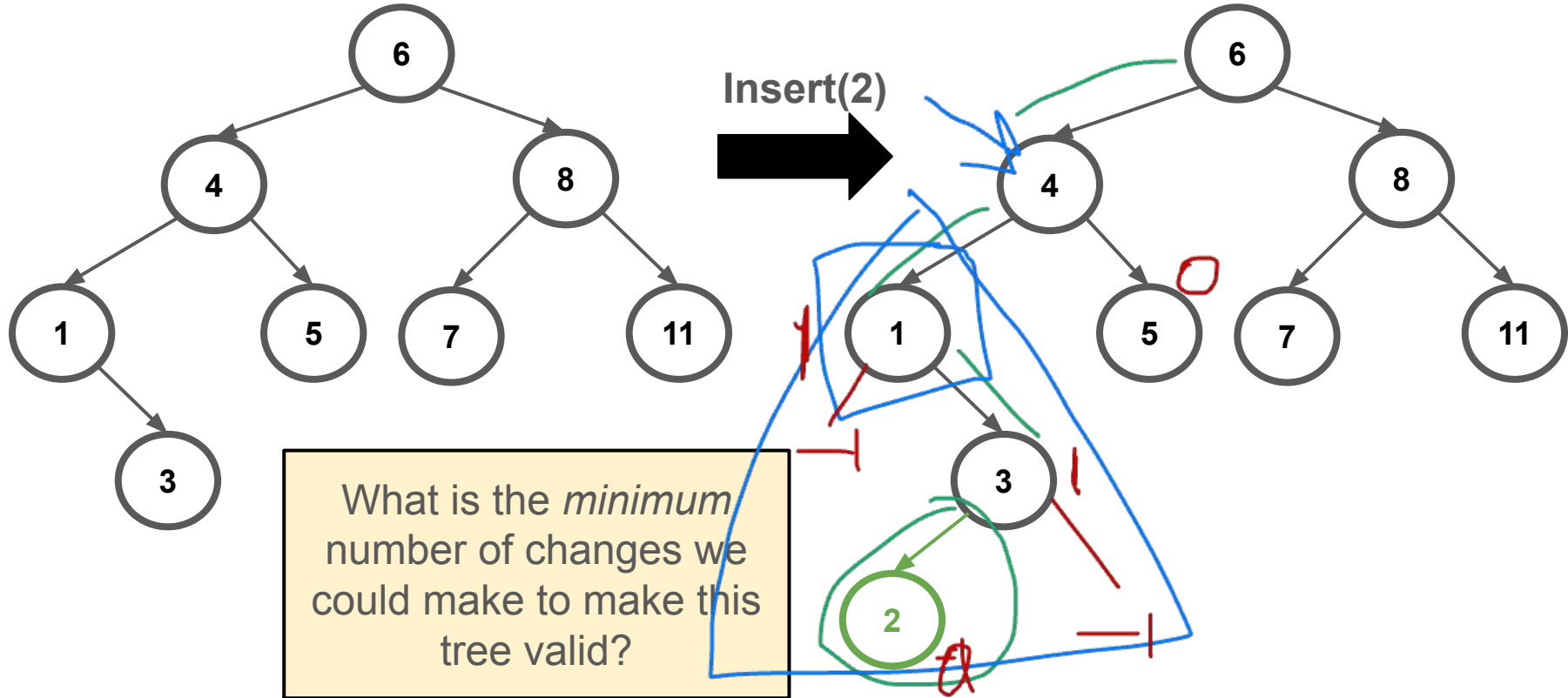
Ex3: An AVL tree?



An AVL Tree



An insert case



AVL tree operations

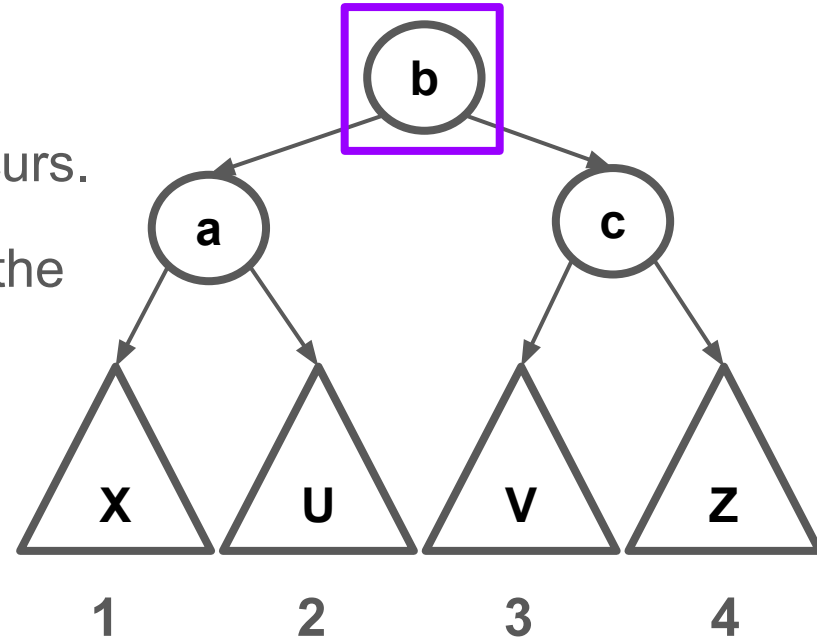
- **AVL find:**
 - Same as BST find
- **AVL insert:**
 - First BST insert, then check balance and potentially “fix” the AVL tree
 - Four different imbalance cases
- **AVL delete:**
 - The “easy way” is lazy deletion
 - Otherwise, like insert we do the deletion and then have several imbalance cases

AVL tree insert

Let **b** be the node where an imbalance occurs.

Four cases to consider. The insertion is in the

1. **left** subtree of the **left** child of **b**.
2. **right** subtree of the **left** child of **b**.
3. **left** subtree of the **right** child of **b**.
4. **right** subtree of the **right** child of **b**.



Idea: Cases 1 & 4 are solved by a **single rotation**.

Cases 2 & 3 are solved by a **double rotation**.

Insert: detect potential imbalance

1. **Insert** the new node as in a BST (a new leaf)
2. For each node on the path from the root to the new leaf, the insertion may (or may not) have changed the node's height
3. So after recursive insertion in a subtree, **detect height imbalance** and perform a rotation to restore balance at that node

All the action is in defining the correct rotations to restore balance

Fact that makes it a bit easier:

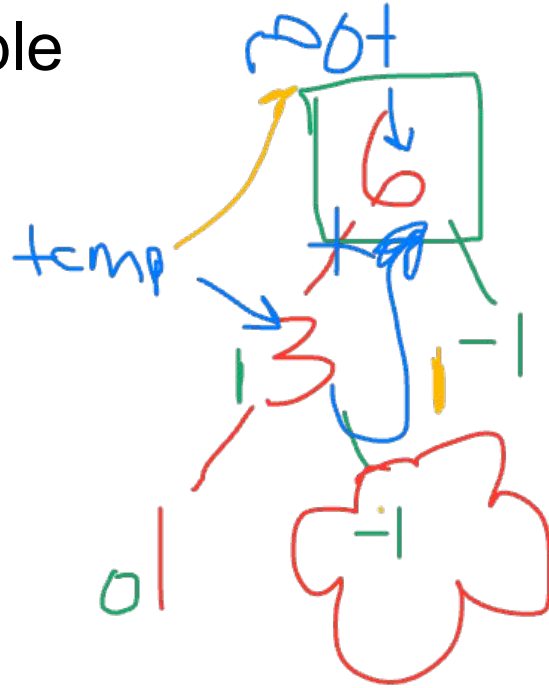
- There must be a deepest element that is imbalanced after the insert
- After rebalancing this deepest node, every node is balanced
- So at most one node needs to be rebalanced

Case #1 Example

Insert(6)

Insert(3)

Insert(1)

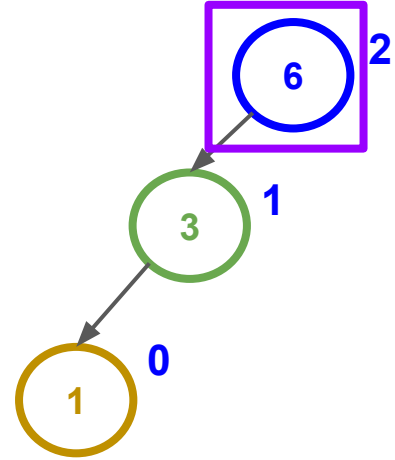
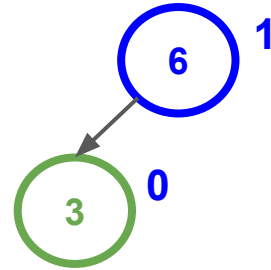


Case #1 Example

Insert(6)

Insert(3)

Insert(1)



Third insertion violates balance property

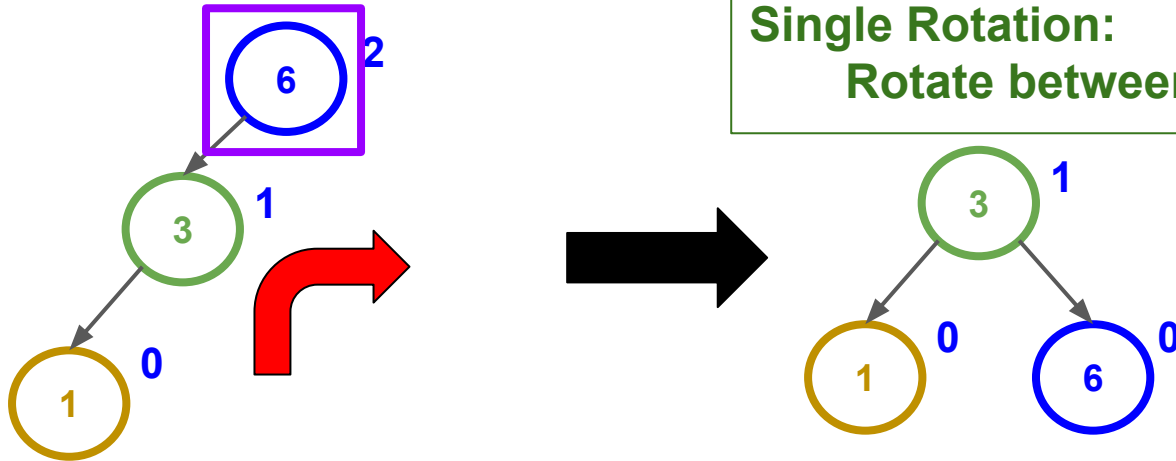
- Happens to be at the root

What is the only way to fix this?

Fix: Apply “Single Rotation”

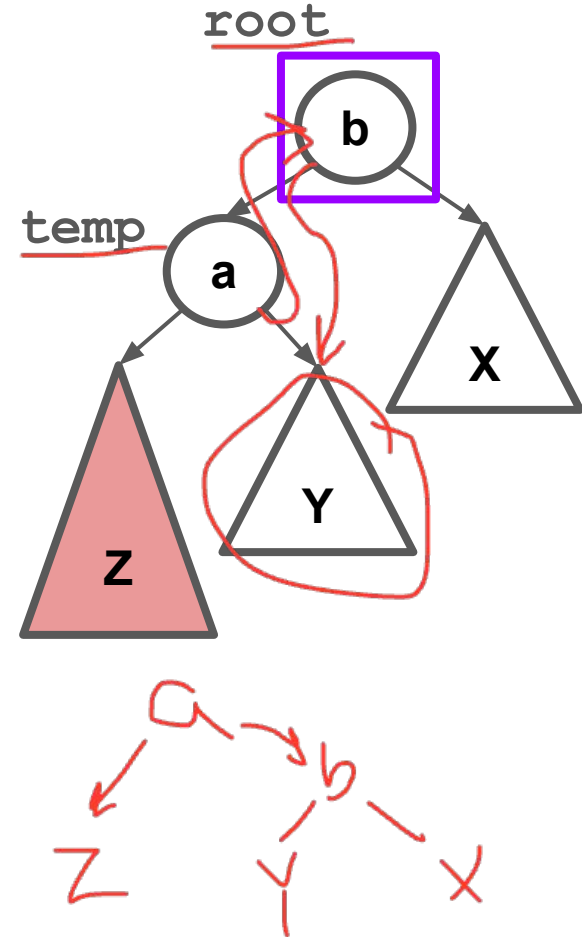
Single rotation: The basic operation we’ll use to rebalance

- Move child of unbalanced node into parent position
- Parent becomes the “other” child (always okay in a BST!)
- Other subtrees move in only way BST allows (next slide)



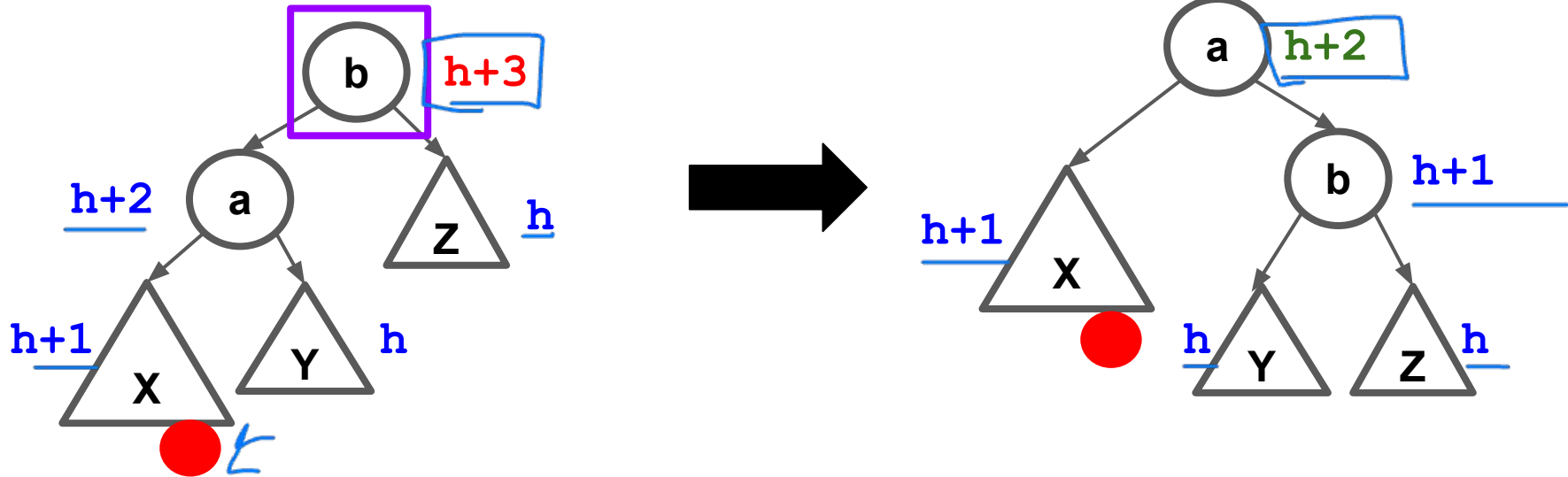
Single Rotation Pseudo-Code

```
void RotateWithLeft(Node root) {  
    Node temp = root.left  
    root.left = temp.right  
    temp.right = root  
    root.height = max(root.right.height(),  
                      root.left.height()) + 1  
    temp.height = max(temp.right.height(),  
                     temp.left.height()) + 1  
    root = temp  
}
```



The general left-left case

- Node imbalanced due to insertion somewhere in **left-left grandchild** increasing height
 - This is 1 of 4 possible imbalance causes (other three coming)
- So we rotate at **b**, using BST facts: $X < a < Y < b < Z$



Case #3 Example

Insert(1)

Insert(6)

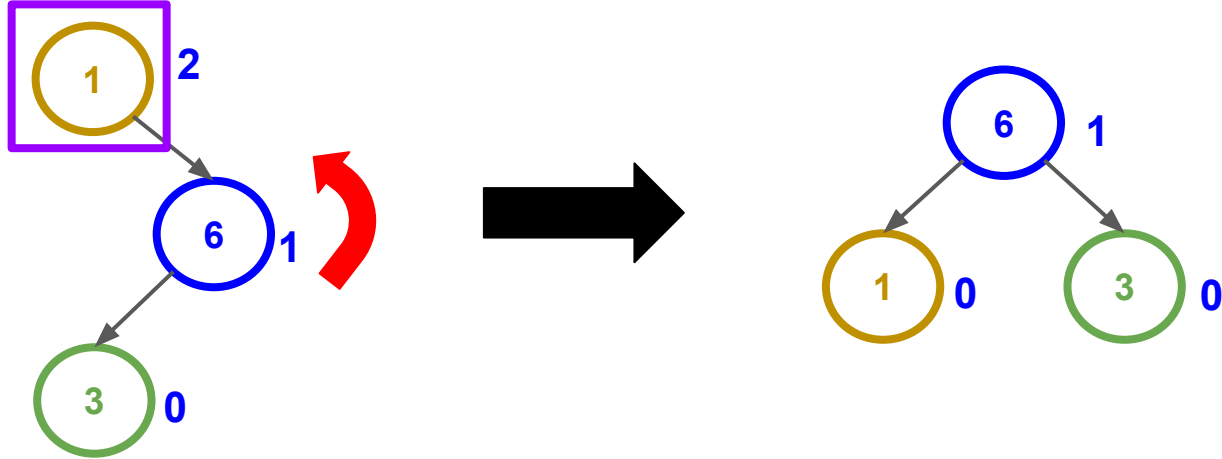
Insert(3)



Two cases to go

Unfortunately, single rotations are not enough for insertions in the left-right subtree or the right-left subtree

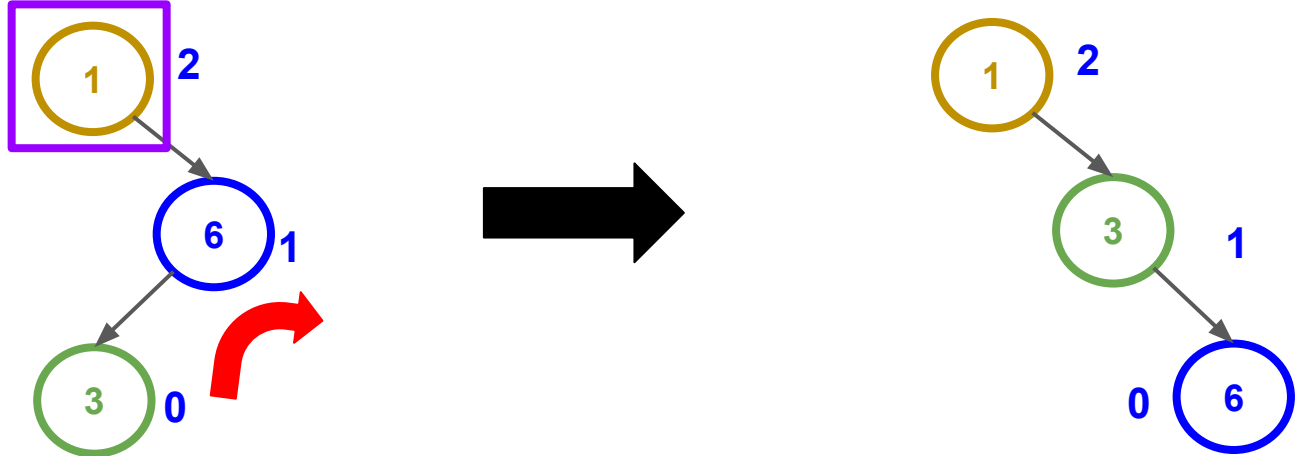
- **First wrong idea:** single rotation like we did for left-left



Two cases to go

Unfortunately, single rotations are not enough for insertions in the left-right subtree or the right-left subtree

- **Second wrong idea:** single rotation on the child of the unbalanced node



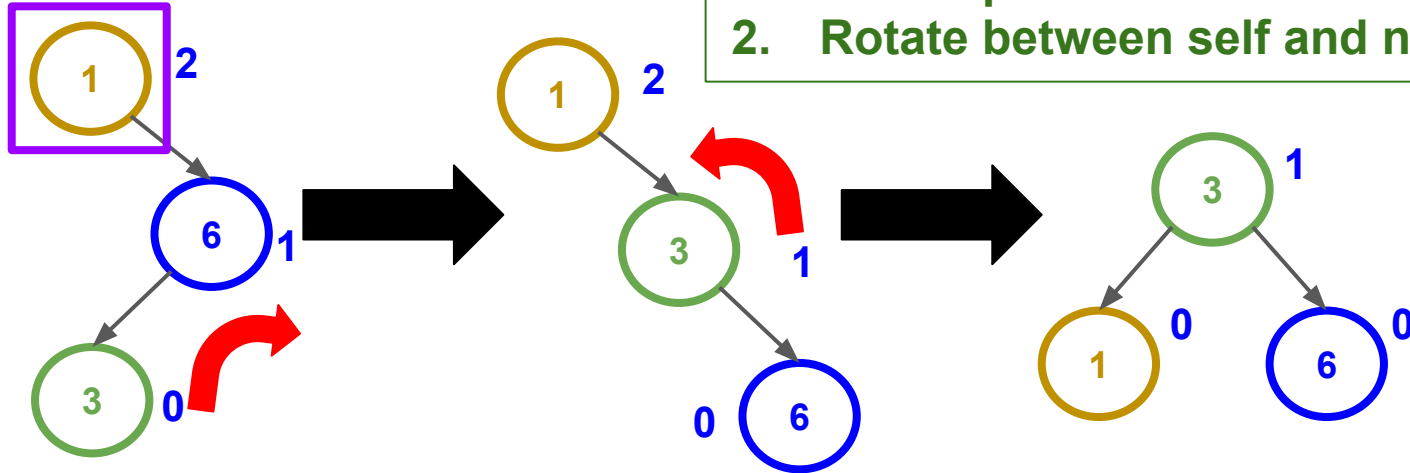
Sometimes two wrongs make a right

- First idea violated the BST property
- Second idea didn't fix balance
- But if we do both single rotations, starting with the second, it works!

(And not just for this example)

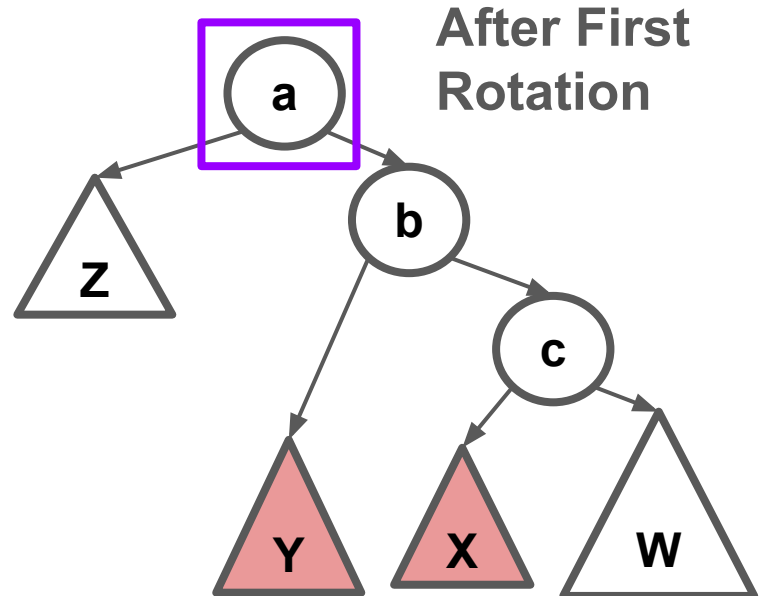
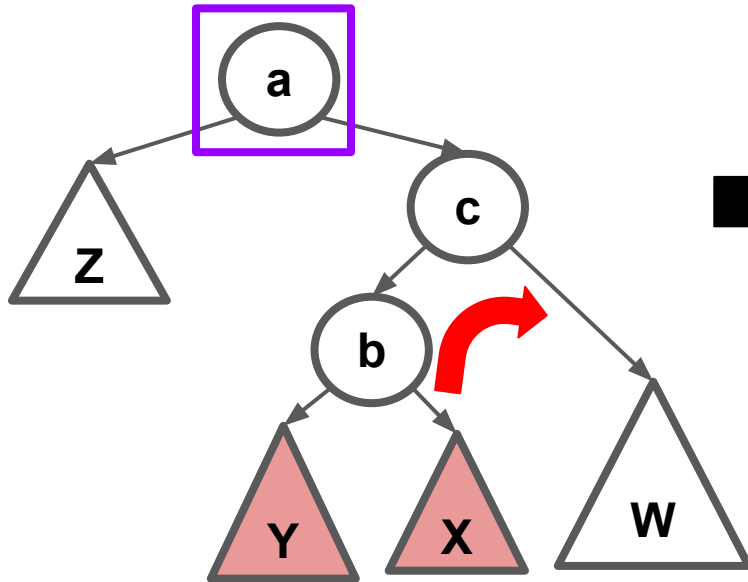
Double Rotation:

1. Rotate problematic child and grandchild
2. Rotate between self and new child

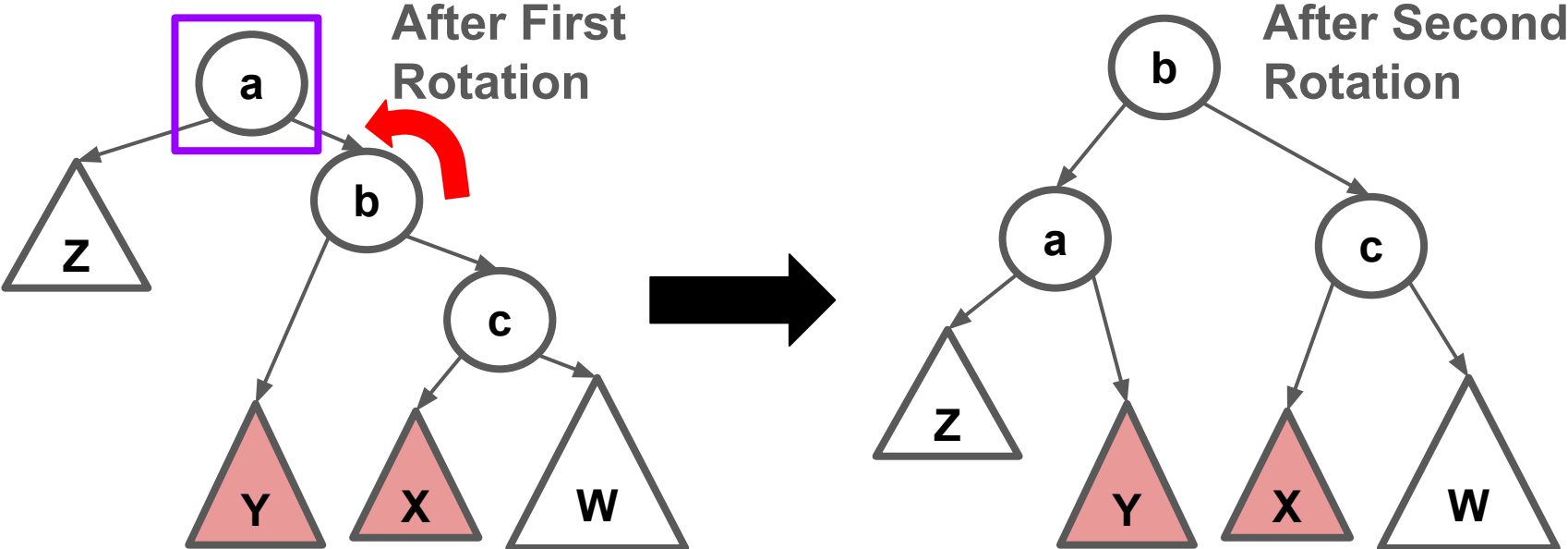


Double Rotation Pseudo-Code

```
void DoubleRotateWithRight(Node root) {  
    RotateWithLeft(root.right)  
    RotateWithRight(root)  
}
```



Double Rotation Completed

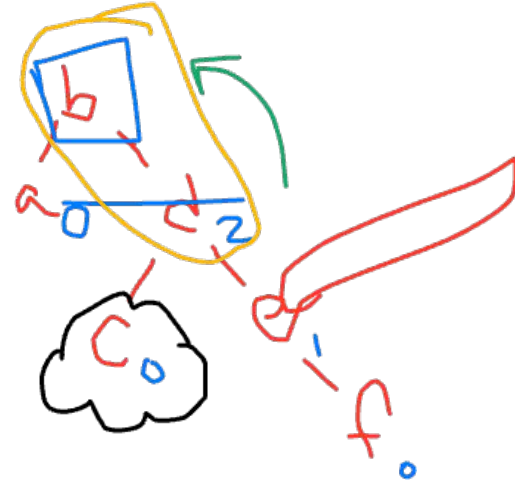


Insert, Summarized





- Insert as in a BST
- Check back up path for imbalance, which will be 1 of 4 cases:
 - node's left-left grandchild is too tall
 - node's left-right grandchild is too tall
 - node's right-left grandchild is too tall
 - node's right-right grandchild is too tall
- Only one case occurs because tree was balanced before insert
- After the appropriate single or double rotation, the smallest-unbalanced subtree has the same height as before the insertion
- So all ancestors are now balanced

Insert into an AVL tree: a b e c d

(if you have time, add f)



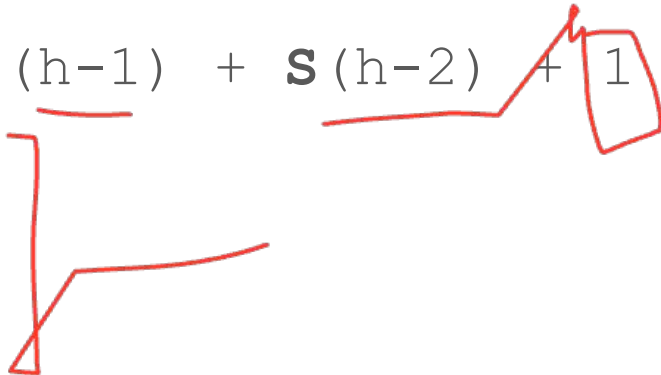
Height of an AVL tree?

height	Minimal AVL Tree	Number of nodes
0		1
1		2
2		4
3		7

Height of an AVL tree?

Using the AVL balance property, we can determine the minimum number of nodes in an AVL tree of height **h**

Let **S(h)** be the minimum # of nodes in an AVL tree of height **h**, then:

$$\begin{aligned} S(h) &= S(h-1) + S(h-2) + 1 \\ S(0) &= 1 \\ S(1) &= 2 \end{aligned}$$
Hand-drawn red lines and a box highlighting the recurrence relation and base cases. A vertical bracket on the left groups the three equations. A horizontal line underlines the term S(h-1) in the first equation. A diagonal line starts from the bottom of the S(1) = 2 equation and goes up to the right, crossing the S(h-2) term in the first equation. A box is drawn around the constant term '+ 1' in the first equation.

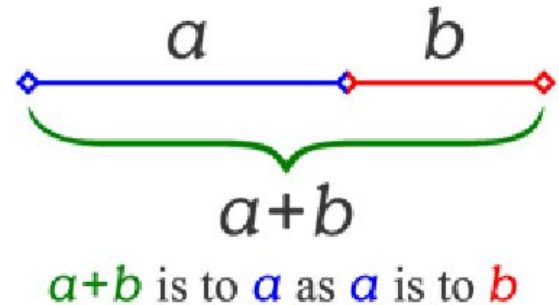
The Golden Ratio

This is a special number

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.62$$

- Aside: Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the **golden ratio**: If $(a+b) / a = a / b$, then $a = \phi b$
- We will need one special arithmetic fact about ϕ :

$$\begin{aligned}\phi^2 &= ((1+5^{1/2}) / 2)^2 \\ &= (1 + 2*5^{1/2} + 5) / 4 \\ &= (6 + 2*5^{1/2}) / 4 \\ &= (3 + 5^{1/2}) / 2 \\ &= 1 + (1 + 5^{1/2}) / 2 \\ &= 1 + \phi\end{aligned}$$



The Proof

Theorem: For all $h \geq 0$, $S(h) > \phi^h - 1$

Proof: By induction on h

Base cases:

$$S(0) = 1 : 1 > \phi^0 - 1$$

$$1 > 1 - 1$$

$$1 > 0$$

$$S(1) = 2 : 2 > \phi^1 - 1$$

$$2 > \text{approx } 0.62$$

The Proof

Inductive case ($k > 1$):

Show $S(k+1) > \phi^{k+1} - 1$ assuming $S(k) > \phi^k - 1$ and $S(k-1) > \phi^{k-1} - 1$

$$\begin{aligned} \mathbf{S(k+1)} &= 1 + S(k) + S(k-1) && \text{by definition of } S \\ &> 1 + \phi^k - 1 + \phi^{k-1} - 1 && \text{by induction} \\ &= \phi^k + \phi^{k-1} - 1 && \text{by arithmetic (1-1=0)} \\ &= \phi^{k-1} (\phi + 1) - 1 && \text{by arithmetic (factor } \phi^{k-1} \text{)} \\ &= \phi^{k-1} \phi^2 - 1 && \text{by special property of } \phi: \phi^2 = 1 + \phi \\ &= \phi^{k+1} - 1 && \text{by arithmetic (add exponents)} \end{aligned}$$

Good news

We know $S(k) > \phi^k - 1$

- Remember that $S(h)$ represented the *minimum number of nodes*

Therefore in Big-O speak:

$$n > \phi^h - 1$$

$$n > \phi^h$$

$$\log_{\phi}(n) > h$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Therefore the *height* of an AVL tree is bounded by $\log n$ - which means our find, insert and delete operations are all $\log n$!

Exercise 4: Single and Double Rotations:

Inserting what integer values would cause the tree to need a:

- 1. Single rotation?
- 2. Double rotation?
- 3. No rotation?

