CSE 326: Data Structures
AVL Trees

Brian Curless

Spring 2008

Potential Balance Conditions

What are some candidate balance conditions on BST's?

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*

Balancing Trees

- Many algorithms exist for keeping trees balanced
  - Adelson-Velskii and Landis (AVL) trees
  - Splay trees and other *self-adjusting* trees
  - B-trees and other multiway search trees (for *very* large trees)

- Today we will talk about **AVL trees**…

The AVL Tree Data Structure

<u>Ordering property</u>

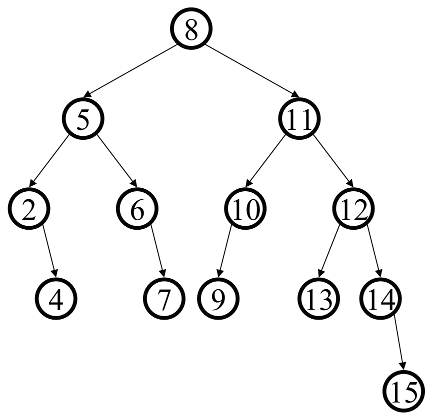
– Same as for BST

<u>Structural properties</u>

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:

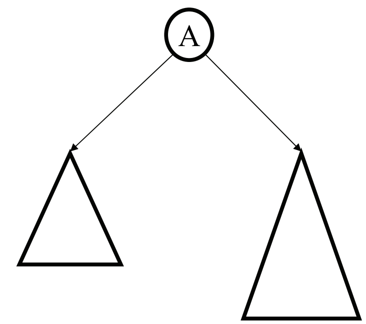
Worst case depth of any node is: O(log n)



Recursive Height Calculation

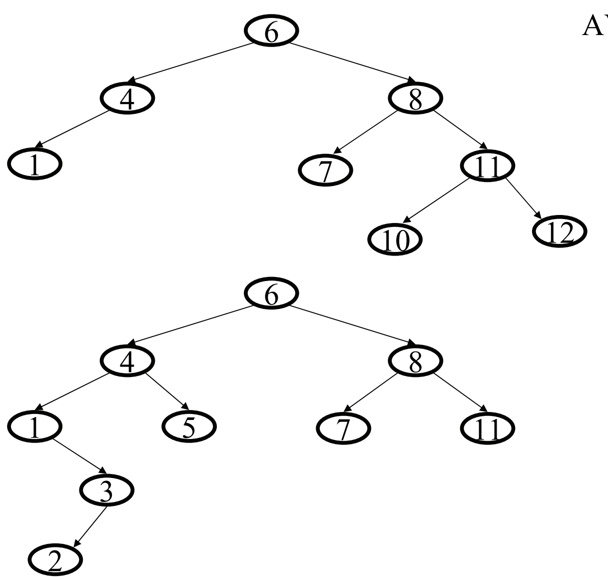
Recall: height is max number of edges from root to a leaf

What is the height at A?

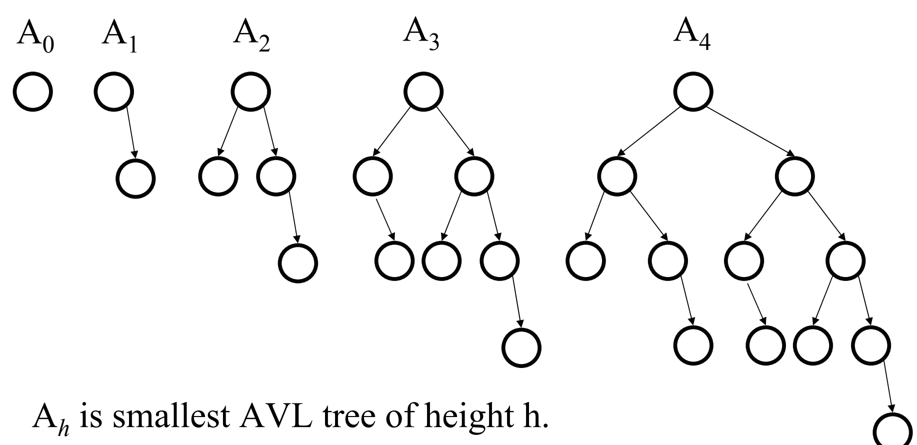


Note: height(null) = -1

AVL trees or not?



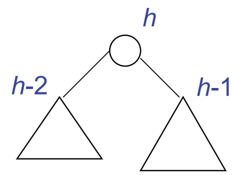
Proving Shallowness Bound



A_h is smallest AVL tree of height h.
Built from $A_{h\text{-}1}$ and $A_{h\text{-}2}$ attached to a new root.
(Note: these A_h's are not unique; e.g., could swap children.)

Minimum Size of an AVL Tree

- $m(h)$ = minimum number of nodes in an AVL tree of height h.
- Base cases
  - $m(0) = 1$, $m(1) = 2$
- Induction
  - $m(h) = m(h\text{-}1) + m(h\text{-}2) + 1$
- Bound solution
  - $m(h) > \phi^h - 1$
  - ϕ is the golden ratio, $(1+\sqrt{5})/2$

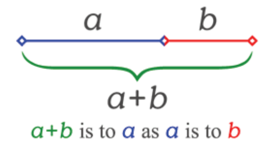


The Golden Ratio

Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the golden ratio:

$$\frac{a+b}{a} = \frac{a}{b} = \phi$$

The golden section:



Set a=1, solve for positive b, compute ratio: $\phi = \dfrac{1+\sqrt{5}}{2} \approx 1.62$

Note: $\phi^2 = \left(\dfrac{1+\sqrt{5}}{2}\right)^2 =$

Proof that $m(h) > \phi^h$ -1

- Base cases h=0,1:

$$m(0) = 1 > \phi^0 - 1 = 0 \qquad m(1) = 2 > \phi^1 - 1 \approx 0.62$$

- Assume true for h-2 and h-1:

$$m(h\text{-}2) > \phi^{h\text{-}2} - 1 \qquad m(h\text{-}1) > \phi^{h\text{-}1} - 1$$

- Induction step:

$$m(h) = m(h\text{-}1) + m(h\text{-}2) + 1 \;>\; (\phi^{h\text{-}1} - 1) + (\phi^{h\text{-}2} - 1) + 1$$

$$\begin{aligned}(\phi^{h\text{-}1} - 1) + (\phi^{h\text{-}2} - 1) + 1 &= \phi^{h\text{-}2}(\phi + 1) - 1\\ &= \phi^{h\text{-}2}(\phi^2) - 1\\ &= \phi^h - 1\end{aligned}$$

$\rightarrow m(h) > \phi^h - 1$

Maximum Height of an AVL Tree

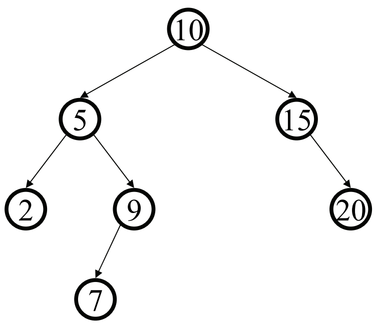
Suppose we have n nodes in an AVL tree of height h.

We can now say:

$$n \geq m(h) > \phi^h - 1$$

What does this say about the complexity of h?

Testing the Balance Property

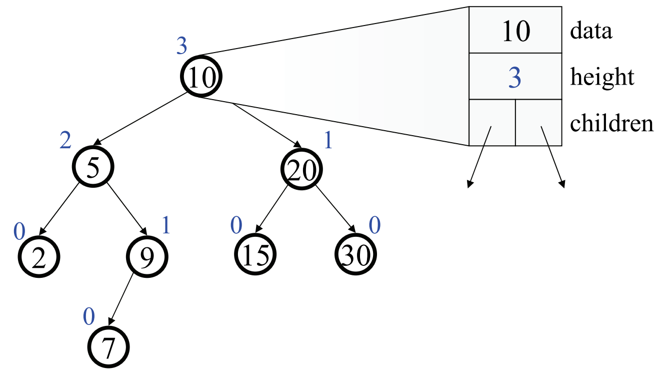


We need to be able to:

1. Track Balance
2. Detect Imbalance
3. Restore Balance

What if we insert(30)?

An AVL Tree

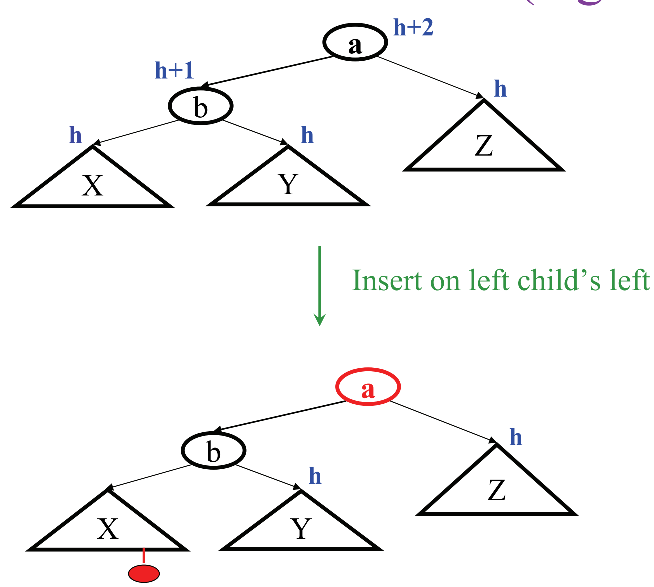


AVL trees: find, insert

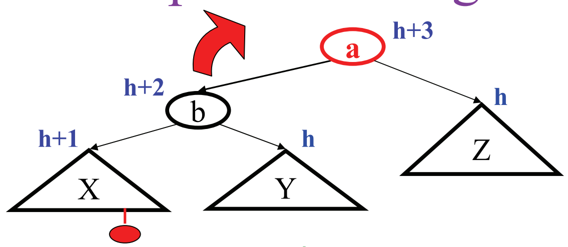
- **AVL find**:
  – same as BST find.
- **AVL insert**:
  – same as BST insert, *except* may need to “fix” the AVL tree after inserting new value.

We will consider the 4 fundamental insertion cases…

Case #1: left-left insertion (zig-zig)

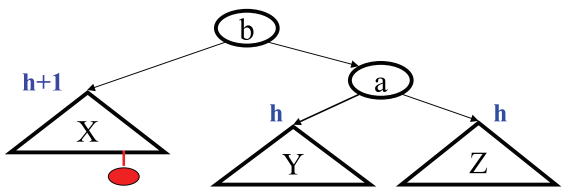


Case #1: repair with single rotation



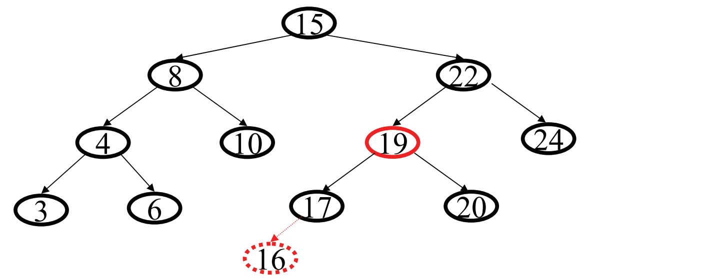
Single rotation

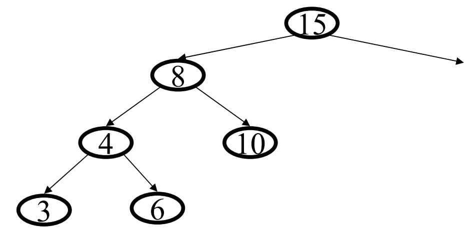
X < b < Y < a < Z



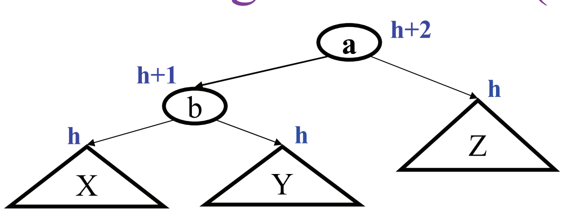
Height of tree before/after? Effect on Ancestors? Cost?

Single rotation example

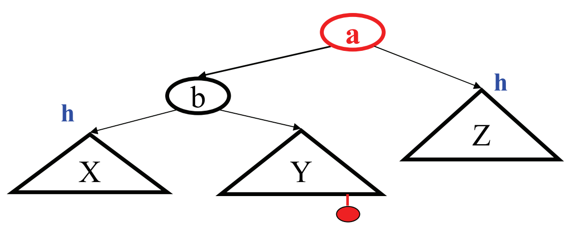




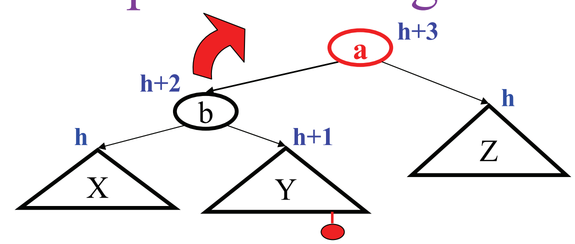
Case #2: left-right insertion (zig-zag)



Insert on left child's right

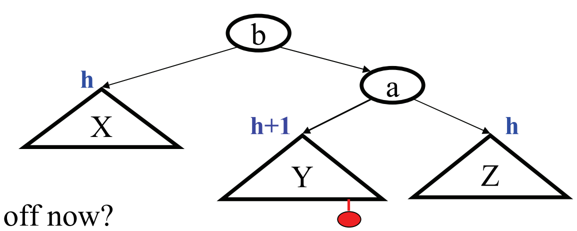


Case #2: repair with single rotation



X < b < Y < a < Z

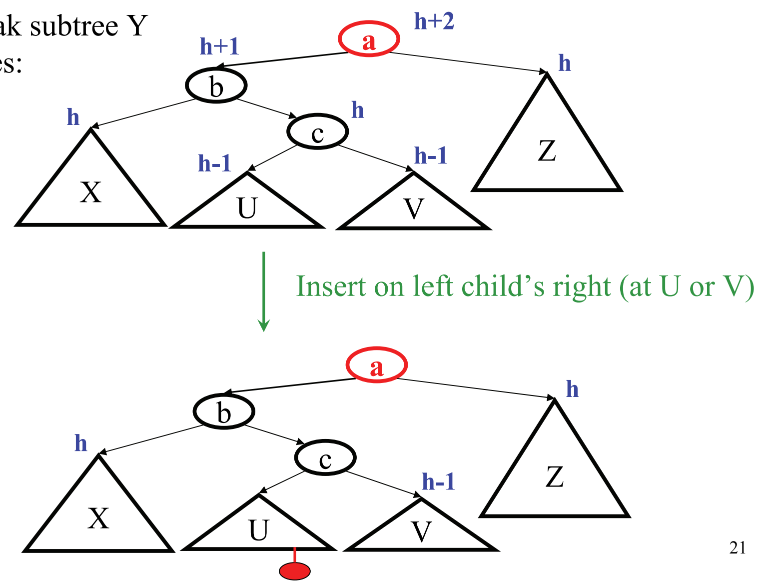
Single rotation



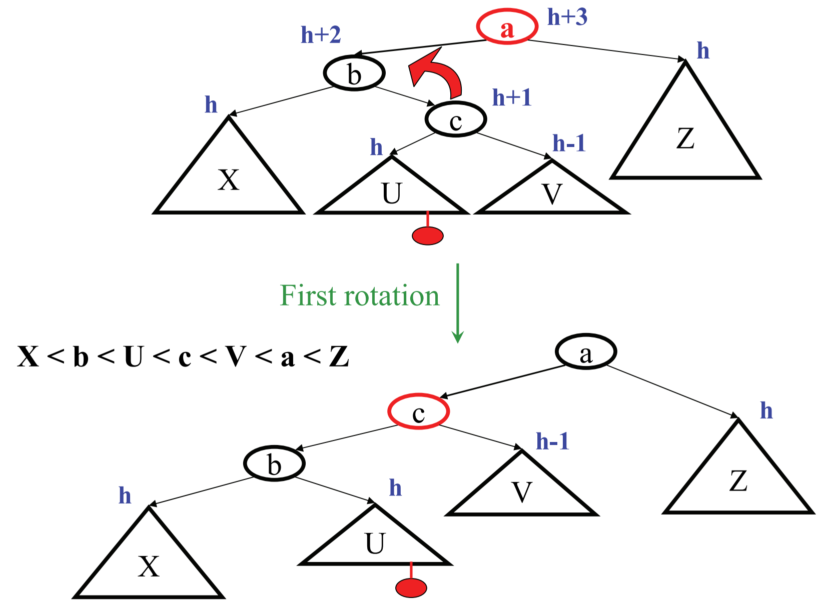
Are we better off now?

Case #2: trying again

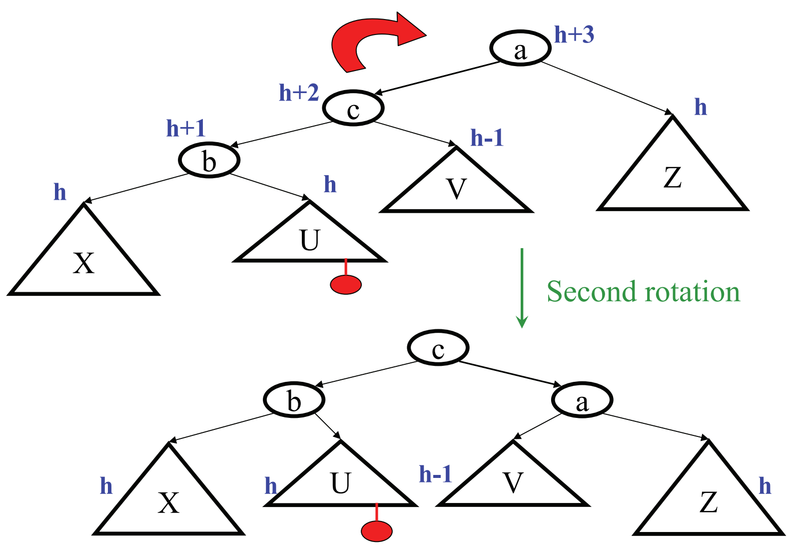
Let's break subtree Y into pieces:



Case #2: first rotation

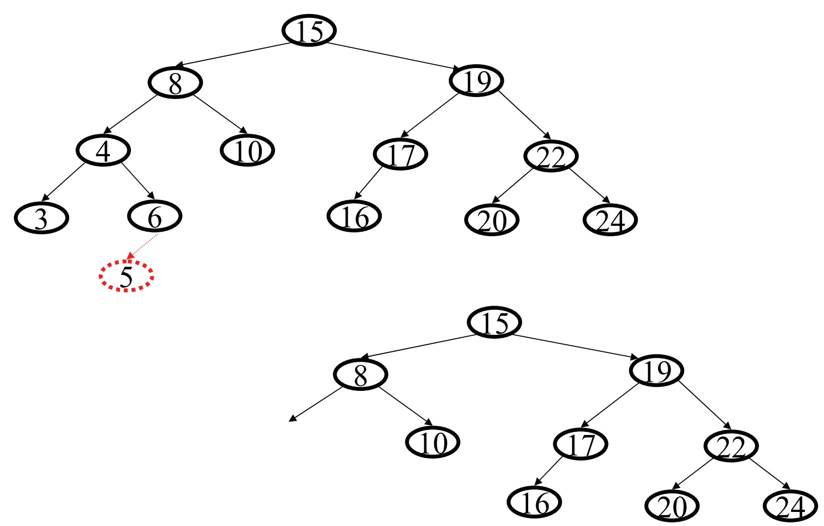


Case #2: second rotation

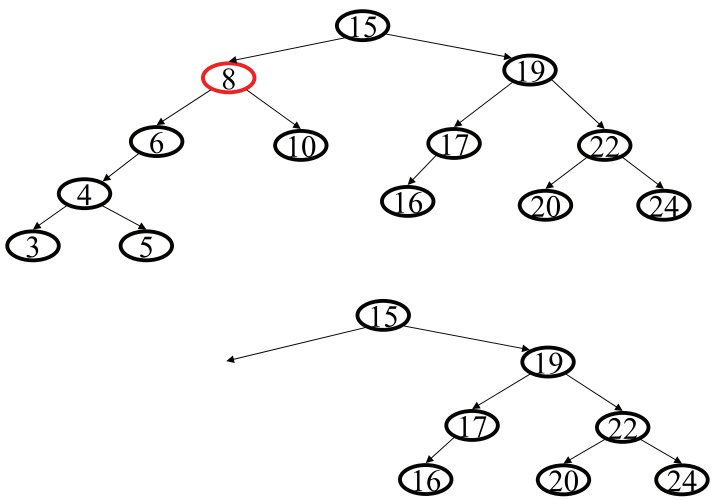


Height of tree before/after? Effect on Ancestors? Cost?

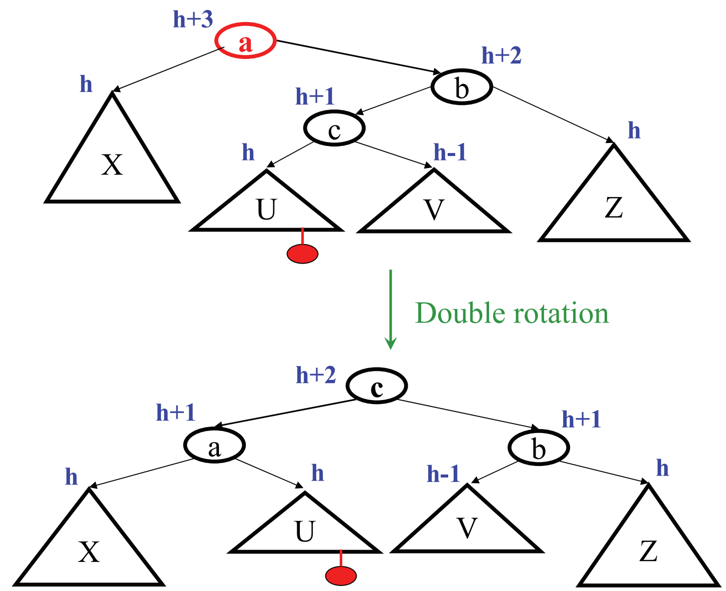
Double rotation, step 1



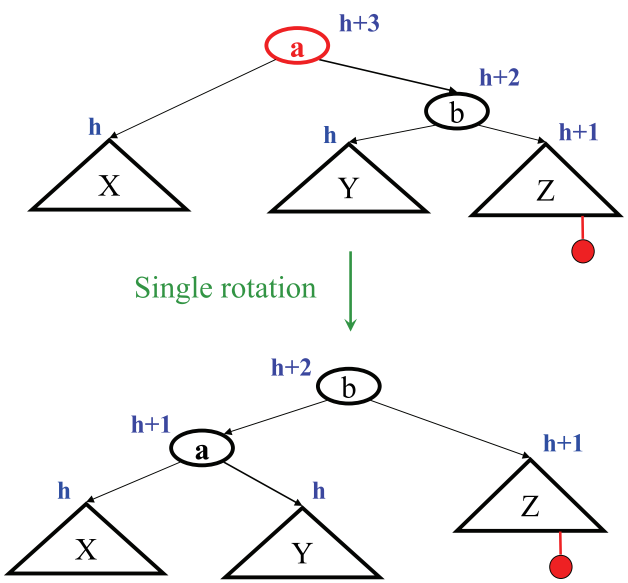
Double rotation, step 2



Case #3: right-left insertion (zig-zag)



Case #4: right-right insertion (zig-zig)



AVL tree case summary

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

Four cases to consider. The insertion below *a* is in the

1. **left** child's **left** subtree. (zig-zig)
2. **left** child's **right** subtree. (zig-zag)
3. **right** child's **left** subtree. (zig-zag)
4. **right** child's **right** subtree. (zig-zig)

Cases 1 & 4 are solved by a single rotation:

1. Rotate between *a* and child

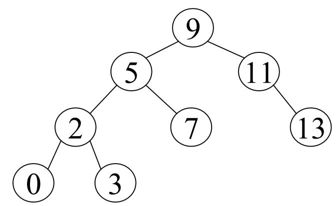
Cases 2 & 3 are solved by a double rotation:

1. Rotate between *a*'s child and grandchild
2. Rotate between *a* and *a*'s new child

Single and Double Rotations:

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

1. single rotation?

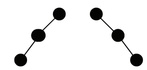


2. double rotation?

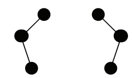
3. no rotation?

Insertion procedure

1. Find spot for new key
2. Hang new node there with this key
3. Search back up the path for imbalance
4. If there is an imbalance:



cases #1,#4: Perform single rotation and exit

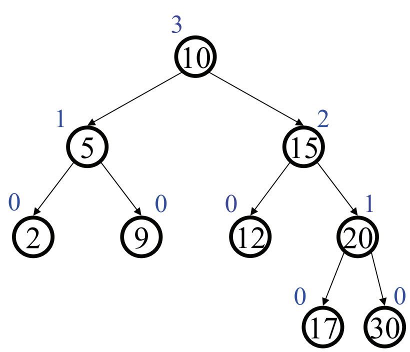


cases #2,#3: Perform double rotation and exit

Both rotations keep the subtree height unchanged.
Hence only one rotation is sufficient!

More insert examples

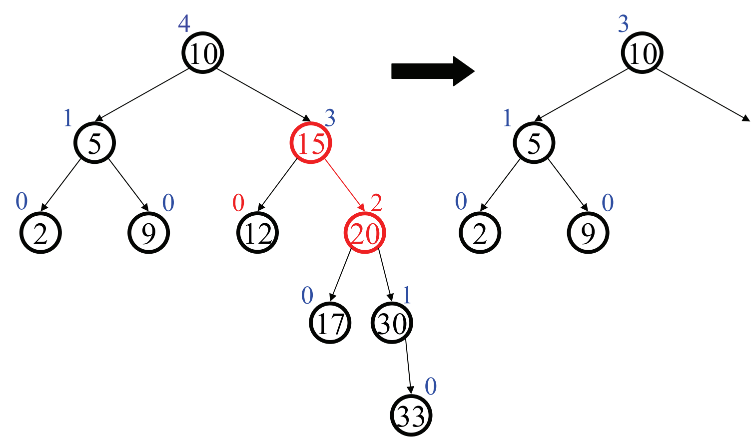
Insert(33)



Unbalanced?

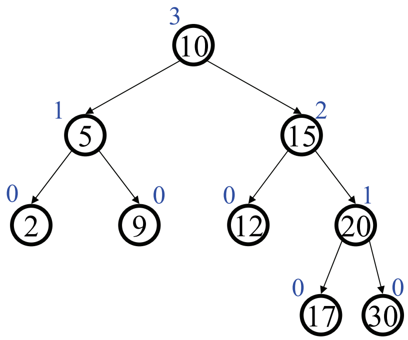
How to fix?

Single Rotation



More insert examples

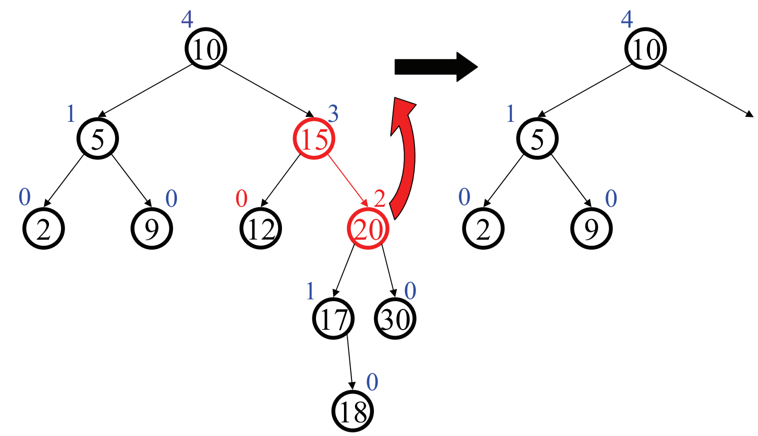
Insert(18)



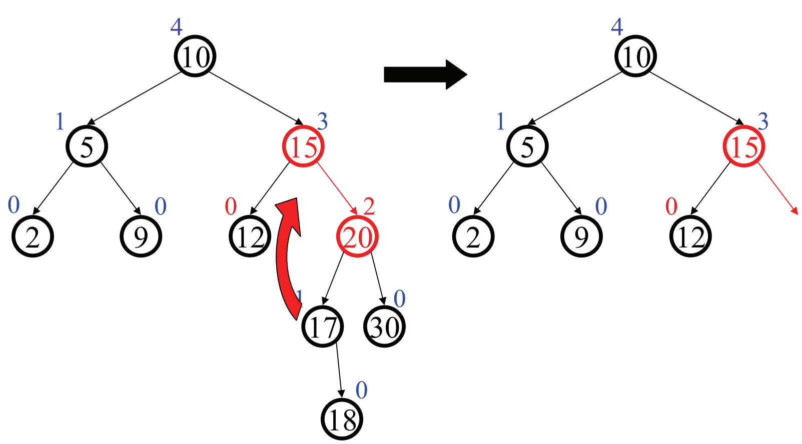
Unbalanced?

How to fix?

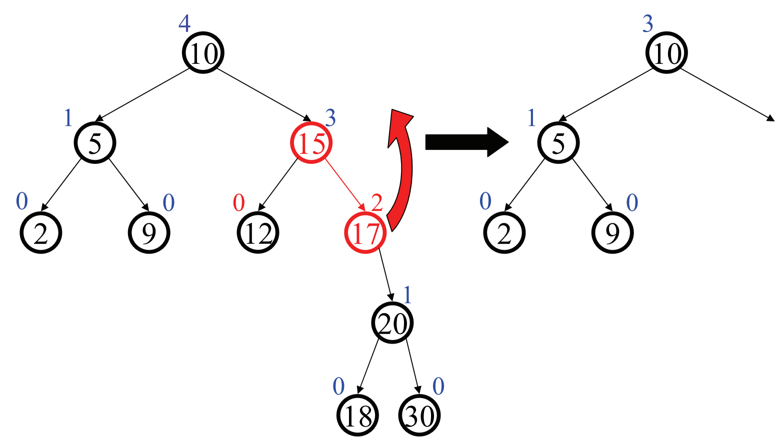
Single Rotation (oops!)



Double Rotation (Step #1)

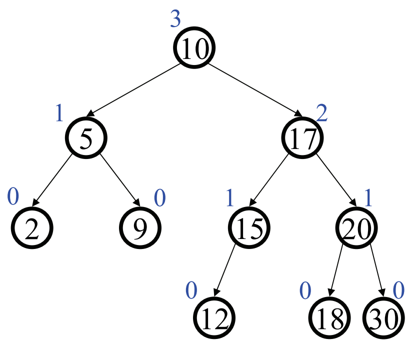


Double Rotation (Step #2)



More insert examples

Insert(3)



Unbalanced?

How to fix?

Insert into an AVL tree: 5, 8, 9, 4, 2, 7, 3, 1

AVL complexity

What is the worst case complexity of an insert?

What is the worst case complexity of a find?