

## AVL Trees (4.4 in Weiss)

CSE 373  
Data Structures & Algorithms  
Ruth Anderson  
Autumn 2012

---

---

---

---

---

---

---

---

## Today's Outline

- **Announcements**
  - Assignment #2 due Fri, Oct 12 at the BEGINNING of lecture
  - Midterm #1, Fri, Oct 19, 2012.
- **Today's Topics:**
  - Binary Search Trees (Weiss 4.1-4.3)
  - AVL Trees (Weiss 4.4)

10/08/2012

2

---

---

---

---

---

---

---

---

## The AVL Balance Condition

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

Define:  $\text{balance}(x) = \text{height}(x.\text{left}) - \text{height}(x.\text{right})$

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

- Ensures small depth
  - Will prove this by showing that an AVL tree of height  $h$  must have a lot of (i.e.  $\Theta(2^h)$ ) nodes
- Easy to maintain
  - Using single and double rotations

10/08/2012

3

---

---

---

---

---

---

---

---

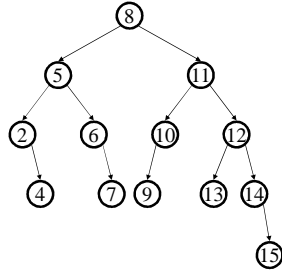
## The AVL Tree Data Structure

### 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

4

---

---

---

---

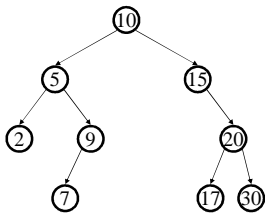
---

---

---

---

## Is this an AVL Tree?



NULLs have height -1

10/08/2012

5

---

---

---

---

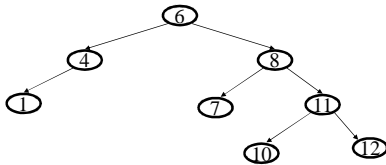
---

---

---

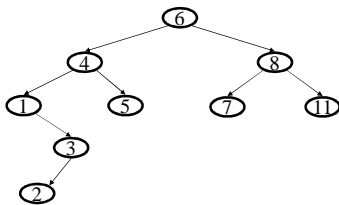
---

Circle One:



AVL

Not AVL



AVL

Not AVL

### Student Activity

If not AVL, put a **box** around nodes where AVL property is violated.

6

---

---

---

---

---

---

---

---

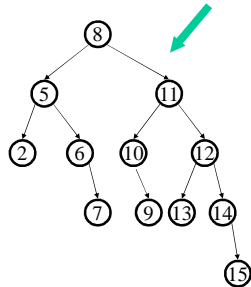
## Proving Shallowness Bound

Let  $S(h)$  be the min # of nodes in an AVL tree of height  $h$

Claim:  $S(h) = S(h-1) + S(h-2) + 1$

Solution of recurrence:  $S(h) = \Theta(2^h)$   
(like Fibonacci numbers)

AVL tree of height  $h=4$   
with the min # of nodes



10/08/2012

---

---

---

---

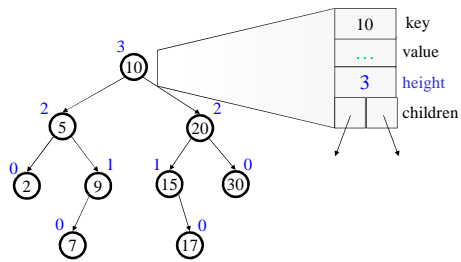
---

---

---

---

## An AVL Tree



10/08/2012

8

---

---

---

---

---

---

---

---

## 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.

10/08/2012

9

---

---

---

---

---

---

---

---

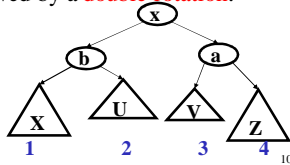
## AVL tree insert

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

Four cases to consider. The insertion is in the

1. left subtree of the left child of  $x$ .
2. right subtree of the left child of  $x$ .
3. left subtree of the right child of  $x$ .
4. right subtree of the right child of  $x$ .

**Idea:** Cases 1 & 4 are solved by a **single rotation**.  
Cases 2 & 3 are solved by a **double rotation**.



10/08/2012

10

---

---

---

---

---

---

---

---

## AVL Insert: detect & fix imbalances

1. Insert the new node just as you would in a BST (as a new leaf)
2. For each node on the path from the inserted node up to the root, the insertion may (or may not) have changed the node's height
3. So after recursive insertion in a subtree, check for height imbalance at each of these nodes and perform a *rotation* to restore balance at that node if needed

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

Fact that makes it a bit easier:

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

10/08/2012

11

---

---

---

---

---

---

---

---

## Bad Case #1

Insert(6)

Insert(3)

Insert(1)

10/08/2012

12

---

---

---

---

---

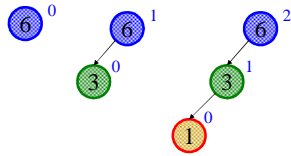
---

---

---

## Bad 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?

10/08/2012

13

---

---

---

---

---

---

---

---

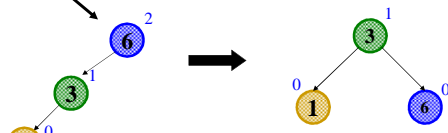
---

---

## 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)

AVL Property violated at this node ("x")



Single Rotation:  
1. Rotate between "x" and child

10/08/2012

14

---

---

---

---

---

---

---

---

---

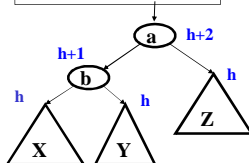
---

## Generalized left-left case

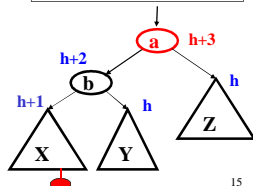
Notational note:  
Oval: a node in the tree  
Triangle: a subtree

- Node **a** imbalanced due to insertion *somewhere* in **left-left grandchild** increasing height of left subtree.
  - 1 of 4 possible imbalance causes (other three coming)
- First we did the insertion, which makes **a** imbalanced:

Before insertion – balanced.



After insertion – unbalanced!



10/08/2012

15

---

---

---

---

---

---

---

---

---

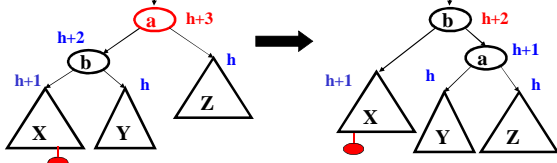
---

## Generalized left-left case (cont.)

- So we rotate at **a**, using BST facts:  $X < b < Y < a < Z$
- A **single rotation to the right** restores balance at the node
  - To same height as before insertion (so ancestors now balanced)

After insertion – unbalanced!

After single rotation – balanced!



16

---

---

---

---

---

---

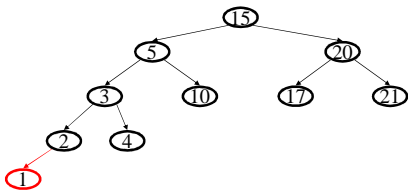
---

---

---

---

## Single rotation example: insert(1)



10/08/2012

17

---

---

---

---

---

---

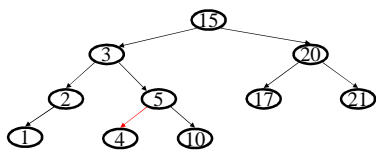
---

---

---

---

Soln:



10/08/2012

18

---

---

---

---

---

---

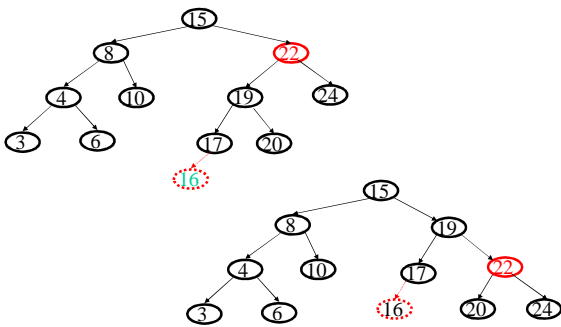
---

---

---

---

### Another example: insert(16)



10/08/2012

19

---

---

---

---

---

---

---

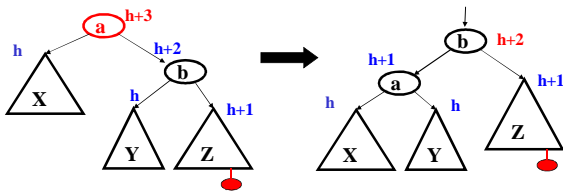
---

### The general right-right case

- Mirror image to left-left case, so you rotate the other way
  - Single rotation to the left
  - Exact same concept, but slightly different code

After insertion – unbalanced!

After single rotation – balanced!



10/08/2012

20

---

---

---

---

---

---

---

---

### Bad Case #3

- Insert(1)
- Insert(6)
- Insert(3)

10/08/2012

21

---

---

---

---

---

---

---

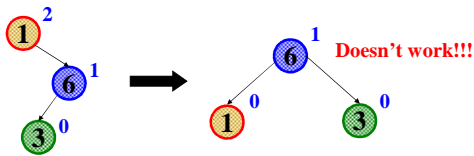
---

### Bad Case #3: Wrong Solution #1

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

Simple example: `insert(1), insert(6), insert(3)`

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



10/08/2012

22

---

---

---

---

---

---

---

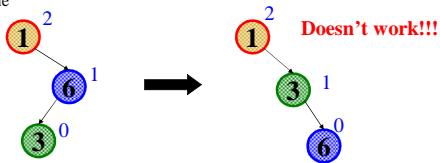
---

### Bad Case #3: Wrong Solution #2

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

Simple example: `insert(1), insert(6), insert(3)`

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



10/08/2012

23

---

---

---

---

---

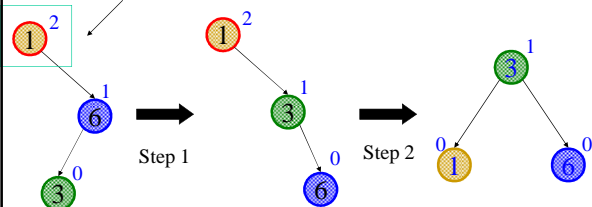
---

---

---

### Bad Case #3: Correct Solution: Double Rotation

AVL Property violated at this node ("x")



Double Rotation

1. Rotate between x's child and grandchild
2. Rotate between x and x's new child

10/08/2012

24

---

---

---

---

---

---

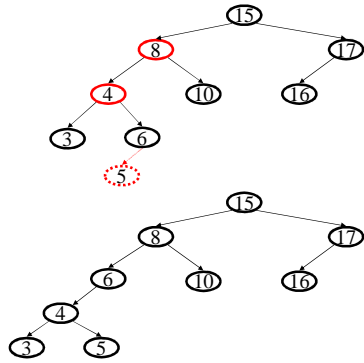
---

---





## Double rotation: insert (5), step 1



10/08/2012

28

---

---

---

---

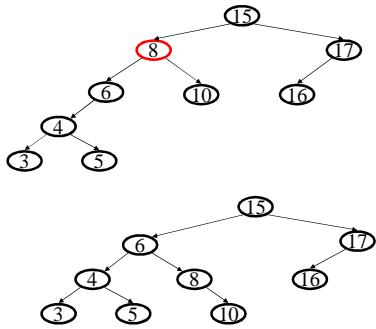
---

---

---

---

## Double rotation: insert (5), step 2



10/08/2012

29

---

---

---

---

---

---

---

---

## AVL Insert - Summary

- 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

10/08/2012

30

---

---

---

---

---

---

---

---

## Imbalance at node X

### Single Rotation

1. Rotate between x and child

### Double Rotation

1. Rotate between x's child and grandchild
2. Rotate between x and x's new child

10/08/2012

31

---

---

---

---

---

---

---

---

## Insert into an AVL tree: a b e c d

Student Activity

Circle your final answer

32

---

---

---

---

---

---

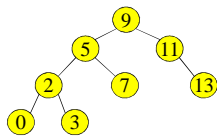
---

---

## Single and Double Rotations:

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

1. single rotation?



2. double rotation?

3. no rotation?

Student Activity

33

---

---

---

---

---

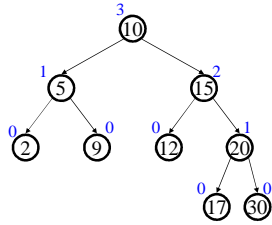
---

---

---

### Insert 3

Insert(3)



Unbalanced?

10/08/2012

34

---

---

---

---

---

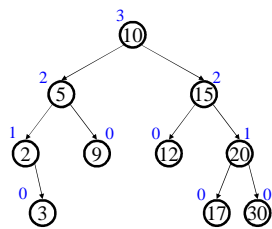
---

---

---

### Insert 33

Insert(33)



Unbalanced?

How to fix?

10/08/2012

35

---

---

---

---

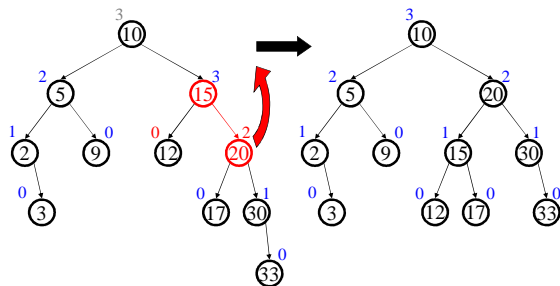
---

---

---

---

### Insert 33: Single Rotation



36

---

---

---

---

---

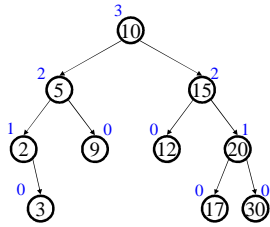
---

---

---

## Insert 18

Insert(18)



Unbalanced?

How to fix?

37

---

---

---

---

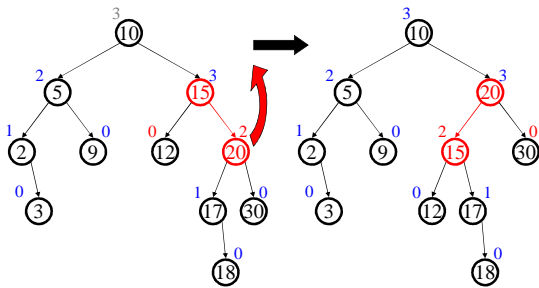
---

---

---

---

## Insert 18: Single Rotation (oops!)



10/08/2012

38

---

---

---

---

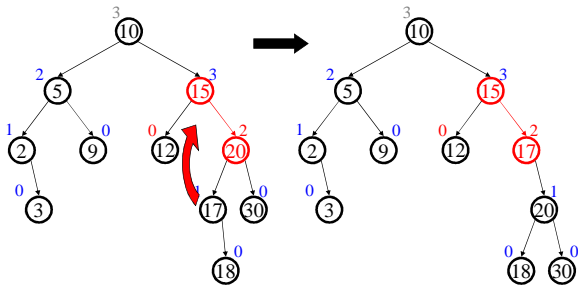
---

---

---

---

## Insert 18: Double Rotation (Step #1)



10/08/2012

39

---

---

---

---

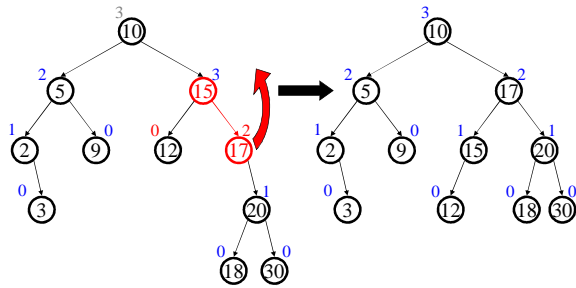
---

---

---

---

## Insert 18: Double Rotation (Step #2)



10/08/2012

40

---

---

---

---

---

---

---

---

## AVL Trees Revisited

- **Balance condition:**
  - For every node  $x$ ,  $-1 \leq \text{balance}(x) \leq 1$ 
    - Strong enough : Worst case depth is  $O(\log n)$
    - Easy to maintain : *one* single or double rotation
- **Guaranteed  $O(\log n)$  running time** for
  - Find ?
  - Insert ?
  - Delete ?
  - buildTree ?

10/08/2012

41

---

---

---

---

---

---

---

---

## AVL Trees Revisited

- What **extra info** did we maintain in each node?
- **Where** were rotations performed?
- How did we **locate** this node?

10/08/2012

42

---

---

---

---

---

---

---

---