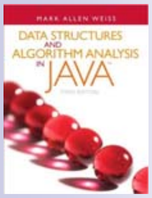
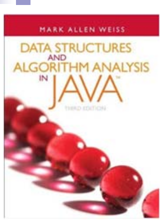


FIU | FLORIDA INTERNATIONAL UNIVERSITY

Trees
(IV)

Dr. Antonio L. Bajuelos

FIU | School of Computing & Information Sciences



COP-3530 - Data Structures

Module #3: Trees (part IV)

Outline:

- **The model of the AVL trees.**
- **AVL tree operations:**
  - **Find.**
  - **Insertion – single & double rotation.**
  - **Remove - lazy deletion.**
  - **Complexity analysis.**
- **Java implementation of AVL trees.**

2

The Model of the AVL Trees

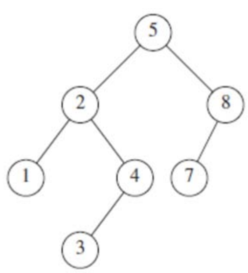
- **Definition:**
  - The **AVL** tree** is a **binary search tree with the balance condition**.
  - **Balance condition**: <u>for every node</u> in the tree, the **height** of the <u>left and right subtrees can differ by at most 1</u>.
  - We assume that the height of an empty tree is defined to be −1.

AVL - Adelson-**V**elskii and **L**andis

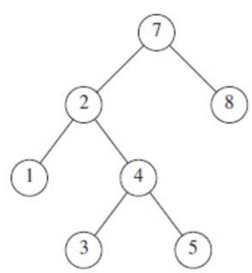
3

The Model of the AVL Trees (cont…)

- **AVL tree – BST with balance condition**
  - **Balance condition**: <u>for every node</u> in the tree, the **height** of the <u>left and right subtrees can differ by at most 1</u>.



AVL Tree

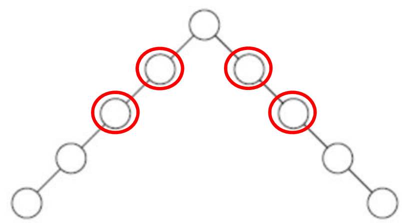


Binary Search Tree

4

The Model of the AVL Trees (cont...)

- **Example:**
  - Is this an AVL tree?



  - Nope!, because the **balance condition** is for <u>every node</u> of the tree! This is a “bad” binary search tree.

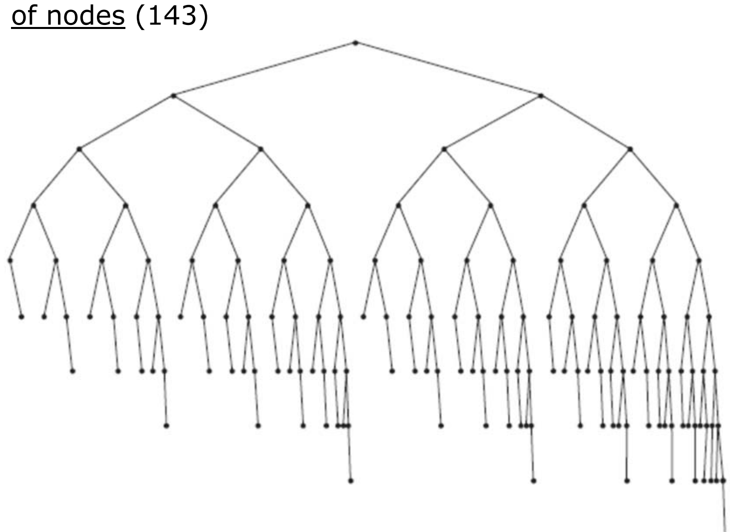
balance(*node*) = height(*node*.left) – height(*node*.right)

AVL *property*: –1 ≤ **balance(*x*)** ≤ **1** ; for every node *x*

5

The Model of the AVL Trees (cont...)

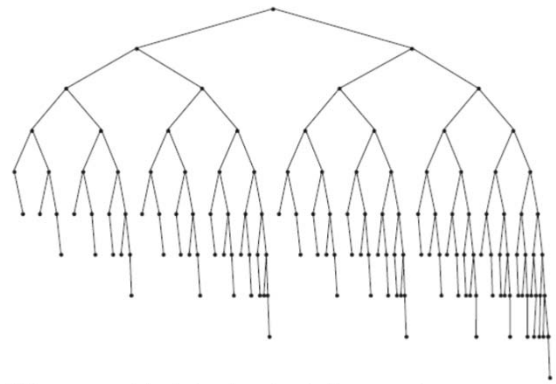
- **Example:**
  - The **AVL tree** of **height 9** with the <u>smallest number of nodes</u> (143)



6

The Model of the AVL Trees (cont…)

- **Example:**



- **Left subtree** - AVL tree of height 7 of minimum size.
- **Right subtree** - AVL tree of height 8 of minimum size.
- Then we have: the **minimum number of nodes**, $S(h)$, in an AVL tree of height h is given by $\boldsymbol{S(h) = S(h-1)+S(h-2)+1}$.
  For $h = 0$, $S(h) = 1$.
  For $h = 1$, $S(h) = 2$.

7

The Model of the AVL Trees (cont…)

- **Theorem:**
  - An **AVL tree** with **N** nodes has **O(logN)** height.

  - **Proof:** (some ideas)
    - N(h) = minimum number of nodes in an AVL tree of height h.
    - Base cases:
      - N(0) = 1, N(1) = 2
    - Induction:
      - N(h) = N(h-1) + N(h-2) + 1
    - Then apply Fibonacci analysis . . .

8

The Model of the AVL Trees (cont…)

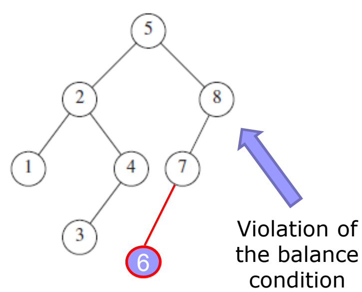
- **Important points:**
  - Operations **Contain/Find, Insert** and **Delete** (lazy deletion) can be performed in ***O*(log*N*) time**.
  - Note that when we do an **insertion**, we need to update all the balancing information for the nodes on the path back to the root, but the reason that insertion is potentially difficult is that inserting a node could violate the AVL tree property.

Lazy deletion: Each node contains a boolean field indicating if they are deleted or not. To delete a key from the tree, just find the node containing that key and mark it as deleted.

9

The Model of the AVL Trees (cont…)

- **Important points:**
  - **Example:** When we insert the key 6 into the AVL tree we would destroy the balance condition



10

AVL tree operations

- **AVL Find:**
  - Same as **BST** find

- **AVL Insert:**
  - **BST** insert, then check balance and potentially "repair" the **AVL** tree
  - Four different imbalance cases

- **AVL Delete:**
  - The "easy way" is lazy deletion.

11

Insert: detect potential imbalance

- **Algorithm Ideas:**
  - Insert the new node as in a **BST** (**a new leaf**)
  - Verify that 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
  - If height imbalance is detected (after insertion) then perform a **rotation** to restore balance at that node

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

12

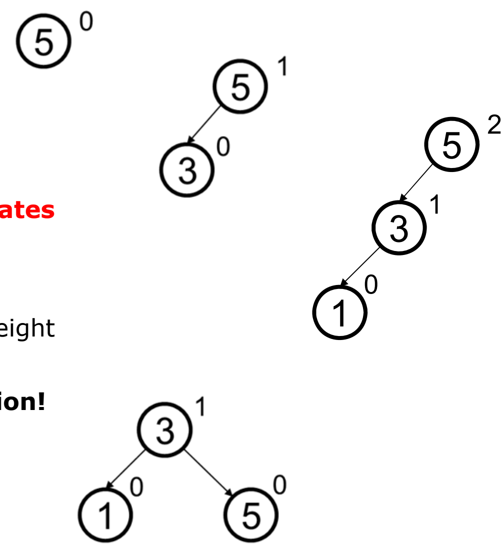
Insert: detect potential imbalance

- **Example:**
  Insert(5)
  Insert(3)
  Insert(1)

  Third insertion violates balance property!!!

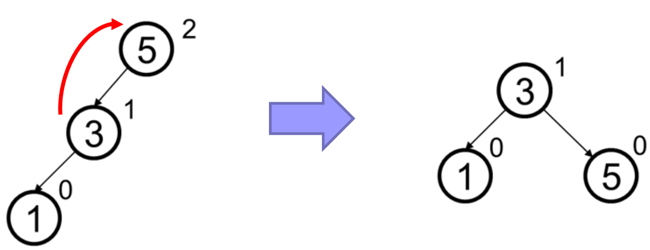
  How to repair this height imbalance?

  Make a single rotation!



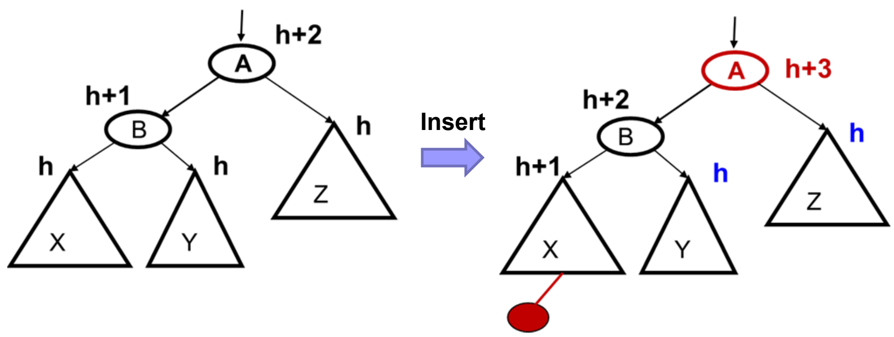
Insert: Single Rotation

- **Single Rotation Ideas:**
  - Basic operation that is used to rebalance an AVL tree
  - Move child of unbalanced node into parent position
  - Parent becomes the "other" child (always ok in a BST!)
  - Other subtrees move in only way BST allows (general case)



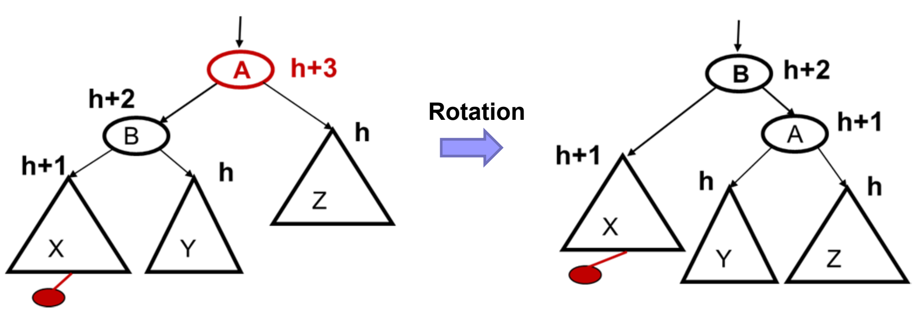
Insert: Single Rotation

- **General Case:**
  - Insertion into **left-left grandchild** causes an height imbalance
    - 1 of 4 possible imbalance causes (other 3 coming up!)
  - When insert a new node (in red) an height imbalance occurs in the **AVL tree** (specifically node A is imbalanced)



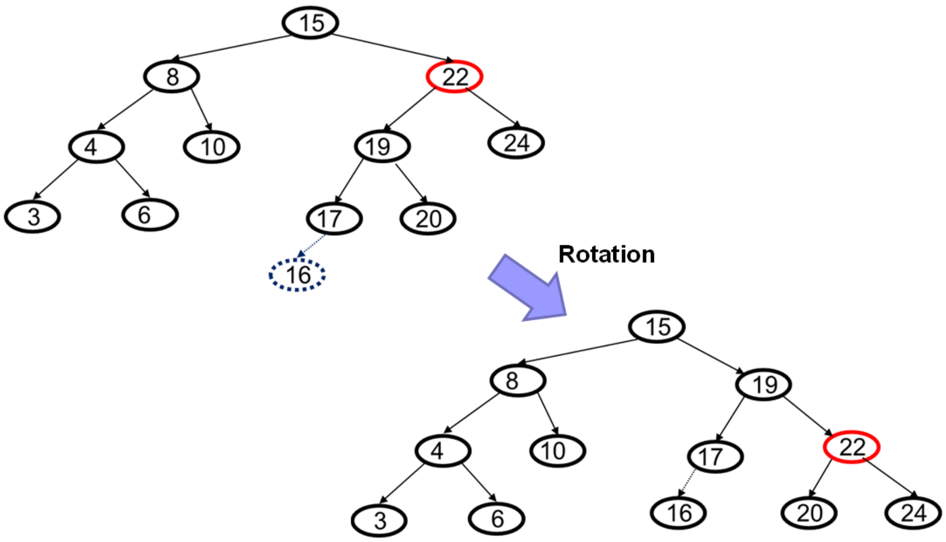
Insert: Single Rotation

- **General Case (left-left grandchild):**
  - So we rotate at node **A**
    - <u>Move child of unbalanced node into parent position</u>
    - Parent becomes the “other” child
    - Other sub-trees move in the only way BST allows:
      - using BST facts: X < b < Y < a < Z



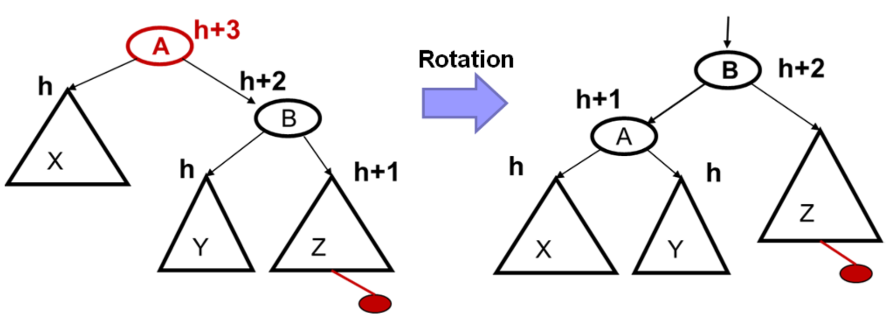
Insert: Single Rotation

- **General Case (left-left grandchild):**
  - **Example:** insert(16)



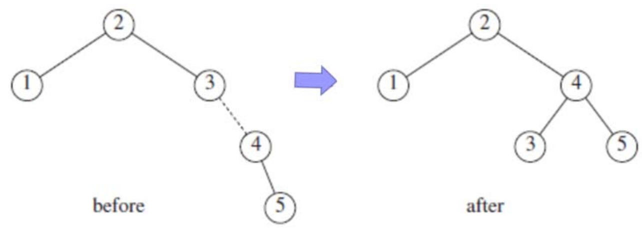
Insert: Single Rotation

- **General Case (right-right grandchild):**
  - Mirror image to left-left general case, so you rotate the other way
  - Note that the concepts are the same concept, but need different code

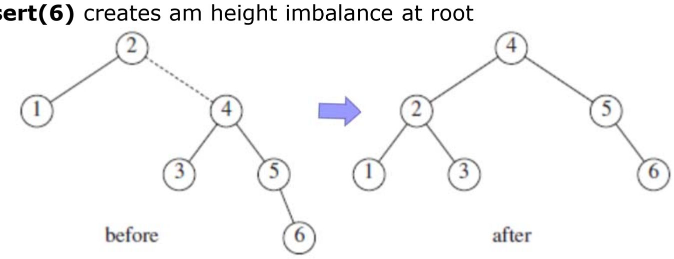


Insert: Single Rotation

- **Example (right-right grandchild):**
  - **insert(5)** creates an height imbalance at node 3

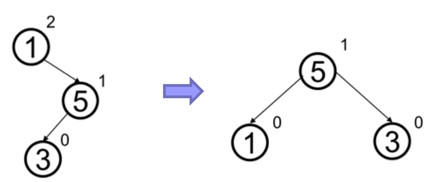


  - **insert(6)** creates am height imbalance at root

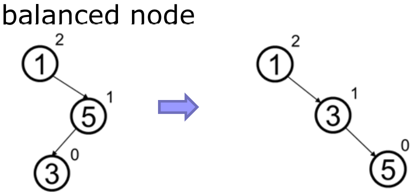


Insert: Other cases

- Note that single rotations are not enough for insertions in the **left-right** subtree or the **right-left** subtree
- **Example:** insert(1), insert(5), insert(3)
  - 1st wrong idea: single rotation for left-left subtree

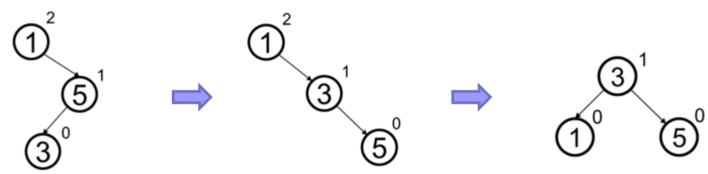


  - 2st wrong idea: single rotation on the child of the unbalanced node



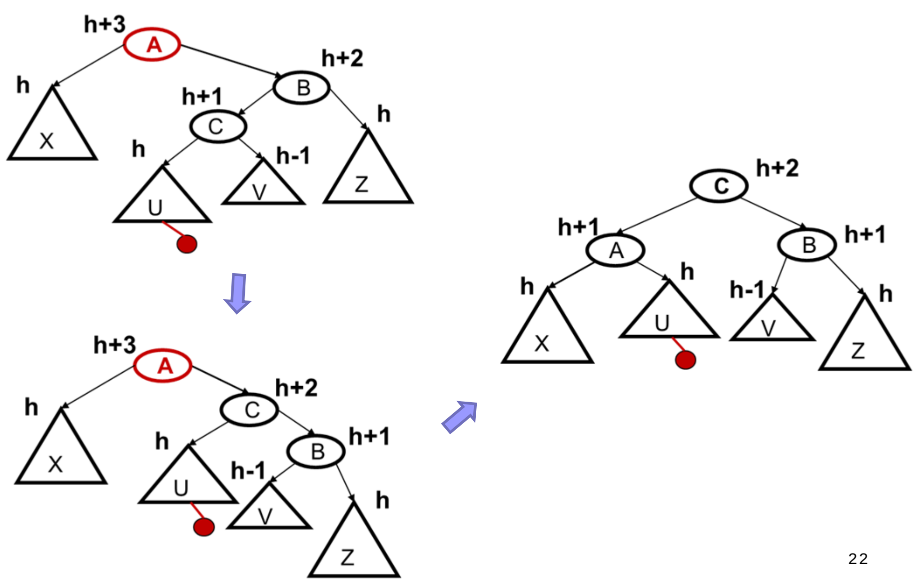
Insert: Double rotation

- If we do <u>both single rotations</u>, starting with the second, it works!
- **Double rotation:**
  - Rotate problematic child and grandchild
  - Rotate between self and new child



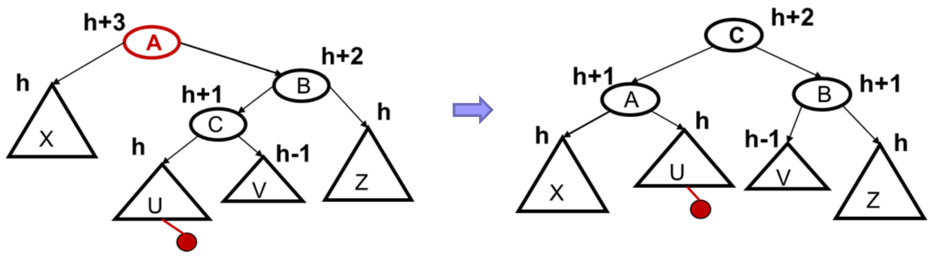
Insert: Double rotation

- General Case - **right-left case**



Insert: Double rotation

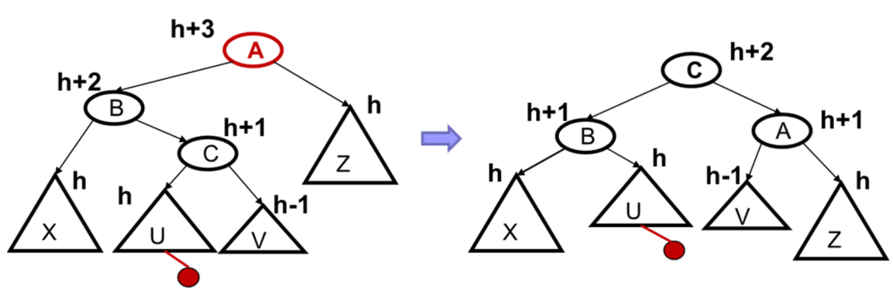
- **Important points (right-left case):**
  - Does not have to be implemented as two rotations; can just do:



  - Easier to remember than you may think:
    - Move C to grandparent's position
    - Put A, B, X, U, and V in the only legal positions for a BST

Insert: Double rotation

- **General case: left-right case:**
  - The left-right is the mirror image of the right-left case



Insert: Summarized Ideas

- **Insert** as in a **BST**

- Check back up path for imbalance height, which will be:
  - 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

- Note that only one case occurs because tree was balanced before insert

- After the **single** or **double rotation**, the smallest-unbalanced subtree has the same height as before the insertion

AVL trees: Efficiency

- Worst-case complexity of **find**: ***O*(log*N*)**
  - Tree is balanced

- Worst-case complexity of **insert**: ***O*(log*N*)**
  - Tree starts balanced
  - A rotation is *O*(1) and there's an ***O*(log*N*)** path to root
  - Tree ends balanced

- Worst-case complexity of **buildTree** with *N* nodes: ***O*(*N*log*N*)**

AVL Trees. Some Java Code

Node Declaration for AVL tree

```
private static class AvlNode<AnyType>
{
  // Constructors
  AvlNode( AnyType theElement )
    { this( theElement, null, null ); }

  AvlNode( AnyType theElement, AvlNode<AnyType> lt, AvlNode<AnyType> rt )
    { element = theElement; left = lt; right = rt; height = 0; }

  AnyType element;            // The data in the node
  AvlNode<AnyType> left;      // Left child
  AvlNode<AnyType> right;     // Right child
  int height; // Height
}
```

Method to compute height of an AVL nodetree

```
/**
* Return the height of node t, or -1, if null.
*/
private int height( AvlNode<AnyType> t )
{
      return t == null ? -1 : t.height;
}
```

AVL Trees. Some Java Code

For a full version of the AVL-tree implementation please consult:

Author: Mark Weiss

http://users.cis.fiu.edu/~weiss/dsaajava/code/DataStructures/AvlTree.java