How do we balance trees?
AVL Trees
AVL Tree Definition

- **AVL trees are balanced**
- An AVL Tree is a binary search tree such that for every internal node $v$ of $T$, the heights of the children of $v$ can differ by at most 1.

An example of an AVL tree where the heights are shown next to the nodes:
Insertion in an AVL Tree

- Insertion is as in a binary search tree.
- Always done by expanding an external node.
- Example:

Before insertion:

```
   44
  /   \
32    78
 /     /\  \
17    50 88
```

After insertion:

```
   44
  /   \
32    78
 /     /\  \
17    50 88
```

Note: The figures show the tree structures before and after the insertion of a new node. The tree is balanced after each insertion.
Trinode Restructuring

- Let \((a, b, c)\) be an inorder listing of \(x, y, z\)
- Perform the rotations needed to make \(b\) the topmost node of the three

Case 1: Single rotation (a left rotation about \(a\))

Case 2: Double rotation (a right rotation about \(c\), then a left rotation about \(a\))

(other two cases are symmetrical)

AVL Trees
Insertion Example, continued

unbalanced...

...balanced
Restructuring
(as Single Rotations)

Single Rotations:

AVL Trees 7
Restructuring
(as Double Rotations)

double rotations:

```plaintext
T_0  a = z
 T_1  b = x
 T_2  c = y
 T_3

T_0  a = z
 T_1  b = x
 T_2  c = y
 T_3

T_0  a = z
 T_1  b = x
 T_2  c = y
 T_3
```

AVL Trees 8
Removal in an AVL Tree

- Removal begins as in a binary search tree, which means the node removed will become an empty external node. Its parent, \( w \), may cause an imbalance.

**Example:**

Before deletion of 32

```
        44
       /   \
      17    62
     /     /   \
    32   50   78
   /     /     /   \
  48   54  88
```

After deletion

```
        44
       /   \
      17    62
     /     /   \
    50   78
   /     /   \
  48   54
```
Rebalancing after a Removal

- Let \( z \) be the first unbalanced node encountered while travelling up the tree from \( w \). Also, let \( y \) be the child of \( z \) with the larger height, and let \( x \) be the child of \( y \) with the larger height.
- We perform \textit{restructure}(x) to restore balance at \( z \).
- As this restructuring may upset the balance of another node higher in the tree, we must continue checking for balance until the root of \( T \) is reached.
Running Times for AVL Trees

- a single restructure is $O(1)$
  - using a linked-structure binary tree
- find is $O(\log n)$
  - height of tree is $O(\log n)$, no restructures needed
- insert is $O(\log n)$
  - initial find is $O(\log n)$
  - Restructuring up the tree, maintaining heights is $O(\log n)$
- remove is $O(\log n)$
  - initial find is $O(\log n)$
  - Restructuring up the tree, maintaining heights is $O(\log n)$