6 – AVL Trees

AVL Trees
• Named after the researchers who invented them: G.M. Adelson-Velskii and E.M. Landis
• First (of many) self-balancing BSTs

AVL Tree Definition
• An AVL tree is defined by the following recursive rule:
  1. The left subtree and the right subtree differ in height by no more than one, and
  2. The subtrees are AVL trees.

Balance Factors
Height of right subtree – left subtree is the balance factor:
always -1, 0, or +1
The nodes must store their balance factor in addition to the usual data and downward linkages, so the space requirements for AVL trees are increased over standard BSTs (but still O(n)).

Analysis (Searching)
• Just a regular BST search
• ≈ 1.44 * log(n+2) in worst case
**Insertions**

1. Search, keeping track of most recent non-zero bf
2. Insert a new node as for regular BST
3. Update balance factors
4. Rebalance if AVL tree properties violated

**Cases**

1. Parent node balanced
   - Height of parent subtree increases
   - Change propagates upward; may cause out of balance condition at some higher level
2. Item added to shorter subtree
   - Parent now balanced
   - No propagation needed
3. Item added to longer subtree
   - Parent node now out of balance
   - Must rebalance starting at parent

**Rotations**

How we modify balance

This is the right rotation. The left rotation is the mirror image of this one.

**Cases**

left-left case

A single right rotation fixes things

**Cases**

Left-right case

A left rotation gives us the left-left case. Then we can do a right rotation to finish the balancing.

**Deletions**

- Similar to insertions:
  - Follow standard BST deletion rule
    - If two children, swap with maximal element on left
    - Remove item
  - This may cause an out of balance condition at or above the removed node
  - Rebalance as for insertions