### AVL Trees

**CSE 373**  
Data Structures  
Winter 2006

#### Binary Search Tree - Best Time
- All BST operations are $O(d)$, where $d$ is tree depth  
- Minimum $d$ is $d = \lceil \log_2 N \rceil$ for a binary tree with $N$ nodes  
  - What is the best case tree?  
  - What is the worst case tree?  
- So, best case running time of BST operations is $O(\log N)$

#### Binary Search Tree - Worst Time
- Worst case running time is $O(N)$  
  - What happens when you insert elements in ascending order?  
    - Insert: 2, 4, 6, 8, 10, 12 into an empty BST  
  - Problem: Lack of “balance”:  
    - Compare depths of left and right subtree  
    - Unbalanced degenerate tree

#### Approaches to balancing trees
- Don’t balance  
  - May end up with some nodes very deep  
- Strict balance  
  - The tree must always be balanced perfectly  
- Pretty good balance  
  - Only allow a little out of balance  
- Adjust on access  
  - Self-adjusting

#### Balanced and unbalanced BST

#### Balancing Binary Search Trees
- Many algorithms exist for keeping binary search trees balanced  
  - Adelson-Velskii and Landis (AVL) trees (height-balanced trees)  
  - Splay trees and other self-adjusting trees  
  - B-trees and other multiway search trees
Perfect Balance

- Want a complete tree after every operation
  - tree is full except possibly in the lower right
- This is expensive
  - For example, insert 2 in the tree on the left and then rebuild as a complete tree

**AVL - Good but not Perfect Balance**

- AVL trees are height-balanced binary search trees
- Balance factor of a node
  - height(left subtree) - height(right subtree)
- An AVL tree has balance factor calculated at every node
  - For every node, heights of left and right subtree can differ by no more than 1
  - Store current heights in each node

**Height of an AVL Tree**

- \( N(h) \) = minimum number of nodes in an AVL tree of height \( h \).
- Basis
  - \( N(0) = 1 \), \( N(1) = 2 \)
- Induction
  - \( N(h) = N(h-1) + N(h-2) + 1 \)
- Solution (Fibonacci)
  - \( N(h) > \phi^h \) \((\phi \approx 1.62)\)

**Node Heights**

- **Tree A (AVL)**
  - height: 3
  - balance factor: 2 - 1 = 1
- **Tree B (AVL)**
  - height: 1

**Worst-case AVL Trees**

- 4 nodes: \( h = 3 \) vs 3
- 7 nodes: \( h = 4 \) vs 3
- 12 nodes: \( h = 5 \) vs 4
- 20 nodes: \( h = 6 \) vs 5
- 33 nodes: \( h = 7 \) vs 6
- 54 nodes: \( h = 8 \) vs 6
- 88 nodes: \( h = 9 \) vs 7
- 143 nodes: \( h = 10 \) vs 8
- 232 nodes: \( h = 11 \) vs 8
- 376 nodes: \( h = 12 \) vs 9
- 609 nodes: \( h = 13 \) vs 10
- 986 nodes: \( h = 14 \) vs 10
- 1596 nodes: \( h = 15 \) vs 11
Insert and Rotation in AVL Trees

- Insert operation may cause balance factor to become 2 or −2 for some node
  - only nodes on the path from insertion point to root node have possibly changed in height
  - So after the Insert, go back up to the root node by node, updating heights
  - If a new balance factor (the difference $h_{left} - h_{right}$) is 2 or −2, adjust tree by rotation around the node

Insertions in AVL Trees

Let the node that needs rebalancing be $\alpha$.

There are 4 cases:

Outside Cases (require single rotation):
1. Insertion into left subtree of left child of $\alpha$.
2. Insertion into right subtree of right child of $\alpha$.

Inside Cases (require double rotation):
3. Insertion into right subtree of left child of $\alpha$.
4. Insertion into left subtree of right child of $\alpha$.

The rebalancing is performed through four separate rotation algorithms.

AVL Insertion: Outside Case

Consider a valid AVL subtree

Inserting into $A$ destroys the AVL property at node $f$
AVL Insertion: Outside Case

Outside Case Completed

AVL Insertion: Inside Case

AVL Insertion: Inside Case
AVL Insertion: Inside Case

Consider the structure of subtree $Y$...

$Y = \text{node } i \text{ and subtrees } V \text{ and } W$

AVL Insertion: Inside Case

We will do a left-right "double rotation" ...

Double rotation: first rotation

Double rotation: second rotation

Double rotation: second rotation

Balance has been restored
Example of Insertions in an AVL Tree

Insert 5, 40

Example of Insertions in an AVL Tree

Now Insert 45

Single rotation (outside case)

Double rotation (inside case)

Now Insert 34

AVL Tree Deletion

- Similar but more complex than insertion
  - Rotations and double rotations needed to rebalance
  - Imbalance may propagate upward so that many rotations may be needed.

Pros and Cons of AVL Trees

Arguments for AVL trees:
1. Search is $O(\log N)$ since AVL trees are always balanced.
2. Insertion and deletions are also $O(\log n)$
3. The height balancing adds no more than a constant factor to the speed of insertion.

Arguments against using AVL trees:
1. Difficult to program & debug; more space for balance factor.
2. Asymptotically faster but rebalancing costs time.
3. Most large searches are done in database systems on disk and use other structures (e.g. B-trees).
4. May be OK to have $O(N)$ for a single operation if total run time for many consecutive operations is fast (e.g. Splay trees).
Deletion

Deletion

Deletion

Deletion

Deletion
Deletion

Deletion

Deletion

Deletion