Today’s Outline

- Announcements
  - Assignment #2 due AT THE BEGINNING OF LECTURE, Fri, April 16, 2010.

- Today’s Topics:
  - Binary Search Trees (Weiss 4.1-4.3)
  - AVL Trees (Weiss 4.4)

The AVL Balance Condition

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

Define: balance(x) = height(x.left) − height(x.right)

AVL property: −1 ≤ balance(x) ≤ 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. Θ(2^h)) nodes
- Easy to maintain
  - Using single and double rotations

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

Is this an AVL Tree?

NULLs have height −1

Circle One:
- AVL
- Not AVL

Student Activity
- If not AVL, put a box around nodes where AVL property is violated
**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).

**Testing the Balance Property**

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

**An AVL Tree**

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.

**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.

**Bad Case #1**

- Insert(6)
- Insert(3)
- Insert(1)
Fix: Apply Single Rotation
AVL Property violated at this node (x)

Single Rotation:
1. Rotate between x and child

Single rotation in general

X < h < Y < a < Z

Height of tree before?  Height of tree after?  Effect on Ancestors?

Single rotation example

Soln:

Bad Case #3
Insert(1)
Insert(6)
Insert(3)

Fix: Apply 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
Double rotation in general

W < h < X < c < Y < a < Z

Double rotation, step 1

Height of tree before?   Height of tree after?   Effect on Ancestors?

Double rotation, step 2

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

Insert into an AVL tree: a b e c d

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: Circle your final answer
**Insertion into AVL tree**

1. Find spot for new key
2. Hang new node there with this key
3. Search back up the path for imbalance
4. If there is an imbalance:
   - case #1: Perform single rotation and exit
   - case #2: Perform double rotation and exit

   Both rotations keep the subtree height unchanged. Hence only one rotation is sufficient!

---

**Easy Insert**

Insert(3)

---

**Hard Insert**

Insert(33)

---

**Single Rotation**

---

**Hard Insert**

Insert(18)

---

**Single Rotation (oops!)**
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 ?

AVL Trees Revisited

- What extra info did we maintain in each node?

- Where were rotations performed?

- How did we locate this node?