Logistics

Reading

- Weiss Ch 19.4: AVL Trees
- Weiss Ch 19.5: Red-Black Trees

Today

Tree Rotations: Balancing via pointer manipulation
From Last Time

- What’s a good way to organize a binary tree?
- Once organized, what kind of ops are there to do?
- Describe each of those ops
  - 2 were relatively easy
  - 1 was more difficulty: why?
- Given a BST, how do I print its elements in sorted order?
Practice/Demo Sites

▶ jGrasp is so-so for seeing tree operations
▶ Play with Blanced Binary Search Trees online using the following applets (titles hyperlinked)

Qmatica: Pretty AVL
▶ Standard BSTs
  ▶ Manual Rotation
  ▶ Great Practice
▶ AVL Trees
▶ Undo/Redo to rewatch
▶ Step by step logging

Adjustable Demo (broken)
▶ Standard BSTs
▶ All three Balanced
  ▶ AVL, Red black, Splay
▶ Slow down, pause, show balance factors

Scaling AVL (broken)
▶ AVL Tree only
▶ Scaling view for large trees
Why Worry About Insertion and Removal?

- **Q:** Why worry about insert/remove messing with the tree? What affect can it have on the performance of future ops on the tree?
- **Q:** What property of a tree dictates the runtime complexity of its operations?
Balancing Trees

- add/remove/find complexity $O(\text{height}(t))$
- Degenerate tree has height $N$: a linked list
- Prevent this by \textit{re-balancing} on insert/remove
- Several kinds of trees do this
  - AVL left/right subtree height differ by max 1
  - Red-black preserve 4 red/black node properties
    - AA red-black tree + all left nodes black
  - Splay amoritized bound on ops, very different
Rotations

Re-balancing usually involves

- Drill down during insert/remove
- Follow path back up to make adjustments
- Adjustments even out height of subtrees
- Adjustments are usually rotations
- Rotation changes structure of tree without affecting ordering
Single Rotation Basics

Right Rotation
Rotation node becomes the right subtree

Left Rotation
Rotation node becomes the left subtree
Fixing an Insertion with a Single Rotation

Insert 1, perform rotation to balance heights

- Right rotation at 8

(a) Before rotation

(b) After rotation
Single Rotation Practice

Problem 1
- 40 was just inserted
- Rebalance tree rooted at 16
- Left-rotate 16

Problem 2
- 85 is being removed
- Rebalance tree rooted at 57
- Right rotate 57
Single Rotations Aren’t Enough

Can we fix the following with a single rotation?

(a) Before rotation

(b) After rotation
Example: Can’t fix this with single rotation
Double Rotation Overview

Left-Right

- Left Rotate at $k_1$
- Right-rotate at $k_3$

Right-Left

- Right Rotate at $k_3$
- Left Rotate at $k_2$
Fixing an Insertion with a Double Rotation

Insert 5, perform rotation to balance heights

(a) Before rotation
(b) After rotation
Double Rotation Practice

Problem 3

- 35 was just inserted
- Rebalance the tree rooted at 36
- Use two rotations, at 33 and 36
- 36 should move
weiss/nonstandad/Rotations.java contains

1. static <T> BinaryNode<T> rotateWithLeftChild( BinaryNode<T> k2 )
2. static <T> BinaryNode<T> rotateWithRightChild( BinaryNode<T> k1 )
3. static <T> BinaryNode<T> doubleRotateWithLeftChild( BinaryNode<T> k3 )
4. static <T> BinaryNode<T> doubleRotateWithRightChild( BinaryNode<T> k1 )

- 1 and 2 are 4 lines
- 3 and 4 are 2 lines (call 1 and 2)
- Write them
- What is the runtime complexity of each rotation?
Node<T> rightRotate( Node<T> t ) {
    Node<T> newRoot = t.left;
    t.left = newRoot.right;
    newRoot.right = t;
    return newRoot;
}

weiss/nonstandad/Rotations.java contains similar code but names the above rotateWithLeftChild(k2)
  ▶ More or less intuitive name?