Announcements

Test #1 on Friday in class. If you need special considerations (extra time, quiet space, different time, etc.) talk to me today either via email or in office hours.

Lab 4 grades just sent out at 9:30 this morning.

Test topics:

- Advanced Java:
  - Abstract classes and interfaces, inheritance, exceptions, Javadoc, JUnit

- Algorithms analysis:
  - Definition of Big-O, big-Omega, little-O, little-omega, and Big Theta
  - Determine the running time of various algorithms (including log n ones)

- Linear and binary search, selection, insertion, and merge sort

- Data structures: operations, implementations, running times.
  - Arrays, stacks, queues, linked lists, doubly linked lists, binary trees

Review of AVL Trees

An AVL Tree is a binary search tree such that for every node, the height of the left and right subtrees differ by at most 1, and the right child is an AVL tree.

We will show that this height restriction guarantees that the height of the whole tree is $O(\log n)$ (this will take a bit of proof-work, but it's true). In particular, we need to re-implement our adds and removes, because they are the methods that mess with our tree, and can potentially unbalance it. The idea is: do our usual insert (as a leaf) or remove, if the new tree is still balanced, great! Otherwise we've unbalanced it, and we need to rebalance.

AVL Rebalancing

To rebalance an AVL tree, do the following steps:

1. Find the smallest unbalanced subtree $Z$, and label its root $Z$.
   - (Note: Find $Z$ by starting at the leaf we inserted or deleted, and move upwards towards the root, checking unbalanced-ness as we go. Since $Z$ is the smallest unbalanced tree, its children are balanced AVL trees.)

2. Let $Y$ be Z’s tallest child.
3. Let $X$ be $Y$’s tallest child.
4. Let a, b, c be data of X, Y, Z in increasing order.
5. Replace the subtree at Z with the subtree at b drawn below:

```
       b
      /|
     / |
    a  c
```

CS 151

AVL Trees
Starting from the empty tree, draw the trees after the following operations:

- **insert 50, 25, 75, 60, then 80.**
- **insert 90.** (requires a rotation)
- **remove 90.**
- **insert 65.** (requires a rotation)

**Notes on rebalancing:**

- Inserts and removes always happen at the leaves.
- If a tree becomes unbalanced, it’s at some node along the leaf-root path.
- The length of this path is O(log n).
- Each rebalance is an O(1) operation.
- We may have to rebalance more than once along the O(log n) path.
- The height of this path is still O(log n).

**To prove that we have correctly rebalanced the tree rooted at Z:**

**After the rotation:**
- t0, t1, t2, and t3 have the same children before, so all are balanced.
- Y has children of height h(t0) and h(t1), which are within one, so balanced.
- Z has children of height h(t2) and h(t3), which are within one, so balanced.
- X has children of height h(t0) and h(t1), which are within one, so balanced.
- Z is balanced before the add/remove: h(t2) = {h(t3), h(t3)+1}.
- X is Y’s tallest child, so h(t0) = {h(t1), h(t1)+1}.
- Similarly, h(t2) = {h(t3), h(t3)-1} <= h(t3).

**Before the rotation:**
- X and Y were balanced but Z was not.
- Z was balanced before the insert/remove: h(t2) = {h(t3)+1, h(t3)}.
- X is Y’s tallest child, so h(t0) = {h(t1), h(t1)-1}.
- Similarly, h(t2) = {h(t3), h(t3)+1}.

**Finally, why does this rebalancing maintain the binary search-order property?**

- X unchanged, so it still has the order property.
- Z balanced before the add/remove: h(t2) = {h(t3), h(t3)+1}.
- Y and Z are in a BST, so h(t2) <= h(t3) = h(X)+1.
- Z = Y’s parent, so h(t2) <= h(t3) = h(X)+1.

**Case 1: val(X) < val(Y) < val(Z) (val(X) > val(Y) > val(Z) is symmetric)**

**proof continued:**

- Before the rotation: X and Y were balanced but Z was not.
- X is Y’s tallest child, so h(t0) = {h(t1), h(t1)+1}.
- Similarly, h(t2) = {h(t3), h(t3)-1} <= h(t3).

**Notes on rebalancing:**

- Insert 65 (requires a rotation).
- Remove 90.
- Insert 90 (requires a rotation).
- Insert 20, 75, 60. Then 80.

Starting from the empty tree, draw the trees after the following operations: