Java 2 Red-Black Trees

- **Comparing R-B to AVL trees**
  - Max height:
    - AVL
      - ~1.44Log_2N
    - R-B
      - 2log_2(N + 1)
  - Insertion
    - AVL
      - Insert according to value, then check balance with a reverse walk.
        - bf = +/- 2?
    - R-B
      - Insert according to value, then check balance with a reverse walk.
        - red-red?
  - If unbalanced
    - AVL
      - Identify a 3-node neighborhood
    - R-B
      - Identify a 4-node neighborhood
  - Repair
    - AVL
      - Rotations
        - At most one repair per insertion
        - Many repairs possible for deletion
    - R-B
      - Rotations and recoloring
        - Many repairs possible for insertion
        - Many repairs possible for deletion

- **Red-Black Trees**
  - A red-black is BST with the following node color rules
    - Each node is either red or black.
    - The root and all empty trees are black.
    - All paths from the root to an empty tree contain the same number of black nodes.
    - A red node can't have a red child.
  - Rule 1 tells us what types of nodes are legal: red ones and black ones.
  - Rule 2 specifies the root must be black and, since empty trees are valid trees, it fives them a color (black). We know what the "boundaries" of a R-B tree look like.
  - Rule 3 + 4 = Balance
    - Rule 3 is half of the balance requirement. It makes a statement about the height of tree in terms of black node. This is often called the tree's black height.
    - Without red nodes, R-B tree could only be full.
    - A red node is used like "filler". It allows a R-B tree to obey rules 1, 2, and 3 without being a perfect triangle
    - Rule 4 prevents a red node from having a red child.
  - Use the standard BST algorithm to insert the new node. Make the addition red.
    - Walk up the tree and look for a red node with a red parent
    - Stop at the first (lowest) red node that has a red parent. Go the grandparent, then its other child (This is the 4 node neighborhood)
- **5 Cases for Repair**
  - A is red
    - Repaired by only recoloring nodes.
    - Re-color the top three nodes in the neighborhood (toggle their state)
  - A is black
    - Repaired by rotations and recoloring Using the same scheme as an AVL tree

- Case 1 from AVL (zig-zag left - right)
    - rotate left
- Case 2 from AVL (left)
    - rotate right
- Case 3 from AVL (zig-zag right - left)
    - rotate right
- Case 4 from AVL (right)
    - rotate left