

# Red Black Trees + Tree Review

---

Nathan Sprague

Fall, 2021

# Red-Black Trees

- Another self-balancing binary search tree.
- Five Rules:
  - All nodes are labeled either red or black
  - The root must be black
  - All (empty) leaves are black
  - If a node is red, all children are black
  - Every path from the root to a leaf contains the same number of black nodes
- The **black height** of a tree is the number of black nodes on any path from the root to a leaf.
- The **black depth** of a node is the number of black nodes from the root to that node.

# Socratic

- Is this a valid red-black tree?

- A) No – violates 1 rule
- B) No – violates 2 rules
- C) No – violates 3 rules
- D) No – violates 4 rules
- E) No – violates 5 rules
- F) Yes

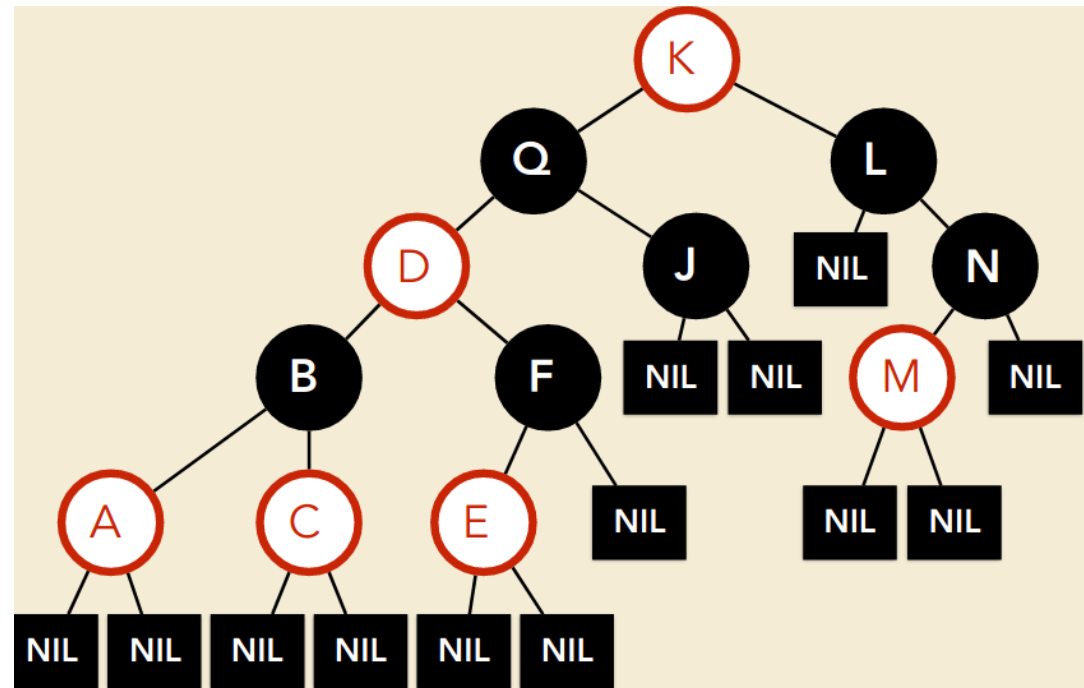


Image credit: Michael Kirkpatrick

# Socratic 2

- The left subtree of the root of a particular red-black tree has a black height of 12. Which of the following could be the *height* of the right subtree.
  - A) 0
  - B) 10
  - C) 20
  - D) 30
  - E) None of the above are possible
  - F) Any of the above are possible

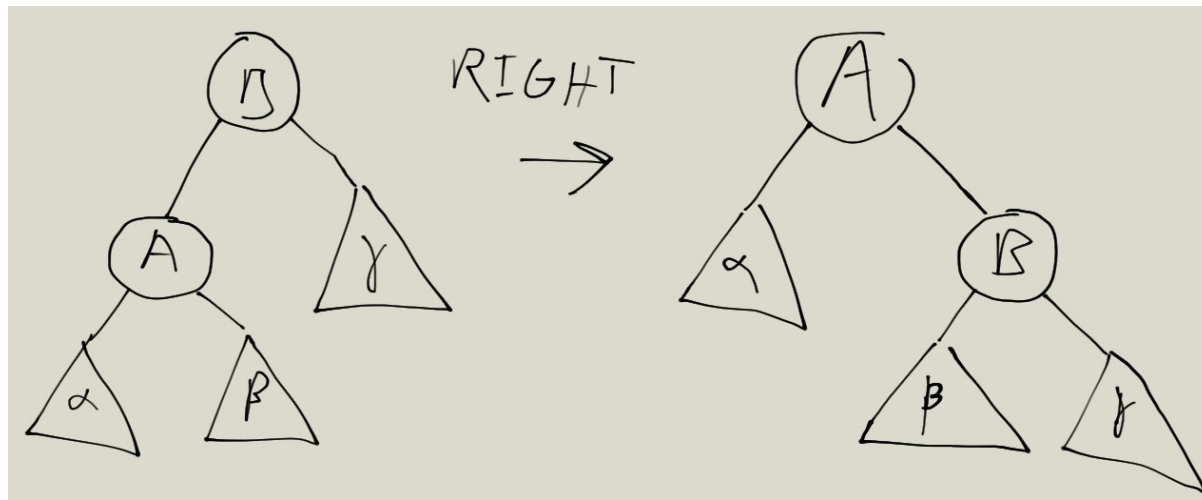
# Insertion/Removal

- Newly inserted nodes are colored red
- Perform rotations and recolorings to restore the red-black tree properties
- (We won't worry about the details of insertion and removal)

# Red-Black vs. AVL

- Both ensure  $O(\log n)$  insertion, removal and lookup.
  - Max depth of a red-black tree:  $2 \log_2(n+1)$
  - Max depth of an AVL Tree:  $\approx 1.44 \log_2(n+2) - 3.28$
- AVL Trees are shorter by a constant factor, but require more rotations.
- Java's [TreeMap](#) and [TreeSet](#) use red-black trees.

# Rotation Reminder



- This does not change the in-order traversal order  
 $\alpha$  **A**  $\beta$  **B**  $\gamma$

# Exercise (if time)

Insert B, A, F, E into the AVL tree on the right

