Red-Black Trees

An AVL tree is a binary search tree such that
• the heights of the left and right subtrees differ by at most 1,
• the left child is an AVL tree,
• the right child is an AVL tree,
• the heights of the left and right subtrees differ by at most 1,

A red-black tree is another balanced BST, easier than AVLs in practice.

Ex.

Is it still a red-black tree if we:
• change the 22 to black? 22 and 27?
• color all the nodes black?

Less restrictive than AVLs, so rebalancing is easier but height is slightly worse.

Operations same as for AVL trees, same big-O runtime.

A red-black tree is a binary search tree such that:
• every node is either red or black,
• the root is black,
• the children of a red node are black,
• for each node v, all v -> null pointer paths contain the same number of black nodes,

The right child is an AVL tree.

Red-Black Trees
We didn't cover the following slides in class, but I'm including them in case you want a bit more information about red-black tree insertion and deletion.

Red-Black Tree Insertion

As with binary search trees and AVL trees, always insert as a leaf node.

• if this new leaf is the root, color it black (to maintain the root property)
• otherwise, color the new leaf node red (to maintain the depth property)
• if this node's parent is black, we're done.
• otherwise, this node's parent is red and we violated the internal property

• grandparent G must be black (because had RB tree before insertion)
• if this node's parent is the root (to maintain the depth property)
• otherwise, color this node red (to maintain the root property)
• if this new leaf is the root, color it black (to maintain the root property)
• if both red, swap to block, make root red
• if both red, swap to block, make root red
• all paths still have same number of black nodes
• if root is root of whole tree, keep it black
• (if root is root of whole tree, keep it black)
• but G's parent may also be red: if so, we know (this affects all paths by 1, no prob.)
• (this affects all paths by 1, no prob.)

We avoided this case completely: When searching for insertion point, at each node, check children's colors:

• if both red, swap to block, make root red
• if both red, swap to block, make root red
• all paths still have same number of black nodes
• if root is root of whole tree, keep it black
• (if root is root of whole tree, keep it black)
• no prob.
• (if root is root of whole tree, keep it black)
• (if root is root of whole tree, keep it black)
• but G's parent may also be red: if so, we know (this affects all paths by 1, no prob.)
• (this affects all paths by 1, no prob.)

Note: we may have to rotate multiple times along the path from the insertion point up to the root, so t0 and t1 are not necessarily empty.

(If X is a leaf then S is empty (depth prop) and this still works.)

For X is a leaf then S is empty (depth prop) and this still works.)

We avoided this case completely: When searching for insertion point, at each node, check children's colors:

• if both red, swap to block, make root red
• if both red, swap to block, make root red
• all paths still have same number of black nodes
• if root is root of whole tree, keep it black
• (if root is root of whole tree, keep it black)
• but G's parent may also be red: if so, we know (this affects all paths by 1, no prob.)
• (this affects all paths by 1, no prob.)

As with binary search trees and AVL trees, always insert as a leaf node.

Red-Black Tree Insertion

Extra Material
Red-Black Tree Deletion

For deletion, use normal BST algorithm until we remove a leaf or single child:

- If node is black, removing it could violate the red property. In that case, we proceed as follows:
  - If node is red, delete it and we're done (maintains depth, internal prop).
  - If node is black, we need to maintain the red property while removing it. To do this, we re-color the tree as we move down it.
  - The goal is to maintain that node to be removed is red.

Let X be the node we possibly want to remove, P its parent, S its sibling.

If X has 2 black children, more case work: decompose into t0, t1, t2, t3.

If one of X's children are red, proceed with case work: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR