CS 315 - Day 25, Red-Black Trees 1

A BST of height h supports the basic dynamic set operations like Search, Successor/Predecessor, Minimum/Maximum, Insert, and Delete, in $O(h)$ time.

Now, we will explore making $h = O(\lg N)$.

A Red-Black (RB) Tree is a BST with an additional attribute, a color.

A tree satisfying the RB properties is guaranteed not to have a path from the root to the leaf more than twice as long as a path from the root to any other leaf.

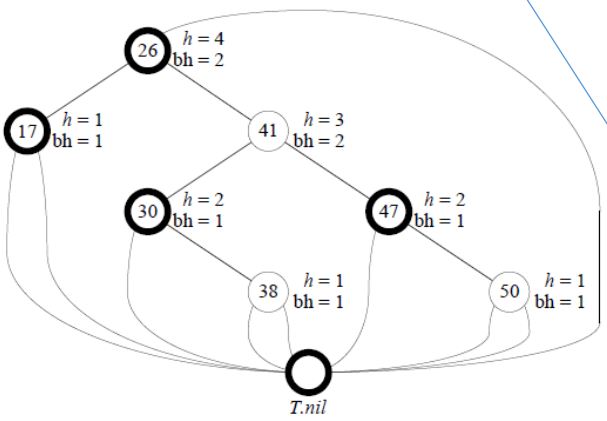
Red-Black Properties

1. Every node is either **red** or **black**.
2. The root is **black**.
3. Every leaf (NIL) is **black**.
4. If a node is **red**, then both its children are **black**.
5. For each node, all simple paths from the node to descendant leaves contain the same number of **black** nodes.

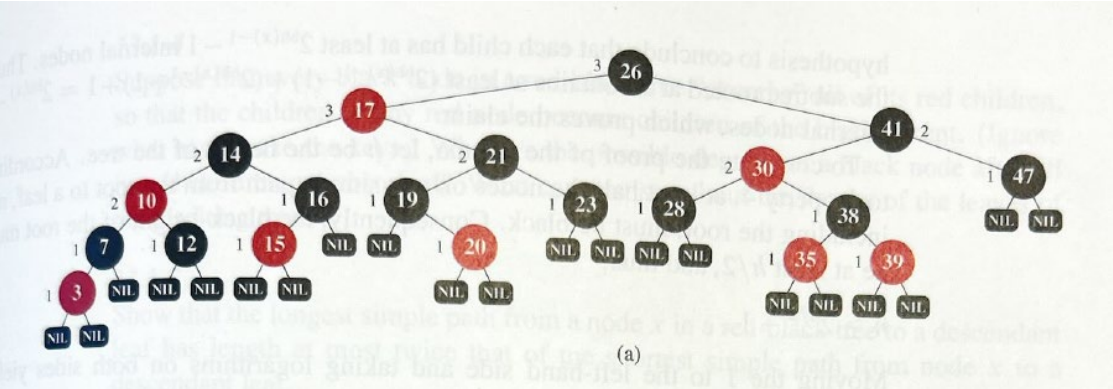
Note that as we draw our tree diagrams, we'll include, and count, the **black** NIL nodes.

Height of a red-black tree

- ***Height of a node*** is the number of edges in a longest path to a leaf.
- ***Black-height*** of a node x: $\text{bh}(x)$ is the number of black nodes (including $T.nil$) on the path from x to leaf, not counting x. By property 5, black-height is well defined.



When counting the $bh(x)$ of a node, ***DO NOT COUNT YOUR OWN COLOR***.



So, while doing operations, if the root is red, does changing it to black change the bh of the tree?

No, since the color of the root isn't counted in its bh. And the bh of the root is equal to the bh of the tree.

A height h node is guaranteed to have $bh = \frac{h}{2}$.

Proof of claim By property 4, $\leq h/2$ nodes on the path from the node to a leaf are red. Hence $\geq h/2$ are black. ■ (claim)

Later, we will prove the big result, that A red-black tree with n internal nodes has height $\leq 2\lg(n+1)$

The non-modifying binary-search-tree operations MINIMUM, MAXIMUM, SUCCESSOR, PREDECESSOR, and SEARCH run in O(height) time. Thus, they take $O(\lg n)$ time on red-black trees.

Insertion and deletion are not so easy.

If we insert, what color to make the new node?

- Red? Might violate property 4.
- Black? Might violate property 5.

Show some examples of insertion on the RB Trees on the board.

Note that we will focus on insertion into RB Trees.

To keep an RB Tree balanced, and maintaining the RB Tree Properties, we will need to update the structure/shape of the RB Trees.

We'll do this with ***rotations***.

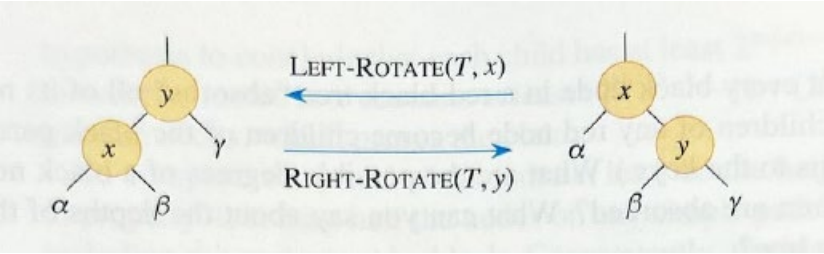
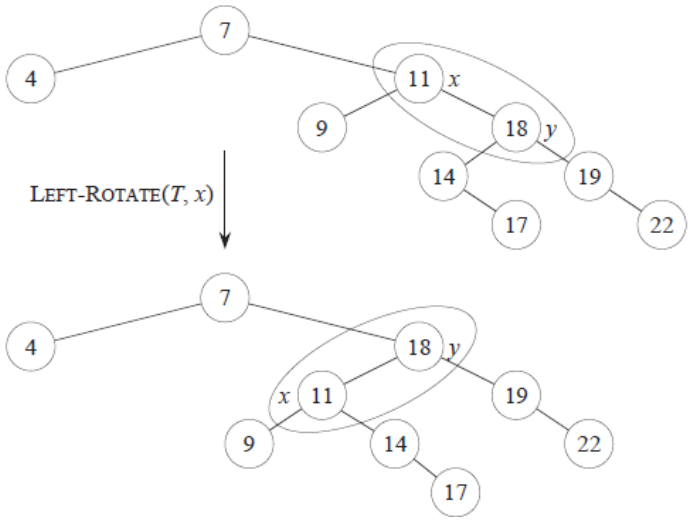


Figure 13.2 The rotation operations on a binary search tree. The operation LEFT-ROTATE(T, x) transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation RIGHT-ROTATE(T, y) transforms the configuration on the left into the configuration on the right. The letters α, β, and γ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in α precede $x.key$, which precedes the keys in β, which precede $y.key$, which precedes the keys in γ.

```
LEFT-ROTATE(T, x)
1   y = x.right
2   x.right = y.left        // turn y's left subtree into x's right subtree
3   if y.left ≠ T.nil       // if y's left subtree is not empty ...
4       y.left.p = x        // ... then x becomes the parent of the subtree's root
5   y.p = x.p               // x's parent becomes y's parent
6   if x.p == T.nil         // if x was the root ...
7       T.root = y          // ... then y becomes the root
8   elseif x == x.p.left    // otherwise, if x was a left child ...
9       x.p.left = y        // ... then y becomes a left child
10  else x.p.right = y      // otherwise, x was a right child, and now y is
11  y.left = x              // make x become y's left child
12  x.p = y
```

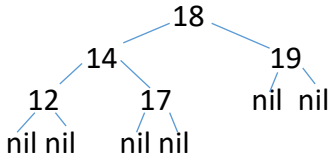
The operations here are all at constant time, so the asymptotic runtimes of the rotates is $O(1)$.

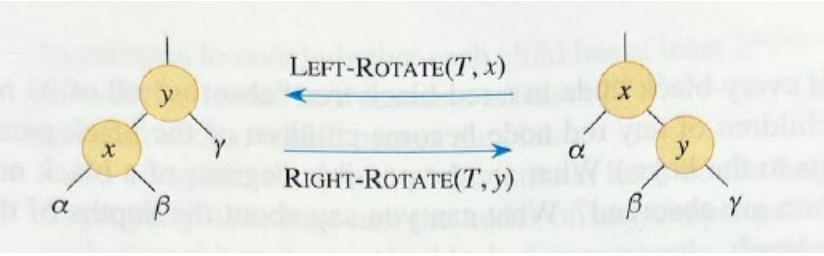


Note that we aren't considering node colors here, just the structure of the BST, and the rotations preserve the BST Tree Properties.

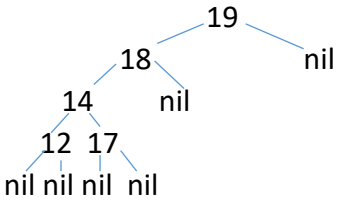
<u>In-class w/***assigned groups of four (4)***, write 0. and 1. on board as ground rule(s):</u>

0. No devices.
1. Intro: Name, POE, Icebreaker.
2. Pick someone on the group who will report out.
3. On the given BST, do a left rotation on nodes 18 and 19.
4. On the given BST, do a right rotation on nodes 14 and 18.

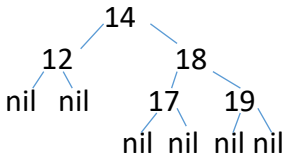




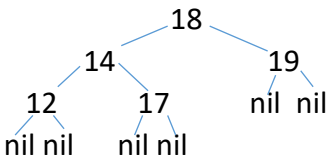
Left-rotation



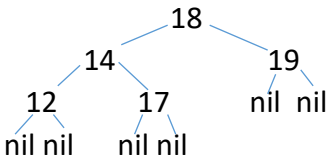
Right-rotation

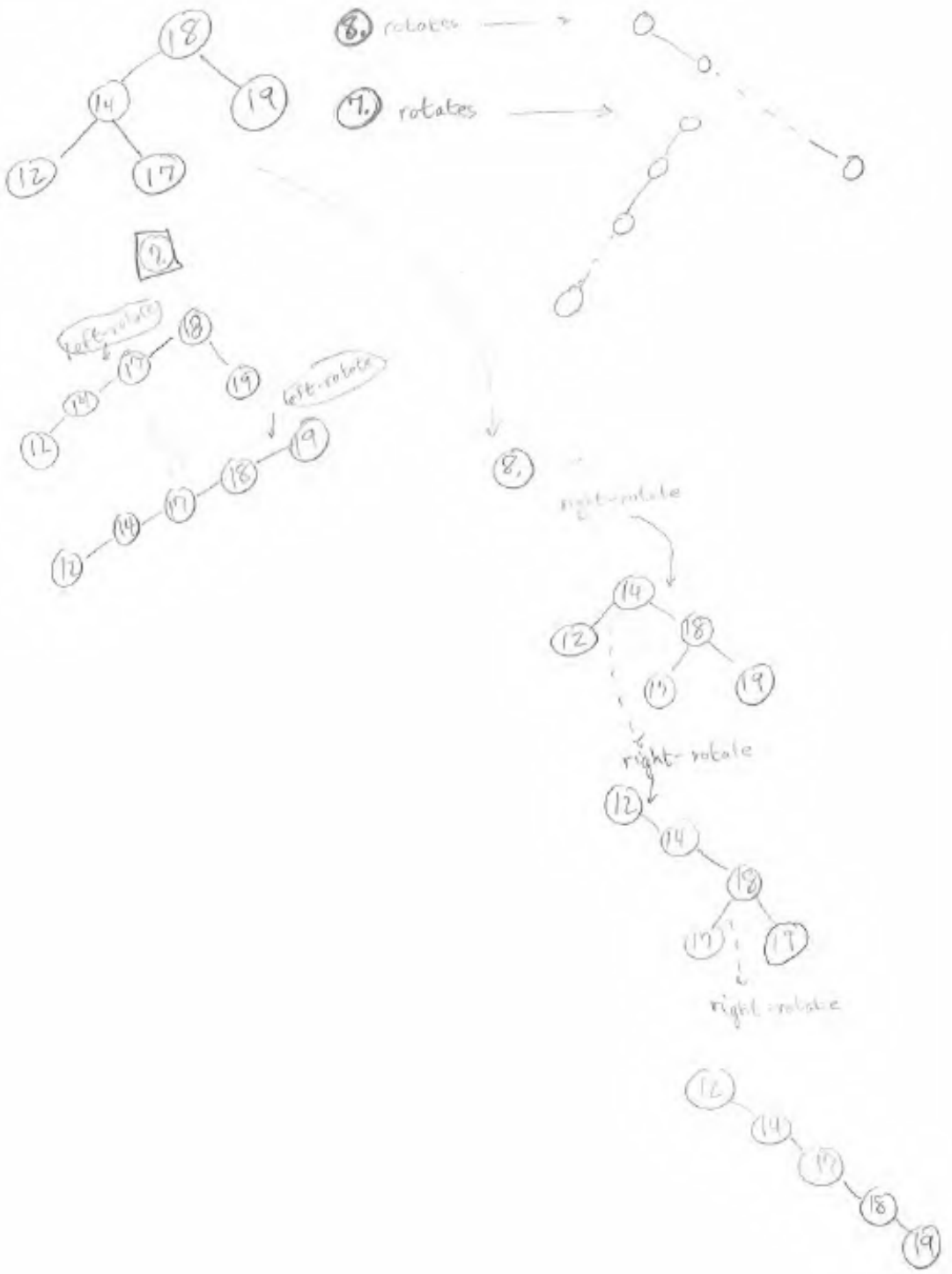


5. On the orignal BST, do a series of rotations to give a tree where all non-NIL nodes, besides the root, are ***LEFT*** children.



6. On the orignal BST, do a series of rotations to give a tree where all non-NIL nodes, besides the root, are ***RIGHT*** children.




18
14
19
12
17
9.
left-rotate
18
17
19
14
12
left-rotate
19
18
17
14
12
8.
rotates
7.
rotates
8.
right-rotate
14
12
18
17
19
right-rotate
12
14
18
17
19
right-rotate
12
14
17
18
19