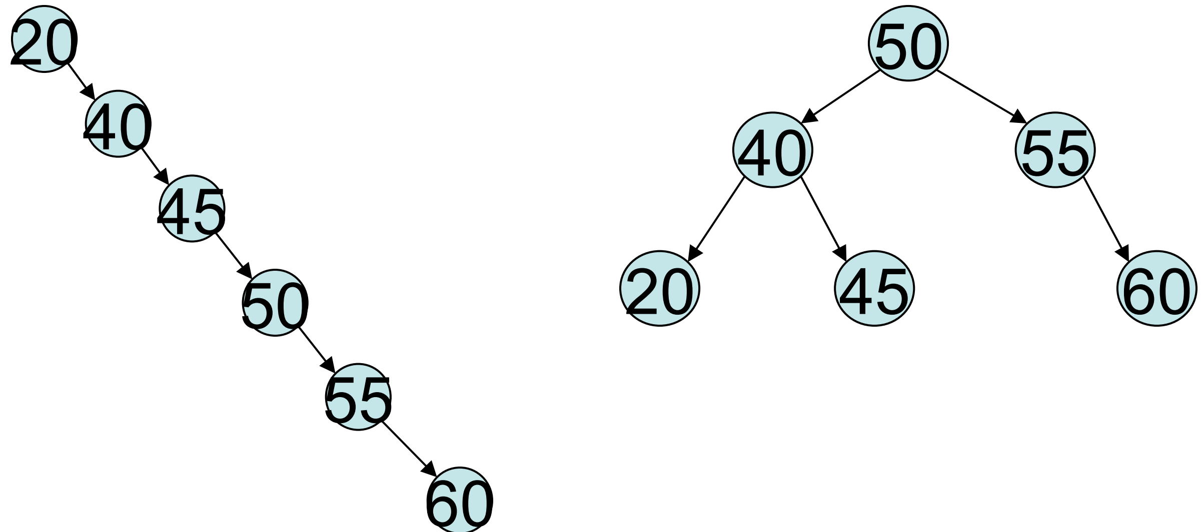
ECE 2574
Introduction to Data Structures and Algorithms

37: Balancing Trees: Red-Black Trees

Chris Wyatt
Electrical and Computer Engineering
Virginia Tech

The average complexity (number of comparisons) when searching a BST is best when the tree is balanced.

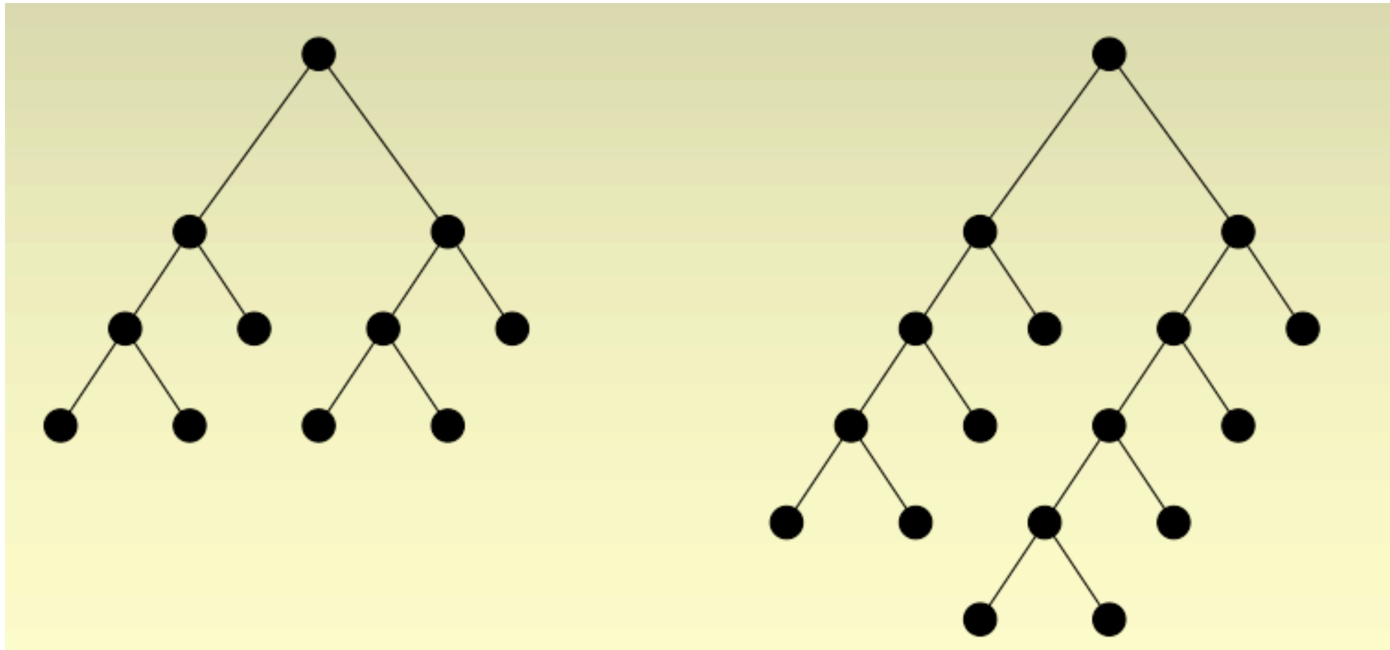
So the question naturally arises, can we make a BST balanced?



Review: Balanced Trees

Recall, a tree of height h is balanced if it is full down to level h-1;

and the depth of a tree was the number of nodes from the root to a leaf.



Basic approach to making a balanced binary tree.

1. Insert/Delete a node
2. Restore the balance of the tree.

The primary tool used to restore balance is called a rotation.

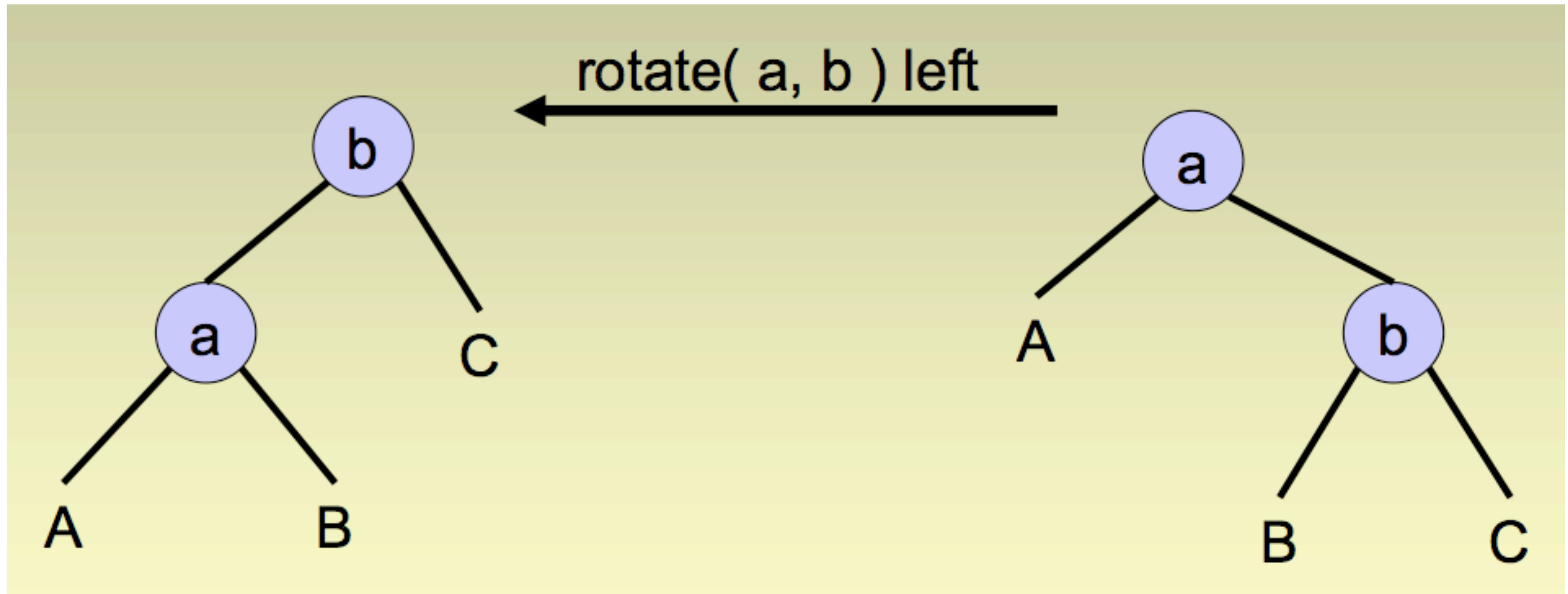
There are left and right rotations

and

the rotation should not violate the binary tree property.

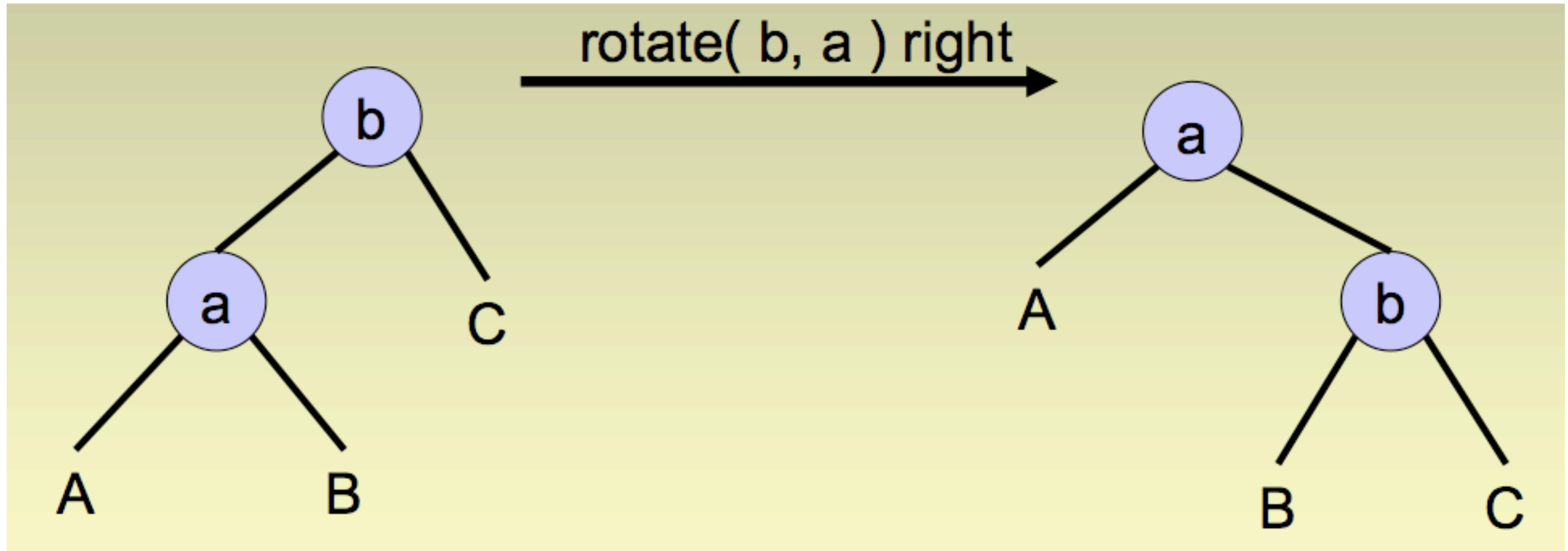
Review: Left rotation

Let A, B, and C be subtrees and a, b nodes in the following tree.



Review: Right rotation

Let A, B, and C be subtrees and a, b nodes in the following tree.



The two most popular balanced Binary Trees are the AVL and the Red-Black Tree.

AVL trees (named after Adel'son-Velsk'ii and Landis who introduced them in 1962) require that the depths of the left and right subtrees of any node differ by at most one.

The rules to enforce the AVL property are complex.

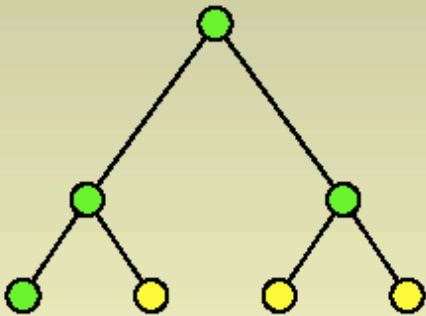
The two most popular balanced Binary Trees are the AVL and the Red-Black Tree.

A related tree, introduced by Bayer in 1972, called red-black trees (symmetric binary B-trees) have the property: no path from the root to the leaf has length more than twice the length of any other path.

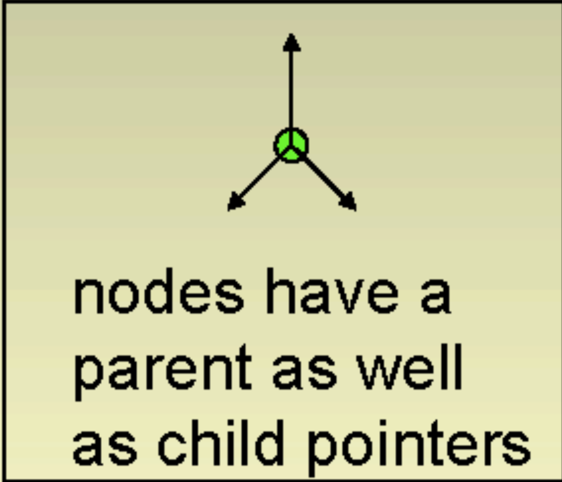
Red-black trees are much easier to implement.

Definition of a red-black tree

A *red-black tree* is a binary search tree, augmented by leaf nodes corresponding to an unsuccessful search.



- node in the search tree
- augmented node



Each node has been assigned either the color red or black subject to three properties.

Coloring properties of the red-black tree.

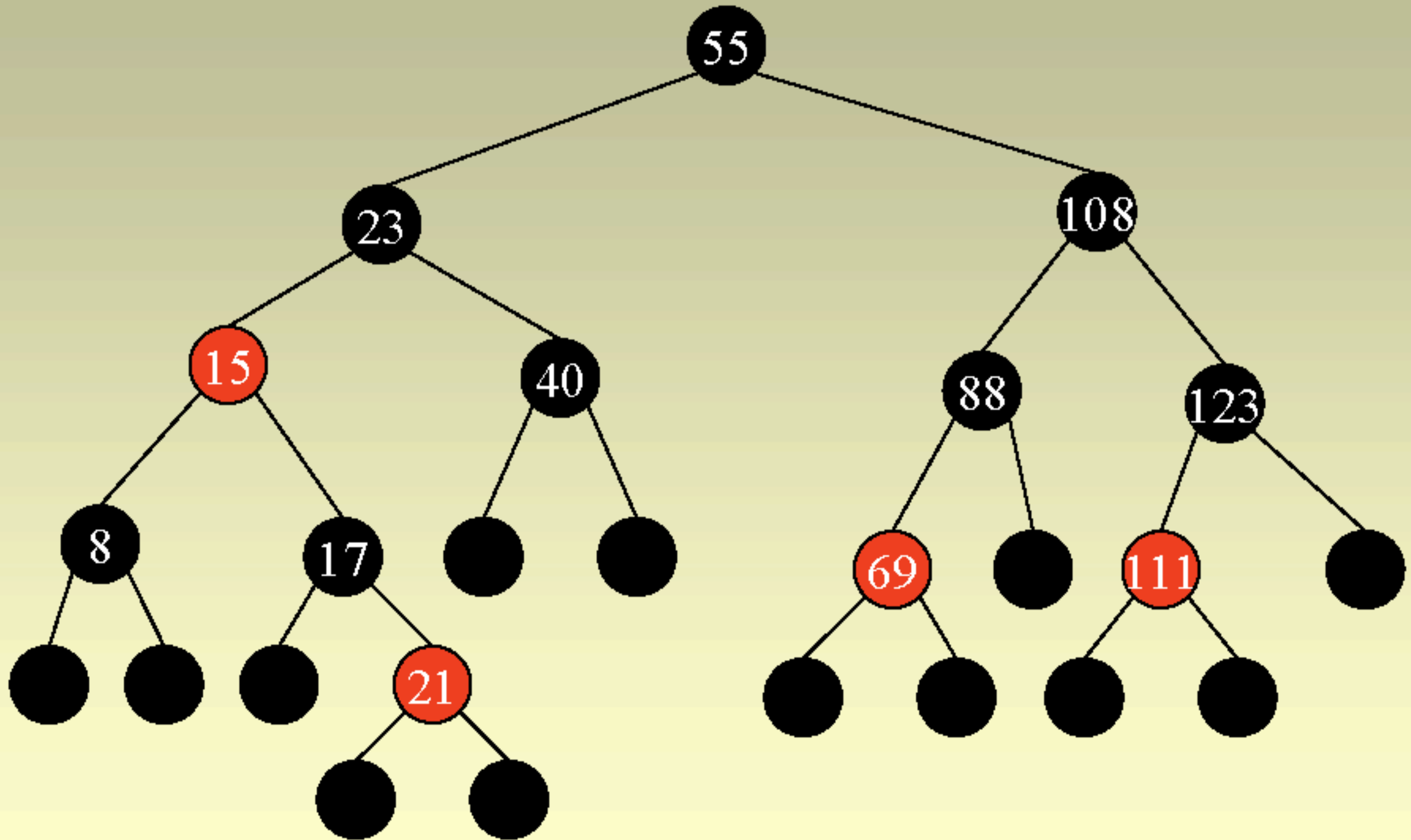
Each node is assigned the color red or black based on the following properties:

1. Every leaf node is black
2. If a node is red, then both its children are black
3. Any two paths from a given node down to a leaf node contain the same number of black nodes.

As a consequence of this coloring scheme, the maximum depth of the red-black tree T with n nodes is at most twice the minimum depth.

$$\text{depth}(T) <= 2 \log_2(n+1)$$

Example red-black tree (12 internal nodes)



Modifications necessary to BST to implement a red-black tree.

Add special augmented nodes corresponding to unsuccessful searches.

Add a color to the definition of TreeNode.

Add a parent pointer to TreeNode.

Modify the insert and delete operations to maintain the parent pointers and augmented nodes.

Inserting into a red-black tree

First call modified BST insert and color the inserted node red.

If the parent of the inserted node is black, we are done.

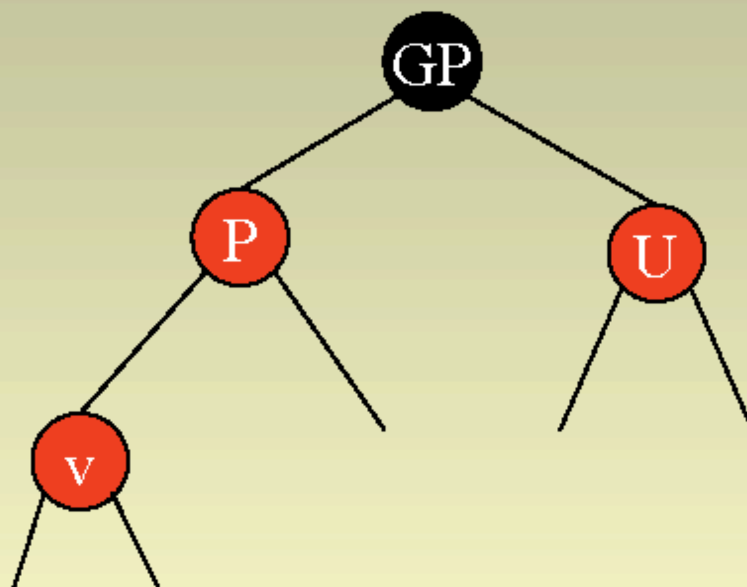
Else, property 2 is violated:

2. If a node is red, then both its children are black

At most two rotations and some re-coloring restores property 2 with violating the other properties.

For any node v in a red-black tree, we define the *local neighborhood* of v as follows:

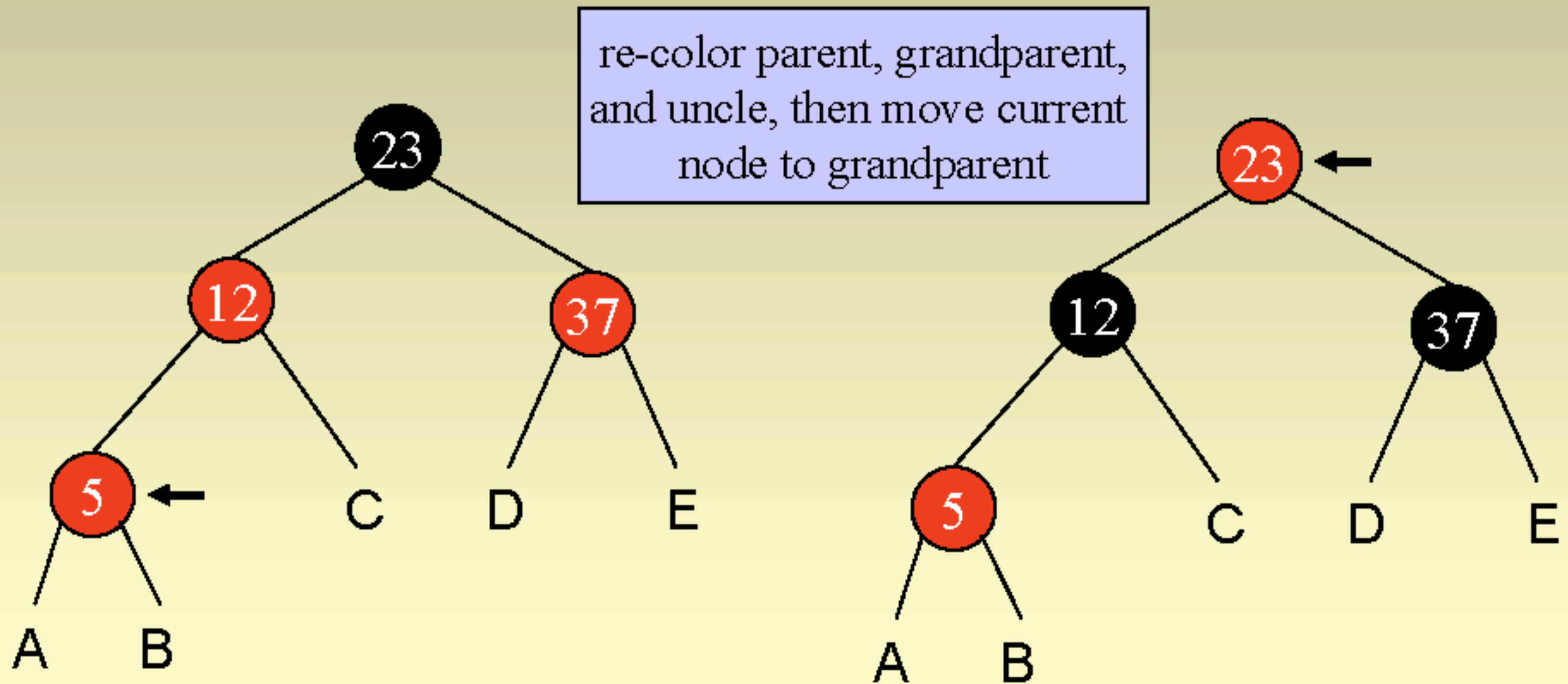
The local neighborhood of v is v with its parent grandparent and uncle.



Restoring property 2 involves re-coloring and (possibly) rotating nodes in the local neighborhood.

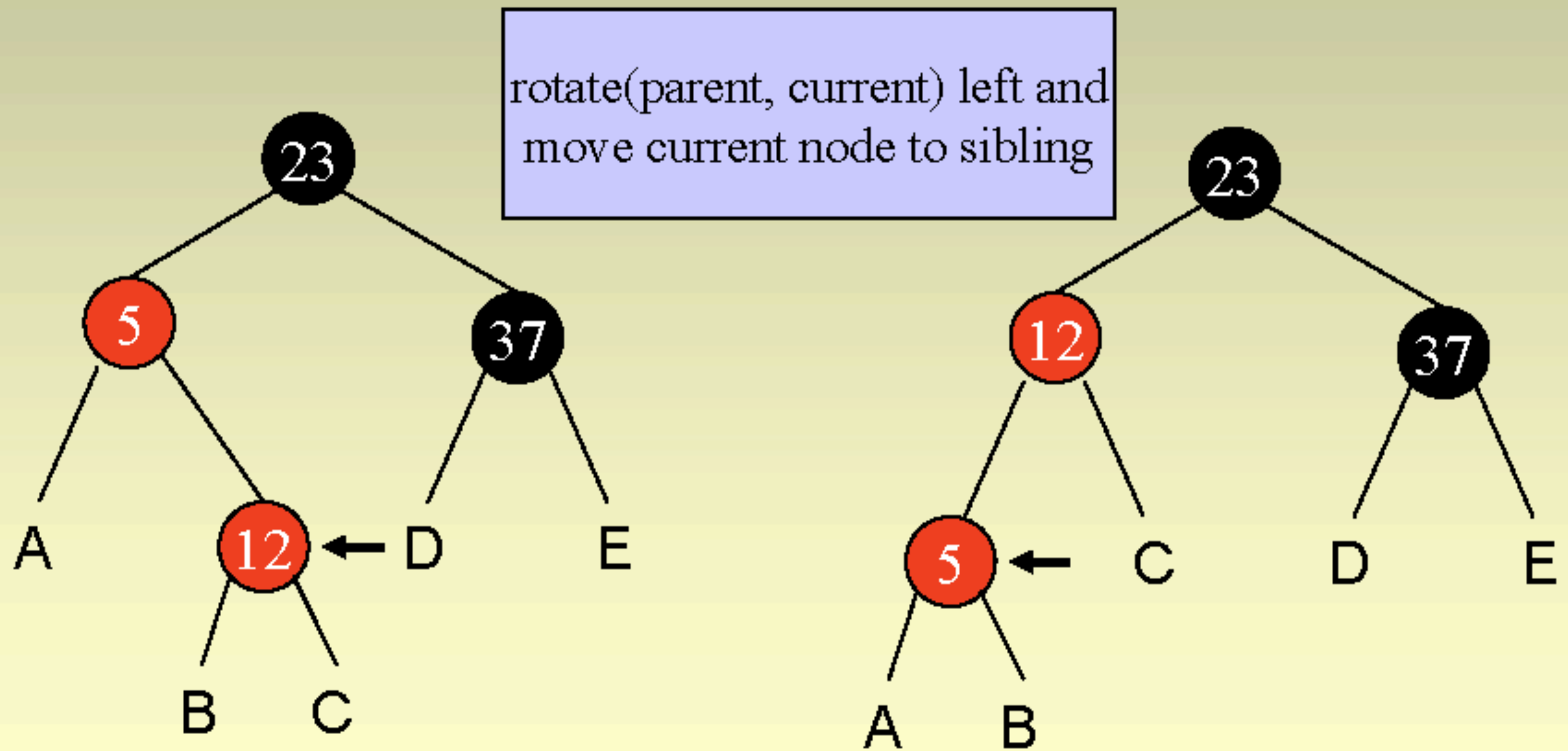
For each restoration on insert there are 6 cases, 3 of which are symmetric to the other three.

Case 1: the current node has a red uncle and its parent node is a red left child.



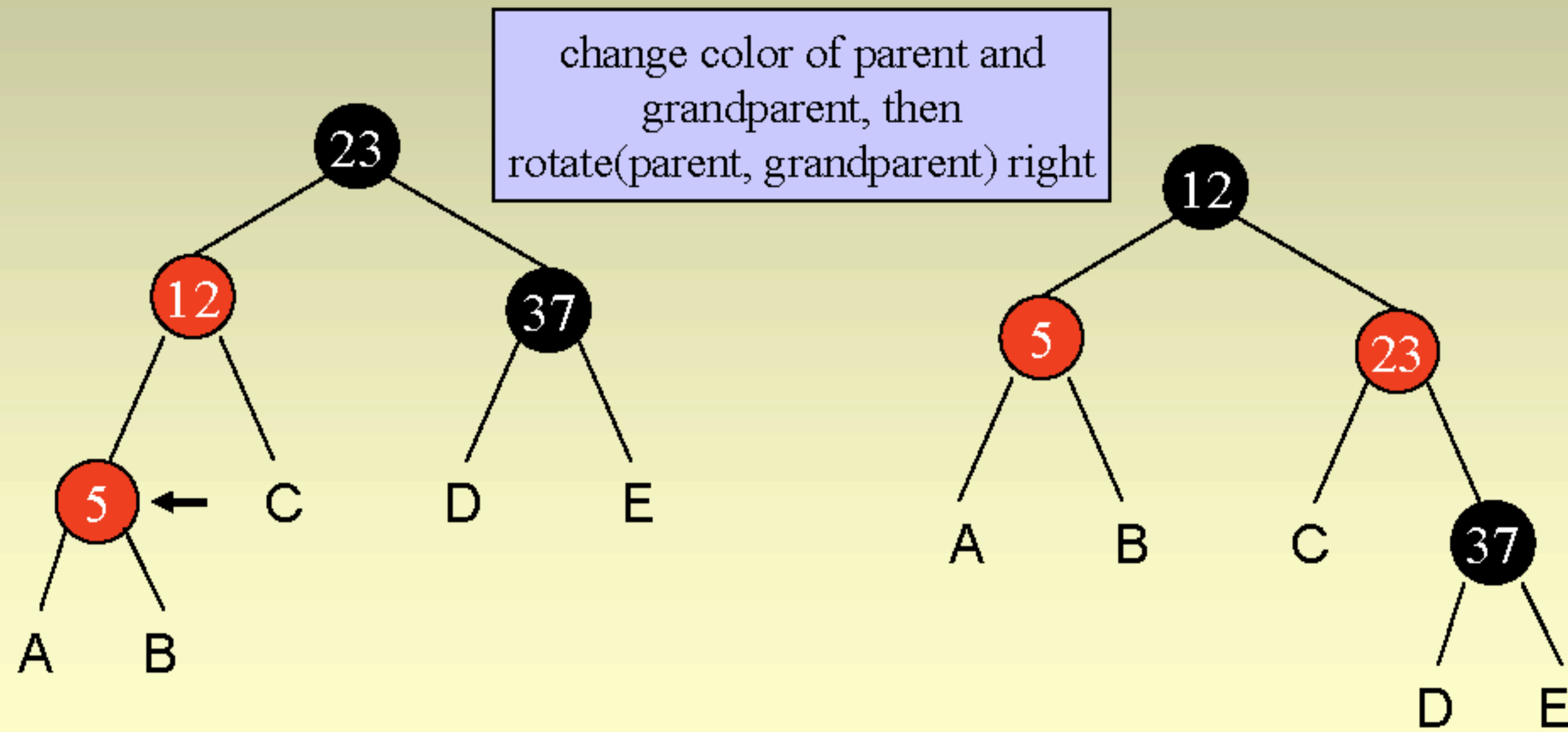
For each restoration on insert there are 6 cases, 3 of which are symmetric to the other three.

Case 2: the current node is a right child, whose parent is a red left child, and whose uncle is black.



For each restoration on insert there are 6 cases, 3 of which are symmetric to the other three.

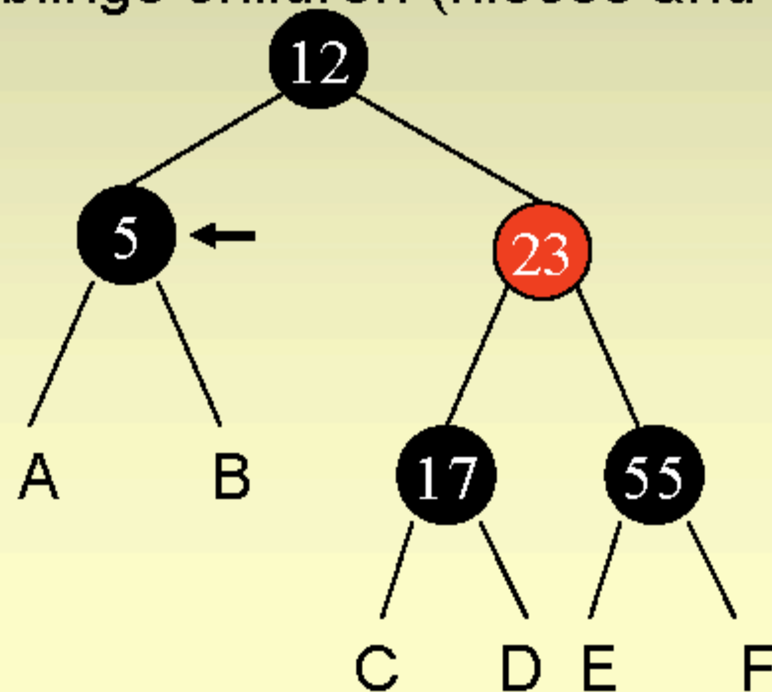
Case 3: the current node is a left child, whose parent is a red left child, and whose uncle is black.



So we see the insertion process is somewhat complex (not as complex as for AVL trees though)

Deletion of a node in a red-black tree is slightly more complex.

The local neighborhood includes the node, its parent, its sibling, and its siblings children (nieces and nephews).



Implementation of a Red-Black Tree

See `RBTree.h`

So what does the red-black tree provide ?

Does the added complexity on inserts and deletes of a red-black tree balance out the increase in average or worst-case search complexity?

The complexity of insertion and deletion in a red-black tree with n internal nodes is proportional to the depth of T.

$$\text{depth}(T) <= 2 \log_2(n+1)$$

Thus we say that the complexity is O(log n)

Next Actions and Reminders

Read CH pp. 603-614 on graphs

Program 5 is due 12/11.