Red-Black Tree Insertion

This lesson will cover the insertion operation in Red-Black trees, discussing all the four insertion cases.

We'll cover the following

Insertion in Red-Black Tree#

Here is a high-level description of the algorithm involved in inserting a value in a Red-Black Tree:

1. Insert the given node using the standard BST Insertion technique that we studied earlier and color it **Red**.
2. If the given node is the root, then change its color to **Black**.
3. If the given node is not the root, then we will have to perform some operations to make the tree follow the Red-Black property.

Rebalancing the Tree#

There are two ways to balance an unbalanced tree:

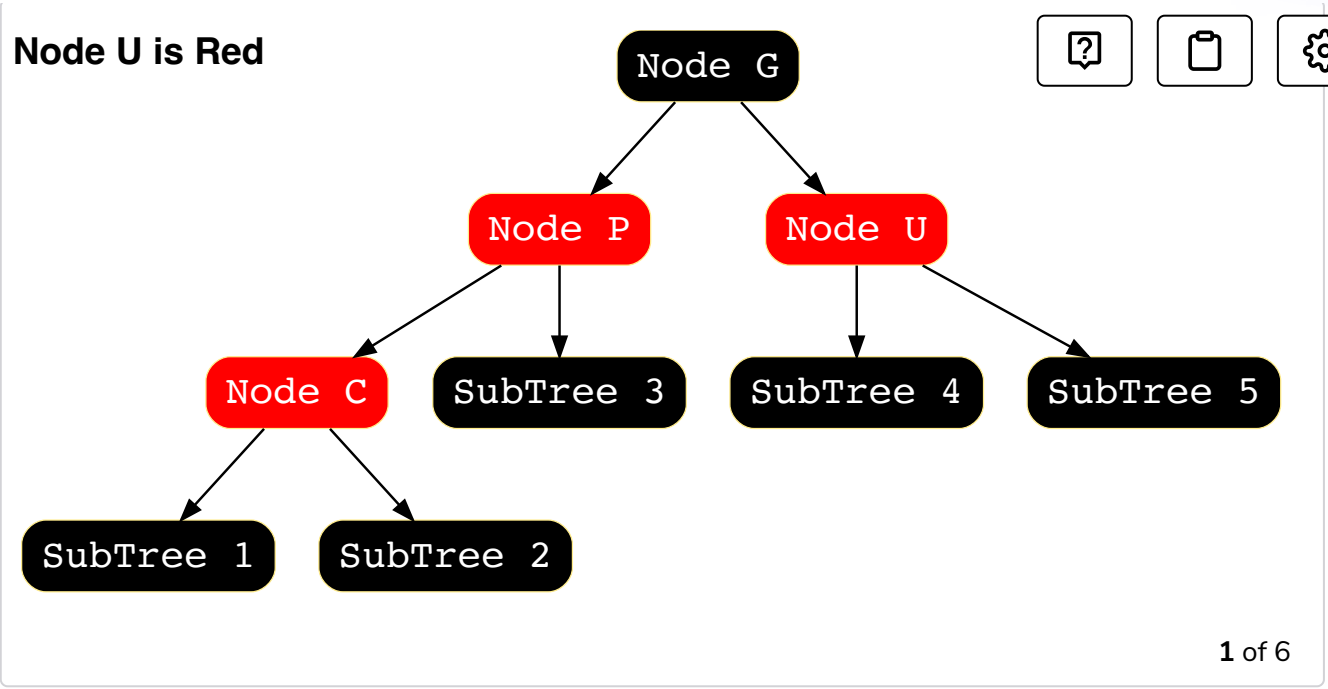
1. Recoloring Nodes
2. Rotating Nodes (left or right)

But before the details are explained, let's define the structure of the Red-Black Tree and some nodes relative to the given node, which is the node that we inserted in the Tree.

- Node C – the newly inserted node
- Node P – the parent of the newly inserted node
- Node G – the grandparent of the newly inserted node
- Node U – the sibling of the parent of the newly inserted node, i.e., the sibling of Node P / child of Node G

If the newly inserted node is not a root and the parent of the newly inserted node is not **Black**, first, we will check Node U and based on Node U's color, we balance the tree. If Node U is **Red**, do the following:

1. Change color of Node P and U to **Black**
2. Change color of Node G to **Red**
3. Make Node G the new *currentNode* and repeat the same process from step two



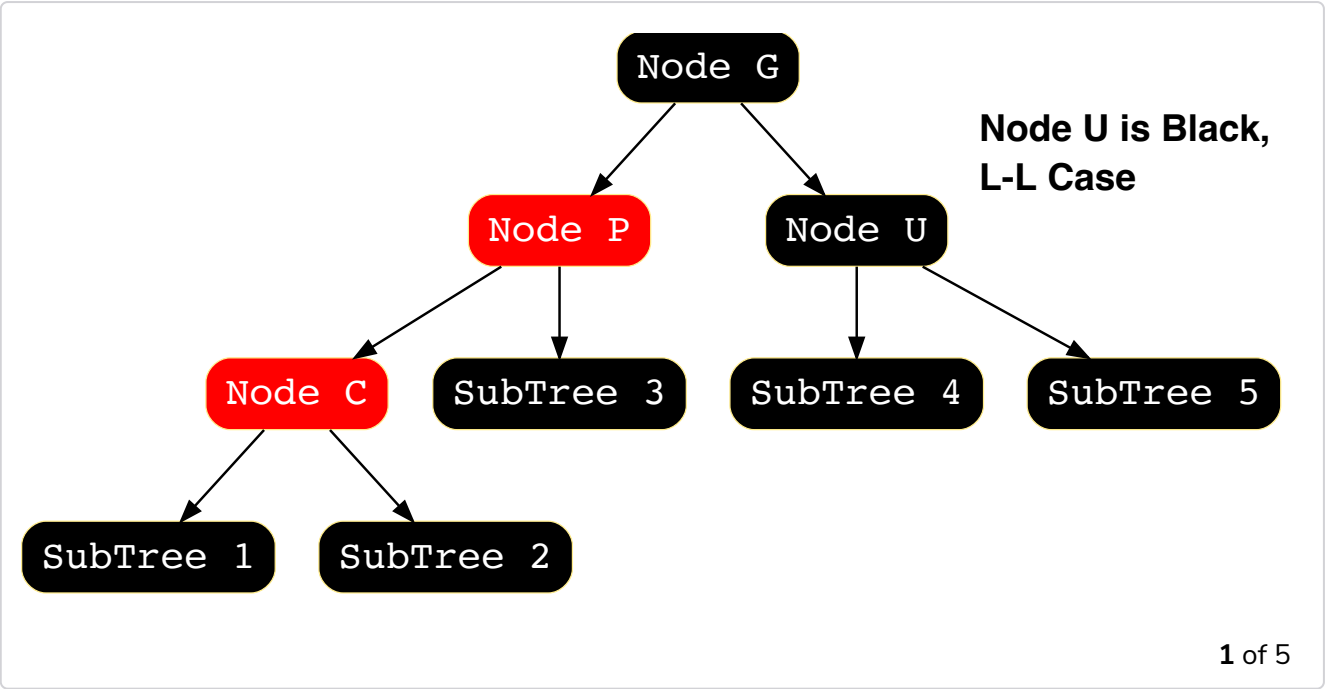
If Node U is **Black**, then we come across four different scenarios based on the arrangements of Node P and G, just like we did in AVL trees. We will cover each of these scenarios and try to help you understand through illustrations.

- Left-Left: Node P is the *leftChild* of Node G and *currentNode* is the *leftChild* of Node P
- Left-Right: Node P is the *leftChild* of Node G and *currentNode* is the *rightChild* of Node P
- Right-Right: Node P is the *rightChild* of Node G and *currentNode* is the *rightChild* of Node P
- Right-Left: Node P is the *rightChild* of Node G and *currentNode* is the *leftChild* of Node P

Case 1: Left-Left#

In case when the Node P is the *leftChild* of Node G and *current* [illegible] *leftChild* of Node P, we perform the following steps. Look at the illustration below for a better understanding.

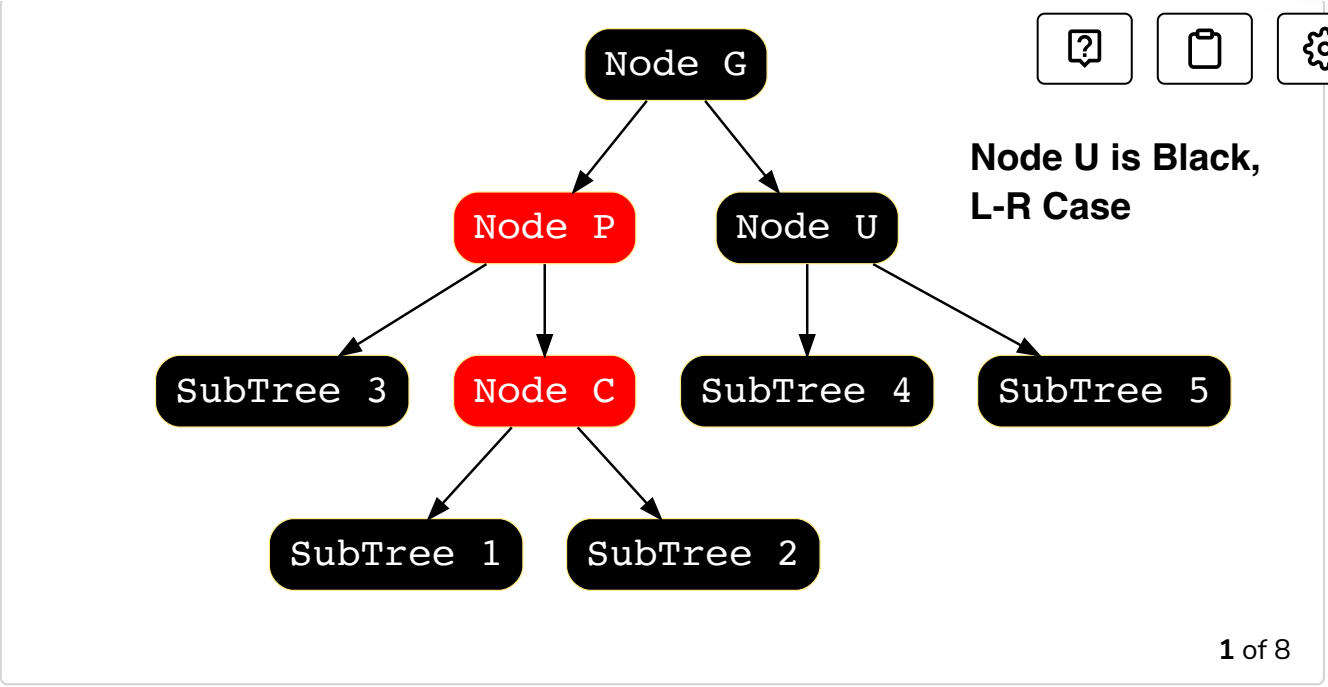
1. Right Rotate Node G
2. Swap the colors of Nodes G and P



Case 2: Left-Right#

In the case when the Node P is the *leftChild* of Node G and the currentNode is the *rightChild* of Node P, we perform the following steps. Look at the illustration below for better understanding:

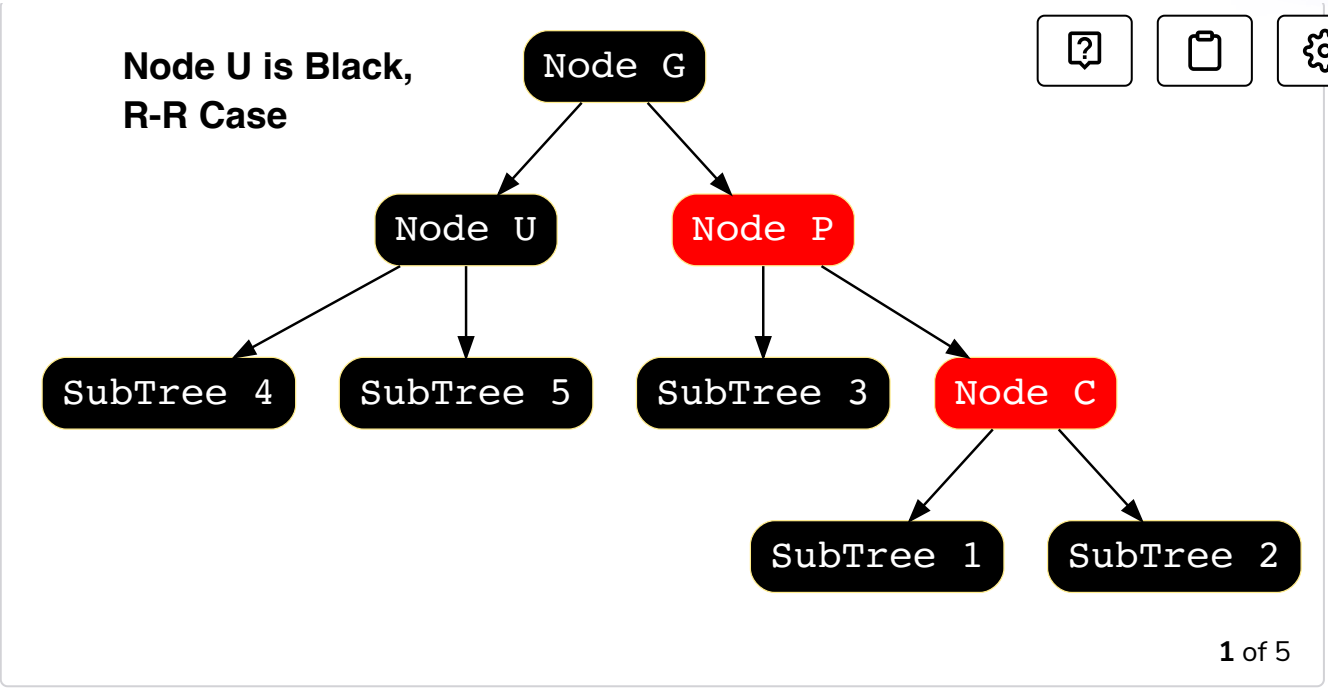
1. Left Rotate Node P
2. After that, repeat the steps that we covered in the Left-Left case



Case 3: Right-Right#

In the case when Node P is the *rightChild* of Node G and the *currentNode* is the *rightChild* of Node P, we perform the following steps. Look at the illustration below for a better understanding.

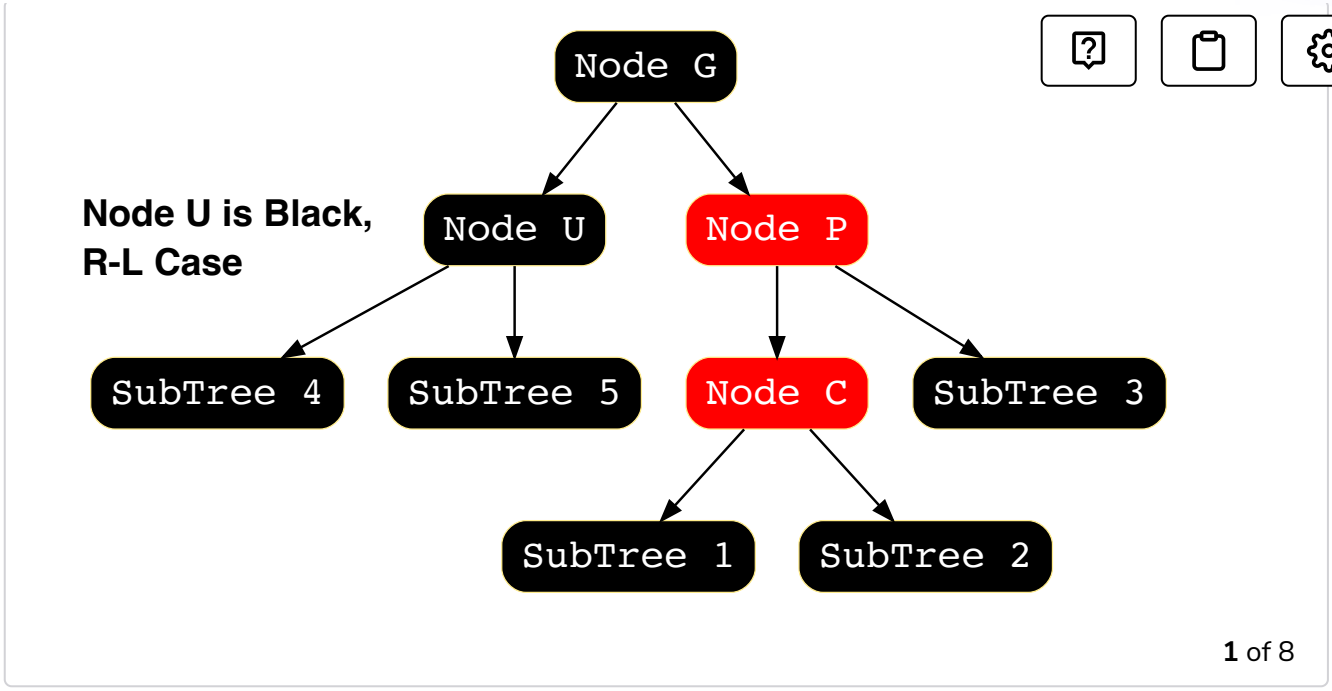
1. Left Rotate Node G
2. Swap colors of Node G and P



Case 4: Right-Left#

In the case when Node P is the *rightChild* of Node G and the *currentNode* is the *leftChild* of Node P, we perform the following steps. Look at the illustration below for a better understanding.

1. Right Rotate Node P
2. After that, repeat the steps that we covered in Right-Right case



In the next lesson, we'll study red-black tree deletion!

Back
What is a Red-Black Tree?

Next
Red-Black Tree Deletion

Mark as Completed

Report an Issue