Red-Black and 2-3-4 Trees

July 12, 2020

Description

Red-black trees are loosely self-balanced BSTs. They have a 1-to-1 correspondence to 2-3-4 trees, and are easier and more efficient to implement; however, the theory behind the rebalancing is clearer to illustrate with 2-3-4 trees.

Compared to AVL trees, lookup times in a red-black tree are generally slower because they are less strictly balanced. However, insertion and deletion are generally faster because they require fewer rotations to rebalance the tree. In general red-black trees are used for language libraries (e.g. C++'s set and map) and AVL trees are used in databases, where fast lookups are preferred.

Properties

2-3-4 trees have the following properties:

1. Each node has either 2, 3, or 4 children (and 1 fewer key).
2. The depth of every leaf node is the same.

Red-black trees have the following properties.

1. Nodes are colored red or black.
2. The root is always black.
3. No red node has a red child.
4. Every path from a given node to each of its descendant leaves has the same number of black nodes.

Note that by Property 4, every root-to-leaf path has the same number of black nodes. We refer to this value as the *black height* of the tree. The black height of a red-black tree is equal to the height of its corresponding 2-3-4 tree.

Insertion

To insert into a red-black tree, we start with a normal BST insert and color the inserted node N red. There are three cases for the position of N.

1. N is the root. In this case, we just color the insertednode the black and we have increase the black height of the tree by 1.

2. N has a black parent. In this case, the black height has not changed and no double red was created, so we are done.

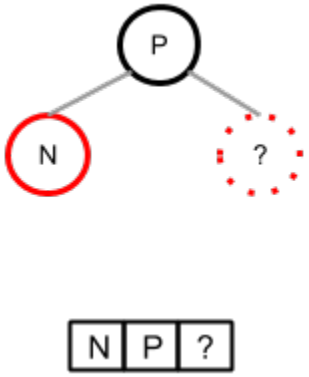

Figure 1: Insertion where N has a black parent

3. N has a red parent. This case is further divided into two cases:

   (a) N's uncle is null. Note that N's uncle cannot be black and S must be a leaf, otherwise black height will not be constant. To rebalance, we simply perform an AVL rotation and recolor.

   (b) N's uncle is red. To rebalance, we color the parent and uncle black, color the grandparent red (if it is not the root), and then recurse on the grandparent if we created a double red. Note how this corresponds to "pushing up" a node in the 2-3-4 tree.

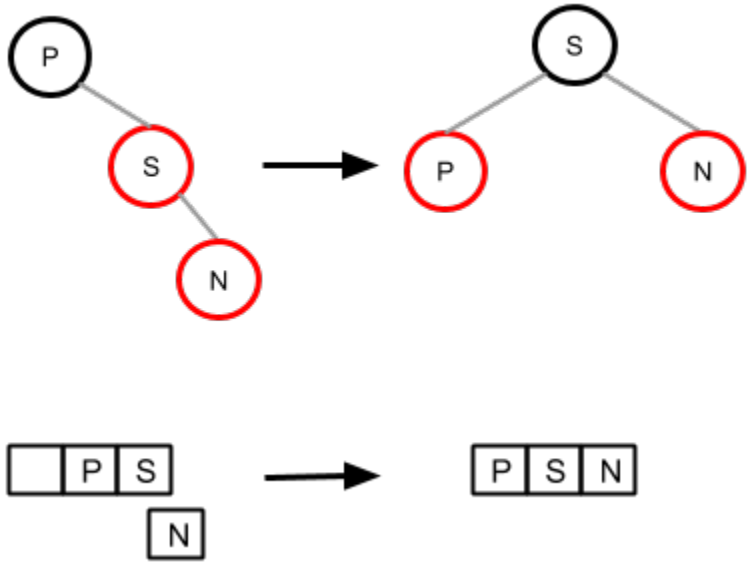

Figure 2: Insertion where N has a red parent and black uncle: RR rotation

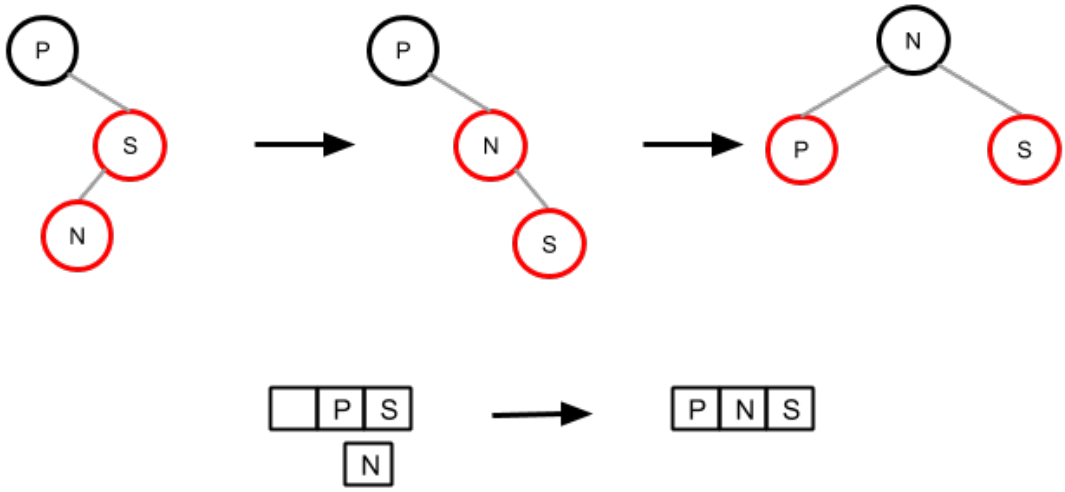

Figure 3: Insertion where N has a red parent and black uncle: RL rotation

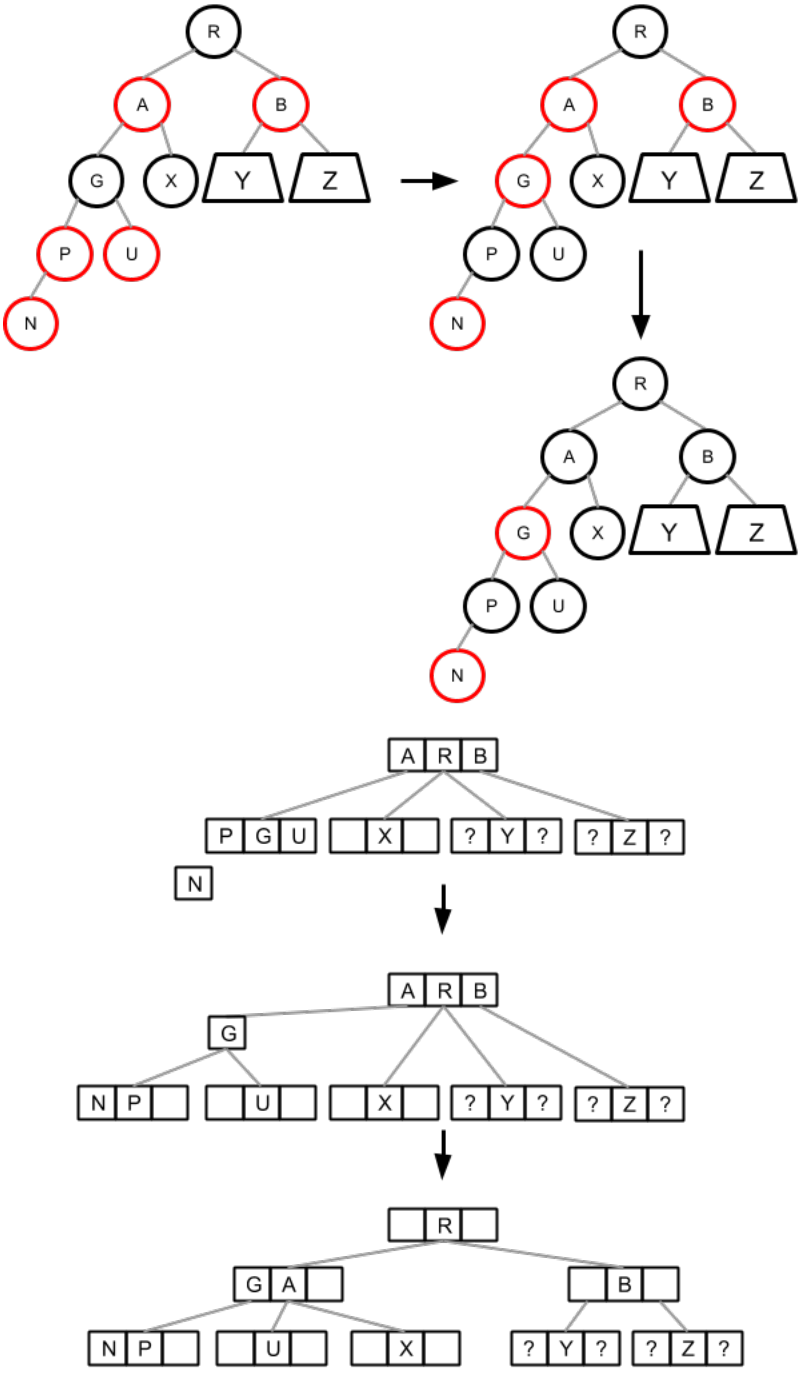

Figure 4: Insertion where N has a red parent and red uncle