Red-Black Tree | Set 1 (Introduction)
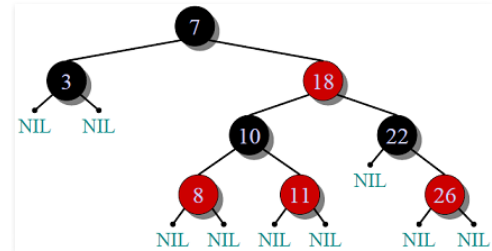
Red-Black Tree is a self-balancing Binary Search Tree (BST) where every node follows following rules.

1) Every node has a color either red or black.

2) Root of tree is always black.

3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4) Every path from root to a NULL node has same number of black nodes.

Why Red-Black Trees?

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that height of the tree remains O(Logn) after every insertion and deletion, then we can guarantee an upper bound of O(Logn) for all these operations. The height of a Red Black tree is always O(Logn) where n is the number of nodes in the tree.
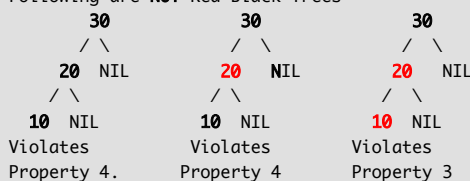
Comparison with AVL Tree

The AVL trees are more balanced compared to Red Black Trees, but they may cause more rotations during insertion and deletion. So if your application involves many frequent insertions and deletions, then Red Black trees should be preferred. And if the insertions and deletions are less frequent and search is more frequent operation, then AVL tree should be preferred over Red Black Tree.
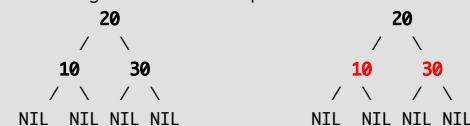
How does a Red-Black Tree ensure balance?

A simple example to understand balancing is, a chain of 3 nodes is not possible in red black tree. We can try any combination of colors and see all of them violate Red-Black tree property.

```
A chain of 3 nodes is nodes is not possible in Red-Black Trees.
Following are NOT Red-Black Trees
        30              30              30
       /  \            /  \            /  \
     20   NIL        20   NIL        20    NIL
    /  \            /  \            /  \
  10   NIL        10   NIL        10   NIL
Violates         Violates        Violates
Property 4.      Property 4      Property 3


Following are different possible Red-Black Trees with above 3 keys
        20                        20
       /   \                     /   \
     10    30                  10     30
    /  \   /  \               /  \    / \
  NIL  NIL NIL NIL          NIL  NIL NIL NIL
```

From the above examples, we get some idea how Red-Black trees ensure balance. Following is an important fact about balancing in Red-Black Trees.

Black Height of a Red-Black Tree :

*Black height is number of black nodes on a path from a node to a leaf. Leaf nodes are also counted black nodes. From above properties 3 and 4, we can derive, **a node of height h has black-height >= h/2**.*

Every Red Black Tree with n nodes has height <= $2Log_2(n+1)$

This can be proved using following facts:

1) For a general Binary Tree, let **k** be the minimum number of nodes on all root to NULL paths, then n >= 2^k – 1 (Ex. If k is 3, then n is atleast 7). This expression can also be written as k <= $2Log_2(n+1)$

2) From property 4 of Red-Black trees and above claim, we can say in a Red-Black Tree with n nodes, there is a root to leaf path with at-most $Log_2(n+1)$ black nodes.

3) From property 3 of Red-Black trees, we can claim that the number black nodes in a Red-Black tree is at least ⌊ n/2 ⌋ where n is total number of nodes.

From above 2 points, we can conclude the fact that Red Black Tree with **n** nodes has height <= $2Log_2(n+1)$

In this post, we introduced Red-Black trees and discussed how balance is ensured. The hard part is to maintain balance when keys are added and removed. We will soon be discussing insertion and deletion operations in coming posts on Red-Black tree.

Exercise:

1) Is it possible to have all black nodes in a Red-Black tree?

2) Draw a Red-Black Tree that is not an AVL tree structure wise?

Insertion and Deletion

Red Black Tree Insertion

Red-Black Tree Deletion

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

http://en.wikipedia.org/wiki/Red%E2%80%93black_tree

Video Lecture on Red-Black Tree by Tim Roughgarden

MIT Video Lecture on Red-Black Tree

MIT Lecture Notes on Red Black Tree

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner   Company Wise Coding Practice

Advanced Data Structure   Self-Balancing-BST

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Red-Black Tree | Set 2 (Insert)

AVL Tree | Set 1 (Insertion)

Red-Black Tree | Set 3 (Delete)

B-Tree | Set 1 (Introduction)

AVL Tree | Set 2 (Deletion)

Sparse Table

Minimum Word Break

Count greater nodes in AVL tree

Burrows – Wheeler Data Transform Algorithm

Segment tree | Efficient implementation