Red Black Trees

R. K. Ghosh

IIT Kanpur

Red Black Trees

Red Black Tree

- Preserves ordering property of a BST - **Ordering Invariant**.
- Nodes are colored either as red or as black.
- No two consecutive nodes can be colored red - **Color Invariant**
- All leaf nodes are colored black.
- Root is colored black.
- All the three coloring properties are preserved.
- Number of black nodes on a path from root to a leaf node is the same - **Height Invariant**.

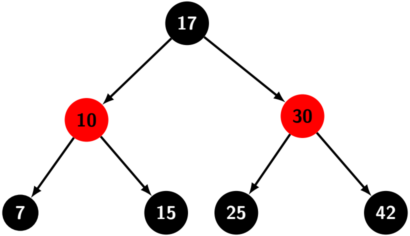
Operations in a Red Black Tree

- Searching is easy and performed in the same way as in a BST.
- Insertion requires rebalancing, where balance is restored by ensuring black height is same - **Height Invariant**.
- Newly inserted node is colored red which may violate color invariant - Two consecutive node color being red.
- Rotations are performed to restore color invariant.

Black Height & Height

- Before dealing insertion operation. Let us find about relationship between **tree height** and **black height**.
- We refer to the number of nodes in path from a node to farthest leaf as its black height.
- Let us consider some examples of Red-Black trees.

Color invariance holds



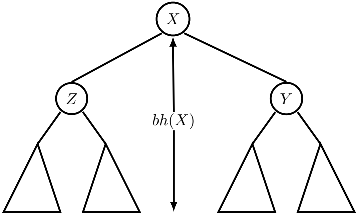
- Every path from root to a leaf has same number of black nodes.
- No two consecutive nodes are colored red.

Internal Nodes and Black Height

Black Height

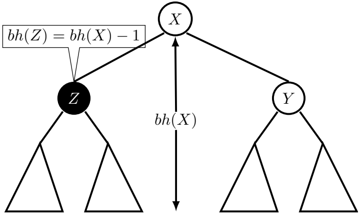
Let $bh(x)$ (black height) of a node x be the number of black nodes on the path to x from a farthest leaf excluding x itself. Then prove that a subtree of red black tree rooted a node x has at least $2^{bh(x)} - 1$ internal nodes,

- If x is a leaf node of an extended RB tree, it is treated as a dummy black node.
- So, black height of a leaf node is 0 which is true as $2^0 - 1 = 0$.
- Now apply induction to prove it.



Internal Nodes and Black Height

- Let Z be black, then $bh(X) = bh(Z)$
- If Z is red, then $bh(X) = bh(X)$.
- This implies, $bh(Z) \geq bh(X) - 1$.



Therefore, the number nodes in X's tree should be at least:

$$\begin{aligned} 2 \times (2^{bh(X)-1} - 1) + 1 &= 2^{bh(X)-1} - 2 + 1 \\ &= 2^{bh(X)-1} - 1 \end{aligned}$$

Height of a Red Black Tree

Height is O($\log n$)

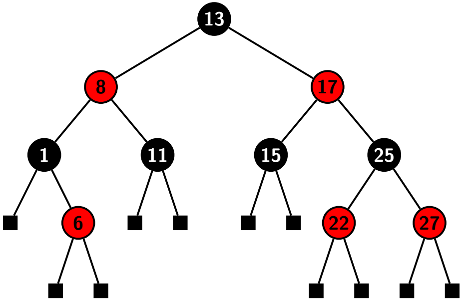
Height of a red black tree with n nodes is $O(\log n)$.

- If a red black tree has n nodes then we have $n \geq 2^{bh(root)} - 1$.
- At least half the nodes on a path from the root to a leaf node are black.
- So, $bh(root) \geq h/2$, implying that $n \geq 2^{h/2} - 1$, where h is the height of the tree.
- Therefore,

$$h/2 \leq \log(n+1), \text{ or } h \leq 2\log(n+1).$$

- In other words, $h = O(\log n)$.

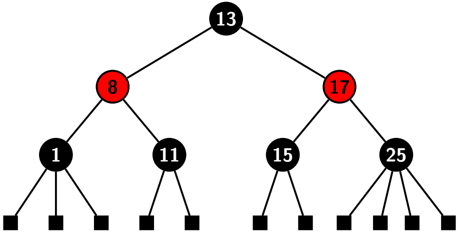
Collapsing the Red Nodes



Initial Red Black Tree

- Collapsing all red nodes into their respect black parents.

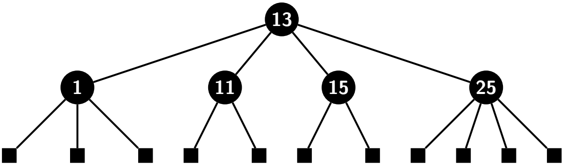
Collapsing the Red Nodes



After collapsing 6, 22, 27

- Each node in such a compact black tree can have 2, 3 or 4 children.

Collapsing the Red Nodes



After collapsing 8 and 17

- The height of collapsed tree is $h' \geq h/2$ and all external node are at same level.
- The number of internal nodes in the tree is

$$n \geq 2^{h'} - 1 \geq 2^{h/2} - 1$$

- Therefore, $h \leq 2\log(n+1)$

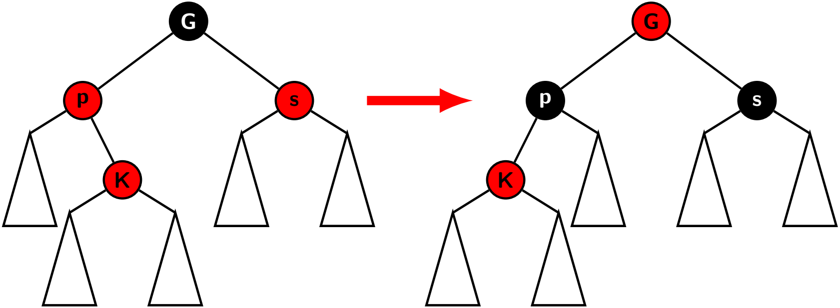
Summary of Properties

1. Every node is colored either red or black.
2. Root is always colored black.
3. Every leaf is colored black.
4. If a node is red, then both its children are black.
5. All paths from a node to descendant leaves have same number of black nodes.

Insertion May Violate Color Invariant

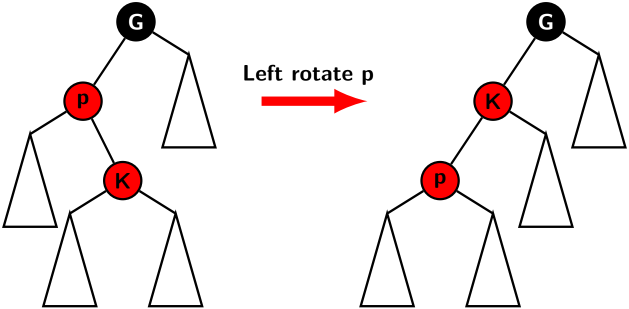
- A node is always inserted as an interior node in the place of a black leaf.
- The inserted node is colored red, and the two children of inserted nodes are leaves colored black.
- Insertion does not disturb black depth of any node. So, third property of **color invariant** is preserved.
- Second property of **color invariant** is also preserved as all leaves are colored black.
- Property 5 is satisfied as well.
- So violation of color invariant may be due to properties 2 and 4 not being preserved.

- New node K's parent $p(K) = X$, X is left child of $p(p(K)) = G$, sibling of X is red. K may be left or right child of $P(K)$.
- Transfer color of G to p and s and recolor G red.
- Pushes problem to G and $p(G)$.



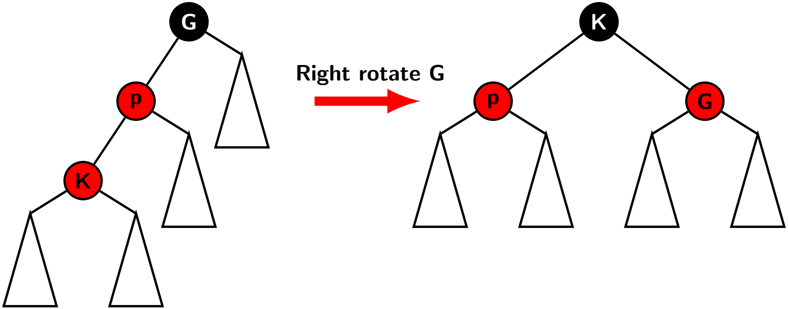
Case 2

- K is right child of $p(K)$ and $p(K)$ is red.
- Rotate p left, transforming it case 3.

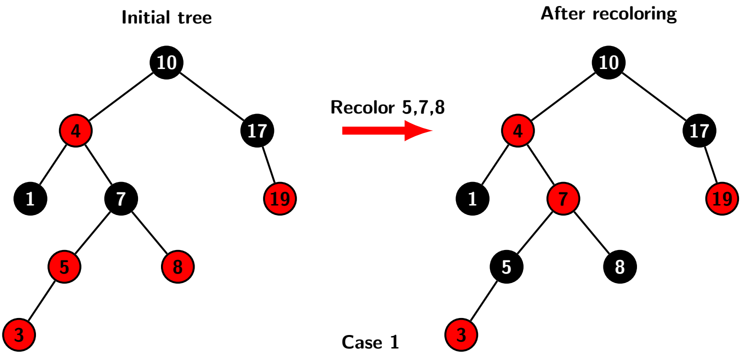


Case 3

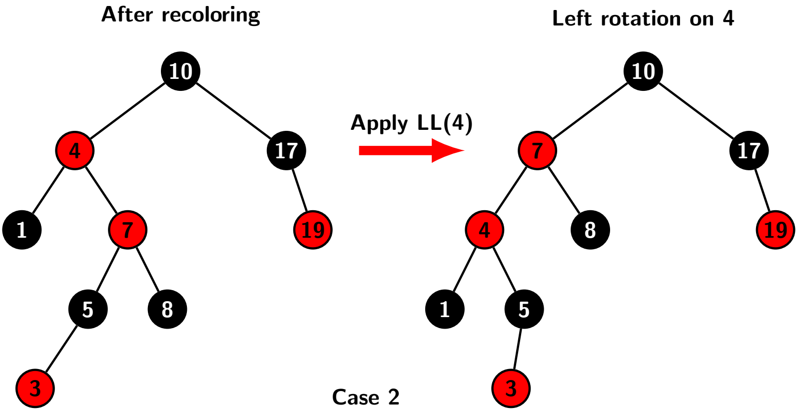
- K is the left child of $p(K)$ and $p(K)$ is red.
- Rotate G right, recolor K black and G red.



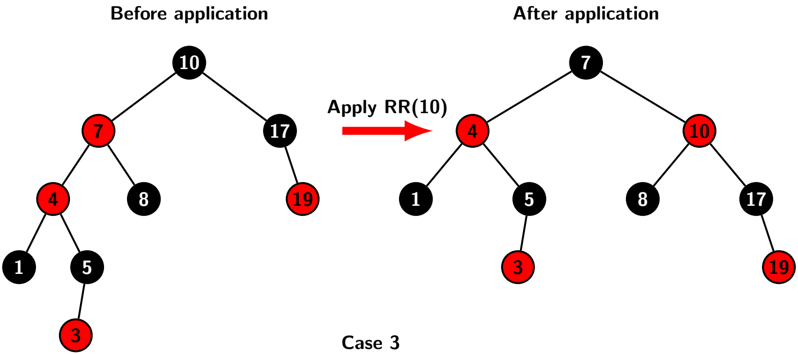
Case Illustrations 1



Case Illustrations 2

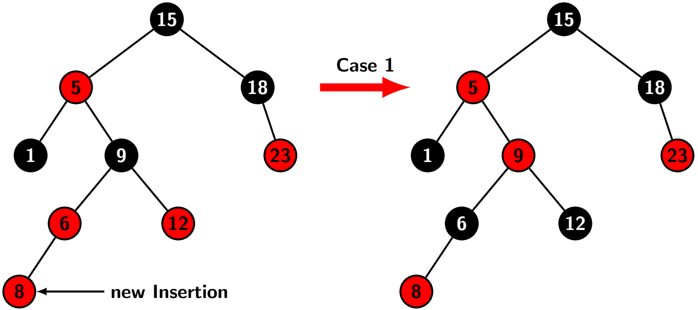


Case Illustrations 3



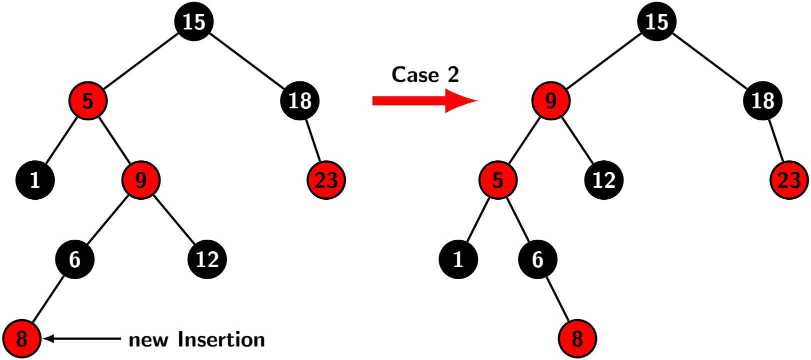
Case 3

Inserting 8 into the Tree



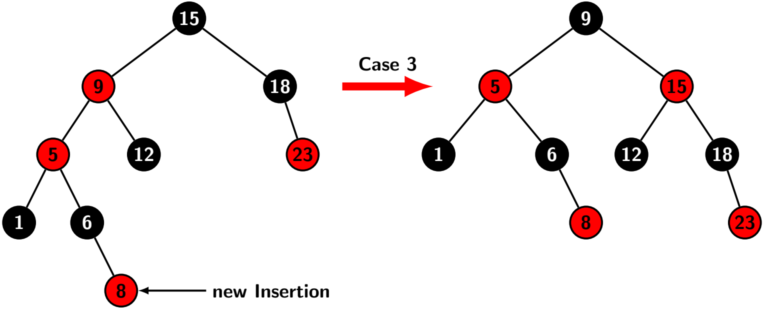
- Insertion violate the color invariants.
- Since uncle of 8 is red, recolor parent and sibling.

Now Case 2 Appears



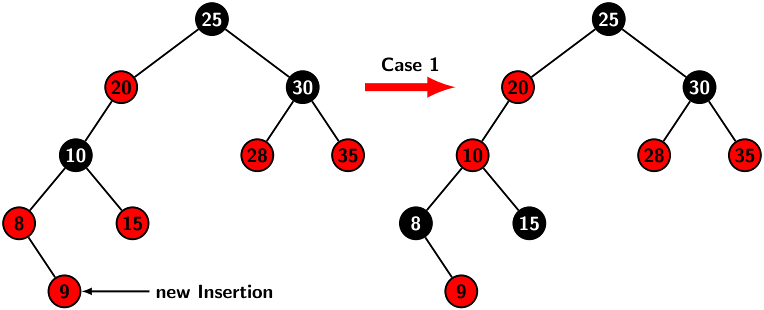
- Rotate left and transform to case 3.

Perform Case 3 Rotation About 9



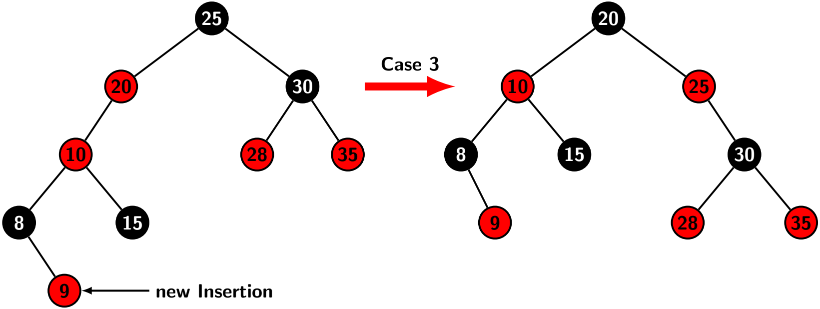
- Color invariant is restored, height invariant is also restored and root is black.

Perform Recoloring as in Case 1

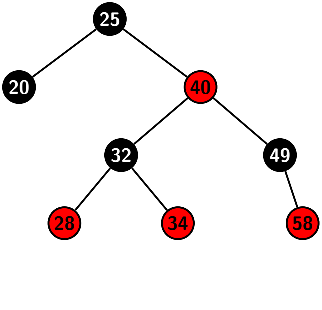


- Now case 3 situation occurs.

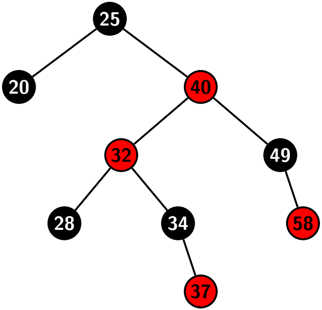
Right Rotation About 20

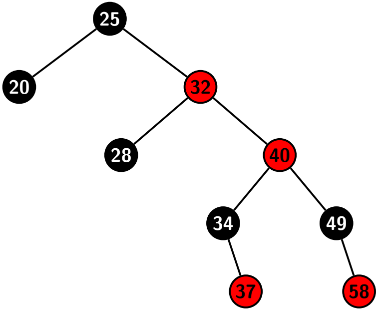


Insert 37

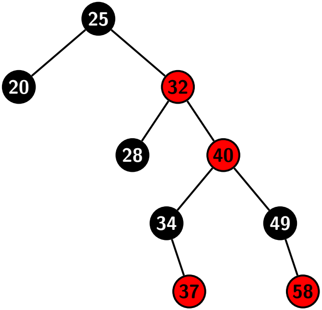


Use Case 1 Followed by Case 2





Now Use Case 3



Pseudo Code

```
insertRBtree(T, x) {
    color[x] = red;
    while (x ≠ root[T] && color[p[x]] == red) {
        if (p[x] == left[G[x]]) {
            y = right[G[x]];
            if (color[y] == red) < Case 1 >
            else if (x == right[p[x]])
                    < Case 2 >
            < Case 3 > // Case 2 ==> Case 3
        }
        else
            < if clause with left and right
               interchanged>;
    }
    color[root[T]] = black;
}
```

Time Complexity

- Recoloring takes on O(1) time.
- Restructuring or a rotation involves three nodes. Hence a single rotation also takes O(1) time.
- Fixing the the color invariant (using rotations) may push the violation of property 4 one level at a time.
- In the worst case fixing operation may have to be executed O(h) time.
- Since $h = O(\log n)$, the time for fixing a violation of color invariance may take up to O($\log n$) time.

Summary

- Red black tree is another interesting way of keeping a BST balanced.
- It uses rotations like AVL tree, but much more sparingly.
- It requires an additional information field for keeping color information. However, the information is just 1 bit.
- It does not require height recomputation as it was required in AVL tree each time an insertion or deletion happen.
- The asymptotic time complexity remains O(h) where h is the height of the tree.