Red-Black Tree

Shiv Shankar Dayal

Red-Black Trees

Red-Black tree(RBT) is of one of the most important data structures. It is a self-balancing binary tree. The most attractive part of RBT is its performance. Because it is self-balanced the tree does not skew in case of poor order of elements during insertion. An example RBT is given below:

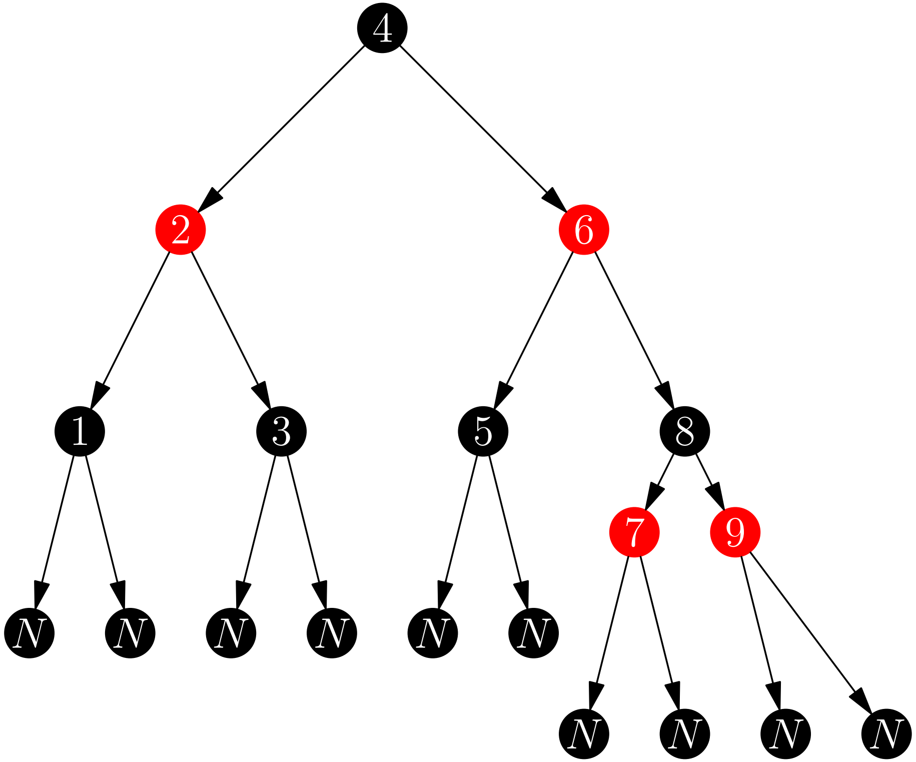

Fig. 1. Red-Black Tree

Properties of Red-Black Trees

Red-Black Trees have following properties

1. Every node is either red or black.
2. The root node is always black.
3. The red nodes cannot have red children.
4. All leaves are black nodes.
5. Every path from a node to its descendant null nodes have the equal no. of black nodes.
6. Insert, search and delete all have time complexity of $O(\log n)$ for RBT.
7. The black height of an RBT is the number of black nodes on a path from the root to a leaf. Leaves are counted as black. So, an RBT of height h has black height $>= h/2$.
8. Height of an RBT with n nodes is $h \leq 2\log_2(n+1)$.
9. The black depth of a node is defined as the number of black nodes from the root to that node

Following these rules we can say that a chain of three nodes is not possible as that will violate one of the properties mentioned above.

Insertions in Red-Black Trees

Insertions might cause unbalance/inconsistency in an RBT. To insert a new element we insert as if it is a BST. After the insertion we check for violations of rules of RBT, and correct the errors if any.

There are two possible cases:

1. Uncle is red. Recolor the parent and uncle to black, and the grandparent to red. Then move up the tree to correct violations.
2. Uncle is black. Two cases arise:

   a. Node is a right child. Perform a left rotation on the parent.
   b. Node is a left child. Perform a right rotaion on the grandparent and recolor.

Deletions in Red-Black Trees

Like insertions, deletions can also cause inconsistencies in an RBT. To delete a node, we perform a deletion as if it is a BST, and then fix the inconsistencies. If a black node is deleted, a "double black" condition might arise, which requires fix. To fix this we need to check color of sibling node.

1. Sibling is red. Rotate the parent and recolor the sibling and parent.
2. Sibling is black. Two cases arise:
   a. Sibling's children are black. Recolor the sibling and propagate the double black upwards.
   b. At least one of shibling's children is black.
      i. If the sibling's far child is red, then we perform a rotation on parent and sibling. Also recolor them properly.
      ii. If sibling's near child is black, then we rotate the sibling and its child and proceed as above.