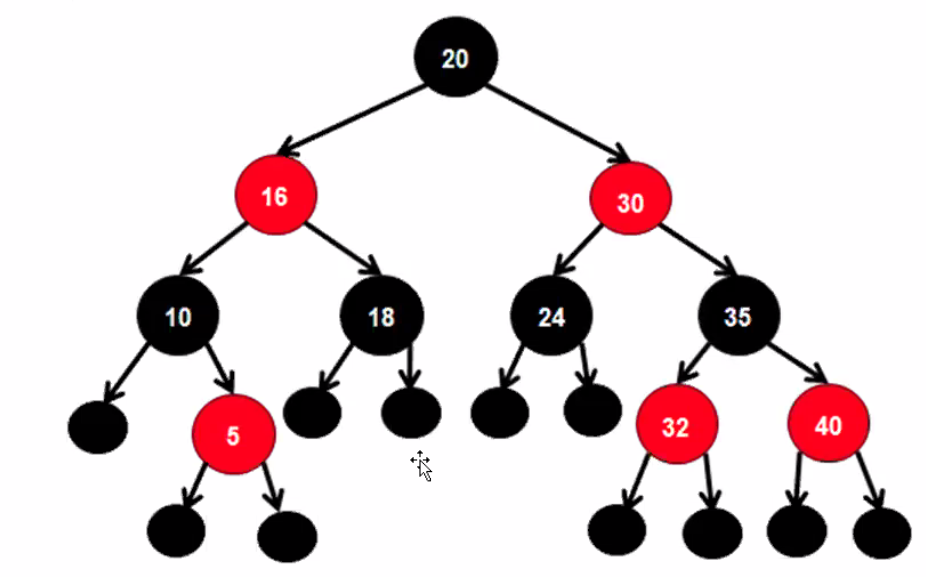
RBTs

Important properties of RBT

* RBTs and BSTs and search with self-balancing insert and deletion operations of time O(log n) and space O(log n)
* RBTs are balanced binary trees by means of longest/shortest <=2. The height of RBTs is O(log n)

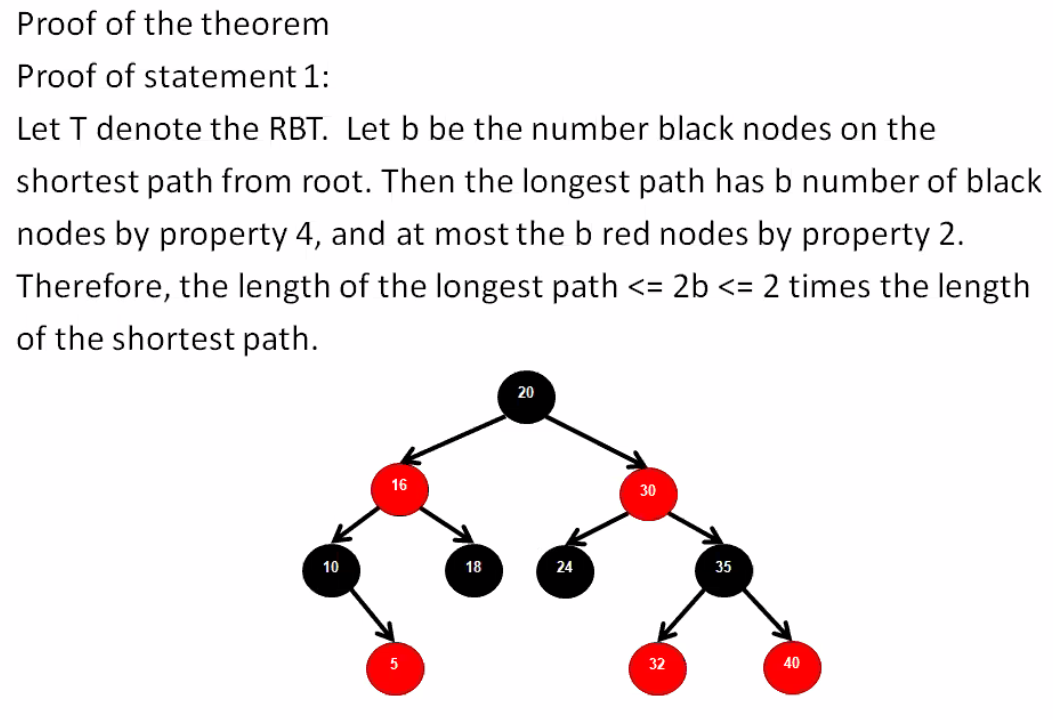
**Concept of Red Black Trees**

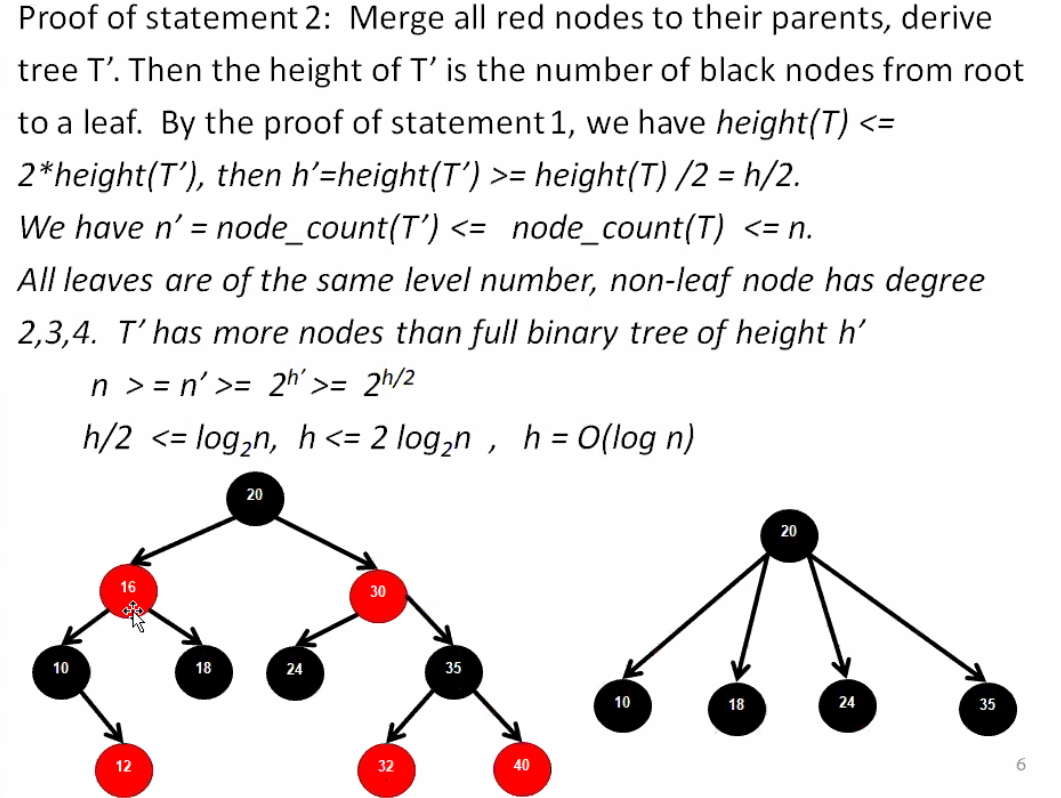
* A red black tree is a bast with self-balancing insert and delete operations satisfying the following conditions:
  + Every node is labeled a color of either red or black
  + The color of the root node is black, and all NULL children are viewed as black nodes
  + The child nodes of a red node are black
  + For any node, every path from the node to any of its NULL node has the same number of black nodes



Theorem

1. In an RBT, for any node, the length of the longest path from the node to a leaf is less or equal to two times the length of the shortest path from the root to a leaf
2. The height of an RBT if n nodes is O(log n)





**RBT insertion**

Algorithm:

1. Insert a node into RBT as BST and set the new node red
2. If the resulted colored BST is not RBT, rebalance to restore it to RBT depending on the following five cases

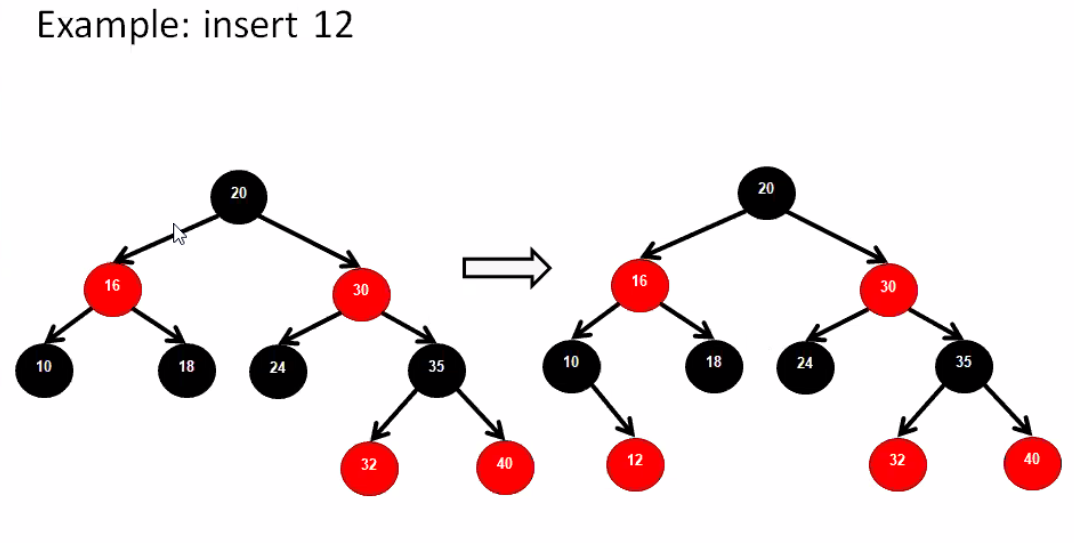
Case 1: the new node N is added as the root of the tree

Action: Repaint N black



Case 2: the new node’s parent P is black

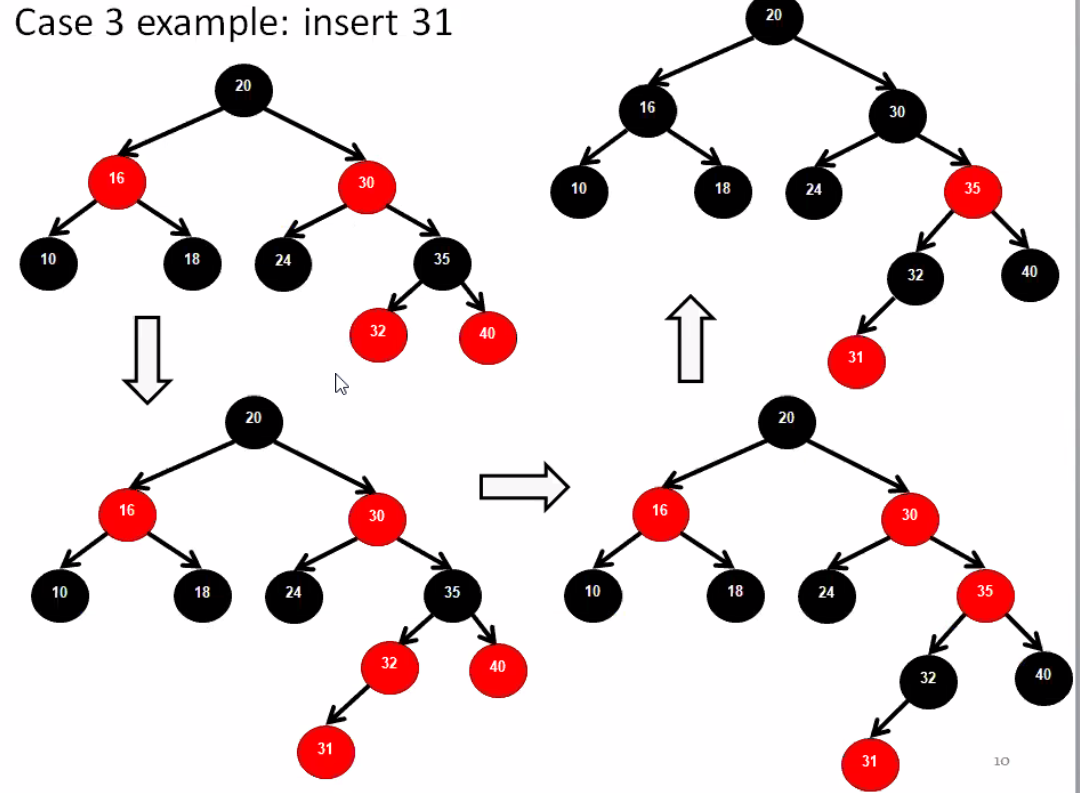
Action: no action is needed as the derived tree is an RBT



Case 3: If both the parent P and the uncle U are red

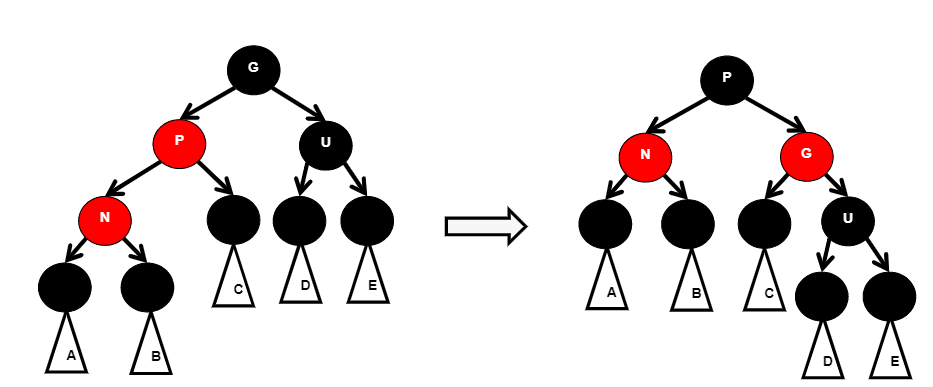
Action: repaint both nodes P and U black, and grandparent G red. If the parent of G is red, go back to case 1 taking G as new node

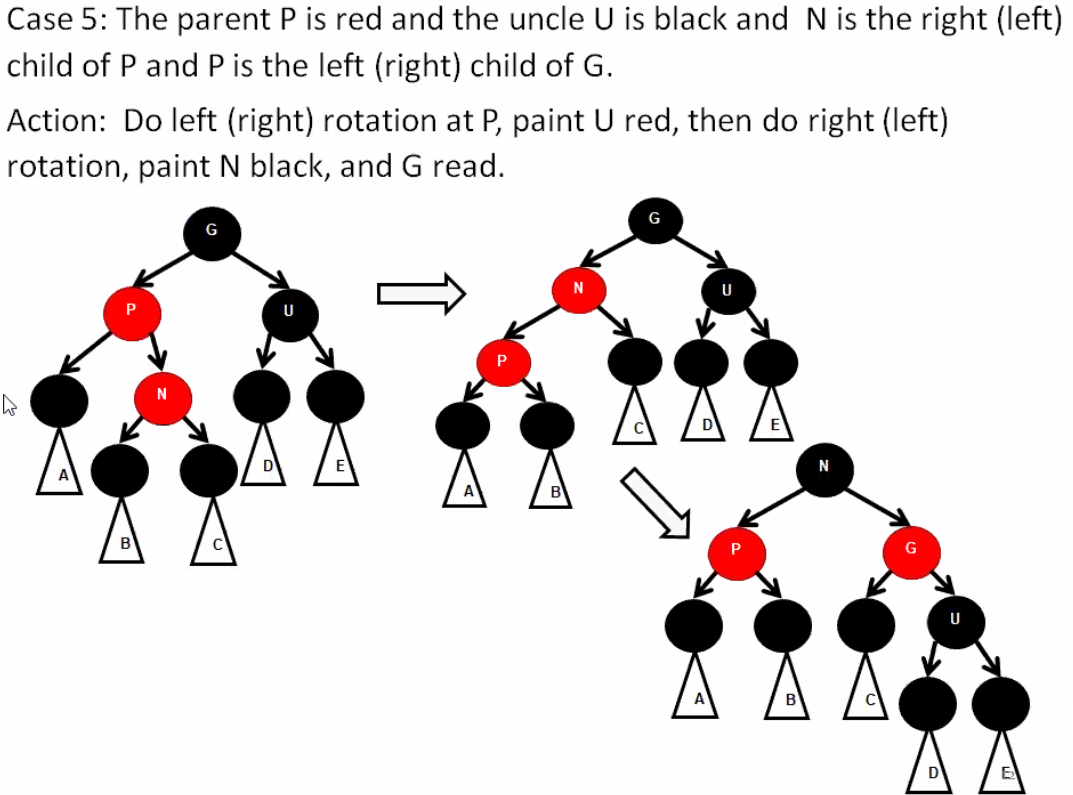




Case 4: the parent P is red and the uncle U is NULL or black and the new node N is the left (right) child of P, and P is the left (right) child of its parent G.

Action: Do right (left) rotation at G, repaint P black and G red





**RBT deletion**

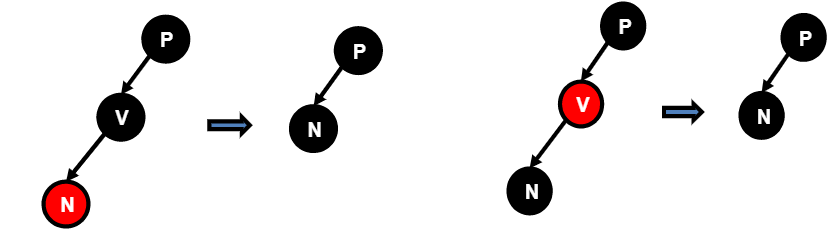
Algorithm:

1. Delete node as BST deletion
2. Rebalance to restore RBT depending on the following cases:

The BST deletion ends at deleting a node which is either a lead node (including the smallest of right subtree) or non-leaf node which has only one Null child. Let V de the deleted node, and N is its child. N is NULL when V is a lead.

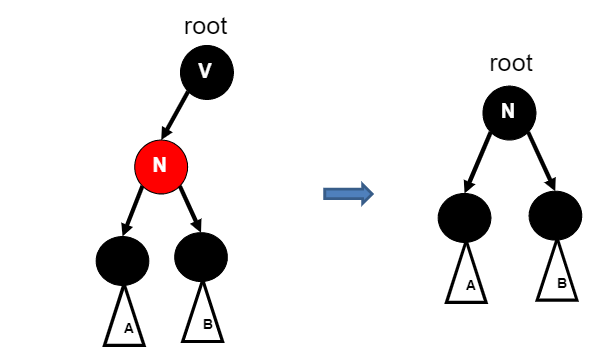
Case 1: At least of one of V and N is red

Action: Paint N black



Case 2: N is root after deletion

Action: Paint N black



Case 3: N is not root. N has parent P

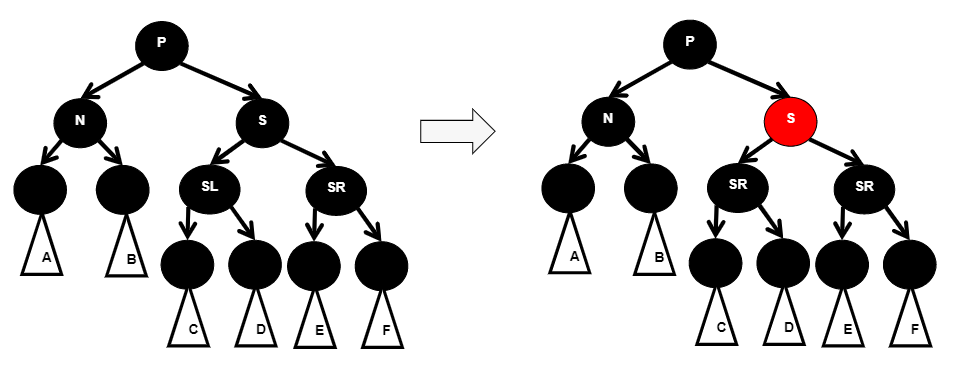
Case 3.1: N has sibling S is red

Action: Paint P red and S black, then rotate left is S is the right child of P; rotate right if S is the right child of P. Set P be the balance node. Go to case 3.3



Case 3.2: P, S, and S’s children are black

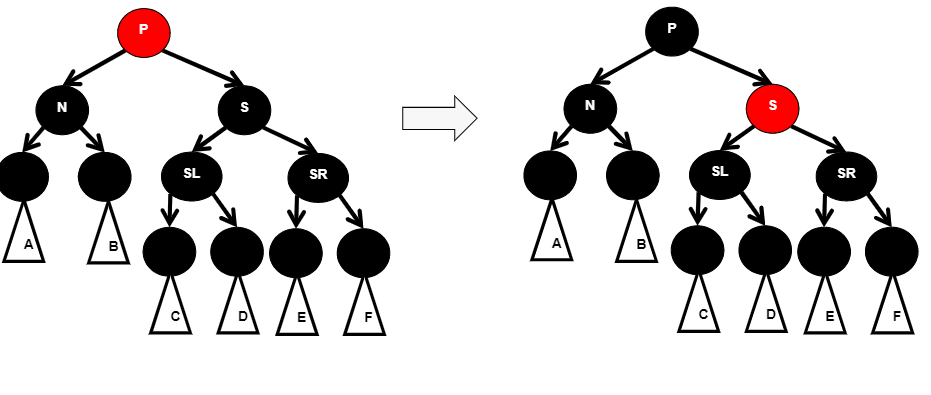
Action: Repaint S with red, rebalancing procedure on P, starting at case 2



Case 3.3: P is red and S is black

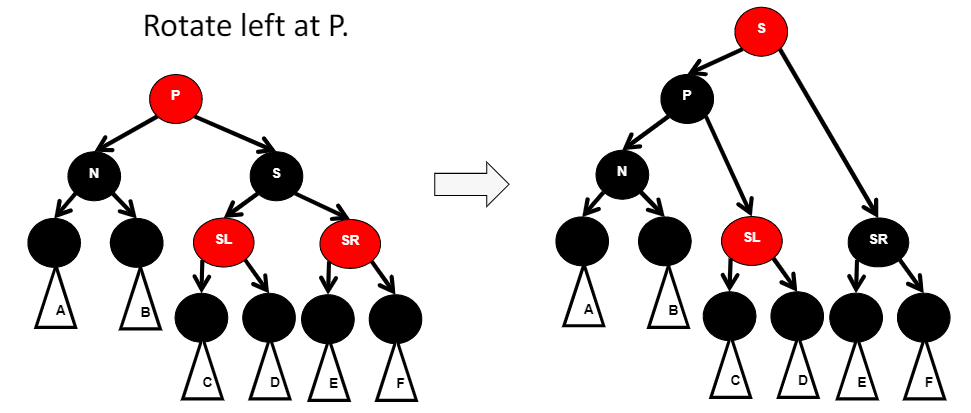
Case 3.3.1: Both children of S are black

Action: Paint P black and S red



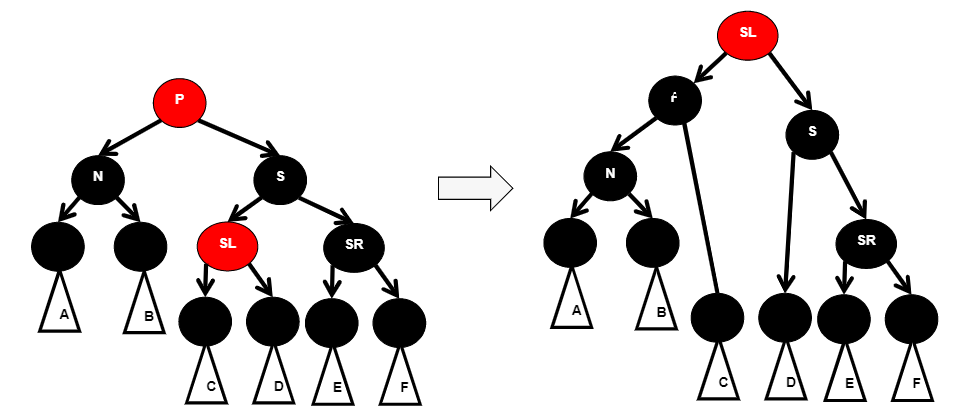
Case 3.3.2: Both children of S are red

Action: Paint P black and S red, and right child of S back. Rotate left at P



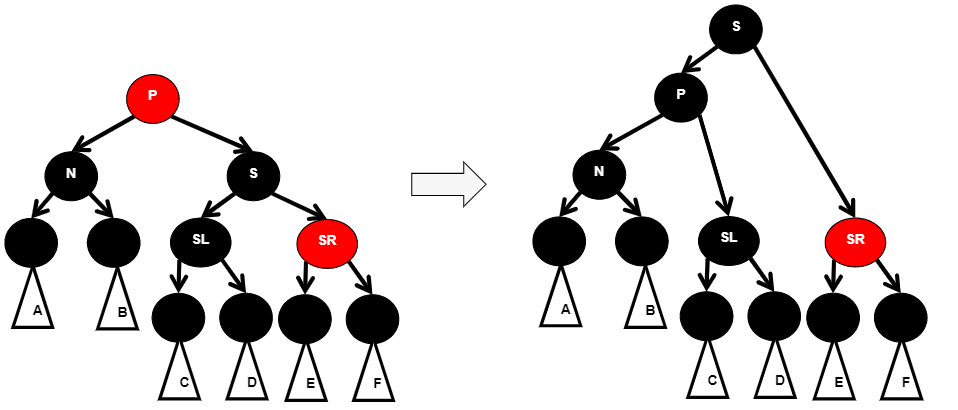
Case 3.3.3 left child of S is red and right child is black

Action: Paint left child of S black S red. Rotate right at S, then rotate left at P



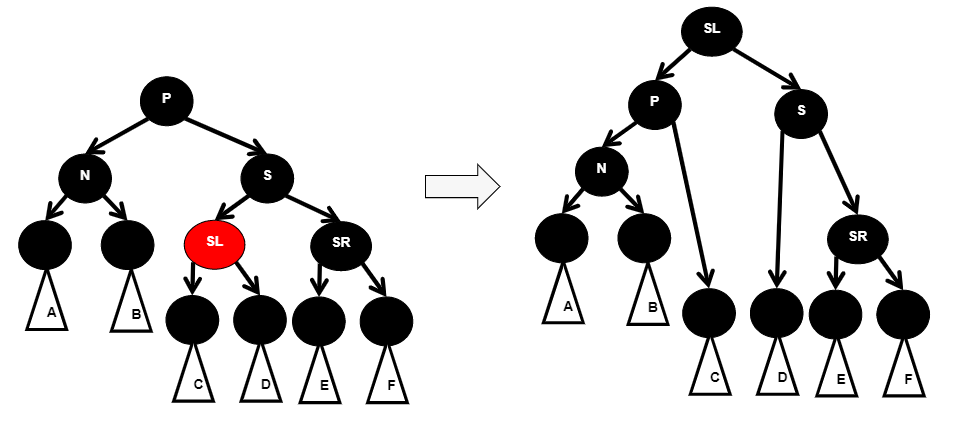
Case 3.3.4 Left child of S is black and right child is red

Action: Paint left child of S black, S red. Rotate left at P



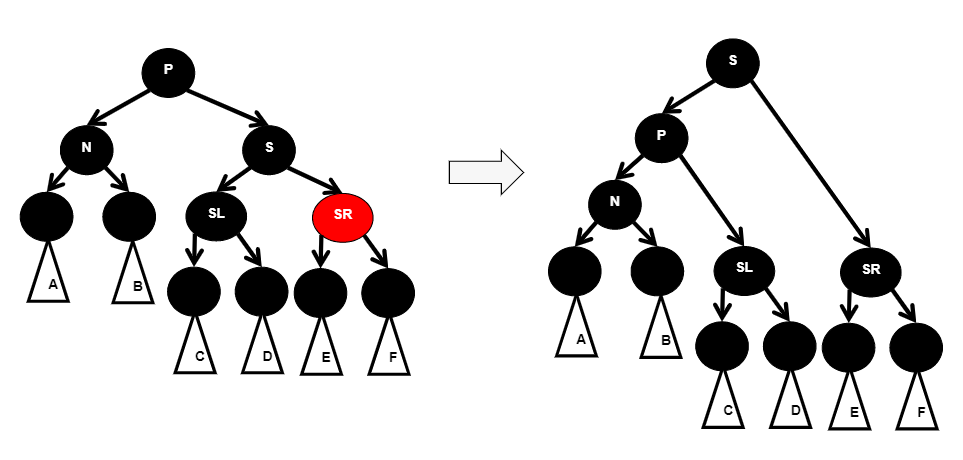
Case 3.4 If N is the left (right) child of P and S is black, S is left (right) child is red, S’s right (left) child is black.

Action: Paint S’s left (right) child black, and rotate right (left) as S, then rotate left (right) at P.



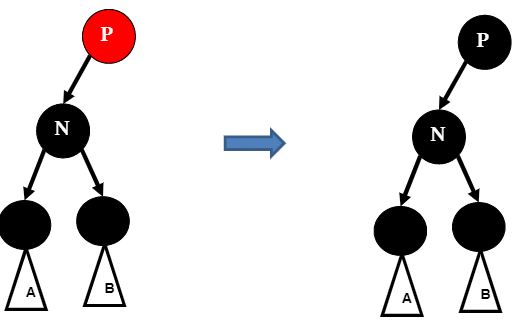
Case 3.5 S is black, S’s right (left) child is red, and N is the left (right) child of P.

Action: Paint the right (left) child of S black. Rotate left (right) at P.



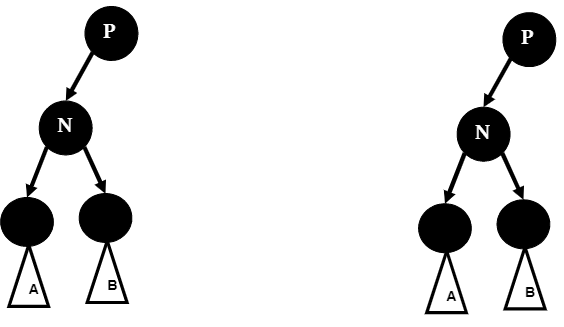
Case 3.6 N has no sibling, and P is red

Action: Paint P black



Case 3.7 N has no sibling, and P is black

Action: set P as new N, start from case 2.



**RBT summary**

1. A BST is balanced by means of all paths from a node to lead nodes have the same number of black nodes. As a result, the length of a longest path is no more than two times of the length of a shortest path from a node to leaves. All paths from a node to leaf nodes have the same number of black nodes.
2. The height of a RBT in n nodes is O(log n)
   1. Insert and delete operations can be done in time O(log n), space O(log n)
   2. If a parent pointer is used in node structure, the insert and delete operations can be done in time O(log n), space O(1).
3. Implementations
   1. Need additional variable color in node structure
   2. Need bottom up rebalance
   3. Many cases to handle, not as simple as AVL tree
   4. Not as efficient as AVL tree in space usage



