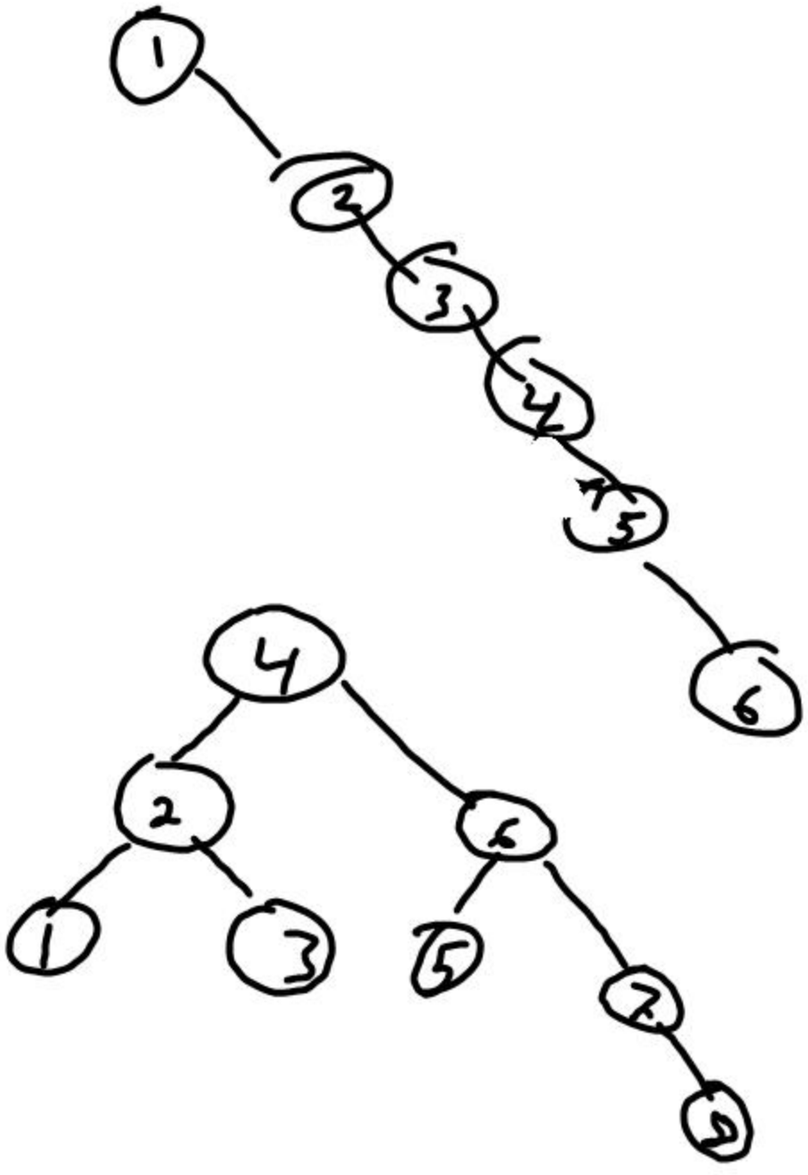
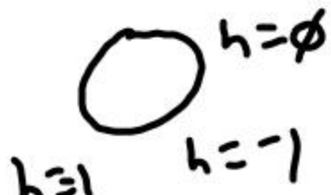
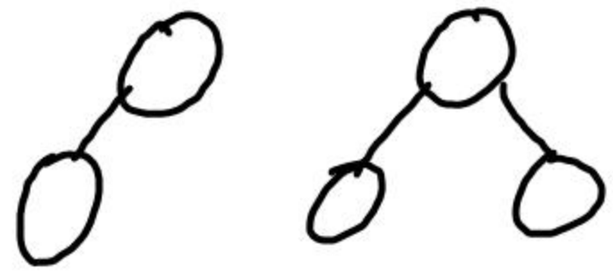
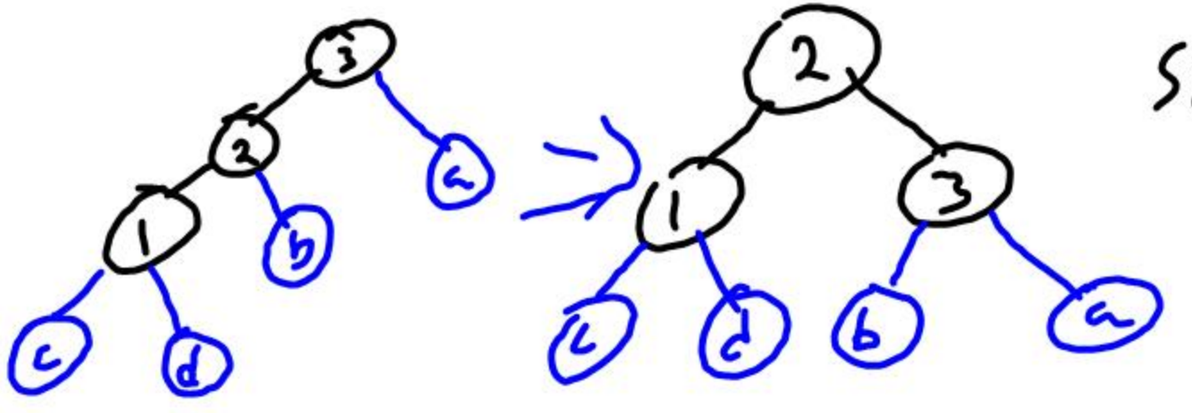


Balanced Binary search tree.

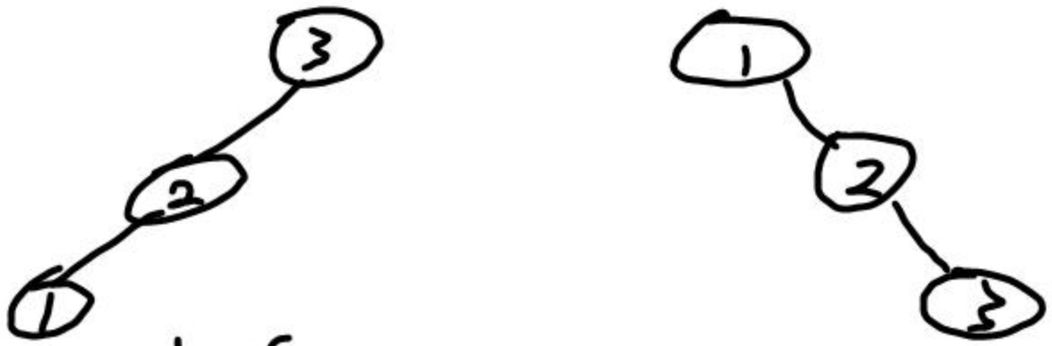
AVL Tree - Binary search tree with the balance property that the height of the left and right subchild cannot differ by more than one.



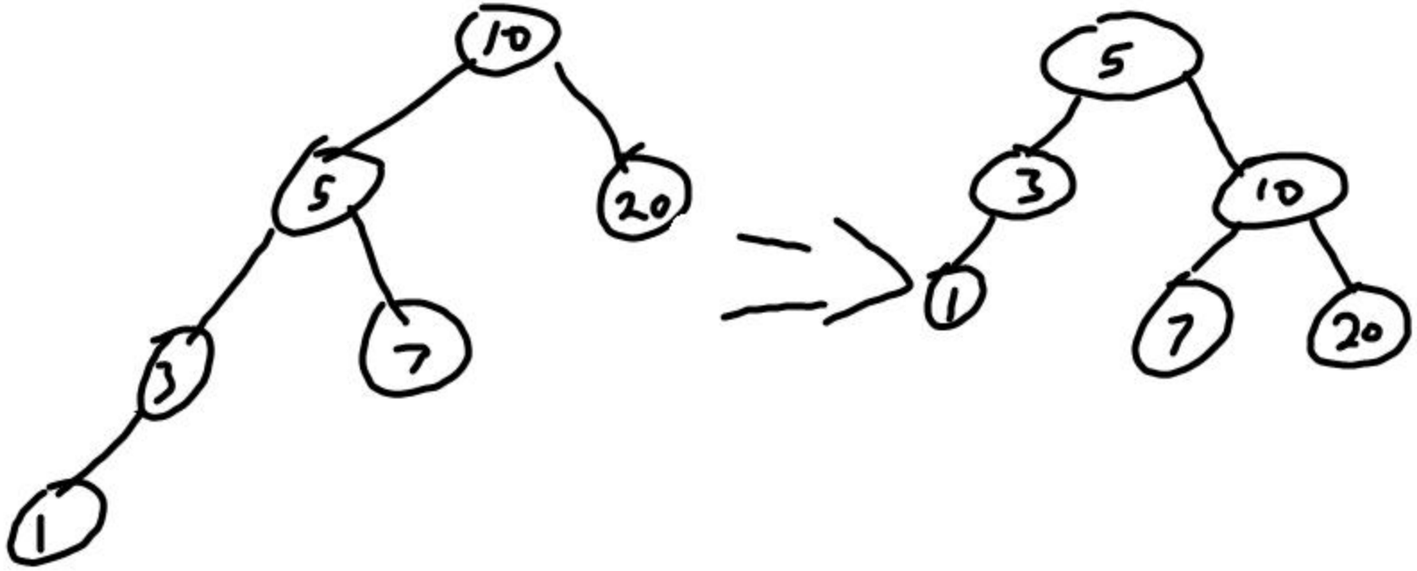


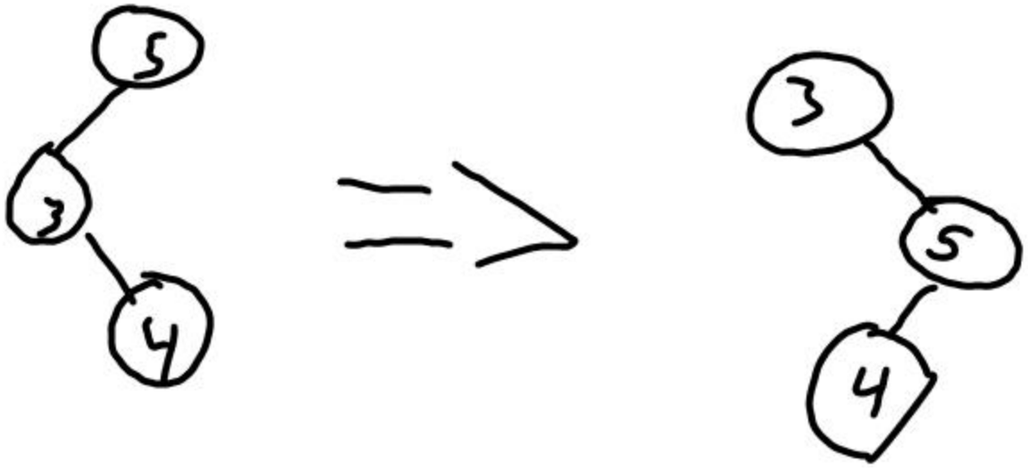


Single rotation with left child.



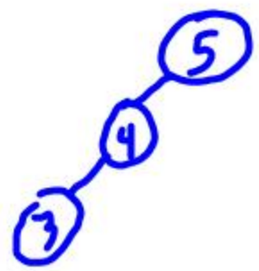
Left-Left > Left-right
= Single rotation



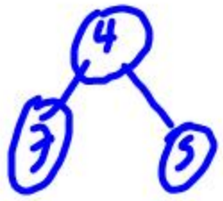


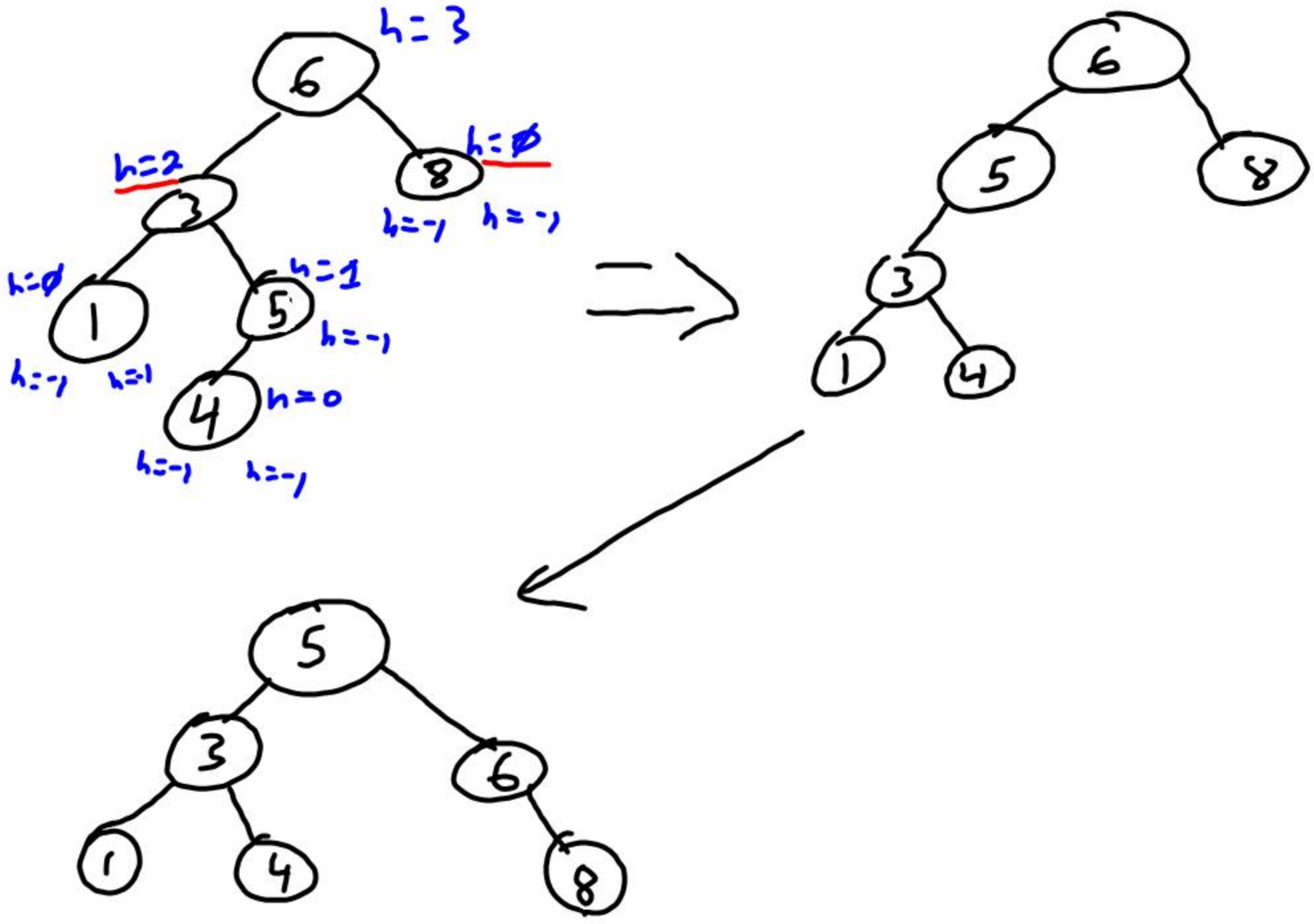
Left-right > Left-left
Double rotation

```
DoubleRotationWithLeftChild (NodeType* root)
{
    SingleRotationWithRightChild (root->left)
```



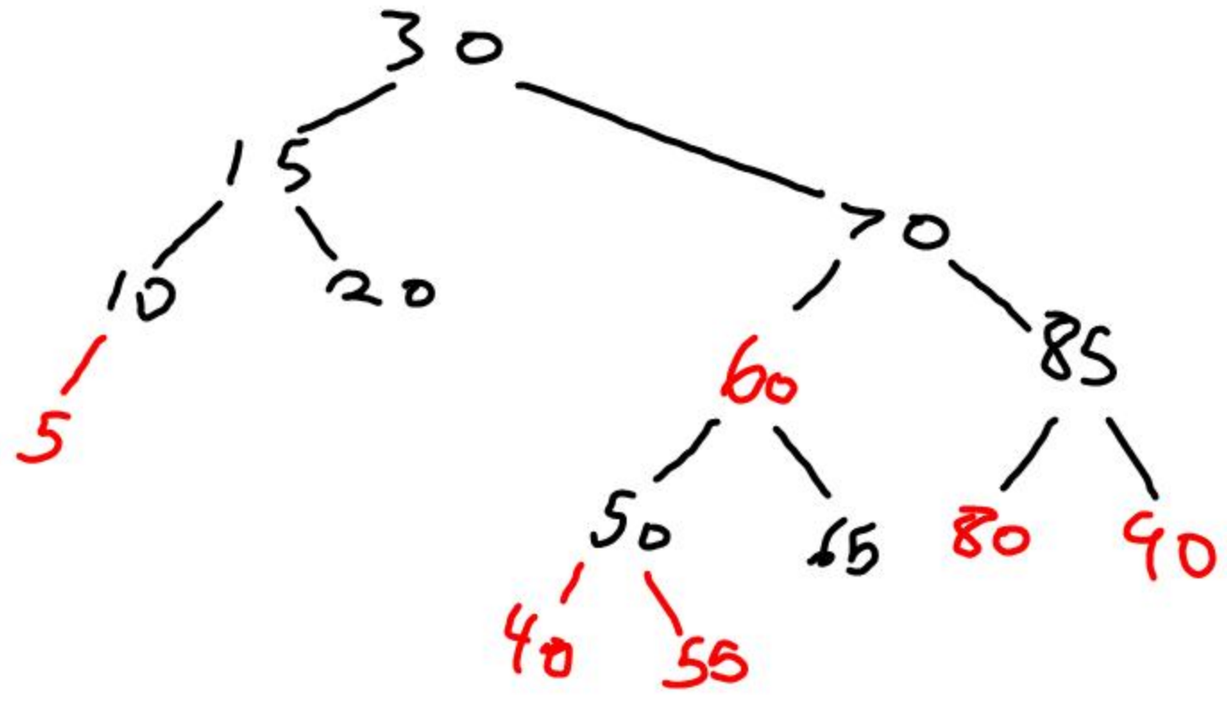

```
    SingleRotationWithLeftChild (root);
}
```




h=3
6
h=2
3
8
h=0
h=-1
h=-1
h=0
1
5
h=1
h=-1
h=-1
h=1
4
h=0
h=-1
h=-1
6
5
8
3
1
4
5
3
6
1
4
8

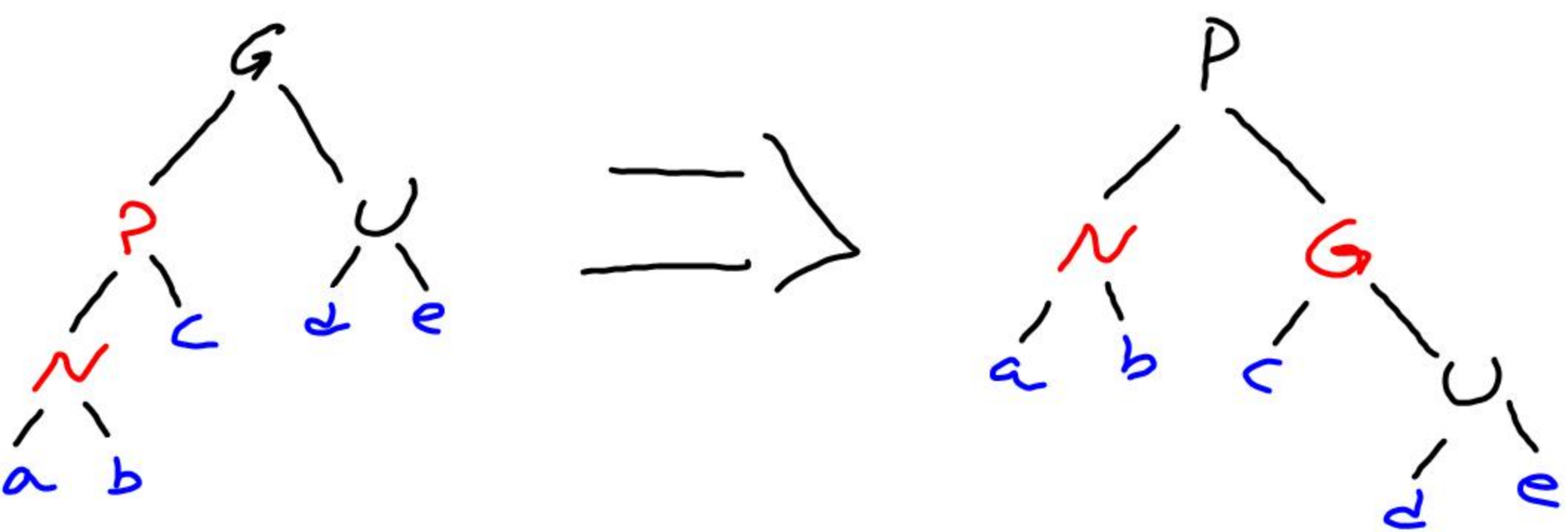
Red-Black Trees

1) Every node has a color, red or black.
2) root is always black.
3) if a node is red, it's children must be black.
4) Every path from a node to a null must traverse the same number of black nodes.



How to implement:

1) new nodes are always colored red.
   a) Parent is Black = Done
   b) Parent is red = problem
      i) if uncle is black (null is black)
         A) if new node is outside - Single rotation (GrandParent)
         B) if new node is inside - Double rotation (GrandParent)
      ii) if uncle is red - AVOID / Impossible.
2) As You go DOWN to find the insertion point, if you encounter a Black Parent with two red children, recolor to red Parent with two black children. if GrandParent is red, rotate.



G

N