```
class TreeNode :
    def init (self, val)
        self.val = val
        self.r = None
        self.l = None
        self.height = 1

def insert (self, root, key)

    if root == None
        return TreeNode(key)
    elif key < root.val
        root.left = self.insert(root.left, key)
    else
        root.right = self.insert(root.right, key)

    balance = self.getbalan
    bal = max (getbal( leftsubtree, rightsubtree)

    if bal > 1 and key < root.left.val
        return self.rightRotate(root)

    if bal < -1 and key > root.right.val
        return self.leftRotate(root)

    if bal > 1 and key > root.left.val
        root.left = self.leftRotate(root.left)
        return self.RightRotate(root)

    if bal < -1 and key < root.right.val
        root.right = self.rightRotate(root.right)
        return self.leftRotate(Root)
```

DATE: PAGE:

```
def delete (self, root, key)

if not root:
    Return root

elif key < root.val:
    root.left = self.delete (root.left, key)

elif key > root.val
    root.right = self.delete (root.right, key)

else:
    if root.left is None:
        temp = root.right
        root = None
        return temp

    elif root.right is None:
        temp = root.left
        root = None
        return temp

    temp = self.getmin val Node (root.right)
    root.val = temp.val
    root.right = self.del (root.right, temp.val)

if root is None:
    return root

root.height = getroot height
bal = self.getBal(root)
```

DATE: PAGE:

```
if bal > 1 & self.getBal(root.left) >= 0
    return self.rightRotate(root)

if bal < -1 and self.get(root.right) <= 0
    return self.leftRotate(root)

if bal > 1 and self.getBal(root.left) < 0
    root.left = self.leftRotate(root.left)
    return self.RightRotate(root)

if balance < -1 and self.getBal(root.right) > 0
    root.right = self.rightRotate(root.right)
    return self.leftRotate(root)
return root
```