# 红黑树

红黑树是平衡二叉树的一种实现。平衡二叉树的定义：一颗空树或左右子树的高度差不超过1。

# 1.红黑树的规则

（1）树上的节点不是黑色就是红色。

（2）根节点总是黑色的。

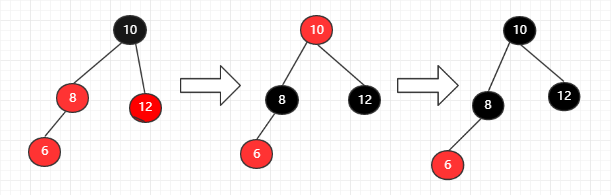
（3）红色节点的子节点一定是黑色的，反之，不一定。

（4）根节点到叶子节点或者空子节点（nil节点）的黑色节点的数量总是相同的。

# 2.红黑树的变换

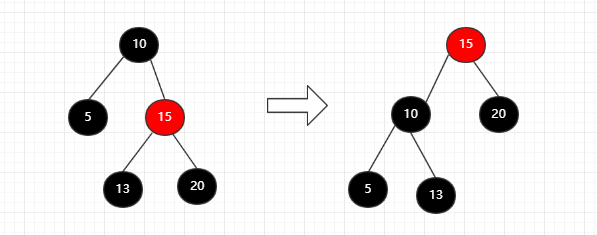
红黑树主要通过变色，左旋和右旋保证红黑树的基本规则。

## 2.1变色



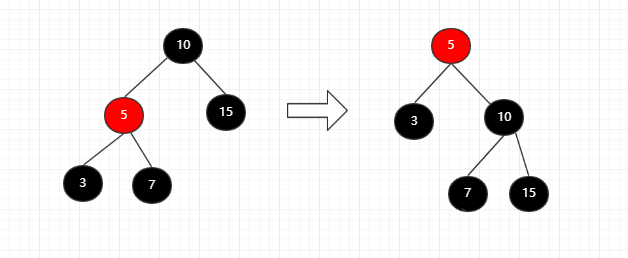
## 2.2左旋

将一颗向右倾斜的树变为向左倾斜。



## 2.3右旋

将一颗向左倾斜的树变为向右倾斜。



# 红黑树基本操作

在插入和删除的过程中，需要通过变色和旋转操作维持红黑树颜色排列的基本规则。

## 3.1插入

一般规定插入红色节点，因为当插入黑色节点时一定会违反规则（4），而插入红色节点，有可能违反规则（3）。

下面看一下插入红色节点的几种情况：

1.插入节点的父节点为黑色，不会违反红黑树的规则。

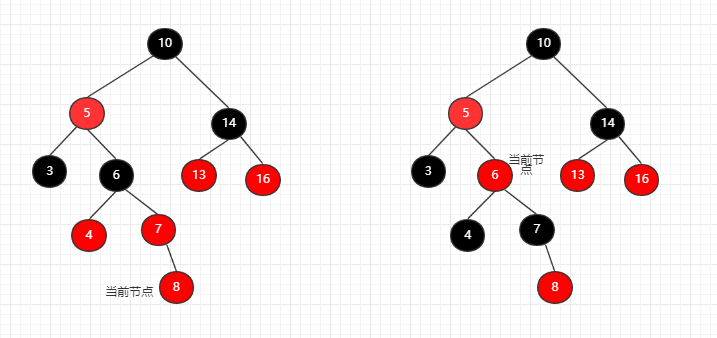
2.插入节点的父节点为红色，

1. 叔叔节点为红色。
2. 叔叔节点为黑色，插入节点为右子节点。
3. 叔叔节点为黑色，插入节点为左子节点。

下面看一下具体的解决办法（以下统一插入节点为当前节点）：

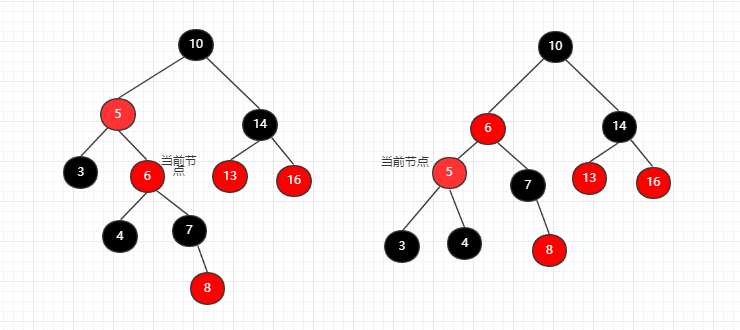
1不需要做处理。

1. a将父节点与叔叔节点涂黑，祖父节点涂红，将祖父节点作为新的当前节点，继续该算法。



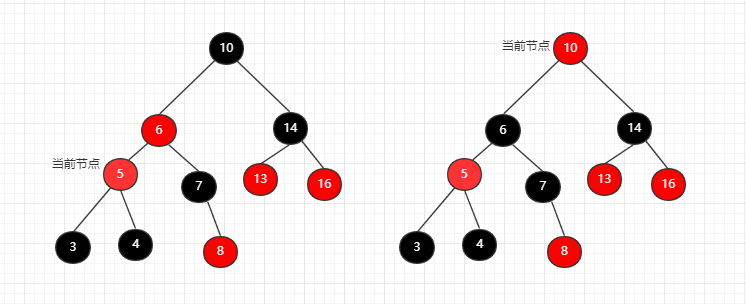
1. b当前节点的父节点为左子树的情况，即上图右图所示：

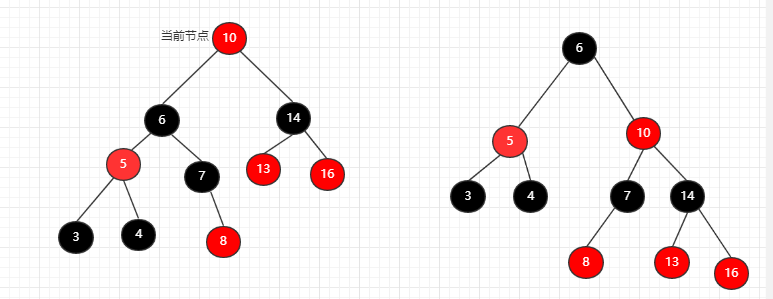
将当前节点的父节点作为新的当前节点，并已新的当前节点作为支点左旋。



2.c当前节点的父节点为左子树的情况，即上图右图所示：

将父节点涂黑，祖父节点涂红，以祖父节点作为支点右旋。





以上均已父节点为左子树情况讨论的。

当父节点为右子树时：

1. 叔叔节点为黑色，插入节点为左子节点时，将当前节点的父节点作为新的当前节点，并以新的当前节点作为支点右旋。
2. 叔叔节点为黑色，插入节点为右子节点时，将父节点涂黑，祖父节点涂红，以祖父节点做为支点左旋。

## 3.2删除

红黑树在二叉搜索树的删除操作之后加了修改树平衡的操作。二叉搜索树的删除主要分为三种情况，一删除节点不含子节点，二删除节点有一个子节点，三删除节点有两个子节点。而第三种情况可以转换为前两种情况处理。下面我们主要看一下第三种情况：

首先要找到删除节点的后继节点，然后用后继节点替换删除节点，最后删除后继节点。也就是删除节点位置上的规则不会被打破，而打破规则的地方只有后继节点的地方。下面分情况讨论下删除后继节点的几种情况，以后继节点的子节点作为当前节点。

当前节点为左子节点：

1. 当前节点为红色，说明后继节点为黑色，只要将当前节点涂黑就可以恢复红黑树的平衡。
2. 当前节点为黑色，

a.兄弟节点为红色，那么兄弟节点的子节点都为黑色。

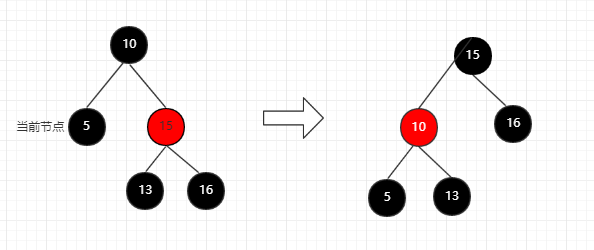
b.当前节点为黑色，兄弟节点为黑色，兄弟节点的子节点都为黑色。

c.当前节点为黑色，兄弟节点为黑色，兄弟节点的左子节点为红色，右子节点为黑色。

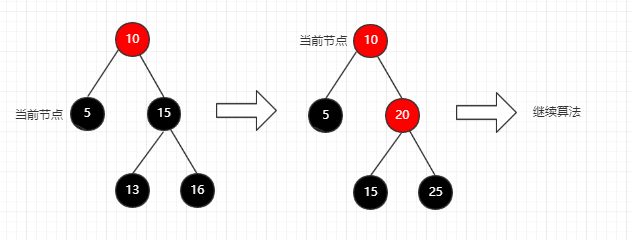
d.当前节点为黑色，兄弟节点为黑色，兄弟节点的右子节点为红色，左子节点为任意颜色。

下面看一下解决方案：

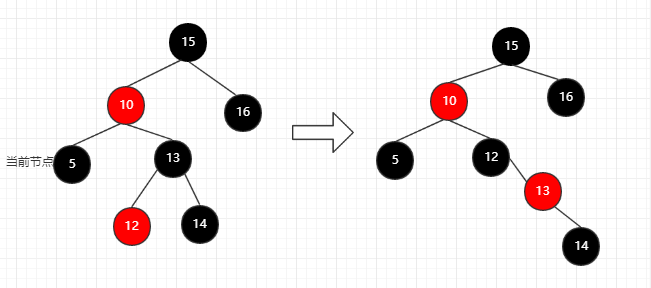
1. a将兄弟节点涂黑，父节点涂红，以父节点为支点右旋。就变成了2.b这种情况，兄弟节点为黑色。



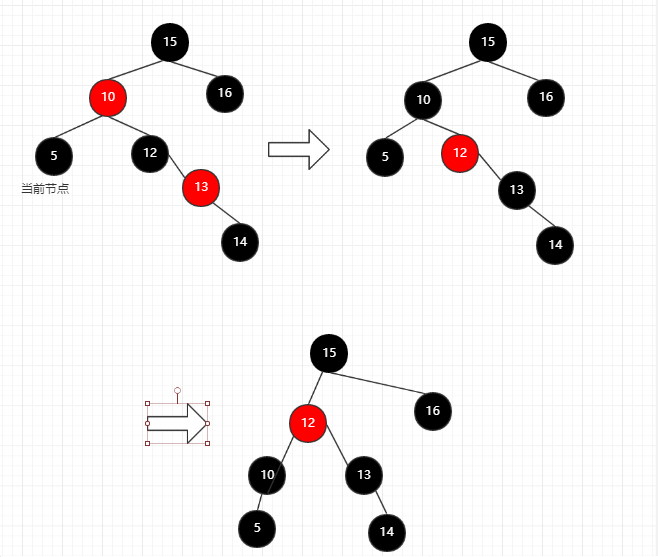
1. b将兄弟节点涂红，将父节点作为新的当前节点，继续该算法。



1. c将兄弟节点涂红，兄弟的左子节点涂黑，以兄弟节点作为支点右旋。就变成了2.d这种情况。



1. d将兄弟节点涂成父节点的颜色，将父节点涂黑，兄弟节点的右子节点涂黑，然后以当前节点的父节点作为支点左旋。



当当前节点为右子节点：

1. c当兄弟节点为黑色，兄弟节点的右子节点为红色，左子节点为黑色。将兄弟节点涂红，兄弟节点右子节点涂黑，以兄弟节点作为支点左旋。

2.d当兄弟节点为黑色，兄弟节点的左子节点为红色，右子节点为任意颜色。将兄弟节点涂成父节点的颜色，将父节点涂黑，兄弟节点的左子节点涂黑，以当前节点的父节点右旋。

源码见：RBSTree.java