**在红黑树之前：**

1.二叉排序树（二叉查找树）、AVL树（平衡二叉树）。

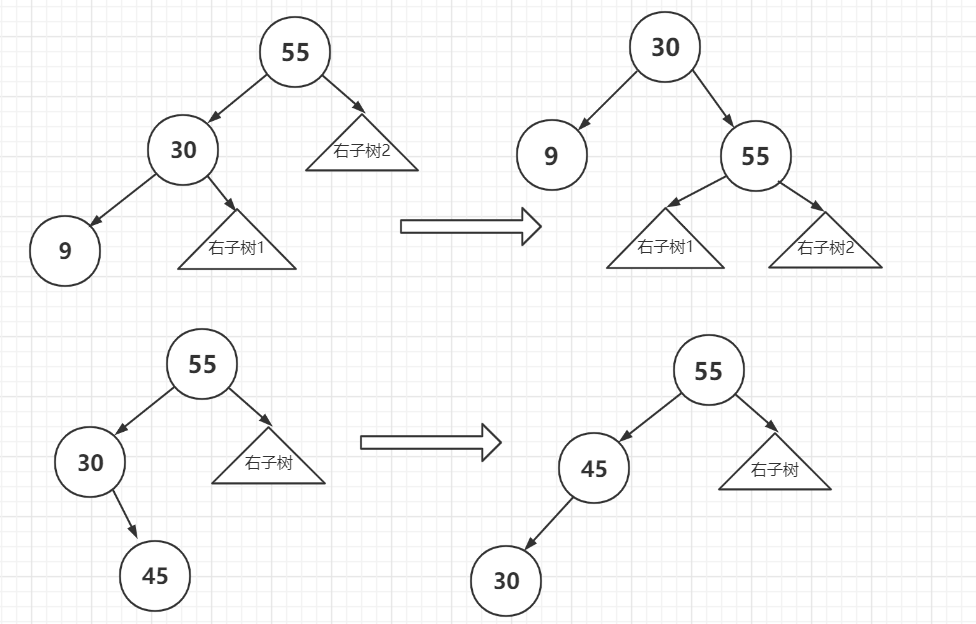
二叉排序树：中序遍历为一个有序序列。

AVL树：满足二叉排序树的基础上，左右子树深度相差不大于1。

红黑树：AVL树的变种。

2.旋转操作。

在不改变二叉排序树性质的基础上，保证AVL树深度性质的操作。



上面为简单的示意图。上图为右旋操作，下图为另一种情况，为上图操作的前置操作（先左旋再右旋）。除此之外，还有在父节点右子树的左旋操作，及先右旋再左旋操作，原理一样，不再赘述。

**红黑树：**

**性质：**

1. 每个节点是红色/黑色。
2. 根节点为黑色。
3. 红色节点的子节点必须是黑色。
4. 从一个节点到null的每一条路径都必须包含的黑色节点数目相同。

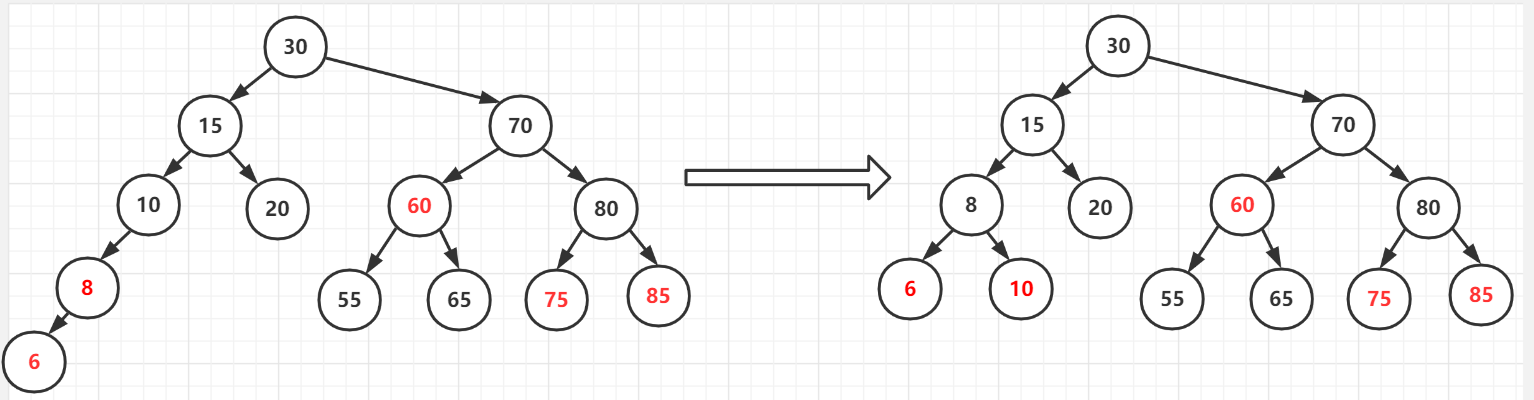
**插入：**

将每个插入的节点置为红色，这样不会直接影响到性质4，只需要关注性质3即可。

设插入节点为X，其父节点为P，父节点的兄弟节点为S，其祖父节点为G。

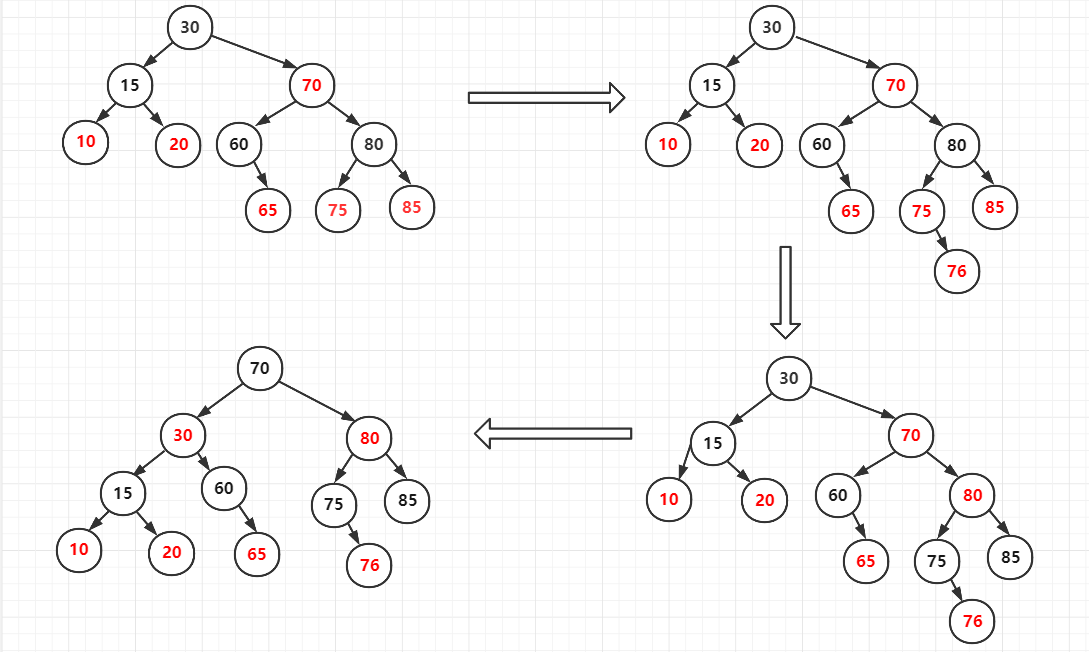
1. 若父节点P为黑色，则无需处理即可满足性质3。
2. 若父节点P为红色，父节点的兄弟节点S为黑色或null。

旋转，父节点变黑，祖父节点变红。



1. 若父节点P为红色，父节点的兄弟节点S也为红色。

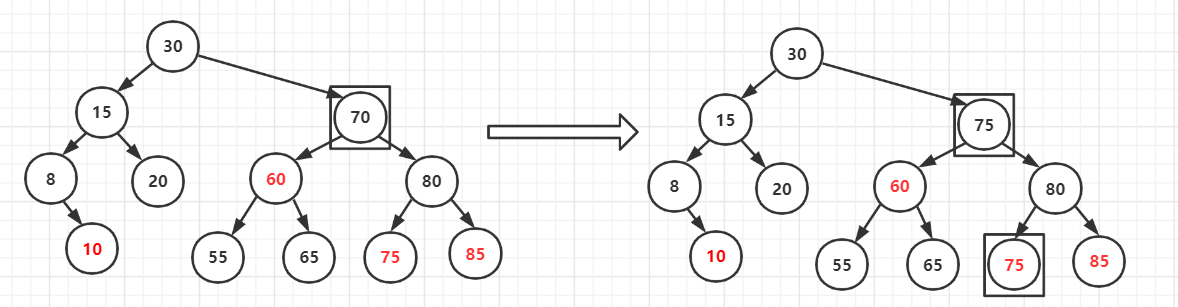
将父节点P和其兄弟节点S置为黑色，将祖父节点G置为红色，此时G的子树整体已满足性质3。然后以G为基准自底向上地递归判断是否满足性质3。



**删除：**

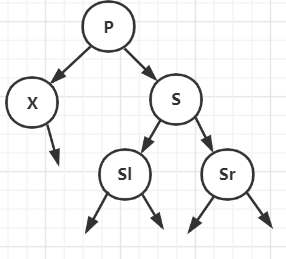
删除节点可以转换成删除叶子节点的方式：

1. 被删除节点只有左子树，找到左子树的最大节点进行替换，然后删除该节点。
2. 被删除节点有右子树，找到右子树的最小节点进行替换，然后删除该节点。

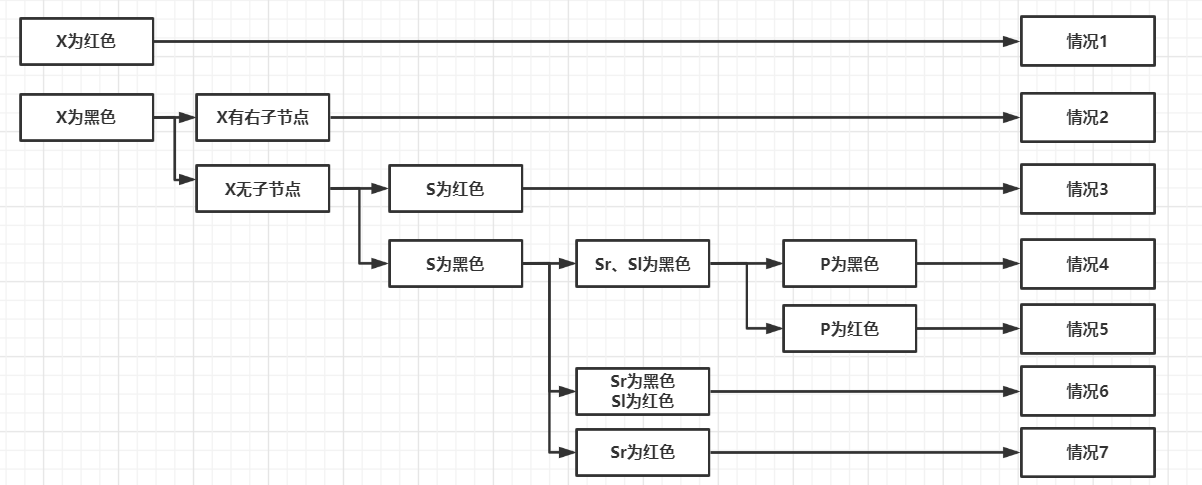


上图由删除节点70，转换成删除叶子节点75。

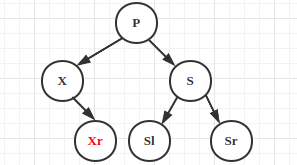
设删除目标节点为X，其父节点为P，父节点的兄弟节点为S，S的两个孩子节点分别为Sl和Sr。



以下以上面情况2（找右子树最小节点替换）来讨论。情况1反着来即可。



1. X为红色，则X一定没有子节点，删除一个红色节点不影响红黑树的性质，直接删除即可。
2. X为黑色。若X有子节点，那么一定是右子节点，并且为红色。此时删除X，并且使用其子节点替换即可。



1. X为黑色，且无子节点。

S为红色。

此时P、Sl、Sr一定为黑色。

1. X为黑色，且无子节点。

S为黑色。

P为黑色。

Sl、Sr为黑色。

1. X为黑色，且无子节点。

S为黑色。

P为红色。

Sl、Sr为黑色。

1. X为黑色，且无子节点。

S为黑色。

Sl为红色，Sr为黑色。

1. X为黑色，且无子节点。

S为黑色。

Sr为红色。