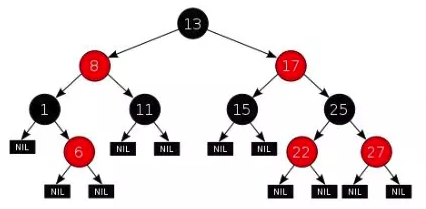
# 红黑树



## 红黑树的 5 个特性

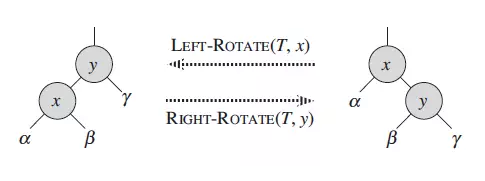
1. 每个节点要么是红色，要么是黑色；
2. 根节点永远是黑色的；
3. 所有的叶节点都是是黑色的（注意这里说叶子节点其实是上图中的 NIL 节点）；
4. 每个红色节点的两个子节点一定都是黑色；
5. 从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点；

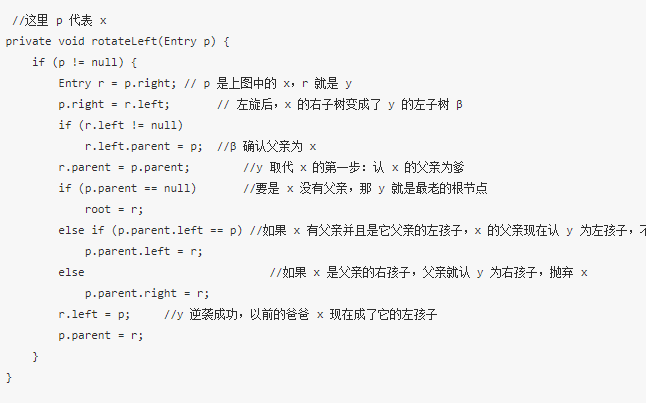
注意：

性质 3 中指定红黑树的每个叶子节点都是空节点，而且并叶子节点都是黑色。但 Java 实现的红黑树将使用 null 来代表空节点，因此遍历红黑树时将看不到黑色的叶子节点，反而看到每个叶子节点都是红色的。

性质 4 的意思是：从每个根到节点的路径上不会有两个连续的红色节点，但黑色节点是可以连续的。 因此若给定黑色节点的个数 N，最短路径的情况是连续的 N 个黑色，树的高度为 N - 1;最长路径的情况为节点红黑相间，树的高度为 2(N - 1)。（有一个空节点）

## 红黑树的左旋右旋





可以看到，x 节点的左旋就是把 x 变成 右孩子 y 的左孩子，同时把 y 的左孩子送给 x 当右子树。

简单点记就是：左旋把右子树里的一个节点（上图 β）移动到了左子树。

## 红黑树的平衡插入

红黑树的第 5 条特征规定，任一节点到它子树的每个叶子节点的路径中都包含同样数量的黑节点。也就是说当我们往红黑树中插入一个黑色节点时，会违背这条特征。

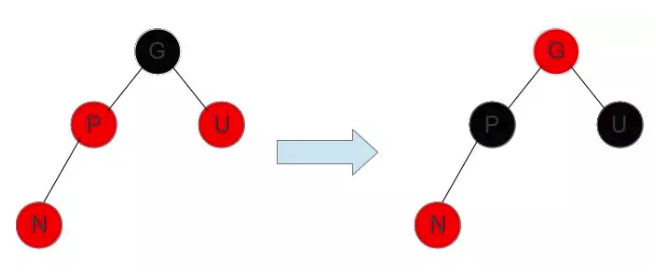
同时第 4 条特征规定红色节点的左右孩子一定都是黑色节点，当我们给一个红色节点下插入一个红色节点时，会违背这条特征。

### 调整思想

前面说了，插入一个节点后要担心违反特征 4 和 5，数学里最常用的一个解题技巧就是把多个未知数化解成一个未知数。我们这里采用同样的技巧，把插入的节点直接染成红色，这样就不会影响特征 5，只要专心调整满足特征 4 就好了。这样比同时满足 4、5 要简单一些。

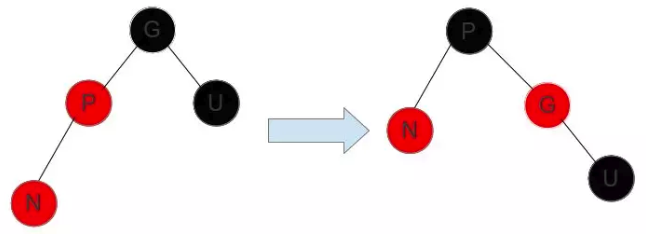
#### 插入、染红后的调整有 2 种情况：

**情况1.父亲节点和叔叔节点都是红色：**



假设插入的是节点 N，这时父亲节点 P 和叔叔节点 U 都是红色，爷爷节点 G 一定是黑色。红色节点的孩子不能是红色，这时不管 N 是 P 的左孩子还是右孩子，只要同时把 P 和 U 染成黑色，G 染成红色即可。这样这个子树左右两边黑色个数一致，也满足特征 4。但是这样改变后 G 染成红色，G 的父亲如果是红色岂不是又违反特征 4 了？ 这个问题和我们插入、染红后一致，因此需要以 爷爷节点 G 为新的调整节点，再次进行调整操作，以此循环，直到父亲节点不是红的，就没有问题了

**情况2.父亲节点为红色，叔叔节点为黑色：**



假设插入的是节点 N，这时父亲节点 P 是红色，叔叔节点 U 是黑色，爷爷节点 G 一定是黑色。红色节点的孩子不能是红色，但是直接把父亲节点 P 涂成黑色也不行，这条路径多了个黑色节点。怎么办呢？既然改变不了你，那我们就此别过吧，我换一个更适合我的！我们怎么把 P 弄走呢？看来看去，还是右旋最合适，通过把 爷爷节点 G 右旋，P 变成了这个子树的根节点，G 变成了 P 的右子树。右旋后 G 跑到了右子树上，这时把 P 变成黑的，多了一个黑节点，再把 G 变成红的，就平衡了。

## 红黑树的平衡删除