首先了解红黑树特性：

1. 根节点必须为黑
2. 节点不是红就是黑
3. 叶子结点必须是黑，也就是nil，其实无所谓
4. 两个红不能互为父子节点
5. 每条从根节点到叶子结点的路径包含的黑节点数目，都是相同的（插入的算法决定的）

插入节点的步骤：

1. 插入节点首先上色为红色，因为红色好调整
2. 按二叉排序树找到插入位置后，开始调整

插入的调整：

情形1：

父节点与叔节点都为红，由此推断祖父必须为黑，只需要将祖父变红，父亲和叔叔变黑即可，以祖父节点为当前节点递归

情形2，默认父节点为红，叔节点为黑或者为nil：

2.1：当前节点作为左子树，父节点作为右子树，也就是>型

处理方法：父节点右旋，变成 \ 型

2.2：当前节点作为右子树，父节点作为左子树，也就是<型

处理方法：父节点左旋，变成 / 型

2.3：当前节点作为右子树，父节点作为右子树，也就是 \ 型

处理方法：祖父节点左旋，祖父变红，父亲变黑。

2.4：当前节点作为左子树，父节点作为左子树，也就是 / 型

处理方法：祖父节点右旋，祖父变红，父亲变黑。

总结：

<型和>型，都只是中间状态，将其转化成 \ /型，再对祖父节点操作，插入完毕。

红黑树与AVL的性能对比：

查找 AVL占优

插入和删除 红黑树占优

因为红黑树保证插入每次都能在三次以内完成，减少了复杂度。

而红黑树的深度，由其特性可知，两棵树的深度之差，不会太多，因为黑节点数目相同，红色节点不能相邻，因此要是该树的黑色路径长度为N，最长的路径也不会超过2N。

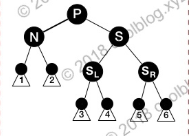
红黑树的删除:

步骤一：删除的节点X分两种情况

1. 删除节点只有或者没有子节点，这就是需要讨论的重点
2. 删除节点有两个子节点，这种情况与二叉平衡树的删除一样，先找到删除节点的后继或者前驱节点Y，交换X和Y的值，从而问题变成删除Y节点，转化成第一点讨论的情况
3. 如果X节点颜色为红，经过前面1.2两点，这里讨论的N节点最多只有一个子节点，只需将N节点的子节点N顶替上来，红黑树性质不变
4. 如果X节点颜色为黑，就是步骤二讨论的内容

步骤二：删除的关键在于理解下面这张图，有以下几点需要注意：

1. N节点为删除节点X的子节点，N替换了X原来的位置。由于步骤一已经将问题简化成，被删除节点只有或者没有子节点，所以X节点只有N一个子节点，如果X没有子节点，N为NIL节点，颜色默认为黑色
2. P为N的父节点，S为N的兄弟节点。经过P点的两条路径，PN&PS。PN路径由于删除了X节点这个黑节点（这里X节点经过步骤一，必须为黑），所以PN和PS两条路径已经不平衡
3. PN为少了一个黑节点的路径，称之为短路径，另外一条称之为兄弟路径。后面只讨论短路径位于左侧这种情况。（短路径位于右侧的情况，可以通过反转操作，也就是，左旋换成右旋，右旋变成左旋）



步骤三：现在情况简化为只讨论上图，讨论的范围包括P,S,SL,SR四个节点的颜色情况，需要分为几种情况

情况1：

P红，其他均为黑

策略1：

PS换色（P黑化，S红化）

结果1：恢复平衡

短路径增加一个黑节点，兄弟路径不变。

情况2：

S红，其他均为黑

策略2：

PS换色&P左旋

 结果2：

短路径依旧存在，但转化成情况1，所以再进行一次换色，问题解决

情况3：

全黑

策略3：

S变红

结果3：

经过P节点的两条路径现在都少了一个黑节点，所以当前节点从N转化成了P，接着继续递归分析P。

情况4：

SL红（因此S黑），SR黑，P无所谓

策略4：

S与SL换色，S右旋

结果4：

PN依旧为短路径，但是转化为情况5

情况5：

SR红（因此S黑），SL无所谓，P无所谓

策略5：

S变成P的颜色

P黑化

SR黑化

P左旋

结果5：恢复平衡

总结：上述情况看似不好记忆，但可以推理出来，下面是帮助记忆的方法。

重点在于五个节点：N  P  S  SL  SR，N的颜色必须为黑，因此就只有四个节点的颜色需要讨论

前面三种比较好记忆，要么全黑，要么除了P，其他全黑，要么除了S，其他全黑。

剩下的是SL或者SR为红的情况，称之为复杂情况

第一种复杂情况，SL为红，SR为黑，那么由于红黑树特性，S必须为黑。剩下P点无所谓

第二种复杂情况，SR为红，SL无所谓，S必须为黑，剩下P点无所谓

第一种复杂情况需要转化为第二种，因此旋转和变色的策略很好推导

第二种复杂情况，可以根据PN短路径需要增加一个黑节点，而SL和SR这两条路径需要和原来保持一致的黑色数，变色和旋转也是很好推导的

下面的图是上面几种情况的可视化总结

