什么是红黑树，为什么要学习红黑树？

红黑树是二叉搜索树(binary search tree)中的一种，它是一种非常重要的数据结构。相较于avl树，它有更小的插入删除开销。

认识什么是红黑树

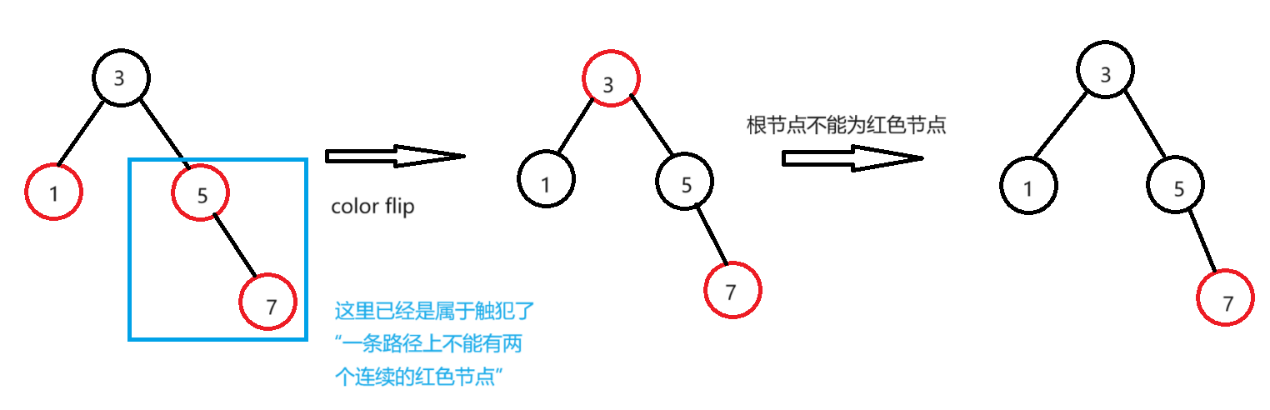
简单说红黑树就是二叉搜索树的节点上多了一个颜色属性”color”，它要么是红要么是黑。通过一系列规则的限制，保证树的有效。接下来我来介绍一下它的规则

1. 树的根节点必须得是黑色
2. 红黑树的每条路径上不能出现连续的红色节点
3. 红黑树的每条路径上的黑色节点数相同

如何保证红黑树是有效的？

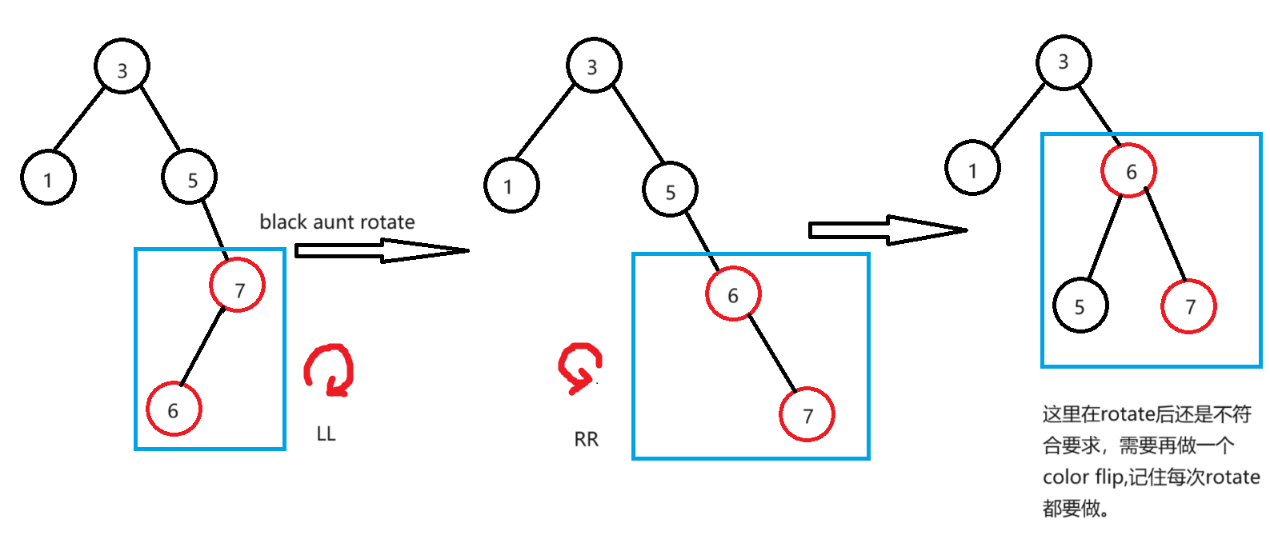
1. 在插入一个新的节点后，如果它有一个红色的叔叔节点，那么我们做”color flip”也就是颜色翻转,
2. 在插入一个新的节点后，如果它有一个黑的叔叔节点，那么我们做”rotate”(具体做什么样的旋转根据数的平衡因子决定)
3. 在每次做完旋转后都要立即做一次”color flip”

情景一：在插入一个节点，如果红黑树“不平衡”或者说不再有效了，需要根据插入节点的叔叔节点的颜色来判断要做什么操作。

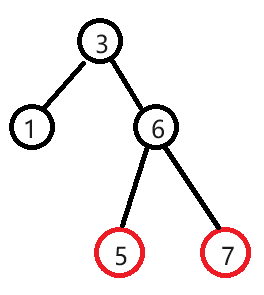


红色的祖父节点，黑色的父或叔叔节点，红色的自己。但上图的情况是，红黑树的根节点必须要是黑色的。

情景二a：在插入一个节点后红黑树不平衡了，但是它有一个黑色的叔叔节点，此时我们要做颜色翻转

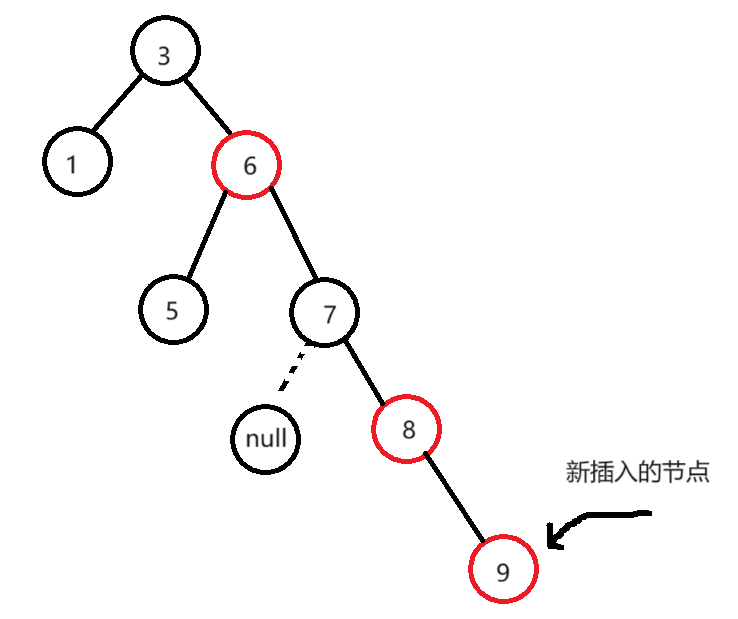


观察上图，当我们插入了新节点6的时候，就已经触犯了“在一条路径上不能有两个连续红色节点”的规则，且此时新插入的节点6有一个黑色的叔叔节点（记住值为null的节点被视作黑色的节点）。利用我们之前学过的avl树的知识，要想将这颗树边平衡，我们需要做向左的双旋转（tip:整体往那边偏，就要做相反方向的旋转。）当我们做了旋转之后，我们需要做“旋转之后特有的颜色翻转”也就是“black father and child is red”。

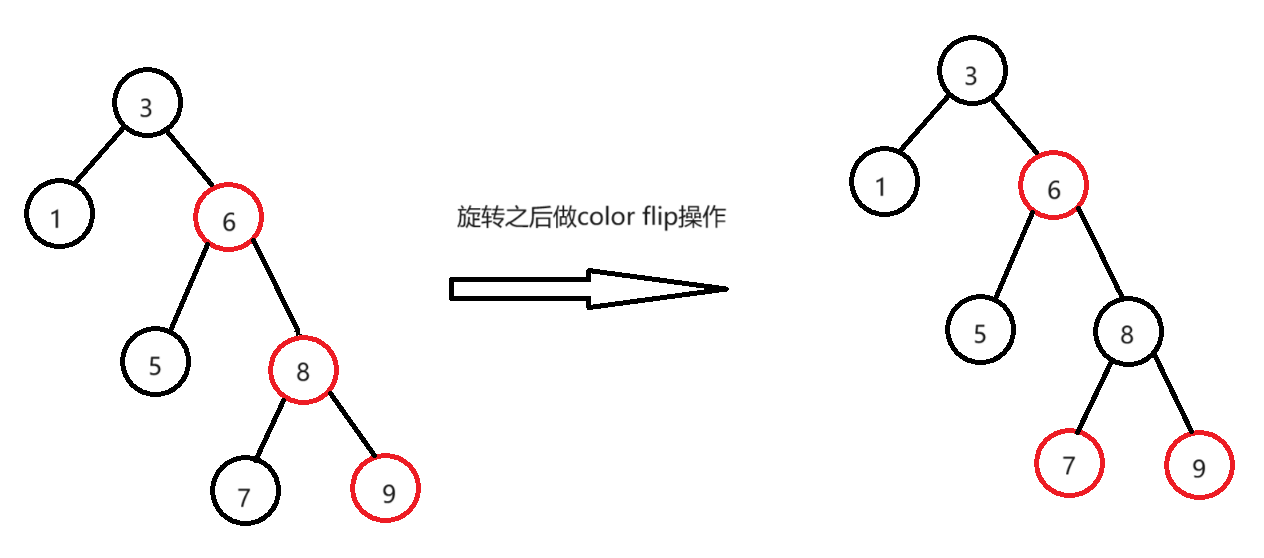


此时，可以观察到做了”color flip”的树已经是符合要求的了。

情景二b:当我插入了一个节点后，发现树不平衡了，且同样他有一个黑色的叔叔节点。



当我们插入了节点9之后，此时我们发现在红黑树的一条路径上出现了两个连续的红色节点，这是不符合二叉树的规则的，又因9这个节点有一个黑色的叔叔节点，所以我们需要做向左的单旋转。在旋转过后做一个”旋转过后的color filp”。



对于上面描述的场景我坐一下总结：

1. 黑色的叔叔节点，旋转。
2. 红色的叔叔节点，color flip。
3. 在每次旋转之后，要做color flip。

红黑树是二叉树的一种，一个合规的红黑树有可能也是一个avl树，也有可能不是。但相较于avl树，它有更小的插入删除开销。它的核心还是avl树的旋转操作，只不过在此基础上加了两种颜色翻转的操作。后面我会尝试使用js完成红黑树的构建。