R-B Tree，全称是Red-Black Tree，又称为“红黑树”，它一种特殊的二叉查找树。红黑树的每个节点上都有存储位表示节点的颜色，可以是红(Red)或黑(Black)。

**红黑树的特性**:  
**（1）每个节点或者是黑色，或者是红色。**  
**（2）根节点是黑色。**  
**（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]**  
**（4）如果一个节点是红色的，则它的子节点必须是黑色的。**  
**（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。**

**注意**：  
(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。  
(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

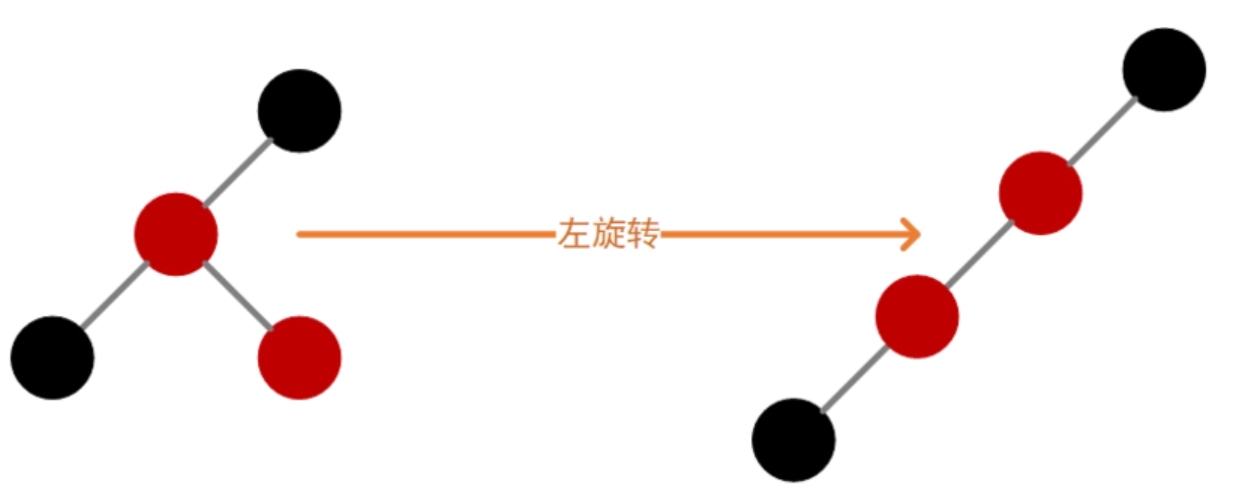


红黑树的时间复杂度是O（log（n））,效率比较高。

**红黑树的自平衡策略**

对于一个红黑树，常用操作是增删改查，其中查和改不会破坏数据结构，但是这种就会破坏原本有的数据结构。平衡策略可以包含三种:**左旋转，右旋转以及变色。**

1. 对于当前结点而言，如果右子结点为红色，左子结点为黑色，则执行左旋转，如下图：



1. 对于当前结点而言，如果左子、左孙子结点均为红色，则执行右旋转，如下图：



**3 红黑树和AVL树的比较**

1. **红黑树并不追求“完全平衡**”——它只要求部分地达到平衡要求，降低了对旋转的要求，从而提高了性能。

红黑树能够以**O(log2 n)**的时间复杂度进行搜索、插入、删除操作。此外，由于它的设计，任何不平衡都会在三次旋转之内解决。当然，还有一些更好的，但实现起来更复杂的数据结构，能够做到一步旋转之内达到平衡，但红黑树能够给我们一个比较“便宜”的解决方案。**红黑树的算法时间复杂度和AVL相同，但统计性能比AVL树更高**。

就插入节点导致树失衡的情况，AVL和RB-Tree都是最多两次树旋转来实现复衡rebalance，旋转的量级是O(1)

删除节点导致失衡，AVL需要维护从被删除节点到根节点root这条路径上所有节点的平衡，旋转的量级为O(logN)，而RB-Tree最多只需要旋转3次实现复衡，只需O(1)，所以说RB-Tree删除节点的rebalance的效率更高，开销更小！

AVL的结构相较于RB-Tree更为平衡，插入和删除引起失衡，如2所述，RB-Tree复衡效率更高；当然，由于AVL高度平衡，因此AVL的Search效率更高啦。