红黑树属于平衡二叉树。它不严格是因为它不是严格控制左、右子树高度或节点数之差小于等于1，但红黑树高度依然是平均log(n)，且最坏情况高度不会超过2log(n)。

红黑树(red-black tree) 是一棵满足下述性质的二叉查找树：

1. 每一个结点要么是红色，要么是黑色。

2. 根结点是黑色的。

3. 所有叶子结点都是黑色的（实际上都是Null指针，下图用NIL表示）。叶子结点不包含任何关键字信息，所有查询关键字都在非终结点上。

4. 每个红色结点的两个子节点必须是黑色的。换句话说：从每个叶子到根的所有路径上不能有两个连续的红色结点

5. 从任一结点到其每个叶子的所有路径都包含相同数目的黑色结点

红黑树相关定理

1. 从根到叶子的最长的可能路径不多于最短的可能路径的两倍长。

      根据上面的性质5我们知道上图的红黑树每条路径上都是3个黑结点。因此最短路径长度为2(没有红结点的路径)。再根据性质4(两个红结点不能相连)和性质1，2(叶子和根必须是黑结点)。那么我们可以得出：一条具有3个黑结点的路径上最多只能有2个红结点(红黑间隔存在)。也就是说黑深度为2（根结点也是黑色）的红黑树最长路径为4，最短路径为2。从这一点我们可以看出红黑树是 大致平衡的。 (当然比平衡二叉树要差一些，AVL的平衡因子最多为1)

2. 红黑树的树高(h)不大于两倍的红黑树的黑深度(bd)，即h<=2bd

      根据定理1，我们不难说明这一点。bd是红黑树的最短路径长度。而可能的最长路径长度(树高的最大值)就是红黑相间的路径，等于2bd。因此h<=2bd。

3. 一棵拥有n个内部结点(不包括叶子结点)的红黑树的树高h<=2log(n+1)

      下面我们首先证明一颗有n个内部结点的红黑树满足n>=2^bd-1。这可以用数学归纳法证明，施归纳于树高h。当h=0时，这相当于是一个叶结点，黑高度bd为0，而内部结点数量n为0，此时0>=2^0-1成立。假设树高h<=t时，n>=2^bd-1成立，我们记一颗树高 为t+1的红黑树的根结点的左子树的内部结点数量为nl，右子树的内部结点数量为nr，记这两颗子树的黑高度为bd'（注意这两颗子树的黑高度必然一 样），显然这两颗子树的树高<=t，于是有nl>=2^bd'-1以及nr>=2^bd'-1，将这两个不等式相加有nl+nr>=2^(bd'+1)-2，将该不等式左右加1，得到n>=2^(bd'+1)-1，很显然bd'+1>=bd，于是前面的不等式可以 变为n>=2^bd-1，这样就证明了一颗有n个内部结点的红黑树满足n>=2^bd-1。

        在根据定理2，h<=2bd。即n>=2^(h/2)-1，那么h<=2log(n+1)

        从这里我们能够看出，红黑树的查找长度最多不超过2log(n+1)，因此其查找时间复杂度也是O(log N)级别的。

红黑树的操作

因为每一个红黑树也是一个特化的二叉查找树，因此红黑树上的查找操作与普通二叉查找树上的查找操作相同。然而，在红黑树上进行插入操作和删除操作会导致不 再符合红黑树的性质。恢复红黑树的属性需要少量(O(log n))的颜色变更(实际是非常快速的)和不超过三次树旋转(对于插入操作是两次)。 虽然插入和删除很复杂，但操作时间仍可以保持为 O(log n) 次 。

红黑树的优势

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,任何不平衡都会在3次旋转之内解决。这一点是AVL所不具备的。

而且实际应用中，很多语言都实现了红黑树的数据结构。比如 TreeMap, TreeSet(Java )、 STL(C++)等。