数据结构与算法

Data Structure and Algorithm

极夜酱

目录

0.1 红黑树

0.1.1 红黑树（Red Black Tree）

红黑树是一种自平衡的二叉查找树，除了符合二叉查找树的基本特性外，它还具有如下附加特性：

1. 结点是红色或黑色的。
2. 根结点是黑色的。
3. 叶子结点都是黑色的空结点 NIL。
4. 红色结点的两个子结点都是黑色的，即从叶子到根的所有路径上不能有连续的两个红色结点。
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。

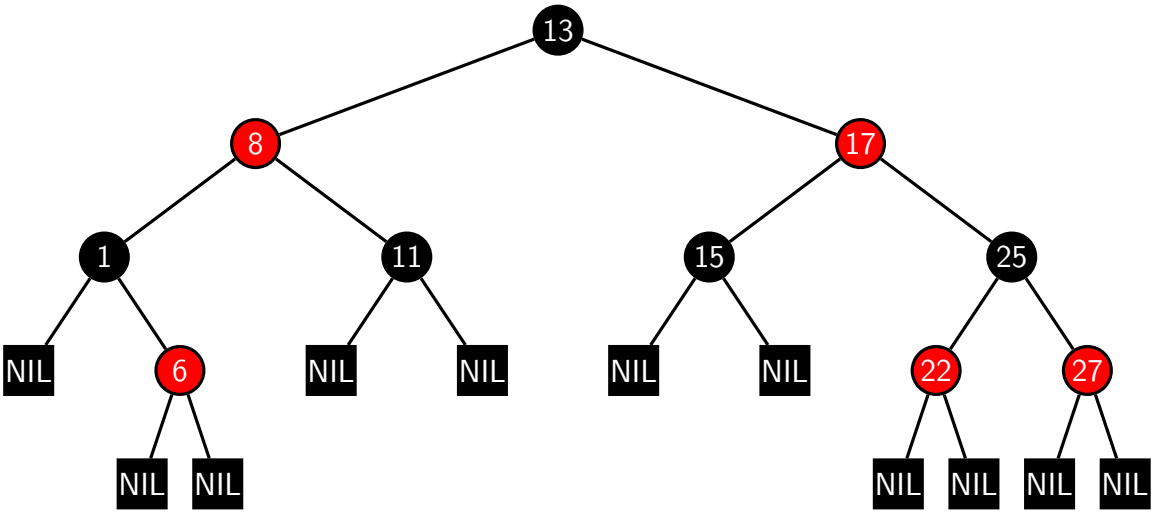


图 1: 红黑树

天呐，这条条框框的太多了吧！

正是因为这些规则限制，才保证了红黑树的自平衡，红黑树从根到叶子的最长路径不会超过最短路径的 2 倍。

红黑树的应用有很多，其中 JDK 的集合类 TreeMap 和 TreeSet 底层就是红黑树实现的。在 Java8 中，连 HashMap 也用到了红黑树。

0.1.2 失衡调整

当插入或删除结点时，红黑树的规则可能被破坏，需要调整使其重新符合规则。

例如向红黑树中插入新结点 14，由于父结点 15 是黑色结点，这种情况不会破坏红黑树的规则，无需做任何调整。

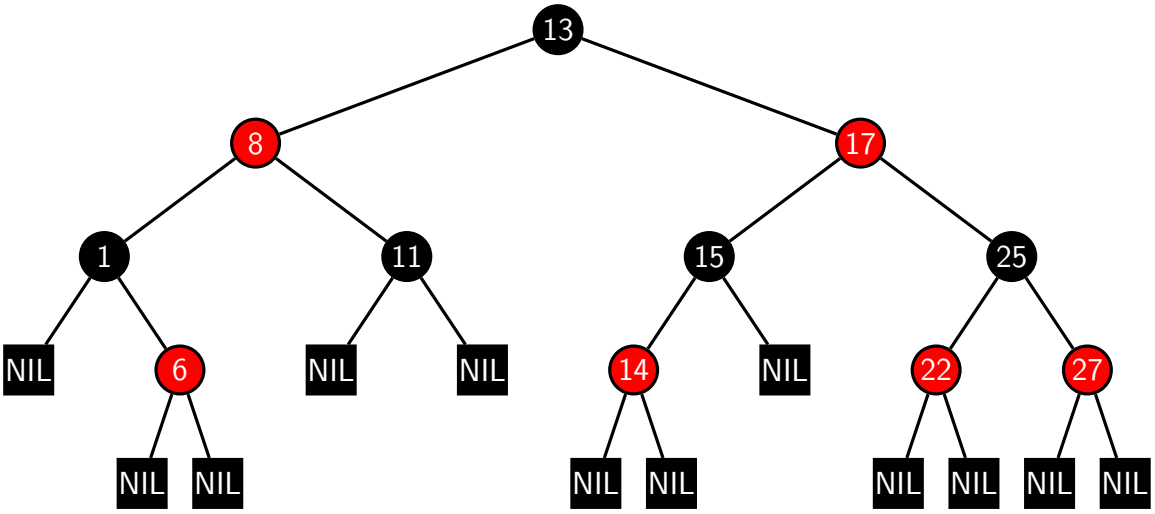


图 2: 插入 14

向红黑树中插入新结点 21，由于父结点 22 是红色结点，违反了红黑树的规则 4（红色结点的两个子结点都是黑色的）。

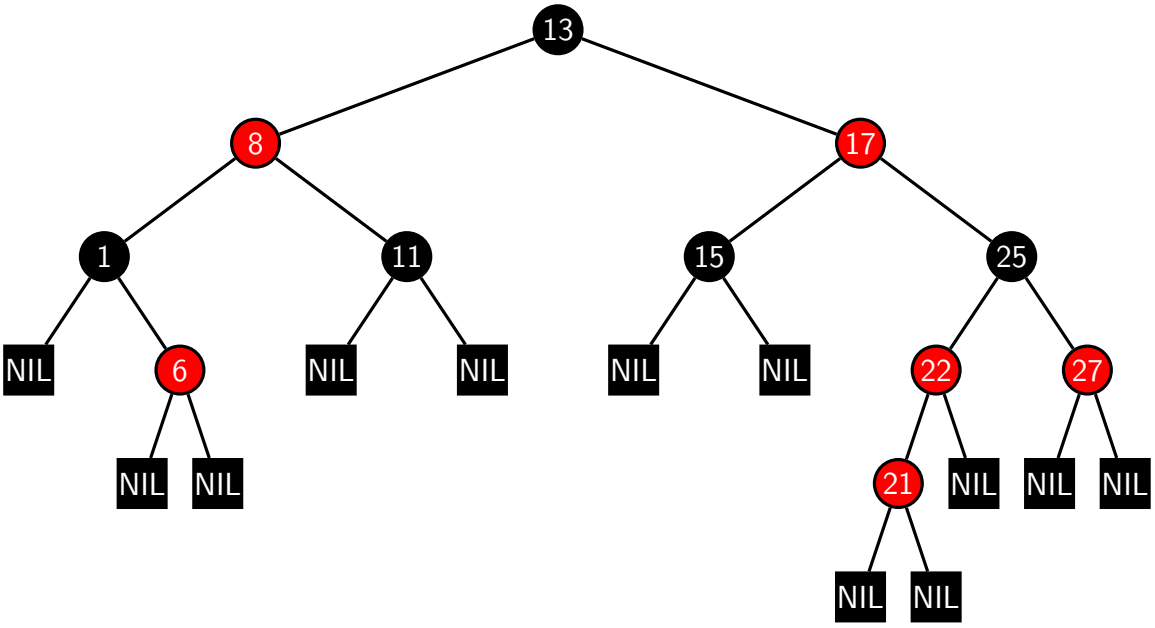


图 3: 插入 21

调整的方法有变色和旋转两种，而旋转又包含左旋转和右旋转两种方式。

为了重新符合红黑树的规则，有时需要把红色结点变为黑色，或是把黑色结点变为红色。

例如对于红黑树的一部分（子树），新插入的结点 Y 是红色结点，它的父结点 X 也是红色结点，不符合规则 4（红色结点的两个子结点都是黑色的），因此可以把结点 X 变为黑色。

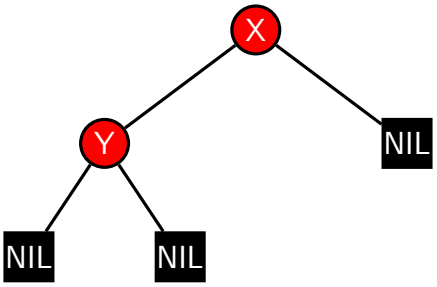


图 4: 违反规则 4

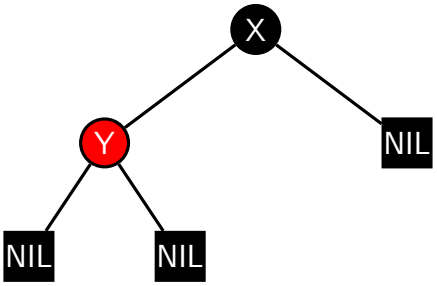


图 5: 变色

但是，如果这是简单的把一个结点变色，会导致相关路径凭空多出一个黑色结点，这样就会打破规则 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），因此还需要其它的调整策略。

0.1.3 红黑树插入结点

红黑树插入新结点时，可以分为五种不同的局面。每一种局面有不同的调整方法。

局面 1

新结点（A）位于树根，没有父结点。

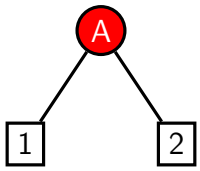
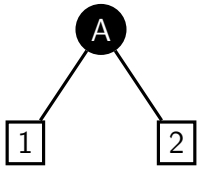


图 6: 局面 1

这种局面，直接让新结点变色为黑色，规则 2（根结点是黑色的）满足。同时黑色的根结点使每条路径上的黑色结点数目都增加了 1，因此并没有打破规则 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）。



局面 2

新结点（B）的父结点是黑色的。新插入的红色结点 B 并没有打破规则，无需调整。

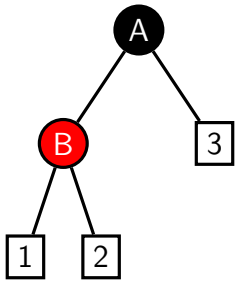


图 7: 局面 2

局面 3

新结点（D）的父结点和叔叔结点都是红色。

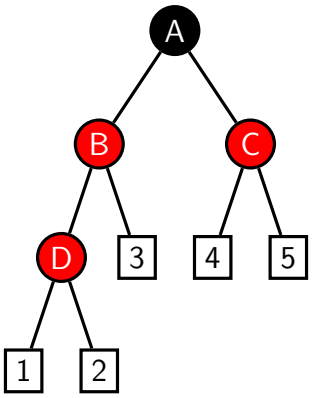
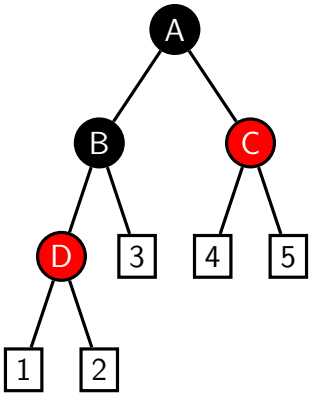
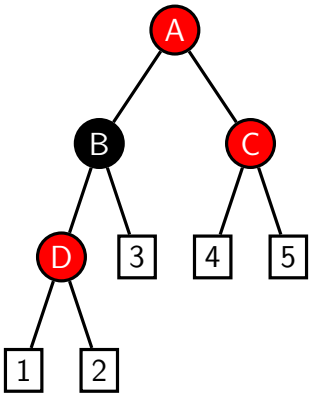


图 8: 局面 3

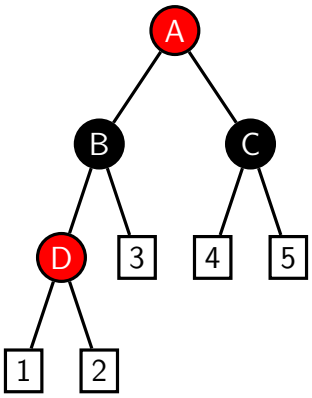
这种局面，两个红色结点 B 和 D 连续，违反了规则 4（红色结点的两个子结点都是黑色的），因此需要先让结点 B 变为黑色。



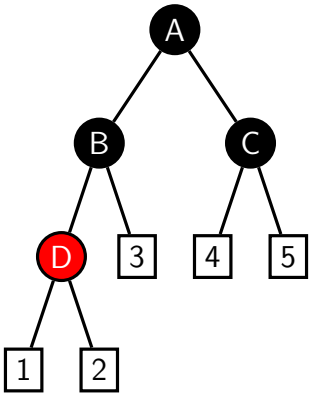
但是这样一来，结点 B 所在路径凭空多出了一个黑色结点，打破了规则 5（从任一结点到其每个叶子的所有路径都包含相同数目的黑色结点），因此再让结点 A 变为红色。



这时结点 A 和 C 又成为了连续的红色结点，再将结点 C 变为黑色。



如果红色结点 A 是根结点，那么违反了规则 2（根结点是黑色），参考局面 1 的方法，将其变为黑色。



局面 4

新结点（D）的父结点是红色，叔叔结点是黑色或者没有叔叔，且新结点是父结点的右孩子，父结点是祖父结点的左孩子。

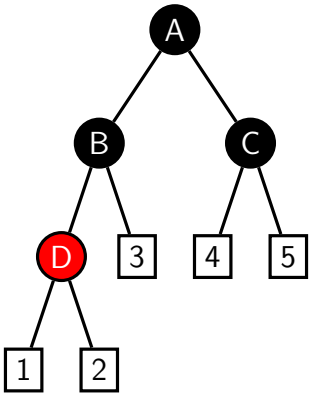


图 9: 局面 4