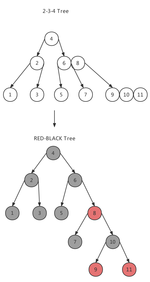
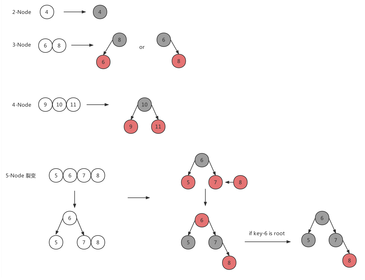


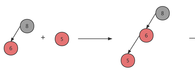


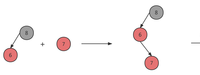
RED-BLACK Tree put

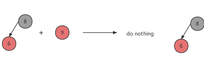
原则：先 put，再调整

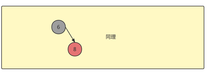
2-Node

3-Node

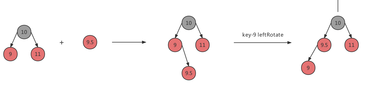


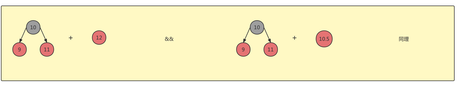






4-Node



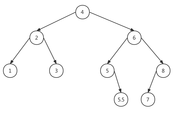


RED-BLACK Tree remove

原则：先 remove，再调整

序

二叉树删除：寻找能够和后续节点顶替（即覆盖指定删除的节点的值，真正删除的是能够或后续节点）



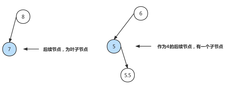
删除有三种情况：

1. 指定删除的节点为叶子节点，直接删除

2. 指定删除的节点只有一个子节点，用子节点代替，并删除子节点

3. 指定删除的节点有两个子节点，寻找能够或后续节点代替，并删除能够或后续节点

以后续节点为例，后续节点有且只有三种形式，如下所示：



故删除节点的问题，红黑树与二叉树一致，最终只会或为删除一个叶子节点或者删除一个只有一个子节点的节点的问题。

对以对应 2-3-4 树来讲，真正删除的永远是最底层的节点

1. 找到指定删除的节点 和 真正要被删除的节点，用真正要删除的节点的值重盖指定删除的节点

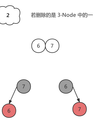
2. 删除并调整

首先必须明白哪些情况再删除后进行调整

1 若删除的是 4-Node 中的一个

结合上述，这种情况下，被删除的只能是 5 或 7，那么这种情况删除的是红色叶子节点，不影响黑高，直接删除即可

2 若删除的是 3-Node 中的一个



假设我们选取后继节点作为真正删除的节点，这种情况下，被删除的只能是 6

无论情况：6 是红色的叶子节点，直接删除即可，同上是 4 -Node 情况

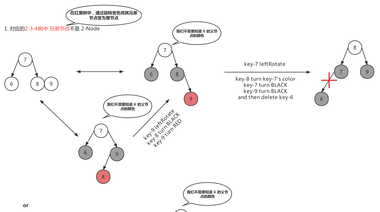
右边情况：6 是带有子节点的黑色节点，用叶子节点覆盖它，并将其让染成黑色

3 若删除的是 2 - Node

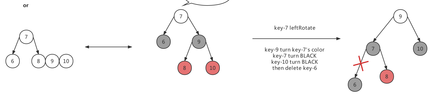
1 排除以上并不复杂两种情况，删除后 真正需要调整的只有 删除一个 2-Node 的情况

被删除的节点必然是黑色叶子节点。若删除的自然是 6，且该节点是父节点的左节点：分以下几种情况

1. 对应的2-3-4树中 兄弟节点不是 2-Node



or



2. 对应的2-3-4树中 兄弟节点 是 2-Node

这种情况，删除 6 节点，会导致黑高条件破打破，我们采取的措施是：将它的兄弟节点变为红色，向上回溯：

1）如果父节点为黑节点或根节点，递出回溯

2）如果父节点不为黑节点，将父节点染黑，将父节点的兄弟节点为黑节点，继续递归。

3）当前节点就是为父节点

以下为该情况的演示

删除 2节点