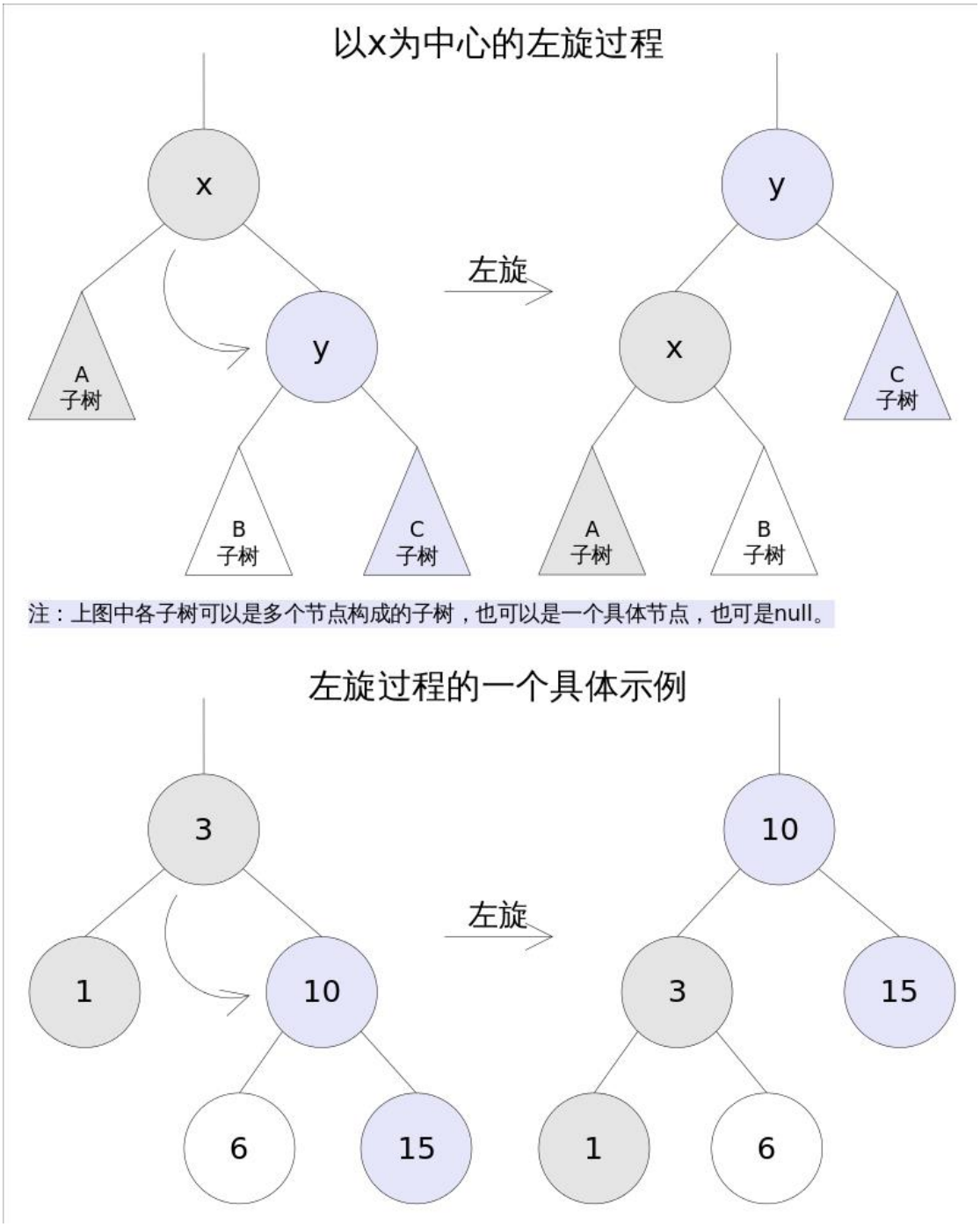
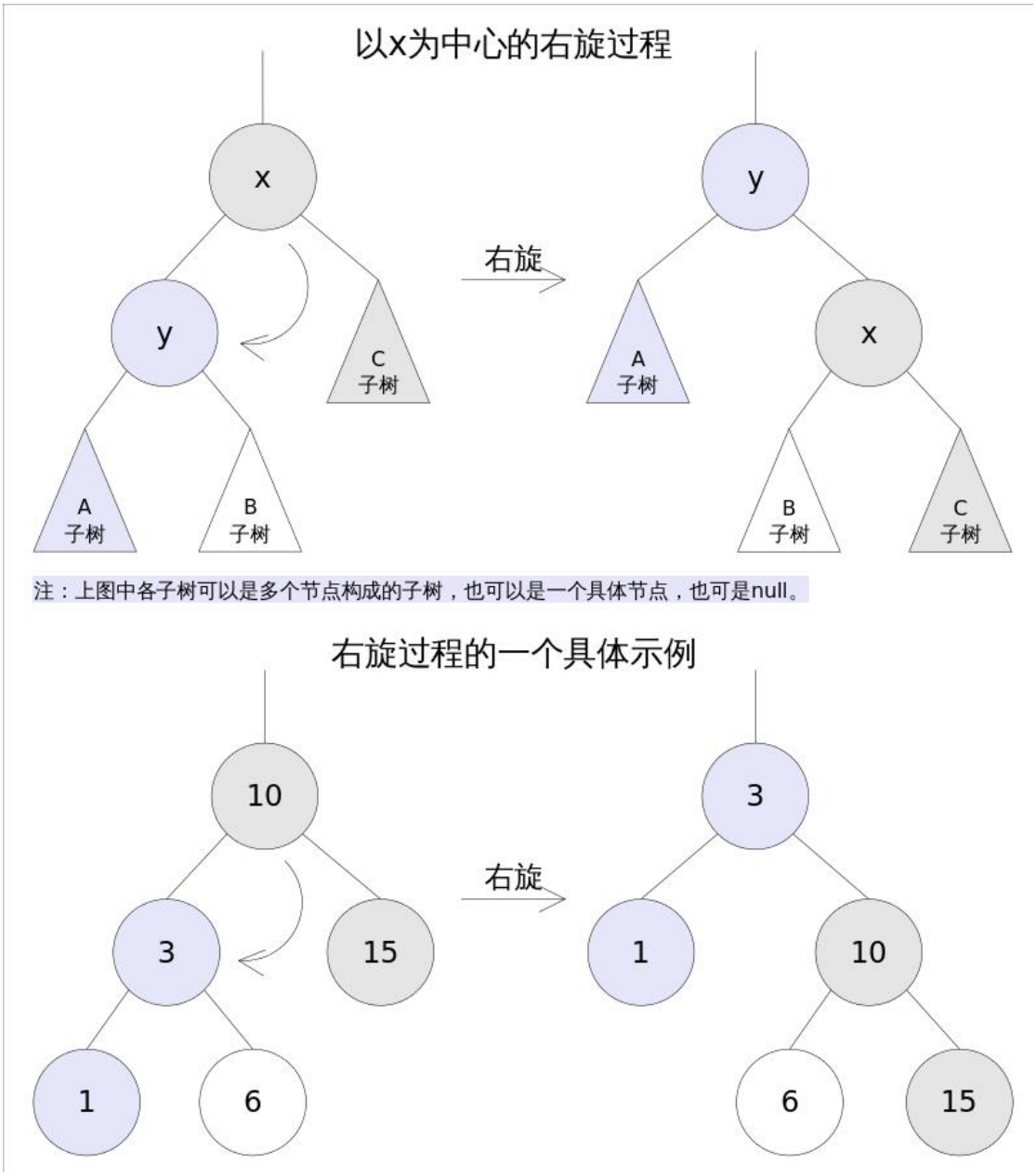
红黑树

本质：二叉查找树，即每个节点左子树的所有节点一定小于本节点，本节点也一定小于右子树的所有节点。

二叉树的旋转
左旋：



右旋：



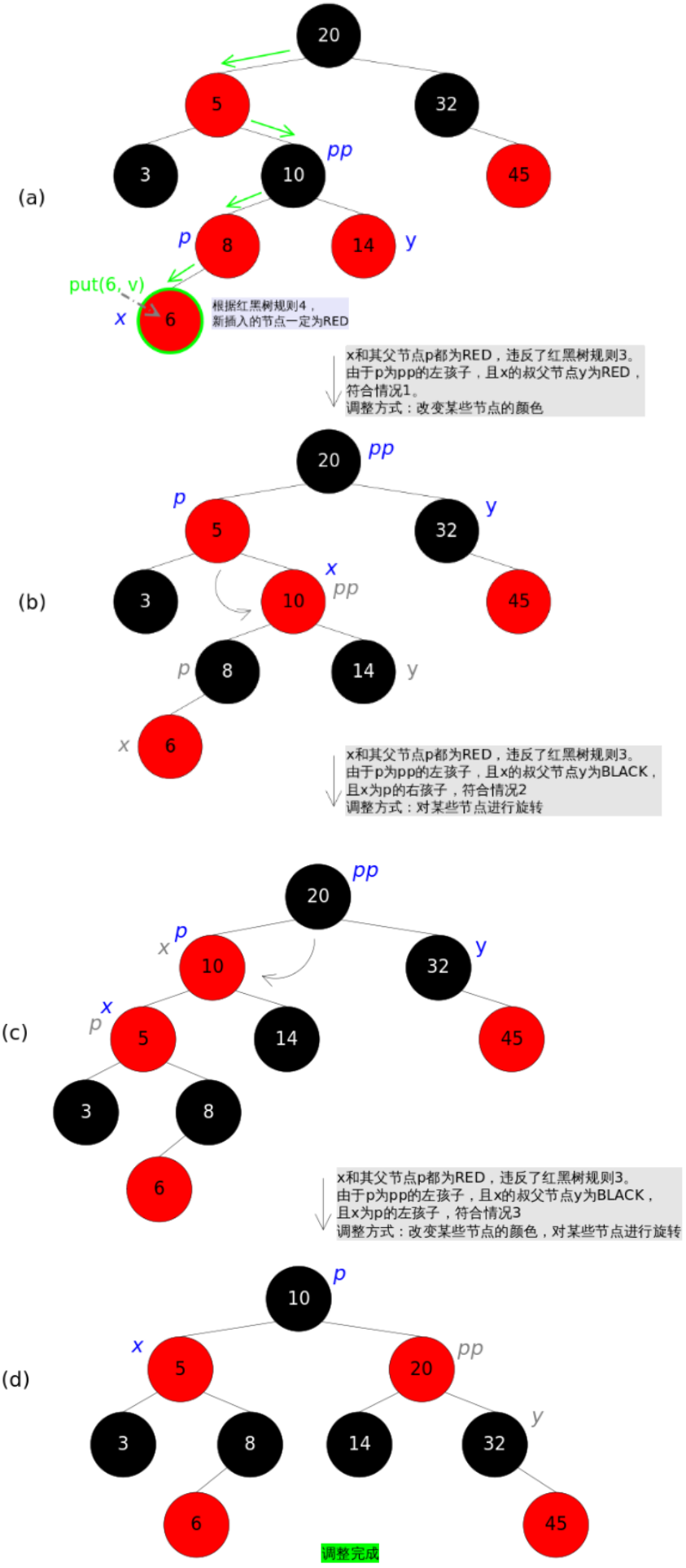
红黑树的性质：

1. 每个节点要么是红色，要么是黑色。
2. 根节点必须是黑色
3. 红色节点不能连续（也即是，红色节点的孩子和父亲都不能是红色）。
4. 对于每个节点，从该点至 null（树尾端）的任何路径，都含有相同个数的黑色节点。

添加节点：

1. 遵照二叉查找树的原则插入
2. 新插入的节点必须为红色（不违背性质 4）
3. 插入后，如果违背了性质 3，则通过旋转、着色来使其重新成为一颗标准的红黑树。

实例：

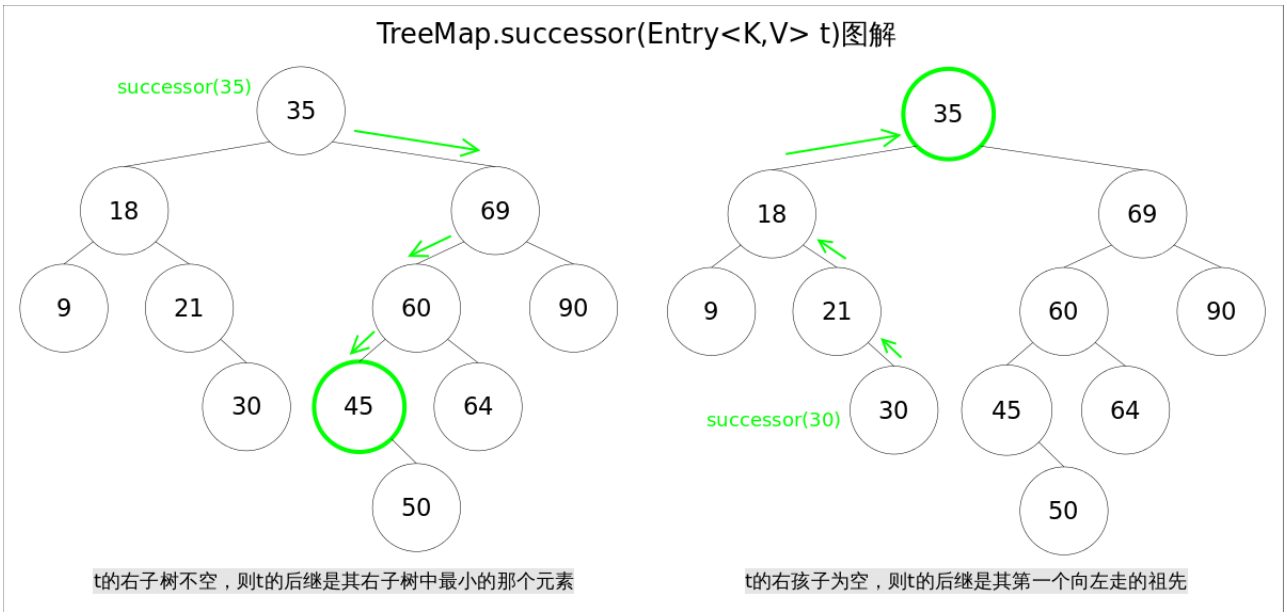

(a)
20
5
32
3
10
pp
45
p
8
14
y
put(6, v)
x
6
根据红黑树规则4，
新插入的节点一定为RED
x和其父节点p都为RED，违反了红黑树规则3。
由于p为pp的左孩子，且x的叔父节点y为RED，
符合情况1。
调整方式：改变某些节点的颜色
(b)
20
pp
p
5
y
32
x
3
10
pp
45
p
8
14
y
x
6
x和其父节点p都为RED，违反了红黑树规则3。
由于p为pp的左孩子，且x的叔父节点y为BLACK，
且x为p的右孩子，符合情况2
调整方式：对某些节点进行旋转
(c)
pp
20
x
p
10
y
32
p
x
5
14
45
3
8
6
x和其父节点p都为RED，违反了红黑树规则3。
由于p为pp的左孩子，且x的叔父节点y为BLACK，
且x为p的左孩子，符合情况3
调整方式：改变某些节点的颜色，对某些节点进行旋转
(d)
p
10
x
5
pp
20
y
3
8
14
32
6
45
调整完成

删除节点：

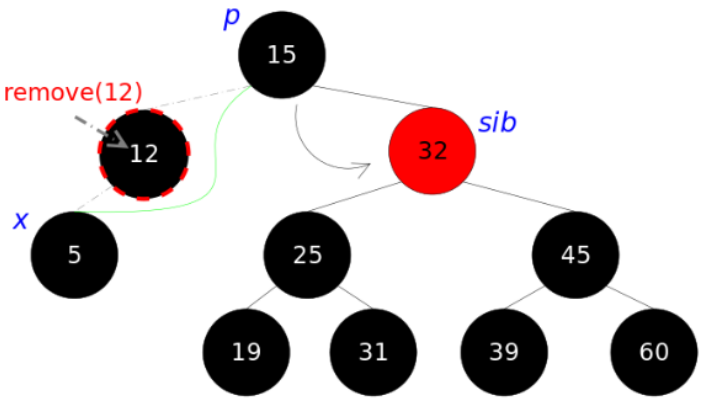
二叉查找树的删除：

1. 节点 t 的右子树不空，则 t 的后继是其右子树中最小的那个元素。
2. 节点 t 的右孩子为空，则 t 的后继是其第一个向左走的祖先。

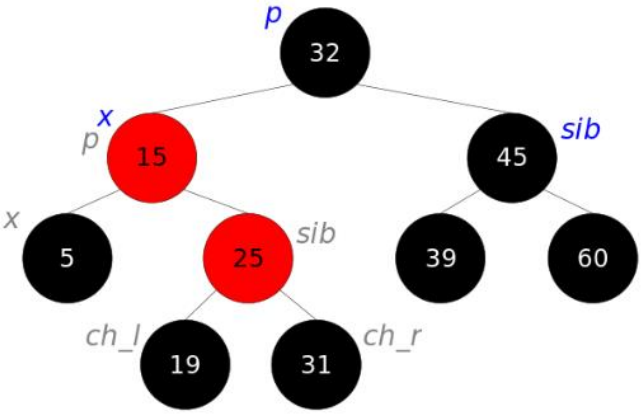
实例：



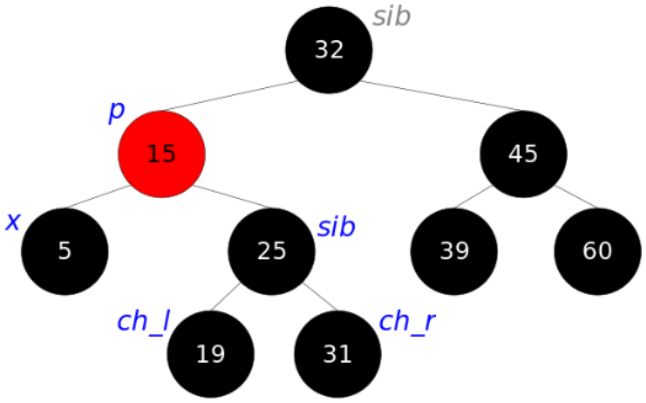
红黑树删除实例 1：



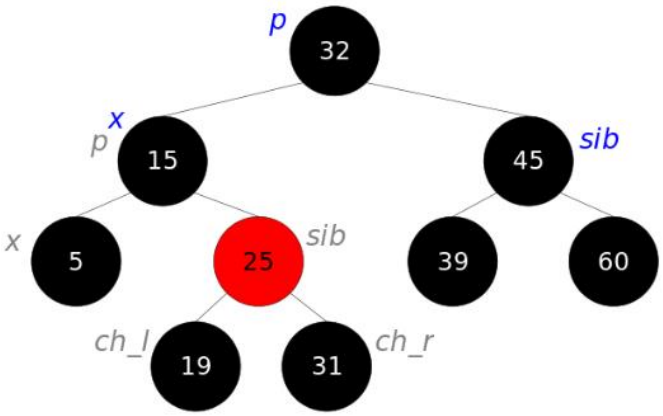
x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为p的左孩子，且x的兄弟节点sib为RED，
符合情况1。
调整方式：改变某些节点的颜色，对某些节点进行旋转



x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为RED，不再符合循环条件。
调整方式：将x改为BLACK

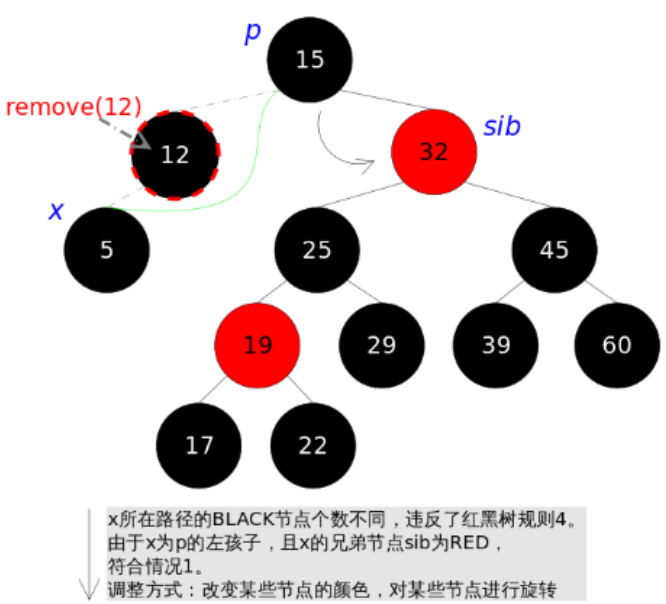


x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为p的左孩子，且x的兄弟节点sib为BLACK，
且sib的两个孩子ch_l和ch_r都为BLACK，符合情况2。
调整方式：改变某些节点的颜色

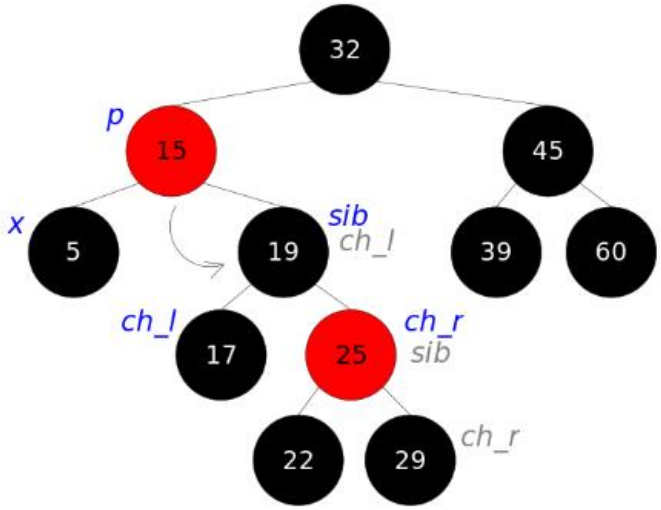


调整完成

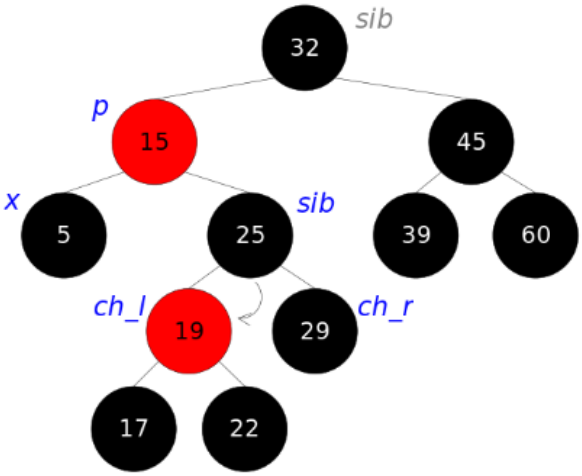
红黑树删除实例 2：



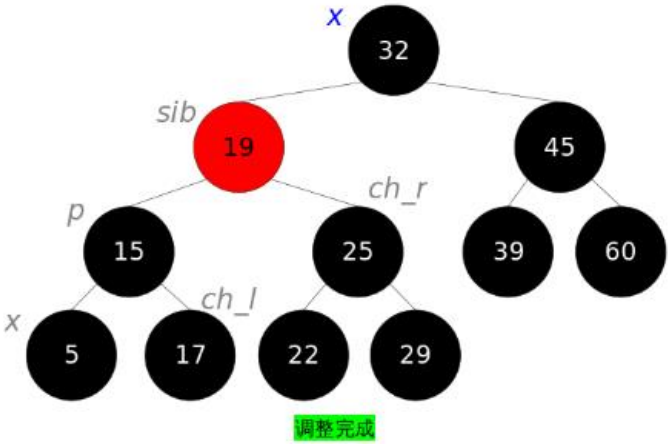
x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为p的左孩子，且x的兄弟节点sib为RED，
符合情况1。
调整方式：改变某些节点的颜色，对某些节点进行旋转



x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为p的左孩子，且x的兄弟节点sib为BLACK，
且sib的右孩子ch_r为RED，符合情况4。
调整方式：改变某些节点的颜色，对某些节点进行旋转

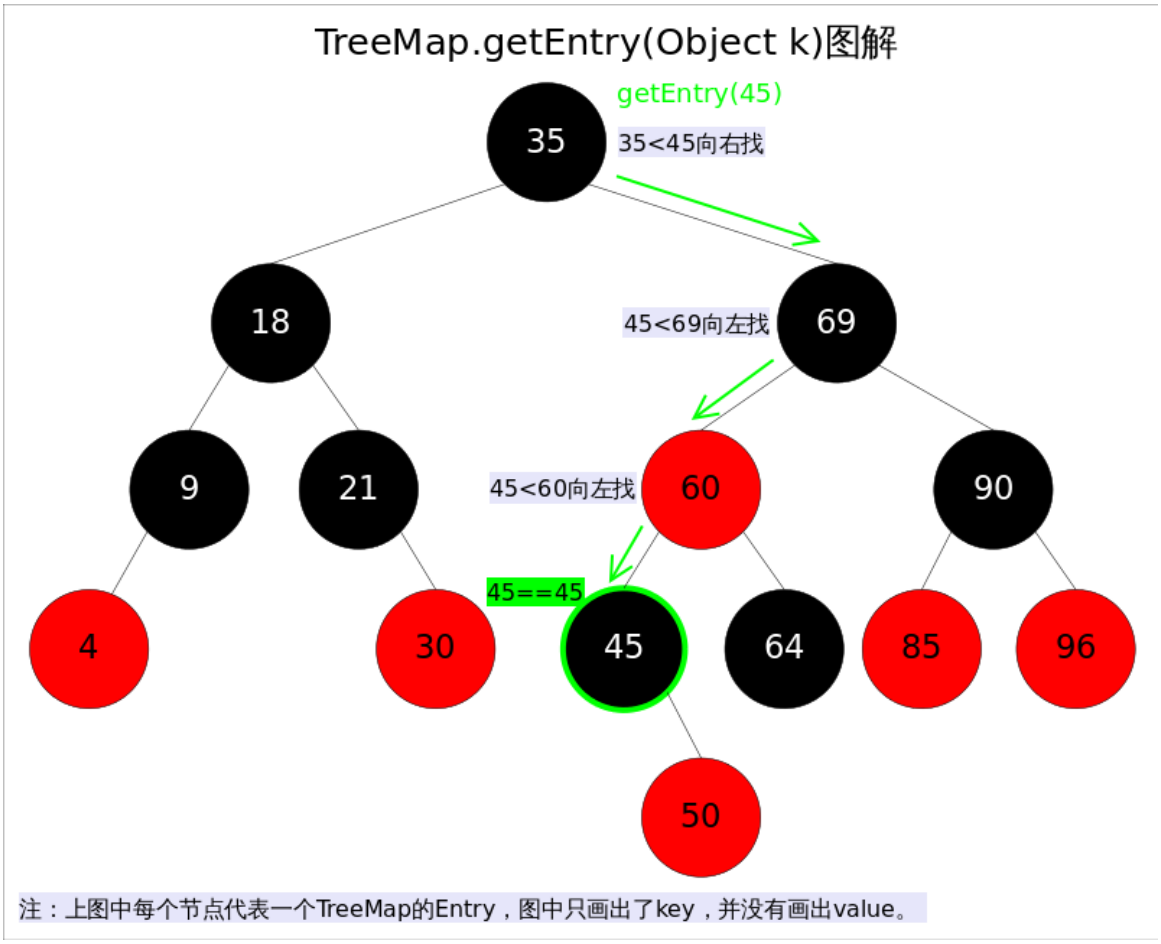


x所在路径的BLACK节点个数不同，违反了红黑树规则4。
由于x为p的左孩子，且x的兄弟节点sib为BLACK，
且sib的左孩子ch_l为RED，右孩子ch_r为BLACK，
符合情况3。
调整方式：改变某些节点的颜色，对某些节点进行旋转



调整完成

红黑树读取：


TreeMap.getEntry(Object k)图解
getEntry(45)
35
35<45向右找
18
45<69向左找
69
9
21
45<60向左找
60
90
45==45
4
30
45
64
85
96
50
注：上图中每个节点代表一个TreeMap的Entry，图中只画出了key，并没有画出value。