

```

/**
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。[这里指到叶子节点的路径]
 */

private static final boolean RED    = false;
private static final boolean BLACK = true;

static final class Entry<K,V> implements Map.Entry<K,V> {
    K key;
    V value;
    Entry<K,V> left;
    Entry<K,V> right;
    Entry<K,V> parent;
    boolean color = BLACK;

}

// 叶子节点颜色为黑色
private static <K,V> boolean colorOf(Entry<K,V> p) {
    return (p == null ? BLACK : p.color);
}

private static <K,V> void setColor(Entry<K,V> p, boolean c) {
    if (p != null)
        p.color = c;
}

private static <K,V> Entry<K,V> parentOf(Entry<K,V> p) {
    return (p == null ? null: p.parent);
}

private static <K,V> Entry<K,V> leftOf(Entry<K,V> p) {
    return (p == null) ? null: p.left;
}

private static <K,V> Entry<K,V> rightOf(Entry<K,V> p) {
    return (p == null) ? null: p.right;
}

/*

```

```

* 左旋示意图: 对节点x进行左旋
*      p          p
*      /          /
*      x          y
*      / \        / \
* 1x  y   ----->  x  ry
*      / \        / \
*  ly  ry        lx  ly
* 左旋做了三件事:
* 1. 将y的左子节点赋给x的右子节点, 并将x赋给y左子节点的父节点(y左子节点非空时)
* 2. 将x的父节点p(非空时)赋给y的父节点, 同时更新p的子节点为y(左或右)
* 3. 将y的左子节点设为x, 将x的父节点设为y
*/
private void rotateLeft(Entry p){
    if(p != null){
        /**
         * 左旋后, 当前节点的右节点变成其父节点; 当前节点失去了右孩子; 右孩子由其原来右节点
         * 的左孩子代替; 之后分别处理根节点的情况以及连接旋转后的节点
         */
        Entry r = p.right;
        p.right = r.left;

        r.parent = p.parent;
        if(r.left != null){
            r.left.parent = p;
        }

        if(p.parent == null){
            this.root = r;
        } else if(p.parent.right == p){
            p.parent.right = r;
        } else {
            p.parent.left = r;
        }

        r.left = p;
        p.parent = r;
    }
}

/*
* 左旋示意图: 对节点y进行右旋
*      p          p
*      /          /
*      y          x
*      / \        / \
*  x  ry   ----->  lx  y
*      / \        / \
*  lx  rx        rx  ry
* 右旋做了三件事:

```

```

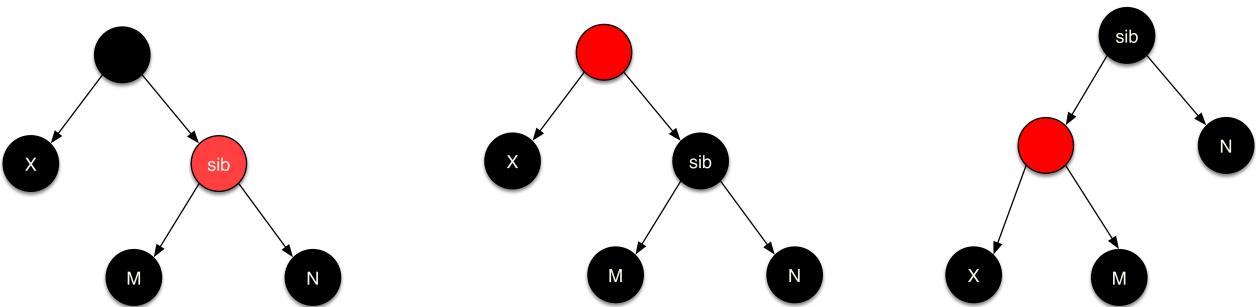
    * 1. 将x的右子节点赋给y的左子节点，并将y赋给x右子节点的父节点(x右子节点非空时)
    * 2. 将y的父节点p(非空时)赋给x的父节点，同时更新p的子节点为x(左或右)
    * 3. 将x的右子节点设为y，将y的父节点设为x
    */
private void rotateRight(Entry p) {
    if (p != null) {
        Entry l = p.left;
        p.left = l.right;
        if (l.right != null) l.right.parent = p;
        l.parent = p.parent;
        if (p.parent == null)
            root = l;
        else if (p.parent.right == p)
            p.parent.right = l;
        else p.parent.left = l;
        l.right = p;
        p.parent = l;
    }
}

```

```

private void fixAfterDeletion(Entry x){
    /**
     * 调整时候需要x的颜色为黑色如果为红色在结束的时候设置为黑色不改变黑高
     * 由于删除时候如果节点是黑色的会使得一侧的黑高减少1；需要进行颜色变换
     * 以及旋转向缺失的一侧进行补偿
    */
    while(x!=root && colorOf(x) == BLACK){
        /*
         * 如果节点是左节点的情况
        */
        if(x == leftOf(parentOf(x))){
            /**
             * 找到其兄弟节点
            */
            Entry sib = rightOf(parentOf(x));
            /**
             * 如果兄弟节点是红点的情况；因为兄弟节点是红色；其父节点一定为黑色
             * 同理兄弟节点的两个子节点也必然是黑色；设置兄弟节点颜色为黑色，设置
             * 父节点颜色为红色同时以父节点左旋后不改变右侧树原来的黑高同时将待调整节点
             * 的兄弟节点变成了黑色；操作并未改变两侧的黑高；由于旋转需要重新确定x的兄弟节点
            */
            if(colorOf(sib) == RED){
                setColor(sib, BLACK);
                setColor(parentOf(x), RED);
                rotateLeft(parentOf(x));
                sib = rightOf(parentOf(x));
            }
        }
    }
}

```



/\*\*

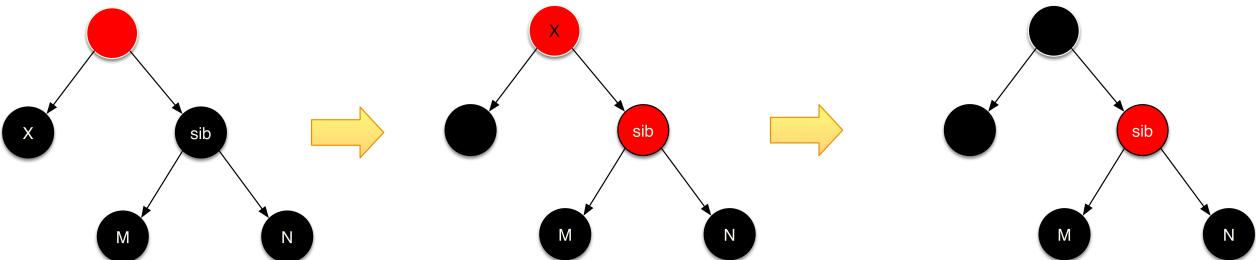
如果兄弟节点是红的的话经过上面的变换兄弟节点一定变为了黑色;所以处理情况2:  
兄弟节点是黑色,并且兄弟节点的两个孩子都是黑色;注意如果节点是null颜色是属于high的;由于兄弟节点这边多出了一个黑高;将兄弟节点设置为红色减少一个黑高;并且将x指向x的父节点;这里如果父节点是红色的循环将会结束;在最后会将x设置为黑色;导致右侧由于兄弟节点变为红色减少的黑高得到补偿;同时左侧缺失的一个黑高也得到了补充完成调整

整

如果父节点是黑色的循环将继续;为x缺失的一个黑高继续做调整;

\*/

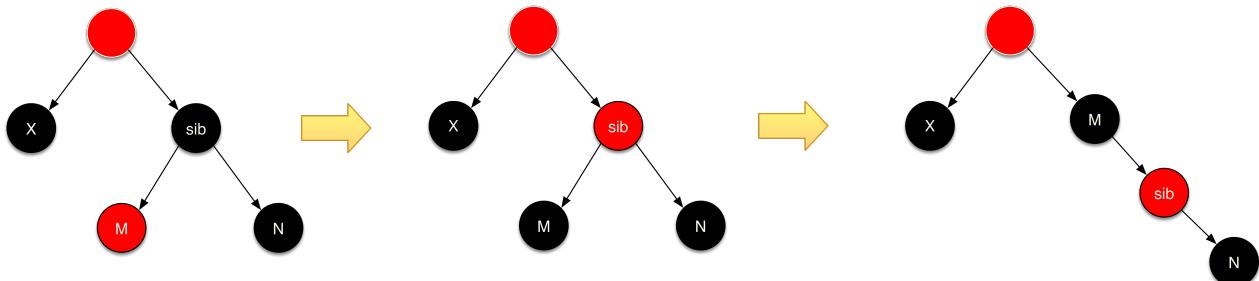
```
if(colorOf(rightOf(sib)) == BLACK && colorOf(leftOf(sib)) ==  
BLACK) {  
    setColor(sib, RED);  
    x = parentOf(x);  
}
```



目的

选择导致

```
else {
    /**
     情况3： 如果兄弟节点的右孩子是黑色的；左孩子一定是红色的；将左孩子设置为黑色
     sib颜色设置为红色；以sib做右旋；更新sib节点后边为sib的右孩子是红色；这样的
     是为下面左旋补偿左侧删除的一个黑高时可以将sib的右孩子至为黑色补充右侧因为
     的黑高缺失
    */
    if(colorOf(rightOf(sib)) == BLACK){
        setColor(leftOf(sib), BLACK);
        setColor(sib, RED);
        rotateRight(sib);
        sib = rightOf(parentOf(x));
    }
}
```



通过左旋去

了保持

色；

色；

设置

```
/**
 情况4： 到达这一步可以确定兄弟节点是黑色的；兄弟节点的右节点是红色的；下一步
 补充由于删除导致的左子树黑高减一的情况；左旋会使得sib节点代替parent节点；为
 parent节点颜色设置sib为parent节点的颜色；因为parent可能为黑色也可能为红
 如果为黑色左旋后右子树黑高 少1需要用sib.right=black弥补；如果parent为红
 sib设置为红色；同样导致右侧少1；用右侧至为 黑色弥补；由于要补充左侧一个黑高
 x的父节点为黑色；以x的父节点做左旋后红黑树恢复；设置x=root结束循环
 */
setColor(sib, colorOf(parentOf(x)));
setColor(parentOf(x), BLACK);
setColor(rightOf(sib), BLACK);
rotateLeft(parentOf(x));
x = root;
}

} else{
```

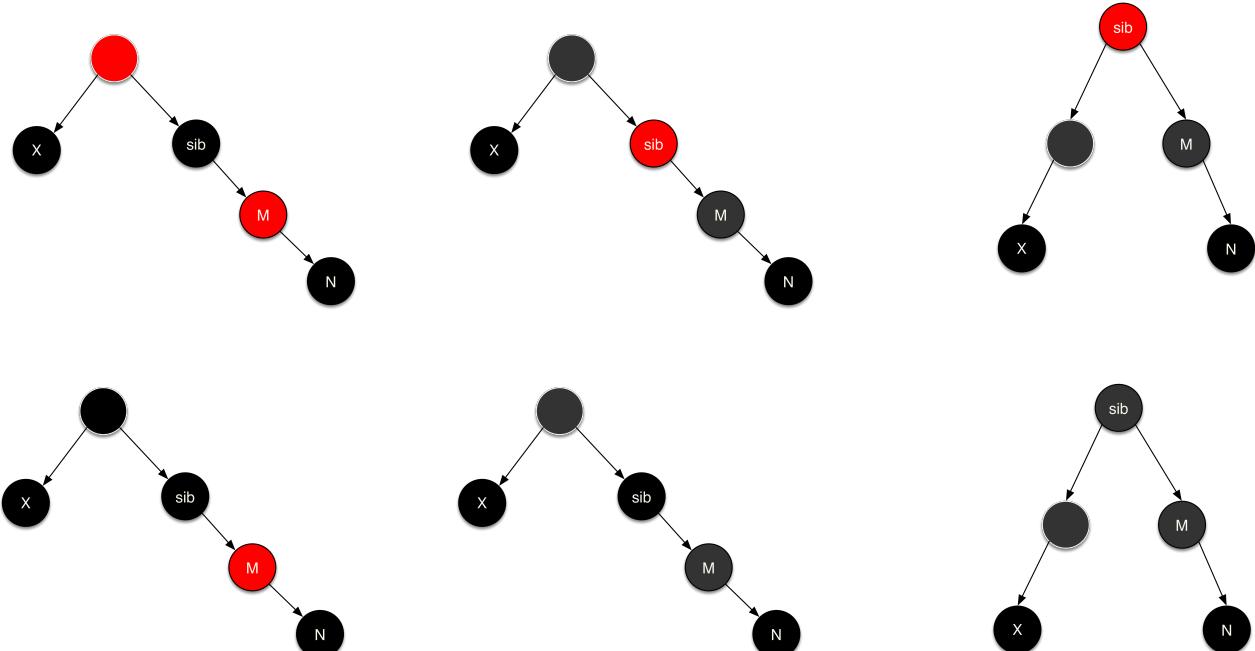
```

    // 因为对称性直接left right 调换即可
}

}

setColor(x, BLACK);
}

```



```

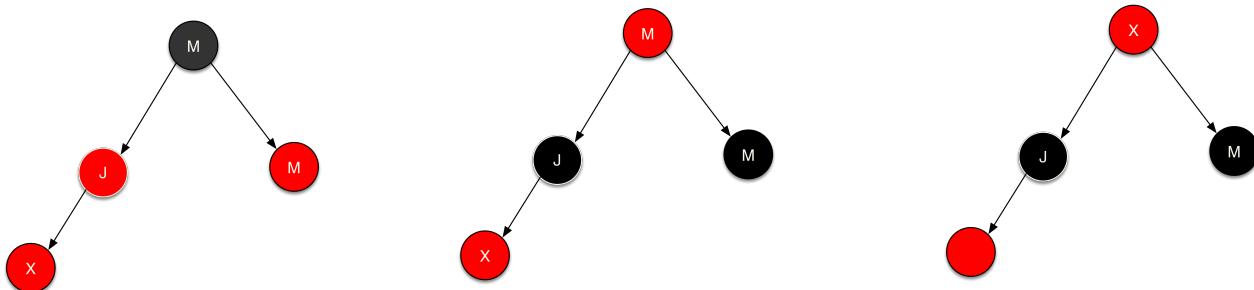
/**
插入后调整
*/
private void fixAfterInsertion(Entry x){
    /**
     * 插入的节点统一至为红色;因为这样违反红黑树的性质最少
     * 易于调整
     */
    x.color = RED;
    /**
     * 当x不为空; 不是root节点(说明有父节点); x父节点的颜色为红色
     * 违反了性质4; 红色节点的子节点必为黑色的性质
     */
    while(x!=null && x != root && x.parent.color == RED){
        /**
         * 如果x父节点是祖父节点的左节点
         */
        if(parentOf(x) == leftOf(parentOf(parentOf(x)))){
            /**
             * 找到x的叔叔节点
             */

```

```

    */
Entry uncle = rightOf(parentOf(parentOf(x)));
/**
如果叔叔节点是红色；说明其祖父节点必然为黑色不然就不满足
红黑树的性质；此时将x父节点和叔叔节点至为黑色解决了违反
的性质4条件；但是引入了新的黑高；所以需要将祖父节点至为
红色解决此问题；将x指向其祖父节点准备下一次操作
*/
if(colorOf(uncle) == RED){
    setColor(parent(x), BLACK);
    setColor(uncle, BLACK);
    setColor(parentOf(parentOf(x)), RED);
    x = parentOf(parentOf(x));
}

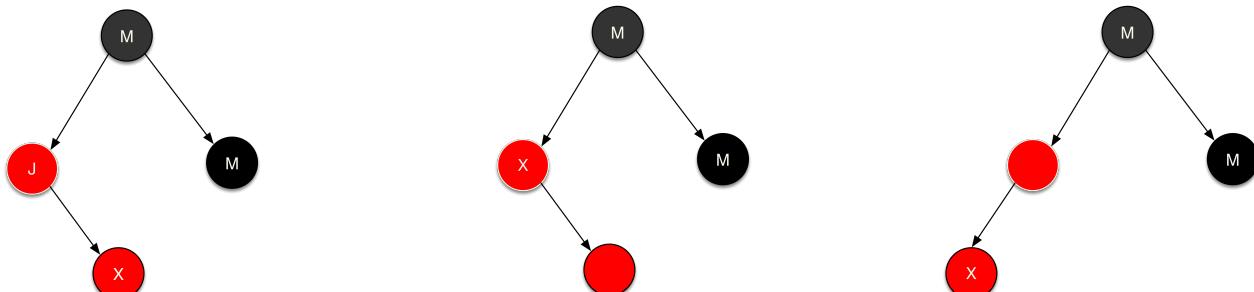
```



```

else {
    /**
如果叔叔节点是黑色；x是右节点；将x指向父节点并以x
做左旋使得两个红色节点都变到左侧便于下面通过设置
颜色右旋修复
*/
if (x == rightOf(parentOf(x))) {
    x = parentOf(x);
    rotateLeft(x);
}

```



```

    /**
     * 由于两个红色节点都在左侧;x的祖父节点是黑色;只需要先调整颜色
     * 设置父节点为黑色;祖父节点为红色;解决了性质4的问题;
     * 通过以祖父节点右旋解决黑高减少的问题(因为祖父节点变为了红色)
     * 右旋后x的父节点变成了原来祖父节点的问题;
     */
    setColor(parentOf(x), BLACK);
    setColor(parentOf(parentOf(x)), RED);
    rotateRight(parentOf(parentOf(x)));
}
} else {
    // 因为对称性直接left right 调换即可
}
}

/**
确保根节点颜色
*/
root.color = BLACK;
}
}

```

