

## 一步一图一代码，一定要让你真正彻底明白红黑树

作者：July 二零一一年一月九日

-----  
本文参考：

I、 The Art of Computer Programming Volume I

II、 Introduction to Algorithms, Second Edition

III、 The Annotated STL Sources

IV、 Wikipedia

V、 Algorithms In C Third Edition

VI、 本人写的关于红黑树的前三篇文章：

第一篇：教你透彻了解红黑树：

[http://blog.csdn.net/v\\_JULY\\_v/archive/2010/12/29/6105630.aspx](http://blog.csdn.net/v_JULY_v/archive/2010/12/29/6105630.aspx)

第二篇：红黑树算法的层层剖析与逐步实现

[http://blog.csdn.net/v\\_JULY\\_v/archive/2010/12/31/6109153.aspx](http://blog.csdn.net/v_JULY_v/archive/2010/12/31/6109153.aspx)

第三篇：教你彻底实现红黑树：红黑树的c源码实现与剖析

[http://blog.csdn.net/v\\_JULY\\_v/archive/2011/01/03/6114226.aspx](http://blog.csdn.net/v_JULY_v/archive/2011/01/03/6114226.aspx)

-----  
前言：

- 1、有读者反应，说看了我的前几篇文章，对红黑树的了解还是不够透彻。
- 2、我个人觉得，如果我一步一步，用图+代码来阐述各种插入、删除情况，可能会更直观易懂。
- 3、既然写了红黑树，那么我就一定要把它真正写好，让读者真正彻底明白红黑树。

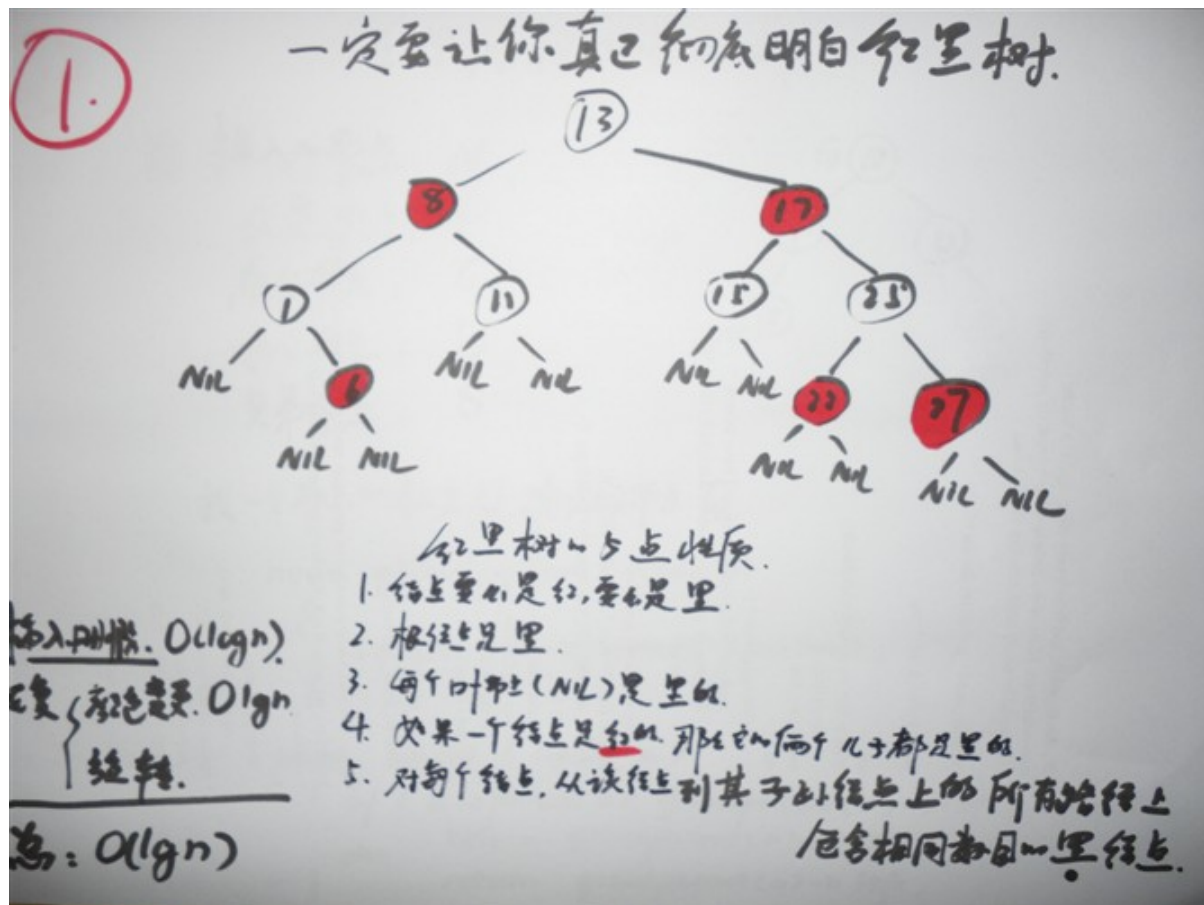
本文相对我前面红黑树相关的3篇文章，主要有以下几点改进：

- 1.图、文字叙述、代码编写，彼此对应，明朗而清晰。
- 2.宏观总结，红黑树的性质与插入、删除情况的认识。
- 3.代码来的更直接，结合图，给你最直观的感受，彻底明白红黑树。

ok, 首先, 以下几点, 你现在应该是要清楚明白了的:

### I、红黑树的五个性质:

- 1) 每个结点要么是红的, 要么是黑的。
- 2) 根结点是黑的。
- 3) 每个叶结点, 即空结点 (NIL) 是黑的。
- 4) 如果一个结点是红的, 那么它的俩个儿子都是黑的。
- 5) 对每个结点, 从该结点到其子孙结点的所有路径上包含相同数目的黑结点。



### II、红黑树插入的几种情况:

情况1: z的叔叔y是红色的。

情况2: z的叔叔y是黑色的, 且z是右孩子

情况3: z的叔叔y是黑色的, 且z是左孩子

### III、红黑树删除的几种情况。

情况1: x的兄弟w是红色的。

情况2: x的兄弟w是黑色的, 且w的俩个孩子都是黑色的。

情况3: x的兄弟w是黑色的, 且w的左孩子是红色, w的右孩子是黑色。

情况4: x的兄弟w是黑色的, 且w的右孩子是红色的。

除此之外, 还得明确一点:

IV、我们知道, 红黑树插入、或删除结点后,

可能会违背、或破坏红黑树的原有的性质,

所以为了使插入、或删除结点后的树依然维持为一棵新的红黑树,

那就要做俩方面的工作:

1、部分结点颜色, 重新着色

2、调整部分指针的指向, 即左旋、右旋。

V、并区别以下俩种操作:

1)红黑树插入、删除结点的操作, RB-INSERT(T, z), RB-DELETE(T, z)

2).红黑树已经插入、删除结点之后,

为了保持红黑树原有的红黑性质而做的恢复与保持红黑性质的操作。

如RB-INSERT-FIXUP(T, z), RB-DELETE-FIXUP(T, x)

以上这5点, 我已经在我前面的2篇文章, 都已阐述过不少次了, 希望, 你现在已经透彻明了。

-----

本文, 着重图解分析红黑树插入、删除结点后为了维持红黑性质而做修复工作的各种情况。

[下文各种插入、删除的情况, 与我的第二篇文章, [红黑树算法的实现与剖析相对应](#)]

ok, 开始。

### 一、在下面的分析中, 我们约定:

要插入的节点为, N

父亲节点, P

祖父节点, G

叔叔节点, U

兄弟节点, S

如下图所示, 找一个节点的祖父和叔叔节点:

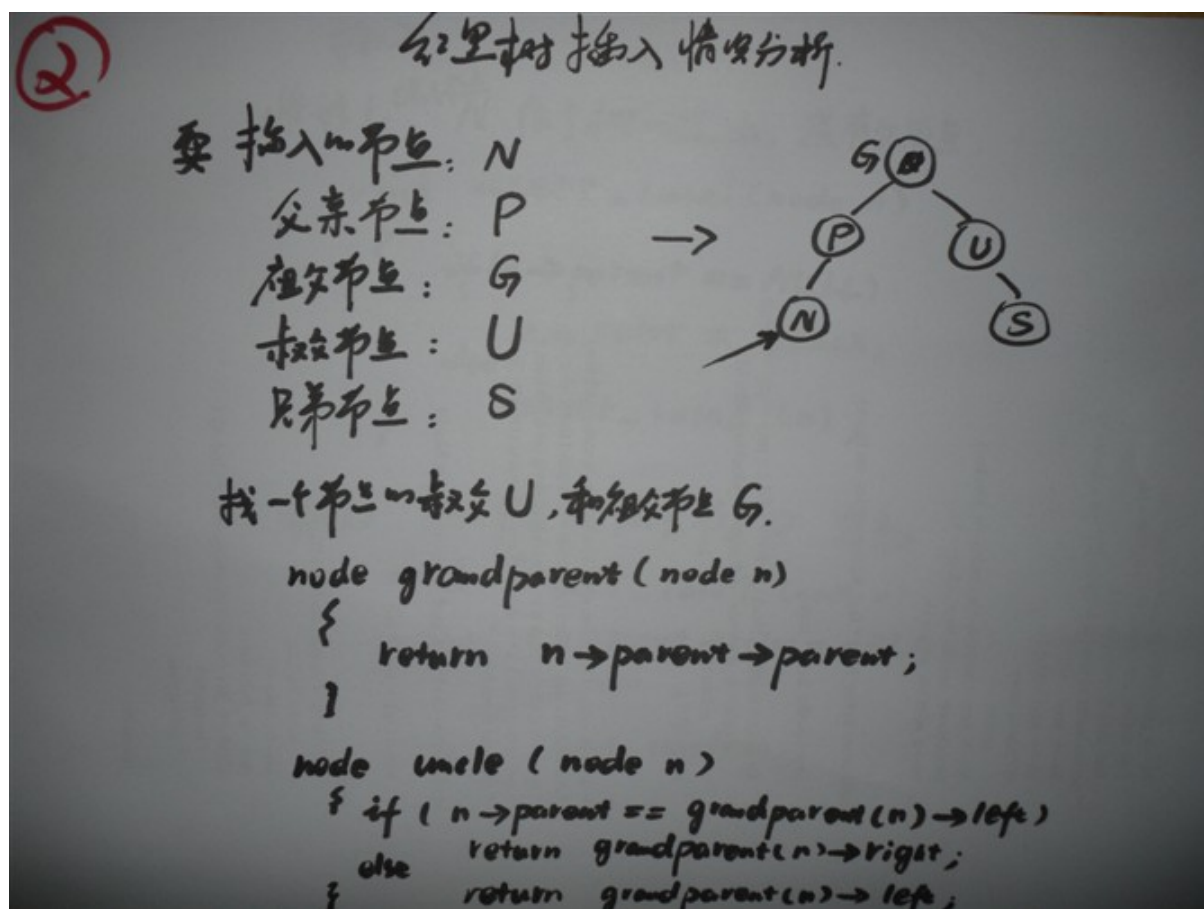
```
node grandparent(node n) //祖父
```

```

{
    return n->parent->parent;
}

node uncle(node n)    //叔叔
{
    if (n->parent == grandparent(n)->left)
        return grandparent(n)->right;
    else
        return grandparent(n)->left;
}

```



## 二、红黑树插入的几种情况

情形1: 新节点  $N$  位于树的根上, 没有父节点

```
void insert_case1(node n) {
```

```

if (n->parent == NULL)

    n->color = BLACK;

else

    insert_case2(n);

}

```

情形2: 新节点的父节点P是黑色

```

void insert_case2(node n) {

    if (n->parent->color == BLACK)

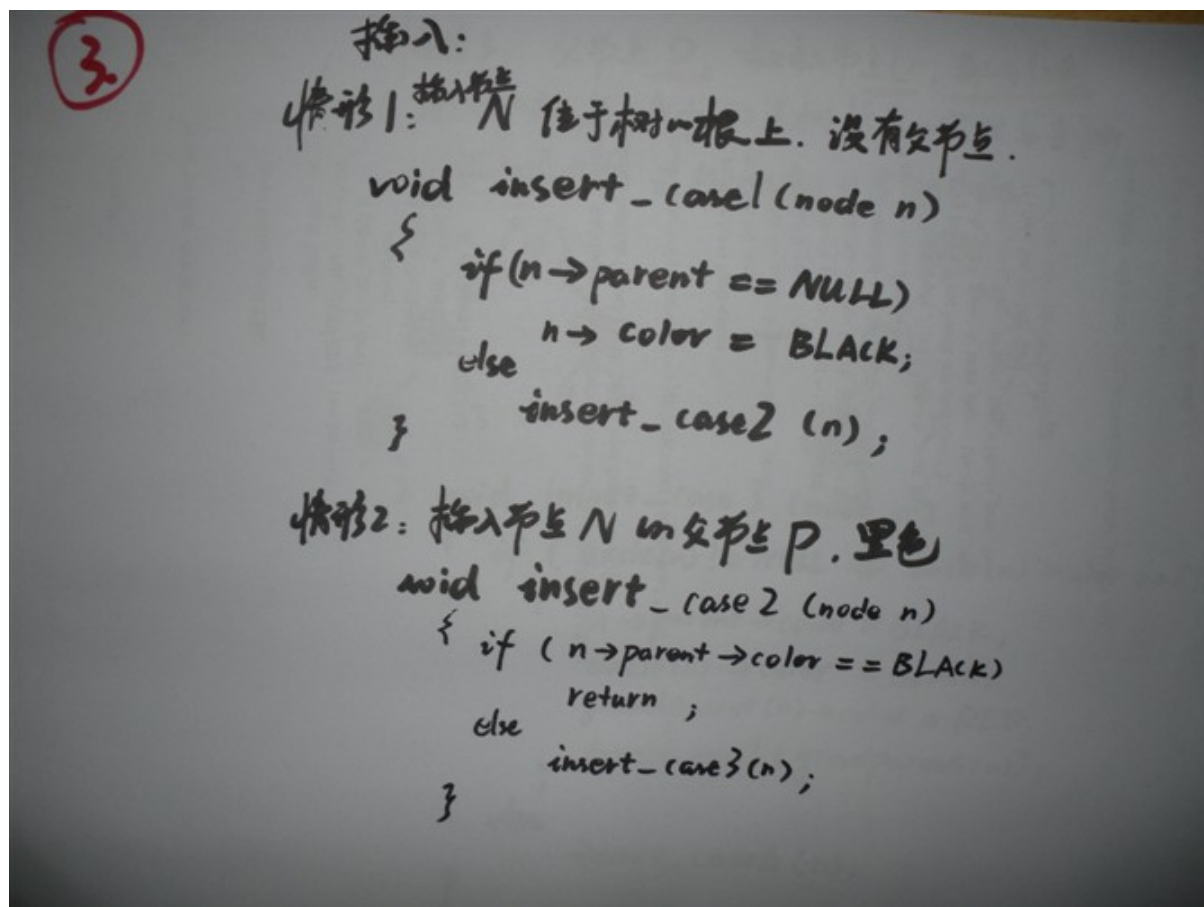
        return; /* 树仍旧有效 */

    else

        insert_case3(n);

}

```



情形3: 父节点P、叔叔节点U，都为红色，

[对应第二篇文章中，的情况1: z的叔叔是红色的。]

```

void insert_case3(node n) {

    if (uncle(n) != NULL && uncle(n)->color == RED) {

        n->parent->color = BLACK;

        uncle(n)->color = BLACK;

        grandparent(n)->color = RED;

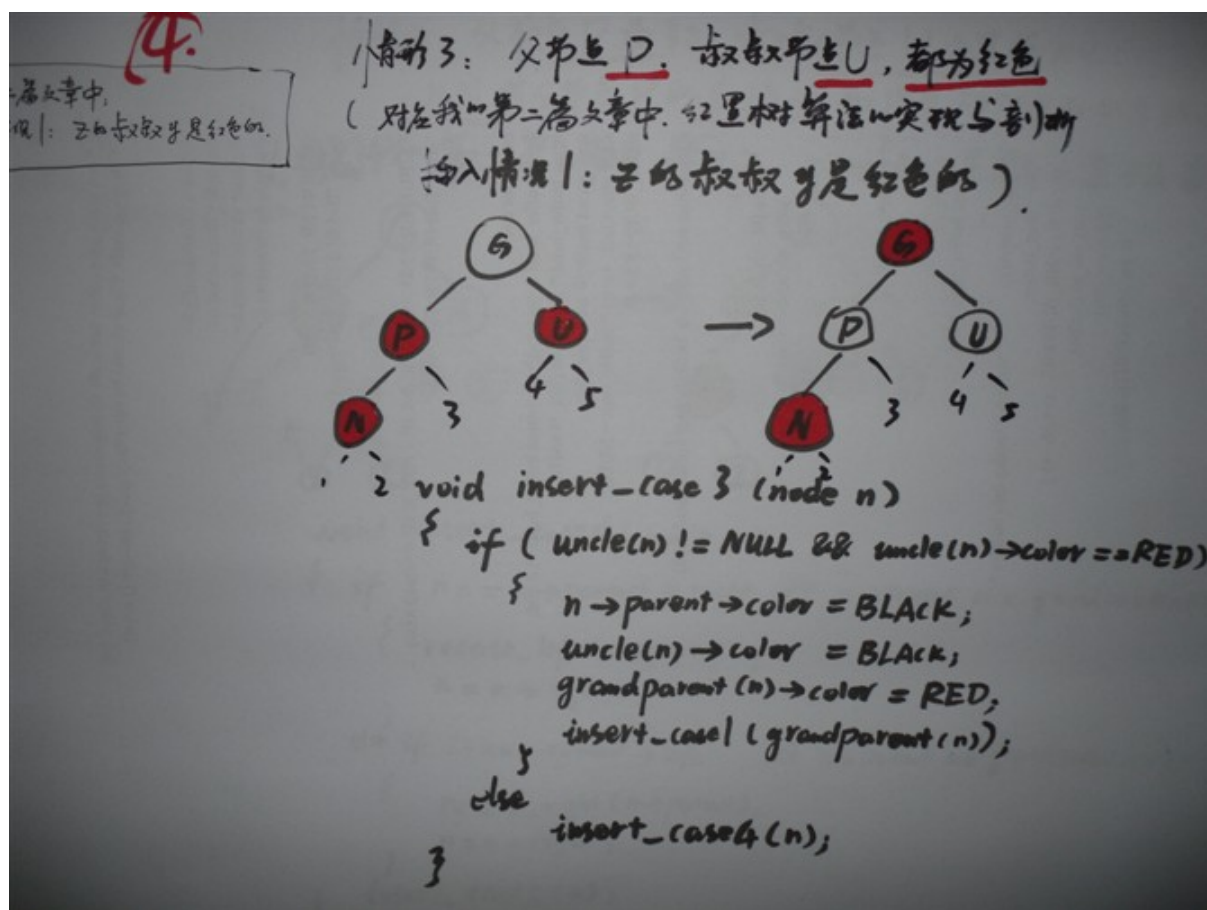
        insert_case1(grandparent(n)); //因为祖父节点可能是红色的，违反性质4，递归情形1.

    }

    else

        insert_case4(n); //否则，叔叔是黑色的，转到下述情形4处理。

```



此时新插入节点N做为P的左子节点或右子节点都属于上述情形3,上图仅显示N做为P左子的情形。

情形4: 父节点P是红色，叔叔节点U是黑色或NIL；

插入节点N是其父节点P的右孩子，而父节点P又是其父节点的左孩子。

[对应我第二篇文章中，的情况2: z的叔叔是黑色的，且z是右孩子]

```

void insert_case4(node n) {

```



```

if (n == n->parent->right && n->parent == grandparent(n)->left) {

    rotate_left(n->parent);

    n = n->left;

} else if (n == n->parent->left && n->parent == grandparent(n)->right) {

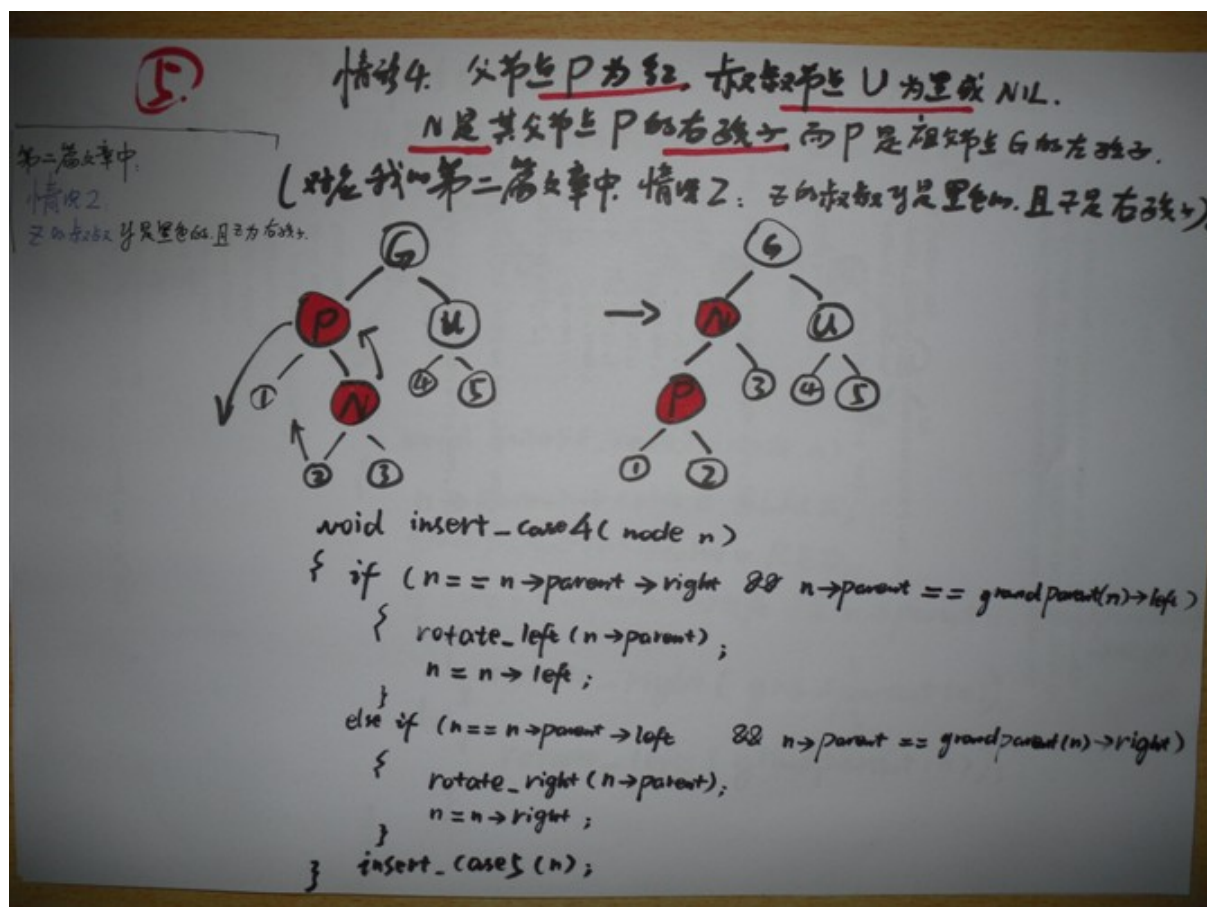
    rotate_right(n->parent);

    n = n->right;

}

insert_case5(n); //转到下述情形5处理。

```



情形5: 父节点P是红色, 而叔父节点U 是黑色或NIL,

要插入的节点N 是其父节点的左孩子, 而父节点P又是其父G的左孩子。

[对应我第二篇文章中, 情况3: z的叔叔是黑色的, 且z是左孩子。]

```

void insert_case5(node n) {

    n->parent->color = BLACK;

    grandparent(n)->color = RED;

```

```

if (n == n->parent->left && n->parent == grandparent(n)->left) {

    rotate_right(grandparent(n));

} else {

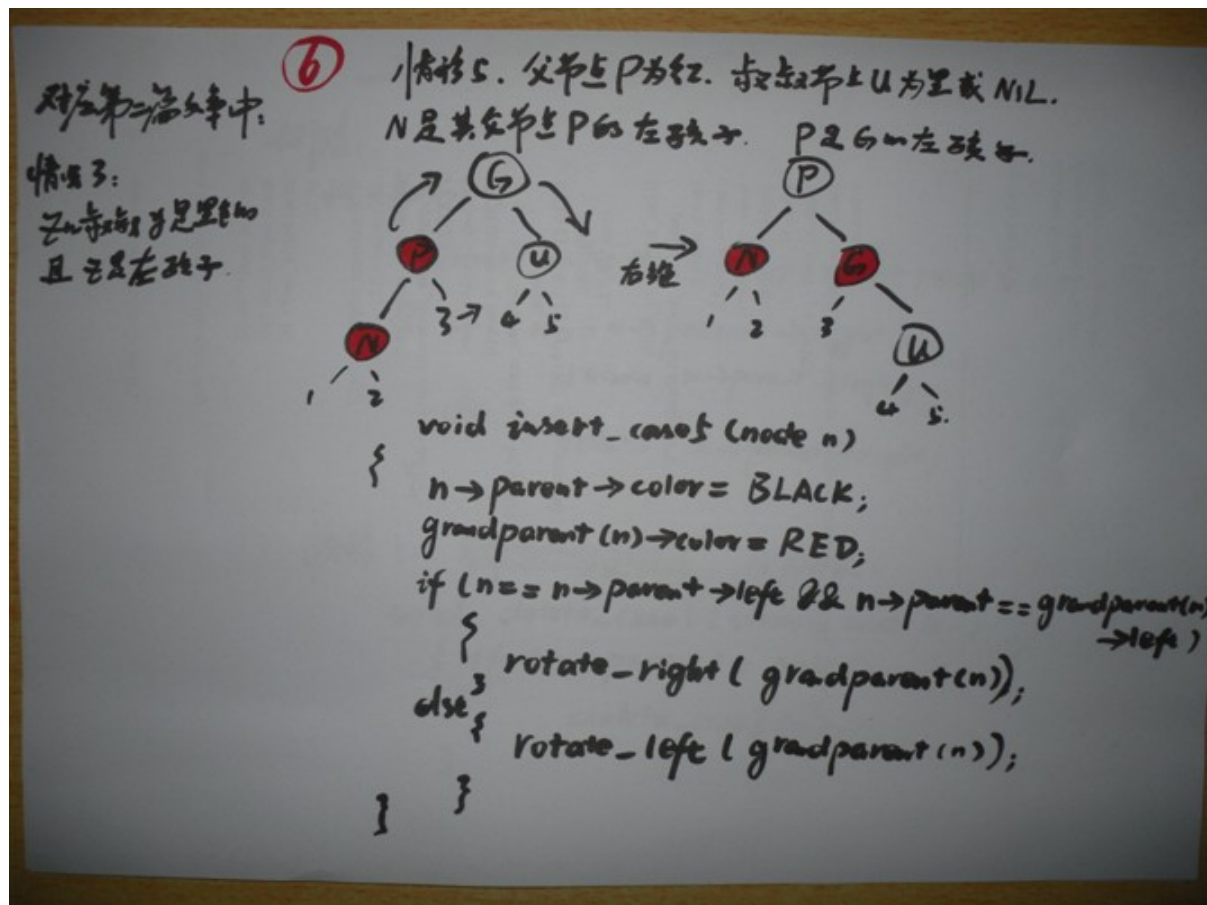
    /* 反情况，N 是其父节点的右孩子，而父节点P又是其父G的右孩子 */

    rotate_left(grandparent(n));

}

}

```



### 三、红黑树删除的几种情况

上文我们约定，兄弟节点设为S，我们使用下述函数找到兄弟节点：

```
struct node * sibling(struct node *n) //找兄弟节点
```

```
{
```

```
    if (n == n->parent->left)
```

```
        return n->parent->right;
```



```

else

    return n->parent->left;

}

```

情况1: N 是新的根。

void

delete\_case1(struct node \*n)

```

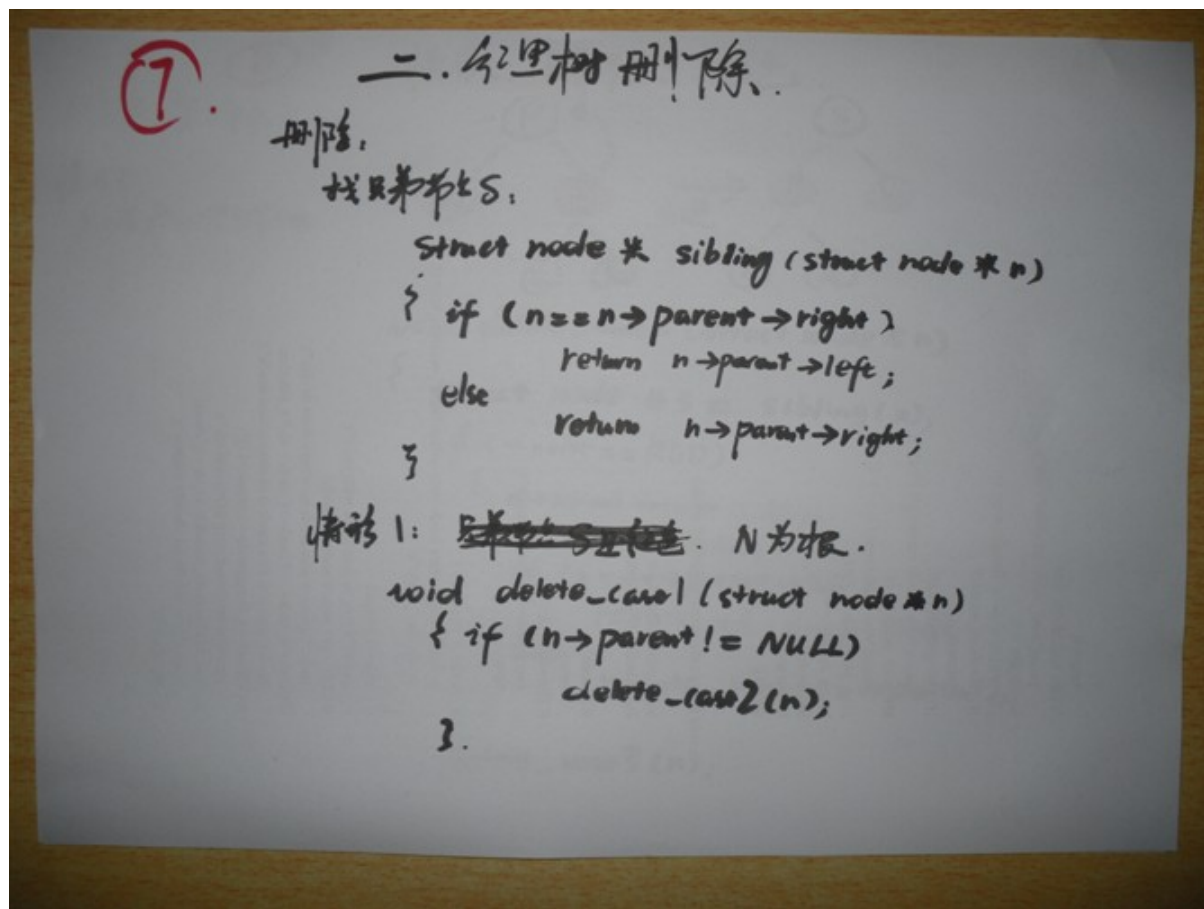
{

    if (n->parent != NULL)

        delete_case2(n);

}

```



情形2: 兄弟节点S是红色

[对应我第二篇文章中，情况1: x的兄弟w是红色的。]

void delete\_case2(struct node \*n)

```

{

```

```

struct node *s = sibling(n);

if (s->color == RED) {

    n->parent->color = RED;

    s->color = BLACK;

    if (n == n->parent->left)

        rotate_left(n->parent); //左旋

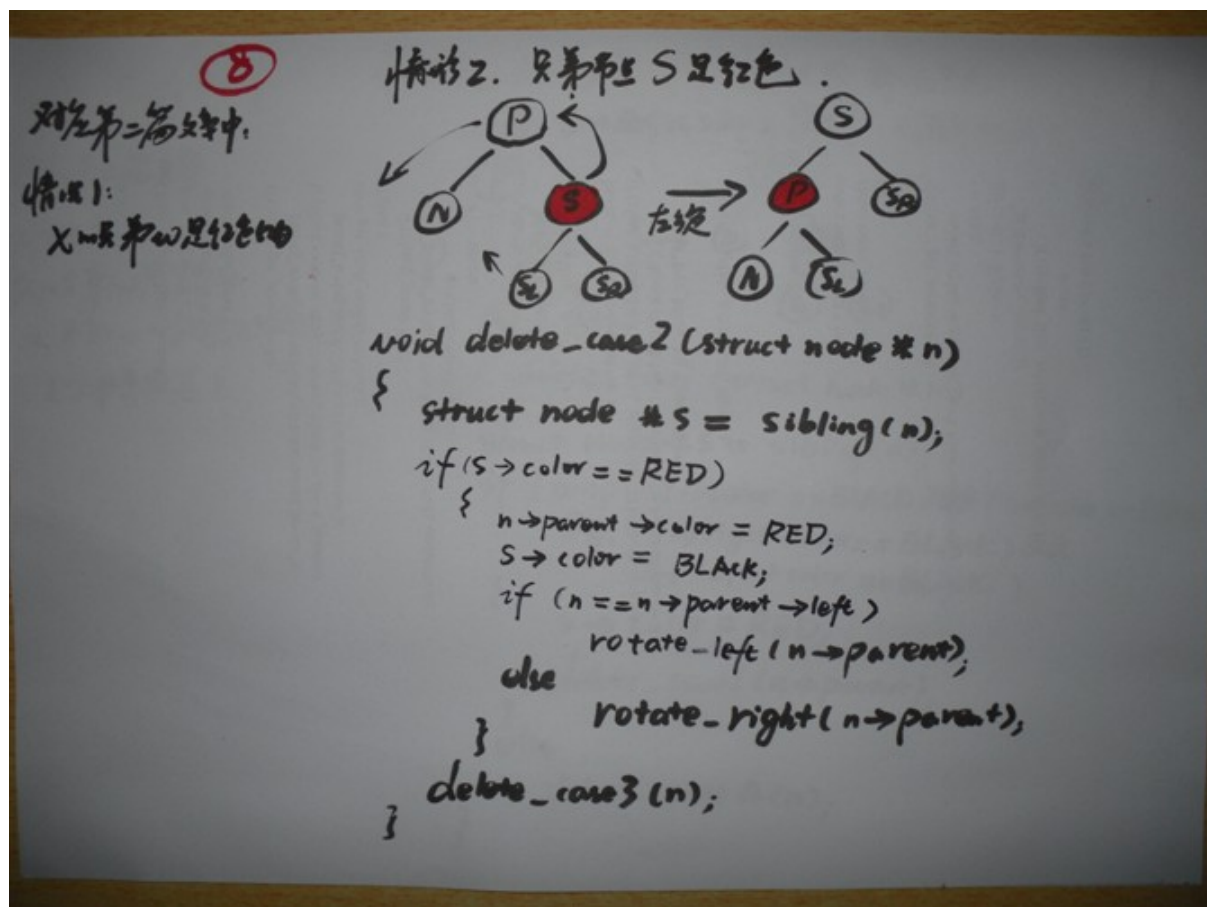
    else

        rotate_right(n->parent);

}

delete_case3(n);
}

```



情况 3: 兄弟节点S是黑色的，且S的俩个儿子都是黑色的。但N的父节点P，是黑色。

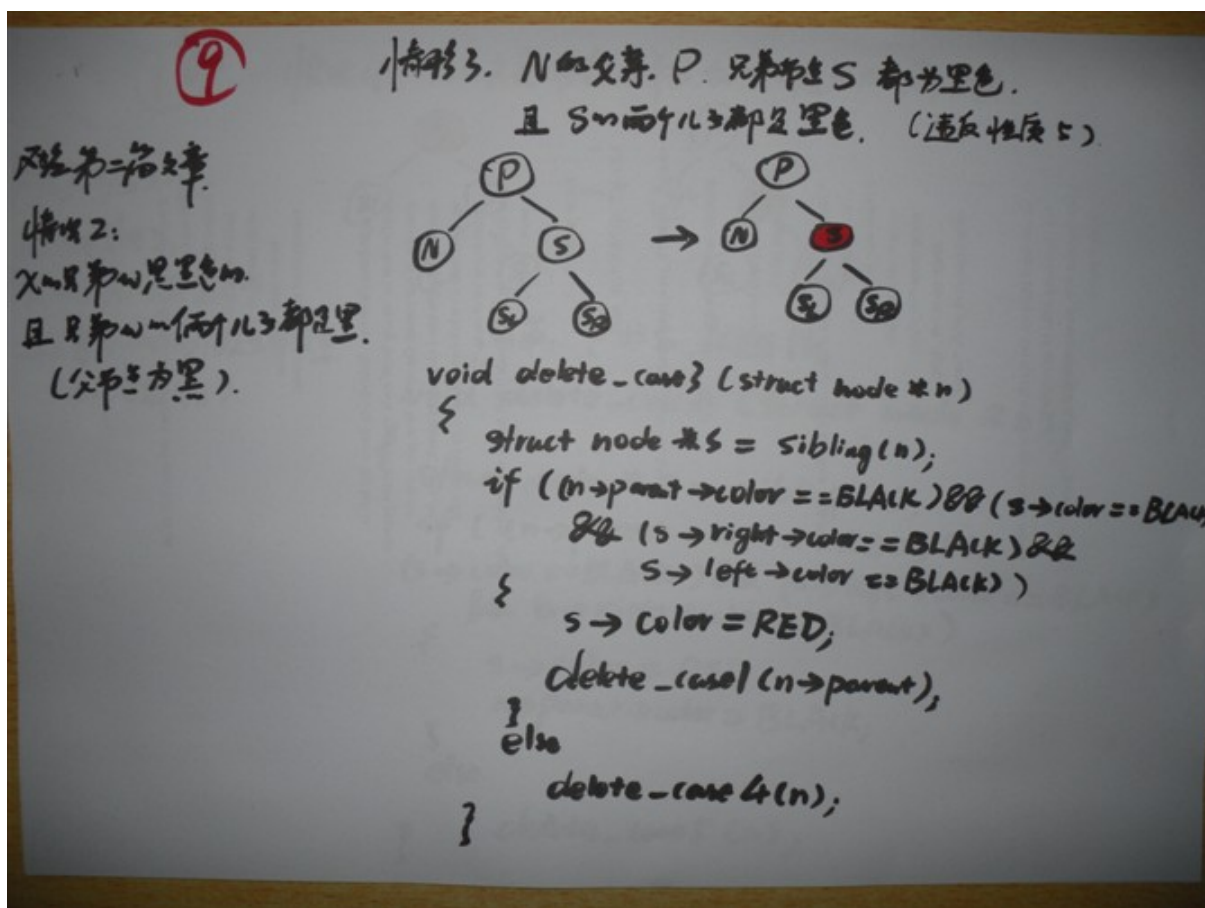
[对应我第二篇文章中，情况2: x的兄弟w是黑色的，且兄弟w的俩个儿子都是黑色的。

(这里，父节点P为黑)

```
void delete_case3(struct node *n)
{
    struct node *s = sibling(n);

    if ((n->parent->color == BLACK) &&
        (s->color == BLACK) &&
        (s->left->color == BLACK) &&
        (s->right->color == BLACK)) {
        s->color = RED;

        delete_case1(n->parent);
    } else
        delete_case4(n);
}
```



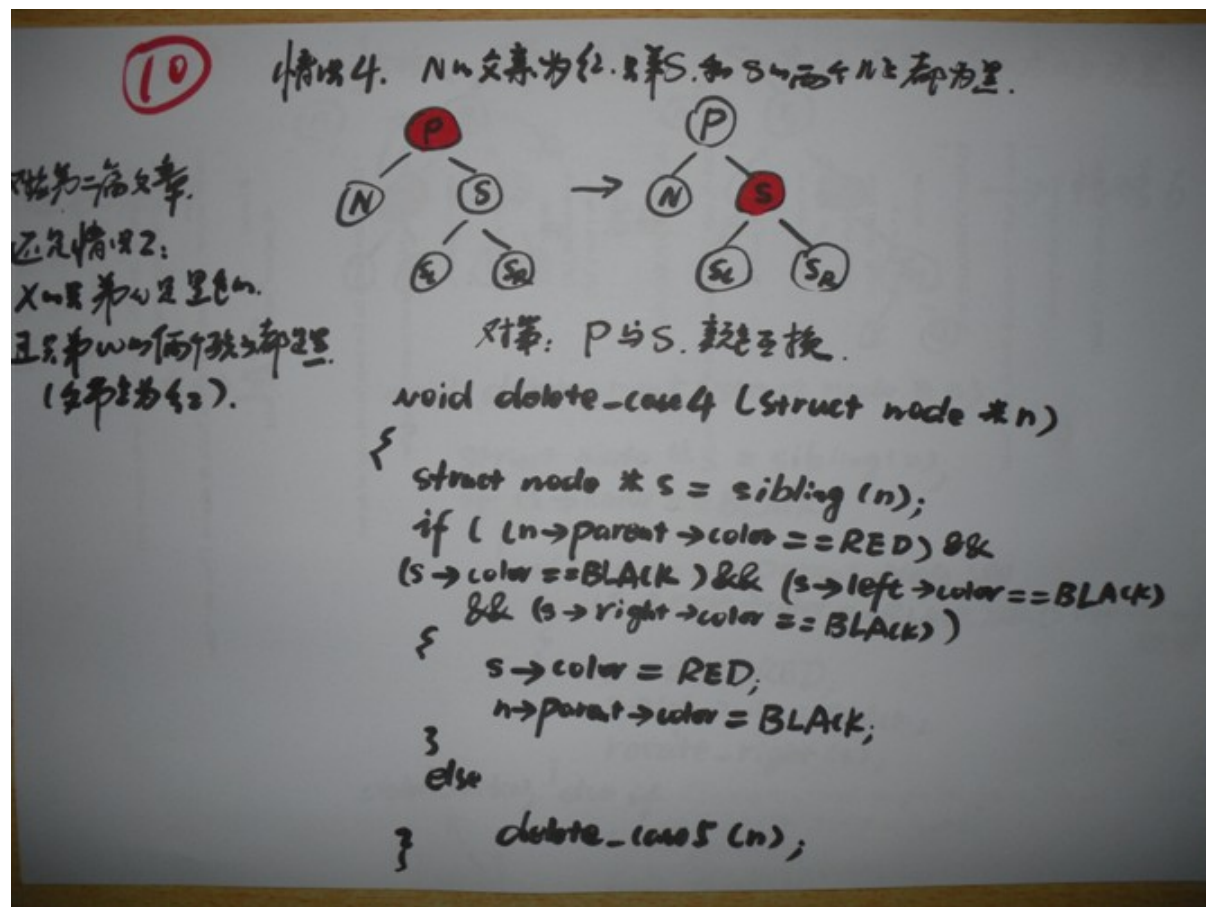
情况4: 兄弟节点S 是黑色的、S 的儿子也都是黑色的，但是 N 的父亲P，是红色。

[还是对应我第二篇文章中，情况2: x的兄弟w是黑色的，且w的俩个孩子都是黑色的。

(这里，父节点P为红)]

```
void delete_case4(struct node *n)
{
    struct node *s = sibling(n);

    if ((n->parent->color == RED) &&
        (s->color == BLACK) &&
        (s->left->color == BLACK) &&
        (s->right->color == BLACK)) {
        s->color = RED;
        n->parent->color = BLACK;
    } else
        delete_case5(n);
}
```



情况5: 兄弟S为黑色, S的左儿子是红色, S的右儿子是黑色, 而N是它父亲的左儿子。

//此种情况, 最后转化到下面的情况6。

[对应我第二篇文章中, 情况3: x的兄弟w是黑色的, w的左孩子是红色, w的右孩子是黑色。]

```
void delete_case5(struct node *n)
```

```
{
```

```
    struct node *s = sibling(n);
```

```
    if (s->color == BLACK)
```

```
        if ((n == n->parent->left) &&
```

```
            (s->right->color == BLACK) &&
```

```
            (s->left->color == RED)) {
```

```
                // this last test is trivial too due to cases 2-4.
```

```
                s->color = RED;
```

```
                s->left->color = BLACK;
```

```

rotate_right(s);

} else if ((n == n->parent->right) &&

(s->left->color == BLACK) &&

(s->right->color == RED)) {

// this last test is trivial too due to cases 2-4.

s->color = RED;

s->right->color = BLACK;

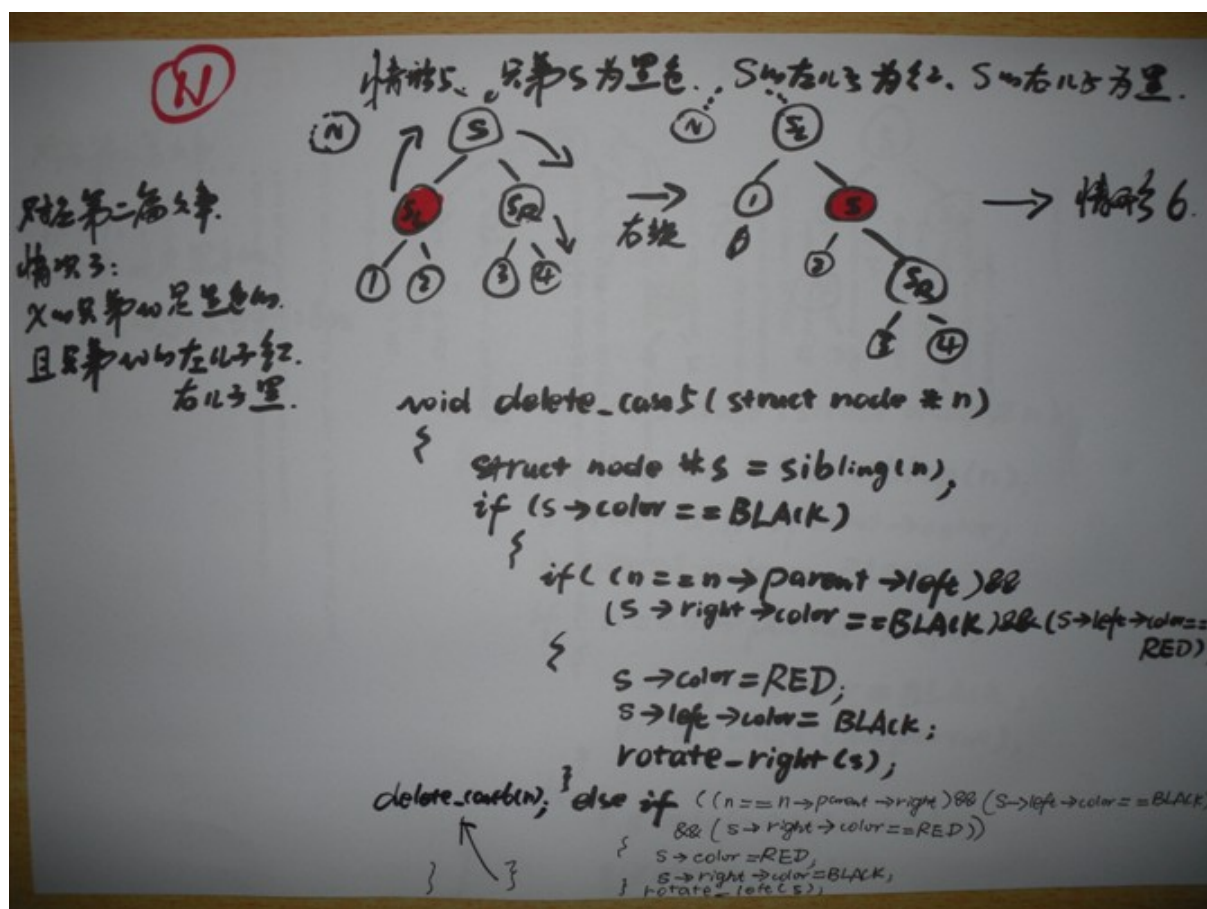
rotate_left(s);

}

}

delete_case6(n); //转到情况6。

```



情况6: 兄弟节点S是黑色，S的右儿子是红色，而N是它父亲的左儿子。

[对应我第二篇文章中，情况4:x的兄弟w是黑色的，且w的右孩子时红色的。]

```
void delete_case6(struct node *n)
```



```

{
    struct node *s = sibling(n);

    s->color = n->parent->color;

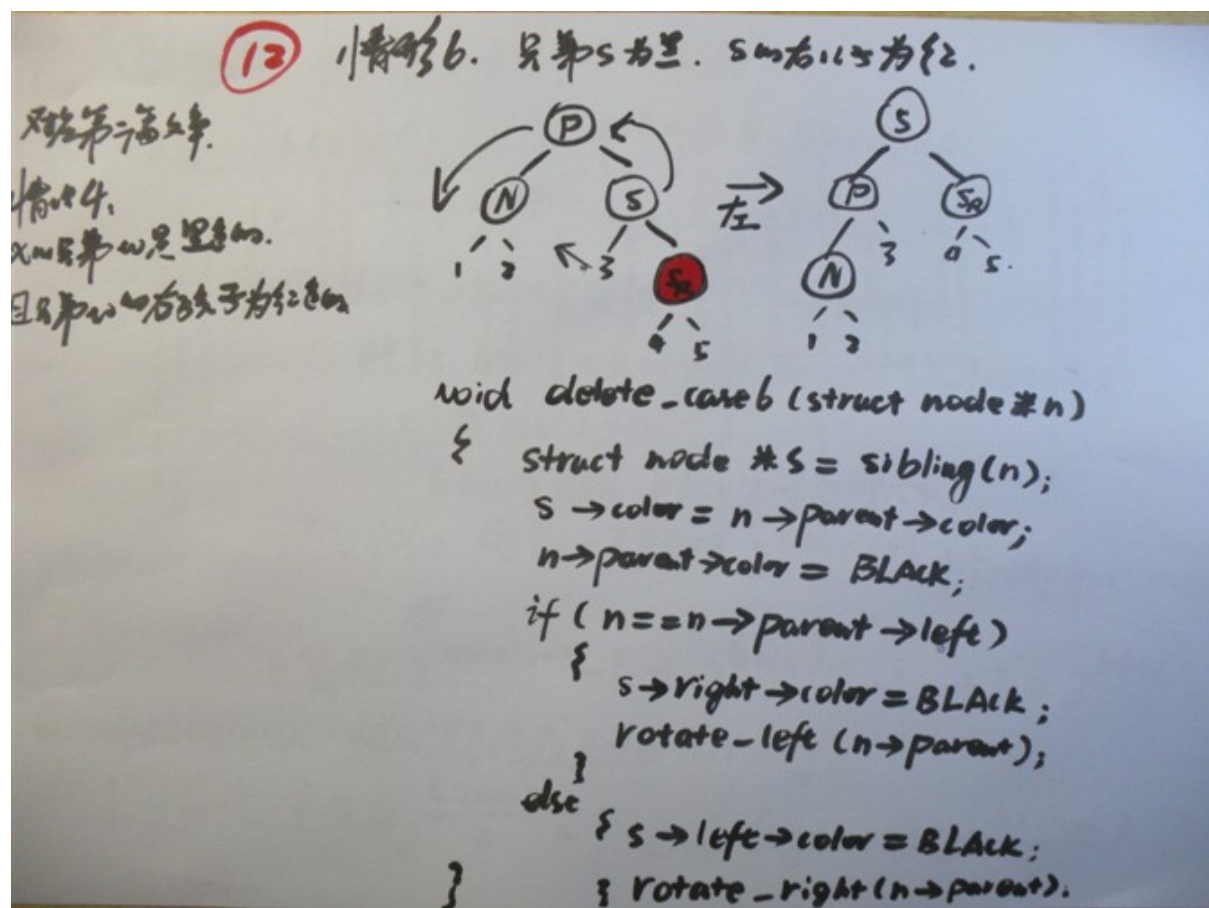
    n->parent->color = BLACK;

    if (n == n->parent->left) {
        s->right->color = BLACK;

        rotate_left(n->parent);
    } else {
        s->left->color = BLACK;

        rotate_right(n->parent);
    }
}

```



//呵呵，画这12张图，直接从中午画到了晚上。希望，此文能让你明白。

#### 四、红黑树的插入、删除情况时间复杂度的分析

因为每一个红黑树也是一个特化的二叉查找树，

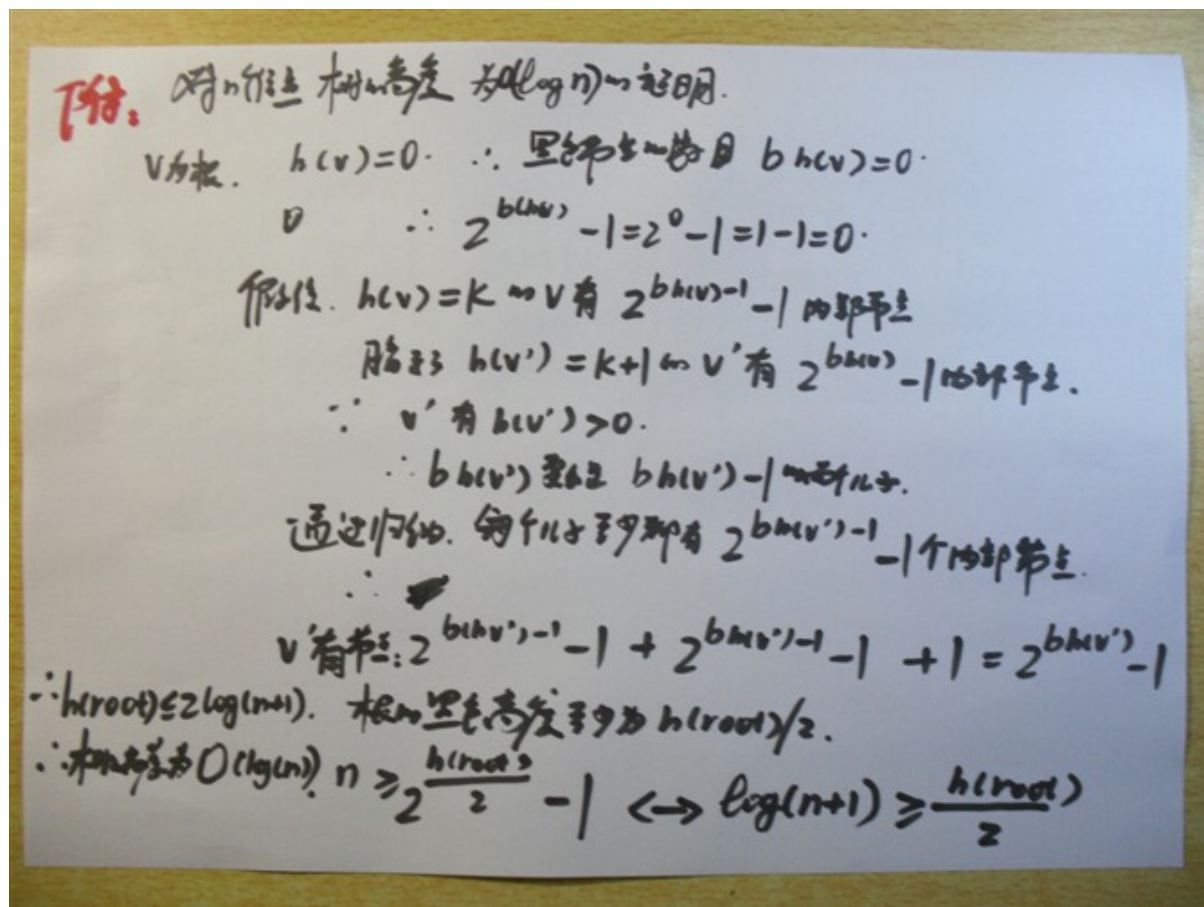
因此红黑树上的只读操作与普通二叉查找树上的只读操作相同。

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。



ok, 完。

后记:

此红黑树系列，前前后后，已经写了4篇文章，如果读者读完了这4篇文章，

对红黑树有一个相对之前来说，比较透彻的理解，

那么，也不枉费，我花这么多篇幅、花好几个钟头去画红黑树了。

真正理解一个数据结构、算法，最紧要的还是真正待用、实践的时候体会。

欢迎，各位，将现在、或以后学习、工作中运用此红黑树结构、算法的经验与我分享。

谢谢。:D。

-----

作者声明:

本人July对本博客所有文章和资料享有版权，转载、或引用任何内容请注明出处。

向您的厚道致敬。谢谢。二零一一年一月九日。

顶

13

踩




2

- 上一篇 程序员的美：极致与疯狂
- 下一篇 永久勘误:微软等面试100题答案V0.2版[第1-20题答案]

相关文章推荐

- |                                  |                             |
|----------------------------------|-----------------------------|
| • 一致性哈希及java实现                   | • 一步一图一代码，一定要让你真正彻底明白红黑树... |
| • MVVM在美团点评酒旅移动端的最佳实践--王禹华       | • Spring Boot 2小时入门基础教程     |
| • (六)2.4 Mysql Hash索引和B-tree索引区别 | • 一步一图一代码，一定要让你真正彻底明白红黑树    |
| • C语言大型软件设计的面向对象--宋宝华            | • Shell脚本编程                 |
| • 一步一图一代码，一定要让你真正彻底明白红黑树         | • 一步一图一代码，一定要让你真正彻底明白红黑树    |
| • Retrofit 从入门封装到源码解析            | • 一步一图一代码，一定要让你真正彻底明白红黑树    |
| • 一步一图一代码，一定要让你真正彻底明白红黑树         | • 一步一图一代码，一定要让你真正彻底明白红黑树    |
| • 跳过Java开发的各种坑                   | • 一步一步彻底实现红黑树，面试不再愁         |

查看评论

	水淹	45楼 2017-04-05 16:17发表
	水淹这篇文章我是真正理解了，谢谢，这篇感觉代码风格写得特别好！	 44楼 2017-03-20 14:29发表

<div>百里屠猪</div> <div>文章写的太乱</div>	<div></div>	<div>43楼 2016-09-11 18:49发表</div>
<div>firoyang</div> <div>你好，看了你的图，想问一下，如果我从空树一直做插入操作，第一个红节点是怎么出现的？是在插入更新之前的插入操作就已经根据性质5进行修改了么？</div>	<div></div>	<div>42楼 2016-05-13 16:05发表</div>
<div>FightingJaguar</div> <div>这不能算是理解，只是照着算法把过程说了一遍啊。</div>	<div></div>	<div>41楼 2016-01-04 14:09发表</div>
<div>明义经典</div> <div>感谢博主的付出</div>	<div></div>	<div>40楼 2015-12-10 20:26发表</div>
<div>希德小子</div> <div>你好博主，很感谢你这么用心的为我们讲解红黑树。 有个问题，删除代码中的 N 节点是被删除节点还是被删除节点的某个节点呢？看到最后删除节点也没有断掉联系。盼答复。  希望在原理方面添加一些描述，比如，插入、删除为什么要那么做，为什么要分那几种情形。希望不是简单得以“为了保持红黑树5条性质”作为那么做的理由。</div>	<div></div>	<div>39楼 2014-09-26 18:50发表</div>
<div> 圣剑ld</div> <div>回复guoshengchang：这个任务交给你了</div>		<div>Re: 2017-10-29 09:47发表</div>
<div> iloveipxzer</div> <div>ceven以为大学中缺少的就是像bz这样自身实力过硬，又能够将知识娓娓道来的老师，bz可以考虑下 ps：现在所谓的教授只会书上三板斧的知识</div>		<div>38楼 2014-09-22 10:35发表</div>
<div>wintersweetzeng</div> <div>博主，在插入的情形6里面，旋转完不是还要改颜色吗？不然p到1和到5的黑节点的个数不一样</div>	<div></div>	<div>37楼 2014-08-30 11:18发表</div>
<div>YorkCai</div> <div>很厉害 没得说</div>	<div></div>	<div>36楼 2014-07-28 20:00发表</div>
		<div>35楼 2013-10-20 18:17发表</div>

最近在看July大神的红黑树系列，讲解十分透彻，July大神也一直在强调要抓住红黑树的5大性质。请问July大神红黑树的5大性质是怎么设计出来的呢？难道是当初的创造者灵机一动吗？但是这也太强了吧。而且背后肯定数学证明，通过这个5个性质可以使红黑树保持近似平衡。



伶依

Re: 2014-07-22 01:16发表

回复YorkCai：红黑树是由2-3树改变来的，也可以说任何一个红黑树都和一个2-3树一一对应



桔凡

34楼 2013-04-06 12:23发表

ljin409177477学习方法。。。我上大二，苦苦挣扎不得要领。。。。



33楼 2013-01-11 17:44发表

hi，博主，红黑树删除的第二种情况有些困扰：“情形2：兄弟节点S是红色

[对应我第二篇文章中，情况1：x的兄弟w是红色的。]“但是我看了这种情况没有违反红黑树的5个性质啊，为什么要拉出来做处理呢？如果有，请问是违反了哪条性质？



小瑾

Re: 2013-01-16 21:48发表

回复ljin409177477：通过 N 的路径上对黑色节点数目减1



ljin409177477

Re: 2013-01-16 22:41发表

回复tang\_jin2015：你的意思是违反了第五条性质：对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点？这点不符合？



小瑾

Re: 2013-01-25 21:39发表

回复ljin409177477：是的呀，若原来删除节点为x，那么现在的n就是x的孩子节点，拉出来讲的原因是x是黑色节点，也就是说经过n的节点少了一个黑色节点，情况2的处理并没有改变经过每一个节点的黑色节点，只是为了更好的处理。从情况2会进入到情况3，4，5



小瑾

Re: 2013-01-25 21:40发表

回复tang\_jin2015：不知这样是否是正解，还望各位指出错误

**i\_Programmer**

唉,算法23爱在心口难开。  
拿什么拯救你，我的算法。

32楼 2012-08-01 00:53发表



31楼 2012-07-12 16:01发表

**yihukurama**

非常感谢博主的分享。(\*^\_\_^\*) 嘻嘻。有些疑问，想请教下~  
红黑树中的红子必生黑子，某结点后代中黑子数目相同使得，其最长路径的长度不会超过最短路径上的2倍。因为每条路径上的黑子数相同固定，红子的数目不会超过黑子。这样根据红黑树近似的递归性质，使得每个局部子树都保持平衡，即平衡因子的绝对值不超过1。从而可以得知红黑树是一棵平衡二叉树。通过其定义的性质以及红黑颜色表征平衡信息。但是从红黑树的五个性质是不能确定其是二叉排序树，除非定义为如此。  
弱弱的问一句，红黑树是平衡二叉树，平衡二叉树不改变其树结构，通过只设置结点颜色，是否可以变成红黑树？平衡二叉树与红黑树的联系和区别是什么？  
看了博客的红黑树的讲解，非常透彻简明，由于本人愚钝，总是知其然，不知其所以然。博主能否简单讲解红黑树插入删除背后的原理或思想。如左旋右旋就是在升降之间来调节平衡~  
再次表示感谢~~



30楼 2012-06-25 16:41发表

**歌神的卖**

谢谢博主，花那么大心思为我们这些小菜。另外还有个问题想问问，删除操作情况5情况6说n是他父亲的左孩子，如果n是他父亲的右孩子呢？为什么没有这种情况，是不可能发生还是已经包含在某些地方了？导论书上写得不详尽啊。



29楼 2012-05-12 16:42发表

**do616691932**

感谢博主



28楼 2012-04-06 22:10发表

**零度空间0520**

膜拜之 好厉害 我看STL看了好久 被绕死了~~~~~



27楼 2012-03-28 14:59发表

**零度空间0520**

你太伟大了，



26楼 2012-03-28 08:30发表

**yuan1024**

太厉害了。好佩服啊。。



25楼 2012-03-24 15:11发表

**SVKING**

楼主真是太牛了，讲得真的够详细的，特别有条理性，非常适合新手看看，红黑树，讲的经典



24楼 2012-03-14 14:15发表



<div>emeraldttt</div> <div>楼主辛苦了，花了这么长时间，不容易啊！！！</div>		23楼 2012-03-01 09:13发表
<div>youqika</div> <div>楼主东西太好了。</div>		22楼 2012-02-01 18:16发表
<div>bz怎么不去大学当老师啊，可以拯救很多学生</div>		
<hr/>		
<div>v_JULY_v</div> <div>回复youqika：额，这个，得看机会，机会来了自然便去了</div>		Re: 2012-03-28 08:52发表
<div>Never_for_Never</div> <div>看了这篇文章！.....楼主辛苦了！ 但是：这样看就算懂了如何操作，也没意思！不是自己的~</div>		21楼 2011-11-27 10:40发表 20楼 2011-10-09 10:36发表
<div>zszjian</div> <div>你好，我刚接触红黑树，是不是使用红黑树时只要记得规则就行了，怎样才能更深一层的理解其精髓呢（每种情况为何要那样处理），谢谢</div>		19楼 2011-08-09 15:48发表
<div>woshinideshenm</div> <div>第二张图的S节点位置好像有点问题？</div>		18楼 2011-07-13 20:35发表
<div>汤圆甜筒</div> <div>太性了，一般人做不到啊</div>		17楼 2011-06-08 10:10发表
<div>randyjiawenjie</div> <div>[e04]</div>		16楼 2011-05-18 16:53发表
<div>xitan198704</div> <div>[e03][e03][e03]</div>		15楼 2011-05-17 23:47发表
<div>angtylook</div> <div>赞楼主这么认真和热心的精神，内容也非常好！</div>		14楼 2011-05-02 22:30发表

## 空穴来风

[e07]还是不知道为什么这样子着色，旋转就能保持红黑树的性质不变。



13楼 2011-04-04 23:57发表

## caoxuanwei

[e01]



12楼 2011-03-14 16:29发表

## mascon

[e01]



11楼 2011-03-12 12:46发表

## yxm870915

项一下，多谢分享哈



10楼 2011-02-19 00:22发表

## qqhmitzk

[e01]



9楼 2011-02-14 11:37发表

## 北极熊

算法方面的好文章，谢谢共享



8楼 2011-01-24 10:47发表

## q363742533

[e01]



7楼 2011-01-21 19:00发表

不错，顶起！



v\_JULY\_v

Re: 2011-01-22 17:33发表

回复 q363742533: thanks。



suds

6楼 2011-01-17 17:35发表

太帅了，我好好看看



v\_JULY\_v

Re: 2011-01-22 17:34发表

回复 suds: :D。



chhrsas

5楼 2011-01-12 14:02发表

不得不顶啊。

chjttony



4楼 2011-01-10 17:02发表

收藏!



v\_JULY\_v

Re: 2011-01-10 17:05发表

回复 chjttony: ~



v\_JULY\_v

3楼 2011-01-09 10:57发表

至于为什么要重新着色，左旋、右旋，请紧紧抓住红黑树的5点性质不放。所有的工作，重新着色、左旋或右旋，都只是为了继续保持红黑树的5点性质。



2楼 2011-01-09 10:41发表

好好体会红黑树插入、删除的那几种情况。



v\_JULY\_v

Re: 2011-01-09 10:41发表

回复 v\_JULY\_v: 红黑树删除的几种情况。

情况1: x的兄弟w是红色的。

情况2: x的兄弟w是黑色的，且w的两个孩子都是黑色的。

情况3: x的兄弟w是黑色的，且w的左孩子是红色，w的右孩子是黑色。

情况4: x的兄弟w是黑色的，且w的右孩子是红色的。



v\_JULY\_v

Re: 2011-01-09 10:41发表

回复 v\_JULY\_v: 红黑树插入的几种情况:

情况1, z的叔叔y是红色的。

情况2: z的叔叔y是黑色的，且z是右孩子

情况3: z的叔叔y是黑色的，且z是左孩子



v\_JULY\_v

1楼 2011-01-09 10:25发表

单单这12张图，就画了我昨天整整一个下午。希望，此文能更有用。