红黑树算法的层层剖析与逐步实现

作者 July 二零一零年十二月三十一日

本文主要参考：算法导论第二版

本文主要代码：参考算法导论。

本文图片来源：个人手工画成、算法导论原书。

推荐阅读：Leo J. Guibas 和 Robert Sedgewick 于1978年写的关于红黑树的一篇论文。

引言：

昨天下午画红黑树画了好几个钟头，总共10页纸。

特此，再深入剖析红黑树的算法实现，教你如何彻底实现红黑树算法。

经过我上一篇博文，“教你透彻了解红黑树”后，相信大家对红黑树已经有了一定的了解。

个人觉得，这个红黑树，还是比较容易懂的。

不论是插入、还是删除，不论是左旋还是右旋，最终的目的只有一个：

即保持红黑树的5个性质，不得违背。

再次，重述下红黑树的五个性质：

一般的，红黑树，满足一下性质，即只有满足一下性质的树，我们才称之为红黑树：

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。

抓住了红黑树的那5个性质，事情就好办多了。

如，

1.红黑红黑，要么是红，要么是黑；

2.根结点是黑；

3.每个叶结点是黑；

4.一个红结点，它的俩个儿子必然都是黑的；

5.每一条路径上，黑结点的数目等同。

五条性质，合起来，来句顺口溜就是：（1）红黑 （2）黑 （3）黑 （4&5）红->黑 黑。

本文所有的文字，都是参照我昨下午画的**十张纸（即我拍的照片）**与算法导论来写的。

希望，你依照此文一点一点的往下看，看懂此文后，你对红黑树的算法了解程度，一定大增不少。

ok，现在咱们来具体深入剖析红黑树的算法，并教你逐步实现此算法。

此教程分为10个部分，每一个部分作为一个小节。且各小节与我给的十张照片一一对应。

一、左旋与右旋

先明确一点：为什么要左旋?

因为红黑树插入或删除结点后，树的结构发生了变化，从而可能会破坏红黑树的性质。

为了维持插入、或删除结点后的树，仍然是一颗红黑树，所以有必要对树的结构做部分调整，从而恢复红黑树的原本性质。

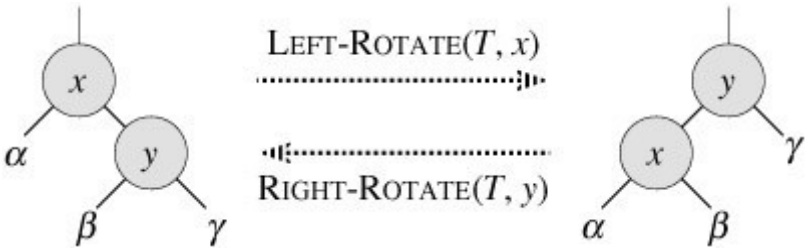
而为了恢复红黑性质而作的动作包括：

结点颜色的改变(重新着色)，和结点的调整。

这部分结点调整工作，改变指针结构，即是通过左旋或右旋而达到目的。

从而使插入、或删除结点的树重新成为一颗新的红黑树。

ok，请看下图：



如上图所示，‘找茬’

如果你看懂了上述俩幅图有什么区别时，你就知道什么是“左旋”，“右旋”。

在此，着重分析左旋算法：

左旋，如图所示（左->右），以x->y之间的链为“支轴”进行，

使y成为该新子树的根，x成为y的左孩子，而y的左孩子则成为x的右孩子。

算法很简单，还有注意一点，各个结点从左往右，不论是左旋前还是左旋后，结点大小都是从小到大。

左旋代码实现，分三步（注意我给的注释）：

The pseudocode for LEFT-ROTATE assumes that right[x] ≠ nil[T] and that the root's parent is nil[T].

```
LEFT-ROTATE(T, x)
 1  y ← right[x]            ▷ Set y.
 2  right[x] ← left[y]              //开始变化，y的左孩子成为x的右孩子
 3  if left[y]   ! =nil[T]
 4  then p[left[y]] <- x
 5  p[y] <- p[x]                    //y成为x的父结点
 6  if p[x] = nil[T]
 7     then root[T] <- y
 8     else if x = left[p[x]]
 9             then left[p[x]] ← y
```

```
10          else right[p[x]] ← y
11  left[y] ← x             //x成为y的左孩子（一月三日修正）
12  p[x] ← y
```

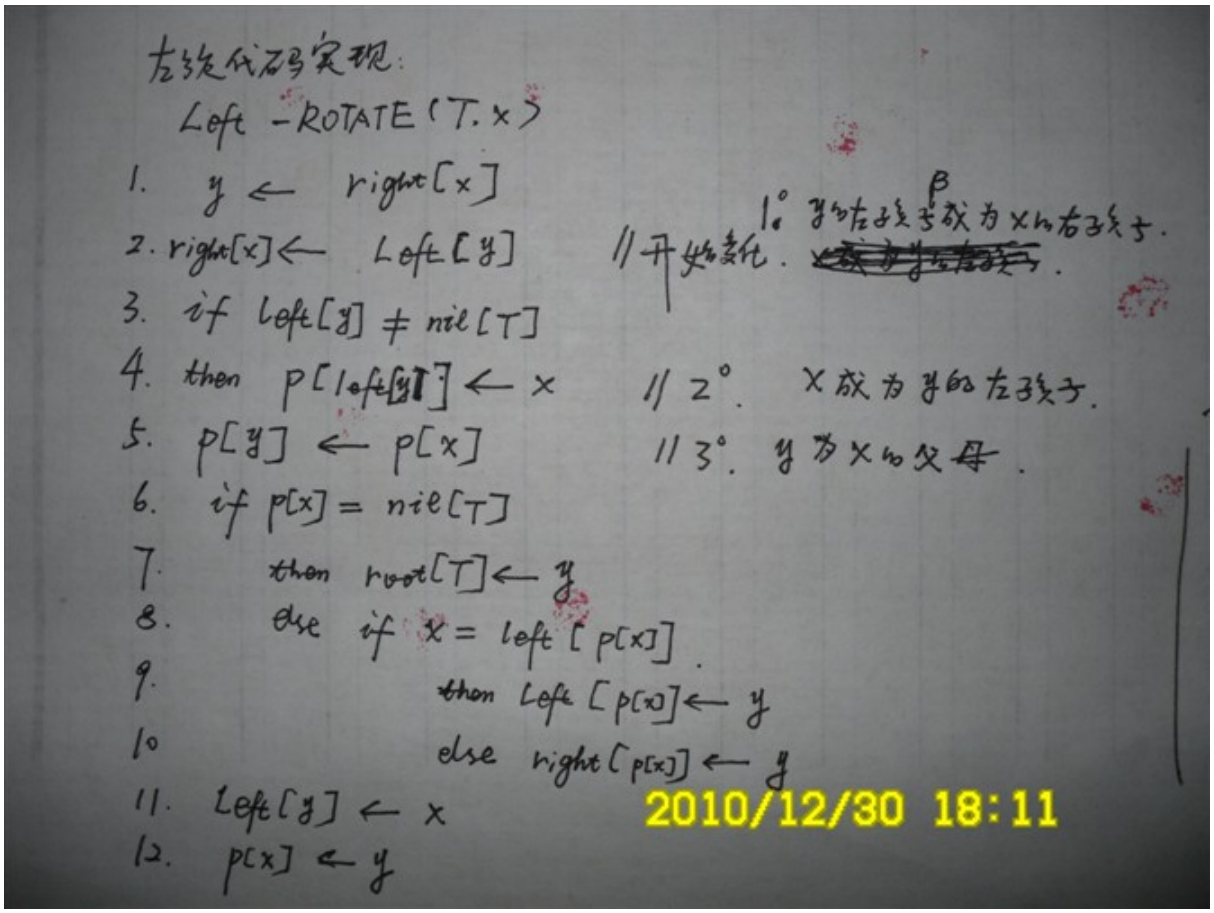
//注，此段左旋代码，原书第一版英文版与第二版中文版，有所出入。

//个人觉得，第二版更精准。所以，此段代码以第二版中文版为准。

左旋、右旋都是对称的，且都是在O（1）时间内完成。因为旋转时只有指针被改变，而结点中的所有域都保持不变。

最后，贴出昨下午关于此左旋算法所画的图：

左旋（第2张图）：

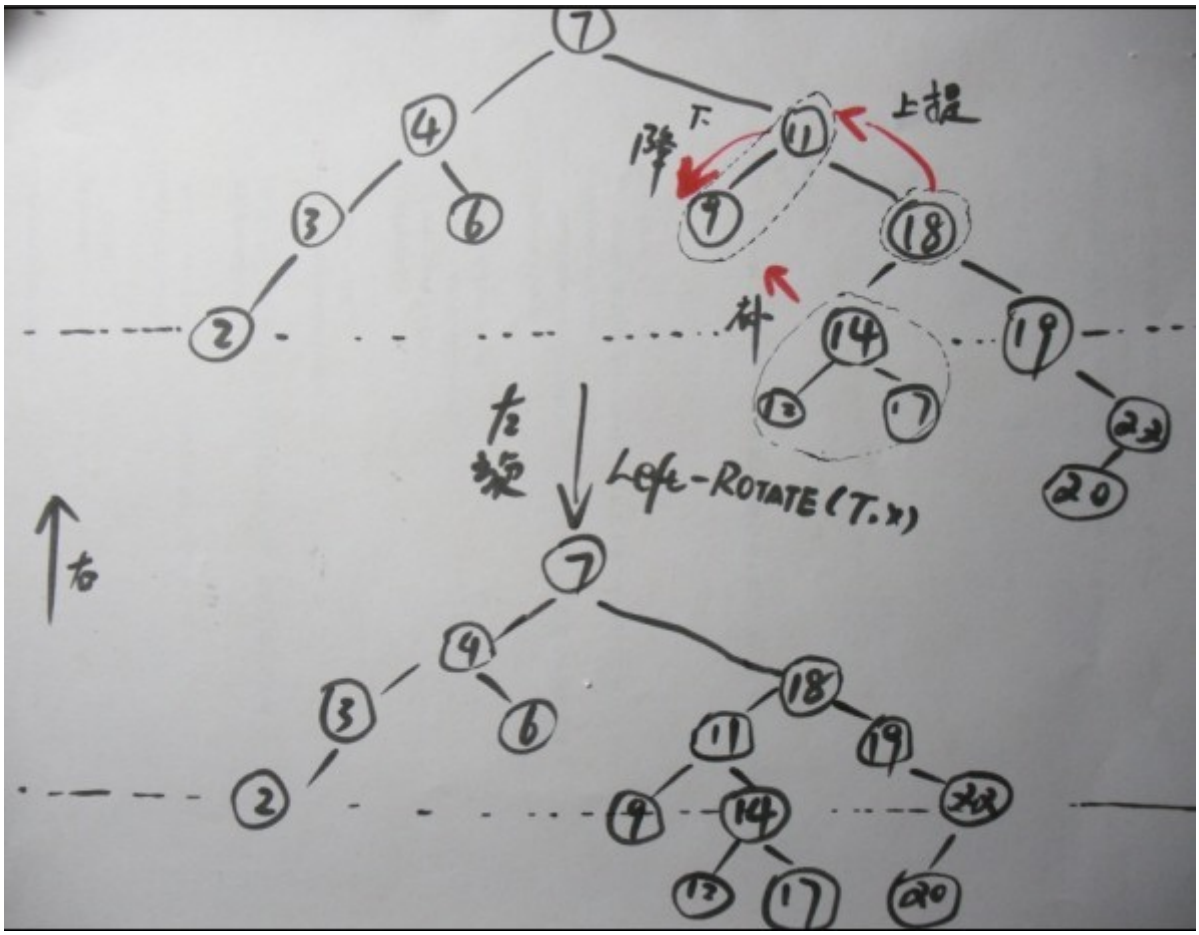


//此图有点bug。第4行的注释移到第11行。如上述代码所示。（一月三日修正）

二、左旋的一个实例

不做过多介绍，看下副图，一目了然。

LEFT-ROTATE(T, x)的操作过程（第3张图）：



提醒，看下文之前，请首先务必明确，区别以下俩种操作：

1.红黑树插入、删除结点的操作

//如插入中，红黑树插入结点操作：RB-INSERT(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。

//如插入中，为了恢复和保持原有红黑性质，所做的工作：RB-INSERT-FIXUP(T, z)。

ok，请继续。

三、红黑树的插入算法实现

RB-INSERT(T, z) //注意我给的注释...

```
1  y ← nil[T]              // y 始终指向 x 的父结点。
2  x ← root[T]             // x 指向当前树的根结点，
3  while x ≠ nil[T]
```

```
 4     do y ← x
 5        if key[z] < key[x]          //向左，向右..
 6           then x ← left[x]
 7           else x ← right[x]        // 为了找到合适的插入点，x 探路跟踪路径，直到x成为NIL 为止。
 8  p[z] ← y        // y置为 插入结点z 的父结点。
 9  if y = nil[T]
10     then root[T] ← z
11     else if key[z] < key[y]
12             then left[y] ← z
13             else right[y] ← z     //此 8-13行，置z 相关的指针。
14  left[z] ← nil[T]
15  right[z] ← nil[T]           //设为空，
16  color[z] ← RED              //将新插入的结点z作为红色
17  RB-INSERT-FIXUP(T, z)   //因为将z着为红色，可能会违反某一红黑性质，
                                //所以需要调用RB-INSERT-FIXUP(T, z)来保持红黑性质。
```

17 行的**RB-INSERT-FIXUP(T, z)** ，在下文会得到着重而具体的分析。

还记得，我开头说的那句话么，

是的，时刻记住，不论是左旋还是右旋，不论是插入、还是删除，都要记得恢复和保持红黑树的5个性质。

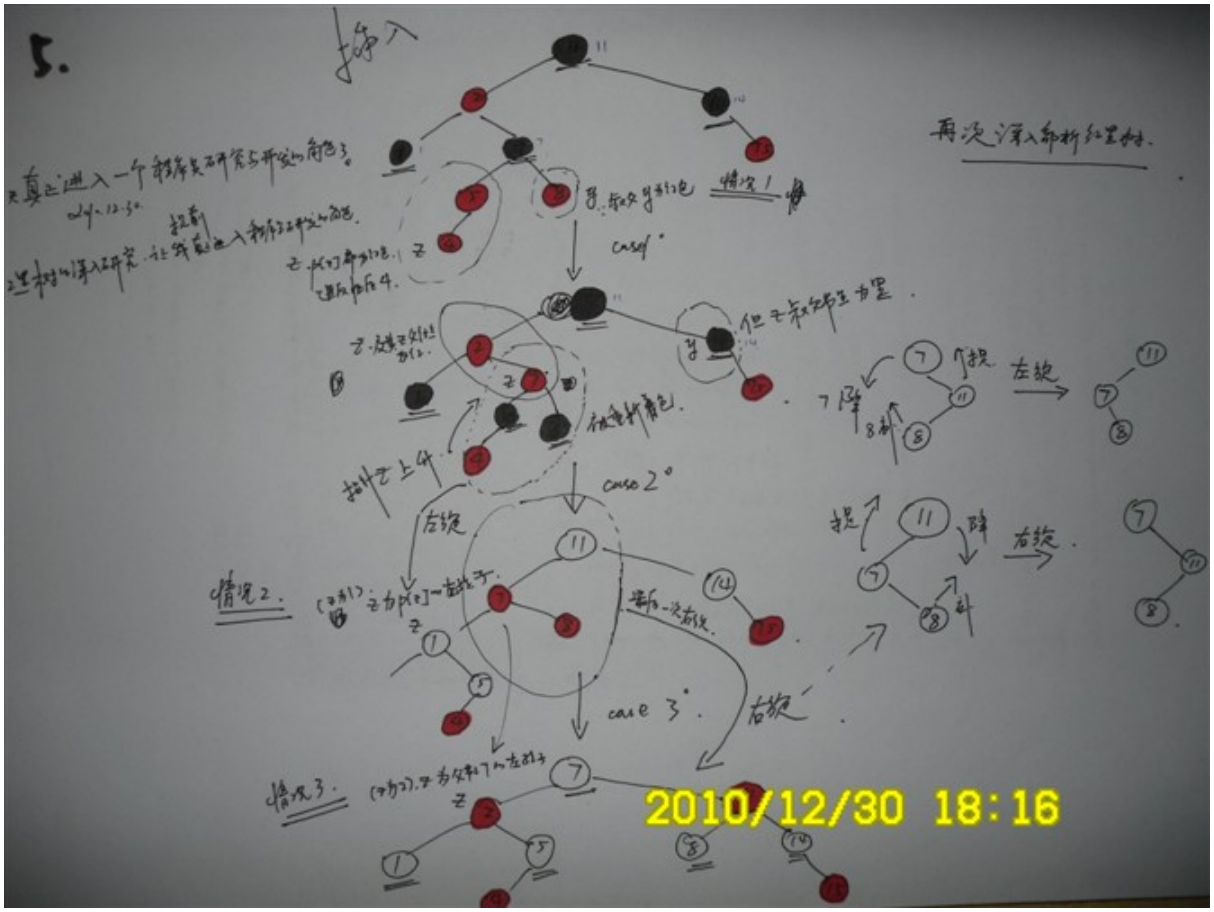
四、调用RB-INSERT-FIXUP(T, z)来保持和恢复红黑性质

```
RB-INSERT-FIXUP(T, z)
 1 while color[p[z]] = RED
 2    do if p[z] = left[p[p[z]]]
 3          then y ← right[p[p[z]]]
 4               if color[y] = RED
 5                  then color[p[z]] ← BLACK                ▷ Case 1
 6                       color[y] ← BLACK                   ▷ Case 1
```

```
 7                color[p[p[z]]] ← RED                ▷ Case 1
 8                z ← p[p[z]]                         ▷ Case 1
 9            else if z = right[p[z]]
10                then z ← p[z]                       ▷ Case 2
11                     LEFT-ROTATE(T, z)              ▷ Case 2
12                color[p[z]] ← BLACK                 ▷ Case 3
13                color[p[p[z]]] ← RED                ▷ Case 3
14                RIGHT-ROTATE(T, p[p[z]])            ▷ Case 3
15        else (same as then clause
                  with "right" and "left" exchanged)
16 color[root[T]] ← BLACK
```

//第4张图略：

五、红黑树插入的三种情况，即**RB-INSERT-FIXUP(T, z)**。操作过程（第5张）：



//这幅图有个小小的问题，读者可能会产生误解。图中左侧所表明的情况2、情况3所标的位置都要标上一点。

//请以图中的标明的case1、case2、case3为准。一月三日。

六、红黑树插入的第一种情况（RB-INSERT-FIXUP(T, z)代码的具体分析一）

为了保证阐述清晰，重述下RB-INSERT-FIXUP(T, z)的源码：

```
RB-INSERT-FIXUP(T, z)
 1 while color[p[z]] = RED
 2     do if p[z] = left[p[p[z]]]
 3           then y ← right[p[p[z]]]
 4                if color[y] = RED
 5                   then color[p[z]] ← BLACK                ▷ Case 1
 6                        color[y] ← BLACK                   ▷ Case 1
 7                        color[p[p[z]]] ← RED               ▷ Case 1
 8                        z ← p[p[z]]                        ▷ Case 1
 9                else if z = right[p[z]]
10                        then z ← p[z]                      ▷ Case 2
11                             LEFT-ROTATE(T, z)             ▷ Case 2
12                        color[p[z]] ← BLACK                ▷ Case 3
13                        color[p[p[z]]] ← RED               ▷ Case 3
14                        RIGHT-ROTATE(T, p[p[z]])           ▷ Case 3
15          else (same as then clause
                     with "right" and "left" exchanged)
16 color[root[T]] ← BLACK
```

//case1表示情况1，case2表示情况2，case3表示情况3.

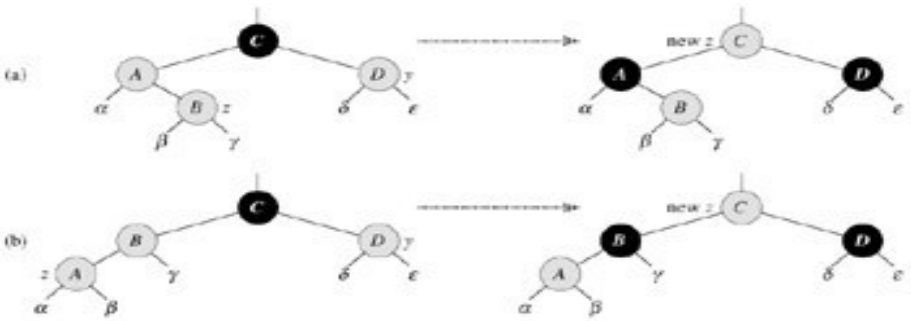
ok，如上所示，相信，你已看到了。

咱们，先来透彻分析红黑树插入的第一种情况：

插入情况1，z的叔叔y是红色的。

第一种情况，即上述代码的第5-8行：

```
5        then color[p[z]] ← BLACK              ▷ Case 1
6             color[y] ← BLACK                 ▷ Case 1
7             color[p[p[z]]] ← RED             ▷ Case 1
8             z ← p[p[z]]                      ▷ Case 1
```



如上图所示，a：z为右孩子，b：z为左孩子。

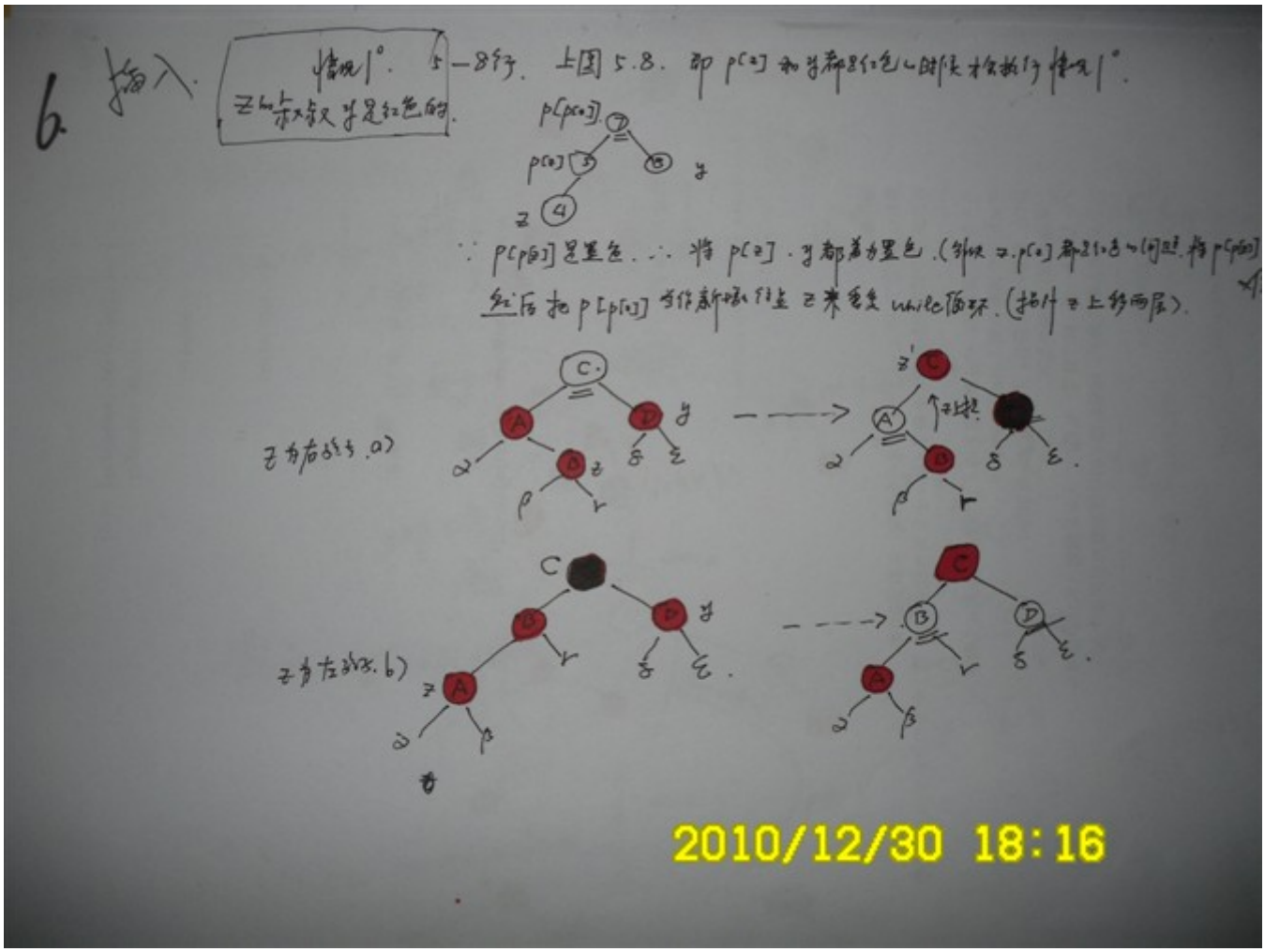
只有p[z]和y（上图a中A为p[z]，D为z，上图b中，B为p[z]，D为y）都是红色的时候，才会执行此情况1.

咱们分析下上图的a情况，即z为右孩子时

因为p[p[z]]，即c是黑色，所以将p[z]、y都着为黑色（如上图a部分的右边），

此举解决z、p[z]都是红色的问题，将p[p[z]]着为红色，则保持了性质5.

ok，看下我昨天画的图（第6张）：



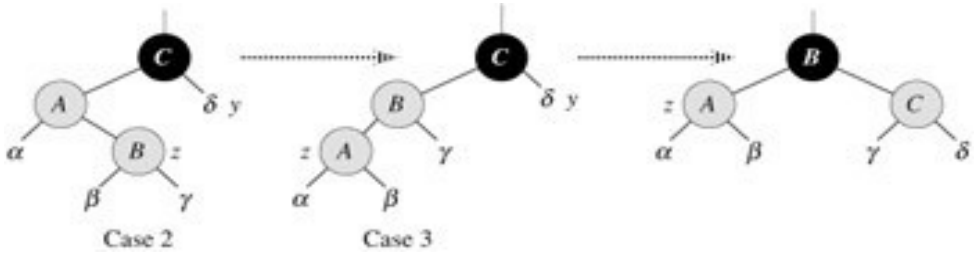
红黑树插入的第一种情况完。

七、红黑树插入的第二种、第三种情况

插入情况2：z的叔叔y是黑色的，且z是右孩子

插入情况3：z的叔叔y是黑色的，且z是左孩子

这俩种情况，是通过z是p[z]的左孩子，还是右孩子区别的。



参照上图，针对情况2，z是她父亲的右孩子，则为了保持红黑性质，左旋则变为情况3，此时z为左孩子，

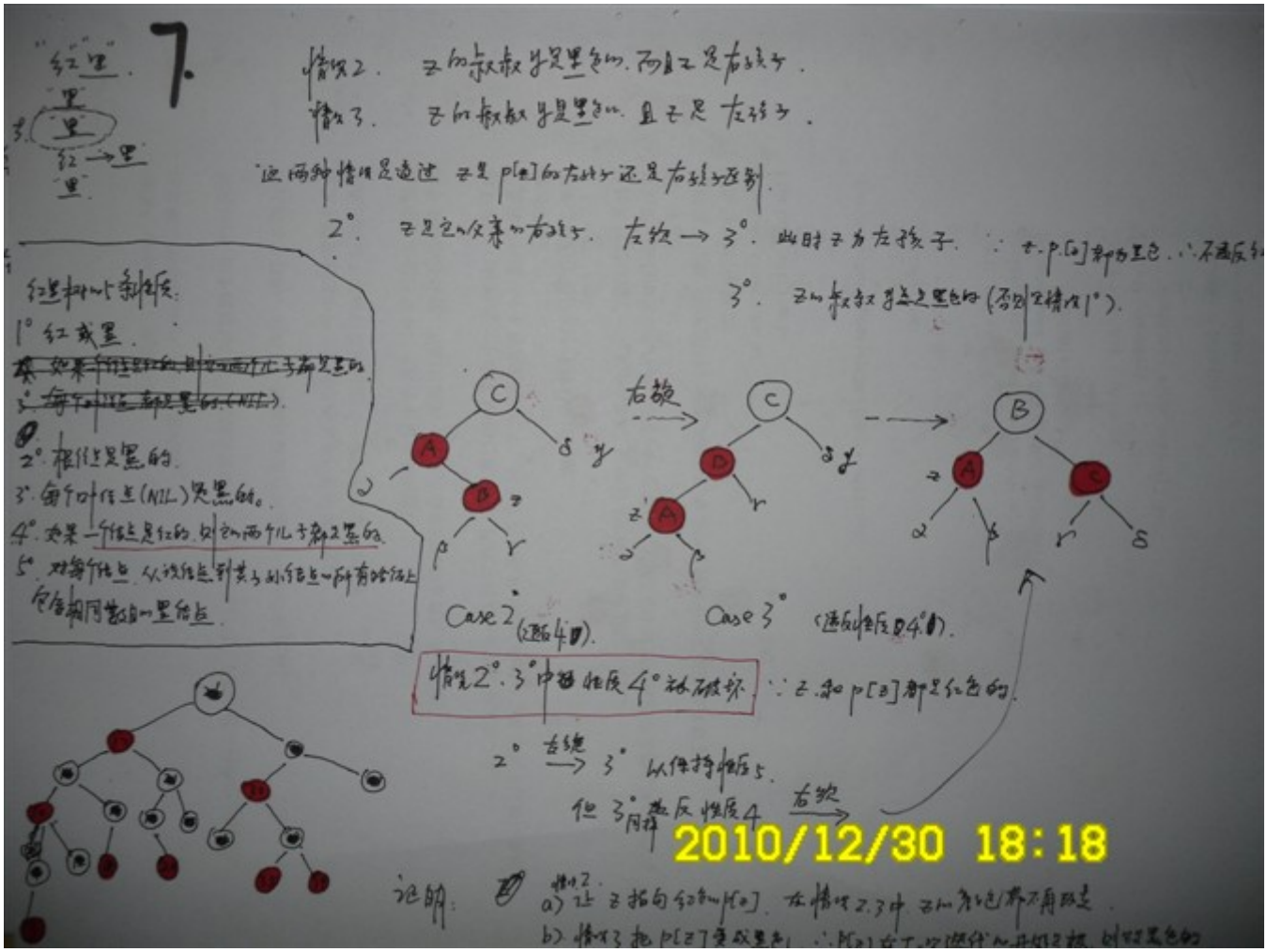
因为z、p[z]都为黑色，所以不违反红黑性质（注，情况3中，z的叔叔y是黑色的，否则此种情况就变成上述情况1 了）。

ok，我们已经看出来了，情况2，情况3都违反性质4（一个红结点的俩个儿子都是黑色的）。

所以情况2->左旋后->情况3，此时情况3同样违反性质4，所以情况3->右旋，得到上图的最后那部分。

注，情况2、3都只违反性质4，其它的性质1、2、3、5都不违背。

好的，最后，看下我画的图（第7张）：



八、接下来，进入红黑树的删除部分。

```
RB-DELETE(T, z)
 1 if left[z] = nil[T] or right[z] = nil[T]
 2    then y ← z
 3    else y ← TREE-SUCCESSOR(z)
 4 if left[y] ≠ nil[T]
 5    then x ← left[y]
 6    else x ← right[y]
 7 p[x] ← p[y]
 8 if p[y] = nil[T]
 9    then root[T] ← x
10    else if y = left[p[y]]
11            then left[p[y]] ← x
12            else right[p[y]] ← x
13 if y 3≠ z
14    then key[z] ← key[y]
15         copy y's satellite data into z
16 if color[y] = BLACK                //如果y是黑色的，
17    then RB-DELETE-FIXUP(T, x)   //则调用RB-DELETE-FIXUP(T, x)
18 return y                //如果y不是黑色，是红色的，则当y被删除时，红黑性质仍然得以保持。不做操作，返回。
                           //因为：1.树种各结点的黑高度都没有变化。2.不存在俩个相邻的红色结点。
                                 //3.因为入宫y是红色的，就不可能是根。所以，根仍然是黑色的。
```

ok，第8张图，不必贴了。

九、红黑树删除之4种情况，RB-DELETE-FIXUP(T, x)之代码

```
RB-DELETE-FIXUP(T, x)
 1 while x ≠ root[T] and color[x] = BLACK
```

```
2   do if x = left[p[x]]
3         then w ← right[p[x]]
4              if color[w] = RED
5                 then color[w] ← BLACK                    ▷ Case 1
6                      color[p[x]] ← RED                   ▷ Case 1
7                      LEFT-ROTATE(T, p[x])                ▷ Case 1
8                      w ← right[p[x]]                     ▷ Case 1
9              if color[left[w]] = BLACK and color[right[w]] = BLACK
10                then color[w] ← RED                      ▷ Case 2
11                     x ← p[x]                            ▷ Case 2
12                else if color[right[w]] = BLACK
13                        then color[left[w]] ← BLACK       ▷ Case 3
14                             color[w] ← RED               ▷ Case 3
15                             RIGHT-ROTATE(T, w)           ▷ Case 3
16                             w ← right[p[x]]              ▷ Case 3
17                      color[w] ← color[p[x]]              ▷ Case 4
18                      color[p[x]] ← BLACK                 ▷ Case 4
19                      color[right[w]] ← BLACK             ▷ Case 4
20                      LEFT-ROTATE(T, p[x])                ▷ Case 4
21                      x ← root[T]                         ▷ Case 4
22        else (same as then clause with "right" and "left" exchanged)
23 color[x] ← BLACK
```

ok，很清楚，在此，就不贴第9张图了。

在下文的红黑树删除的4种情况，详细、具体分析了上段代码。

十、红黑树删除的4种情况

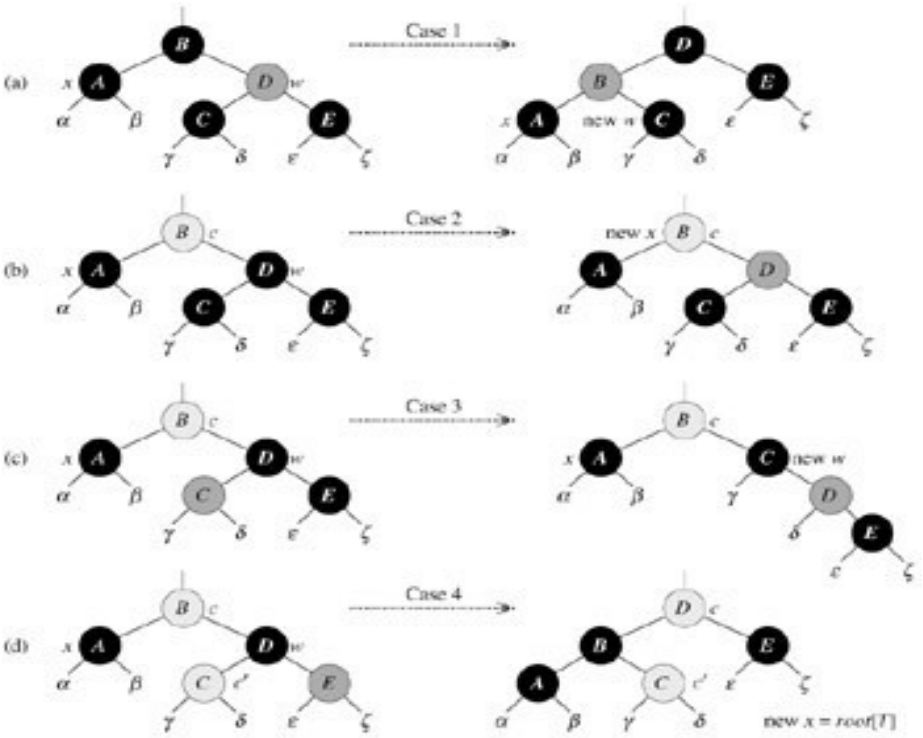
情况1：x的兄弟w是红色的。

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，w的左孩子是红色，w的右孩子是黑色。

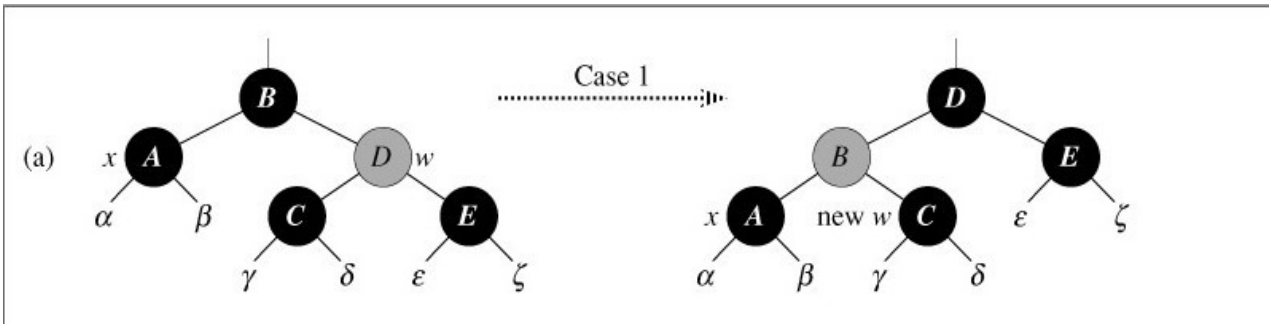
情况4：x的兄弟w是黑色的，且w的右孩子时红色的。

操作流程图：



ok，简单分析下，红黑树删除的4种情况：

针对情况1：x的兄弟w是红色的。



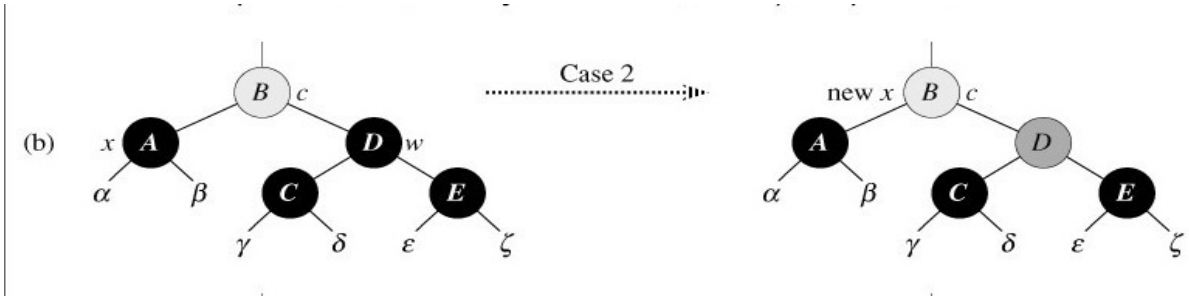

```
5        then color[w] ← BLACK                ▷ Case 1
6        color[p[x]] ← RED                    ▷ Case 1
7        LEFT-ROTATE(T, p[x])                 ▷ Case 1
8        w ← right[p[x]]                      ▷ Case 1
```

对策：改变w、p[z]颜色，再对p[x]做一次左旋，红黑性质得以继续保持。

x的新兄弟new w是旋转之前w的某个孩子，为黑色。

所以，情况1转化成情况2或3、4。

针对情况2：x的兄弟w是黑色的，且w的俩个孩子都是黑色的。



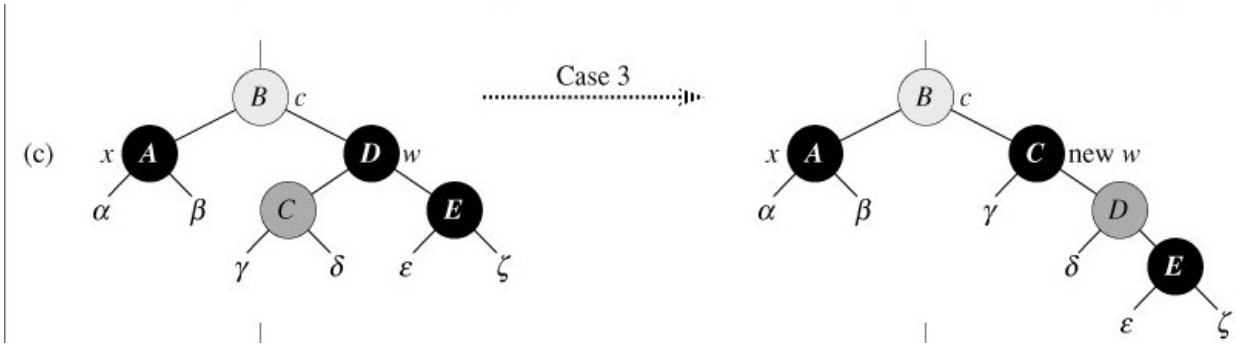

```
10          then color[w] ← RED                ▷ Case 2
11              x <-p[x]                       ▷ Case 2
```

如图所示，**w的俩个孩子都是黑色的**，

对策：因为w也是黑色的，所以x和w中得去掉一黑色，最后，w变为红。

p[x]为新结点x，赋给x，x<-p[x]。

针对情况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。




```
13          then color[left[w]] ← BLACK        ▷ Case 3
14              color[w] ← RED                 ▷ Case 3
15              RIGHT-ROTATE(T, w)             ▷ Case 3
16              w ← right[p[x]]                ▷ Case 3
```

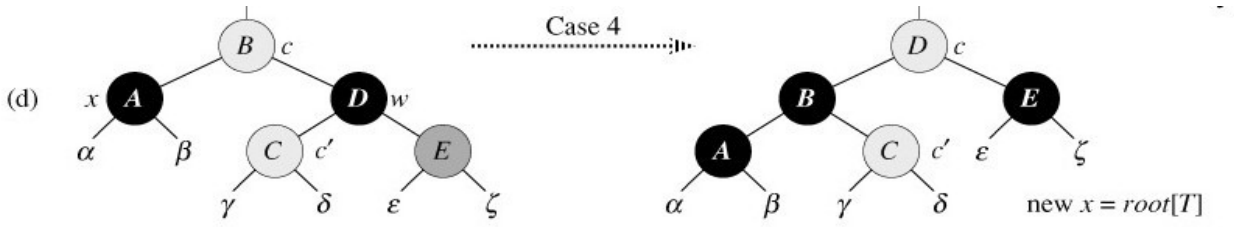
w为黑，其左孩子为红，右孩子为黑

对策：交换w和和其左孩子left[w]的颜色。 即上图的D、C颜色互换。:D。

并对w进行右旋，而红黑性质仍然得以保持。

现在x的新兄弟w是一个有红色右孩子的黑结点，于是将情况3转化为情况4.

针对情况4：x的兄弟w是黑色的，且w的右孩子时红色的。



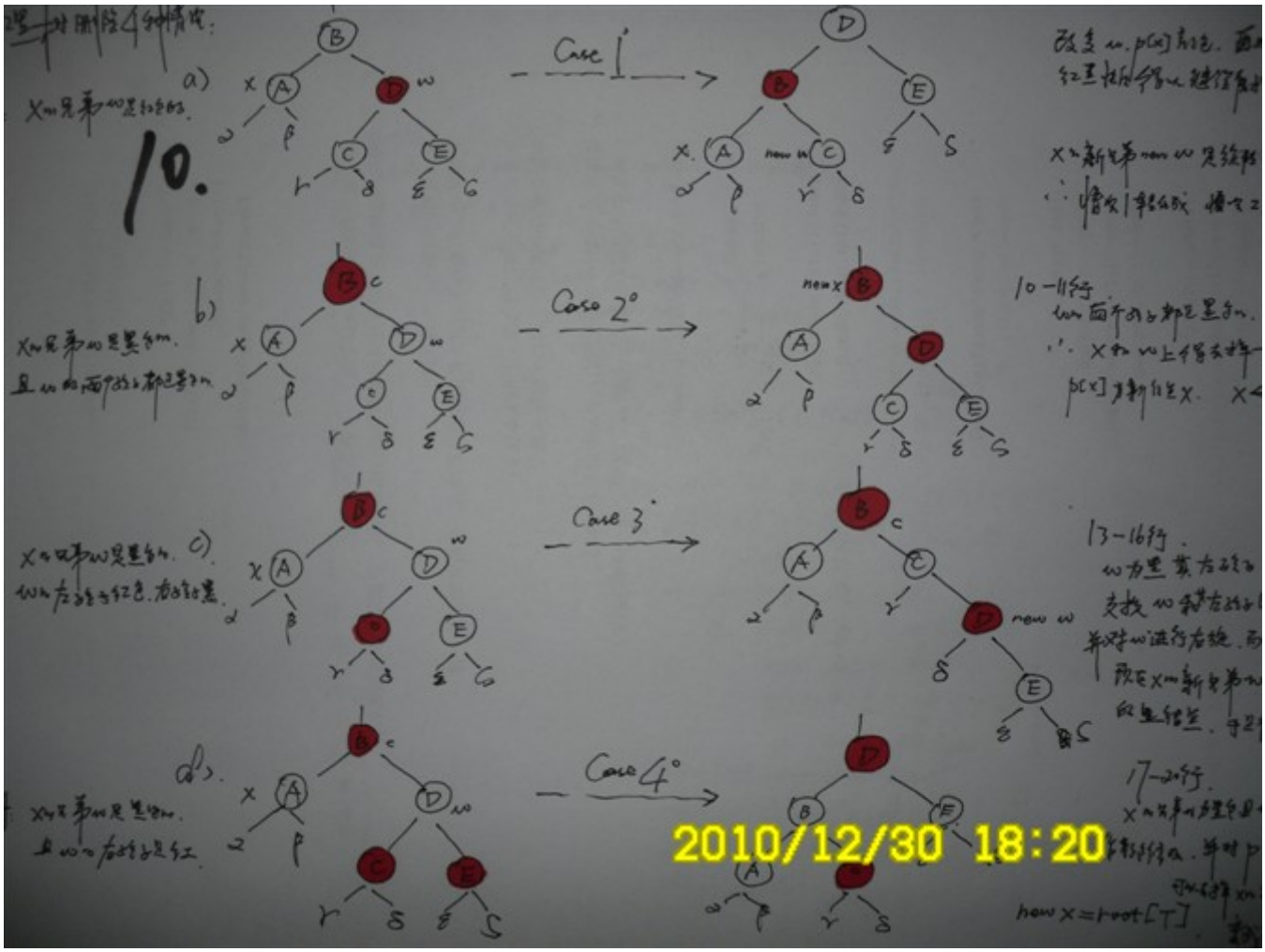

```
17      color[w] ← color[p[x]]          ▷ Case 4
18      color[p[x]] ← BLACK             ▷ Case 4
19      color[right[w]] ← BLACK         ▷ Case 4
20      LEFT-ROTATE(T, p[x])            ▷ Case 4
21      x ← root[T]                     ▷ Case 4
```

x的兄弟w为黑色，且w的右孩子为红色。

对策：做颜色修改，并对p[x]做一次旋转，可以去掉x的额外黑色，来把x变成单独的黑色，此举不破坏红黑性质。

将x置为根后，循环结束。

最后，贴上最后的第10张图：



ok，红黑树删除的4中情况，分析完成。

结语：只要牢牢抓住红黑树的5个性质不放，而不论是树的左旋还是右旋，

不论是红黑树的插入、还是删除，都只为了保持和修复红黑树的5个性质而已。

顺祝各位， 元旦快乐。完。

July、二零一零年十二月三十日。

扩展阅读：***Left-Leaning Red-Black Trees***, Dagstuhl Workshop on Data Structures, Wadern, Germany, February, 2008.

直接下载：http://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf

1、教你透彻了解红黑树

2、红黑树算法的实现与剖析

- 上一篇 网友答案整理II：微软等面试100题系列之网友精彩回复 [二]
- 下一篇 三、动态规划算法解最长公共子序列LCS问题（2011.12.13重写）

相关文章推荐

- 二叉搜索树之Java实现
- MVVM在美团点评酒旅移动端的最佳…
- 堆排序算法原理及JAVA实现
- C语言大型软件设计的面向对象--宋…
- 经典算法研究系列：五、红黑树算法…
- Retrofit 从入门封装到源码解析
- 经典算法研究系列：五、红黑树算法…
- 跳过Java开发的各种坑
- 经典算法研究系列：五、红黑树算法…
- Spring Boot 2小时入门基础教程
- 微软面试、经典算法、编程艺术、红…
- Shell脚本编程
- 微软面试、经典算法、编程艺术、红…
- 微软面试、经典算法、编程艺术、红…
- 六个经典算法研究：A*.Dijkstra.动态…
- 六个经典算法研究：A*.Dijkstra.动态…

查看评论

yuanhang926 71楼 2016-09-06 19:43发表

淋雨的男生 70楼 2015-11-03 20:55发表

xyz谁是谁啊，能不能用其他的名字，

xiaoqiaoxq 69楼 2015-09-07 15:49发表

“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。”

这条是不是说错了，我看着一直迷糊呢，你给的图里面也不能满足这条啊。
去查了英文描述才发现：Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes. The uniform number of black nodes in the paths from root to leaves is called the black-height of the red–black tree.

应该是【后裔NIL叶节点】，而不是什么“子孙节点”

AstrayLinux 68楼 2015-07-21 13:51发表

mark

qq_15898925 67楼 2014-07-03 13:16发表

第一个第五行的注释很容易误导，一直以为注释是解释那一行的。。。

星夜落尘 66楼 2014-04-01 15:10发表

for(int i=20;i>0;i-=2)
楼主，这样插入后，就会有两个红节点在一起，请问怎么破

胖虎 65楼 2014-03-30 19:28发表

没接触过红黑树，看了你的博文，还是没弄懂红黑树的原理，惭愧。我觉得最好先将红黑树的历史缘由，性质，然后怎么创建一颗红黑树，最好有个例子。最后再讲红黑树的左旋、右旋，插入删除等主题。

sheiok 64楼 2013-12-14 17:55发表

博主写得文章真的超级好，有个地方我觉得有点小问题
左旋代码实现：
5 p[y] <- p[x] //y成为x的父结点

这个地方应该是 x的父亲作为y的父亲？？？
不知道说得对不对?

删除的几种case，知道这几种旋转能达到平衡。
我所不能理解的是，是怎样得出的从case 1->case4渐进的过程，比

如说为什么case1要转换成case2是基于什么理论，以及推倒思路。
再比如为什么case4做完之后，x就等于root了？单看那4种转换，很好理解。

shelok Re: 2013-12-14 18:03发表

回复shelok：其实说，我更想知道的是思维导图，而不是结论。如果说前三种case还有将“额外黑色”上移的指导思想，但是case4，将x设置为root且没有给出半点推倒逻辑，我只能认为是其经验判断。不但想知道how，还想知道why

yaolihust Re: 2014-01-23 15:17发表

回复shelok：很有同感，其实这样去学习更像是一种记忆，记住了觉得自己会了，但是时间长了还是会忘记。总是觉得少了一些推演的过程。

轩轩昊昊 63楼 2013-12-11 20:27发表

shizhong, 我喜欢你在纸上自己画的东西，那样更能说明你理解的更透彻，我也是自己在纸上画，嘿嘿，深有同感！！！

62楼 2013-12-10 23:45发表

fengshengwei3 61楼 2013-08-25 19:29发表

针对delete-fix-up操作中的case1，也就是x的兄弟结点w如果是红色的话，感觉w的left结点可能为NULL？？？不知道我是不是哪儿的理解有偏差，因为结合红黑树的5条性质，我们最多能推断出w结点的left和right至少有一个存在，但不能就此断定left一定存在吧？还请楼主指点下:)

左旋代码的注释写错了把亲。。。那应该是让x的父节点成为y的父节点，可不是让“y成为x的父母”。。。这句话真坑人啊。。。

v_JULY_v Re: 2013-11-20 17:54发表

回复fengshengwei3：thanks，我修改下。

mrpeterchen 60楼 2013-07-23 11:54发表

再追问下，LZ用的什么源代码，看着怪别扭的
C代码有y<-p[z]这样的？是伪码？只是为了表明逻辑？

v_JULY_v Re: 2013-11-20 17:54发表

回复mrpeterchen：伪代码:-)

mrpeterchen 59楼 2013-07-23 11:52发表

yingjt722 58楼 2013-04-23 22:20发表

博主，小弟菜求教2个问题。。。

1. 插入调整代码里，p[z]是个什么东东，z又是什么?
2. 什么时候用到左旋右旋？什么时候只需要改颜色就可以了?

duqi_2009 57楼 2012-11-28 10:20发表

左旋第二张图 5：应该是//将x的父亲变为y的父亲，并没有建立y为x父亲这一关系

GuominVicky 56楼 2012-11-25 21:37发表

5 p[y] <- p[x] //y成为x的父母
这一行，应该是把x的父母成为y的父母吧；
而使y成为x的父母的实现应该是：12 p[x] ← y
这一行吧。

我问下，删除情况二，如果B是红色的，那么最后把x赋为B，那么x的右孩子W现在是红色，不就不满足红黑树性质了么~~~

cschmin Re: 2012-12-25 16:02发表

回复GuominVicky：如果B是红色的，把B赋给x后，x的颜色就是红色，那再次进入循环的时候就不符合 while x ≠ root[T] and color[x] = BLACK中的color[x]=BLACK条件了，循环就终止了，再看RD-TREE-FIXED(T, x)的23行：color[x] ← BLACK 循环终止后执行这行代码把B涂成黑色了

SupremeHover 55楼 2012-10-21 16:34发表

我问下 删除弥补部分 为什么一开始就能假定w有两个儿子，而不是说只有一个儿子呢甚至没有儿子呢?

v_JULY_v Re: 2012-10-22 10:59发表

回复SupremeHover：方便分析罢了,实际中具体情况当然不止如此:-)

SupremeHover 54楼 2012-10-20 17:54发表

v_JULY_v 53楼 2012-10-19 18:02发表

因为p[z]都为黑色，所以不违反红黑性质
这句话没看太明白哟~他们在任何时候都没同为黑色吧~
你直接搜索语句的部分文字就可以定位它们在哪里，快捷键是Ctrl+F。

回复SupremeHover：能否告诉我在文章中具体哪一处，哪一节?

SupremeHover Re: 2012-10-20 17:40发表

回复v_JULY_v：六、红黑树插入的第一种情况（RB-INSERT-FIXUP(T, z)代码的具体分析一）
你直接搜索我的部分文字就找到它在哪里了噻。不管是chrome还是IE内核的浏览器，搜索快捷键都是Ctrl+F。

SupremeHover 52楼 2012-10-19 16:02发表

最后，贴出昨下午关于此右旋算法所画的图：

左旋（第2张图）：

第一行应该改成“左旋”吧，又是笔误咯~

v_JULY_v Re: 2012-10-19 16:25发表

回复SupremeHover：EN，是的，我改正下:-)

wdxz6547 51楼 2012-10-05 01:12发表

看完了你的关于红黑树的文章，关于红黑树好像没有看到插入和删除的时候，为什么有那几种情况的原因，如果能分析原因，我想真正理解了红黑数。
还有就是本应该先讲二叉排序树，平衡树、再讲AVL树、2－3－4树和红黑树的。这个是一个逐层深入了，难度逐渐上升，而且貌似历史也是这样的发展的。如果lz再能考证下历史，研究下各个树的脉络，及应用，那么绝对称得上高质量。

v_JULY_v Re: 2012-10-05 08:12发表

回复wdxz6547：是的，明白

wdxz6547 50楼 2012-10-04 23:47发表

看了楼主的很多文章，精神可嘉，应该考虑对整个博客的重整了，是你真正进入高手的关键一步啦。

v_JULY_v Re: 2012-10-05 08:09发表

回复wdxz6547：于我心有戚戚焉，然毕离校后不再有大段大段时间.
只能待两三年后了

stecdeng 49楼 2012-07-04 16:38发表

fangtao314609005 48楼 2012-05-14 16:34发表

左旋示例代码中
5 p[y] <- p[x] //y成为x的父母
这个应该是X的父节点复制到Y的父节点记录中

12句的代码
12 p[x] ← y
这个才能称为 Y称为X的父节点

傅里叶变变 47楼 2012-05-10 17:15发表

为什么不考虑被删结点的子结点为红色的情况？当被删除的结点是黑色结点，且该节点的父亲是红色结点，该节点删除后，被删除节点的子树会接到被删除节点的父节点上，如果连结到父亲结点的子树也是红色结点的时候，就破坏了性质四，请问对应哪种删除情况呢？

weiyier 46楼 2012-04-16 08:46发表

楼主 有一点请赐教，就是插入操作的情况一。为什么一定要把z的祖父节点涂成黑色以维护性质五。就算是z的祖父节点是红色 会改变性质五吗 请楼主赐教呀

你好，我问一下，针对删除中去掉x额外的黑色，变成单独的黑色，怎么理解啊，求教

v_JULY_v Re: 2012-04-16 09:33发表

回复weiyidemaomao：具体哪一节？

weiyier Re: 2012-04-16 17:58发表

回复v_JULY_v：对策：做颜色修改，并对p[x]做一次旋转，可以去掉x的额外黑色，来把x变成单独的黑色，此举不破坏红黑性质。

将x置为根后，循环结束。

SVKING 45楼 2012-03-14 14:06发表

重构删除而导致有一些路径的黑节点少了一个的时候，解决方法可以简单的理解为两类：①对这些路径增加一个黑节点（情况4的解决方法）②将其他所有的路径的黑节点减少一个（情况2，上升到根节点结束的请况）

44楼 2011-12-27 15:13发表

eaglesky1990 43楼 2011-12-16 21:12发表

问下16楼《C算法》的全名就是《C算法》，还是叫什么名字？我好像没找到，麻烦告知下，作者和出版社，谢谢哦！

赞此文！但我觉得应该指出这些操作的动机，否则仍不好懂……表面的解释看看书就知道了，但理解操作可不易啊！希望楼主能解释得在深刻一些！

emeraldttt

Re: 2012-03-02 19:22发表

回复eaglesky1990：其实就是使得树结构不退化，当大量操作时如果简单使用二叉查找树会使得树退化，这样树高大大多于logN，当需要保存大量数据并查找，插入，删除时就要求使用AVL树或红黑树，但是AVL是完全平衡的要求变动的数据更多。map类就是使用红黑树实现的。

v_JULY_v

Re: 2012-03-03 01:02发表

回复emeraldttt：赞

Never_for_Never

42楼 2011-11-26 22:05发表

楼主的思路很清晰，赞一个！

可是为什么？我不明白为什么？为什么要那样操作？

就说那插入时吧的一种情况：

当前节点的父节点和叔叔节点是红的；

我想的是把父节点和叔叔节点都变成黑色，不就行了？

为什么还要把祖节点（黑）变成红色呢？

然后再以祖节点为起点在检索？？

求指教！！！！！！！！！！！！！！！！！！！！！

bessie2525

41楼 2011-08-11 18:36发表

推荐一下维基百科对红黑树的讲解：

http://zh.wikipedia.org/wiki/%E7%BA%A2%E9%BB%91%E6%A0%91

感觉上更清晰一些~ :)

v_JULY_v

Re: 2011-08-11 18:46发表

回复bessie2525：呵呵。

zszjian

40楼 2011-08-09 15:01发表

请问博主，在第六大点上的11~12之间是否需要一个else if z = left[p[z]]????就是说，插入的新节点处于这么一种情况：父节点是红色，叔节点是黑色。这个时候应该进入CASE 3，将父节点涂黑，祖父节点涂宏，然后右旋转祖父节点？？

zszjian

Re: 2011-08-09 15:02发表

回复zszjian：且自己为左子

wangque145

39楼 2011-06-19 23:50发表

你能否先证明一下，为什么满足上面5个条件它就是平衡树？

fdl

38楼 2011-06-13 08:44发表

其实我更想知道如何检验第五条性质.

SVKING Re: 2012-03-14 14:08发表

回复fdisksys：不需要验证吧，空树的时候是满足条件的，然后一直保持这些条件

yao050421103 37楼 2011-06-09 14:32发表

楼主好人！v 极品文章，对俺看懂STL的map大有帮助！期待楼主再接再厉！[e03][e01][e04]

v_JULY_v 36楼 2011-04-06 12:31发表

v_JULY_v 35楼 2011-04-06 08:48发表

to dazhi316：是的，那RB-DELETE(T, z)中z是指要删除的元素。y，是一个临时结点变量。而x，是后来要在RB-DELETE-FIXUP(T, x)中调整的结点。

to dazhi316（3）：而第7行中的无条件赋值则保证了无论x是有关键字的内结点或哨兵nil[T]，x现在的父结点都为先前y的父结点。

dazhi316 Re: 2011-04-06 09:35发表

回复 v_JULY_v：那RB-DELETE(T, z)中z是要删除的元素吗？和y又是什么关系呢?

kylidboy Re: 2011-05-31 13:05发表

回复 dazhi316：z是原意要删除的节点，
因为二叉树的删除性质，所以直接找到最简单后继节点，后继节点一定是可以在z的位置上取代z，所以就变成了删除z的后继节点也就是y，不过如果z只有单子树的时候，y就等于z的，

v_JULY_v 34楼 2011-04-06 08:48发表

v_JULY_v 33楼 2011-04-06 08:47发表

to dazhi316（2）：所以，第17行中，传递给RB-DELETE-FIXUP(T, x)的结点x有俩种情况：1、在y被删除之前，如果y有个不是哨兵nil[T]的孩子，则x为y的为孩子；2、如果y没有孩子，则x为哨兵nil[T]。

dazhi316 32楼 2011-04-06 07:46发表

to dazhi316（1）：你首先得具体看：八、接下来，进入红黑树的删除部分中，第17行是怎么出现x的：then RB-DELETE-FIXUP(T, x) // 则调用RB-DELETE-FIXUP(T, x)。

v_JULY_v  31楼 2011-03-28 16:08发表

博主好，很喜欢你的博客，感觉受益匪浅啊！最近刚接触红黑树，有一点不明白的是删除节点的时候，那个x节点是哪一个呢?

to liexusong：ok，没问题。算法导论上，很多细节和具体的删除情况，并没有提及。它上面有的只是一些具体典型的3种插入情况，所以，你的对算法导论上有困惑有依据的。:D。

 liexusong Re: 2011-03-28 16:18发表

回复 v_JULY_v：好,谢谢

 v_JULY_v 30楼 2011-03-28 15:57发表

to liexusong，稍后，我将写一篇文章，题为“红黑树插入和删除全过程的演示图”，到时，您一看，就一目了然了，：D。

 liexusong Re: 2011-03-28 16:01发表

回复 v_JULY_v：好的, 暂时还不知道怎么做, 最好就是把我说的那种情况写清楚一点, 谢谢了. 还有, 算法导论是不是没有提到我说的那种情况啊, 有的话在哪里呢?

 liexusong 29楼 2011-03-28 14:45发表

LZ, 我想问一下, 如果删除的节点没有子节点, 并且颜色为黑色的时候怎么处理呢?

 michaelliudl Re: 2011-06-17 13:46发表

回复 liexusong：算法导论上所有叶子内节点的子节点都是NIL[T]，所以这种情况就是x=NIL[T]

 v_JULY_v Re: 2011-03-28 15:06发表

回复 liexusong：直接删除，同时，其兄弟结点着为红色。

 v_JULY_v Re: 2011-03-28 16:25发表

回复 v_JULY_v：我又细想了下。觉得这么说，只是一种情况而已。有失妥当。ok，一切，以稍后的文章，为准。

 liexusong Re: 2011-03-28 15:45发表

回复 v_JULY_v：还有算法导论都没错处理到这种情况
~所以想知道怎么处理

liexusong Re: 2011-03-28 15:42发表

回复 v_JULY_v：那不是不符合红黑树的性质?

v_JULY_v Re: 2011-03-28 15:57发表

回复 liexusong：符合的。

engrossment 28楼 2011-03-15 23:54发表

感觉对第五点性质的描述不是很好喔，让读者不好理解。觉得这样说会好点：从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。[e04]

v_JULY_v Re: 2011-03-16 07:20发表

回复 engrossment：恩，非常好的建议。谢谢你。[e04]

xiqiubo1985 27楼 2011-03-04 15:56发表

删除操作不太懂，我觉得应该将删除前是一个什么样子，给画出来。然后删除后是一个什么样子。如果删除后违反了那5个性质，那么还要将修复后的图给弄出来。。

v_JULY_v Re: 2011-03-06 19:59发表

回复 xiqiubo1985：建议，非常好。[e03]。

工程师WWW 26楼 2011-03-03 13:07发表

哥啊，你拍这图片怎么看啊？还不如用画图工具画。

v_JULY_v Re: 2011-03-06 19:59发表

回复 weiqubo：是。感谢批评。也可以不用看我画的图的。我的图只是一个补充而已。

v_JULY_v Re: 2011-03-06 19:58发表

回复 weiqubo：是的。感谢批评。

potty15 25楼 2011-02-23 08:27发表

potty15 哦，明白了，呵呵，原来删除的是黑色，所以现在有2个黑色节点还是平衡的。 24楼 2011-02-21 20:08发表

zhijializhangcao 23楼 2011-02-16 16:09发表

不错，只是有个问题，就是在删除后fixup的第四个情况：
x的兄弟w为黑色，且w的右孩子为红色，
按case4调整后，原来的X即最后图里的A，以及B，E都是黑色，这样B-A路径的黑色不就比其他的多一个？
此时X跑到根去了，fixup就结束了？

冀博 22楼 2011-02-15 12:28发表

哥，你好强[e01]

v_JULY_v 21楼 2011-02-14 18:01发表

好文章[e01]

我所写的这个红黑树系列文章，做的只是表面功夫。想彻底深究内部原理的同志，还是请去参考：Leo J. Guibas 和 Robert Sedgewick 于1978年写的关于红黑树的一篇论文。

---

v_JULY_v Re: 2011-02-14 18:02发表

回复 v_JULY_v：当然，我自己个人会抽空仔细研读此篇论文，然后再写第五篇红黑树系列的文章。

kaly0204 20楼 2011-01-11 15:27发表

kaly0204 确实很牛，不过我现在还没用过红黑树，说不定以后会用上，先看看，了解一下 19楼 2011-01-11 13:50发表

ylvlike 18楼 2011-01-10 23:16发表

[e01]不错，相当详细！

chjttony 17楼 2011-01-10 16:59发表

[e03][e03][e03][e03][e03]

Yelena_Lee 16楼 2011-01-08 14:03发表

学习了，收藏

leisongsong1987 15楼 2011-01-07 17:02发表

[e03][e03][e03][e01]

skyowner2009 14楼 2011-01-07 14:17发表

[e01]linux kernel现在的调度算法就是用的红黑树

建议看下《C算法》
直接提到了红黑书基本上就是2-3-4树的升级版，简明通俗易懂。

毫无疑问，算法导论对于红黑树的讨论势比较晦涩的，没有说明白红黑树的目的以及源头，直接上复杂变换，十分不易理解。

v_JULY_v Re: 2011-01-07 15:26发表

回复 hyde963：恩，谢谢，我一定看下。

czk740960212 13楼 2011-01-06 22:48发表

jyt 12楼 2011-01-06 21:56发表

顶楼主一个····[e01]

红烧大肠 11楼 2011-01-06 15:36发表

[e01]学习了~~~~~厉害

什么时候能有人讲讲R树就好了

---

nash635 Re: 2011-01-06 21:03发表

回复 wanruirui：搞数据挖掘的?

v_JULY_v Re: 2011-01-06 18:45发表

回复 wanruirui：我来讲罗，呵呵~

红烧大肠 Re: 2011-01-07 15:32发表

回复 v_JULY_v：期待中。

v_JULY_v Re: 2011-12-27 16:17发表

回复wanruirui：已经讲好了：从B树、B+树。B*树谈到R树。

酱油棍 10楼 2011-01-05 23:35发表

这么学习红黑树感觉还是挺让人迷糊的。就那5条性质，要融会贯通，够费脑筋的，个把月之后可能又忘了。红色，黑色，节点数目等概念本身对人脑来说，不太直观。建议从2-3-4树的概念入手，弄明白红黑两色究竟啥意思之后，红黑树就没啥好说的了。

v_JULY_v Re: 2011-01-06 12:20发表

回复 whyel：谢谢提议，我看下2-3-4树。

kk520dd 9楼 2011-01-05 21:15发表

不知道什么意思[e03][e03][e03][e03]

8楼 2011-01-05 19:58发表

v_JULY_v 7楼 2011-01-03 14:47发表

学习ING

mide_c 6楼 2011-01-03 14:17发表

文章，一经发现 bug，立马修正。永久勘误。

博主不是一般的牛！！[e03]

v_JULY_v Re: 2011-01-03 14:39发表

回复 mide_c：兄台过分抬举了。文章 刚刚修正了一些小bug以及对部分内容做了调整。有误、或写的不够透彻之处，还请继续指出。

v_JULY_v 5楼 2011-01-01 22:57发表

v_JULY_v

RB-INSERT-FIXUP(T, z)，和RB-DELETE-FIXUP(T, x) 是分别针对在插入和删除时，因可能违反红黑树的规则，而做的调整工作。

4楼 2011-01-01 20:59发表

v_JULY_v 3楼 2011-01-01 20:58发表

此文，关于红黑树删除的4种情况与第一篇文章红黑树删除的情况3、4、5、6是完全相对应的。因为第一篇文章中的情况1、2并未破坏红黑性质，所以，这里，省略了第一篇的情况1、2。

p569354158 2楼 2011-01-01 19:29发表

此文，关于红黑树删除的4种情况与第一篇文章的情况3、4、5、6是完全相对应的。因为第一篇文章中的情况1、2并未破坏红黑性质，所以，这里，省略了第一篇的情况1、2。

在看STL源码剖析的时候才接触到红黑树，现在发现好多东西都不会

---

v_JULY_v Re: 2011-01-01 20:59发表

回复 p569354158：呵呵~我是看了 算法导论，才注意到红黑树的。

v_JULY_v 1楼 2010-12-31 12:05发表

注，上述关于红黑树的插入、删除情况的分析，都只是针对树的结点插入、或删除后，对红黑树进行的恢复与保持操作。日后，给出一个红黑树的结点插入、删除的示例。

---

v_JULY_v Re: 2011-01-07 19:26发表

回复 v_JULY_v：突然，感觉，自己手工画的那几幅图，丑死拉。改日，有空，再重新好好画下。

QQIANQQ Re: 2011-01-10 17:57发表

回复 v_JULY_v：楼主其实你可以用visio画的，简单而且方便。
还有删除和插入操作。你列举的那些情况。其实都可以用一个流程图表示出来，那样看起来也方便的多，理解起来也很方便。其实我就是通过帮流程图画出来来理解的。

---

v_JULY_v Re: 2011-01-10 18:00发表

回复 QQIANQQ：恩，非常感谢您的提议。谢谢。下次，我试试~

---

v_JULY_v Re: 2010-12-31 12:39发表

回复 v_JULY_v：区别俩种操作：

1.红黑树插入、删除结点的操作

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树
的操作。