

红黑树C源码实现与剖析

作者：July、那谁 时间：二零一一年一月三日

前言：

红黑树作为一种经典而高级的数据结构，相信，已经被不少人实现过，但不是因为程序不够完善而无法运行，就是因为程序完全没有注释，初学者根本就看不懂。

此份红黑树的c源码最初从linux-lib-rbtree.c而来，后经一网友那谁(<http://www.cppblog.com/converse/>)用c写了出来。在此，向原作者表示敬意。

考虑到原来的程序没有注释，所以我特把这份源码放到编译器里，一行一行的完善，一行一行的给它添加注释，至此，红黑树c带注释的源码，就摆在了您眼前，有不妥、不正之处，还望不吝指正。

红黑树的六篇文章：

- 1、教你透彻了解红黑树
- 2、红黑树算法的实现与剖析
- 3、红黑树的c源码实现与剖析
- 4、一步一图一代码，R-B Tree
- 5、红黑树插入和删除结点的全程演示
- 6、红黑树的c++完整实现源码

ok，咱们开始吧。

相信，经过我前俩篇博文对红黑树的介绍，你应该对红黑树有了透彻的理解了（没看过的朋友，可事先查上面的俩篇文章，或与此文的源码剖析对应着看）。

本套源码剖析把重点放在红黑树的3种插入情况，与红黑树的4种删除情况。其余的能从略则尽量简略。

目录：

一、左旋代码分析

二、右旋

三、红黑树查找结点

四、红黑树的插入

五、红黑树的3种插入情况

六、红黑树的删除

七、红黑树的4种删除情况

八、测试用例

好的，咱们还是先从树的左旋、右旋代码，开始（大部分分析，直接给注释）：

```
//一、左旋代码分析
/*-----
|      node          right
|      / /    ==>   / /
|      a  right    node  y
|      / /        / /
|      b  y      a  b    //左旋
|-----*/
static rb_node_t* rb_rotate_left(rb_node_t* node, rb_node_t* root)
{
    rb_node_t* right = node->right;    //指定指针指向 right<--node->right

    if ((node->right = right->left))
    {
        right->left->parent = node;    //好比上面的注释图，node成为b的父母
    }
    right->left = node;    //node成为right的左孩子

    if ((right->parent = node->parent))
    {
        if (node == node->parent->right)
        {
            node->parent->right = right;
        }
        else
        {
            node->parent->left = right;
        }
    }
    else
    {
        root = right;
    }
    node->parent = right;    //right成为node的父母

    return root;
}

//二、右旋
/*-----
|      node          left
|      / /          / /
|      left  y    ==>  a  node
|      / /          / /
|      a  b          b  y    //右旋与左旋差不多，分析略过
|-----*/
```

```

-----*/
static rb_node_t* rb_rotate_right(rb_node_t* node, rb_node_t* root)
{
    rb_node_t* left = node->left;

    if ((node->left = left->right))
    {
        left->right->parent = node;
    }
    left->right = node;

    if ((left->parent = node->parent))
    {
        if (node == node->parent->right)
        {
            node->parent->right = left;
        }
        else
        {
            node->parent->left = left;
        }
    }
    else
    {
        root = left;
    }
    node->parent = left;

    return root;
}

//三、红黑树查找结点
//-----
//rb_search_auxiliary: 查找
//rb_node_t* rb_search: 返回找到的结点
//-----
static rb_node_t* rb_search_auxiliary(key_t key, rb_node_t* root, rb_node_t** save)
{
    rb_node_t *node = root, *parent = NULL;
    int ret;

    while (node)
    {
        parent = node;
        ret = node->key - key;
        if (0 < ret)
        {
            node = node->left;
        }
        else if (0 > ret)
        {
            node = node->right;
        }
        else
        {
            return node;
        }
    }

    if (save)
    {
        *save = parent;
    }

    return NULL;
}

//返回上述rb_search_auxiliary查找结果
rb_node_t* rb_search(key_t key, rb_node_t* root)
{
    return rb_search_auxiliary(key, root, NULL);
}

```

```

//四、红黑树的插入
//-----
//红黑树的插入结点
rb_node_t* rb_insert(key_t key, data_t data, rb_node_t* root)
{
    rb_node_t *parent = NULL, *node;

    parent = NULL;
    if ((node = rb_search_auxiliary(key, root, &parent))) //调用rb_search_auxiliary找到插入结
    点的地方
    {
        return root;
    }

    node = rb_new_node(key, data); //分配结点
    node->parent = parent;
    node->left = node->right = NULL;
    node->color = RED;

    if (parent)
    {
        if (parent->key > key)
        {
            parent->left = node;
        }
        else
        {
            parent->right = node;
        }
    }
    else
    {
        root = node;
    }

    return rb_insert_rebalance(node, root); //插入结点后, 调用rb_insert_rebalance修复红黑
    树的
    性质
}

```

```

//五、红黑树的3种插入情况
//接下来, 咱们重点分析针对红黑树插入的3种情况, 而进行的修复工作。
//-----
//红黑树修复插入的3种情况
//为了在下面的注释中表示方便, 也为了让下述代码与我的俩篇文章相对应,
//用z表示当前结点, p[z]表示父母、p[p[z]]表示祖父、y表示叔叔。
//-----
static rb_node_t* rb_insert_rebalance(rb_node_t *node, rb_node_t *root)
{
    rb_node_t *parent, *gparent, *uncle, *tmp; //父母p[z]、祖父p[p[z]]、叔叔y、临时结点
    *tmp

    while ((parent = node->parent) && parent->color == RED)
    {
        //parent 为node的父母, 且当父母的颜色为红时
        gparent = parent->parent; //gparent为祖父

        if (parent == gparent->left) //当祖父的左孩子即为父母时。
            //其实上述几行语句, 无非就是理顺孩子、父母、祖父的关系。:D。
        {
            uncle = gparent->right; //定义叔叔的概念, 叔叔y就是父母的右孩子。

            if (uncle && uncle->color == RED) //情况1: z的叔叔y是红色的
            {
                uncle->color = BLACK; //将叔叔结点y着为黑色
                parent->color = BLACK; //z的父母p[z]也着为黑色。解决z, p[z]都是红色的问题。
                gparent->color = RED;
                node = gparent; //将祖父当做新增结点z, 指针z上移两层, 且着为红色。
                //上述情况1中, 只考虑了z作为父母的右孩子的情况。
            }
            else //情况2: z的叔叔y是黑色的,
            {

```

```

    if (parent->right == node) //且z为右孩子
    {
        root = rb_rotate_left(parent, root); //左旋[结点z, 与父母结点]
        tmp = parent;
        parent = node;
        node = tmp; //parent与node 互换角色
    }
    //情况3: z的叔叔y是黑色的, 此时z成为了左孩子。
    //注意, 1: 情况3是由上述情况2变化而来的。
    //.....2: z的叔叔总是黑色的, 否则就是情况1了。
    parent->color = BLACK; //z的父母p[z]着为黑色
    gparent->color = RED; //原祖父结点着为红色
    root = rb_rotate_right(gparent, root); //右旋[结点z, 与祖父结点]
}
}

else
{
    // if (parent == gparent->right) 当祖父的右孩子即为父母时。(解释请看本文评论下第23楼, 同
    时, 感谢SupremeHover指正!)
    uncle = gparent->left; //祖父的左孩子作为叔叔结点。[原理还是与上部分一样的]
    if (uncle && uncle->color == RED) //情况1: z的叔叔y是红色的
    {
        uncle->color = BLACK;
        parent->color = BLACK;
        gparent->color = RED;
        node = gparent; //同上。
    }
    else //情况2: z的叔叔y是黑色的,
    {
        if (parent->left == node) //且z为左孩子
        {
            root = rb_rotate_right(parent, root); //以结点parent、root右旋
            tmp = parent;
            parent = node;
            node = tmp; //parent与node 互换角色
        }
        //经过情况2的变化, 成为了情况3.
        parent->color = BLACK;
        gparent->color = RED;
        root = rb_rotate_left(gparent, root); //以结点gparent和root左旋
    }
}
}

root->color = BLACK; //根结点, 不论怎样, 都得置为黑色。
return root; //返回根结点。
}

```

//六、红黑树的删除

//-----

//红黑树的删除结点

rb_node_t* rb_erase(key_t key, rb_node_t *root)

```

{
    rb_node_t *child, *parent, *old, *left, *node;
    color_t color;

```

if (!(node = rb_search_auxiliary(key, root, NULL))) //调用rb_search_auxiliary查找要删

除的
结点

```

{
    printf("key %d is not exist!\n");
    return root;
}

old = node;

if (node->left && node->right)
{
    node = node->right;
    while ((left = node->left) != NULL)
    {
        node = left;
    }
}

```

```
    }
    child = node->right;
    parent = node->parent;
    color = node->color;

    if (child)
    {
        child->parent = parent;
    }
    if (parent)
    {
        if (parent->left == node)
        {
            parent->left = child;
        }
        else
        {
            parent->right = child;
        }
    }
    else
    {
        root = child;
    }

    if (node->parent == old)
    {
        parent = node;
    }

    node->parent = old->parent;
    node->color = old->color;
    node->right = old->right;
    node->left = old->left;

    if (old->parent)
    {
        if (old->parent->left == old)
        {
            old->parent->left = node;
        }
        else
        {
            old->parent->right = node;
        }
    }
    else
    {
        root = node;
    }

    old->left->parent = node;
    if (old->right)
    {
        old->right->parent = node;
    }
}
else
{
    if (!node->left)
    {
        child = node->right;
    }
    else if (!node->right)
    {
        child = node->left;
    }
    parent = node->parent;
    color = node->color;

    if (child)
    {
        child->parent = parent;
    }
    if (parent)
```

```

    {
        if (parent->left == node)
        {
            parent->left = child;
        }
        else
        {
            parent->right = child;
        }
    }
    else
    {
        root = child;
    }
}

free(old);

if (color == BLACK)
{
    root = rb_erase_rebalance(child, parent, root); //调用rb_erase_rebalance来恢复红黑
    树性
    质
}

return root;
}

```

//七、红黑树的4种删除情况

//-----

//红黑树修复删除的4种情况

//为了表示下述注释的方便，也为了让下述代码与我的俩篇文章相对应，

//x表示要删除的结点，*other、w表示兄弟结点，

//-----

```

static rb_node_t* rb_erase_rebalance(rb_node_t *node, rb_node_t *parent, rb_node_t *root)
{

```

```

    rb_node_t *other, *o_left, *o_right;    //x的兄弟*other，兄弟左孩子*o_left,*o_right

```

```

    while ((!node || node->color == BLACK) && node != root)
    {

```

```

        if (parent->left == node)
        {

```

```

            other = parent->right;

```

```

            if (other->color == RED)    //情况1: x的兄弟w是红色的
            {

```

```

                other->color = BLACK;

```

```

                parent->color = RED;    //上俩行，改变颜色，w->黑、p[x]->红。

```

```

                root = rb_rotate_left(parent, root);    //再对p[x]做一次左旋

```

```

                other = parent->right;    //x的新兄弟new w 是旋转之前w的某个孩子。其实就是左旋

```

后

的效果。

```

            }
            if ((!other->left || other->left->color == BLACK) &&

```

```

                (!other->right || other->right->color == BLACK))

```

```

                //情况2: x的兄弟w是黑色，且w的两个孩子也

```

都是黑色的

```

            {
                //由于w和w的两个孩子都是黑色的，则在x和w上得去掉一黑色，

```

```

                other->color = RED;    //于是，兄弟w变为红色。

```

```

                node = parent;    //p[x]为新结点x

```

```

                parent = node->parent;    //x<-p[x]
            }

```

```

            else

```

```

                //情况3: x的兄弟w是黑色的，

```

```

                //且，w的左孩子是红色，右孩子为黑色。

```

```

            if (!other->right || other->right->color == BLACK)
            {

```

```

                if ((o_left = other->left))    //w和其左孩子left[w]，颜色交换。
                {

```

```

                    o_left->color = BLACK;    //w的左孩子变为由黑->红色
                }

```

```

                other->color = RED;    //w由黑->红

```

复。
点
w。

```
root = rb_rotate_right(other, root); //再对w进行右旋，从而红黑性质恢
other = parent->right; //变化后的，父结点的右孩子，作为新的兄弟结
```

```
}
//情况4: x的兄弟w是黑色的
other->color = parent->color; //把兄弟节点染成当前节点父节点的颜色。
parent->color = BLACK; //把当前节点父节点染成黑色
if (other->right) //且w的右孩子是红
{
    other->right->color = BLACK; //兄弟节点w右孩子染成黑色
}
root = rb_rotate_left(parent, root); //并再做一次左旋
node = root; //并把x置为根。
break;
}
```

//下述情况与上述情况，原理一致。分析略。

```
else
{
    other = parent->left;
    if (other->color == RED)
    {
        other->color = BLACK;
        parent->color = RED;
        root = rb_rotate_right(parent, root);
        other = parent->left;
    }
    if ((!other->left || other->left->color == BLACK) &&
        (!other->right || other->right->color == BLACK))
    {
        other->color = RED;
        node = parent;
        parent = node->parent;
    }
    else
    {
        if (!other->left || other->left->color == BLACK)
        {
            if ((o_right = other->right))
            {
                o_right->color = BLACK;
            }
            other->color = RED;
            root = rb_rotate_left(other, root);
            other = parent->left;
        }
        other->color = parent->color;
        parent->color = BLACK;
        if (other->left)
        {
            other->left->color = BLACK;
        }
        root = rb_rotate_right(parent, root);
        node = root;
        break;
    }
}
```

```
if (node)
{
    node->color = BLACK; //最后将node[上述步骤置为了根结点]，改为黑色。
}
return root; //返回root
}
```

//八、测试用例
//主函数
int main()
{


```
int i, count = 100;
key_t key;
rb_node_t* root = NULL, *node = NULL;

srand(time(NULL));
for (i = 1; i < count; ++i)
{
    key = rand() % count;
    if ((root = rb_insert(key, i, root)))
    {
        printf("[i = %d] insert key %d success!\n", i, key);
    }
    else
    {
        printf("[i = %d] insert key %d error!\n", i, key);
        exit(-1);
    }

    if ((node = rb_search(key, root)))
    {
        printf("[i = %d] search key %d success!\n", i, key);
    }
    else
    {
        printf("[i = %d] search key %d error!\n", i, key);
        exit(-1);
    }
    if (!(i % 10))
    {
        if ((root = rb_erase(key, root)))
        {
            printf("[i = %d] erase key %d success!\n", i, key);
        }
        else
        {
            printf("[i = %d] erase key %d error!\n", i, key);
        }
    }
}

return 0;
}
```

ok, 完。

后记:

一、欢迎任何人就此份源码, 以及我的前述俩篇文章, 进行讨论、提议。

但任何人, 引用此份源码剖析, 必须得注明作者本人July以及出处。

红黑树系列, 已经写了三篇文章, 相信, 教你透彻了解红黑树的目的, 应该达到了。

二、本文完整源码, 请到处下载:

<http://download.csdn.net/source/2958890>

- 1、教你透彻了解红黑树
- 2、红黑树算法的实现与剖析
- 3、红黑树的c源码实现与剖析
- 4、一步一图一代码, R-B Tree

5、红黑树插入和删除结点的全程演示

6、红黑树的c++完整实现源码

转载本BLOG内任何文章，请以超链接形式注明出处。非常感谢。

顶

6

踩

0

- 上一篇




微软面试100题系列：一道合并链表问题的解答[第42题]
- 下一篇

[最新答案V0.4版]微软等数据结构+算法面试100题[第41-60题答案]

相关文章推荐

- 一步一图一代码，一定要让你真正彻底明白红黑树
- 红黑树C++完整源码
- 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
- 经典算法研究系列：五、红黑树算法的实现与剖析
- 红黑树c源码实现与剖析
- C语言大型软件设计的面向对象--宋宝华
- Shell脚本编程
- 一个红黑树实现c源码
- 红黑树实现源码
- Retrofit 从入门封装到源码解析
- 通用红黑树（Tree-Map）容器纯C实现
- 红黑树的C++完整实现源码
- 红黑树源码
- 跳过Java开发的各种坑
- 红黑树的源码与测试函数

查看评论

	zwp1982feng	41楼 2017-09-01 18:37发表	
	<pre>static rb_node_t* rb_rotate_right(rb_node_t* node, rb_node_t* root) { rb_node_t* left = node->left; // left 可能为NULL if ((node->left = left->right)) // left为NULL，导致崩溃 {</pre>		40楼 2017-09-01 18:25发表

Inqariel



39楼 2017-03-20 14:16发表

```
static rb_node_t* rb_erase_rebalance(...)
{
//...
while ((!node || node->color == BLACK) && node != roo
t)
{
//...
if (parent->left == node)
{
other = parent->right;
if (other->color == RED)
{
//...
other = parent->right; // 这里other可能为NULL
}
// other为NULL时下面语句会崩溃
if ((!other->left || other->left->color == BLACK) &&
```

HoldHope



38楼 2016-11-17 11:27发表

插入算法，如果插入结点的父结点是红，但没有叔叔结点，该怎么处理呢？

是不是直接以祖父结点为支点左旋就可以了

HoldHope



37楼 2016-11-16 17:38发表

哦，不还意思，懂了。

李先森LSP



36楼 2015-07-28 22:57发表

博主，你好，看了您的代码，总算是慢点在弄懂红黑树了，就是这

```
node->parent = old->parent;
node->color = old->color;
node->right = old->right;
node->left = old->left;
```

按博主的意思，node是要顶替的节点，old是要删除的节点，但如果node是old的右儿子，那 node->right = old->right; 这句不就使node的右指向自己了吗，是我理解错了吗，博主能给我讲解下吗

linux_player_c



35楼 2015-06-27 16:35发表

你好，在rb_node_t* rb_erase () 函数中，在以下几行代码中

```
node->parent = old->parent;
node->color = old->color;
node->right = old->right;
node->left = old->left;
```

为什么会有node->color = old->color这句

难道说删除旧结点的时候，新顶替旧结点的结点还要继承旧结点的颜

色？不应该吧，还有，在(node->left && node->right)情况中，顶替被删结点的为node，而在非 (node->left && node->right)情况中，顶替被删的结点为child，而在函数尾调用的都是 root = rb_erase_rebalance(child, parent, root);恐怕不妥吧

ab0000_2008



34楼 2014-02-17 11:21发表

```
if ((o_left = other->left))    //w和其左孩子left
[w], 颜色交换。
{
    o_left->color = BLACK;    //w的左孩子变为由黑-
>红色
}
```

july你好，这个注释应该有问题吧，other的左孩子颜色应该是由红色变成黑色。请查看一下。

ab0000_2008



33楼 2014-02-17 11:06发表

```
static rb_node_t* rb_rotate_left(rb_node_t* node)
{
    rb_node_t* right = node->right;

    if(right)
    {
        right->parent = node->parent;
        right->left = node;
        node->parent = right;
        node->right = right->left;
        right->left->parent = node;
    }

    return right;
}
```

chenjingli1988



32楼 2013-11-03 19:54发表

左旋，右旋函数的入参之一是root，也就是红黑树根节点，这样安排欠妥，而且返回值也是root，这完全没有必要。你返回root干嘛呢？因为旋转函数往往是插入，删除，搜索函数递归实现中的一个小步骤。

tkwtkwtkw



31楼 2013-10-31 16:45发表

```
在rb_erase中
324. else
325. {
326. if (!node->left)
327. {
328. child = node->right;
329. }
```

```

330. else if (!node->right)
331. {
332. child = node->left;
333. }
else
{
//应该加上这个代码，处理没有儿子的情况，防止后面用到child是出
现野指针问题 (**以防万一嘛**)
child = NULL;
}
334. parent = node->parent;
335. color = node->color;
336.
337. if (child)

```

```

rb_node_t* delete_node(rb_node_t *p ,key_t record)
{
    rb_node_t *parent = NULL,*pchild = NULL,*child = NULL,*head = NULL;
    color_t color = 0;
    head=p;
    if (!(p = rb_search_auxiliary(record, p, NULL)))
    {
        printf("key %d is not exist!\n");
        return head;
    }
    child = pchild = (p->right) ? p->right : p->left;
    if(p->right == NULL || p->left == NULL)
    {
        parent = p->parent;
        color = p->color;
        //要删除的是根节点
        if(parent == NULL)
            head = pchild;
        else
        {
            if(parent->right == p)
                parent->right = pchild;
            else
                parent->left = pchild;
        }
        if (pchild)
        {
            pchild->parent = parent;
        }
        free(p);
    }
    else
    {
        pchild = p->right;
        parent = p;
        while(pchild->left)
        {
            parent = pchild;
            pchild = pchild->left;
        }
        color = pchild->color;
        child = pchild->right;
        p->data = pchild->data;
        p->key = pchild->key;
        if(parent == p)
            p->right = pchild->right;
    }
}

```

```
else
parent->left = pchild->right;
if (pchild->right)
{
pchild->right->parent = parent;
}
}
if (color == BLACK)
head = rb_erase_rebalance(child, parent, head);
return head;
}
```



guoxin52416

Re: 2016-03-05 20:21发表

回复tkwtktkw: 不过你这个修改的代码的当删除节点的左右孩子都不为空的时候是否应该加入free掉断树时的节点, 不然会有内存泄露。我修改了一下, 不知道对不对

```
else
{
pchild = p->right;
parent = p;
while(pchild->left)
{
parent = pchild;
pchild = pchild->left;
}
color = pchild->color;
child = pchild->right;
p->data = pchild->data;
p->key = pchild->key;
if(parent == p)
p->right = pchild->right;
else
parent->left = pchild->right;
if (pchild->right)
{
pchild->right->parent = parent;
}

if(pchild)
{//释放删除节点防止内存泄露
free(pchild);
}
}
```



guoxin52416

Re: 2016-03-05 20:15发表

回复tkwtktkw: 这样确实简单了不少, 因为删除的过程本来就是用右子树的最左孩子替换, 只关心值key,value即可, 不用整个节点, 全部都过去。



tkwtktkw

Re: 2013-10-31 16:48发表

回复tkwtktkw: 感觉删除部分比较繁琐, 修改了一下, 原来是删除old, 将node放到了old的位置, 现在删除的是node, 用node的数据, 覆盖old的数据, 结果都是一样的, 但是觉得这样比较简单!



warren05

Re: 2014-09-07 13:20发表

回复tkwtktkw: 确实是删除node比删除old, 在用node去将old原来的指针全部找回来要简单, 省了很多指针的操作。



shihai1118

30楼 2013-08-27 11:32发表

我觉得楼主的删除部分写的不好, 假如要删除的节点的左右子树全不为空, 找到后继节点后, 把后继节点的数据交换, 然后将后继节点删了就行了, 感觉楼主写的好繁琐



29楼 2013-08-27 11:24发表

shihai1118



28楼 2013-08-26 10:01发表

```
if (parent)
{
if (parent->left == node)
{
parent->left = child;
}
else
{
parent->right = child;
}
}
```

删除部分, 后继节点的父节点肯定存在, 而且, 后继节点肯定是父节点的左孩子

ayedsa

thank you



27楼 2013-05-24 18:14发表

毫无影响力

写的真好, 正需要用到红黑树作为大数据存储用, 谢谢!



26楼 2013-03-19 15:49发表

SupremeHover

朋友, 你这有Java实现的红黑树吗?
C看起来有点费劲。。。



25楼 2012-10-26 11:12发表

今天写红黑树的时候，重新看了下注释，发现插入修复那块有点不对哦。

//五、红黑树的3种插入情况 【中的】

//上述情况1中，只考虑了z作为父母的右孩子的情况

//这部分是特别为情况1中，z作为左孩子情况，而写的

这两条应该不对，因为不管z作为左孩子还是右孩子，根据对称性，在情况1中的处理都是一样的。还是按着代码本身的思路，实际上考虑的是父亲是祖父的左孩子，还是父亲是祖父的右孩子。



v_JULY_v

Re: 2012-11-09 11:15发表

回复SupremeHover：你好，关于你在本文评论下第23楼指出来的注释错误已经修正，再次谢谢你！



v_JULY_v

Re: 2012-10-26 11:17发表

回复SupremeHover：EN，是的，你已经在本文评论下第23楼指出来了呢，我也回复你了，只是原文还未修改过来，你忘了？
(我会找时间，尽快修改过来的，再次谢谢)



SupremeHover

Re: 2012-10-26 11:30发表

回复v_JULY_v：我汗，我记忆力衰退得厉害啊~o(∩_∩)o 哈哈~又给你添麻烦了哈~



v_JULY_v

Re: 2012-10-26 11:39发表

回复SupremeHover：没事，有问题，欢迎继续指正:-)



SupremeHover

24楼 2012-10-22 09:18发表

而且他的删除部分逻辑上存在一定混乱，不如linux的源码写得好的，希望博主能够注释linux的源码。



v_JULY_v

Re: 2012-10-22 10:49发表

回复SupremeHover：出来后，时间真的是个奢侈的东西，不像一年多前在学校写这篇文章时能为所欲为了。不过，有机会，会做的！



SupremeHover

23楼 2012-10-22 08:46发表

他的删除部分有一些多余的判断。比如：
if (node->left && node->right)
{
node = node->right;


```
while ((left = node->left) != NULL)
{
    node = left;
}
child = node->right;
parent = node->parent;
color = node->color;
```

```
if (child)
{
    child->parent = parent;
}
if (parent)
{
    if (parent->left == node)
    {
        parent->left = child;
    }
    else
    {
        parent->right = child;
    }
}
else
{
    root = child;
}
```

在old有双非空子结点的情况下，parent根本不可能为NULL，root = child纯属瞎写。



v_JULY_v

Re: 2012-10-22 10:48发表

回复SupremeHover：恩？再说具体点？



SupremeHover

Re: 2012-10-22 14:53发表

回复v_JULY_v：因为它的大前提if中：old有双非空子结点，接着又去判断用来替换old的结点node的parent是否存在，这是没有意义的，因为node是树中的一个结点，而且不是根结点，他老爸必须存在。



SupremeHover

22楼 2012-10-22 08:40发表

插入修补的部分：

//这部分是特别为情况1中，z作为左孩子情况，而写的。

上一句注释对我产生了误导，让我以为是因为情况1的z有左右孩子的区别，才需要专门把判断p[z]是left还是right分开写成两段大致相同的代码。

其实跟z是左孩子还是右孩子完全没有关系，写成两段是因为二叉树的对称性！

按照代码来看，原原本本就是专门针对p[z]是left还是right来写的。

- 1、p[z]是left和z是left的情况对称于p[z]是right和z是right的情况
- 2、p[z]是left和z是right的情况对称于p[z]是right和z是left的情况
- 3、p[z]是right和z是right的情况对称于p[z]是left和z是left的情况（根据对称来说，其实就是1）
- 4、p[z]是right和z是left的情况对称于p[z]是left和z是right的情况（其实就是2）

同样是由于对称性，情况1不管z是左孩子还是右孩子，操作都是一样的！



v_JULY_v

Re: 2012-10-22 10:43发表

回复SupremeHover：1、那部分注释确实有问题，应该与“if (parent == gparent->left) //当祖父的左孩子即为父母时。”对应

也就是说，

“//这部分是特别为情况1中，z作为左孩子情况，而写的。”

应改正为=>

“// if (parent == gparent->right) 当祖父的右孩子即为父母时。”

2、后来我又仔细看了此文中关于修复删除的部分：http://blog.csdn.net/v_JULY_v/article/details/6105630（文中，只考虑了父结点为祖父左子的情况），也就是说当父结点为祖父右子的时候，便是上述让你产生误解注释下的那部分代码，也就是如你所说的：“按照代码来看，原原本本就是专门针对p[z]是left还是right来写的”。

3、你说的“同样是由于对称性，情况1不管z是左孩子还是右孩子，操作都是一样的！”，我认为，是左孩子，和右孩子，对应的具体右旋还是左旋操作应该是不一样的，一年多前的东西，要稍稍花点时间温故，后续我会再看下。你看得很认真，谢谢你，朋友！



v_JULY_v

Re: 2012-10-22 11:19发表

回复v_JULY_v：回复SupremeHover：在此文 http://blog.csdn.net/v_JULY_v/article/details/6105630，的插入修复具体操作中：

“情况3：当前结点的父结点是红色且祖父结点的另一个子结点（叔叔结点）是红色。

此时父结点的父结点一定存在，否则插入前就已不是红黑树。

与此同时，又分为父结点是祖父结点的左子还是右子，对于对称性，我们只要解开一个方向就可以了。

在此，我们只考虑父结点为祖父左子的情况。

同时，还可以分为当前结点是其父结点的左子还是右子，但是处理方式是一样的。我们将此归为同一类。”

不过，这里的“当前结点是其父结点的左子还是右子，但是处理方式是一样的”仅仅是针对的上述情况3，所以具体到当前节点是左孩子，和右孩子，我认为对应的具体右旋还是左旋操作应该是不一样的，正如后面的插入修复情况4、5所述，也能说明此观点。

SupremeHover觉得呢？



v_JULY_v

Re: 2012-10-22 11:30发表

回复v_JULY_v: 尽管那篇文章的插入修复的情况3、4、5，可以认为是插入了4以后的一系列插入修复操作！
但涉及到的后续是右旋还是左旋的具体操作还是不一样的。



v_JULY_v

Re: 2012-10-22 12:57发表

回复v_JULY_v: 回复SupremeHover: 刚又看了“http://blog.csdn.net/v_JULY_v/article/details/6105630”文中插入修复操作的几张图，插入节点4以后，当前节点N指针发生了变化，跟节点4最初是左孩子还是右孩子无关，因为之后的无论是左旋操作还是右旋操作，针对的都是非节点4的新的当前节点（节点4早已经不是当前节点了）！
SupremeHover，你是对的！感谢你的指正！



SupremeHover

Re: 2012-10-22 14:47发表

回复v_JULY_v: 呵呵，很感谢你认真地看了我的评论，我已经在你和saturnma的大力帮助下会写红黑树代码了，谢谢哈~



v_JULY_v

Re: 2012-10-22 14:56发表

回复SupremeHover: GOOD! 代码写出来后，欢迎与我分享！



SupremeHover

21楼 2012-10-21 17:51发表

回复v_JULY_v: 源代码是对的，他没看懂。
if ((node->right = right->left))
等价于if ((node->right = right->left) != NULL)
略去判等运算符是C语言的特色。



v_JULY_v

Re: 2012-10-22 09:55发表

回复SupremeHover: EN, 是的, 第19楼的xujian871226
已经看出来是正确的了! 之前未仔细看代码, 多谢你。



stecdeng

20楼 2012-07-04 18:35发表

很少有学习借鉴别人的代码后 又说代码有诸多问题, 这样不太好, 对
拿出代码的人也不太公平

另

```
static rb_node_t* rb_rotate_left(rb_node_t* node, rb_node_t* root)
{
    rb_node_t* right = node->right; //指定指针指向 right<--node->right

    if ((node->right = right->left))
    {
        right->left->parent = node; //好比上面的注释图, node成为b的父母
    }
}
```

这个= 是否写成== ?



v_JULY_v

Re: 2012-10-22 09:57发表

回复stecdeng: 源代码是对的
if ((node->right = right->left))
等价于if ((node->right = right->left) != NULL)



lizhenhuan517

Re: 2013-03-14 12:23发表

回复v_JULY_v: 回复stecdeng: 源代码是对的
if ((node->right = right->left))
等价于if ((node->right = right->left) != NULL)
这个是不是还是写错了?
等价于if ((node->right == right->left) != NULL)
是吗?



lizhenhuan517

Re: 2013-03-14 12:34发表

回复lizhenhuan517: 嗯。我已经理解了。打扰了。
判断right->left是否为空,
并且做了一次复制操作。



歌神的卖

19楼 2012-05-12 16:39发表

感谢博主
xujian871226



18楼 2012-03-13 14:09发表

```
if ((node->right = right->left))
{
right->left->parent = node; //好比上面的注释图，node成为b的父母
}
```

左旋中这句话如何理解：node->right不是就等于right嘛？怎么会等于right->left呢？



xujian871226

Re: 2012-03-13 14:11发表

回复xujian871226: sorry, 我看错了, 我看成括号内是=了, 其实就是判单right->left不为null嘛。
是自己不小心。



xujian871226

17楼 2012-03-13 13:55发表

xy56308890左旋右旋代码上面的那些图为何的右孩子为何不用"\"呢, 既然左孩子用的是"l", 右孩子用"\"的话不是刚好对应嘛。



16楼 2011-12-09 15:18发表

楼主, 请问 rb_delete 中有一段代码:

```
if (node->parent == old) {
    parent = node;
}
```

没弄白什么意思? 恰恰这儿没注释, 求指点啊!



ikillmeba

Re: 2016-06-22 11:36发表

回复xy56308890: 我也没看懂! 为啥要这样做



ouen333

15楼 2011-08-01 14:57发表

houshi在么?

请教下,代码运行老有问题呀,同14楼,用F11一步一步运行,感觉都没问题,呀



14楼 2011-06-06 22:41发表

v_JULY_v



13楼 2011-04-08 11:12发表





照着你的代码敲, 不知道到底是为什么, 插入修复时老是出错, 比如说, 在你修复时, 左旋和右旋时, left和right可能是NULL的情况考虑了吗? 我调试一直都是这个问题

algorithm__



12楼 2011-02-27 19:47发表

代码以插入的形式重新编辑了下。

<div>algorithm__</div> <div>[e10]</div>	<div></div>	<div>11楼 2011-02-27 19:47发表</div>
<div>algorithm__</div> <div>[e03]</div>	<div></div>	<div>10楼 2011-02-27 19:47发表</div>
<div>algorithm__</div> <div>[e01]</div>	<div></div>	<div>9楼 2011-02-27 19:47发表</div>
<div>v_JULY_v</div> <div>做的太棒了。狂顶.0.....</div>	<div></div>	<div>8楼 2011-01-28 19:14发表</div>

本BLOG 内所有任何文章，永久勘误，任何问题、错误，一经发现，
立马修正。

<div> v_JULY_v</div> <div>回复 v_JULY_v: 非常感谢，网友のWhatever来信指正与 指导。当我一经确认，此源码，确实有误，立马修正。谢 谢，各位。</div>	<div>Re: 2011-01-31 12:21发表</div>
<div> chjttony</div> <div>v_JULY_v</div>	<div>7楼 2011-01-10 17:01发表</div> <div>6楼 2011-01-07 19:08发表</div>
<div>turbsky2010</div> <div>资源，已经上传好。</div>	<div></div> <div>5楼 2011-01-05 09:57发表</div>
<div>turbsky2010</div> <div>哦 我看错了。。。</div>	<div></div> <div>4楼 2011-01-05 09:55发表</div>
<div>v_JULY_v</div> <div>很多==被写成了=</div>	<div></div> <div>3楼 2011-01-04 12:10发表</div>

[i = 1] insert key 77 success!
[i = 1] search key 77 success!
[i = 2] insert key 2 success!
[i = 2] search key 2 success!

```
[i = 3] insert key 33 success!  
[i = 3] search key 33 success!  
[i = 4] insert key 25 success!  
[i = 4] search key 25 success!  
[i = 5] insert key 83 success!  
[i = 5] search key 83 success!  
[i = 6] insert key 29 success! [i = 6] search key 29 success!  
.....  
[i = 99] search key 85 success! Press any key to continue
```



v_JULY_v

Re: 2011-01-04 12:11发表

回复 v_JULY_v: 这是程序 的运行结果。



v_JULY_v

2楼 2011-01-04 12:09发表

v_JULY_v

```
[i = 1] insert key 77 success!  
[i = 1] search key 77 success!  
[i = 2] insert key 2 success!  
[i = 2] search key 2 success!  
[i = 3] insert key 33 success!  
[i = 3] search key 33 success!  
[i = 4] insert key 25 success!  
[i = 4] search key 25 success!  
[i = 5] insert key 83 success!  
[i = 5] search key 83 success!  
[i = 6] insert key 29 success!  
[i = 6] search key 29 success!  
.....  
[i = 99] search key 85 success!  
Press any key to continue
```



1楼 2011-01-03 21:58发表

作为红黑树系列的结尾文章，最后一篇第4篇文章，将直接剖析linux内核中红黑树的实现。