



数据结构和算法  
码龄14年 暂无认证

742 8002 6716 126万+  
原创 周排名 总排名 访问 等级

1万+ 1万+ 6354 1102 1万+  
积分 粉丝 获赞 评论 收藏



私信

关注



搜博主文章



热门文章

c语言基础程序——经典100道实例。 63482




Android Paint的使用详解 62293

android10获取相册失败的方式 56380

什么是递归，通过这篇文章，让你彻底搞懂递归 43681

2021毛概知识点章节整理(完整版) 40692

分类专栏

-  dart语言学习 4篇
-  C语言谭浩强课后答案 1篇
-  其他资料 34篇

## TreeMap红黑树源码详解

原创

数据结构和算法

于 2017-08-22 16:28:49 发布

阅读量4k

收藏 21

点赞数 9

版权

分类专栏：

数据结构和算法

文章标签：

Android

TreeMap

算法

源码



数据结构和算法 专栏收录该内容

62 订阅

293 篇文章

订阅专栏

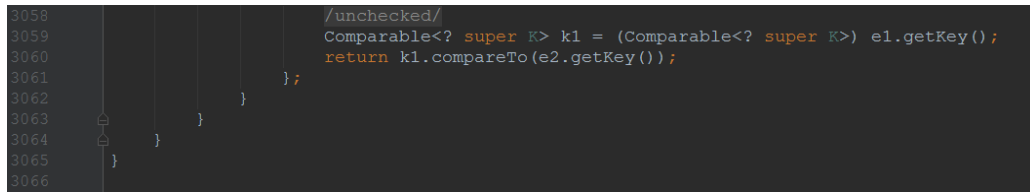
尊重原创，转载请标明出处 <http://blog.csdn.net/abcdef314159>

在分析 **源代码** 之前，最好要标注出处，因为在Java中和Android中同一个类可能代码就会不一样，甚至在Android中不同版本之间代码也可能会有很大的差别，下面分析的是红黑树 **TreeMap**，在\sources\android-25中。

**红黑树** 的几个性质要先说一下，

1. 每个节点是红色或者黑色的。
2. 根节点是黑色的。
3. 每个叶节点的子节点是黑色的（叶子节点的子节点可以认为是null的）。
4. 如果一个节点是红色的，则它的左右子节点都必须是黑色的。
5. 对任意一个节点来说，从它到叶节点的所有路径必须包含相同数目的黑色节点。

TreeMap还有一个性质，就是他的左子树比他小，右子树比他大，这里的比较是按照key排序的。存放的时候如果key一样就把他替换了。



乍一看代码TreeMap有3000多行，其实他里面有很多内部类，有Values,EntrySet,KeySet,PrivateEntryIterator,EntryIterator,ValueIterator,KeyIterator,DescendingKeyIterator,NavigableSubMap,AscendingSubMap,DescendingSubMap,SubMap,TreeMapEntry,TreeMapSplitter,KeySplitter,DescendingKeySplitter,ValueSplitter,

EntrySplitter多达十几个内部类。其实很多都不需要了解，下面主要来看一下TreeMapEntry这个类，它主要是红黑树的节点

```
1 /**
2  * Node in the Tree. Doubles as a means to pass key-value pairs back to
3  * user (see Map.Entry).
4  */
5
```



数据结构和算法

关注

9

21

9

分享

打赏




...

专栏

```
9 TreeMapEntry<K,V> left = null;//左子树
10 TreeMapEntry<K,V> right = null;//右子树
```

既然是棵树，那么肯定就会有put方法以及remove方法，那么这里就从最简单的着手，先看一下put方法

```
1 /**
2  * Associates the specified value with the specified key in this map.
3  * If the map previously contained a mapping for the key, the old
4  * value is replaced.
```

	工具	35篇
	数据结构	1篇
	LeetCode-初级算法	16篇

## 最新评论

《C语言程序设计》课后习题答案(第四...  
Just Follow JJ\_Lin: what? If you want the c  
ode answers for exercises on data struc ...

《C语言程序设计》课后习题答案(第四...  
liuhuit2006: really? thanks.

c语言基础程序——经典100道实例。  
周山涛: sxau 🤔

c语言基础程序——经典100道实例。  
2022 杰: 003如果是奇数呢?

c语言基础程序——经典100道实例。  
盼小辉 \: 优质好文，收藏起来慢慢学...

## 最新文章

C语言程序设计第五版(谭浩强)第一章课后答  
案

双端队列详解

数据结构——笛卡尔树详解

2024年 214篇	2023年 24篇
2022年 22篇	2021年 236篇
2020年 215篇	2019年 14篇
2018年 3篇	2017年 12篇
2016年 27篇	

```
5 *  
6 * @param key key with which the specified value is to be associated  
7 * @param value value to be associated with the specified key  
8 *  
9 * @return the previous value associated with {@code key}, or  
10 *      {@code null} if there was no mapping for {@code key}.
```

put方法存放的时候，首先是会存放到叶子节点，然后在进行调整。上面有一个重量级的方法fixAfterInsertion还没有分析，在分析fixAfterInsertion方法之前来看一下其他的几个方法，

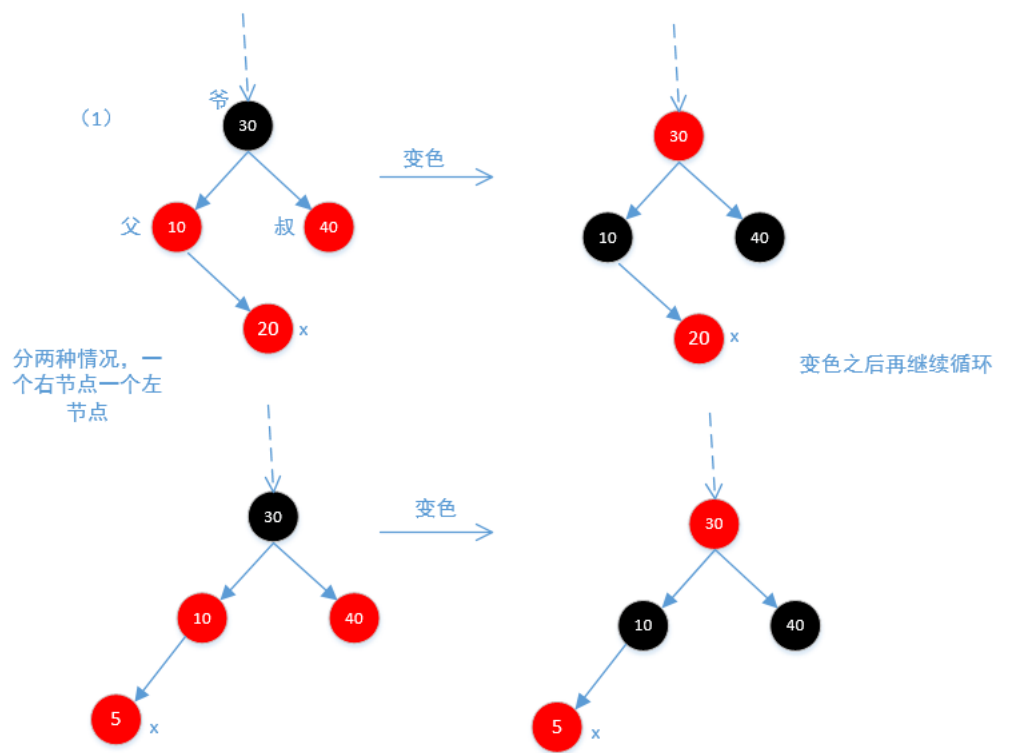
```
1 private static <K,V> boolean colorOf(TreeMapEntry<K,V> p) {  
2     // 获取树的颜色，如果为null，则为黑色，这一点要记住，待会下面分析的时候会用到  
3     return (p == null ? BLACK : p.color);  
4 }  
5 // 找父节点  
6 private static <K,V> TreeMapEntry<K,V> parentOf(TreeMapEntry<K,V> p) {  
7     return (p == null ? null: p.parent);  
8 }  
9 // 设置节点颜色  
10 private static <K,V> void setColor(TreeMapEntry<K,V> p, boolean c) {
```

下面再来看一下fixAfterInsertion方法

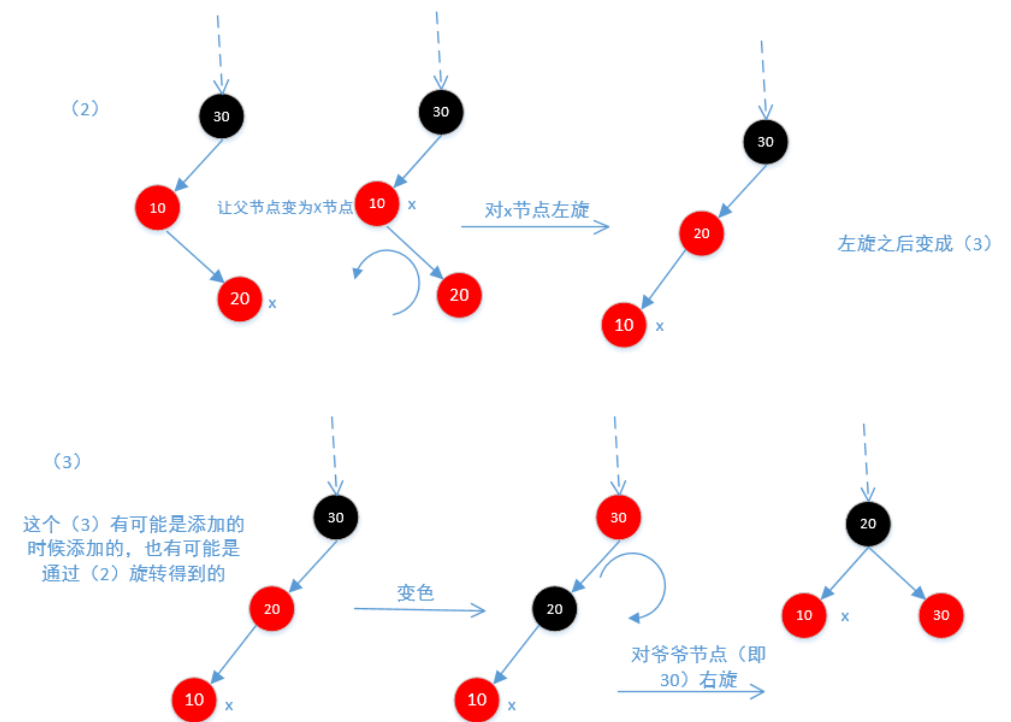
```
1 /** From CLR */  
2 private void fixAfterInsertion(TreeMapEntry<K,V> x) {  
3     // 在红黑树里面，如果加入一个黑色节点，则导致所有经过这个节点的路径黑色节点数量增加1，  
4     // 这样就肯定破坏了红黑树中到所有叶节点经过的黑色节点数量一样的约定。所以，  
5     // 我们最简单的办法是先设置加入的节点是红色的。  
6     x.color = RED;  
7     // 当前节点变为红色，如果他的父节点是红色则需要调整，因为父节点和子节点不能同时为红色，但可  
8     // 所以这里的循环条件是父节点必须为红色才需要调整。  
9     while (x != null && x != root && x.parent.color == RED) {  
10        // 这里会分多种情况讨论，
```

上面列出了6中可能，下面通过6张图来说明

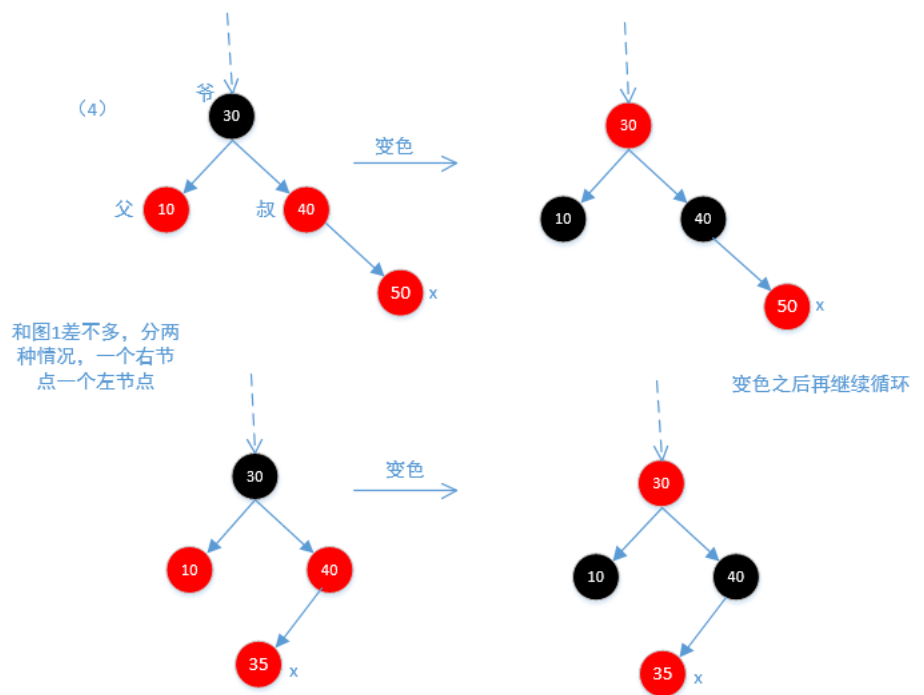
下面是图1，不需要旋转，只需要调整颜色即可



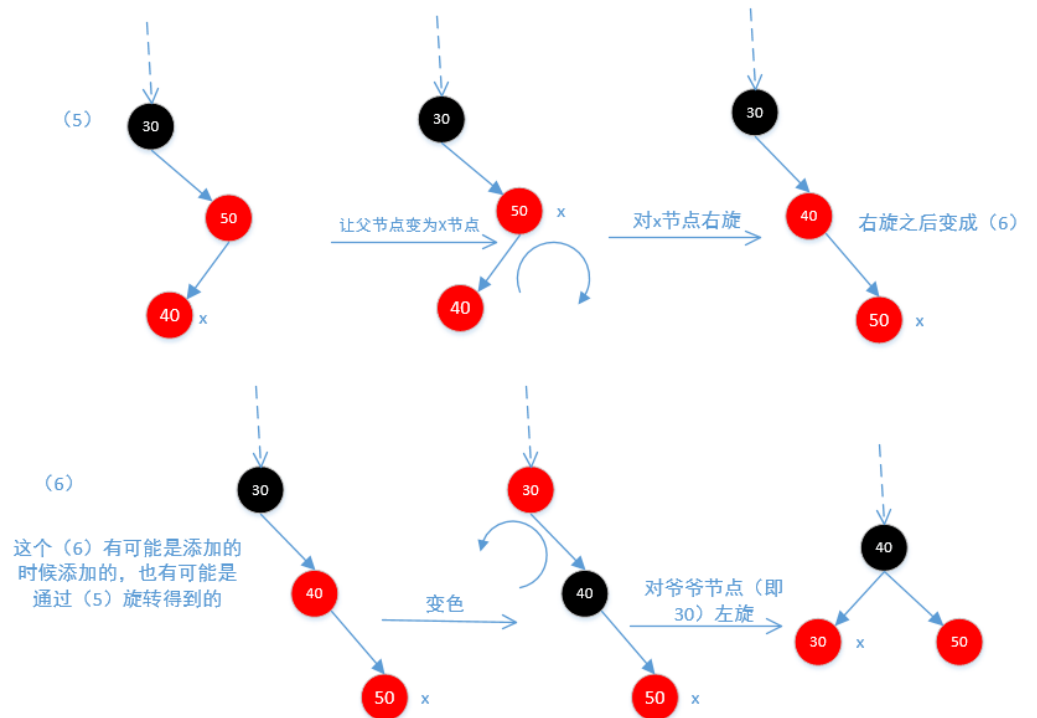
下面是图2和图3，因为不平衡，所以需要旋转



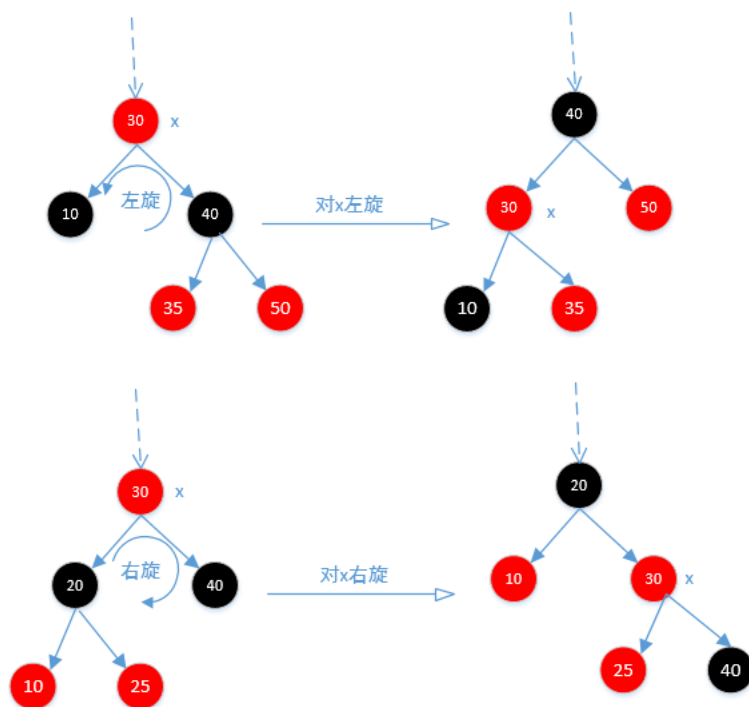
下面是图4，和图1差不多，也分两种情况，一种是左节点一种是右节点



下面是图5和图6，因为不平衡，所以需要旋转



无论怎么旋转，他的左节点永远小于他，右节点永远大于他。通过不断的while循环，最终保证红黑树的平衡。下面来看一下旋转的方法，先看一下图



```

1  /** From CLR */
2  private void rotateLeft(TreeMapEntry<K,V> p) {
3  // 参照上面旋转的图来分析，p就是图中的x
4      if (p != null) {
5          TreeMapEntry<K,V> r = p.right; // r相当于图中的40节点
6          p.right = r.left; // 让35节点 (r.left也就是图中40节点的左节点) 等于p的右节点，看上l
7          // 如果r.left != null，让p等于他的父节点，因为在上一部他已经等于p的右节点，自然就是他
8          // 所以他的父节点自然就变成p了
9          if (r.left != null)
10             r.left.parent = p;

```

而右旋方法rotateRight和左旋差不多，这里就不在分析。put方法分析完了，那么下一个就是remove方法了，

```

1  public V remove(Object key) {
2  //getEntry(Object key)方法是获取TreeMapEntry，如果比当前节点大则找右节点，如果比当前节点小
3  //通过不断的循环，知道找到为止，如果没找着则返回为null。
4      TreeMapEntry<K,V> p = getEntry(key);
5      if (p == null)
6          return null;
7
8      V oldValue = p.value;
9      // 找到之后删除
10     deleteEntry(p);
11     return oldValue;
12 }

```

下面再看一下删除方法deleteEntry。

```

1  /**
2   * Delete node p, and then rebalance the tree.
3   */
4  private void deleteEntry(TreeMapEntry<K,V> p) {
5      modCount++;
6      size--; // 删除，size减1
7
8      // If strictly internal, copy successor's element to p and then make p
9      // point to successor

```

```

9 // point to successor.
10 // 当有两个节点的时候不能直接删除，要删除他的后继节点，后继节点最多只有一个子节点。因为如果

```



上面分析的时候有两个方法successor和fixAfterDeletion没有分析，下面先来看一下successor方法，这个方法很简单，其实就是返回大于节点p的最小值，看一下代码

```

1 /**
2  * Returns the successor of the specified Entry, or null if no such.
3  */
4 static <K,V> TreeMapEntry<K,V> successor(TreeMapEntry<K,V> t) {
5     if (t == null)
6         return null;
7     else if (t.right != null) { // t的右节点不为空
8         TreeMapEntry<K,V> p = t.right;
9         // 循环左节点，如果左节点一开始就为null，那么就直接返回p，此时p是t的右节点，如果p的左
10        // 存在，那么会一直循环，一直在找左节点，直到为null为止，

```



OK，下面再来看一下fixAfterDeletion方法，因为x所在分支少了一个黑色的节点，所以他的主要目的就是让x分支增加一个黑色节点。这个比fixAfterInsertion方法还难理解，看代码

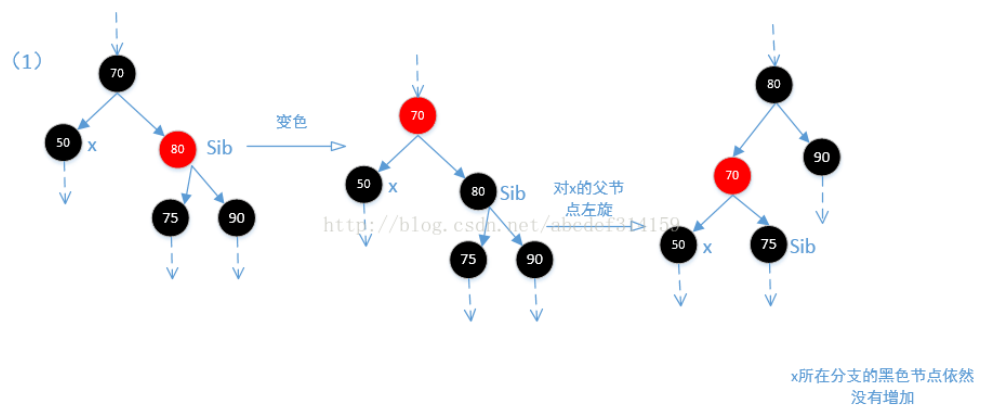
```

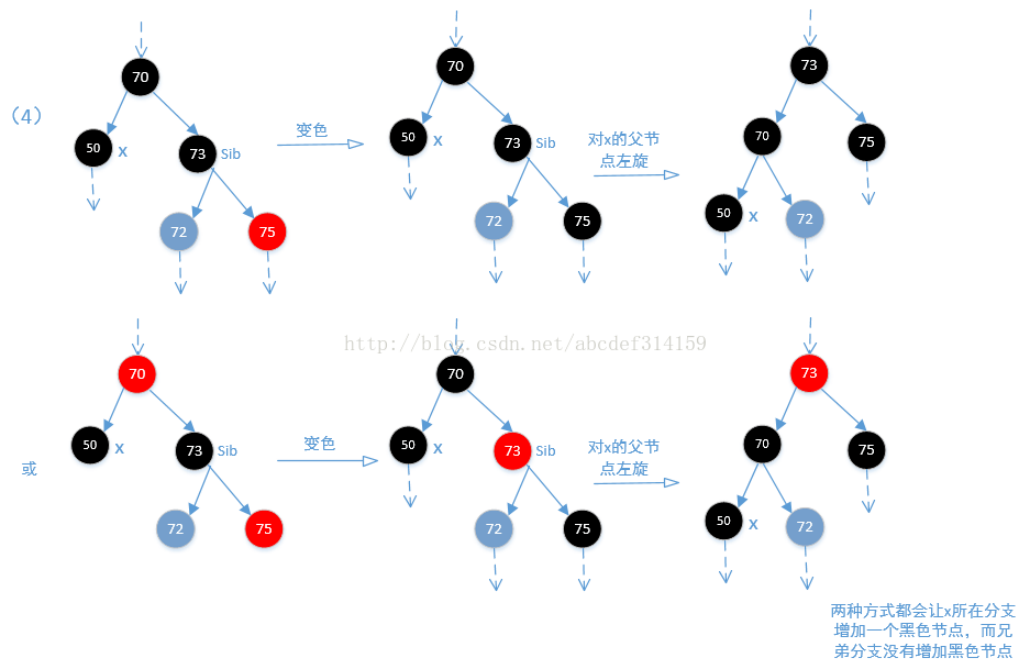
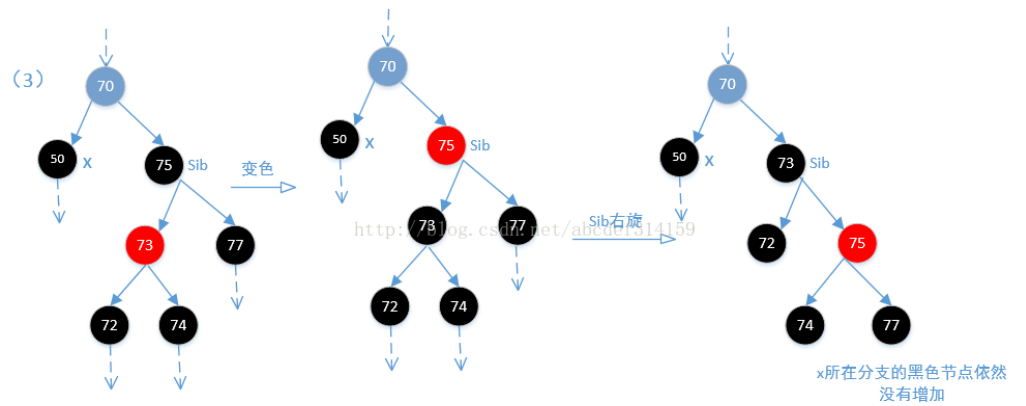
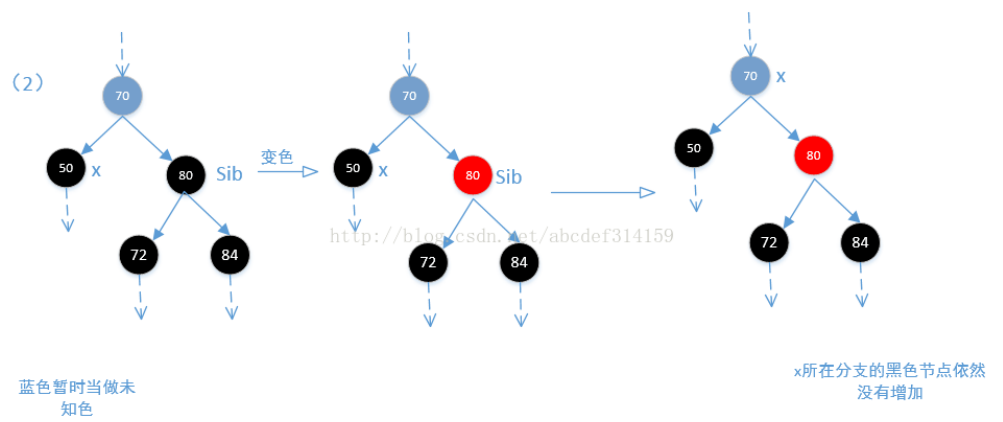
1 /** From CLR */
2 private void fixAfterDeletion(TreeMapEntry<K,V> x) {
3     // 再看这个方法之前先看一下最后一行代码，他会把x节点设置为黑色
4     // 很明显，在x只有黑色的时候才会调整，因为删除黑色打破了红黑平衡，但deleteEntry方法中的删
5     // 一种是替换之后的replacement，这个replacement不是删除的节点，需要删除的节点在这之前就
6     // 他是来找平衡的，因为删除之后在这一分支上少了一个黑色节点，如果replacement节点为红色，那
7     // while循环，直接在最后把它置为黑色就正好弥补了删除的黑色节点，如果replacement是黑色，
8     // 下面的while循环（前提是replacement不等于root）。还有一种就是没有子节点的，先调整完在删
9     // 红色，就不用执行while循环，直接删除就是了，下面置不置为黑色都一样，如果是黑色，就必须执
10    // 因为删除黑色会打破红黑平衡。

```



结合上面代码看一下下面的四张图





OK，到现在为止put和 move 方法都已经分析完了，下面看一下其他方法，

```

1  /**
2   * Returns the first Entry in the TreeMap (according to the TreeMap's
3   * key-sort function). Returns null if the TreeMap is empty.
4   */
5   // 返回第一个节点，最左边的，也是最小值
6   final TreeMapEntry<K,V> getFirstEntry() {
7       TreeMapEntry<K,V> n = root;

```

```

7         if (compareEntry(g, e) < 0) p = e;
8         if (p != null)
9             while (p.left != null)
10                p = p.left;

```



再来看一下containsValue方法

```

1 // 通过不停的循环查找，先从第一个查找，getFirstEntry() 返回的是树的最小值，如果不等，再找比e大
2 // successor(e) 返回的是e的后继节点，其实就是比e大的最小值，他还可以用于输出排序的大小
3 public boolean containsValue(Object value) {
4     for (TreeMapEntry<K,V> e = getFirstEntry(); e != null; e = successor(e))
5         if (value.equals(e.value))
6             return true;
7     return false;
8 }

```

再看一个getCeilingEntry，这个方法比较绕

```

1 /**
2  * Gets the entry corresponding to the specified key; if no such entry
3  * exists, returns the entry for the least key greater than the specified
4  * key; if no such entry exists (i.e., the greatest key in the Tree is less
5  * than the specified key), returns {@code null}.
6  */
7 // 返回最小key大于或等于指定key的节点。
8 final TreeMapEntry<K,V> getCeilingEntry(K key) {
9     TreeMapEntry<K,V> p = root;
10    while (p != null) {

```



下面再看一个和getCeilingEntry方法类似的方法getFloorEntry。

```

1 /**
2  * Gets the entry corresponding to the specified key; if no such entry
3  * exists, returns the entry for the greatest key less than the specified
4  * key; if no such entry exists, returns {@code null}.
5  */
6 // 返回最大key小于或等于指定key的节点。
7 final TreeMapEntry<K,V> getFloorEntry(K key) {
8     TreeMapEntry<K,V> p = root;
9     while (p != null) {
10        int cmp = compare(key, p.key);

```



getHigherEntry函数和getCeilingEntry函数有点类似，不同点是如果有相同的key，getCeilingEntry会直接返回，而getHigherEntry仍然会返回比key大的最小节点，

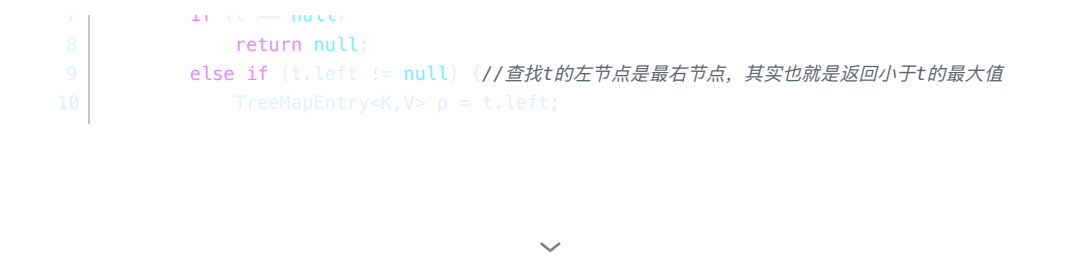
同理getLowerEntry函数和getFloorEntry函数很相似，这里就不在详述。下面在看一个方法predecessor

```

1 /**
2  * Returns the predecessor of the specified Entry, or null if no such.
3  */
4 static <K,V> TreeMapEntry<K,V> predecessor(TreeMapEntry<K,V> t) {
5     // 这个和successor相反，他返回的是前继节点。后继节点返回的是大于t的最小节点，而前继节点返回的是
6     // t的最大节点
7     if (t == null)

```





OK，目前为止TreeMap主要方法都已整理完毕。

参阅：[Java 集合系列12之 TreeMap详细介绍\(源码解析\)](#)和使用示例

<b>《华为手机升级鸿蒙系统 – 用户调研》</b>		广告
本次调研旨在了解您未升级到鸿蒙系统的原因和对系统的需求，以便我们更好地为您服务。		
JAVA Map详解之TreeMap(红黑树)	jy317358306的博客	2040
前言 上一章的HashMap并没有提到红黑树，就是因为本章的TreeMap就是一棵红黑树。TreeMap是存储键值对(key-value结		
Java面试要点59 - Java TreeMap源码解析:红黑树实现		1-14
TreeMap是Java集合框架中基于红黑树实现的有序映射类,它可以保证键值对按照键的自然顺序或指定的比较器顺序进行排		
TreeMap源码详解—彻底搞懂红黑树的平衡操作_treemap 红黑树		1-22
JavaTreeMap实现了SortedMap接口,也就是说会按照key的大小顺序对Map中的元素进行排序,key大小的评判可以通过其本		
“红黑树”详解   红黑树的应用场景	Linuxhus的博客	2508
今天我们要说的红黑树就是就是一棵非严格均衡的二叉树，均衡二叉树又是在二叉搜索树的基础上增加了自动维持平衡的性		
红黑树实现源码		07-22
关于红黑树的功能实现		
红黑树的Java实现参考源码_arraylist源码资源-CSDN文库		2-7
算法导论中红黑树的C代码实现,已添加注释 红黑树Java实现 浏览:30 红黑树java实现,代码能实现,详细。 红黑树源码 浏览:7		
红黑树java实现资源-CSDN文库		1-30
红黑树java实现,代码能实现,详细。 红黑树的Java实现参考源码 浏览:60 5星 - 资源好评率100% 红黑树的增删查的Java实		
红黑树的概念以及基本模拟 最新发布	zc331的博客	720
若uncle存在且为黑色，那么cur一定不是新插入的红色节点，而是cur下面的节点通过变色变上来的红色节点，否则parent的		
红黑树源码实现	最帅惋红曲	1198
继上一篇《思考红黑树》自己亲自动手把红黑树实现一遍，发现理论成功了要转换为实际代码还是有点难度的，困难点主		
红黑树java原理及实现源码_java 红黑树		2-7
红黑树 Red-Black Tree,简称R-B Tree),它一种特殊的二叉查找树。 红黑树是特殊的二叉查找树,意味着它满足二叉查找树的		
JAVA8 hashmap源码阅读笔记(红黑树链表)_transient node<k,v>[] ta...		2-9
本文详细介绍了Java8 HashMap的13个成员变量,包括默认初始容量、最大容量、负载因子等,并探讨了构造方法、Node存		
红黑树的添加删除操作 热门推荐	好好学习	2万+
  来自: http://hi.baidu.com/coolinc/blog/item/3aa07f3e162502eb54e723b1.html 介绍另一种平衡二叉树: 红黑		
红黑树源代码		03-03
c编写的红黑书，根据算法导论编写，宏内红黑书红黑书大地啊的		
红黑树详解		1-21
红黑树本质上是一种二叉查找树,在节点类中添加类一个用来标识颜色的字段,同时具有一定的规则。同时具备这亮点使得红		
通过Java实现红黑树及其可视化_红黑树java实现资源-CSDN文库		2-8
通过Java实现红黑树及其可视化 对于节点的定义,包含节点的数值、节点的颜色、节点的父节点、节点的左右叶子结点 通过		
Java TreeMap源码深度解析: 红黑树的秘密		
Java TreeMap 是一个有序的键值对集合，它基于红黑树数据结构实现。红黑树是一种自平衡的二叉搜索树，能够保证在插		
红黑树原理详解	yang201610的博客	410
红黑树红黑树定义和性质新的改变功能快捷键合理的创建标题，有助于目录的生成如何改变文本的样式插入链接与图片如何		

IT笔试面试-- <b>红黑树</b> 详细解析,代码+图解资源-CSDN文库	1-30
<b>红黑树详解</b> <b>红黑树</b> 是一种自平衡二叉查找树,它的统计性能要好于平衡二叉树(AVL 树),因此, <b>红黑树</b> 在很多地方都有应用。	
数据结构与 <b>算法</b> 【 <b>红黑树</b> 】的 <b>Java</b> 实现+图解_java实现 <b>红黑树</b>	1-15
建议先阅读普通二叉搜索树与平衡二叉搜索树的文章。理解一些基本的二叉树知识数据结构与 <b>算法</b> 【二叉搜索树】 <b>Java</b> 实	
<b>红黑树源码</b>	04-09
第一次从无到有写代码。从二叉树到 <b>红黑树</b> 到打印树的设计,写了将近2个星期。	
<b>Android</b> 版数据结构与 <b>算法</b> (十): 终极之树- <b>红黑树</b> 与 <b>TreeMap</b> 详细解析	qq_16386581的博客 670
本文目录 一、为什么要创建 <b>红黑树</b> 这种数据结构 在上篇我们了解了AVL树,既然已经有了AVL这种平衡的二叉排序树,为	
<b>红黑树源码</b> (含注释)	暗星涌动 298
<b>红黑树源码</b> (含注释) #include <stdio.h> #include <stdlib.h> #include <string.h> #include <time.h> typed	
<b>Java</b> 提高篇 (二七) ----- <b>TreeMap</b>	weixin_30580943的博客 2091
原文出自: http://cmsblogs.com/?p=1013。尊重作者的成果,转载请注明出处! 个人站点: http://cmsblogs.com-----	
<b>红黑树算法</b> 源代码 (摘自linux内核代码)	fanxiushu的专栏 1913
在应用层开发中, STL标准库的map容器内部实现就是 <b>红黑树算法</b> , 既然有现成的, 可能大部分人都不会写 <b>红黑树算法</b> 来管	
<b>红黑树</b> 与 <b>TreeMap</b> 详细解析	李利科 553
一、为什么要创建 <b>红黑树</b> 这种数据结构 在上篇我们了解了AVL树,既然已经有了AVL这种平衡的二叉排序树,为什么还要有	
<b>红黑树</b> 完全解析(基于 <b>TreeMap</b> )	gin_yz的博客 457
<b>红黑树</b> 的例子 在线例子网址 https://www.cs.usfca.edu/~galles/visualization/RedBlack.html 内部节点及类定义 Get/Set方法省	