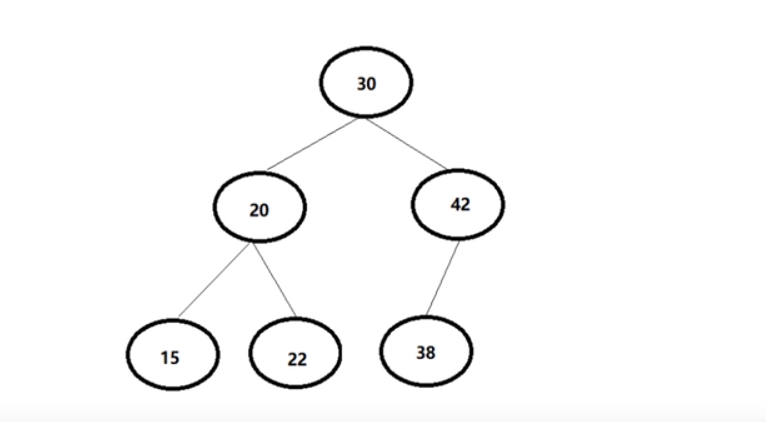
# TreeMap底层原理

## 6.1 二叉查找树

特点：

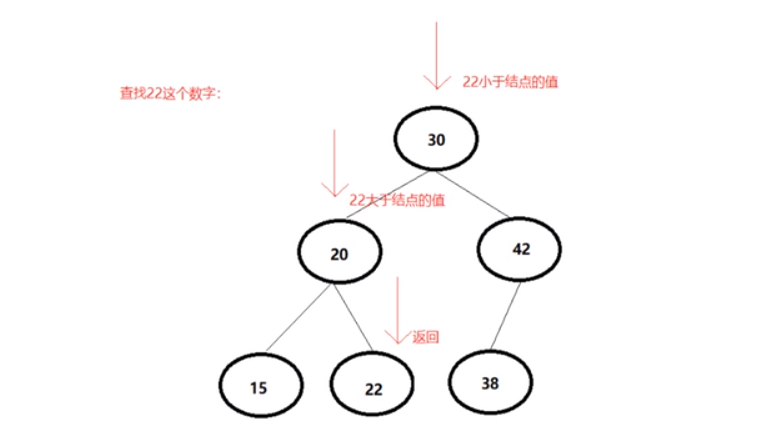
* + 1. 若左子树不空，则左子树上所有的值均小于它的根节点的值
    2. 若右子树不空，则右子树上所有的值均大于它的根节点的值
    3. 左右子树均分别为二叉排序树
    4. 没有相等的节点

结论：二叉查找树就是每个节点的值按照大小排列的二叉树，二叉查找树方便对节点的值进行查找



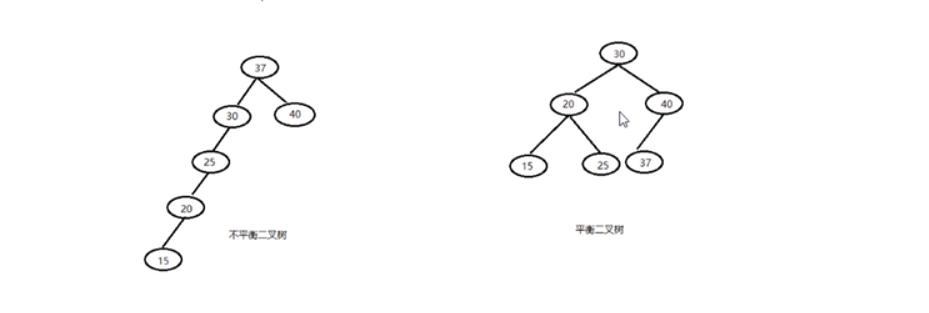
## 6.2 二叉查找树的查找规则

* 1. 从根节点开始，如果要查找的值等于节点的值，则返回。
  2. 如果要查找的值小于节点的值，那就在左子树中递归查找。
  3. 如果要查找的值大于节点的值，那就在左子树中递归查找。



## 6.3 平衡二叉树

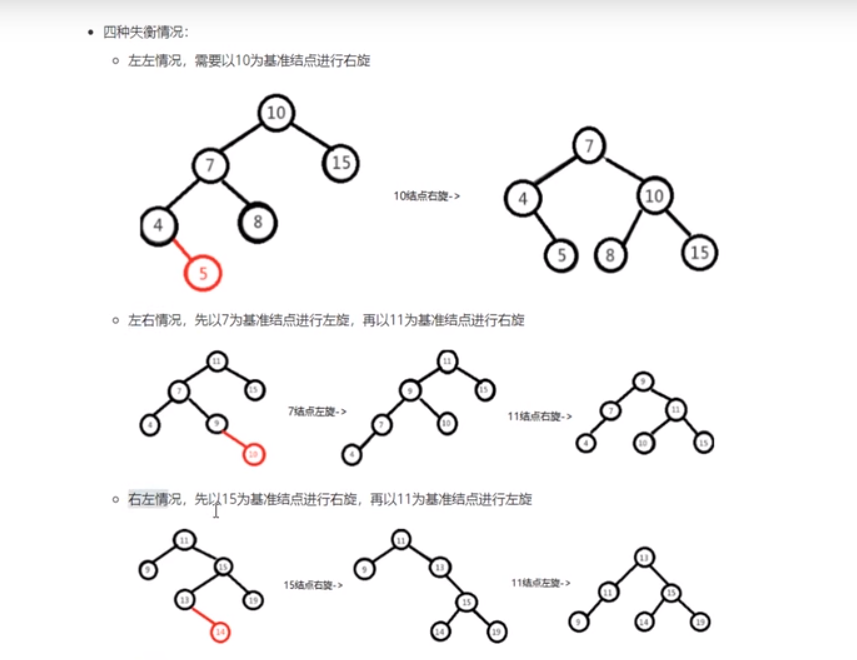
为了避免出现“瘸子”的现象，减少树的高度，提高我们的搜索效率，又存在一种树的结构“平衡二叉树”，它的左右两个子树高度差的绝对值不超过1，并且左右两个子树都是一颗平衡二叉树。

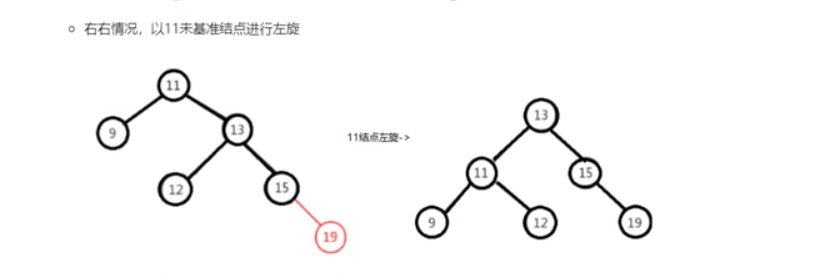


## 6.4 平衡二叉树的旋转

左旋：就是将节点的右支往左拉，右子节点变成父节点，并把晋升之后多余的左子节点让给降级节点的右子节点。

右旋：就是将节点的左支往右拉，左子节点变成父节点，并把晋升之后多余的右子节点让给降级节点的左子节点。





## 6.5 红黑树

### 6.5.1 概述

* 1. 红黑树是一种自平衡的二叉查找树。
  2. 红黑树的每一个节点都有存储位表示节点的颜色，可以是红或黑。
  3. 红黑树不是高度平衡的，它的平衡是通过红黑树的特性进行实现的。

### 6.5.2 特性

* 1. 每一个节点或是红色，或是黑色。
  2. 根节点必须是黑色。
  3. 每个叶子节点是黑色的。（Nil这里叶子节点，是指为空）。
  4. 如果一个节点是红色，则它的子节点必须是黑色。
  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）。

## 6.6 TreeMap

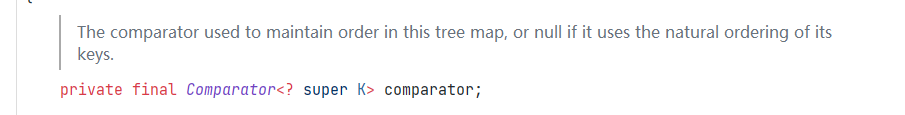
底层是红黑树的数据结构，元素中的键重复会覆盖；元素会按照大小顺序排序。

### 6.6.1 类的层级关系



### 6.6.2 类的相关属性

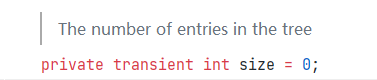
比较器用于维护此树形图中的顺序，如果比较器使用其键的自然顺序，则为null



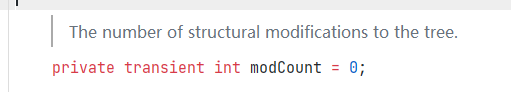
root节点



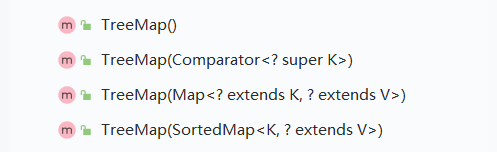
size容量



修改次数



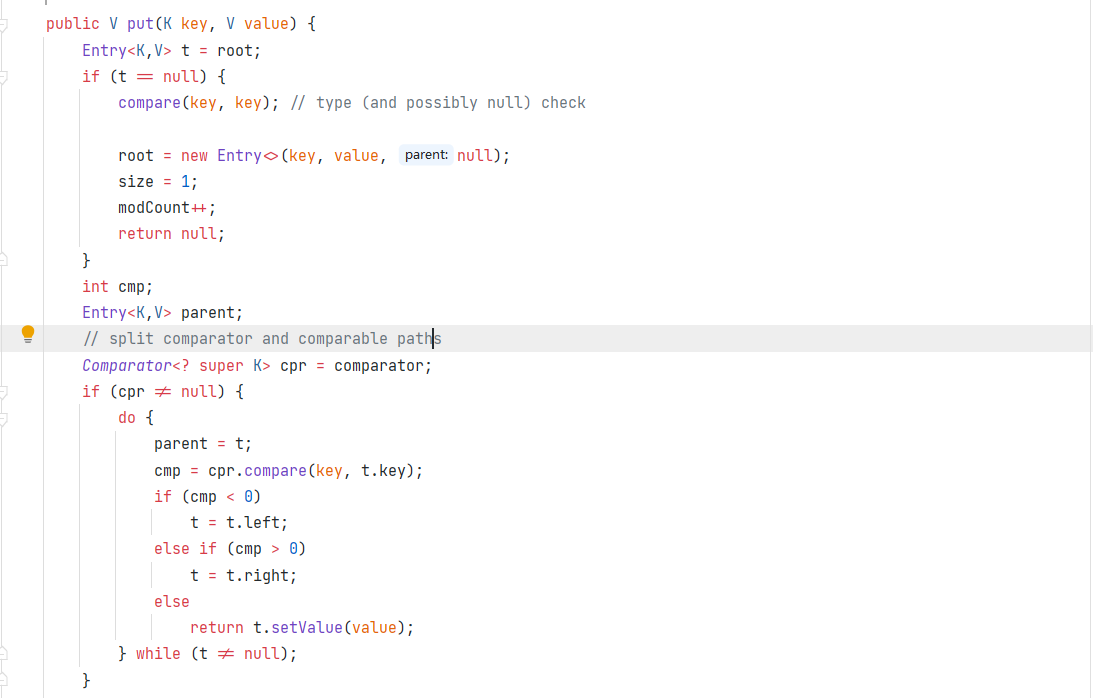
### 6.6.3 构造函数

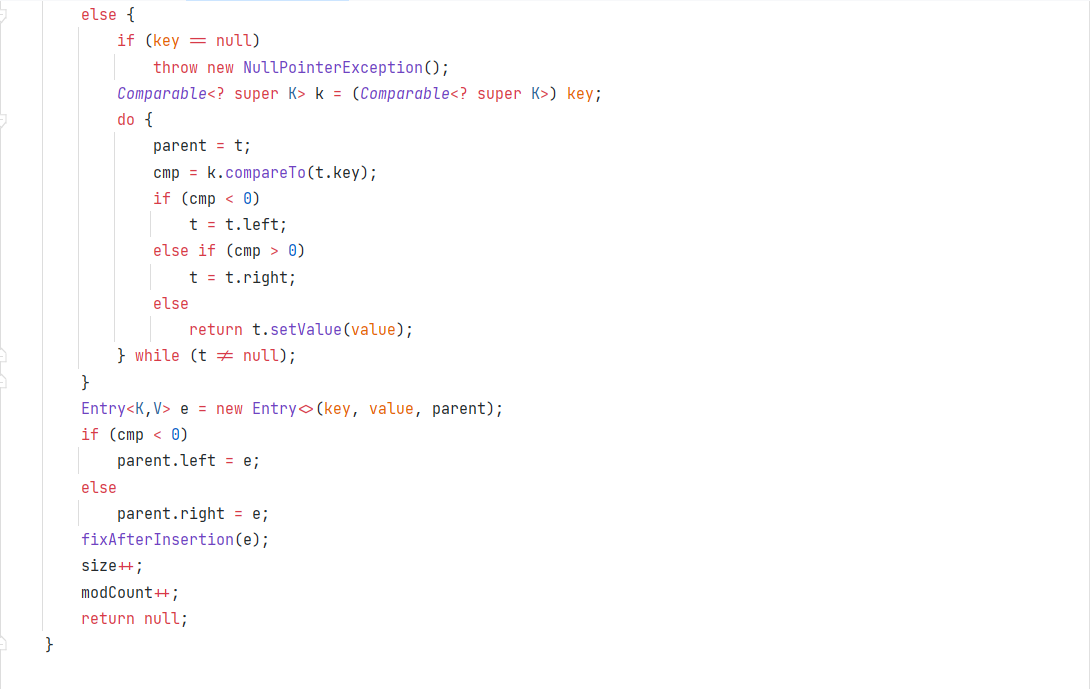


* 1. 无参构造
  2. 传入一个Comparator
  3. 传入一个Map
  4. 传入一个SortedMap

### 6.6.4 put方法底层原理

参考<https://www.jianshu.com/p/96e652ccf720>

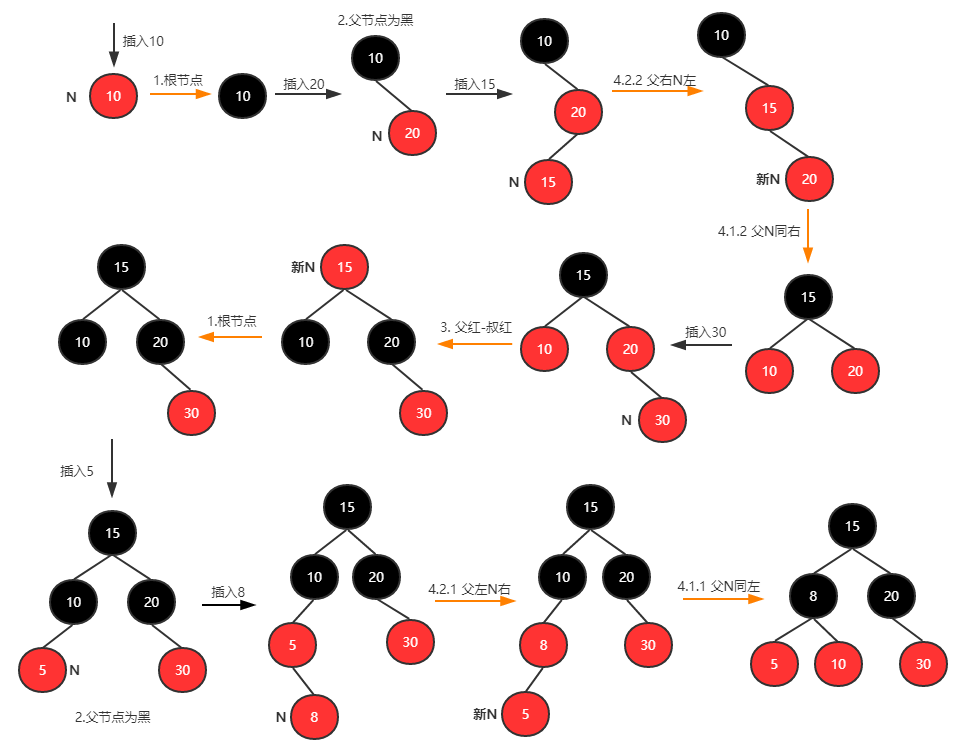




1. 首先，判断是否需要初始化根节点
   1. compare(key,key)主要是作类型检查
   2. 初始化一个Entry，它的parent节点是null
2. 如果传入的comparator不为空，则使用传入的comparator作为比较器
3. 从根节点开始循环遍历，当t为叶子节点时则退出循环遍历
4. 首先取root节点的key和当前key作比较
5. 小于0则将root的左子节点赋值给t，此t将准备进行下一次循环
6. 大于0则将root的右子节点赋值给t，此t将准备进行下一次循环
7. 等于0，则两个key相等，直接覆盖其值，并将之前的value返回出去
8. 如果传入的comparator为空，则使用默认的comparator作为比较器（思路与上雷同）
9. 如果循环结束之后还没找到相同的key并覆盖其值 的话，就取最后一次比较出来的结果，判断是大于0还是小于0，并插入到相应的子树中。

接下来是红黑树颜色的赋值，主要是用于维护红黑树的平衡



每次新增节点都需要维护红黑树的平衡， 下面是TreeMap中关于怎样维护平衡的介绍

1. 首先，每次新增节点默认颜色都是红色， 此处我们设新增的节点为x

2. 当满足条件时进入循环结构（x不能为空 & x不能是根节点& x的parent的颜色必须是红色）

注释：因为x默认为红色，只有当x的parent也是红色的时候才破坏了红黑树的平衡，因为不满足红黑树的性质（红色节点下面的两个子节点只能是黑色）

3. 这里分为两种情况

第一种情况

它的parent是它grad\_parent的左子节点

看它parent的右子节点（parent\_bro）的颜色

（1）如果是红色，则将它parent和parent\_bro的两个节点的颜色设置为黑色，grad\_parent设置为红色，即满足红色节点的子节点必须是黑色的性质。

（2）如果是黑色

a) 如果它是parent的左子节点，则将它parent节点设置为黑色，它的grad\_parent的节点设置为红色，然后以grad\_parent节点进行右旋，

b) 如果它是parent的右子节点，则将它parent节点进行左旋，然后以祖父进行右旋，然后设置grad\_parent为黑色，sup\_grad\_parent为红色。

第二种情况

它的parent是它grad\_parent的右子节点

看它grad\_parent的左子节点（parent\_bro）的颜色

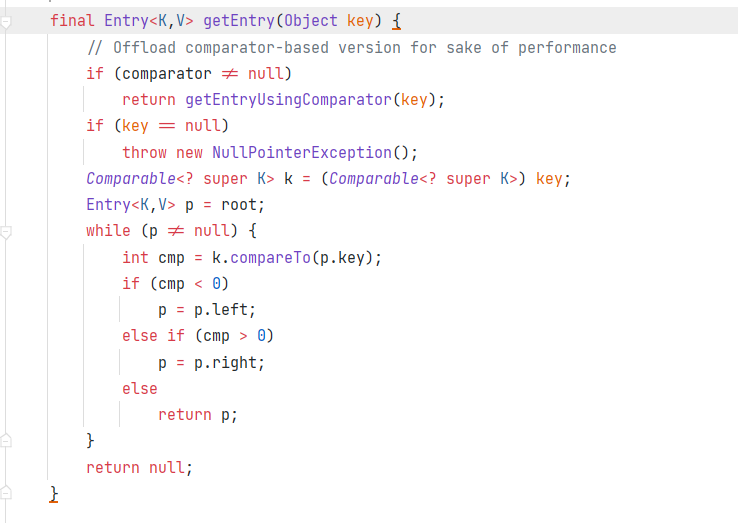
（1）如果是红色，则将它parent和parent\_bro的两个节点的颜色设置为黑色，grad\_parent设置为红色，即满足红色节点的子节点必须是黑色的性质。

（2）如果是黑色

a)如果它是parent的右子节点，则将它parent节点设置为黑色，它的grad\_parent的节点设置为红色，然后以grad\_parent节点进行左旋。

b)如果它是parent的左子节点，则将它parent节点几星右旋，然后以祖父进行左旋，然后设置grad\_parent为黑色，sup\_grad\_parent为红色。

### 6.6.5 get方法原理



1. 判断传入的comparator不为空，则使用传入的comparator进行查找；为空则使用默认的comparator。
2. 如果key为null，则抛出空指针。
3. 首先，把p指向根节点，和根节点的key作比较，大于根节点则p指向p的右节点；小于根节点则p指向p的左节点；等于p则直接返回p。
4. 由此递归的查找，直到找到为止。

### 6.6.6 remove方法原理

