# 树的族谱介绍

## 二叉树

1.1 二叉树是每个字节最多有两个子树的树结构。

二叉树的性质：

1. 在非空二叉树中，第i层的结点总数不超过 2^(i-1) , i>= 1;
2. 深度为h的二叉树最多有2^-1 个结点(h >= 1)，最少有h个结点；
3. 对于任意一颗二叉树，如果其叶结点数为N0，而度数为2的结点总数为N2，则N0=N2+1；
4. 具有n个结点的完全二叉树的深度 [log 2 n] + 1，（注:[ ] 表示向下取整）。
5. 有N个结点的完全二叉树各结点如果用顺序方式存储。则结点之间有如下关系：

若i为结点编号则 如果i>1, 则其父结点的编号为i>2;

如果2\*I <= N , 则其左儿子的编号为2\*I, 若2\*i>N, 则无左儿子

如果2\*i+1 <= N, 则其右儿子的结点编号为2\*i+1; 若2\*i+1 > N, 则无右儿子。

1.2 二叉排序树

二叉排序的定义：

1. 若左子树不为空，则左子树上所有结点的值均小于或者等于它的根结点的值。

2. 若右子树不为空，则右子树上所有结点的值均大于或者等于它的根结点的值。

3. 左右子树也分别为二叉排序树。

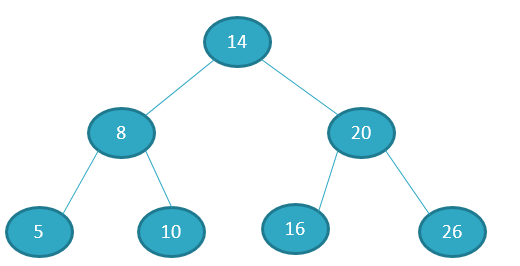
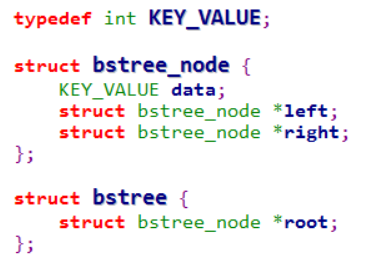


图1、二叉排序树

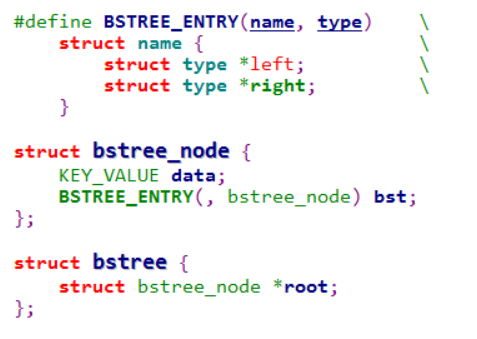
1.3 二叉树结构体定义：如下图所示



大部分在写二叉树的时候会采用如此定义，但是该定义方式直白，如此定义有几个方面不好的地方

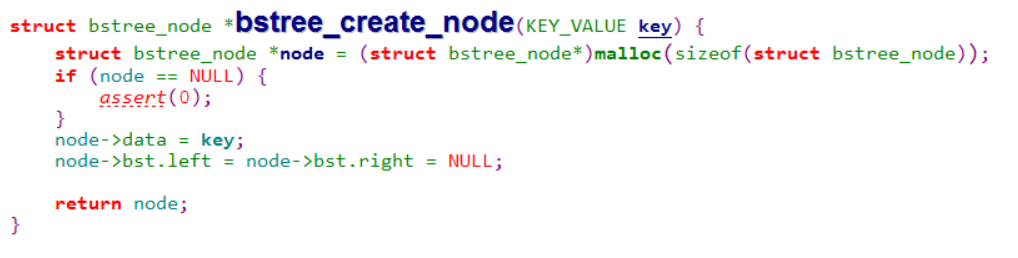
1. 不适合扩展。使用多个二叉树的结构体域的时候。
2. 代码不易于维护。

下面再定义一个扩展版的二叉树，更加实用。

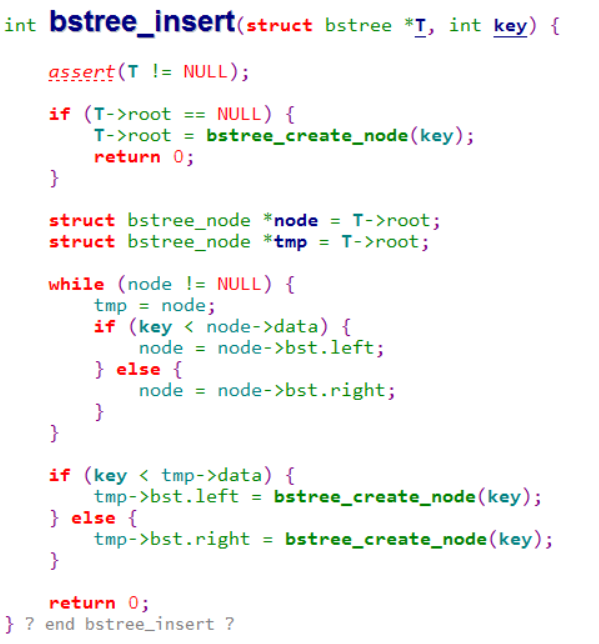


以上定义方式将树的左右子树分离出来，方便实用。

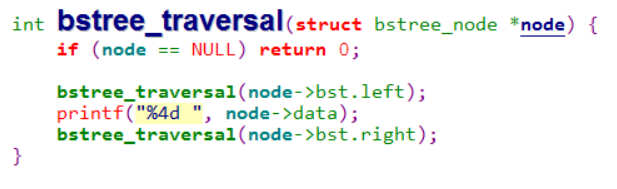
1.4 创建结构体



1.5 插入节点：



1.6 遍历树。



## 红黑树

* 1. 在学习红黑树之前，先来思考两个方案该如何设计

案例一、服务器端高并发IO的keep alilve方案，满足一下几个需求

1. 每个IO都是自己的时间戳
2. 每个IO收到自己的beat后，重置自己的定时器
3. 若IO定时没有收到beat，则执行IO的回调函数，并重置定时器
4. 若再次没有收到beat，销毁IO，注销定时器。

案例二、设计一个线程或者进程的运行体R与运行体调度器S的结构体

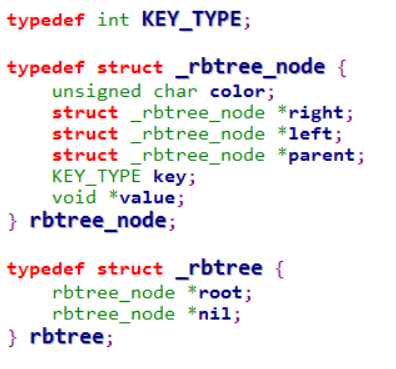
1. 运行体R：包含运行状态{新建，准备，挂起{IO等待读，IO等待写，睡眠，延时}， 退出}，运行体回调函数，回调参数
2. 调度器S：包含栈指针，栈大小，当前运行体
3. 调度器S：包含执行集合{就绪，延时，睡眠，等待}。

以上两个设计方案与红黑树有什么样的密切关联？此题先留在这里。当做每位读者的思考题

### 红黑树的性质与定义

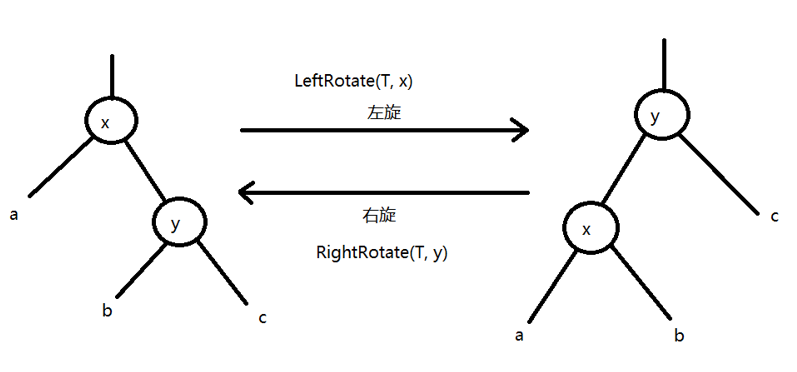
1. 每个结点是红的或者黑的
2. 根结点是黑色的
3. 每个叶子结点是黑色的
4. 如果每个结点是红色的，则它的两个儿子结点都是黑色的。
5. 对每个结点，从该结点到其子孙结点的所有路径上的包含相同数目的黑结点。

### 结构体定义

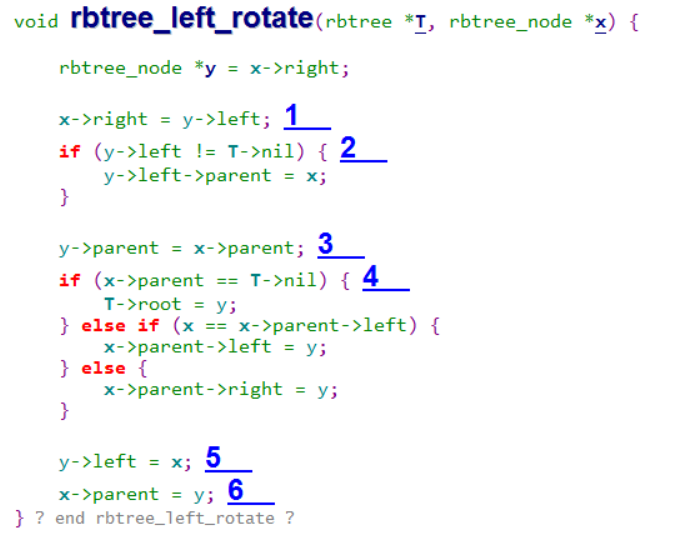


理解了定义以后，红黑树不同于二叉排序的重要原因就是能够自动均衡左右子树的高度。如何均衡左右子树的高度。

### 旋转



不论左旋还是右旋，都需要修改三个方向指针，六个指针变量。



### 添加新结点的维护

在新添加结点以前，有两个前提是确定的。

1. 未插入当前结点以前的红黑树是已经满足了红黑树的所有条件，是一颗红黑树的。
2. 当前结点是红色的。

从红黑树的5条性质，有两条推断

1. 结点更改后，只需要判断祖父结点到叶子结点的黑色结点一样多就满足性质了。
2. 由于当前结点是红色的，只有父节点是红色，才需要调整。所以能确定了当前结点x是红色的，父节点是红色的，祖父结点肯定是黑色的，那叔父结点是红色或者黑色。

基于以上的推断。分以下几种情况来讨论。假设：父节点到叶子结点所已经过的黑色结点数为N（不包含父结点），叔父结点到叶子结点经过的黑色结点数为M（不包含叔父结点）。

情况1. 叔父结点为红色。由于父结点是红色的，所以在当前结点没有插入以前，可以推断出M==N的。故父节点与叔父结点同时更改颜色即可。

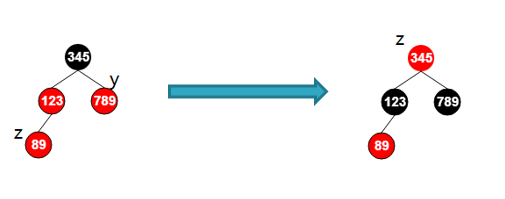
情况2. 叔父结点为黑色。由于父结点是红色的，所以可以推断N > M。在祖父结点的子树，整体需要向叔父结点旋转。（如果叔父结点是右子树，就需要右旋；如果叔父结点是左子树就需要左旋）

进行旋转是依赖当前结点是父结点的左子树还是右子树。

情况2.1． 如果父结点是祖父结点的左子树，叔父结点是祖父结点的右子树，且当前结点是父结点的右子树。此时颜色不需要更改，只需要以父结点为中心左旋。

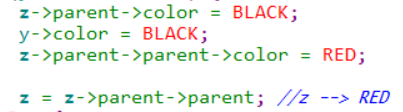
情况2.2．如果父结点是祖父结点的左子树，叔父结点是祖父结点的右子树，且当前结点是父结点的左子树。更改父结点颜色，与祖父结点颜色。以祖父结点为中心进行右旋。此段逻辑推理，帮助读者深入理解红色树在添加修复的时候，会分为三种情况。效果如下图所示：

情况1. 叔父结点为红色，父结点为红色。

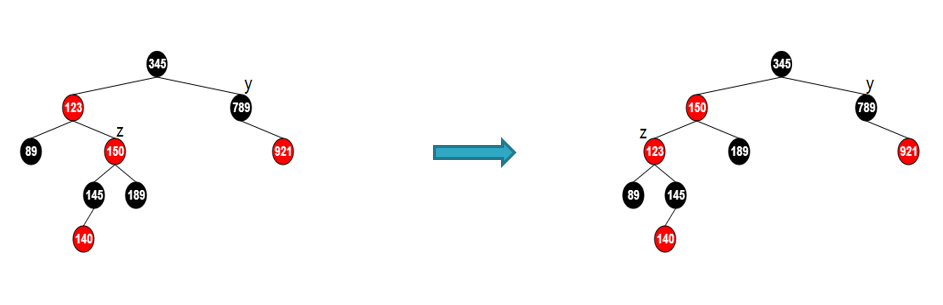




代码如下：



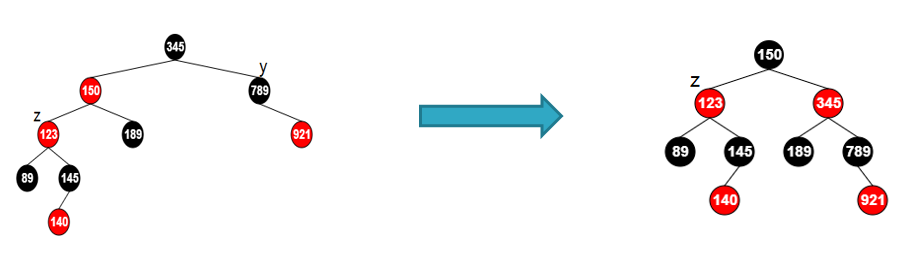
情况 2.1



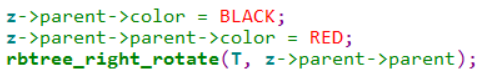
代码如下：



情况 2.2



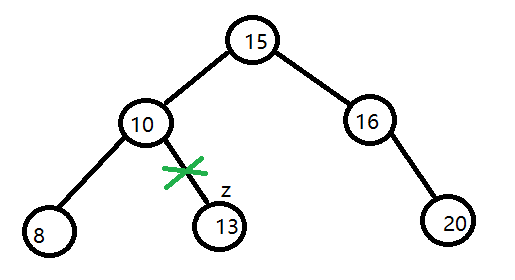
代码如下：



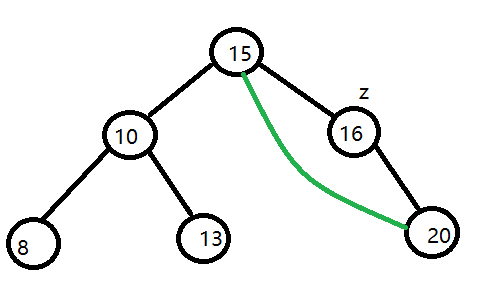
### 删除结点的维护

在理解删除结点之前，有一个情况大家一定要情况，就是删除结点z与释放的结点y，结点y是结点z的后继。而结点x是释放结点y的替补（左孩子或者右孩子）。一起分为三种情况，

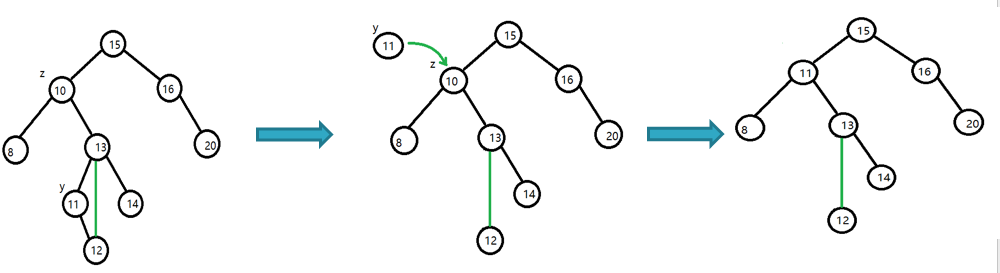
1. 没有左右子树



1. 有左子树或者右子树



1. 有左子树且有右子树



在删除结点进行维护，假设Y的父节点左子树的黑色结点高度为M，右子树的黑色结点高度为N，X为右子树。释放结点y，如果是红色的，是不会违反红黑树的黑高性质的。只有y为黑色的时候，是需要调整的。整体子树需要向y方向旋转的。是为了减低w子树的黑色结点高度。

情况1 w是x的兄弟结点。W是红色的。Y的删除使得，y子树的黑色结点高度减少。将x的父节点染成红色，w染成黑色，以x的父节点为中心右旋。

情况2 w是x的兄弟结点。W是黑色的。是为了减低w子树的黑色结点高度。

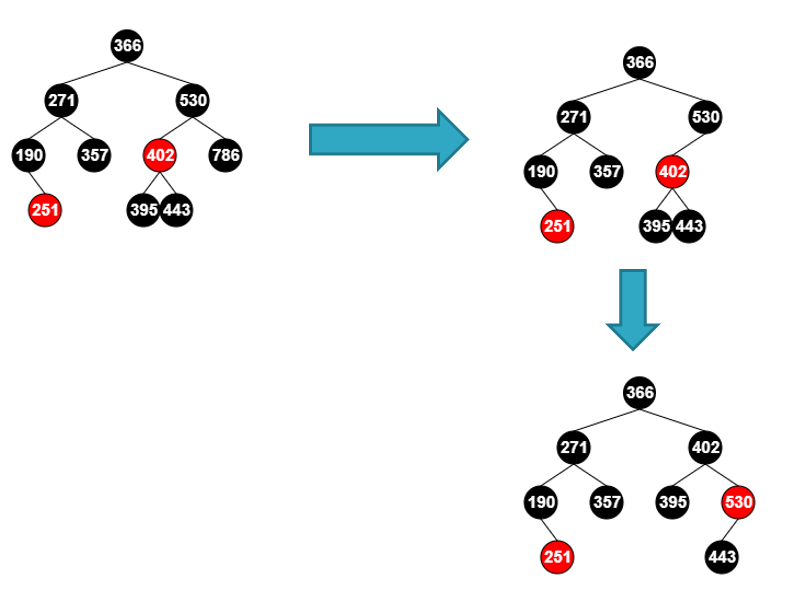
情况 2.1 W的两个子树为黑色。将w结点染为红色，w子树黑色结点高度减1。即可。

情况 2.2 W的左子树为黑色的。将w染成红色，w的右子树染成黑色，以w为中心左旋。

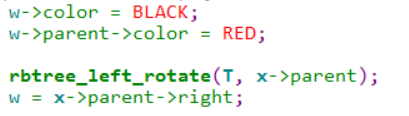
情况 2.3 W的左子树为红色的，W的右子树为黑色。将w染成x父节点颜色，x的父节点染成黑色，然后以父节点为中心右旋。

效果如下图所示：

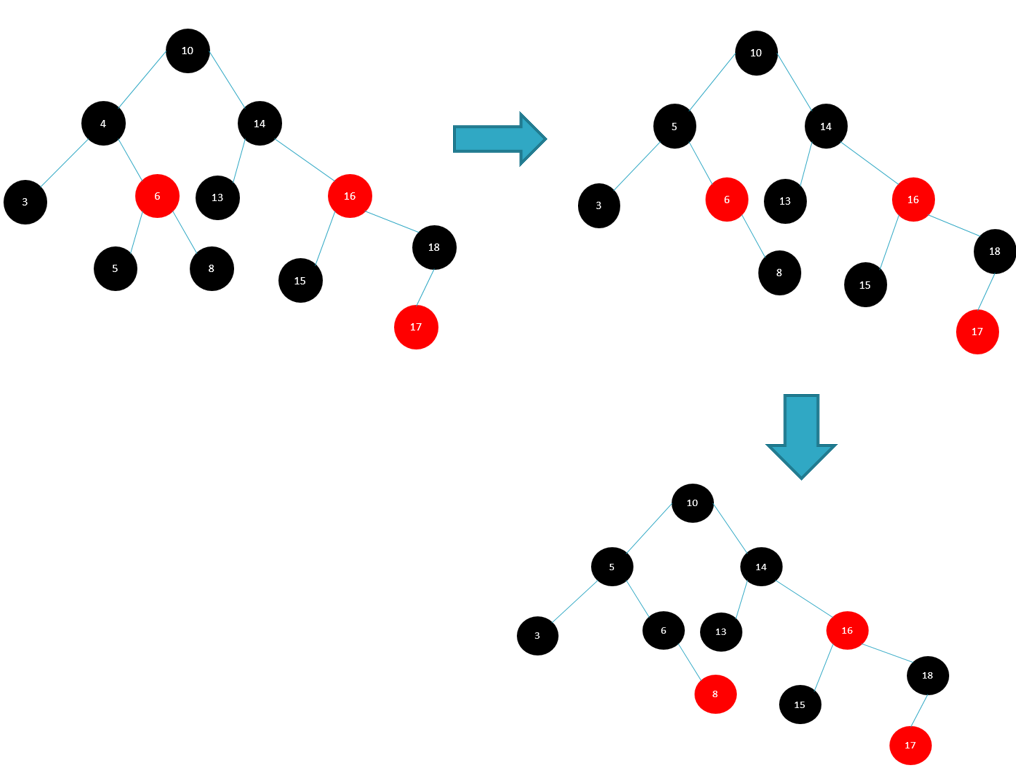
情况1：w是x的兄弟结点。W是红色的



代码实现：



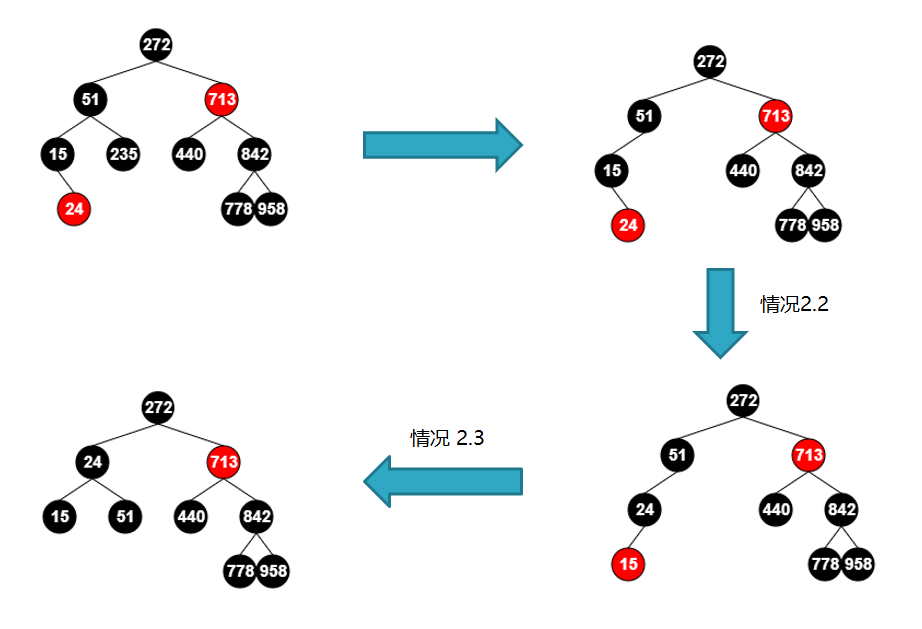
情况2.1：



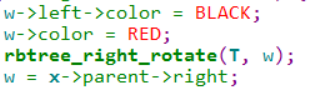
情况2.1 代码实现



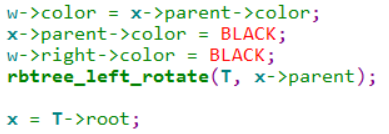
情况2.2与情况 2.3



情况 2.2 的代码



情况 2.3 的代码



# 三、B-树，B+树