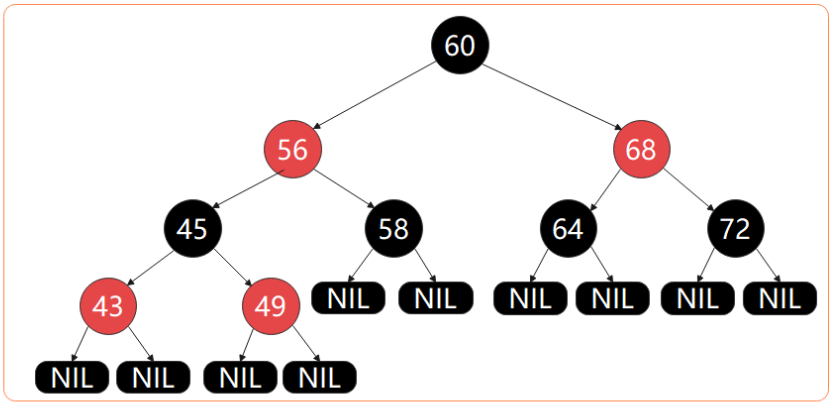
【第05话：红黑树这个问题太常见了】

请介绍一下红黑树(月薪1万的回答示范)

1、节点为红色或黑色
2、根节点为黑色
3、所有叶子节点必须为黑色
4、不允许连续两个红色节点
5、从任意节点到所有叶子节点过程中包含的黑色节点数量一致



红黑树

首先给大家说一下“请介绍一下红黑树"月薪1万的回答示范。

红黑树Red-Black Tree,是一种平衡因子可能大于1的平衡二叉树。红黑树除了是二叉查找树，同时还满足5点要求

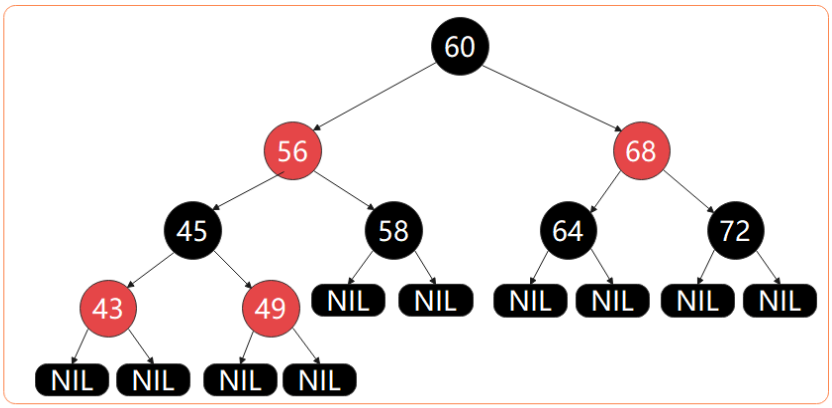
- 节点为红色或黑色
- 根节点为黑色
- 所有叶子节点必须为黑色
- 不允许连续两个红色节点
- 从任意节点到所有叶子节点过程中包含的黑色节点数量一致

Java中TreeMap就是红黑树的具体实现。

以上就是请介绍一下红黑树，月薪1万的回答示范。

请介绍一下红黑树(月薪1.5万的回答示范)

1、节点为红色或黑色
2、根节点为黑色
3、所有叶子节点必须为黑色
4、不允许连续两个红色节点
5、从任意节点到所有叶子节点过程中包含的黑色节点数量一致



红黑树

红黑树是为了防止AVL平衡二叉树频繁自旋，影响性能，而存在的。属于一种基本平衡或黑平衡的平衡二叉树。因为没有对平衡因子强制限制为1，所以自旋的次数更少。

红黑树首先满足二叉查找树的要求。即除叶子节点外，任意节点的左子树的值都小于节点值，右子树的值都大于节点值。

同时还要满足5点颜色方面要求：

- 节点为红色或黑色
- 根节点为黑色
- 所有叶子节点必须为黑色
- 不允许连续两个红色节点
- 从任意节点到所有叶子节点过程中包含的黑色节点数量一致

红黑树为了保证颜色的5点要求，需要进行的左旋或右旋及颜色变化。

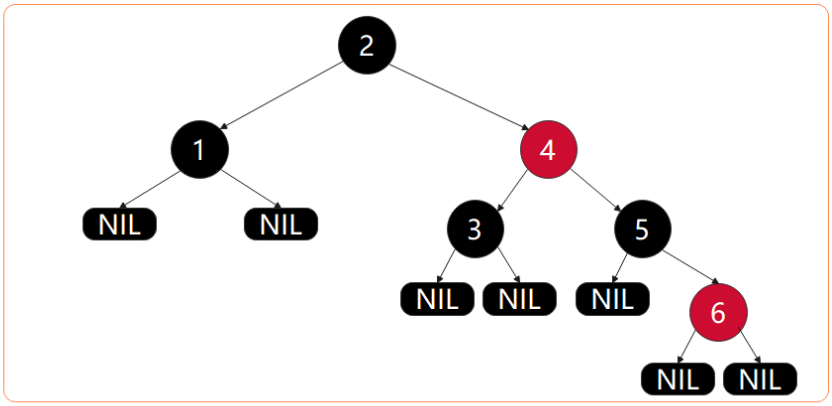
其中颜色所有叶子节点必须是黑色，不需要在操作节点时频繁考虑，这点是为了和任意节点到叶子节点包含黑色节点数量一致时使用的。要求根节点为黑色好实现，只要当前为空树，则直接插入一个黑色节点就可以了。只要有了根节点，插入的节点就直接设置为红色，这样一下就满足颜色要求1节点为红色或黑色，且对要求5从任意节点到所有叶子节点过程中包含的黑色节点数量一致没有影响。但是如果新增节点的父节点为红色时，就需要判断叔父节点颜色、父节点在左子树还是右子树，来进行旋转，旋转后还需要进行变色操作，如果变色后还是出现连续两个红色节点，对爷爷节点重复上面操作，甚至可能对根节点进行旋转。

而删除时，都是使用删除节点左子树中最大值进行补充被删除节点。删除后也会判断是否满足5点颜色要求，如果不满足依然需要旋转和改变节点颜色。

以上就请介绍一下红黑树月薪1.5万的回答示范

请介绍一下红黑树(月薪2万+的回答示范)

1、节点为红色或黑色
2、根节点为黑色
3、所有叶子节点必须为黑色
4、不允许连续两个红色节点
5、从任意节点到所有叶子节点过程中包含的黑色节点数量一致
6、最长路径不会超过最短路径2倍



红黑树

红黑树是为了防止AVL平衡二叉树频繁自旋，影响性能，而存在的。属于一种基本平衡或黑平衡的平衡二叉树。因为没有对平衡因子强制限制为1，所以自旋的次数更少。虽然没有严格限制平衡因子。但是因节点颜色要求。最长路径不会大于最短路径的两倍。因为保证任意路径黑色节点数量相同情况，且不能连续出现两个红色节点，只能在两个黑色节点中间存在一个红色节点。所以红黑树的查询，时间复杂度为O(logn)。

红黑树实现时在新增和删除操作需要考虑到左旋、右旋、变色问题。而且里面又细分为多种情况。所以给人的感觉比较复杂。只要清晰记忆颜色要求，谨记所有的旋转和变色都是为了达到颜色要求就可以了。

5点颜色要求：

- 节点为红色或黑色
- 根节点为黑色
- 所有叶子节点必须为黑色

- 不允许连续两个红色节点
- 从任意节点到所有叶子节点过程中包含的黑色节点数量一致

先说一下新增。借助动画USFCA的动画演示配置回答。

新增时，如果当前为空树，直接新增一个黑色的根节点。例如：数字100。

以后新增的所有节点，默认都是红色，因为红色不影响树路径中黑色节点的数量。减少旋转和改色的可能性。

例如插入数字200，会在根节点下添加红色右子节点。添加50会在根节点左侧添加红色子节点。

每次新增都是添加到叶子节点中，再次添加300，会从根节点向下找到需要放置位置，发现父节点是红色，叔父节点也是红色，只需要把父节点和叔父节点直接变为黑色就可以了，爷爷节点变为红色，但是爷爷节点为根节点，此处不变色。这也是红黑树中除了父节点是黑色以外，唯一不需要旋转的情况。

再次添加400，依然是从根节点出发，找到节点放置位置。发现父节点为红色，叔父节点为黑色，且父节点在右子树，当前节点也在右子树，需要对父节点左旋。让爷爷节点变为父节点的左子节点。

再次添加500时，还是从根节点往下进行找，找到存放节点的位置，由于父节点是红色，叔父节点也是红色，改变父节点和叔父节点颜色为黑色，同时修改爷爷节点为红色

再添加450时，放在红色父节点500左子节点。由于父节点为红色叔父节点为黑色，且父节点在右子树，当前节点在左子树，需要右旋，让当前节点变为父节点的父节点，父节点变为当前节点的右子节点。这是发现依然有连续两个红色节点。对节点450进行左旋，让400变为450的左子节点，并让450为黑色，400为红色。父节点在右子树的情况就是这样。父节点在左子树时和当前情况类似。改变颜色还是这个规则，主要是旋转时反方向旋转。

而删除时，如果要删除300，从根节点出发，找到要删除的节点，把左子树中最大节点拿过来。然后检查是否满足任意路径黑色节点数量相同。发现不同，对节点450进行左旋，让200成为400的左子节点，400成为200的右子节点，最后变色。

所以红黑树中无论新增还是删除时，都要时刻记忆任意路径黑色节点数量相同，以保证查询，删除，新增时间复杂度为O(logn)就可以了。

在Java中TreeMap是对红黑树的实现，Java 8后HashMap底层也有红黑树。当需要使用红黑树时直接使用TreeMap即可。

以上就是请介绍一下红黑树月薪2万+的回答示范