

平衡二叉树

Balanced Binary Tree, AVL

G. M. Adelson-Velskey and E. M. Landis

平衡二叉树的定义

G. M. Adelson-Velsky和E. M. Landis

平衡二叉树(Balanced Binary Tree), 简称平衡树(AVL树)——树上任一结点的左子树和右子树的高度之差不超过1。

结点的平衡因子=左子树高-右子树高。

平衡二叉树结点的平衡因子的值只可能是-1、0或1。

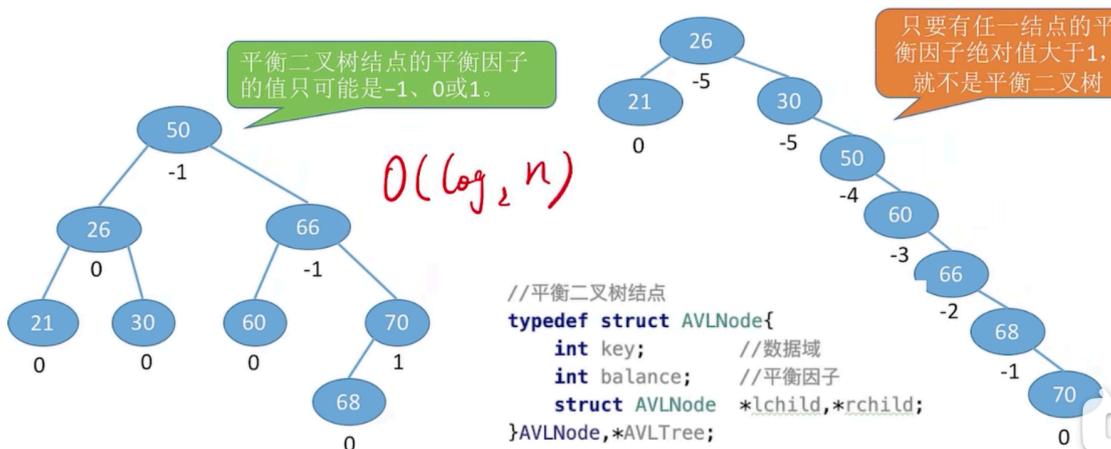
$O(\log_2 n)$

//平衡二叉树结点

```
typedef struct AVLNode{  
    int key; //数据域  
    int balance; //平衡因子  
    struct AVLNode *lchild,*rchild;  
}AVLNode,*AVLTree;
```

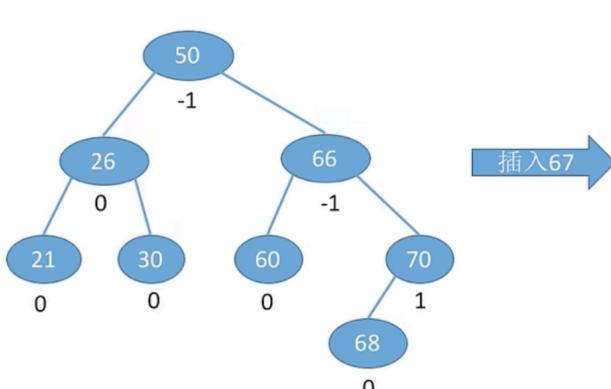
只要有任一结点的平衡因子绝对值大于1,就不是平衡二叉树

王道考研/CSKAOYAN.COM

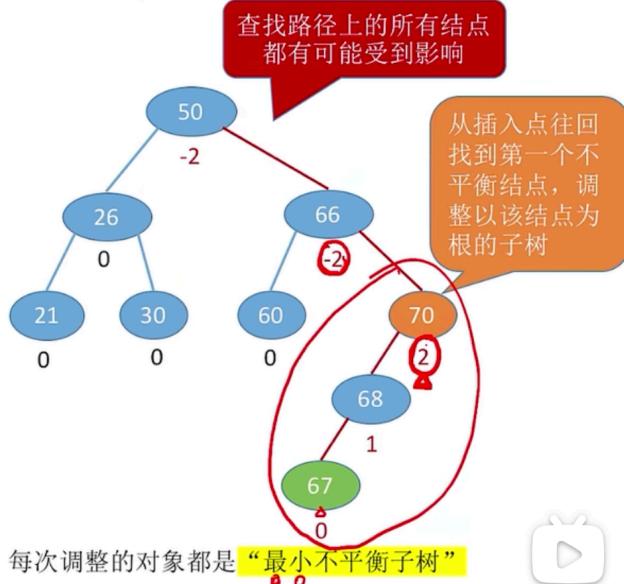


平衡二叉树的插入

在二叉排序树中插入新结点后, 如何保持平衡?

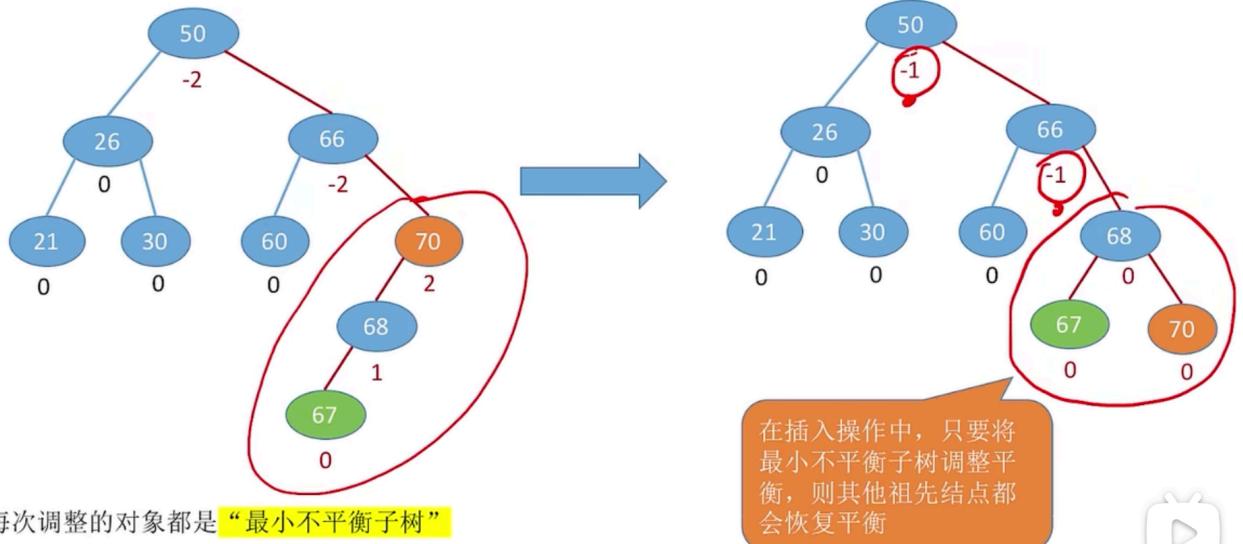


插入67





平衡二叉树的插入



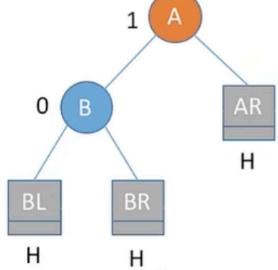
调整最小不平衡子树

调整最小不平衡子树A

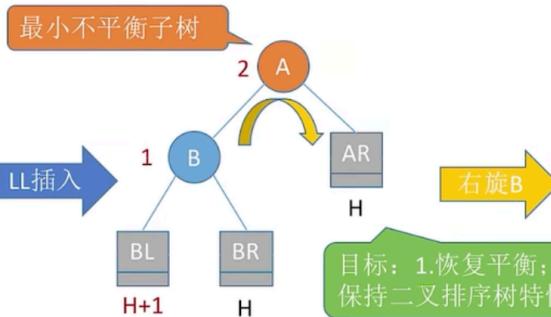
- LL ⊖ 在A的左孩子的左子树中插入导致不平衡
- RR ⊖ 在A的右孩子的右子树中插入导致不平衡
- LR ⊖ 在A的左孩子的右子树中插入导致不平衡
- RL ⊖ 在A的右孩子的左子树中插入导致不平衡

bilibili

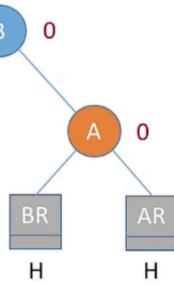
调整最小不平衡子树（LL）



思考：为什么要假定所有子树的高度都是H？



目标：1.恢复平衡；2.保持二叉排序树特性



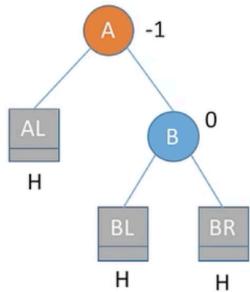
二叉排序树的特性：左子树结点值 < 根结点值 < 右子树结点值

$BL < B < HR < A < AR$

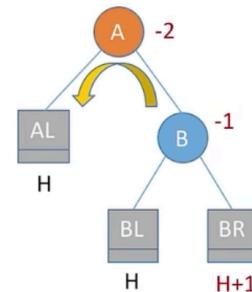
1) LL平衡旋转（右单旋转）。由于在结点A的左孩子（L）的左子树（L）上插入了新结点，A的平衡因子由1增至2，导致以A为根的子树失去平衡，需要一次向右的旋转操作。将A的左孩子B向右上旋转代替A成为根结点，将A结点向右下旋转成为B的右子树的根结点，而B的原右子树则作为A结点的左子树。

bilibili

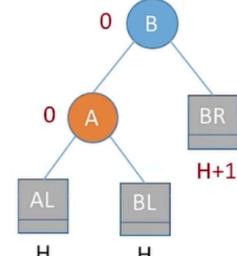
调整最小不平衡子树（RR）



RR插入



左旋B



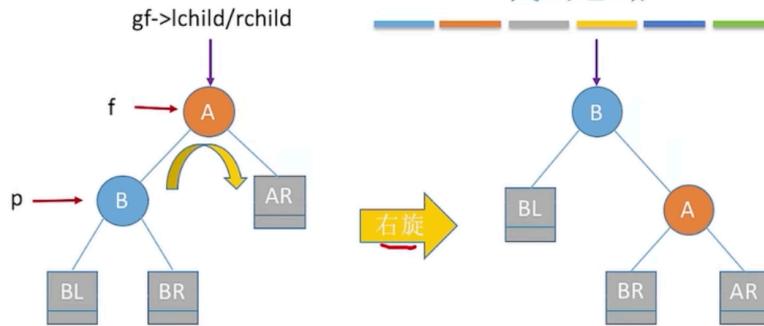
二叉排序树的特性：左子树结点值 < 根结点值 < 右子树结点值

$AL < A < BL < B < BR$

2) RR平衡旋转（左单旋转）。由于在结点A的右孩子（R）的右子树（R）上插入了新结点，A的平衡因子由-1减至-2，导致以A为根的子树失去平衡，需要一次向左的旋转操作。将A的右孩子B向左上旋转代替A成为根结点，将A结点向左下旋转成为B的左子树的根结点，而B的原左子树则作为A结点的右子树。

B站视频 bili

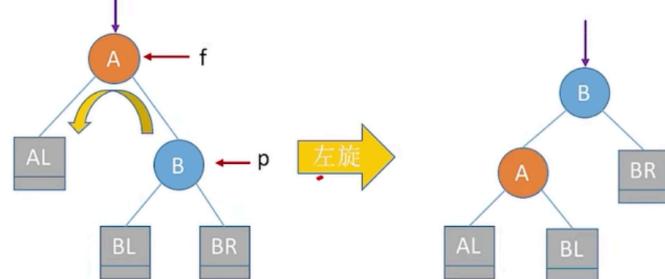
代码思路



实现 f 向右下旋转, p 向右上旋转:
其中 f 是爹, p 为左孩子, gf 为 f 的爹
① f->lchild = p->rchild;
② p->rchild = f;
③ gf->lchild/rchild = p;

BL < B < HR < A < AR

左旋、右旋操作后可以
保持二叉排序树的特性

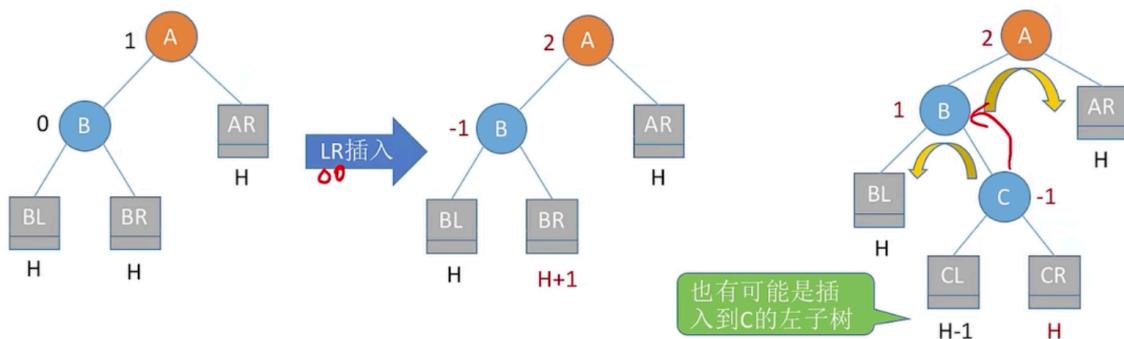


实现 f 向左下旋转, p 向左上旋转:
其中 f 是爹, p 为右孩子, gf 为 f 的爹
① f->rchild = p->lchild;
② p->lchild = f;
③ gf->lchild/rchild = p;



B站视频 bili

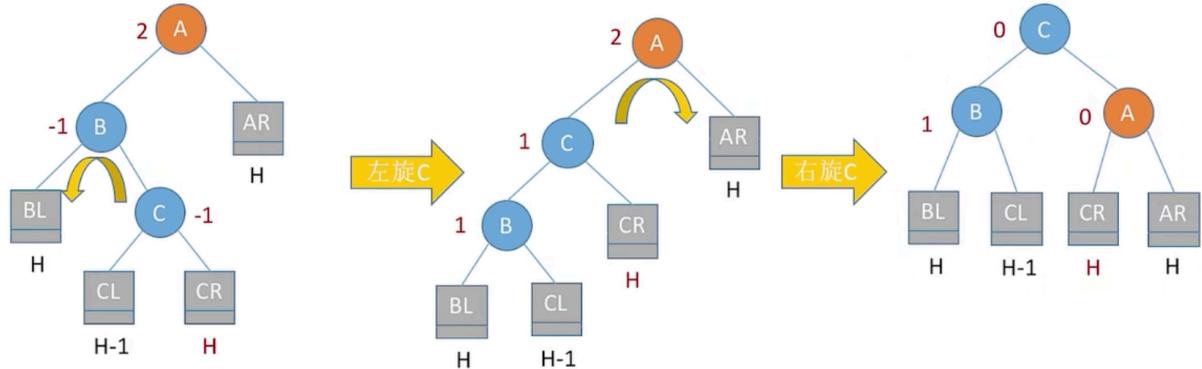
调整最小不平衡子树 (LR)



3) LR平衡旋转 (先左后右双旋转)。由于在 A 的左孩子 (L) 的右子树 (R) 上插入新结点, A 的平衡因子由 1 增至 2, 导致以 A 为根的子树失去平衡, 需要进行两次旋转操作, 先左旋转后右旋转。先将 A 结点的左孩子 B 的右子树的根结点 C 向左上旋转提升到 B 结点的位置, 然后再把该 C 结点向右上旋转提升到 A 结点的位置。

王道考研 b站

调整最小不平衡子树 (LR)

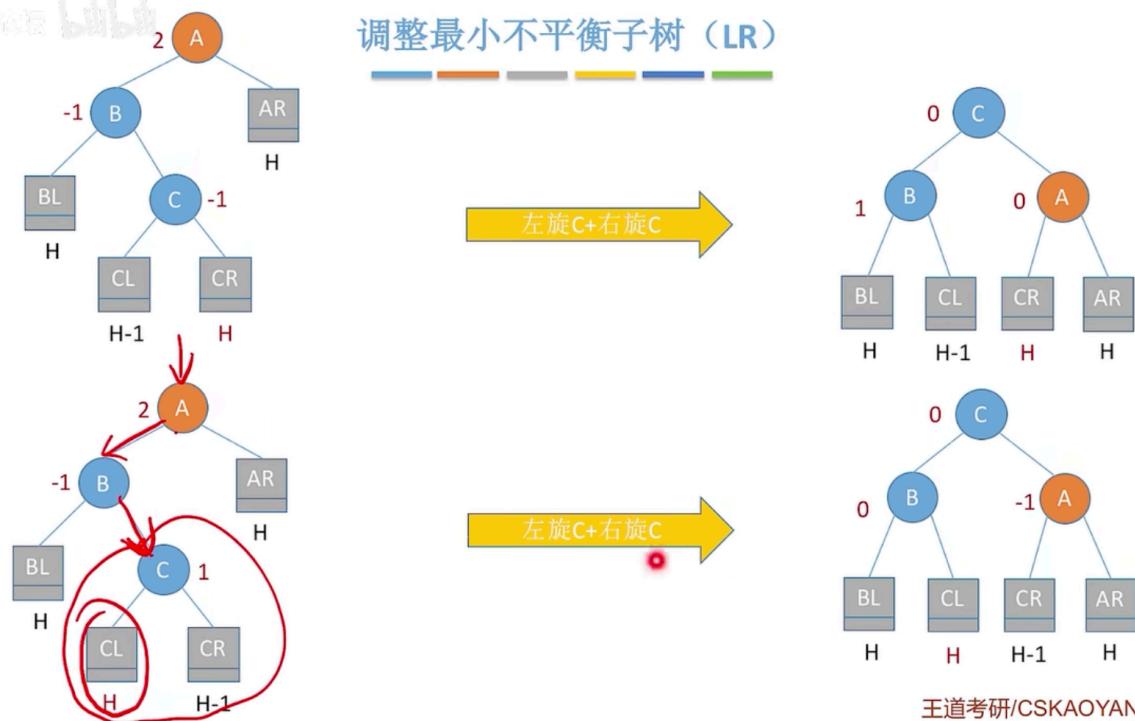


BL < B < CL < C < CR < A < AR



王道考研 b站

调整最小不平衡子树 (LR)

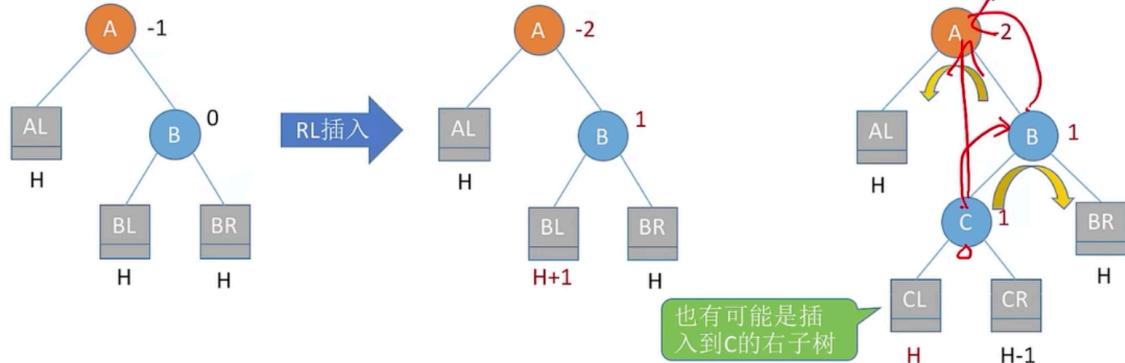


王道考研/CSKAOYAN.COM

bilibili

调整最小不平衡子树（RL）

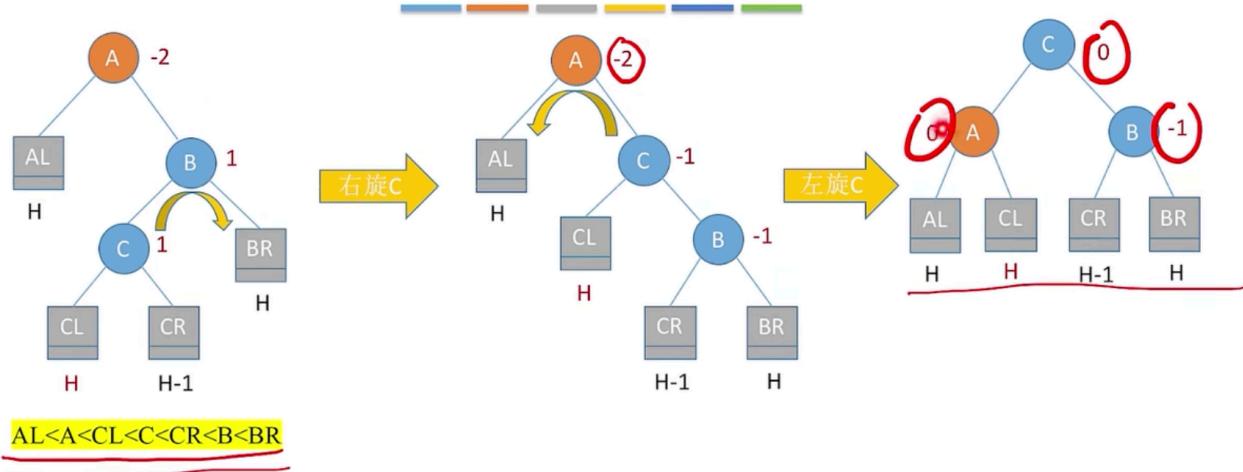
AL < A < CL < C < CR < B < BR



4) RL平衡旋转（先右后左双旋转）。由于在A的右孩子（B）的左子树（CL）上插入新结点，A的平衡因子由-1减至-2，导致以A为根的子树失去平衡，需要进行两次旋转操作，先右旋转后左旋转。先将A结点的右孩子B的左子树的根结点C向右上旋转提升到B结点的位置，然后再把该C结点向左上旋转提升到A结点的位置。

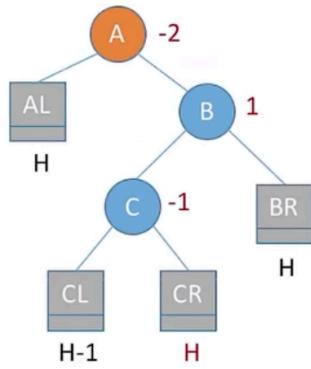
bilibili

调整最小不平衡子树（RL）

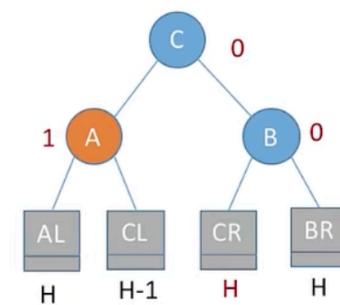


道家考研

调整最小不平衡子树 (RL)



右旋C+左旋C



工道考研/CSSKAQYAI

道家考研

调整最小不平衡子树

只有左孩子
才能右上旋

实现 f 向右下旋转, p 向右上旋转:
其中 f 是爹, p 为左孩子, gf 为 f 的爹
 ① f->lchild = p->rchild;
 ② p->rchild = f;
 ③ gf->lchild/rchild = p;

调整最小不平衡子树 A

- LL** 在 A 的左孩子的左子树中插入导致不平衡
调整: A 的左孩子结点右上旋
- RR** 在 A 的右孩子的右子树中插入导致不平衡
调整: A 的右孩子结点左上旋
- LR** 在 A 的左孩子的右子树中插入导致不平衡
调整: A 的左孩子的右孩子 先左上旋再右上旋
- RL** 在 A 的右孩子的左子树中插入导致不平衡
调整: A 的右孩子的左孩子 先右上旋后左上旋

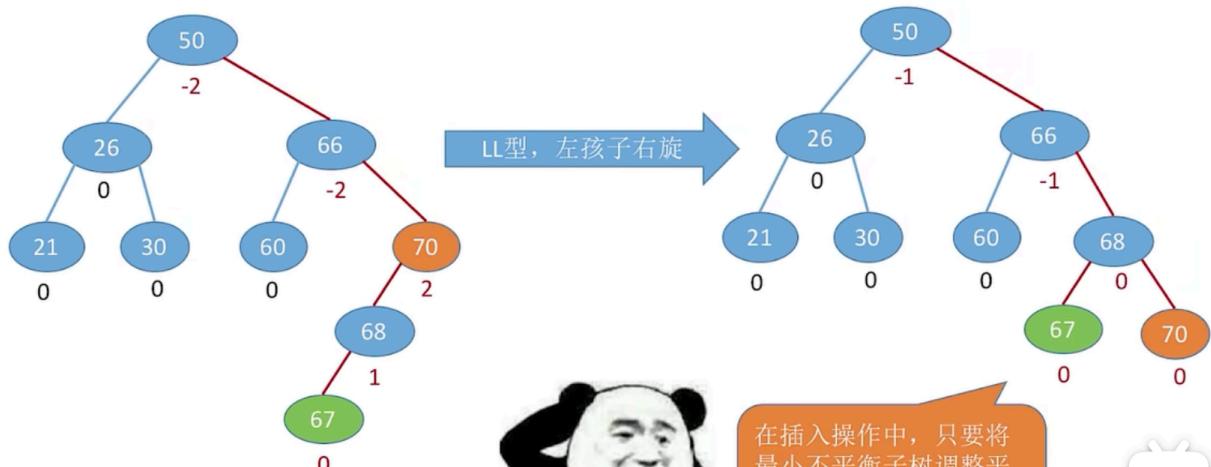
实现 f 向左下旋转, p 向左上旋转:
其中 f 是爹, p 为右孩子, gf 为 f 的爹
 ① f->rchild = p->lchild;
 ② p->lchild = f;
 ③ gf->lchild/rchild = p;

只有右孩子
才能左上旋



填个坑

填个坑



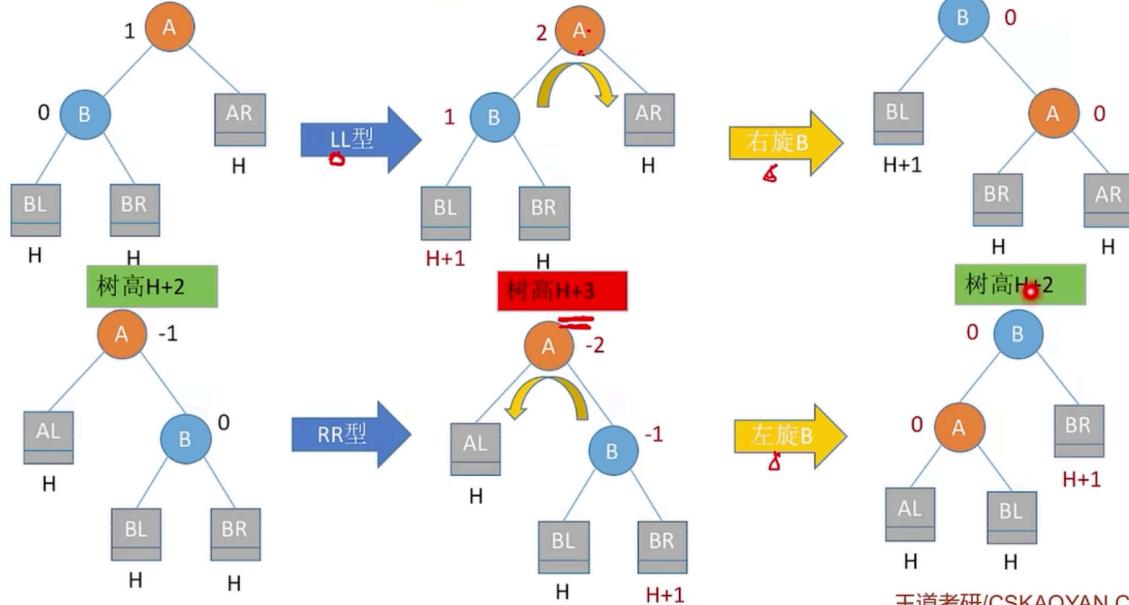
每次调整的对象都是“最小不平衡子树”

在插入操作中，只要将
最小不平衡子树调整平衡，则其他祖先结点都
会恢复平衡



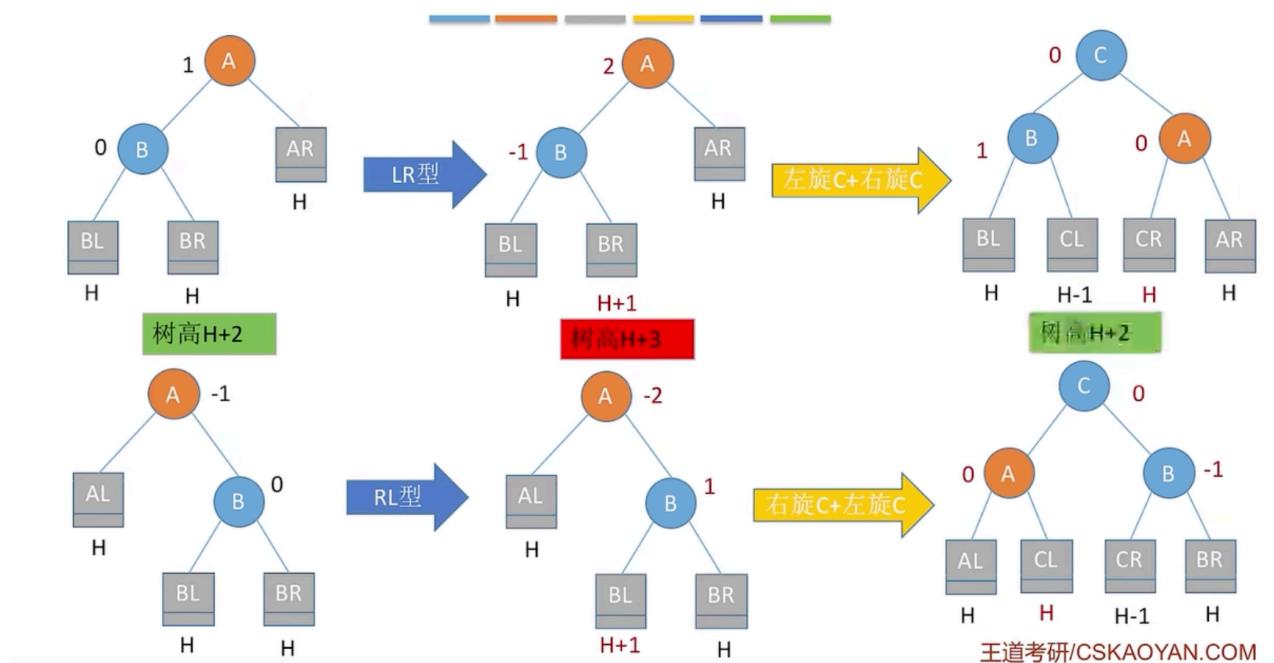
填个坑

填个坑



bilibili

填个坑



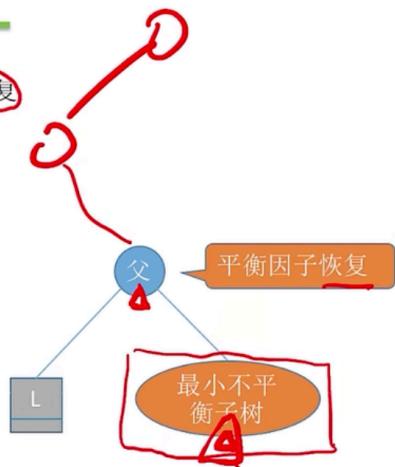
bilibili

填个坑

插入操作导致“最小不平衡子树”高度+1，经过调整后高度恢复

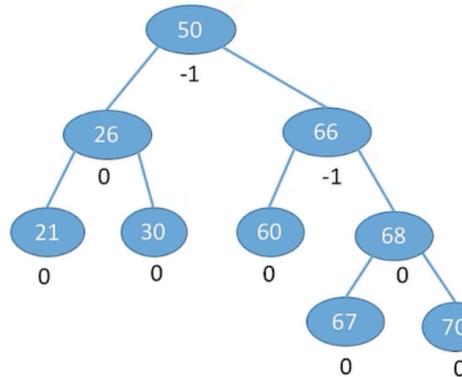


在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

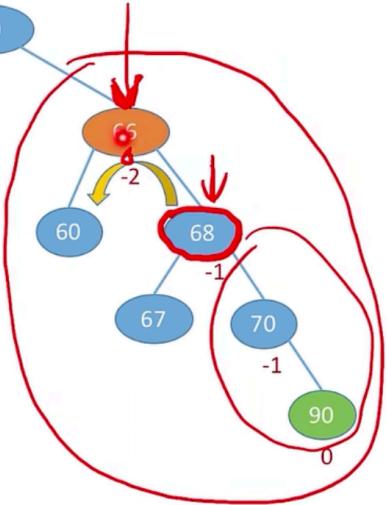
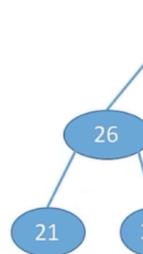


王道数据结构

练习

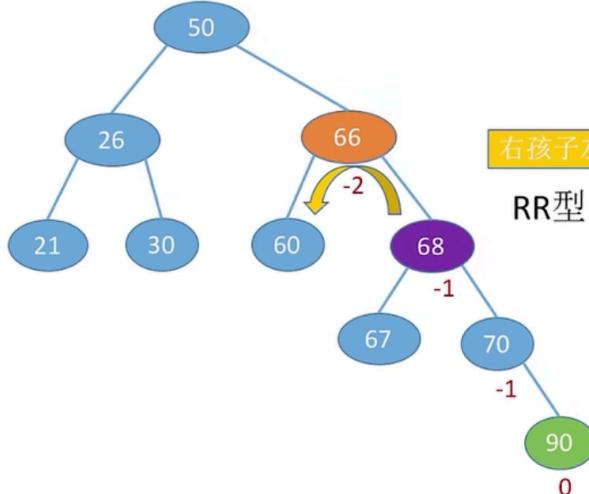


插入 90
RR型

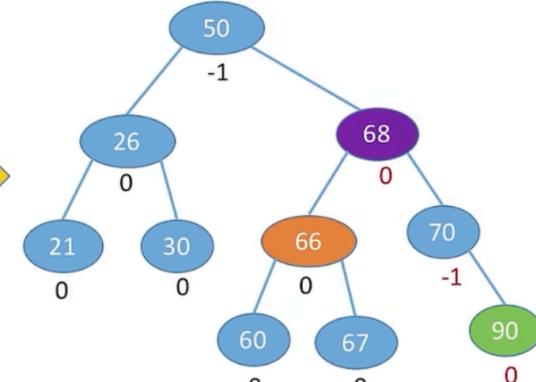


王道数据结构

练习 ↴



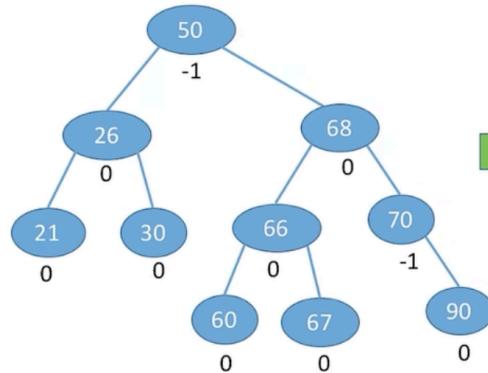
右孩子左旋
RR型



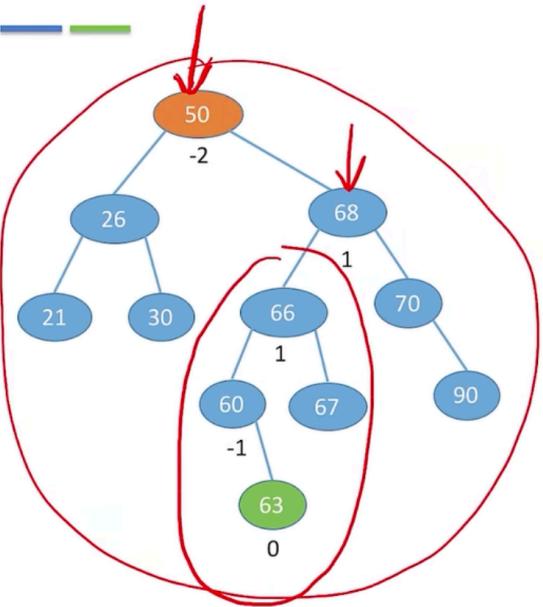
注意检查：是否符合 左<根<右

上机实验 二叉树

练习

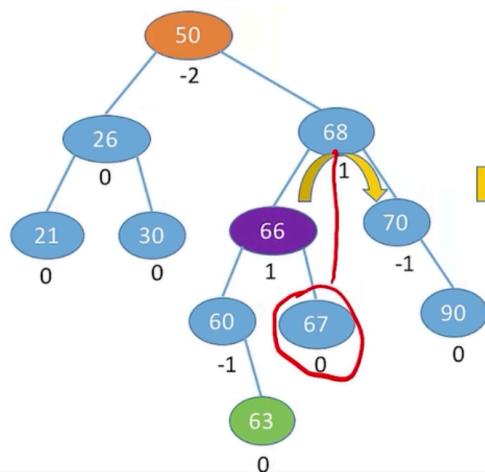


插入 63
RL型

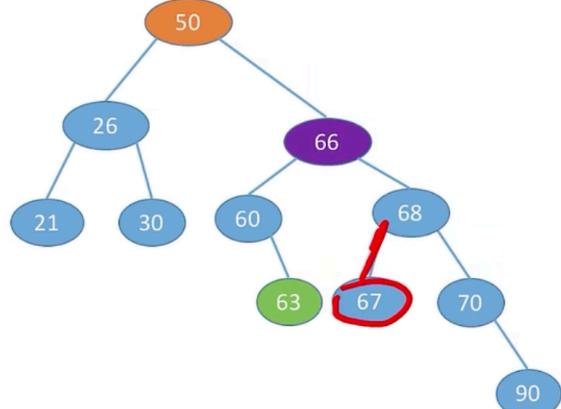


上机实验 二叉树

练习

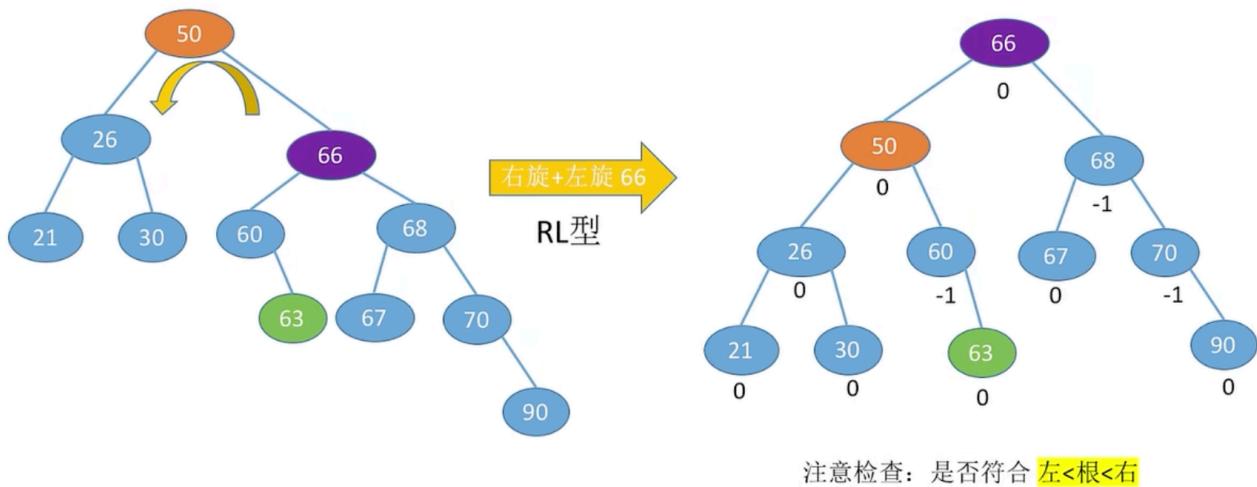


右旋+左旋 66
RL型



上道面试 b站

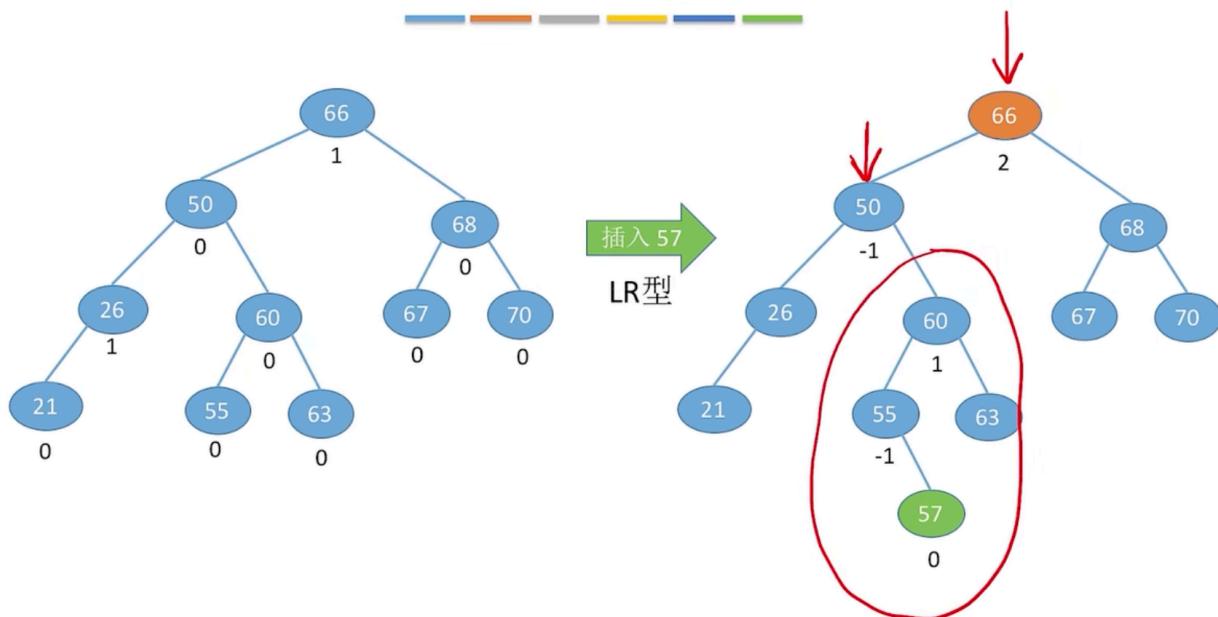
练习

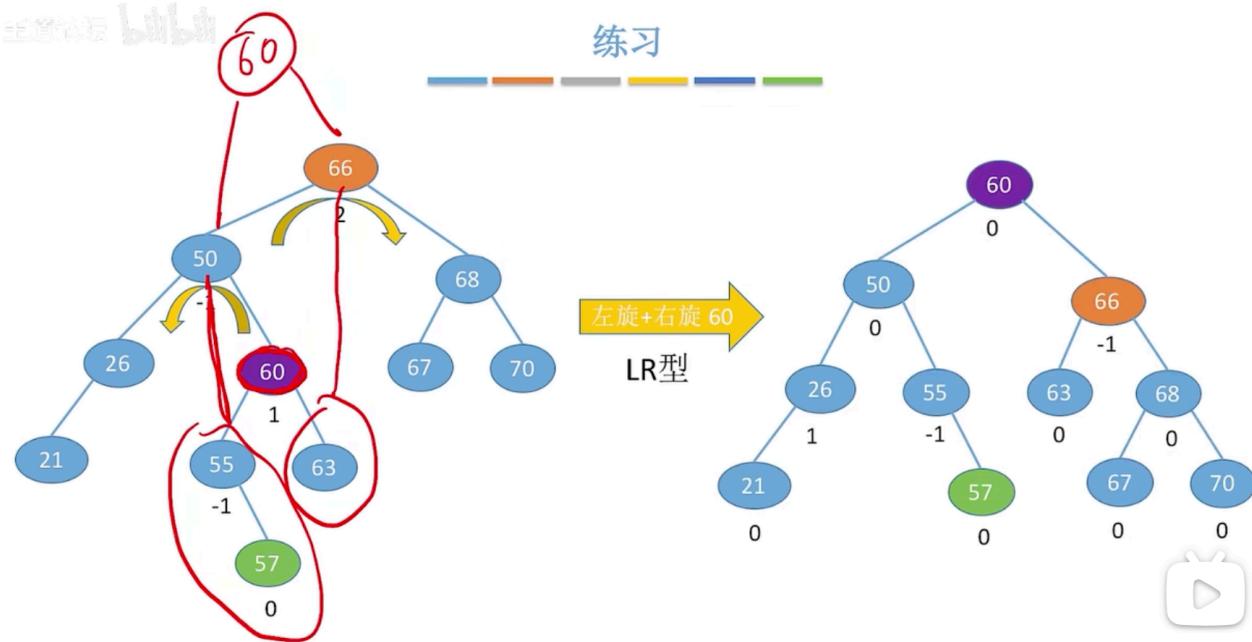


注意检查：是否符合 左<根<右

上道面试 b站

练习





查找效率分析

若树高为 h , 则最坏情况下, 查找一个关键字最多需要对比 h 次, 即查找操作的时间复杂度不可能超过 $O(h)$

平衡二叉树——树上任一结点的左子树和右子树的高度之差不超过1。

假设以 n_h 表示深度为 h 的平衡树中含有的最少结点数。

则有 $n_0 = 0$, $n_1 = 1$, $n_2 = 2$, 并且有 $n_h = n_{h-1} + n_{h-2} + 1$

b = 0

NULL

$$\kappa_0 = 0$$

$$h = 1$$

1

1

$$h = 2$$

1

$$n = 3$$

$n_3 = 4, n_4 = 7, n_5 = 12$



查找效率分析



若树高为 h , 则最坏情况下, 查找一个关键字最多需要对比 h 次, 即查找操作的时间复杂度不可能超过 $O(h)$

平衡二叉树——树上任一结点的左子树和右子树的高度之差不超过1。

假设以 n_h 表示深度为 h 的平衡树中含有的最少结点数。

则有 $n_0 = 0, n_1 = 1, n_2 = 2$, 并且有 $n_h = n_{h-1} + n_{h-2} + 1$

可以证明含有 n 个结点的平衡二叉树的最大深度为 $O(\log_2 n)$, 平衡二叉树的平均查找长度为 $O(\log_2 n)$

AVL删除

平衡二叉树的插入&删除

平衡二叉树的插入操作：

- 插入新结点后，要保持二叉排序树的特性不变（左<中<右）
- 若插入新结点导致不平衡，则需要调整平衡

→ LL RR LR RL

不一样？



平衡二叉树的删除操作：✓

- 删除结点后，要保持二叉排序树的特性不变（左<中<右）
- 若删除结点导致不平衡，则需要调整平衡

→ LL RR LR RL

一样..一样

平衡二叉树的删除

平衡二叉树的删除操作：

- 删除结点后，要保持二叉排序树的特性不变（左<中<右）
- 若删除结点导致不平衡，则需要调整平衡

平衡二叉树的删除操作具体步骤：

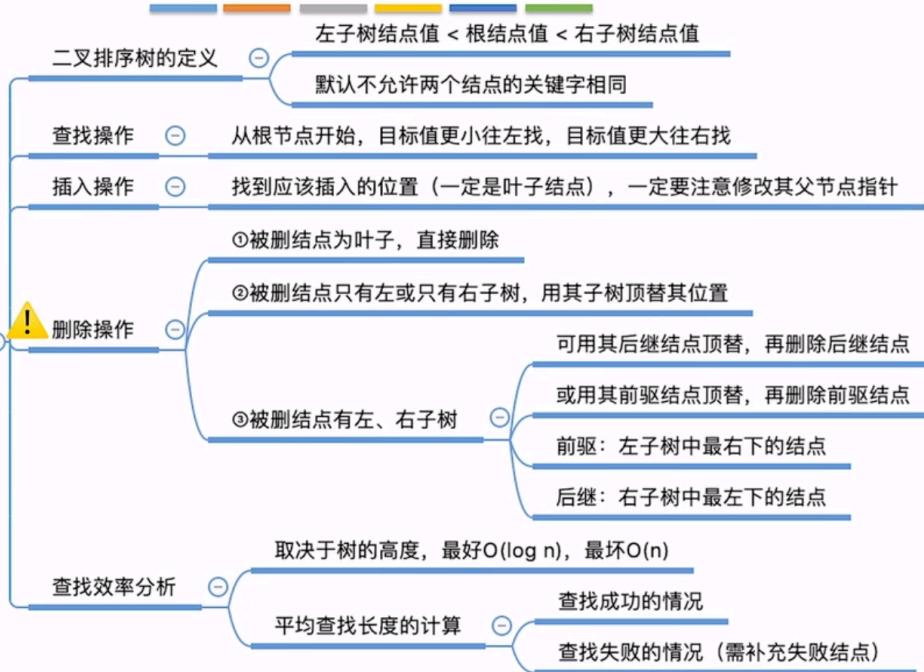
- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②



暂停偷看：二叉排序树的删除操作



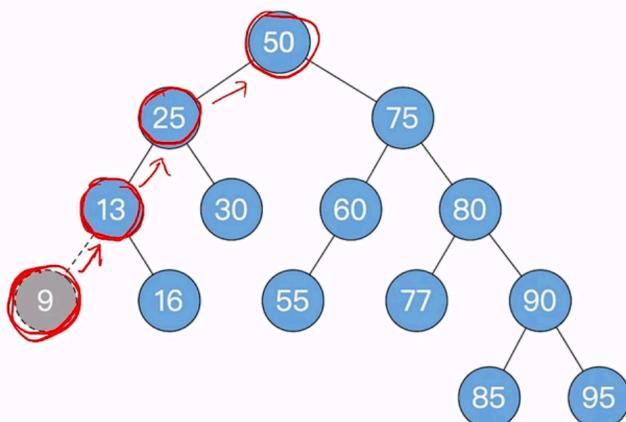
二叉排序树



AVL树删除操作——例1

平衡二叉树的删除操作具体步骤：

- ①删除结点（方法同“二叉排序树”）
 - 若删除的结点是叶子，直接删。
 - 若删除的结点只有一个子树，用子树顶替删除位置
 - 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②





AVL树删除操作——例1

平衡二叉树的删除操作具体步骤：

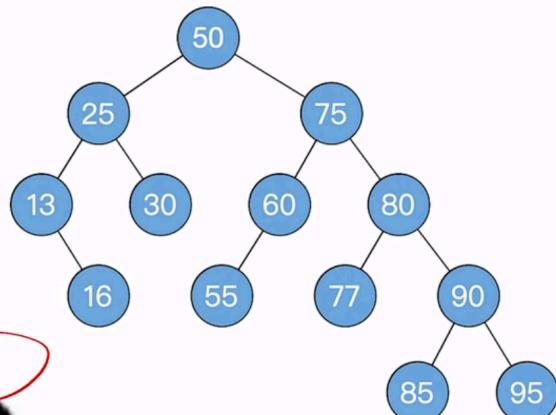
①删除结点（方法同“二叉排序树”）

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例2

平衡二叉树的删除操作具体步骤：

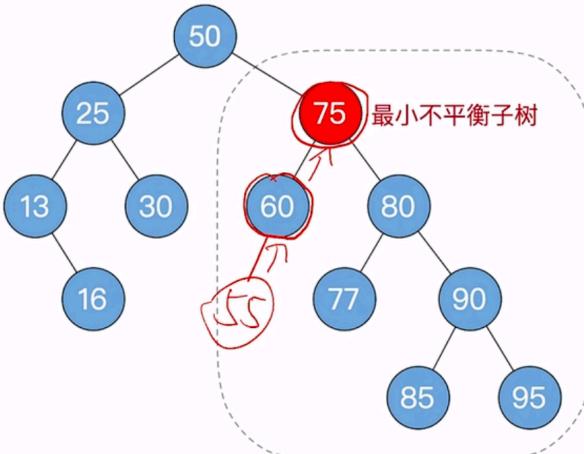
①删除结点（方法同“二叉排序树”）

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例2

平衡二叉树的删除操作具体步骤：

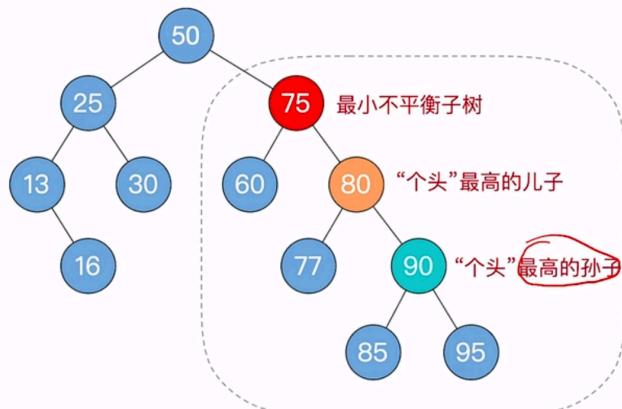
①删除结点（方法同“二叉排序树”）

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②





AVL树删除操作——例2

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

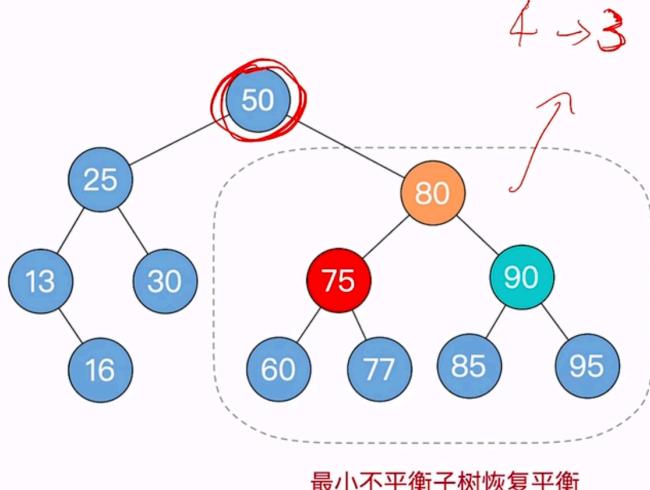
②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

- 孙子在LL：儿子右单旋
- 孙子在RR：儿子左单旋
- 孙子在LR：孙子先左旋，再右旋
- 孙子在RL：孙子先右旋，再左旋

⑤如果不平衡向上传导，继续②



AVL树删除操作——例3

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

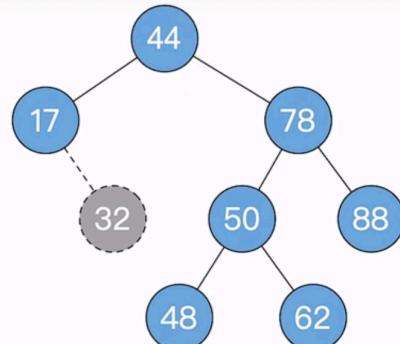
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例3

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

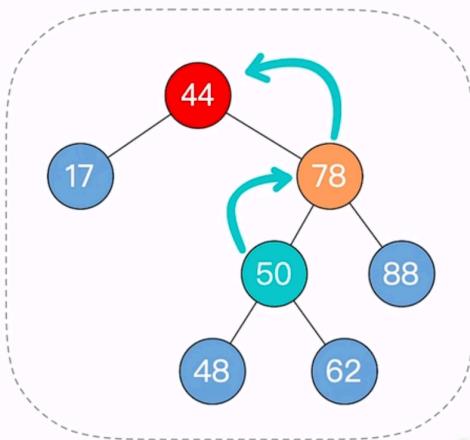
②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

- 孙子在LL：儿子右单旋
- 孙子在RR：儿子左单旋
- 孙子在LR：孙子先左旋，再右旋
- 孙子在RL：孙子先右旋，再左旋

⑤如果不平衡向上传导，继续②





AVL树删除操作——例4



平衡二叉树的**删除**操作具体步骤：

①删除结点（方法同“二叉排序树”）

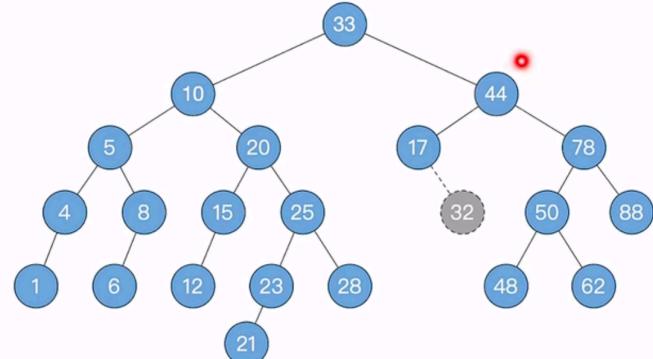
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例3



平衡二叉树的**删除**操作具体步骤：

①删除结点（方法同“二叉排序树”）

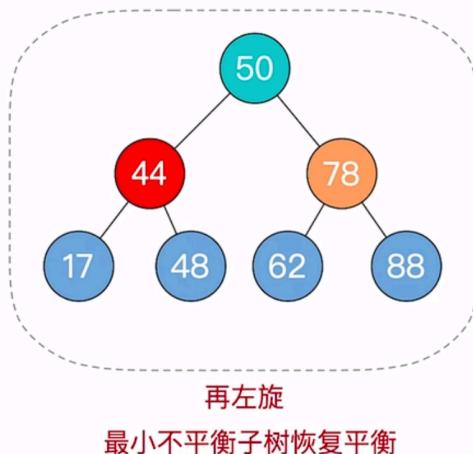
②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

- 孙子在LL：儿子右单旋
- 孙子在RR：儿子左单旋
- 孙子在LR：孙子先左旋，再右旋
- 孙子在RL：孙子先右旋，再左旋

⑤如果不平衡向上传导，继续②

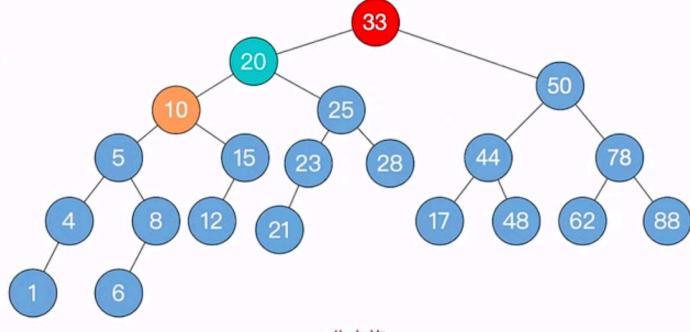


EG4

AVL树删除操作——例4

平衡二叉树的删除操作具体步骤：

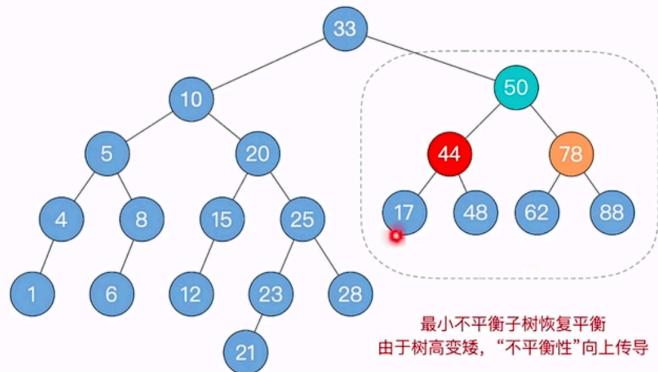
- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
 - 孙子在LL：儿子右单旋
 - 孙子在RR：儿子左单旋
 - 孙子在LR：孙子先左旋，再右旋
 - 孙子在RL：孙子先右旋，再左旋
- ⑤如果不平衡向上传导，继续②



AVL树删除操作——例4

平衡二叉树的删除操作具体步骤：

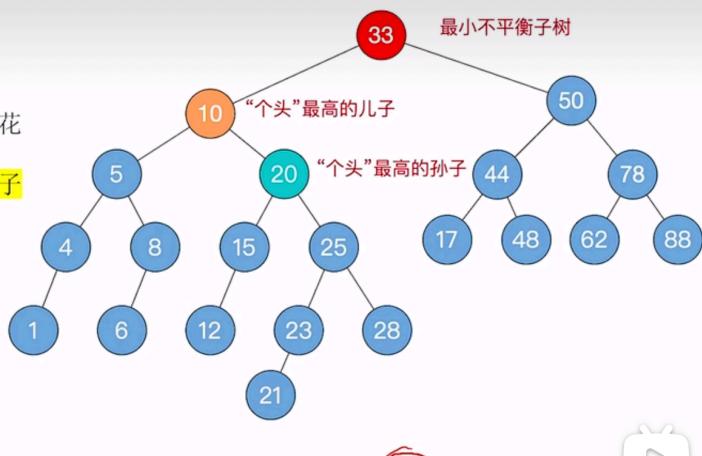
- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②
 - 对最小不平衡子树的旋转可能导致树变矮，从而导致上层祖先不平衡（不平衡向上传递）



AVL树删除操作——例4

平衡二叉树的删除操作具体步骤：

- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②





AVL树删除操作——例4

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

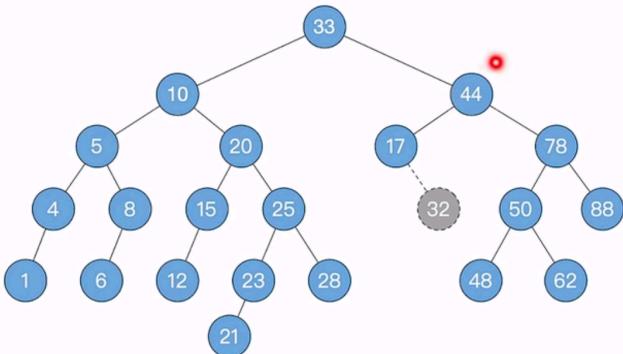
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例4

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

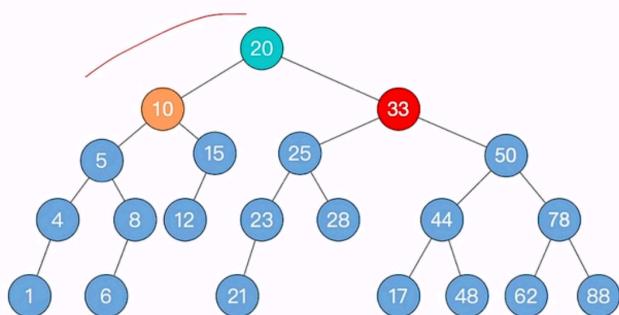
②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②

- 对最小不平衡子树的旋转可能导致树变矮，从而导致上层祖先不平衡（不平衡向上传递）



Over !





AVL树删除操作——例5

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

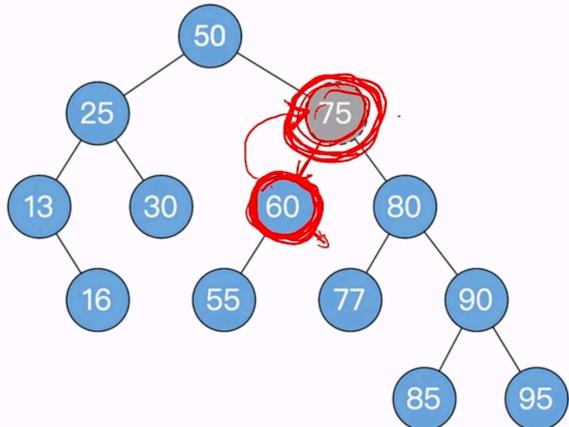
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置。
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



被删除结点有左右子树，用前驱结点顶替（复制数据即可
并转化为对前驱结点的删除

EG5



AVL树删除操作——例5

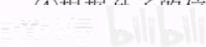
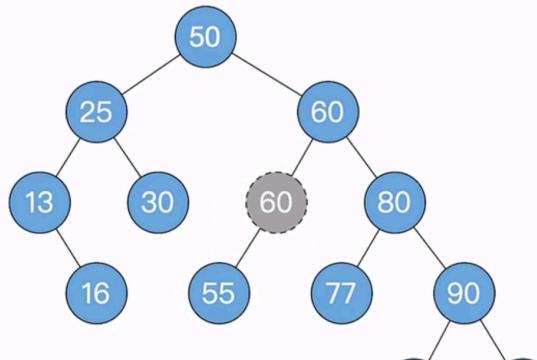
平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置。
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子



AVL树删除操作——例5

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

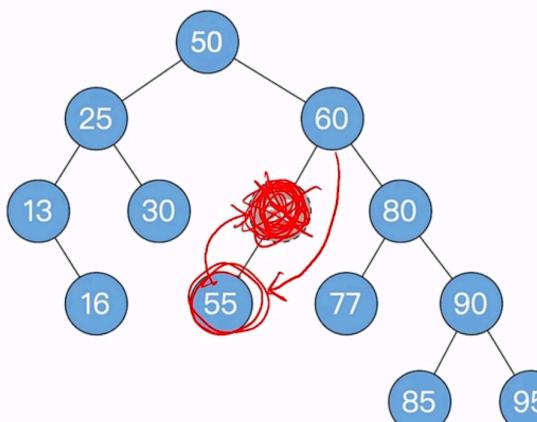
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置。
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



被删除结点只有左子树，用子树顶替删除位置（用结点实体顶替）

AVL树删除操作——例5

平衡二叉树的删除操作具体步骤：

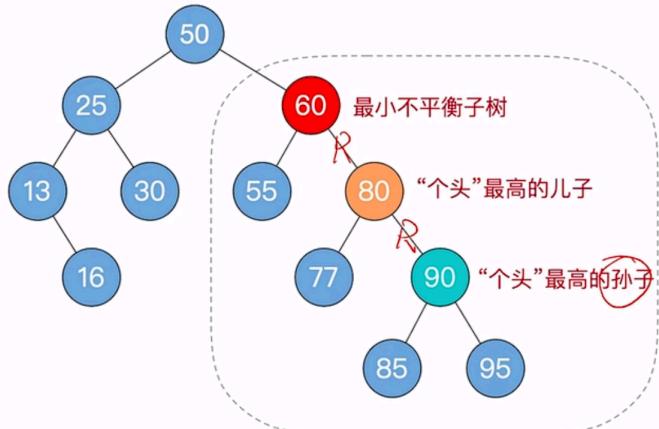
①删除结点（方法同“二叉排序树”）

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



AVL树删除操作——例5

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

②一路向北找到最小不平衡子树，找不到就完结撒花

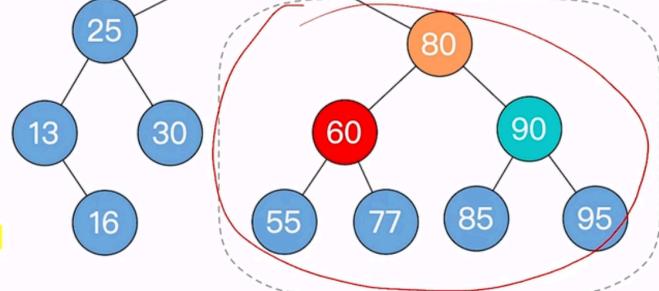
③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②

- 对最小不平衡子树的旋转可能导致树变矮，从而导致上层祖先不平衡（不平衡向上传递）

Over !



EG6



AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

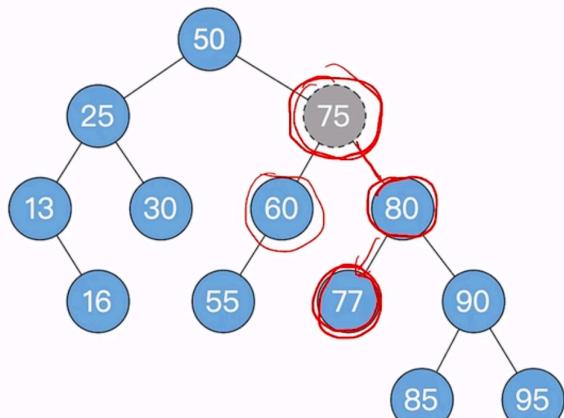
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



被删除结点有左右子树，用后继结点顶替（复制数据即可）
并转化为对后继结点的删除



AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

①删除结点（方法同“二叉排序树”）

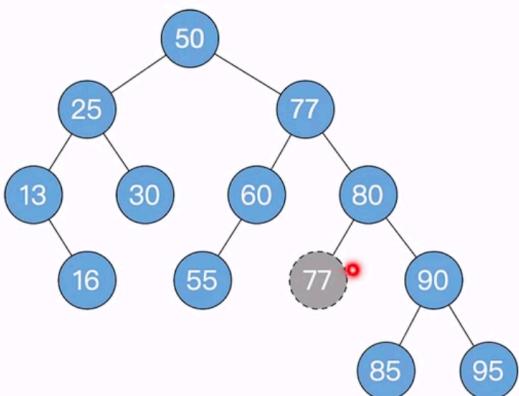
- 若删除的结点是叶子，直接删。
- 若删除的结点只有一个子树，用子树顶替删除位置
- 若删除的结点有两棵子树，用前驱（或后继）结点顶替，并转换为对前驱（或后继）结点的删除。

②一路向北找到最小不平衡子树，找不到就完结撒花

③找最小不平衡子树下，“个头”最高的儿子、孙子

④根据孙子的位置，调整平衡（LL/RR/LR/RL）

⑤如果不平衡向上传导，继续②



被删除结点为叶子，直接删即可

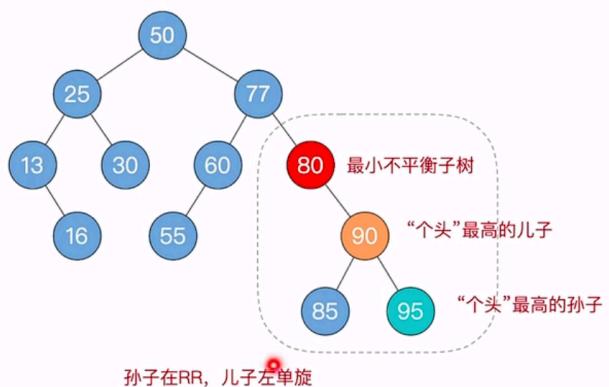




AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

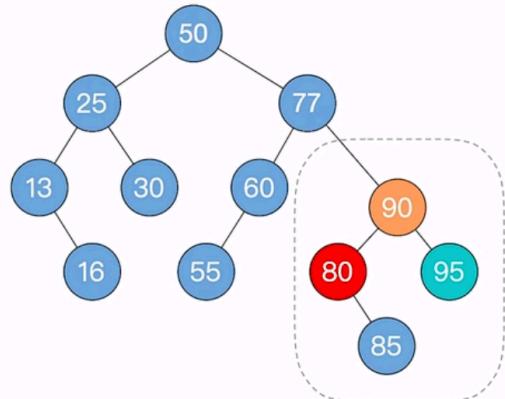
- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②



AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②
 - 对最小不平衡子树的旋转可能导致树变矮，从而导致上层祖先不平衡（不平衡向上传递）



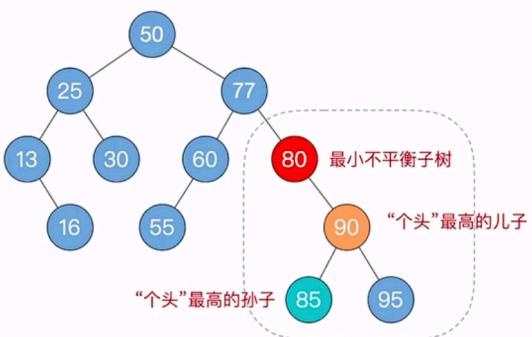
Over !



AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
- ⑤如果不平衡向上传导，继续②

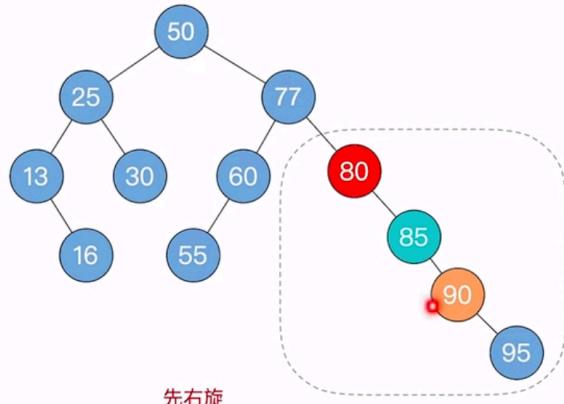




AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

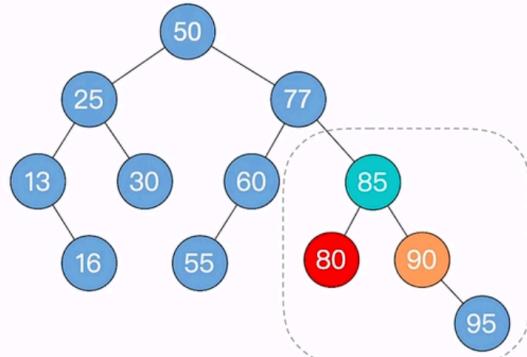
- ①删除结点（方法同“二叉排序树”）
- ②一路向北找到最小不平衡子树，找不到就完结撒花
- ③找最小不平衡子树下，“个头”最高的儿子、孙子
- ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
 - 孙子在LL：儿子右单旋
 - 孙子在RR：儿子左单旋
 - 孙子在LR：孙子先左旋，再右旋
 - 孙子在RL：孙子先右旋，再左旋
- ⑤如果不平衡向上传递，继续②



AVL树删除操作——例6

平衡二叉树的删除操作具体步骤：

- ①删除结点（方法同“二叉排序树”）
 - ②一路向北找到最小不平衡子树，找不到就完结撒花
 - ③找最小不平衡子树下，“个头”最高的儿子、孙子
 - ④根据孙子的位置，调整平衡（LL/RR/LR/RL）
 - ⑤如果不平衡向上传递，继续②
- 对最小不平衡子树的旋转可能导致树变矮，从而导致上层祖先不平衡（不平衡向上传递）



Over !



不可能考这种有多种处理方式的题目！