本节内容

平衡二叉树

(AVL)

知识总览

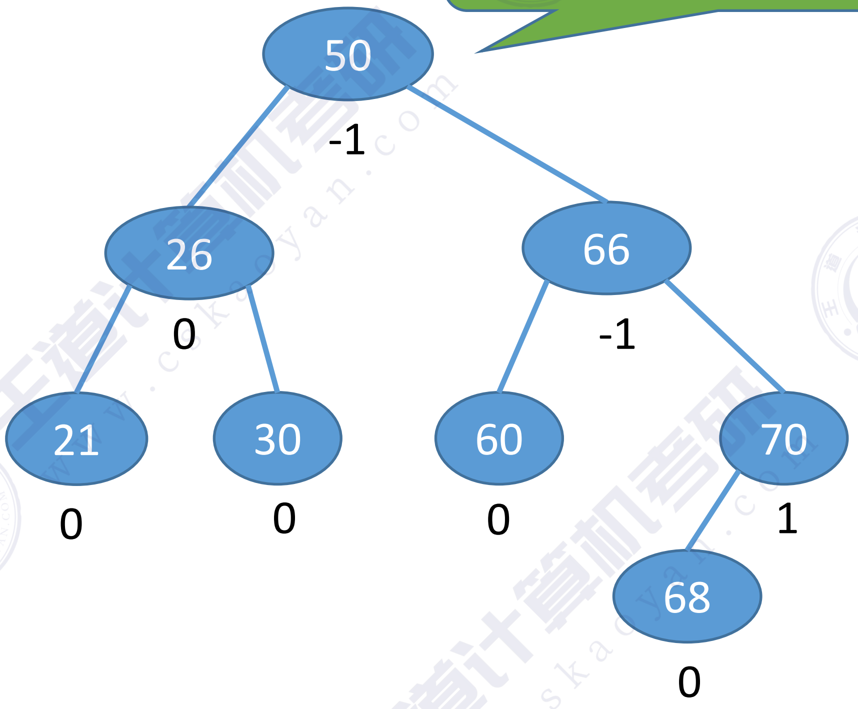
- 平衡二叉树
  - 定义
  - 插入操作
  - 插入新结点后如何调整“不平衡”问题
  - 查找效率分析

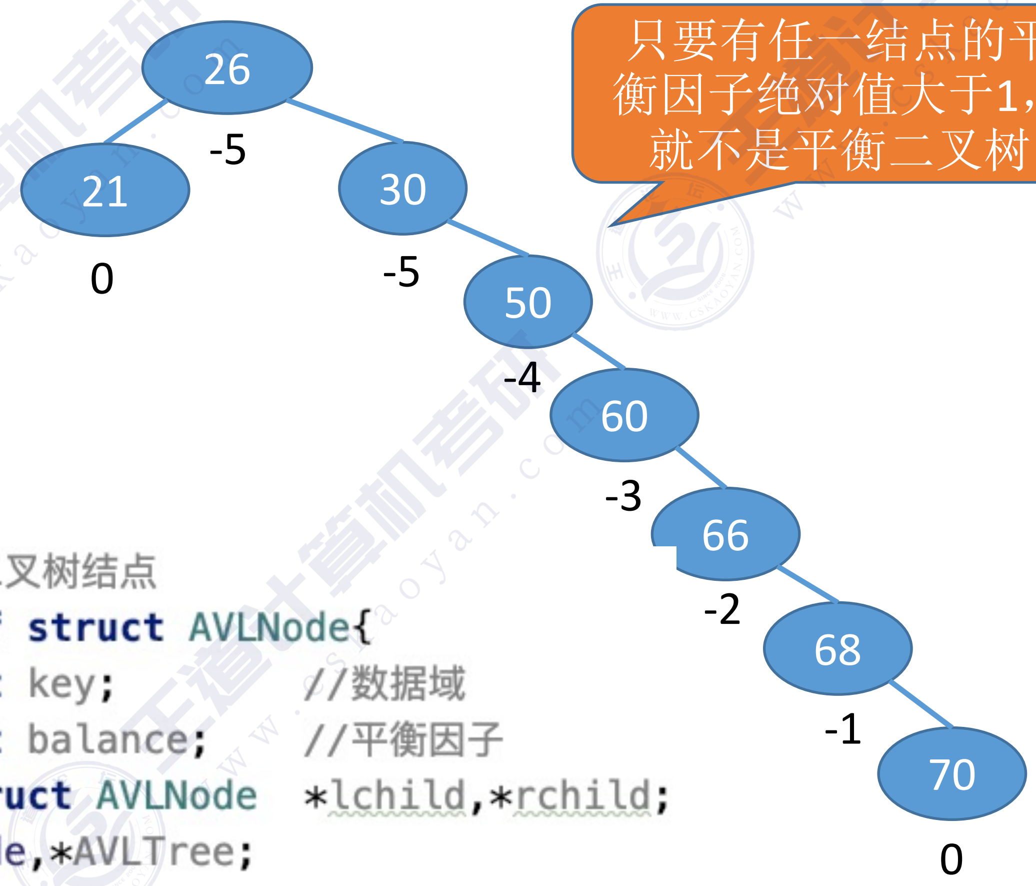
平衡二叉树的定义

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。



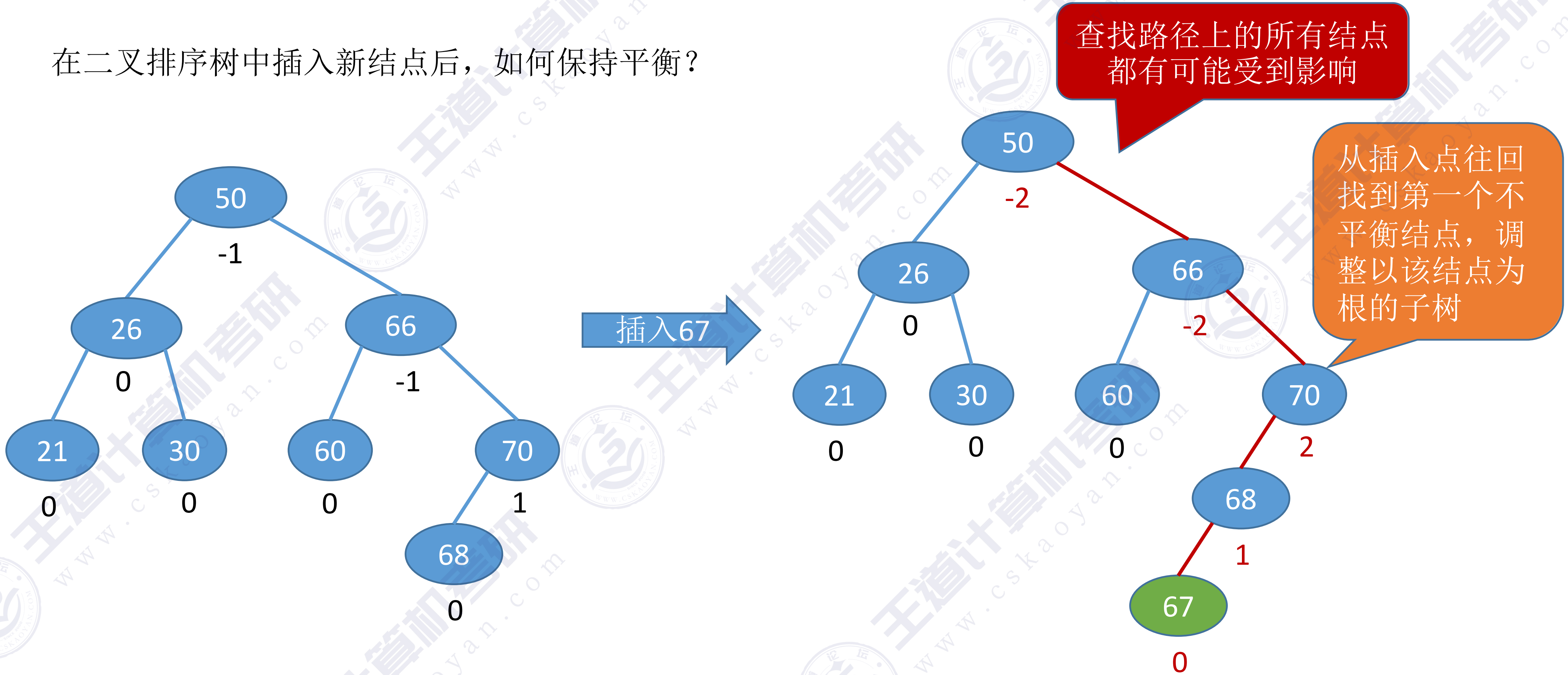


只要有任一结点的平衡因子绝对值大于1，就不是平衡二叉树

```
//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;
```

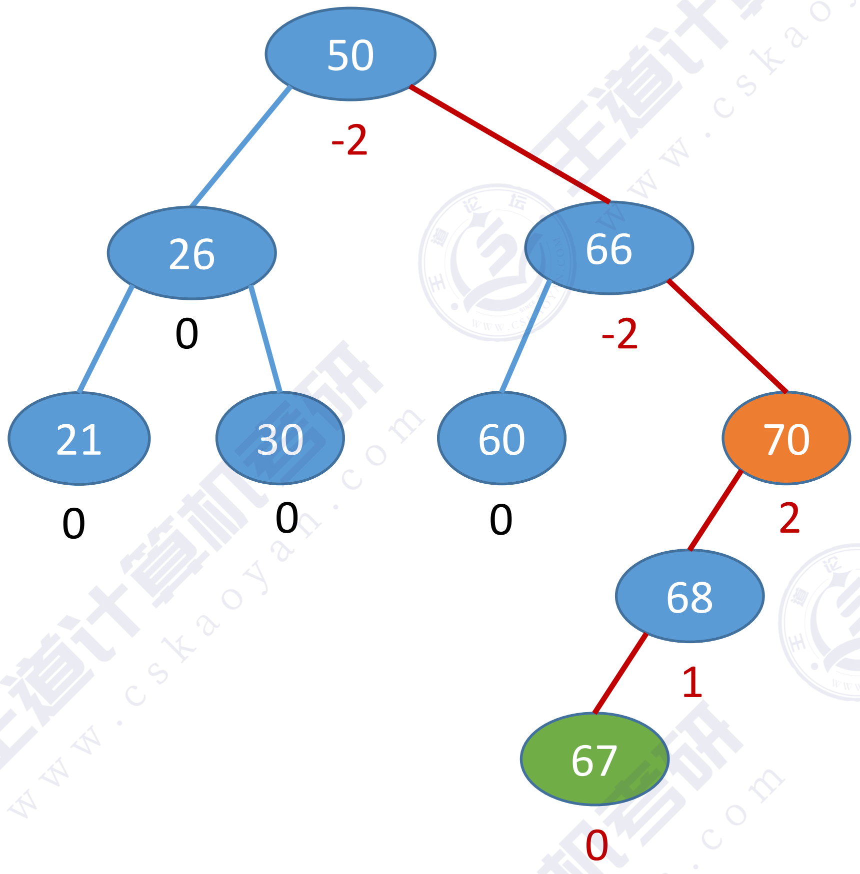
平衡二叉树的插入

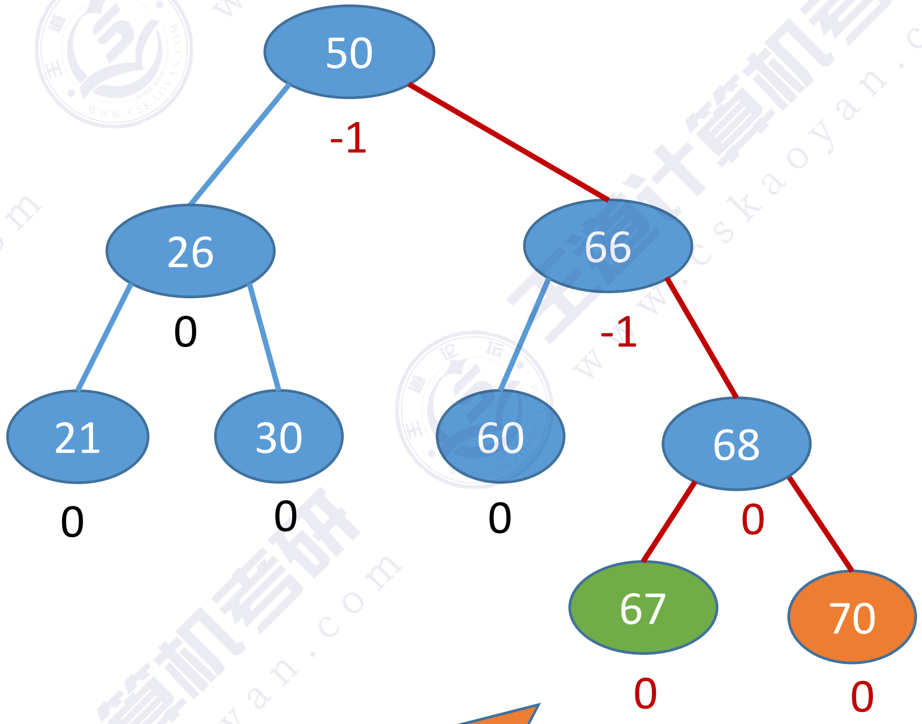
在二叉排序树中插入新结点后，如何保持平衡？



每次调整的对象都是“最小不平衡子树”

平衡二叉树的插入





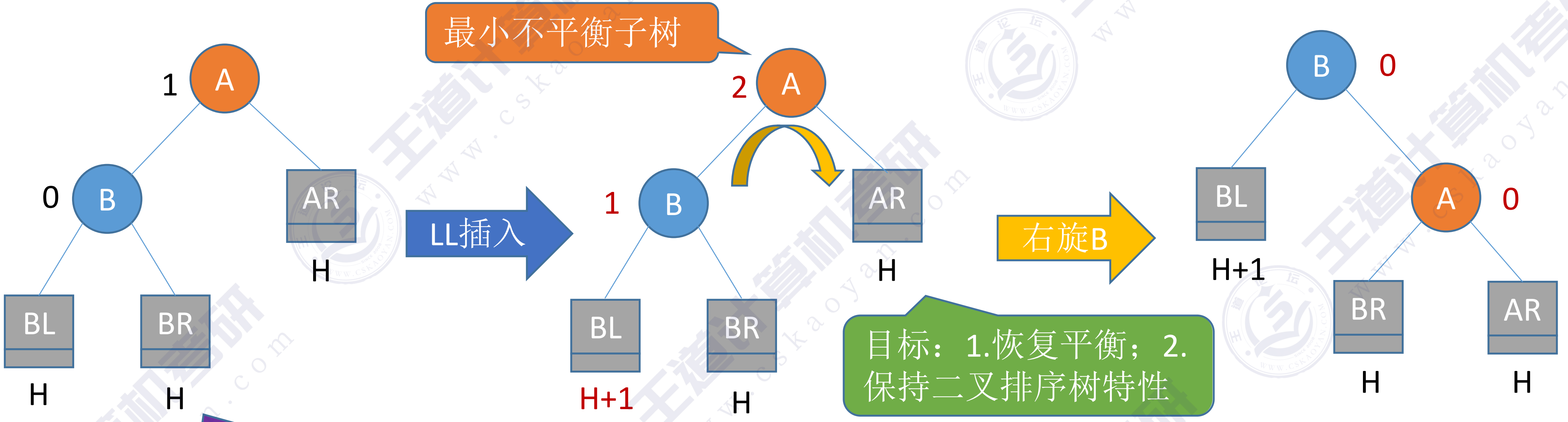
在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

每次调整的对象都是“最小不平衡子树”

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

调整最小不平衡子树（LL）



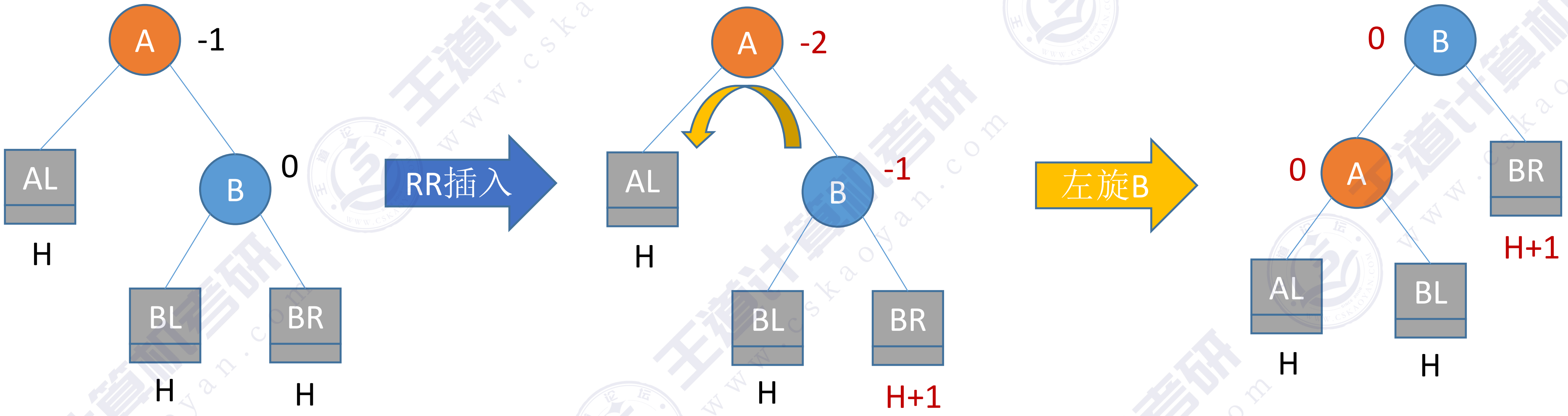
思考：为什么要假定所有子树的高度都是H？

二叉排序树的特性：左子树结点值 < 根结点值 < 右子树结点值

$BL<B<BR<A<AR$

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。

调整最小不平衡子树（RR）

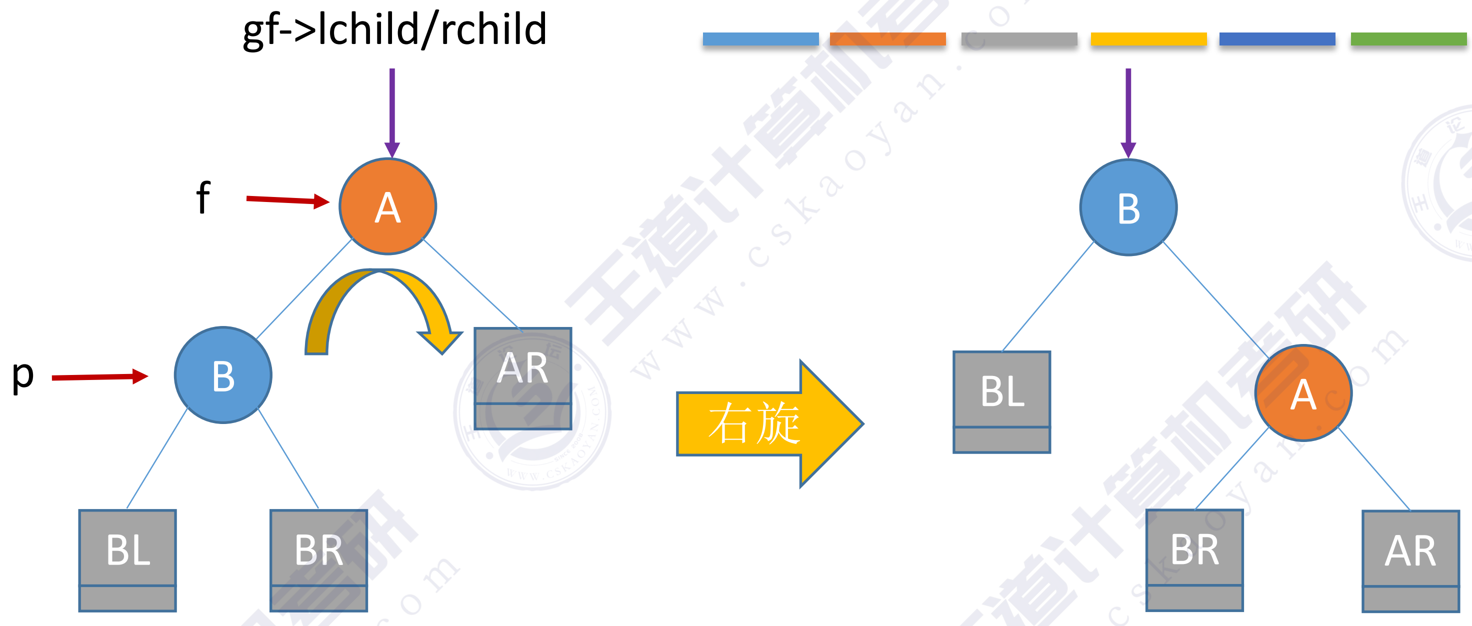


二叉排序树的特性：左子树结点值 < 根结点值 < 右子树结点值

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

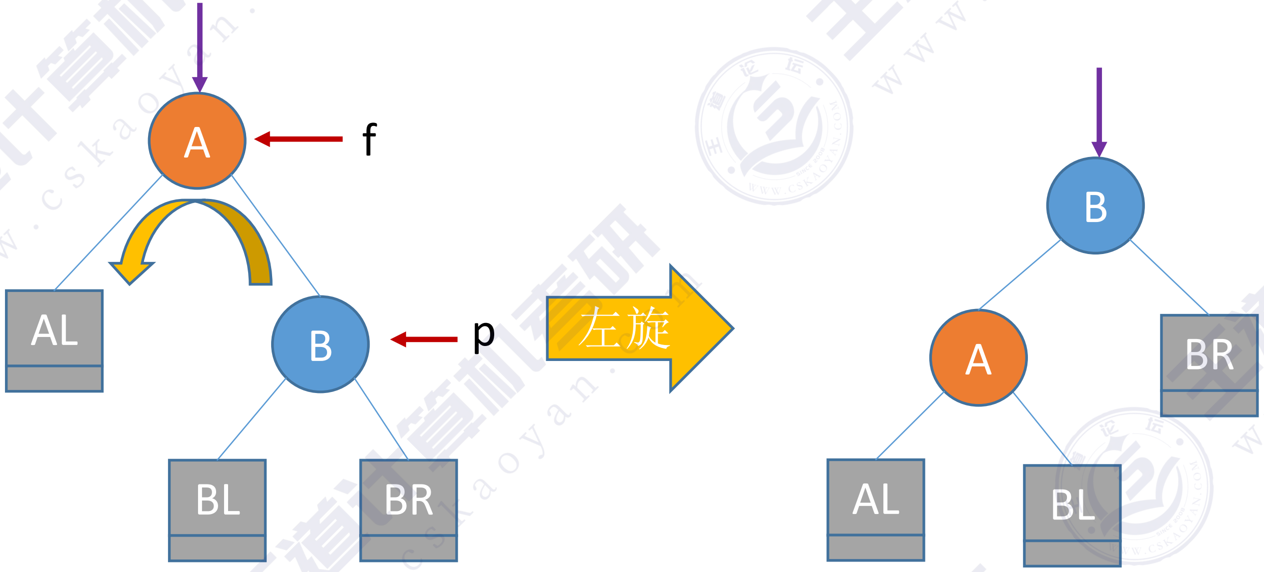
代码思路



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他爹
① f->lchild = p->rchild;
② p->rchild = f;
③ gf->lchild/rchild = p;

BL<B<BR<A<AR

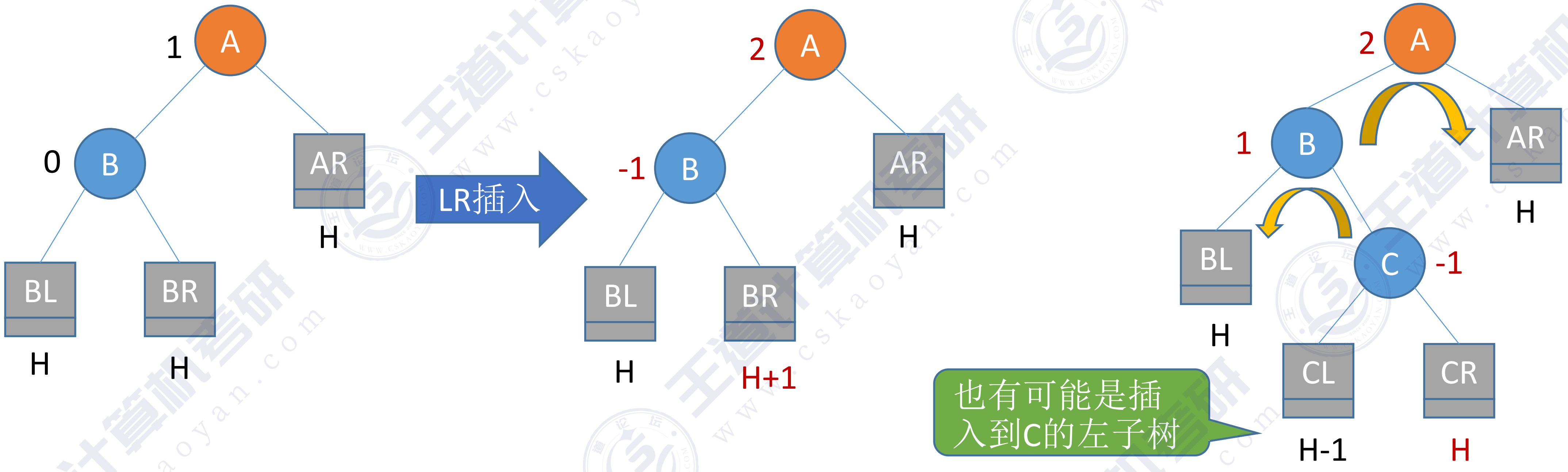
左旋、右旋操作后可以保持二叉排序树的特性



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他爹
① f->rchild = p->lchild;
② p->lchild = f;
③ gf->lchild/rchild = p;

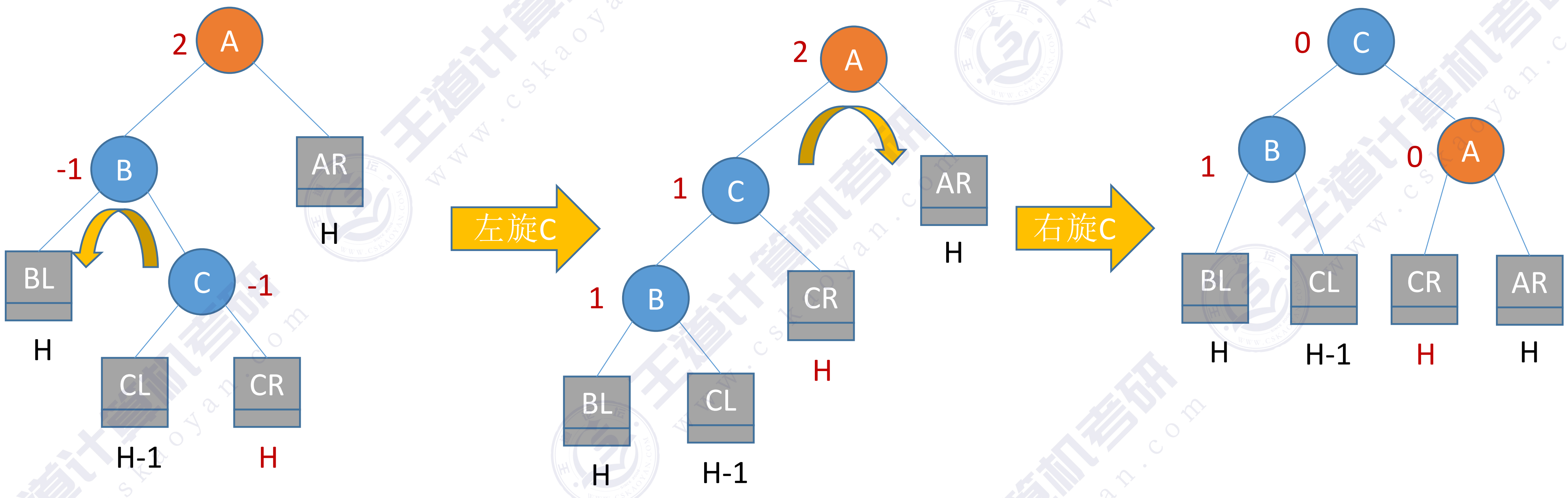
AL<A<BL<B<BR

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

调整最小不平衡子树（LR）

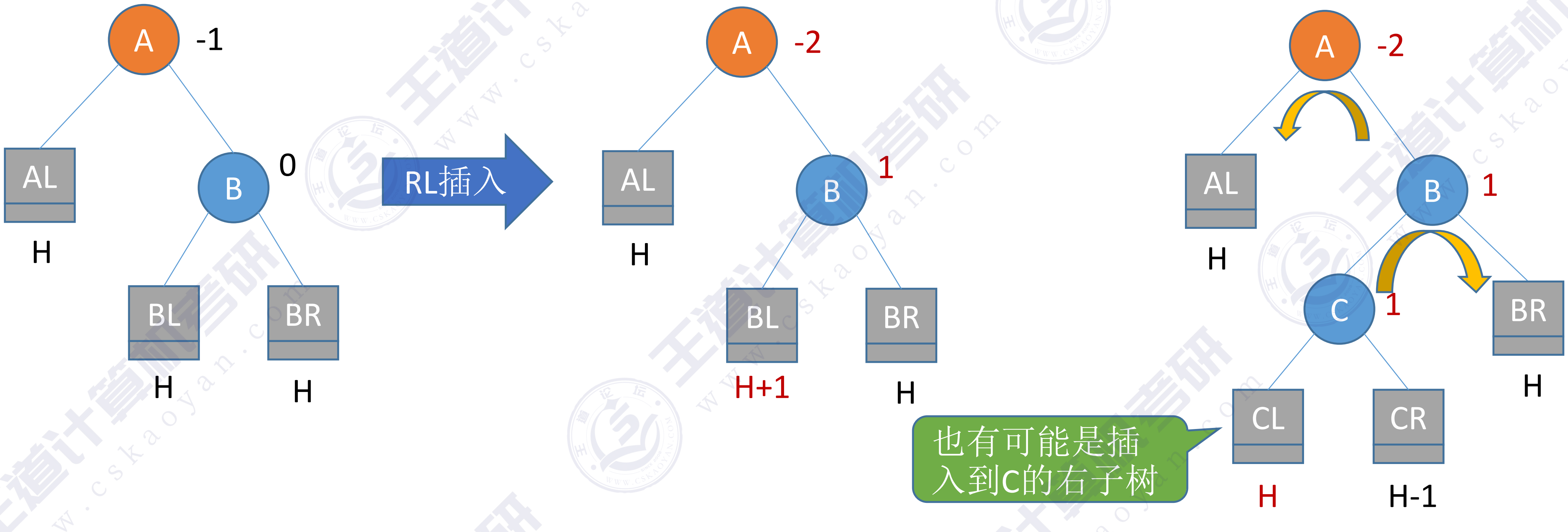


BL<B<CL<C< CR<A<AR

调整最小不平衡子树（LR）

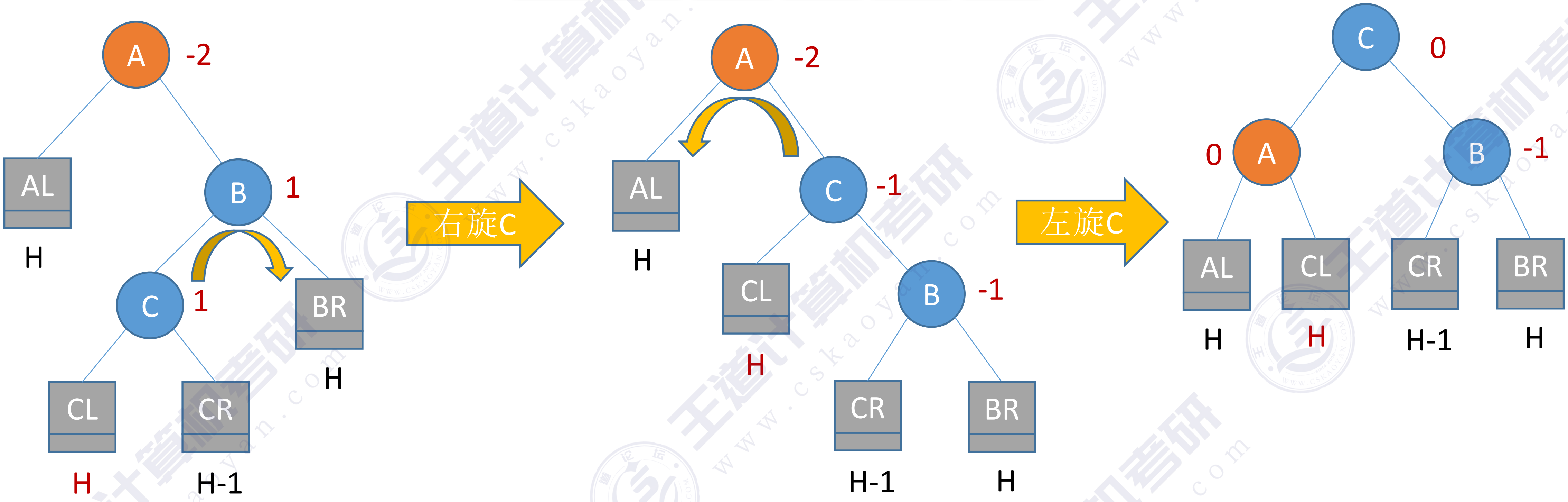
调整最小不平衡子树（RL）

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



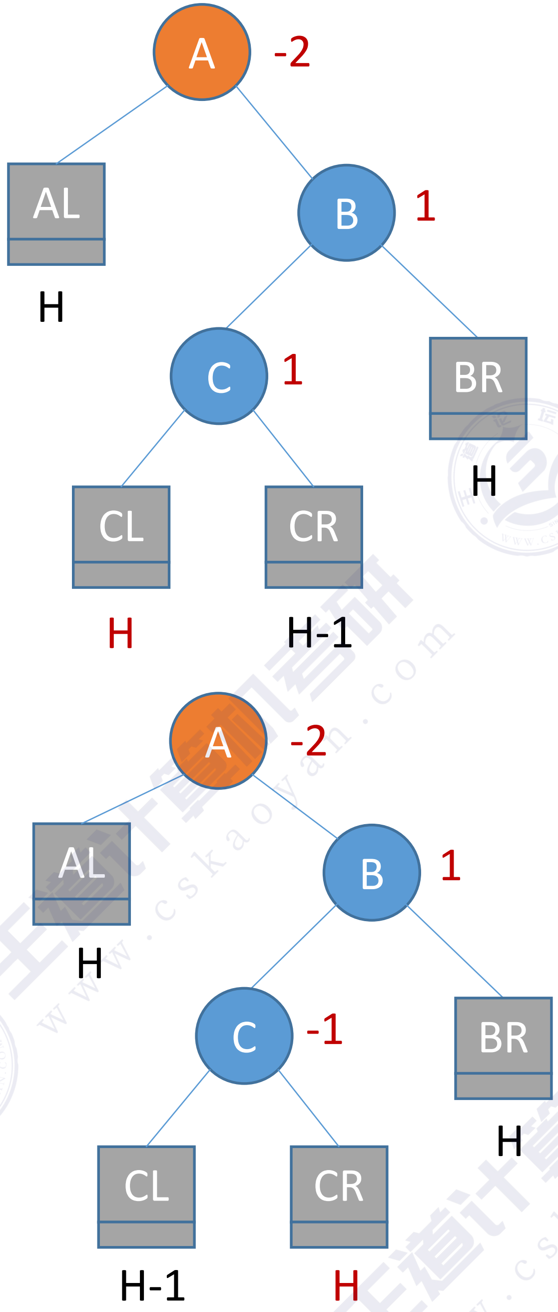
4）RL平衡旋转（先右后左双旋转）。由于在*A*的右孩子（R）的左子树（L）上插入新结点，*A*的平衡因子由-1减至-2，导致以*A*为根的子树失去平衡，需要进行两次旋转操作，先右旋转后左旋转。先将*A*结点的右孩子*B*的左子树的根结点*C*向右上旋转提升到*B*结点的位置，然后再把该*C*结点向左上旋转提升到*A*结点的位置

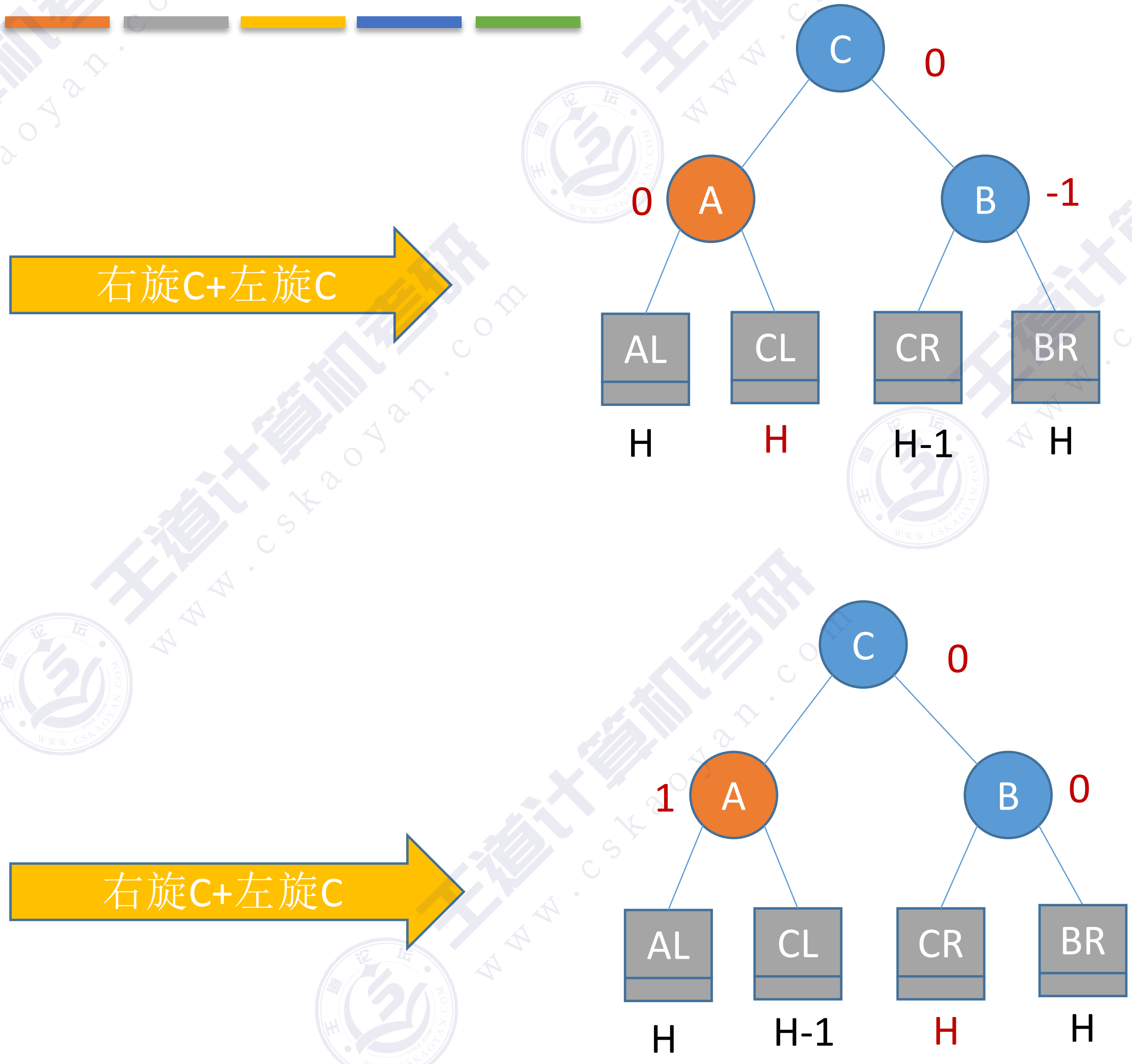
调整最小不平衡子树（RL）



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

调整最小不平衡子树（RL）

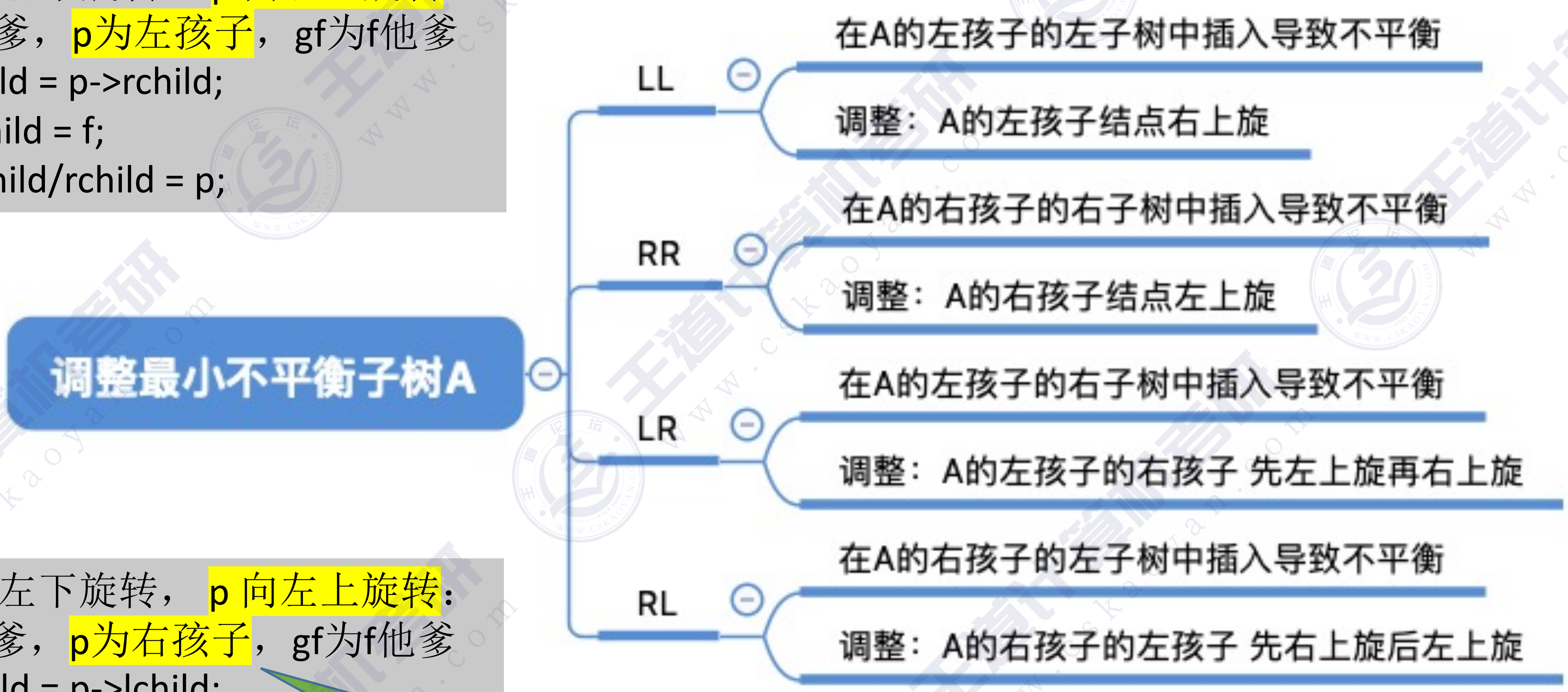




调整最小不平衡子树

只有左孩子
才能右上旋

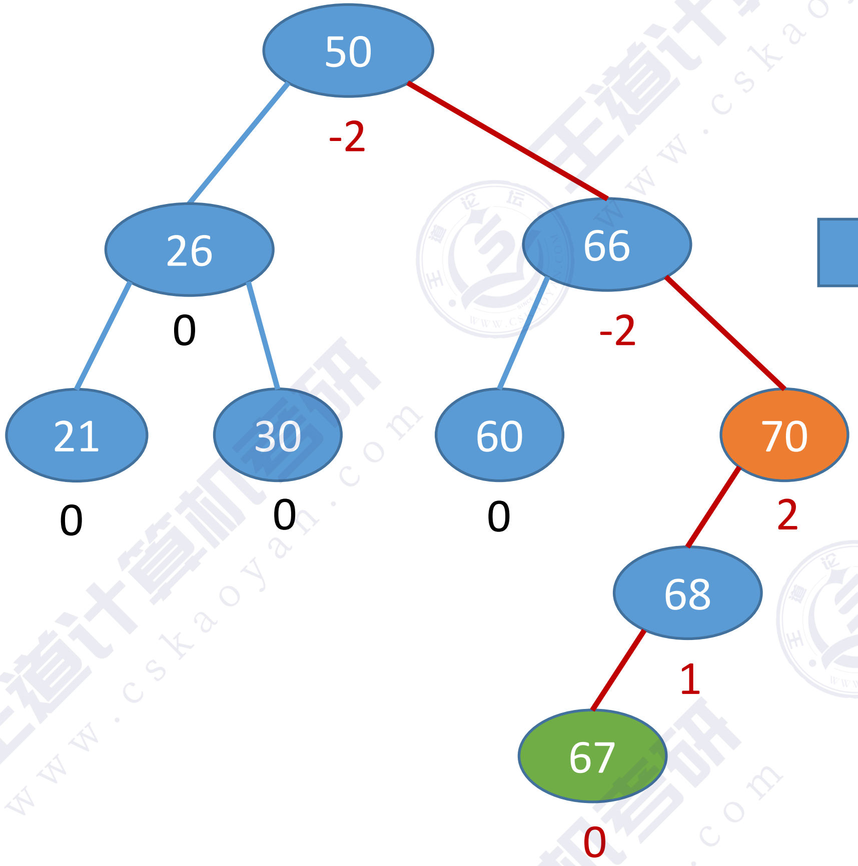
实现 f 向右下旋转， p 向右上旋转：
其中 f是爹， p为左孩子， gf为f他爹
① f->lchild = p->rchild;
② p->rchild = f;
③ gf->lchild/rchild = p;

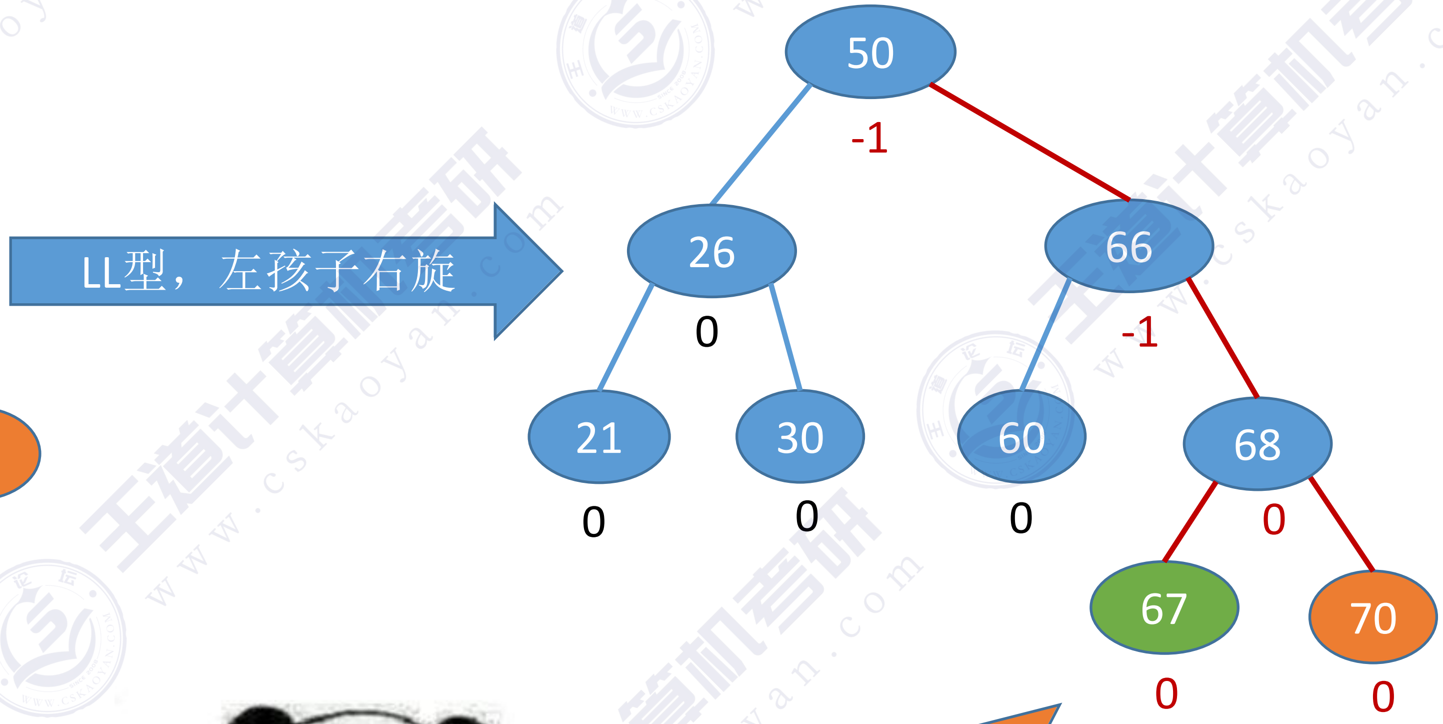


实现 f 向左下旋转， p 向左上旋转：
其中 f是爹， p为右孩子， gf为f他爹
① f->rchild = p->lchild;
② p->lchild = f;
③ gf->lchild/rchild = p;

只有右孩子
才能左上旋

填个坑

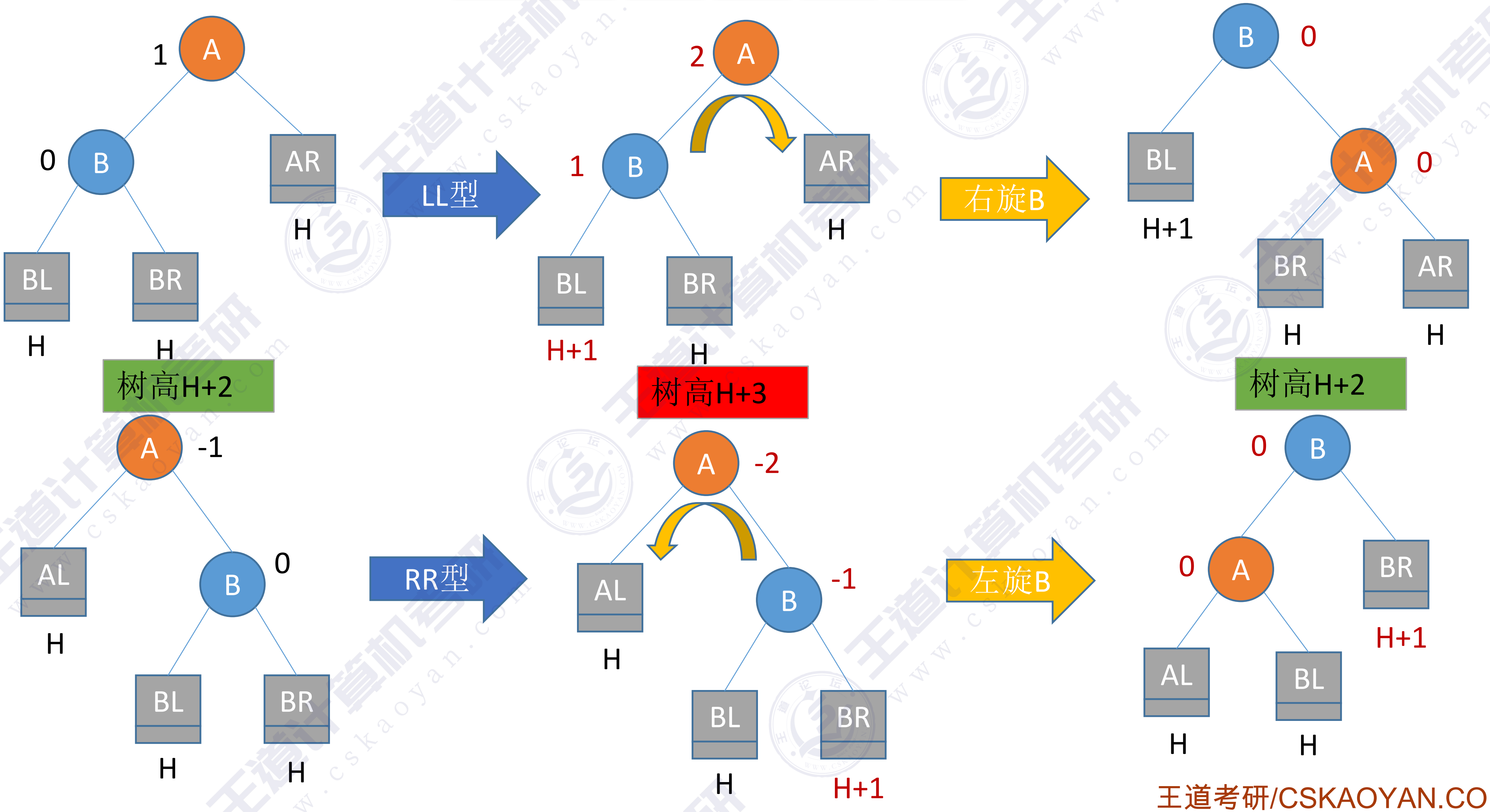




每次调整的对象都是“最小不平衡子树”

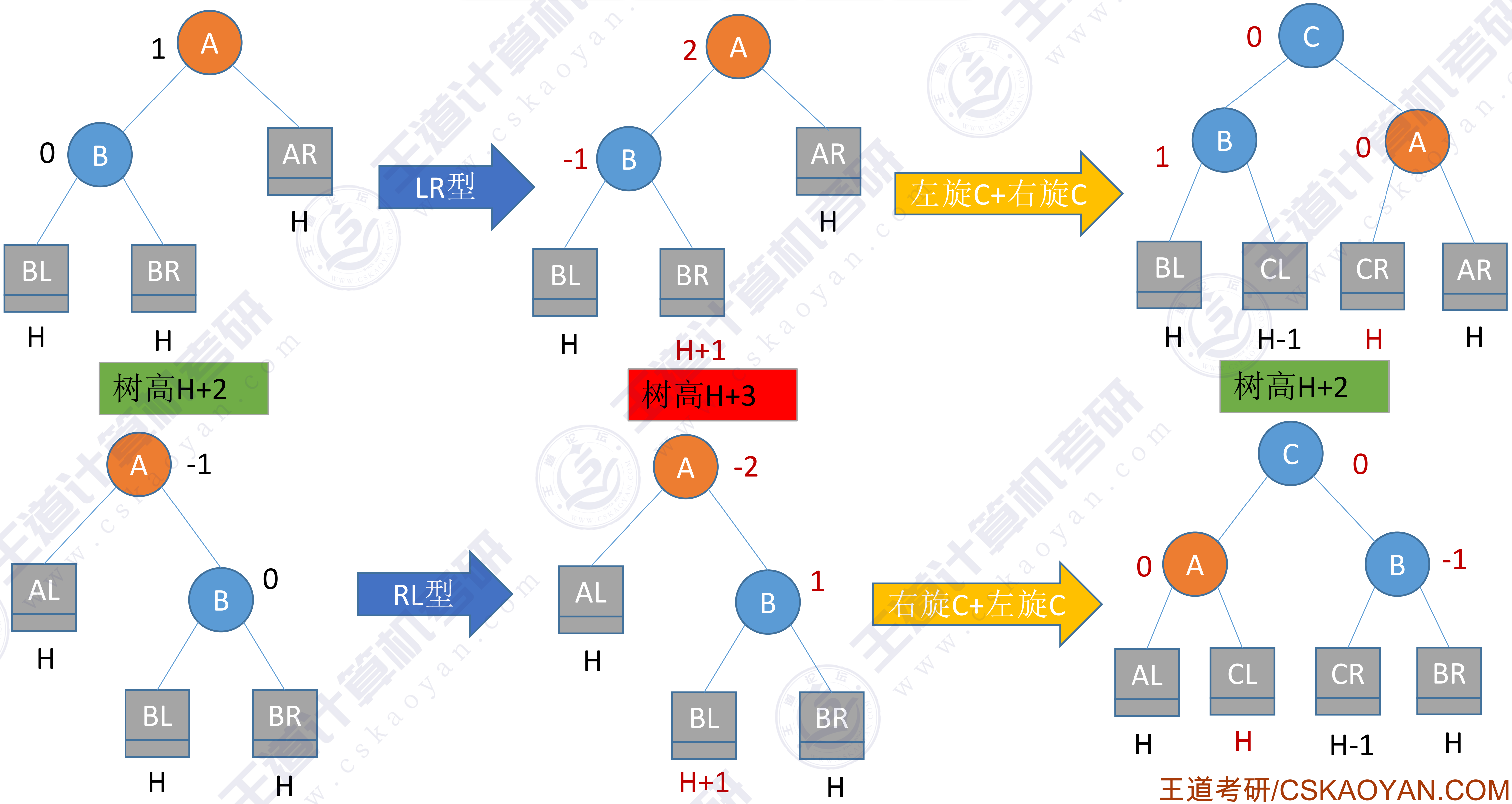
在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

填个坑



A
1
B
0
AR
H
BL
BR
H
H
树高H+2
LL型
A
2
B
1
AR
H
BL
BR
H+1
H
树高H+3
右旋B
B
0
BL
H+1
A
0
BR
AR
H
H
树高H+2
A
-1
AL
H
B
0
BL
BR
H
H
RR型
A
-2
AL
H
B
-1
BL
BR
H
H+1
左旋B
B
0
A
0
BR
H+1
AL
BL
H
H

填个坑

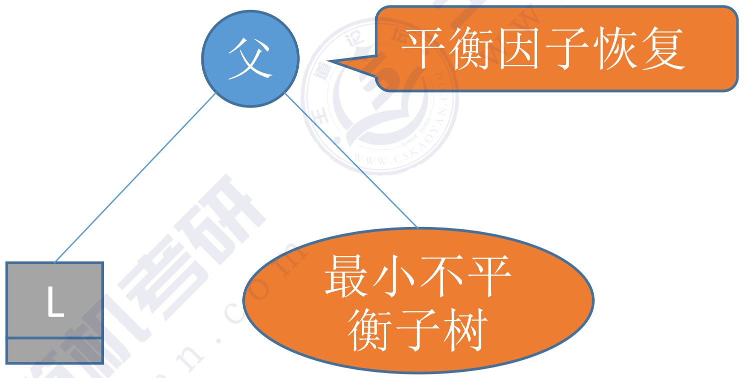


1 A
0 B
AR
H
BL
BR
H
H
树高H+2
LR型
2 A
-1 B
AR
H
BL
BR
H
H+1
树高H+3
左旋C+右旋C
0 C
1 B
0 A
BL
CL
CR
AR
H
H-1
H
H
树高H+2
A -1
AL
H
B 0
BL
BR
H
H
RL型
A -2
AL
H
B 1
BL
BR
H+1
H
右旋C+左旋C
C 0
0 A
B -1
AL
CL
CR
BR
H
H
H-1
H

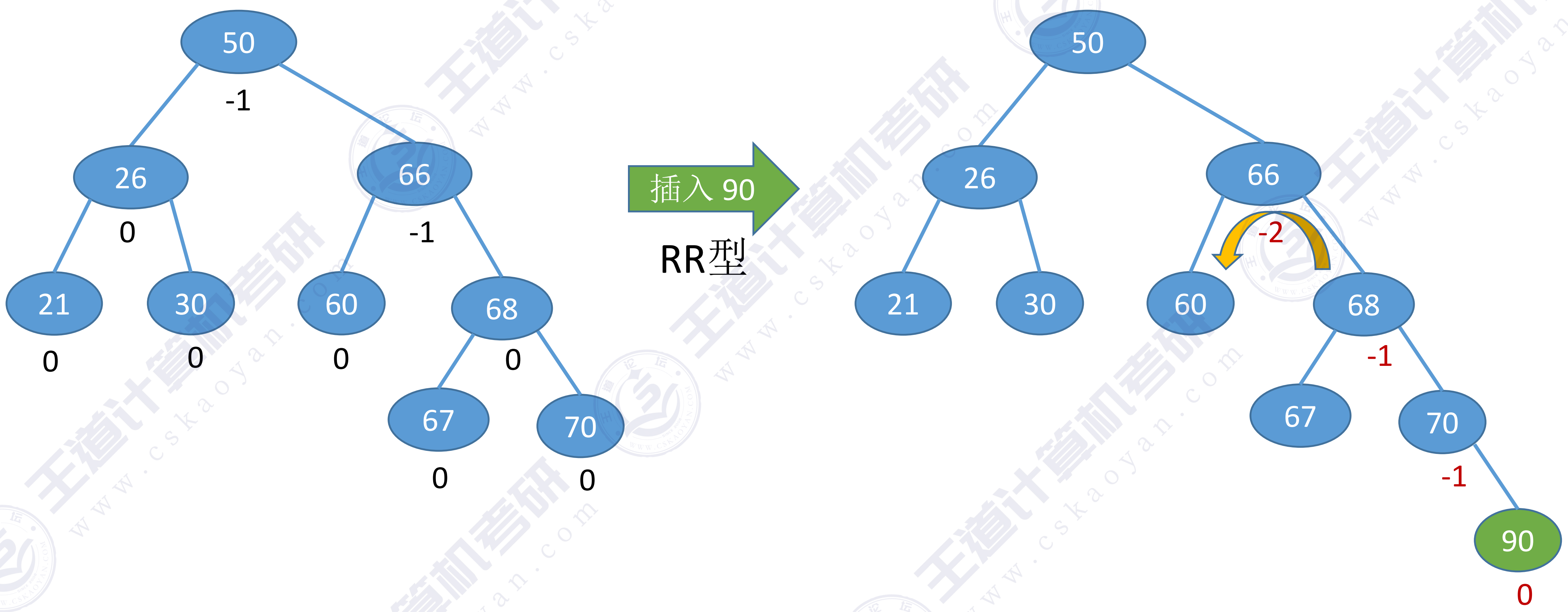
填个坑

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

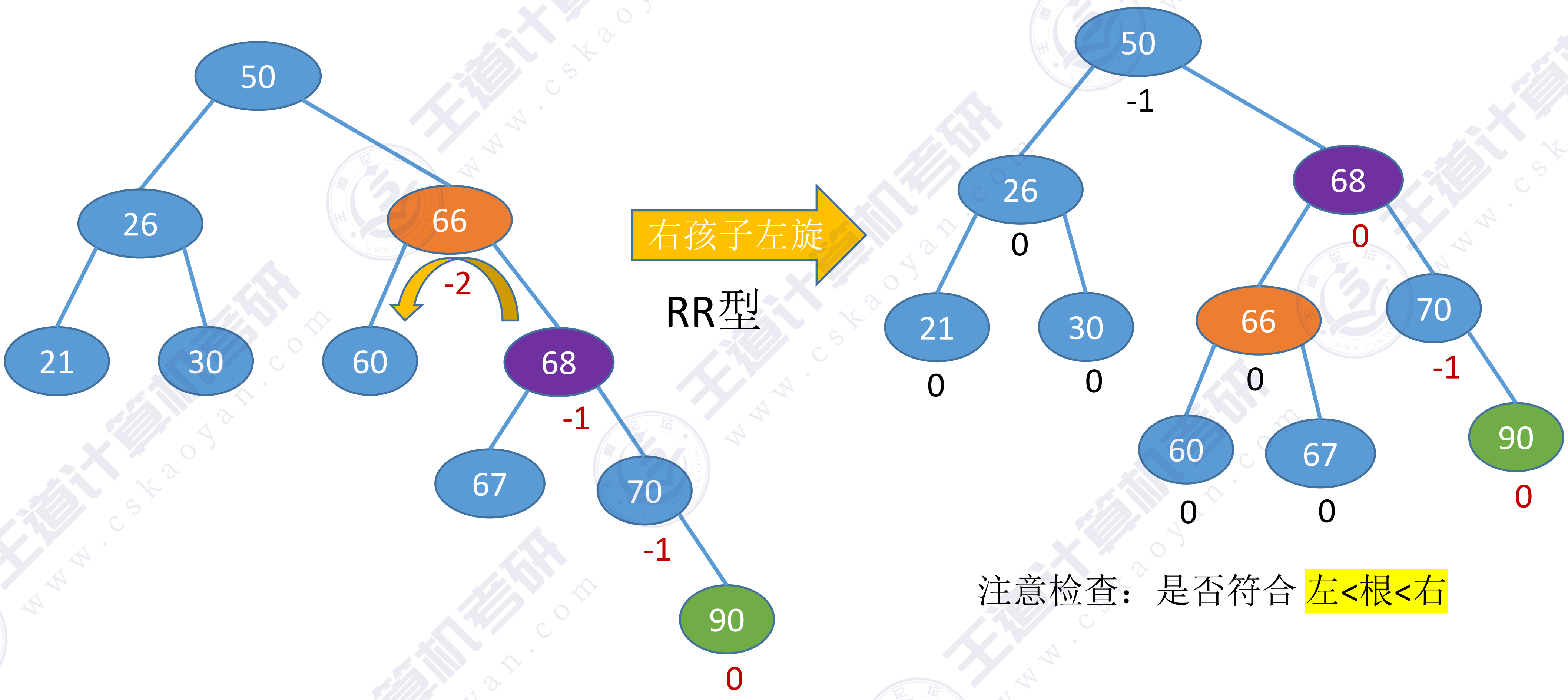


练习



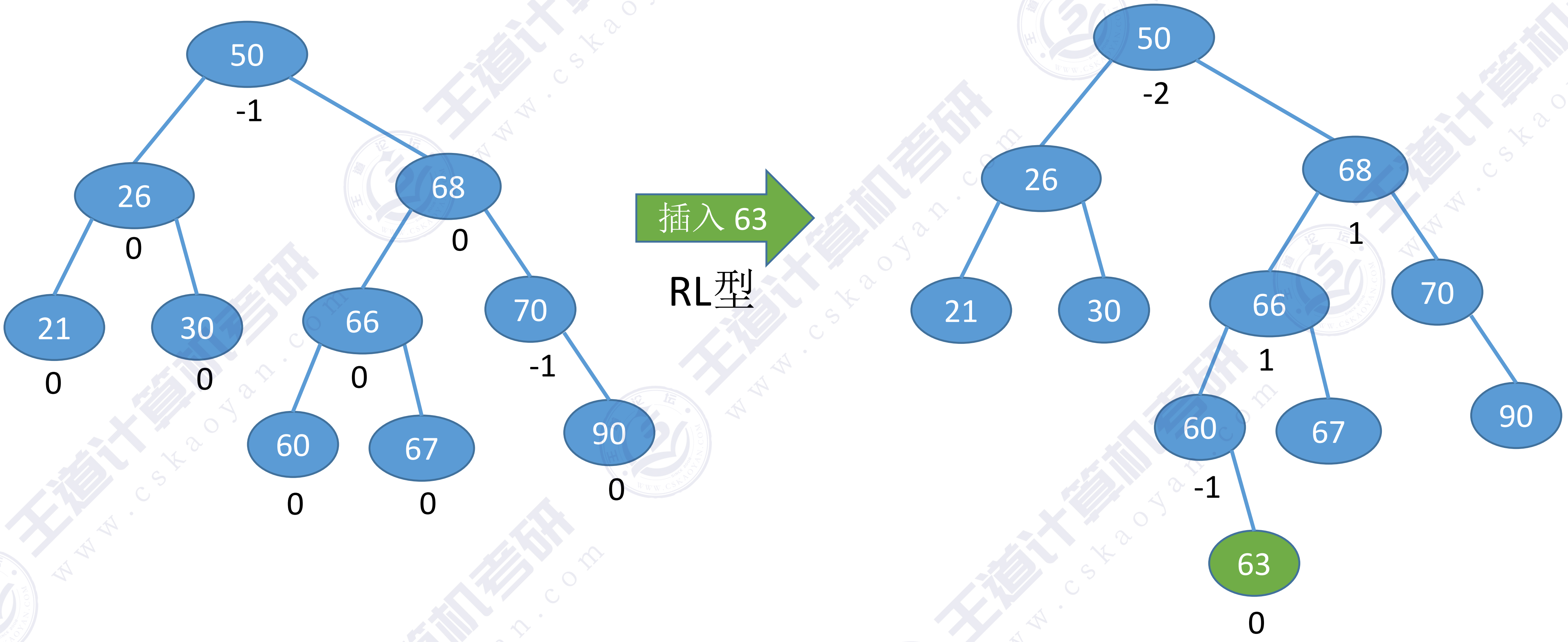
50
-1
26
0
66
-1
21
0
30
0
60
0
68
0
67
0
70
0
插入 90
RR型
50
26
66
-2
21
30
60
68
-1
67
70
-1
90
0

练习



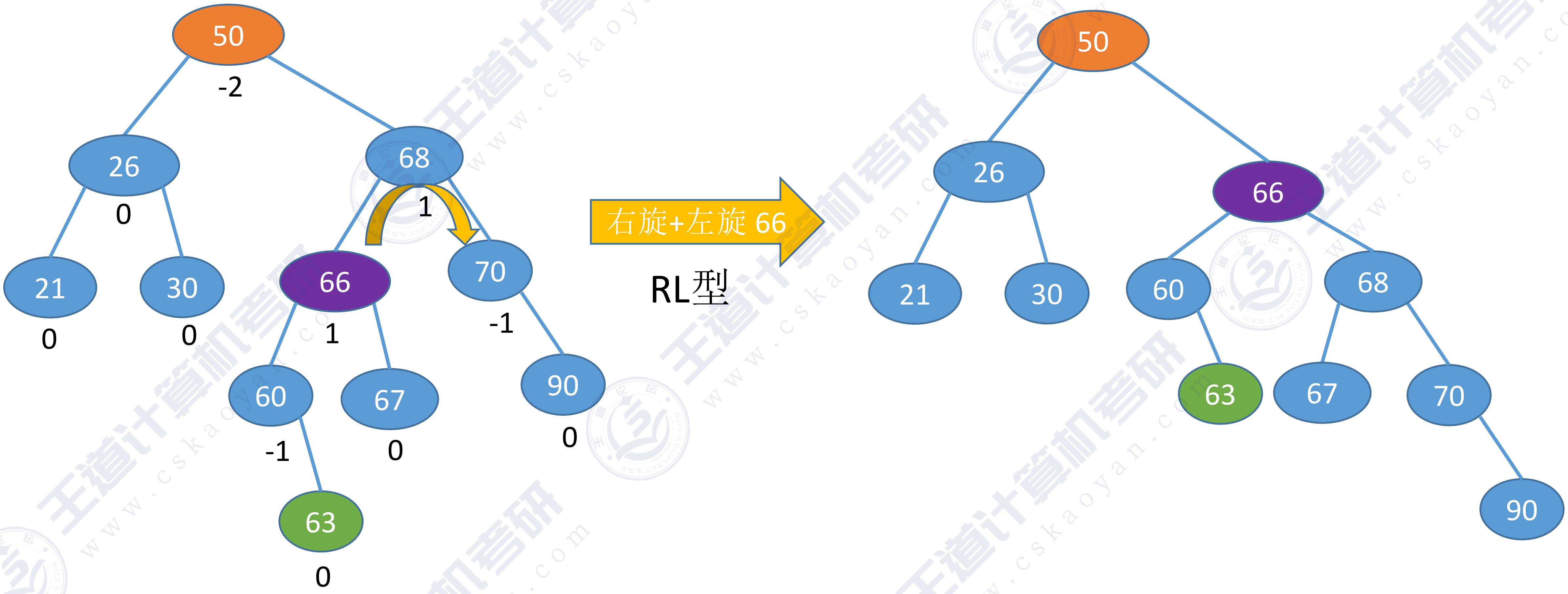
注意检查：是否符合 左<根<右

练习

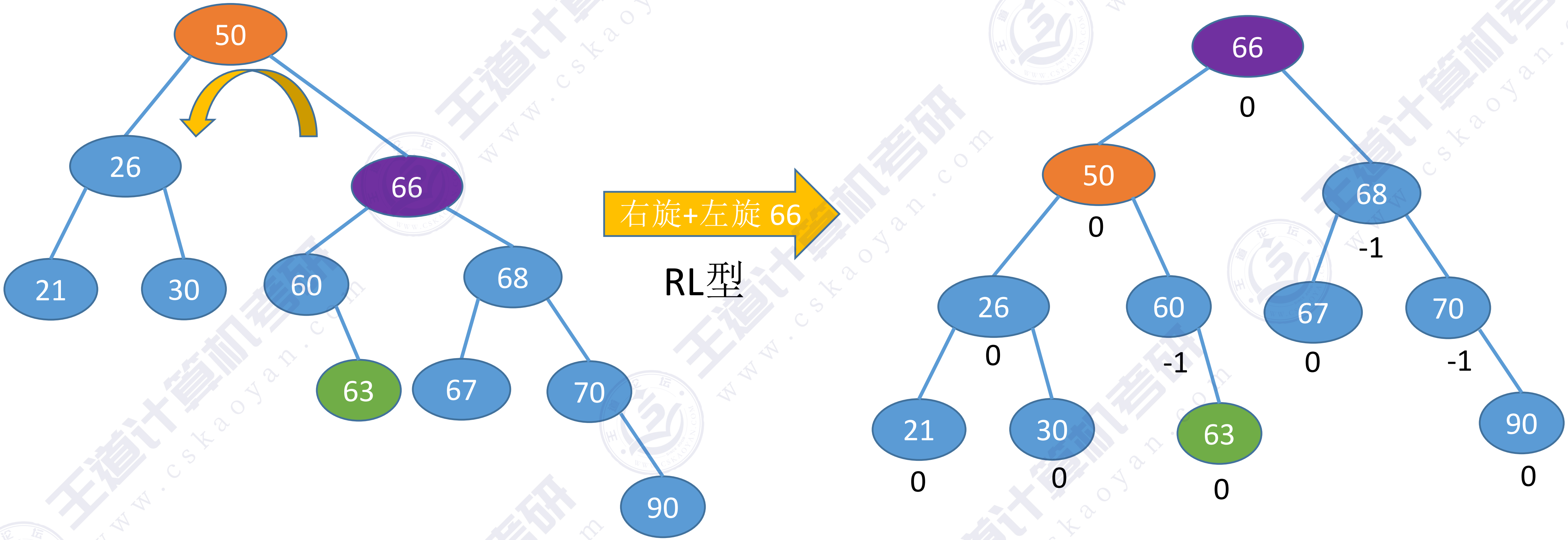


50
-1
26
0
68
0
21
0
30
0
66
0
70
-1
60
0
67
0
90
0
插入 63
RL型
50
-2
26
68
1
21
30
66
1
70
60
-1
67
90
63
0

练习

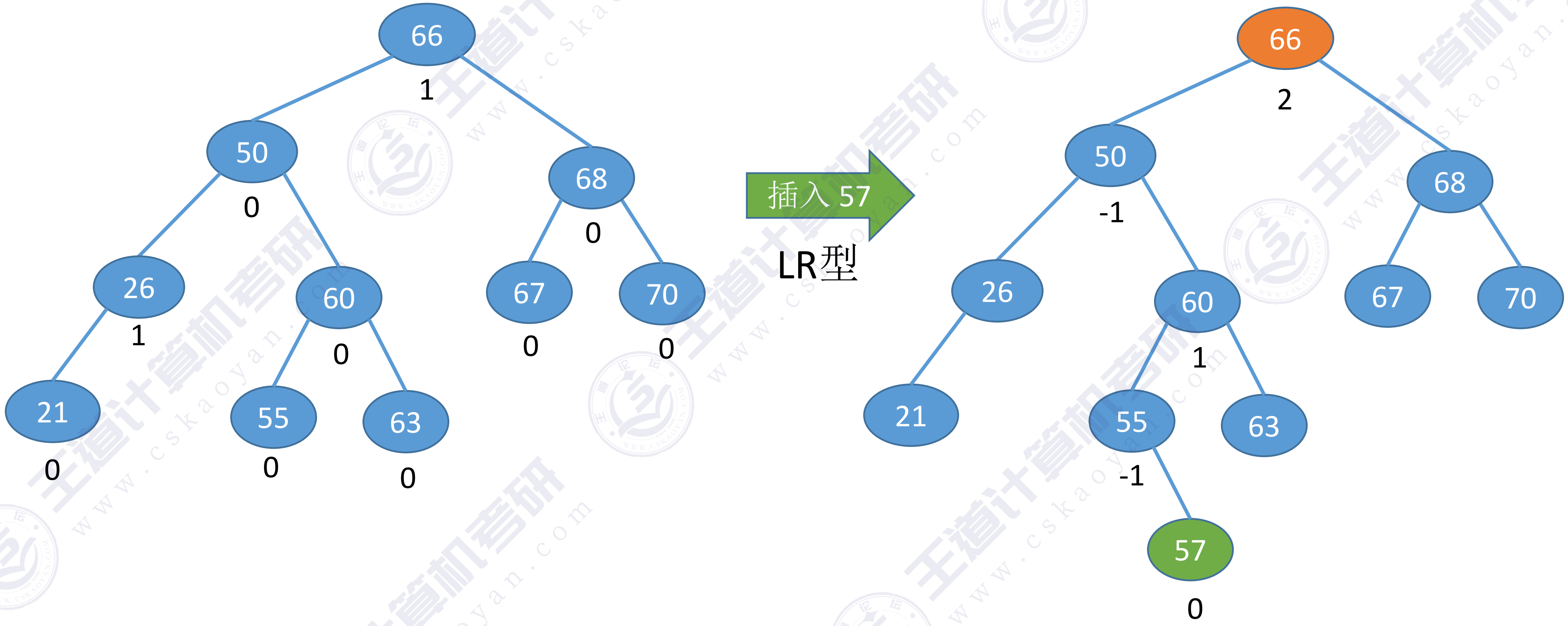

50
-2
26
0
68
1
21
0
30
0
66
1
70
-1
60
-1
67
0
90
0
63
0
右旋+左旋 66
RL型
50
26
66
21
30
60
68
63
67
70
90

练习

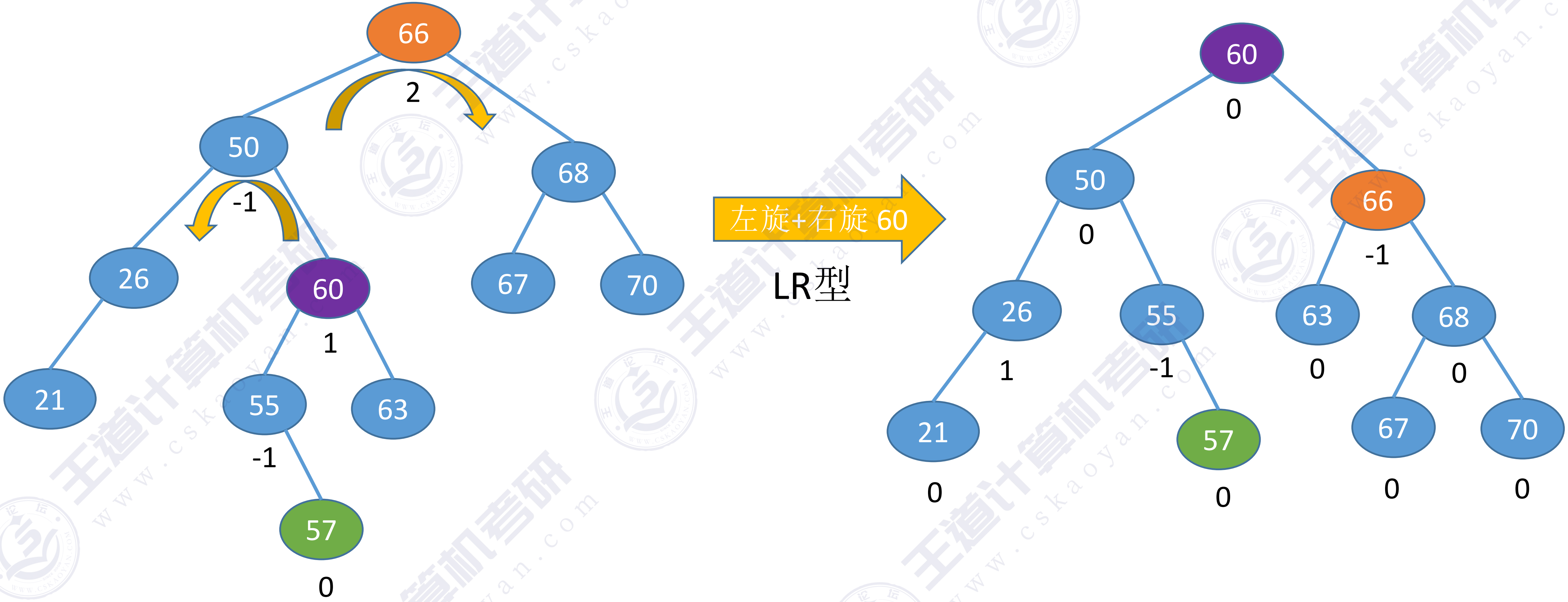


注意检查：是否符合 左<根<右

练习


66
1
50
0
68
0
26
1
60
0
67
0
70
0
21
0
55
0
63
0
插入 57
LR型
66
2
50
-1
68
26
60
1
67
70
21
55
-1
63
57
0

练习


66
2
50
-1
68
26
60
67
70
1
21
55
63
-1
57
0
左旋+右旋 60
LR型
60
0
50
0
66
-1
26
55
63
68
1
-1
0
0
21
57
67
70
0
0
0
0

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$O(\log_2 n)$

查找效率分析

《An algorithm for the organization of information》——G.M. Adelson-Velsky 和 E.M. Landis ,1962

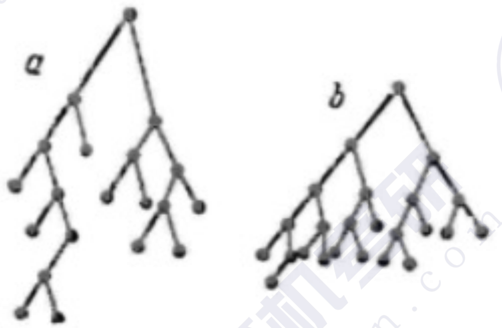


Figure 1

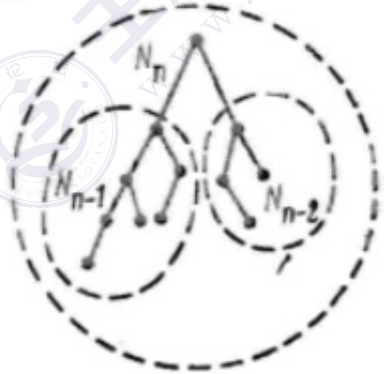


Figure 2

The recording algorithm is such that at each moment, the reference board is an admissible tree.

Lemma 1. *Let the number of cells of the admissible tree be equal to* N. *Then the maximum length of the branch is not greater than* $(3/2)\log_2(N+1)$.

Proof. Let us denote by N_n the minimum number of cells in the admissible tree when the given maximum length of the branch is n. Then it can be easily proven (see Figure 2) that $N_n = N_{n-1} + N_{n-2} + 1$.

When we solve this equation in finite remainders, we get

$$N_n=\left(1+\frac{2}{\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^n+\left(1-\frac{2}{\sqrt{5}}\right)\left(\frac{1-\sqrt{5}}{2}\right)^n-1.$$

Whence

$$n<\log_{\frac{1+\sqrt{5}}{2}}(N+1)<\frac{3}{2}\log_2(N+1),$$

知识回顾与重要考点

- 平衡二叉树
  - 定义
    - 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
  - 插入操作
    - 和二叉排序树一样，找合适的位置插入
    - 新插入的结点可能导致其祖先们平衡因子改变，导致失衡
  - ⚠ 调整“不平衡”
    - 找到最小不平衡子树进行调整，记最小不平衡子树的根为A
    - LL
      - 在A的左孩子的左子树插入导致A不平衡，将A的左孩子右上旋
    - RR
      - 在A的右孩子的右子树插入导致A不平衡，将A的右孩子左上旋
    - LR
      - 在A的左孩子的右子树插入导致A不平衡，将A的左孩子的右孩子 先左上旋再右上旋
    - RL
      - 在A的右孩子的左子树插入导致A不平衡，将A的右孩子的左孩子 先右上旋再左上旋
  - 查找效率分析
    - 考点：高为h的平衡二叉树最少有几个结点——递推求解
    - 平衡二叉树最大深度为 O(log n)，平均查找长度/查找的时间复杂度为 O(log n)

⚠ 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他爹
① f->lchild = p->rchild;
② p->rchild = f;
③ gf->lchild/rchild = p;

⚠ 实现 f 向左下旋转， p 向左上旋转：
① f->rchild = p->lchild;
② p->lchild = f;
③ gf->lchild/rchild = p;