上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

第9章 动态查找表

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

二叉查找树

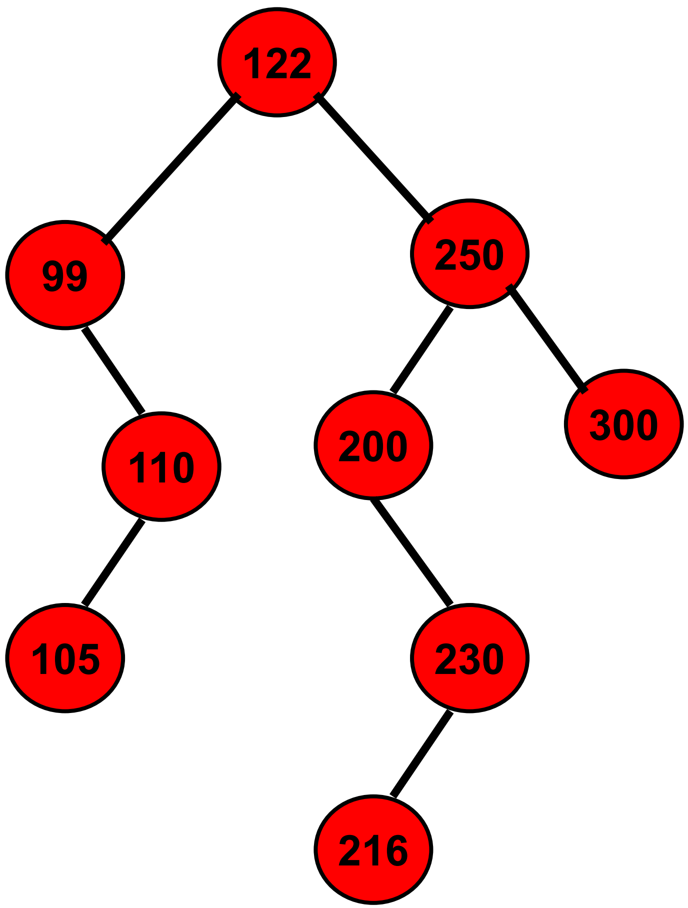
- 二叉查找树的定义
- 二叉查找树的存储实现
- 二叉查找树的操作
- 二叉查找树的性能

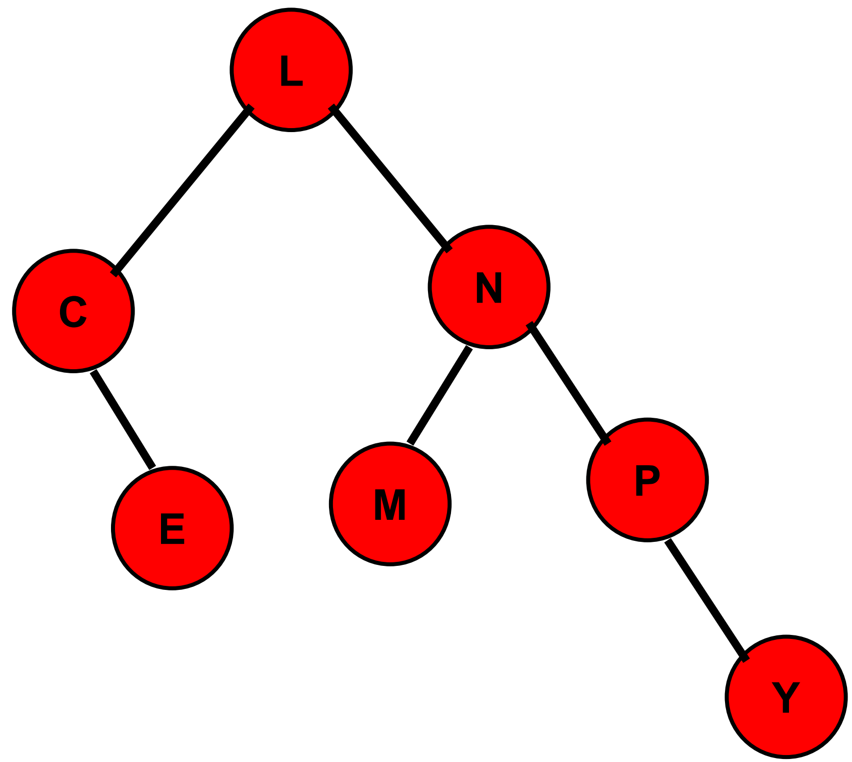
二叉查找树

二叉查找树是二叉树在查找方面的重要应用。二叉查找树或者为空，或者具有如下性质：对任意一个结点p而言

- 如果p的左子树若非空，则左子树上的所有结点的关键字值均小于p结点的关键字值。
- 如果p的右子树若非空，则右子树上的所有结点的关键字值均大于p结点的关键字值。
- 结点p的左右子树同样是二叉查找树。

举例：二叉查找树





中序遍历一棵二叉查找树所得到的序列是按键值递增的，因此二叉查找树也可以用来排序。

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

二叉查找树

- 二叉查找树的定义
- 二叉查找树的存储实现
- **二叉查找树的操作**
- 二叉查找树的性能

SHANGHAI JIAO TONG UNIVERSITY

插入

- 若二叉树为空，则新插入的结点成为根结点。
- 如二叉树非空
  - 首先执行查找算法，找出被插结点的父亲结点。
  - 判断被插结点是其父亲结点的左、右儿子。将被插结点作为叶子结点插入。

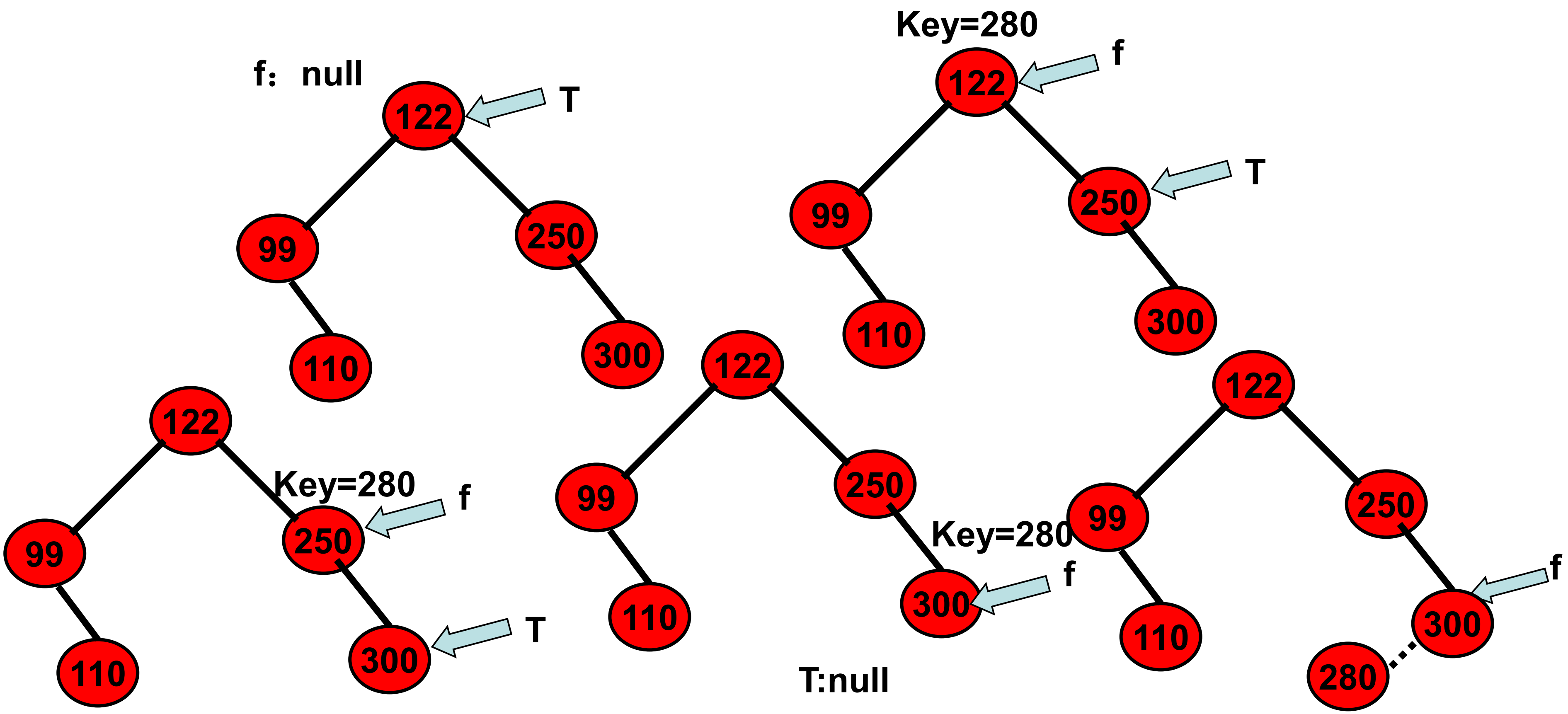
注意：新插入的结点总是叶子结点

SHANGHAI JIAO TONG UNIVERSITY

插入操作的递归实现

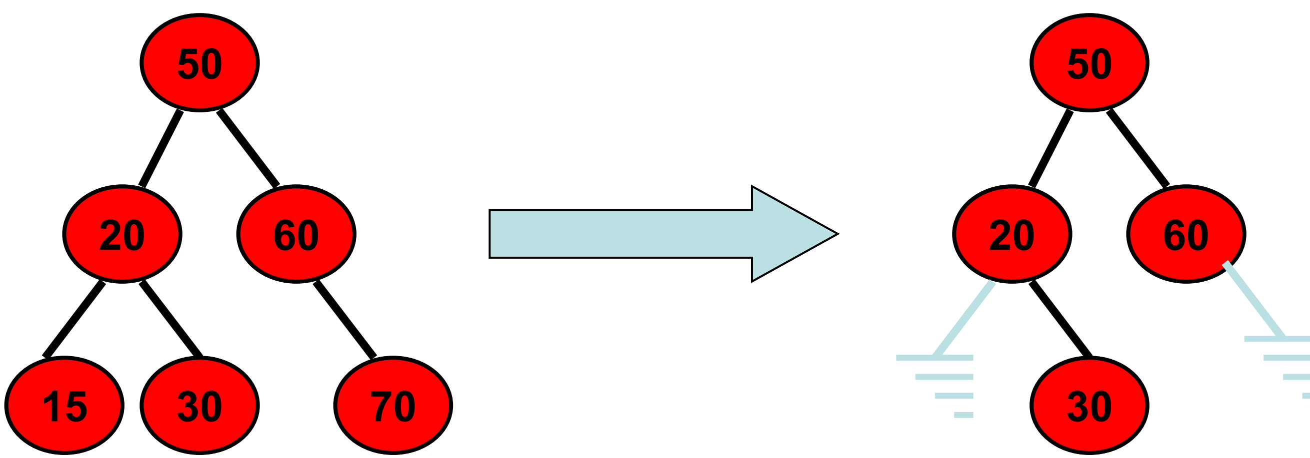
- 如果当前树为空，插入值作为树的根节点，返回
- 如果插入值小于根节点，插入到左子树，否则插入到右子树

执行实例：插入值为 **280** 的结点



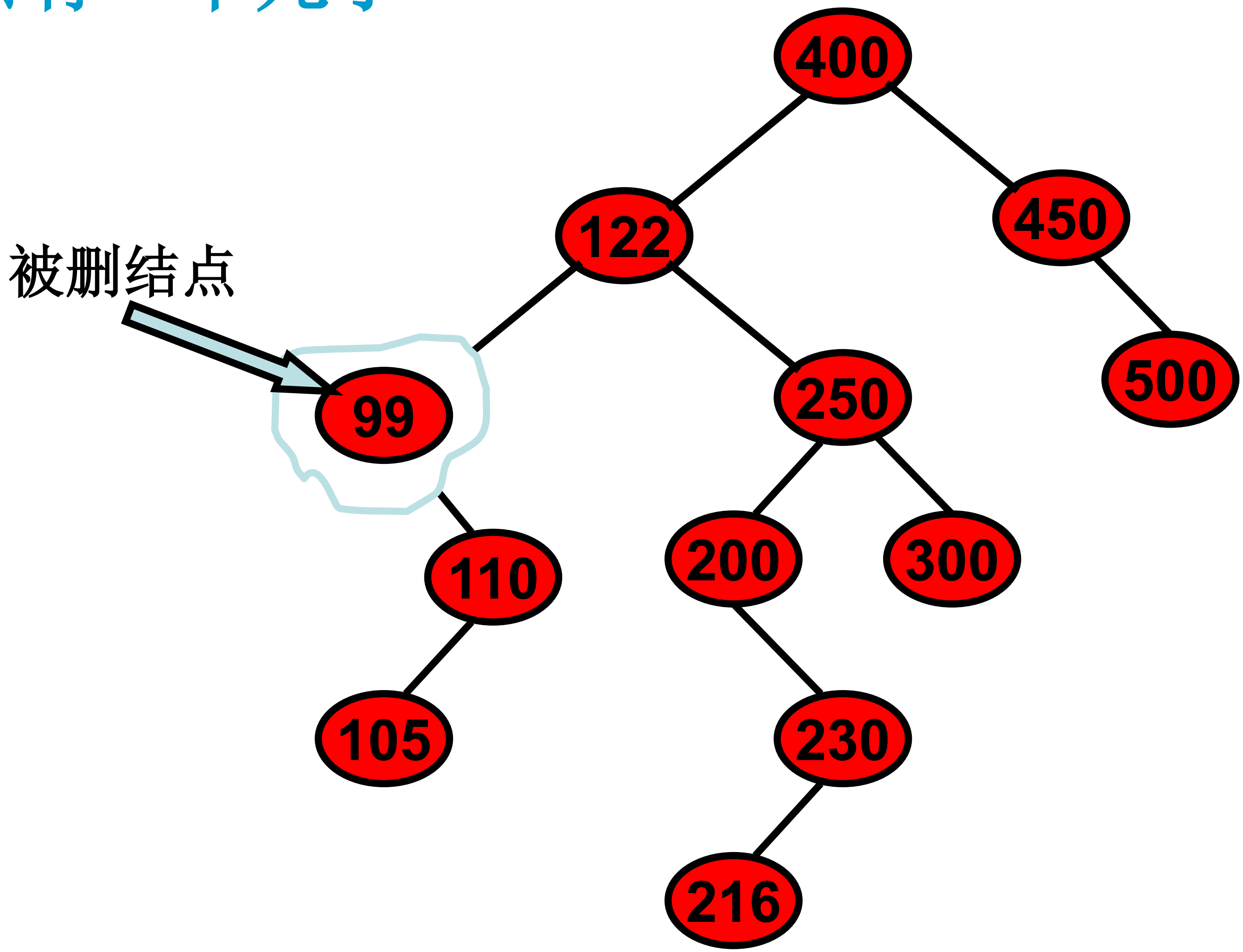
删除叶结点

直接删除，更改它的父亲结点的相应指针字段为空。这不会改变二叉查找树的特性。如：删除数据字段为 15、70 的结点。

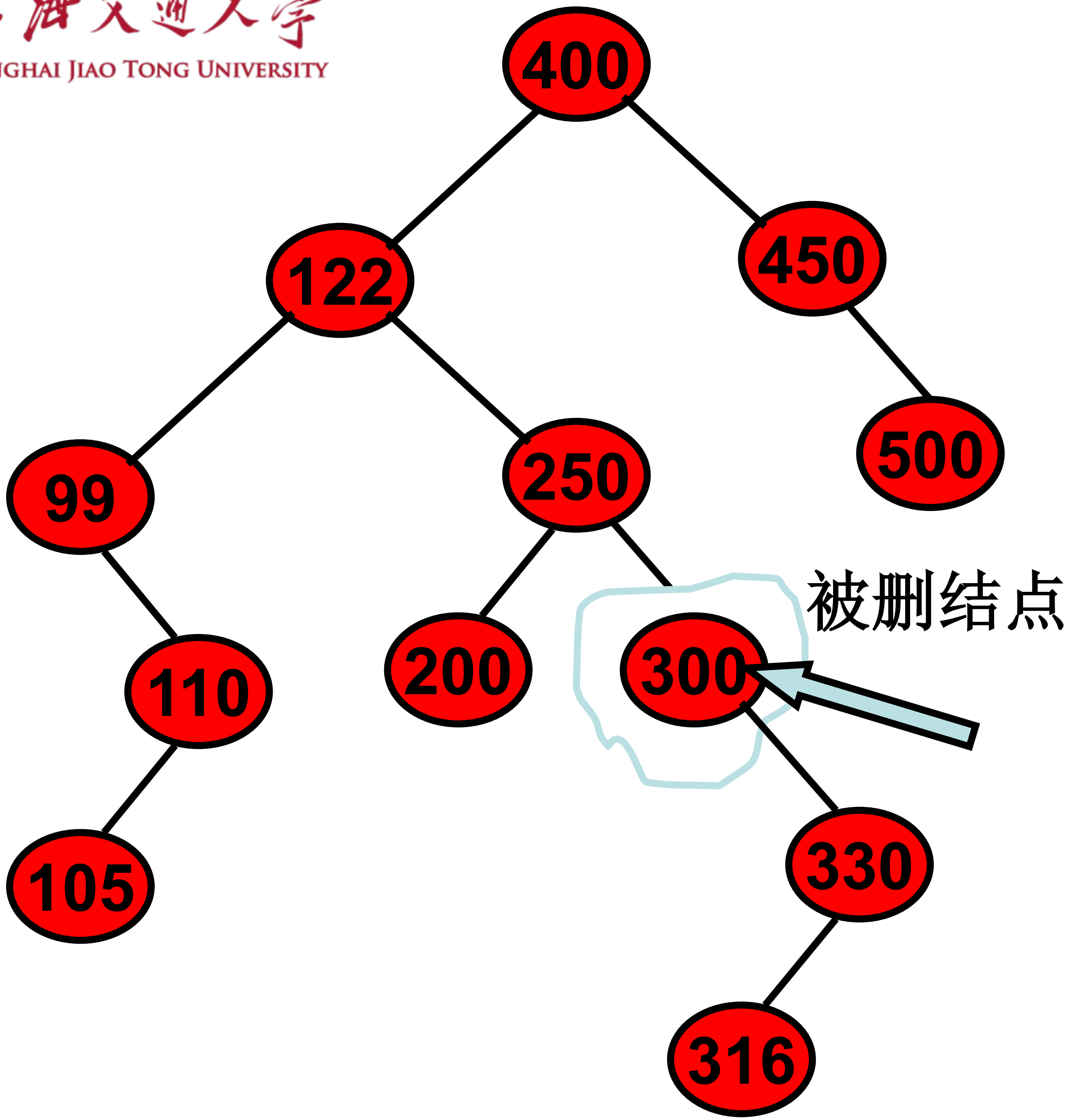


删除

只有一个儿子



删除



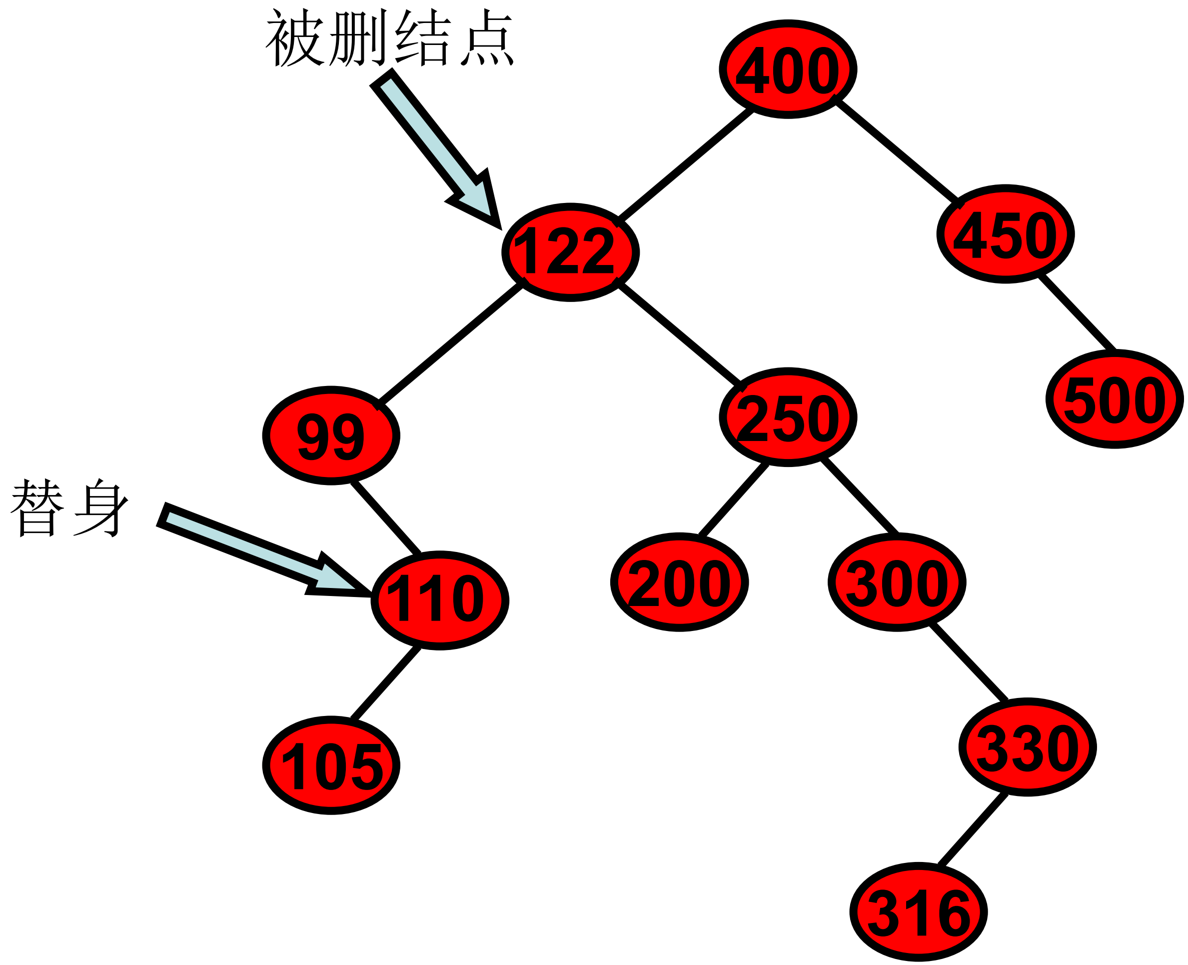
- 若被删结点只有一个唯一的儿子，将此儿子取代被删结点的位置。
- 能保持二查查找树的有序性

被删结点有两个儿子

删除这个结点会使其他结点从树上脱离。

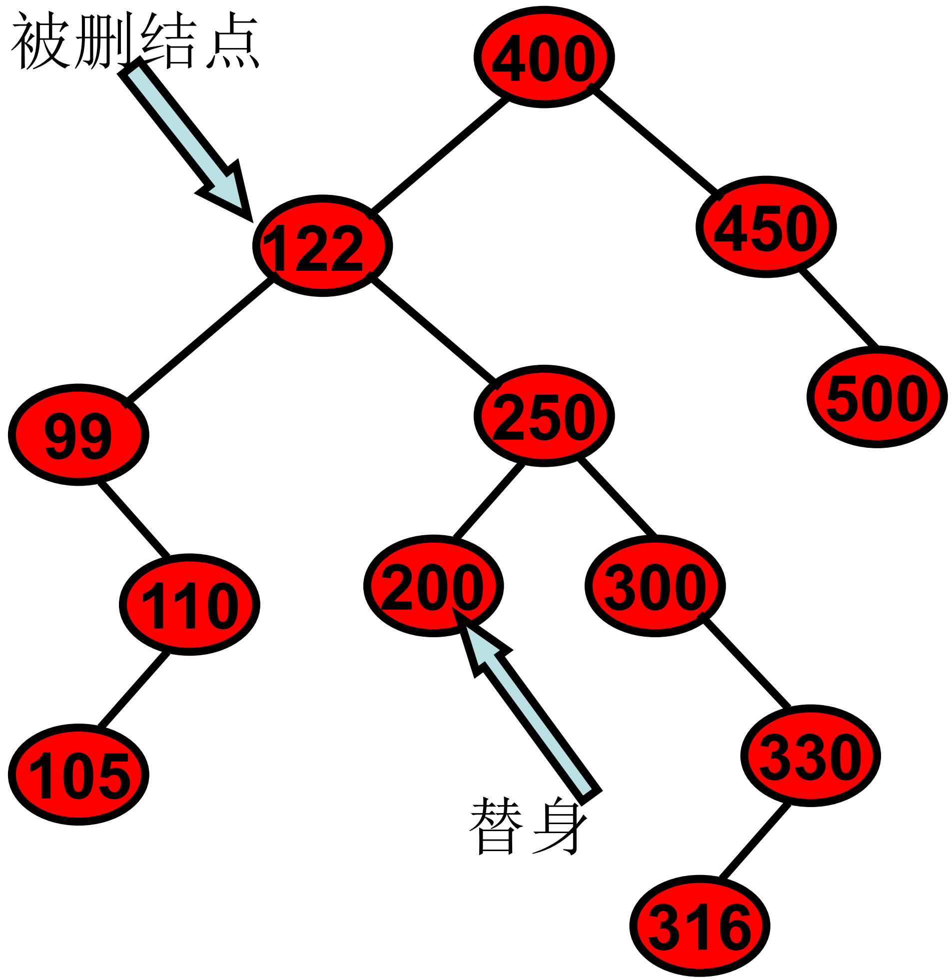
- 通常的做法：选取“替身”取代被删结点。有资格充当该替身的是谁哪?
- 替身的要求：维持二叉查找树的特性不变。因此，只有在中序周游中紧靠着被删结点的结点才有资格作为“替身”，即，**左子树中最大的结点** 或 **右子树中最小的结点**。

用左子树中的最大值做替身



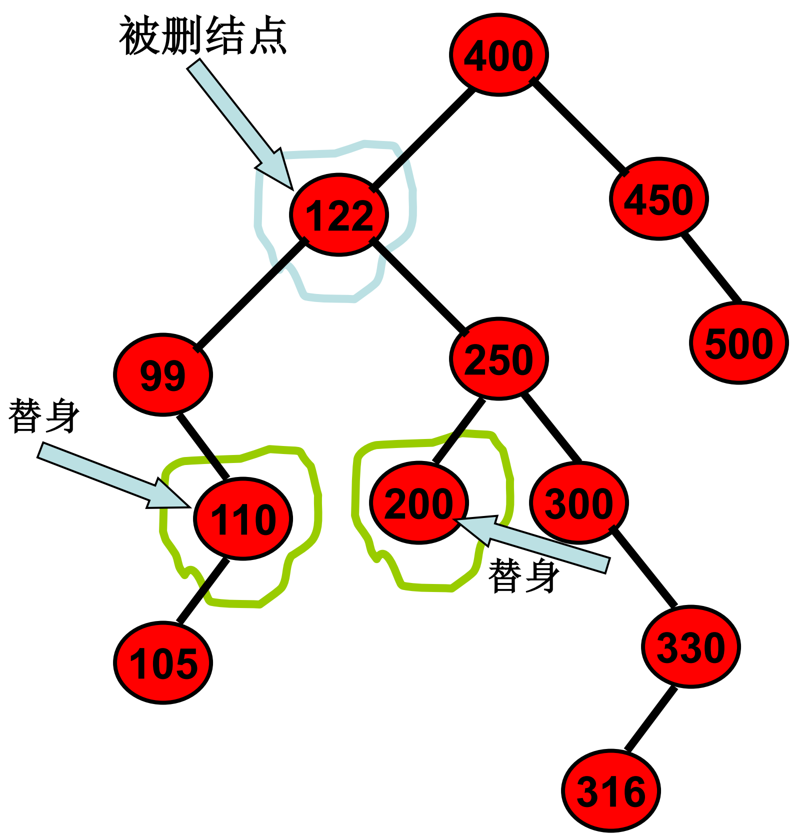
做法：将替身110的数据字段复制到被删结点的数据字段。
将结点 110 的左儿子作为 99 的右儿子。

用右子树的最小值做替身



做法：将替身的数据字段复制到被删结点的数据字段。将结点 200 的右儿子作为200 的父结点的左儿子。

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY



结论：

•先将替身的数据字段复制到被删结点

•将原替身的另一儿子作为原替身父亲结点的儿子

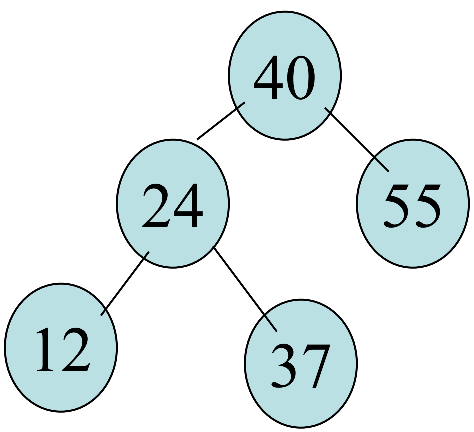
•释放原替身结点的空间。

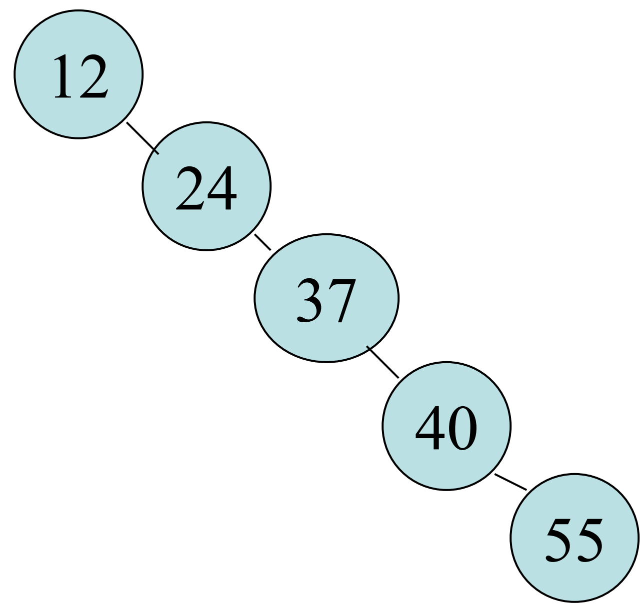
二叉查找树

- 二叉查找树的定义
- 二叉查找树的存储实现
- 二叉查找树的操作
- **二叉查找树的性能**

二叉查找树性能

- 二叉查找树的操作（包括insert、find和remove等）的代价正比于操作过程中要访问的结点数。如果所操作的二叉查找树是完全平衡的，那么访问的代价将是对数级别的
- 在最坏的请况下，二叉查找树会退化为一个单链表。时间复杂度是O(N)





平均性能

- n个结点二叉查找树的可能有n种形态，如果这些形态出现的概率是相等的。设P(n)为查找n个结点的二叉查找树的平均查找时间，则

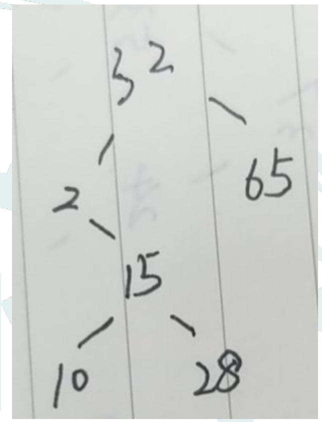
$$P(n)=\frac{\sum_{i=0}^{n-1}[1+(P(i)+1)*i+(P(n-i-1)+1)*(n-i-1)]}{n^2}$$

$$\le 2(1+\frac{1}{n})\ln n$$

$$\approx 1.38\log n$$

1.将{ 32, 2, 15, 65, 28, 10 }依次插入初始为空的二叉搜索树。则该树的前序遍历结果是：（ B ）

A. 32, 2, 10, 15, 28, 65
B. 32, 2, 15, 10, 28, 65
C. 10, 28, 15, 2, 65, 32
D. 2, 10, 15, 28, 32, 65



2. 查找效率最高的二叉排序树为（C）。

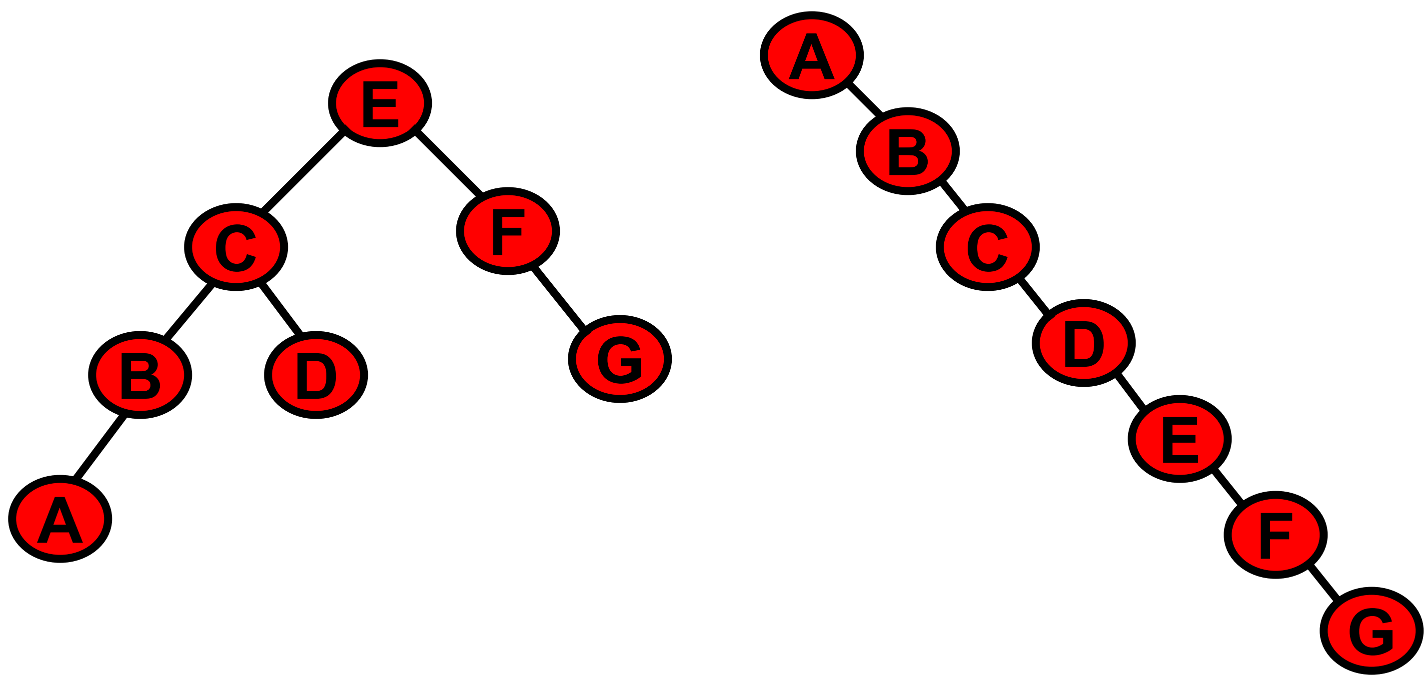
A.所有结点的左子树都为空的二叉排序树
B.所有结点的右子树都为空的二叉排序树
C.平衡二叉树
D.没有左子树的二叉排序树

AVL树

- AVL树的定义
- AVL树的存储实现
- AVL树的查找
- AVL树的插入
- AVL树的删除

平衡二叉查找树

当树退化为链表时，树的优点荡然无存。要使查找树的性能尽可能好，就要使得树尽可能丰满。要构造一个丰满树很困难，一种替代的方案是平衡树。

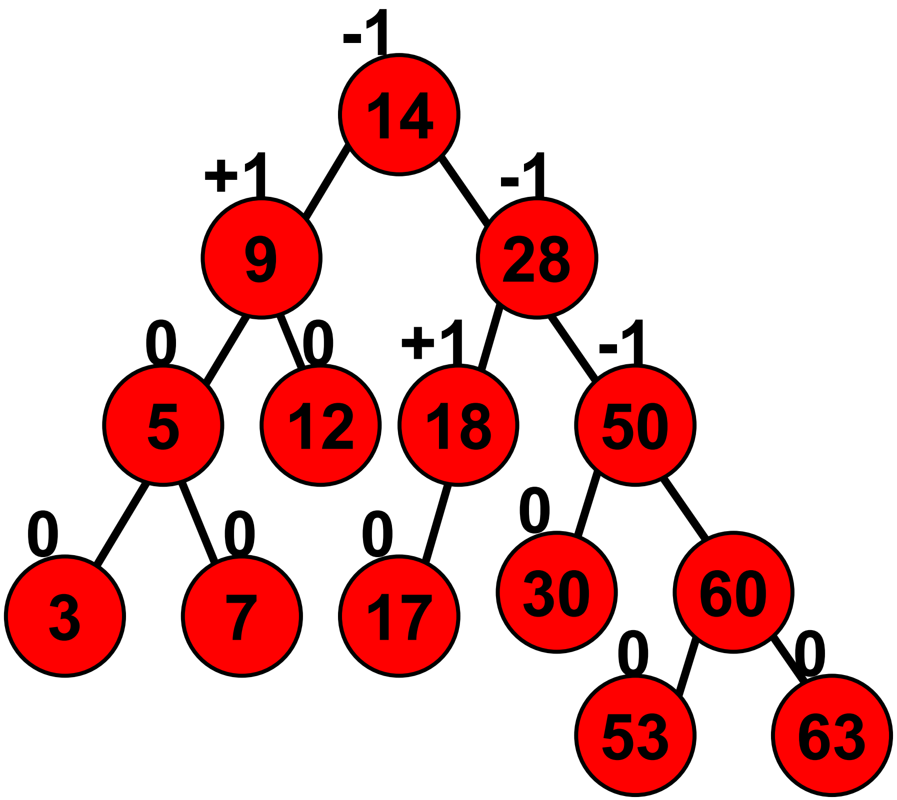


平衡条件

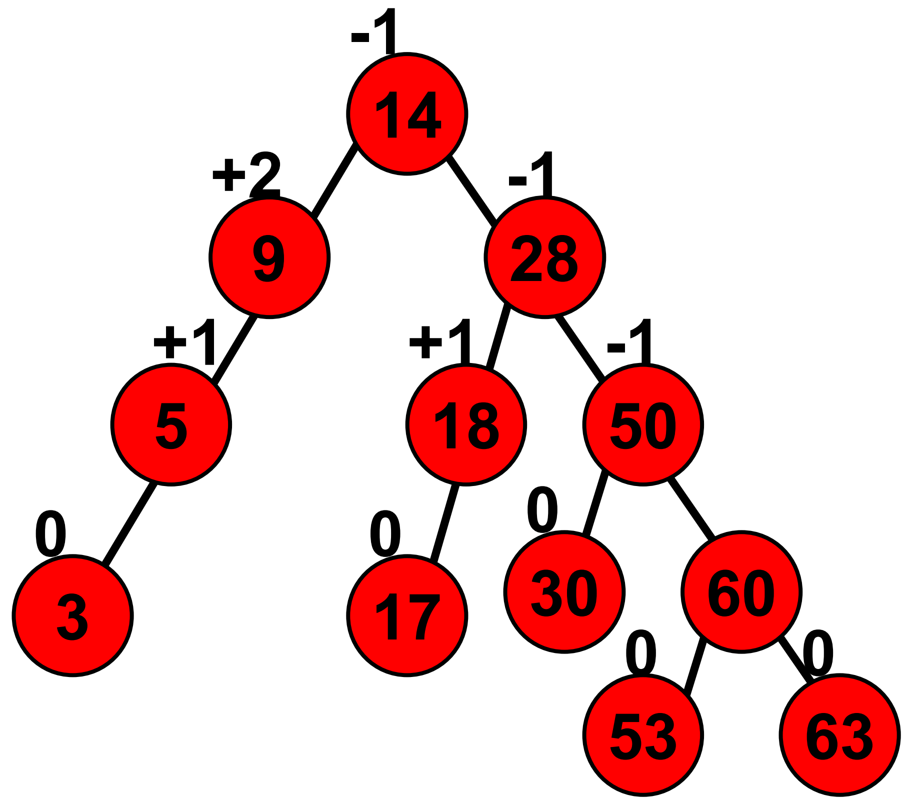
- 最简单的想法是让左右子树有同样的高度，但它并不能保证树是矮胖的。
- 另一个想法是要求每个节点的左右子树都有同样的高度。这个条件能保证树是矮胖的，但这个条件太苛刻，只有满二叉树才满足这个条件。
- 将上述条件稍微放宽一些就是二叉平衡查找树。

平衡树的定义

- 平衡因子（平衡度）：结点的平衡度是结点的左子树的高度 - 右子树的高度。
- **空树的高度定义为-1。**
- 平衡二叉树：每个结点的平衡因子都为 +1、 -1、0 的二叉树。或者说每个结点的左右子树的高度最多差1的二叉树。
- 可以证明平衡树的高度至多约为：1.44log(N+2) -1.328



是平衡树不是丰满树



不是平衡树

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

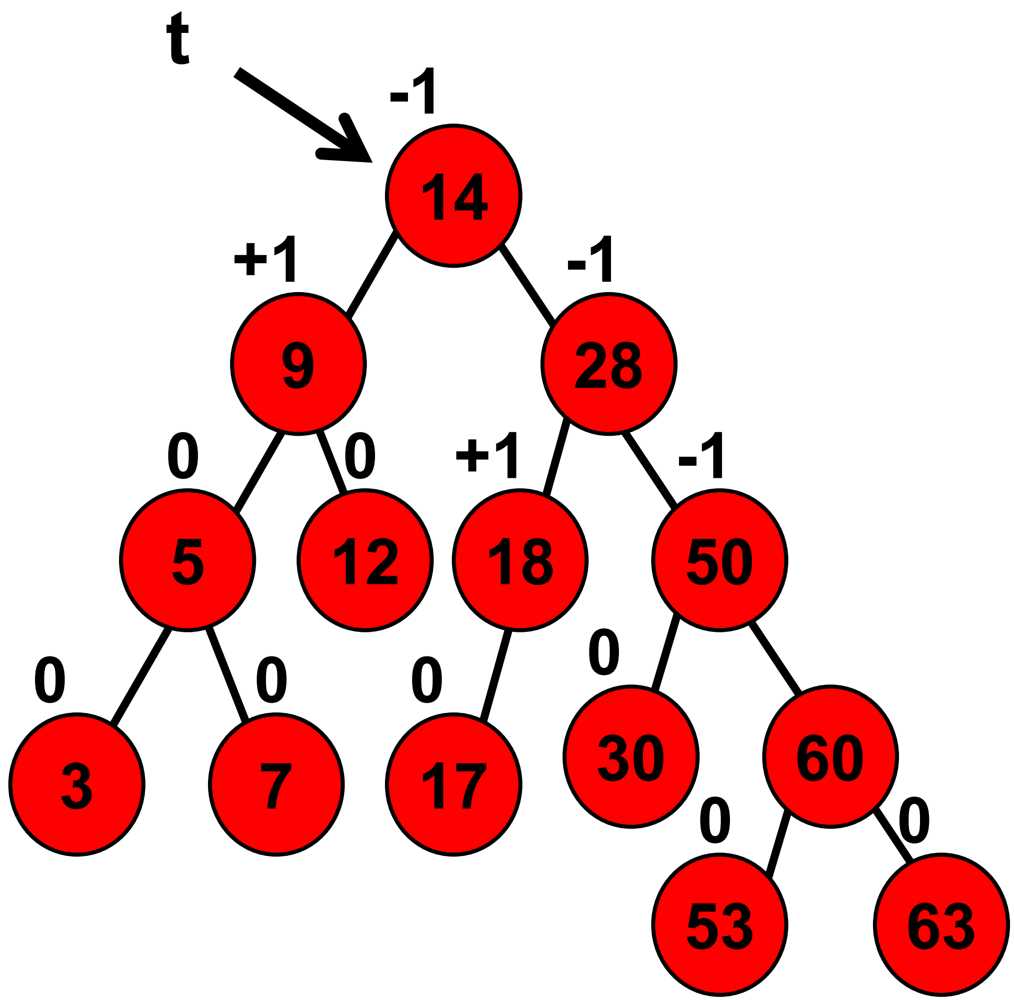
AVL树的查找

- 查找过程与二叉查找树完全相同
- 二叉查找树类采用递归实现
- AVL树展示非递归的实现

查找的非递归实现

- 设当前结点为根结点。
- 只要当前结点非空，并且当前结点的值不等于被查找的元素，则根据当前结点和被查元素的的大小，将新的当前结点设为原当前结点的左孩子或右孩子。
- 当前结点为空或当前结点的值等于被查元素，查找停止。
- 如果是因为当前结点为空而停止查找的，返回空指针，表示没有找到；否则表示找到了该结点，返回该结点的地址。

查找12



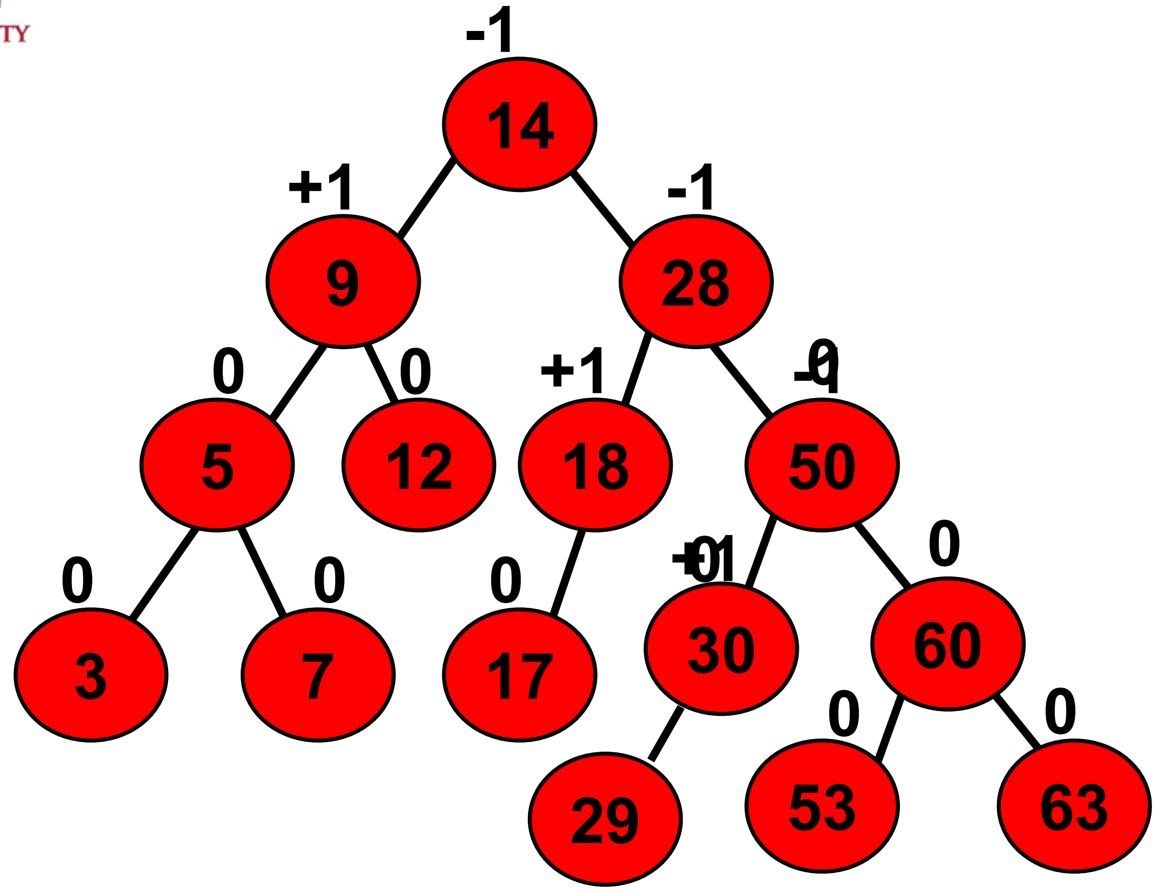
SHANGHAI JIAO TONG UNIVERSITY

插入操作

- 插入过程与二叉查找树的插入相同，只是插入后可能出现两种情况：
  - 插入后，不破坏平衡性，只是改变了树根到插入点的路径上的某些结点的平衡度，因此需要自底向上修改节点的平衡度
  - 破坏了路径上的某些结点的平衡性，需要向上调整树的结构

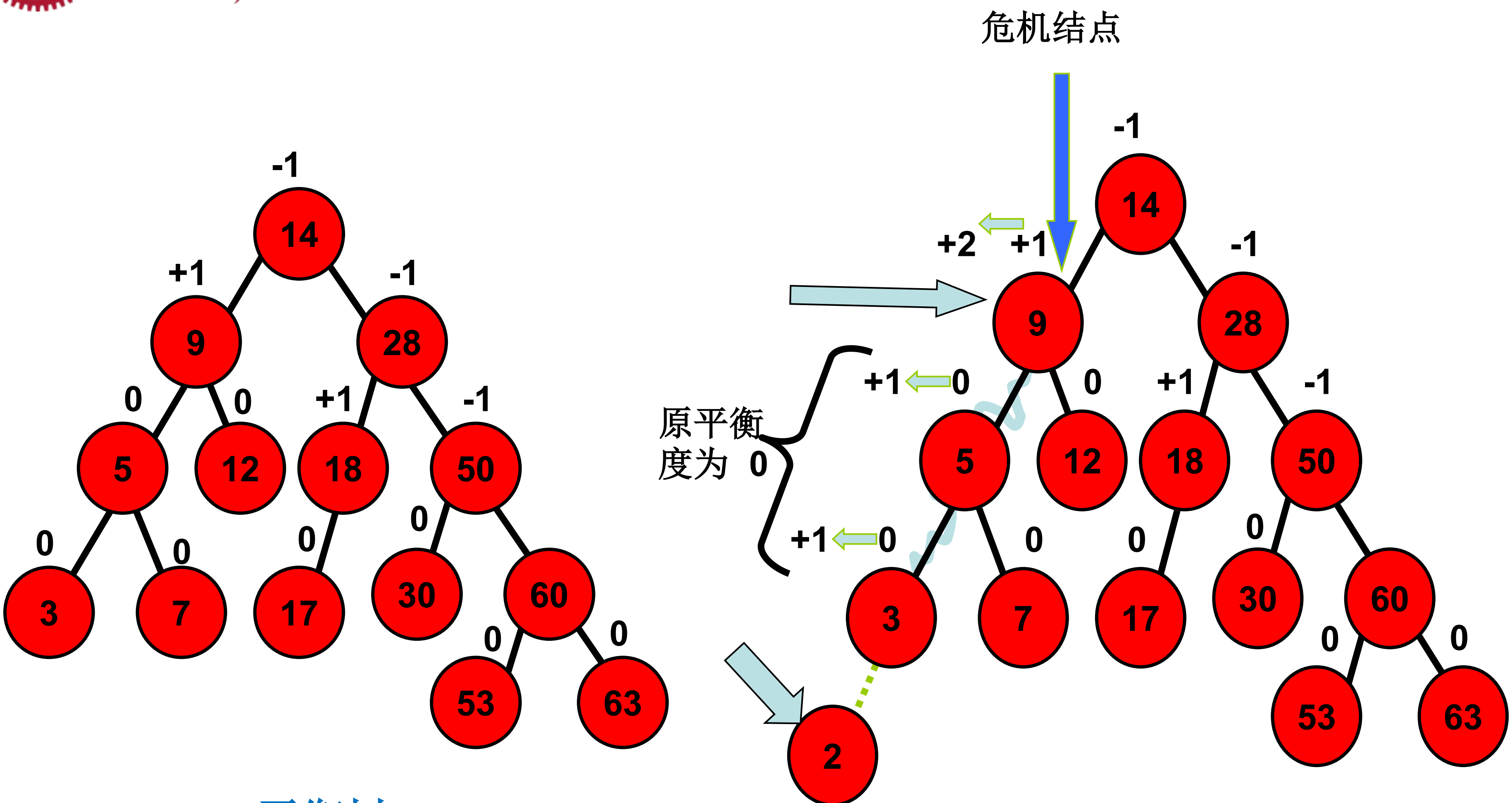
SHANGHAI JIAO TONG UNIVERSITY

插入29



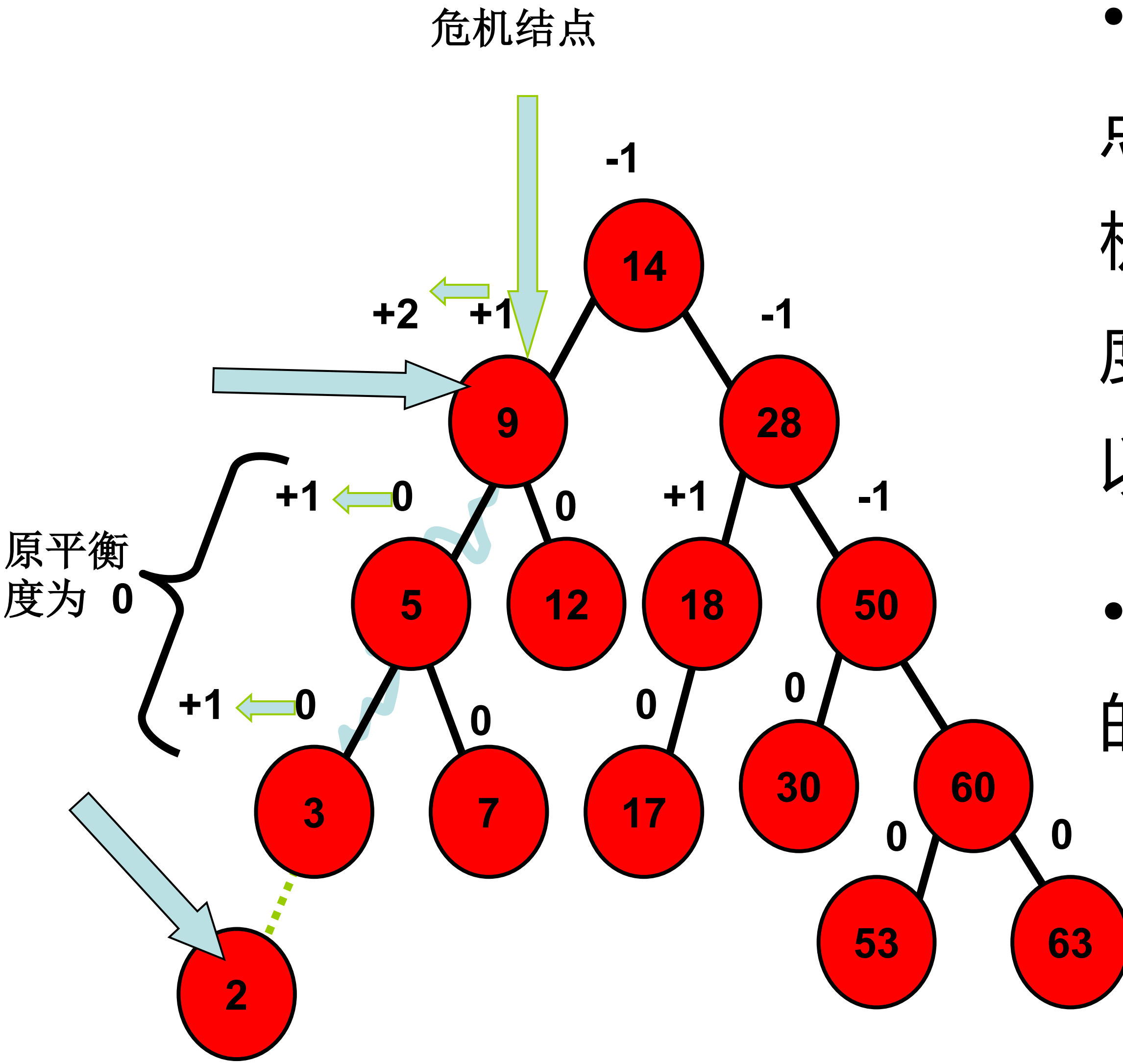
只改变了某些结点的平衡度，需要自底向上的调整。只要有一个节点的平衡度不变，它上面的节点的平衡度也不变。调整可以结束。

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY



平衡树

插入2以后，变得不平衡了。如何用最简单、最有效的办法保持平衡分类二叉树的性质不变？



调整要求：

- 希望不涉及到危机结点的父亲结点，即以危机结点为根的子树的高度应保持不变。调整可以到此结束。

- 仍应保持查找二叉树的性质不变

SHANGHAI JIAO TONG UNIVERSITY 上海交通大学

重新平衡

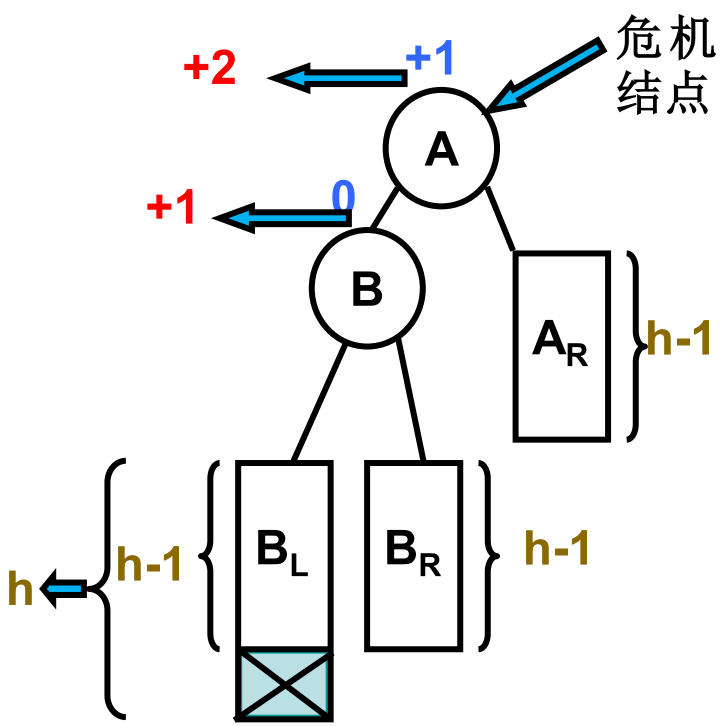
从插入位置向根回溯

- 如果节点原来的平衡度为0，则插入后不可能失衡，重新计算平衡度，继续往上回溯
- 如果节点原来的平衡度非0，可能成为失衡节点
  - 重新计算平衡度
  - 如果平衡度在合法范围，调整结束
  - 如果失去平衡，重新调整树的结构，调整结束

可能引起节点不平衡的情况

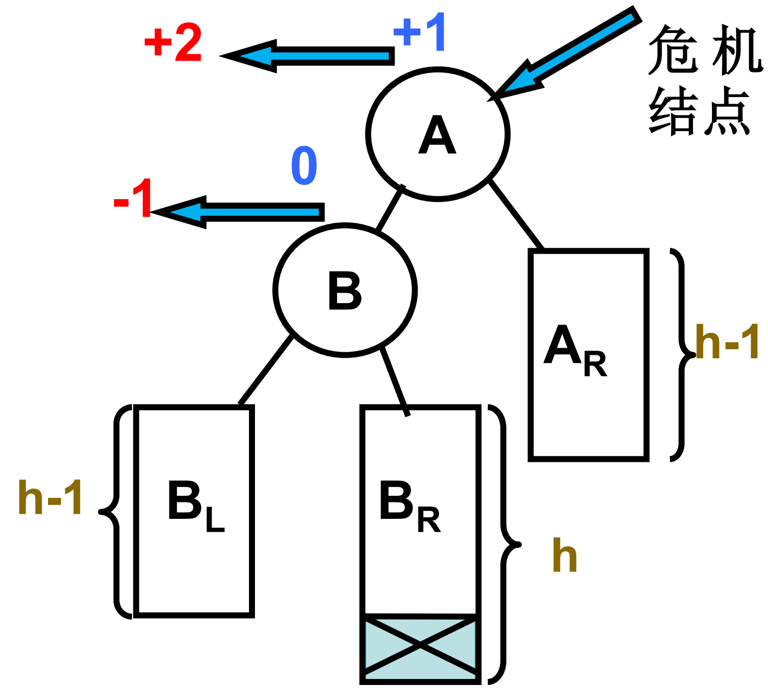
- 在节点的左孩子的左子树上插入（LL）
- 在节点左孩子的右子树上插入（LR）
- 在节点的右孩子的左子树上插入（RL）
- 在节点的右孩子的右子树上插入（RR）

可能引起不平衡的情况



LL

RR：LL的镜像对称



LR

RL：LR的镜像对称

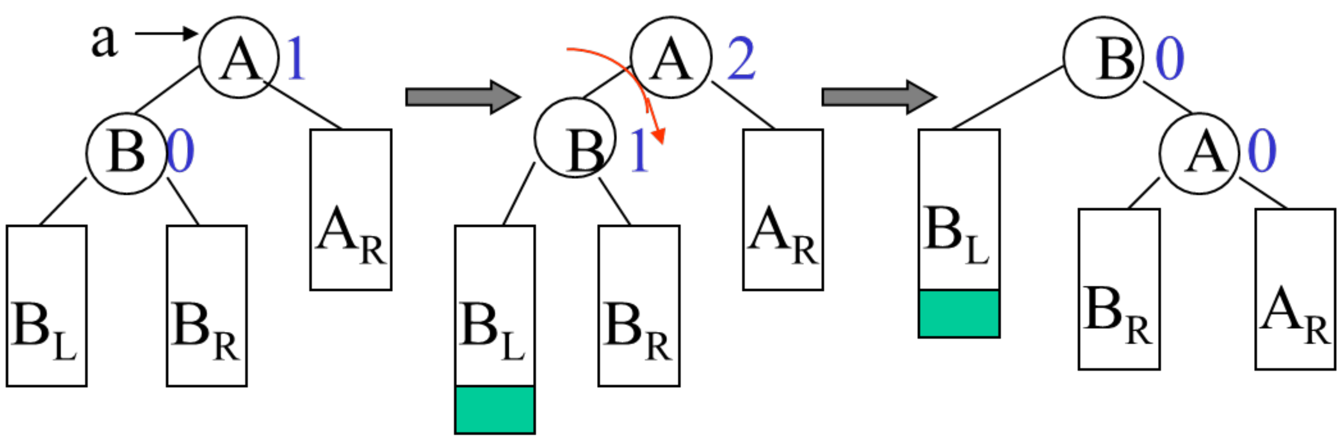
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

[调整范围的确定]

插入结点后，找到离插入结点最近且平衡因子绝对值超过1的祖先结点，则以该结点为根的子树将是可能不平衡的**最小子树**，可将重新平衡的范围局限于这棵子树。

[调整的规律] 设失去平衡的最小子树的根结点指针为a

- **LL型平衡旋转**——单向右旋（一次顺时针旋转）

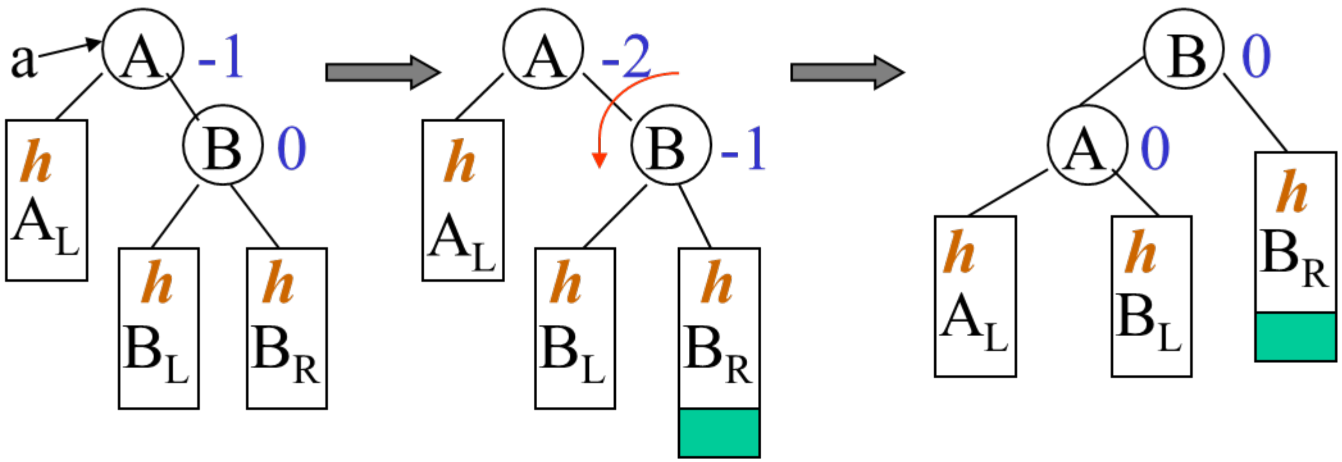


插入的结点

•保持了树的有序性

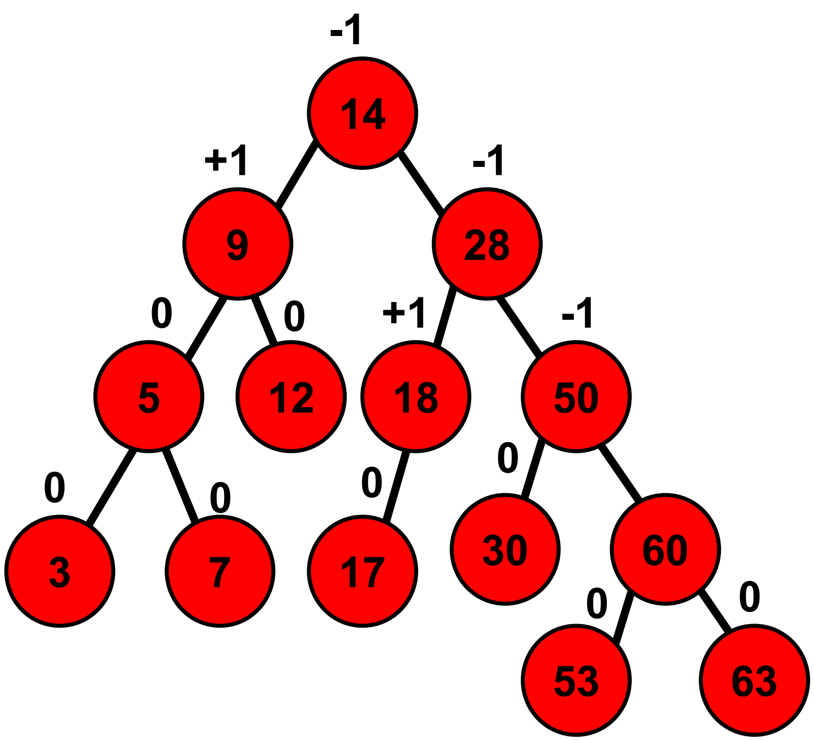
•保持了原先的高度

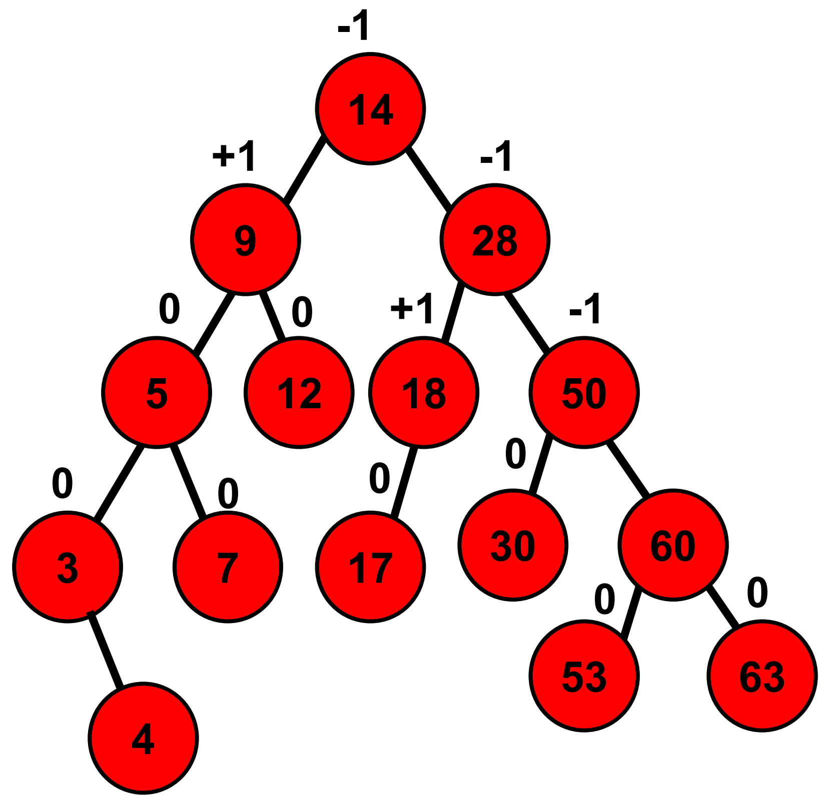
- **RR型平衡旋转**——单向左旋（一次逆时针旋转）



SHANGHAI JIAO TONG UNIVERSITY 上海交通大學

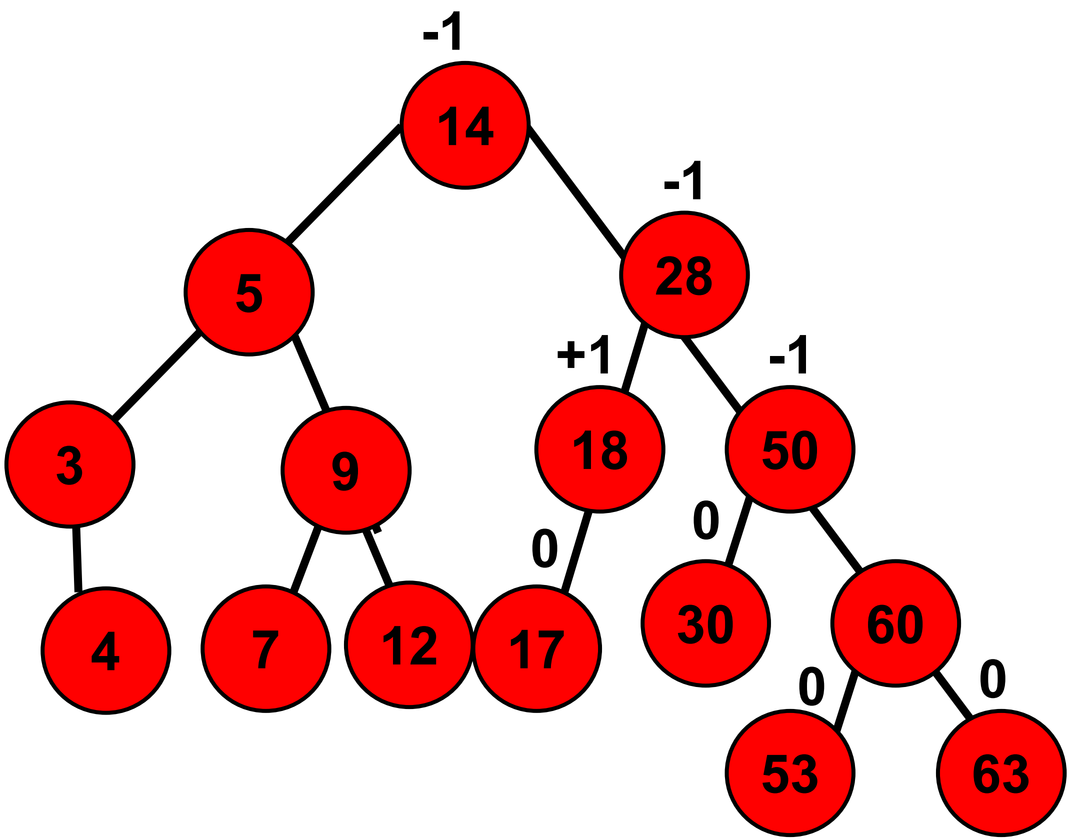
在下列树中插入4，将会使得9失去平衡。这是在9的左孩子的左子树上插入引起失衡，是LL情况





上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

旋转后的结果



- 保持了树的有序性
- 保持了原先的高度

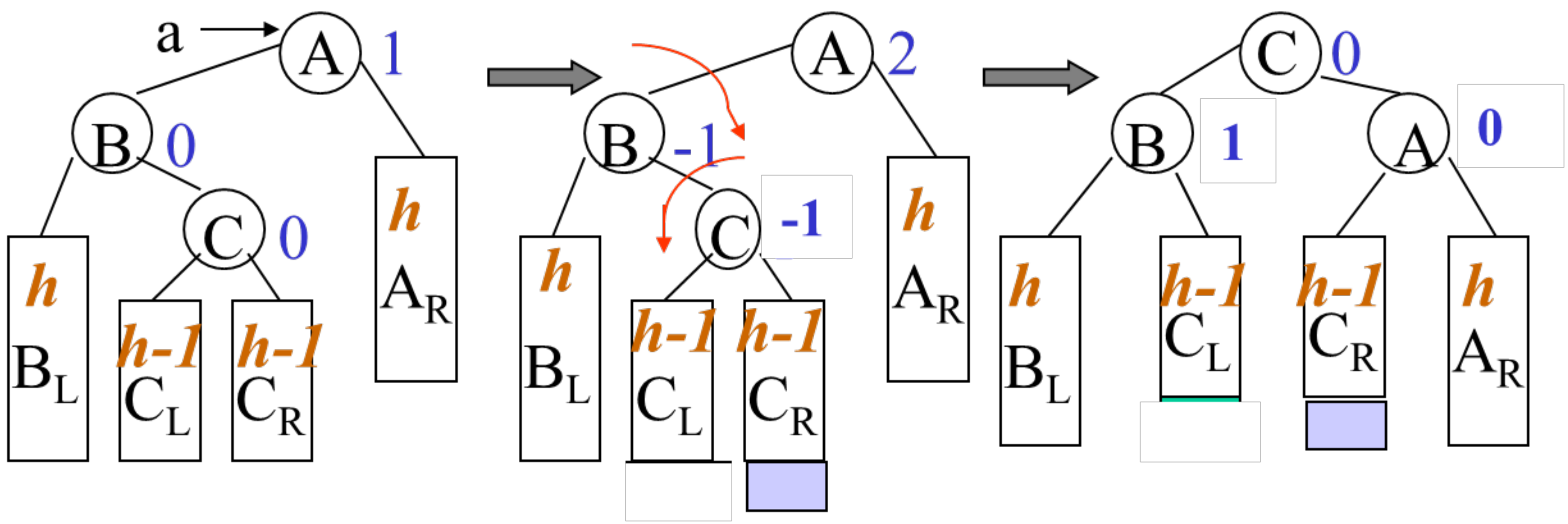
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

LR和RL问题

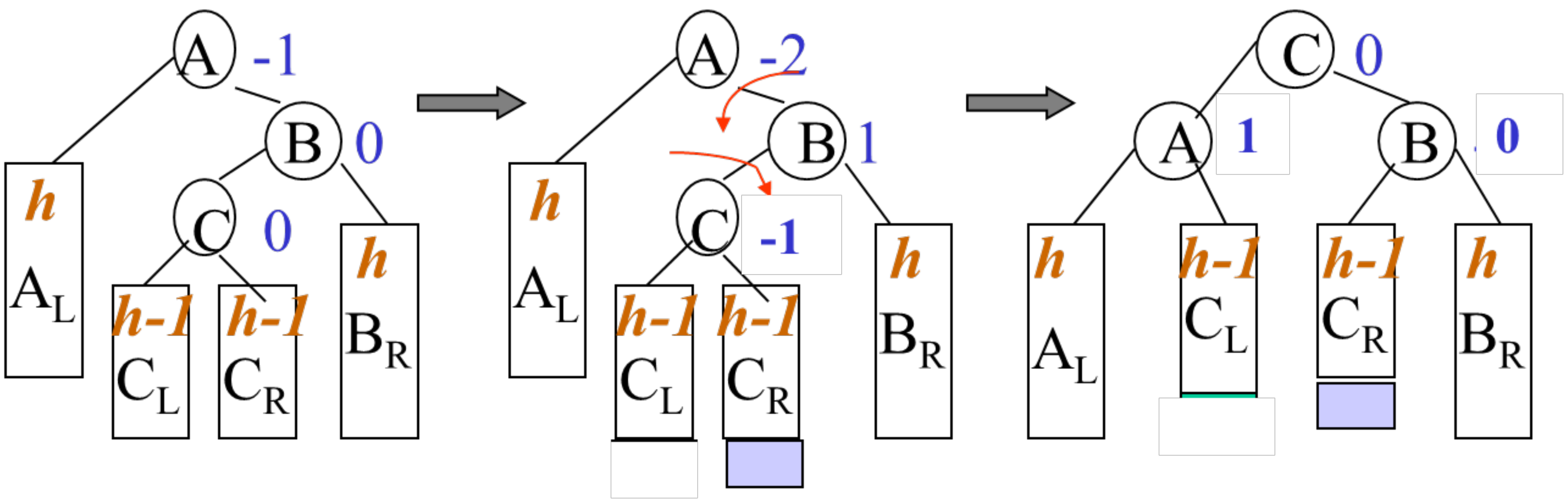
- 通过双旋转来解决，即两次单旋转。现对危机结点的儿子和孙子进行一次单旋转，使孙子变成儿子。然后是危机结点和新的儿子进行一次单旋转。

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

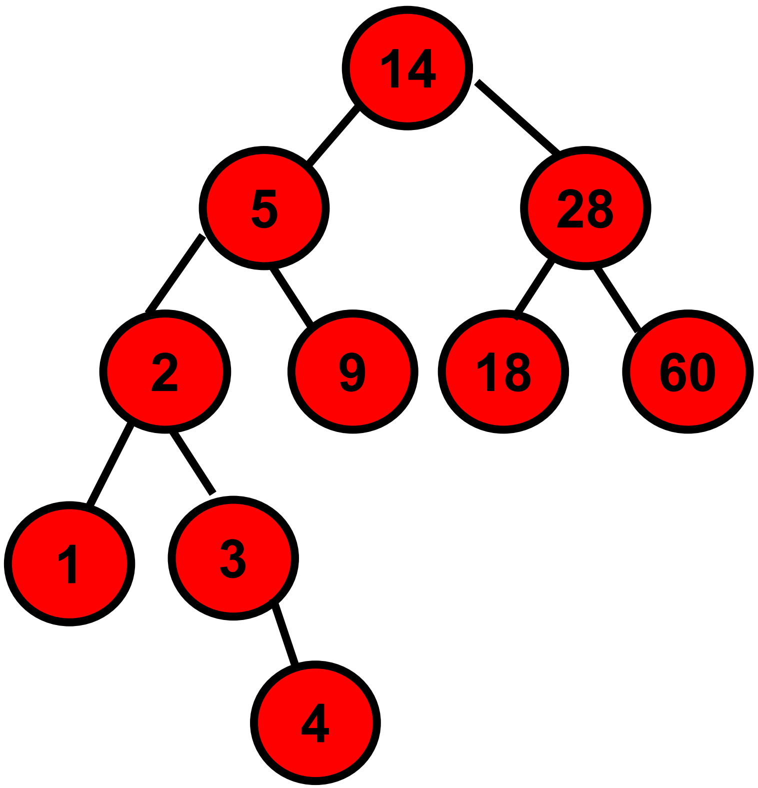
- **LR型平衡旋转**——一次逆时针旋转+一次顺时针旋转



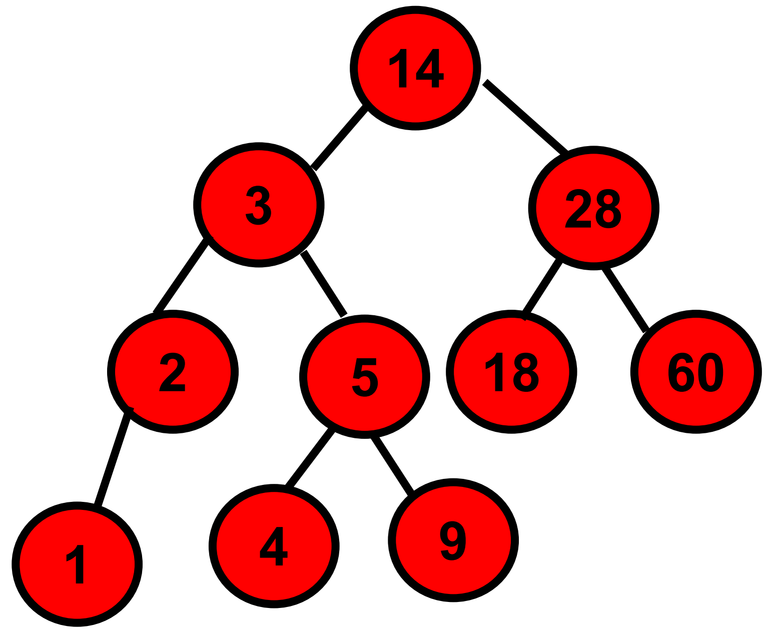
- **RL型平衡旋转**——一次顺时针旋转+一次逆时针旋转



插入4后



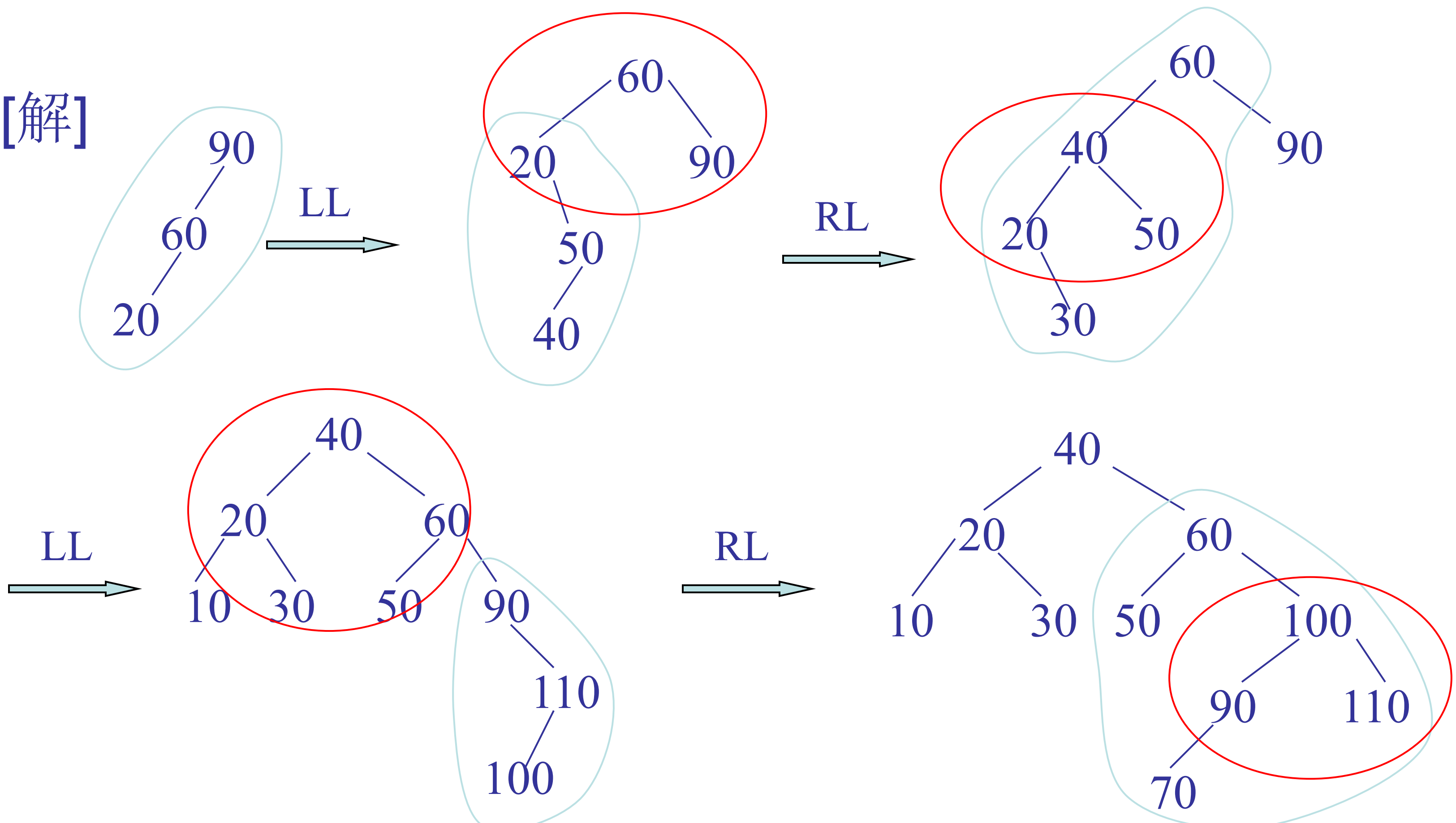
调整后

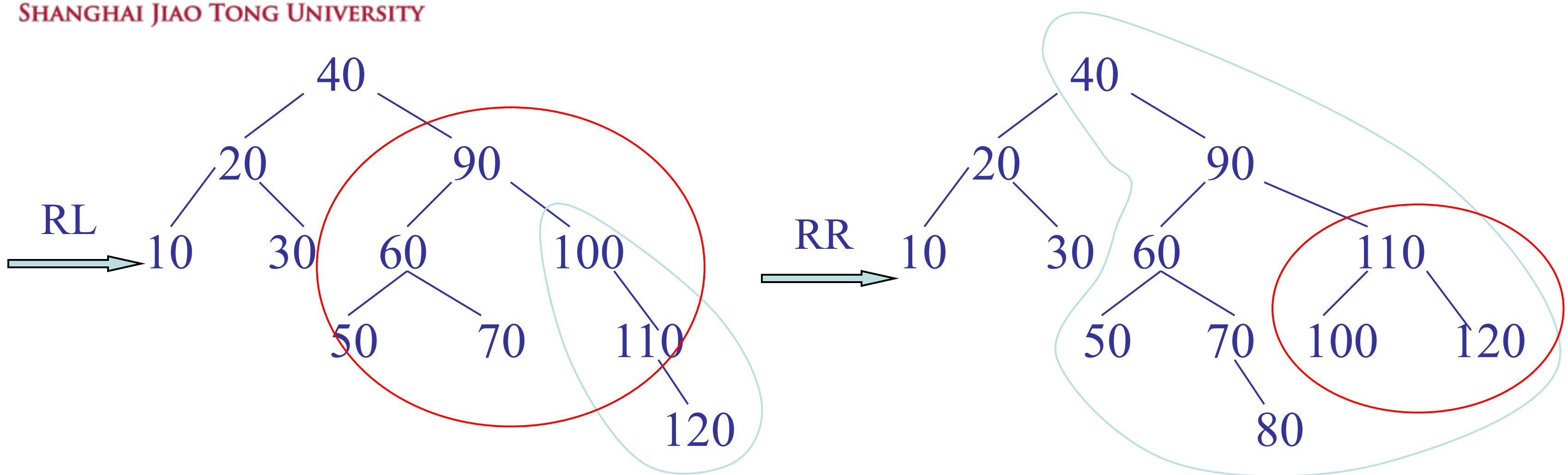


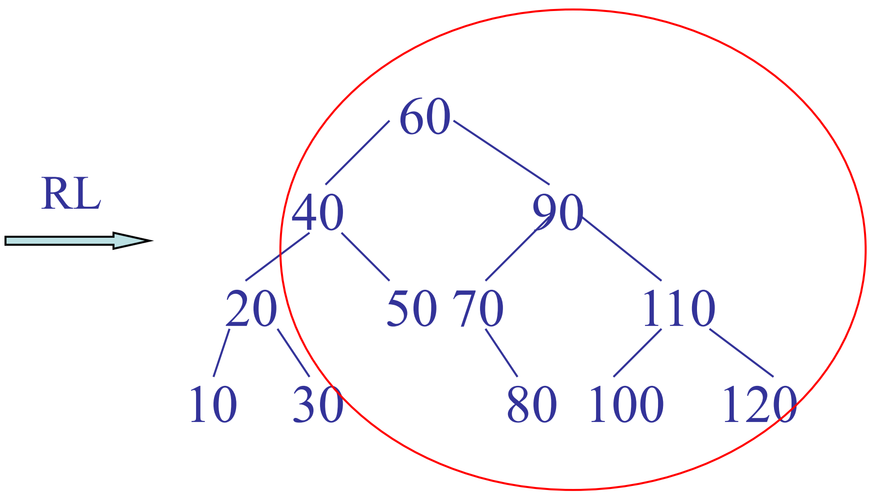
以下表顺序建立平衡二叉排序树，并求在等概率情况下查找成功的平均查找长度。

（**90,60,20,50,40,30,10,110,100,70,120,80**）

[解]







$$ASL=(1\times1+2\times2+4\times3+5\times4)/12$$

$$=37/12$$

AVL树

- AVL树的定义
- AVL树的存储实现
- AVL树的查找
- AVL树的插入
- AVL树的删除

AVL树的删除

- 首先在AVL树上删除结点x
- 然后调整平衡

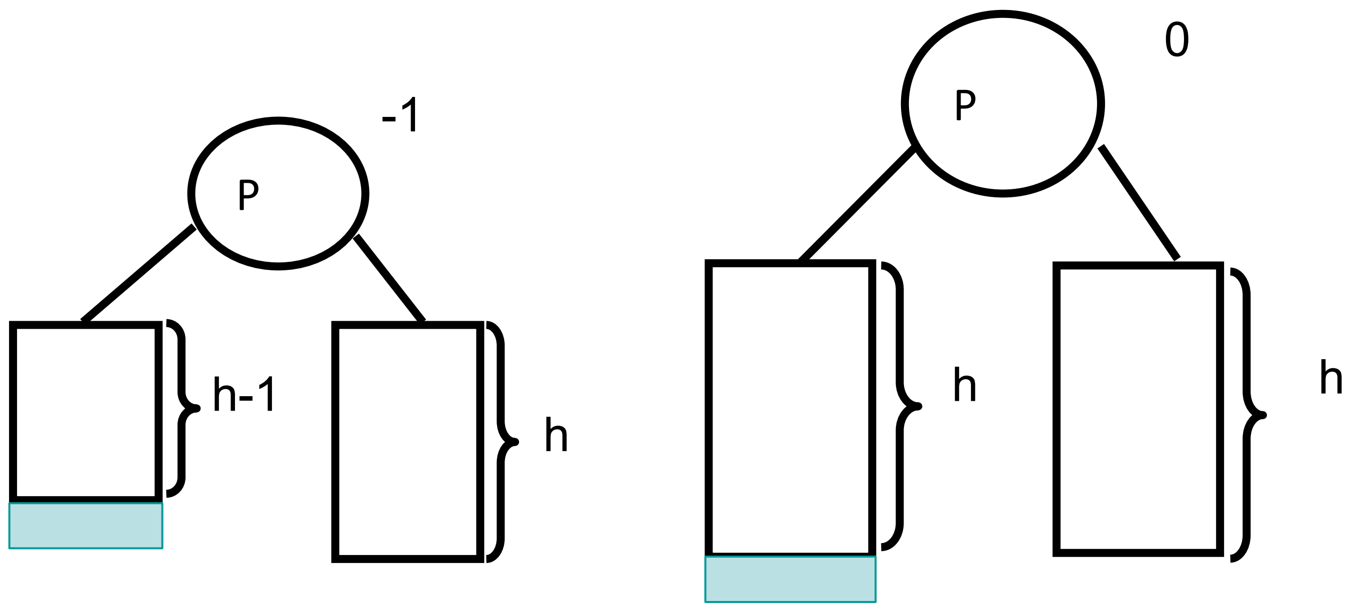
上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

调整平衡

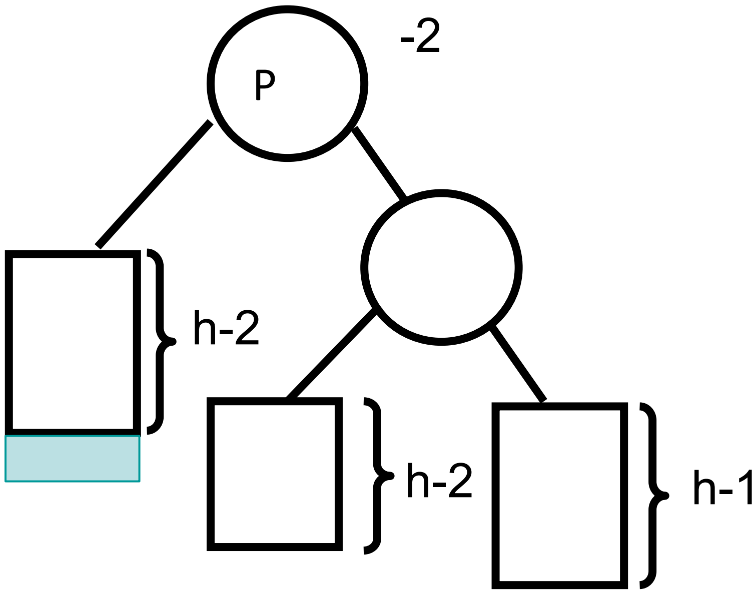
- 与插入操作一样，失衡节点存在于被删节点到根节点的路径上
- 在删除了一个结点后，必须沿着到根结点的路径向上回溯，随时调整路径上的结点的平衡度。
- 删除操作没有插入操作那么幸运。插入时，最多只需要调整一个结点。而删除时，我们无法保证子树在平衡调整后的高度不变。只有当某个结点的高度在删除前后保持不变，才无需继续调整。
- 递归的删除函数有一个布尔型的返回值。当返回值为true时，调整停止。当返回值为false时，继续调整。

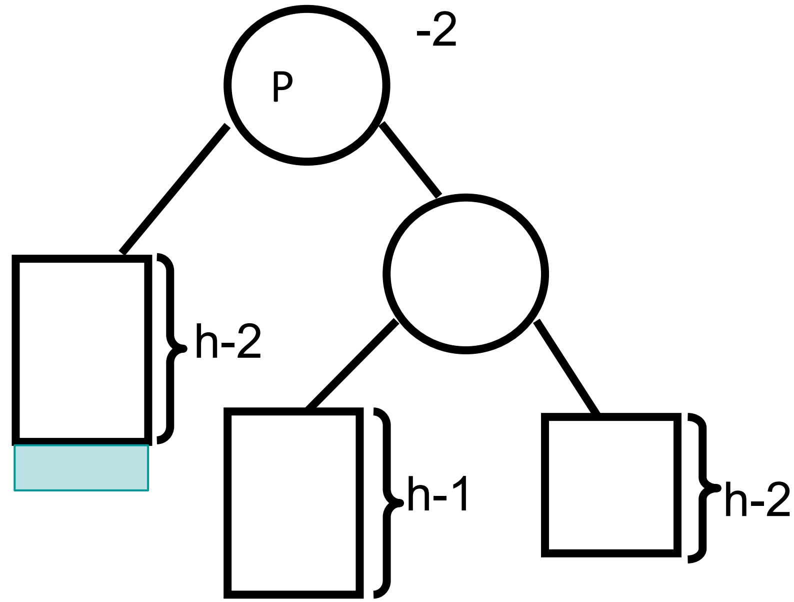
删除可能出现的情况

假设删除发生在左子树，导致左子树矮了一层



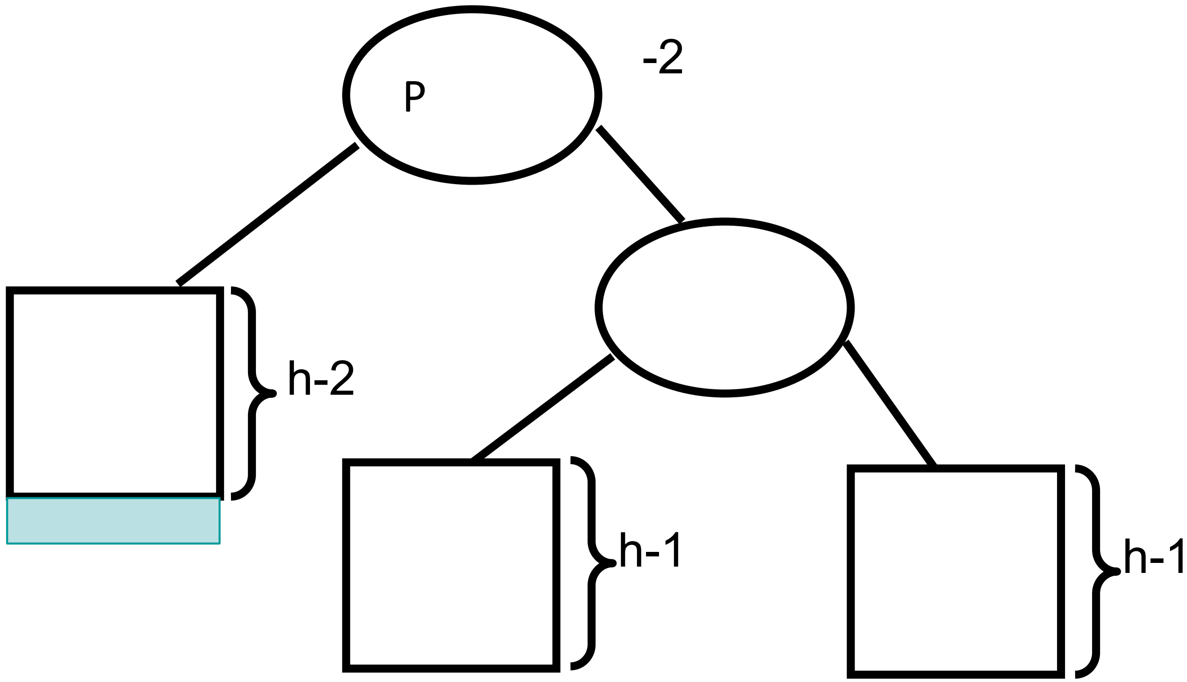
没有失衡，高度没变　　　没有失衡，高度变矮





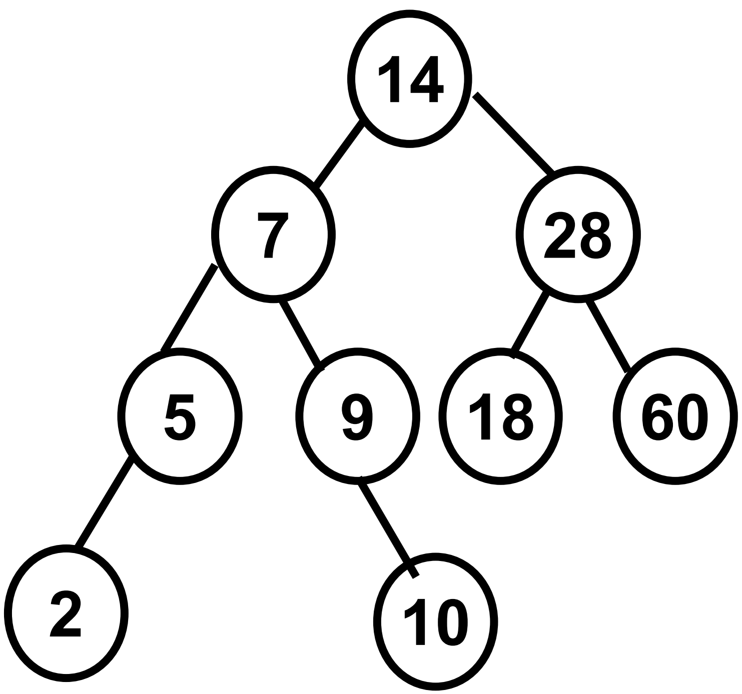
失衡， RR旋转，高度变矮

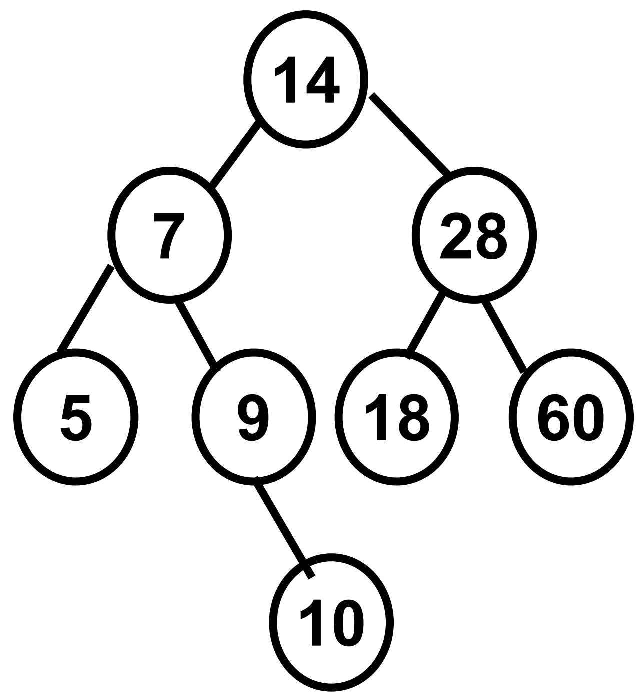
失衡， RL旋转，高度变矮



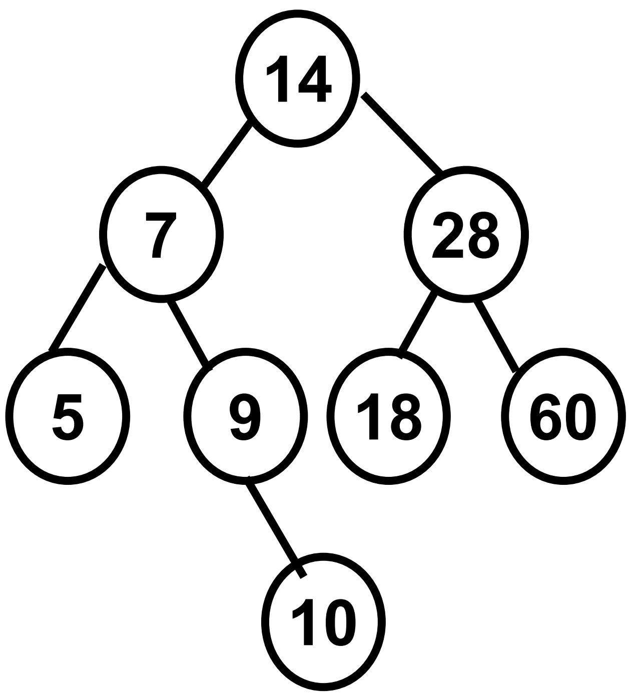
失衡， RR或RL旋转都可以，高度没变

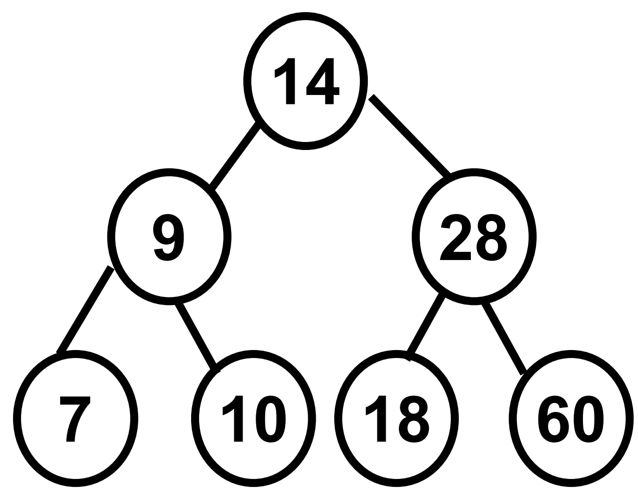
删除2：原来结点5是左高右低，属于情况2。删除2以后，结点5的平衡因子变为0，以结点5为根结点的子树也矮了一层，这样就会影响结点7的平衡度，所以继续往上调整。对结点7而言，正好属于情况一。所以修改7的平衡因子，调整结束。





删除5：结点7属于情况3。对7执行RR旋转。7平衡了，但这棵子树矮了一层，所以继续往上调整。对结点14而言，正好属于情况2。本身是平衡的，但子树矮了一层，需要继续往上调整。但由于14是根结点，所以调整结束





删除总结

- 结点删除同二叉查找树。在删除了叶结点或只有一个孩子的结点后，子树变矮，返回false
- 每次递归调用后，检查返回值。如果是true，直接返回true。否则分5种情况进行处理

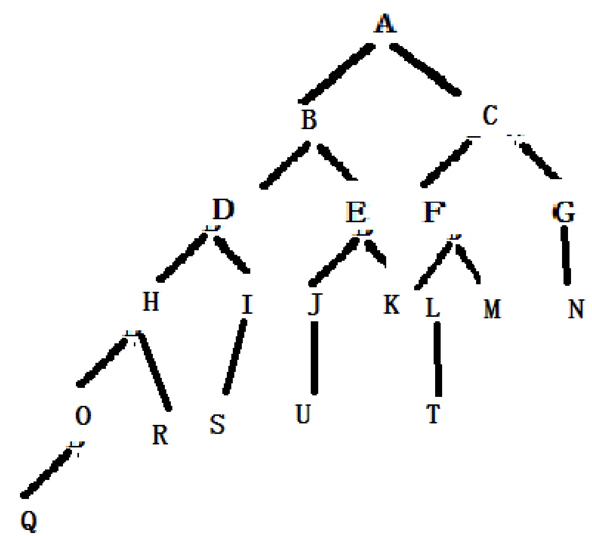
随堂练习

1. （AVL树）在一个空的AVL树内，依次插入关键字为10，20，30，40，50，60的结点，画出所有结点都插入完后的AVL树

2. (AVL树)若平衡二叉树的高度为6，且所有非叶子结点的平衡因子均为1，则该平衡二叉树的结点总数为（B）

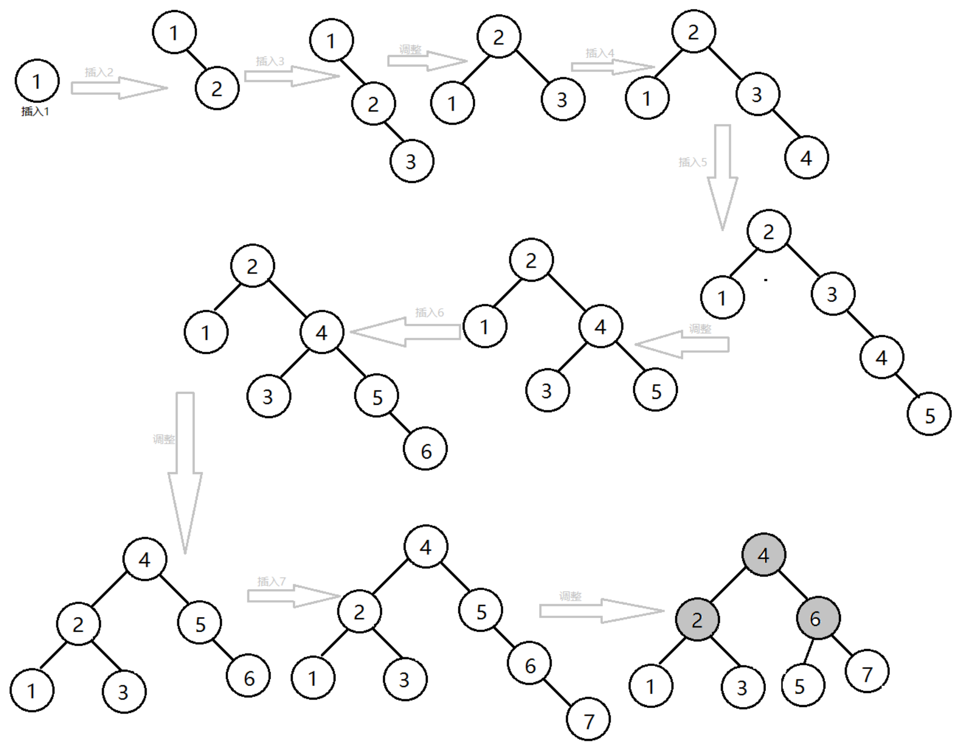
A. 12　　B. 20　　C. 32
D. 33

斐波那契数列，平衡二叉树的最小结点数：f(n)=f(n-1)+f(n-2)+1
其中 f(1)=1, f(0)=0。括号中的数代表深度。
从下往上数：
f(2)=f(1)+f(0)+1=2；
f(3)=f(2)+f(1)+1=4; f(4)=7, f(5)=12,f(6)=20



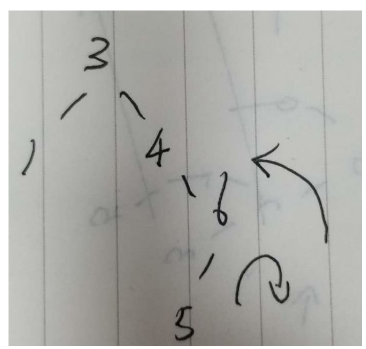
3. (AVL树)若将关键字1, 2, 3,4,5,6,7依次插入到初始为空的平衡二叉树 T 中，则 T 中平衡因子为 0 的分支结点的个数是（D）

A. 0　　B. 1　　C. 2　　D. 3



4. (AVL树)将一系列数字顺序一个个插入一棵初始为空的AVL树。下面哪个系列的第一次旋转是“RL”双旋：（B）

A. 6，5，4，3，2，1
B. 3，1，4，6，5，2
C. 4，2，5，6，3，1
D. 1，2，3，4，5，6



第9章 动态查找表

9.6 散列表

之前各种查找表的结构特点：

- 记录在表中的位置和它的关键字之间不存在一个确定的关系
- 查找的过程为给定值依次和集合中各个关键字进行比较，查找的效率取决于和给定值进行比较的关键字个数。
- 不同的表示方法，其差别仅在于：关键字和给定值进行比较的顺序不同。

9.6 散列表

常用的哈希函数

- 直接地址法
- 除留取余法
- 数字分析法
- 平方取中法
- 折叠法

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

除留余数法

- H(key)= key MOD M
- 最常用，余数总在 0 ~ M-1 之间。
- 经验表明选取M为素数时，散列函数值分布比较均匀

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

冲突问题

- 要选择一个一一对应的哈希函数很困难。一般的哈希函数都是多对一。当两个以上的关键字映射到一个存储单元时，称为冲突或碰撞。
- 冲突解决的方法：
  - 闭散列表：利用本散列表中的空余单元
    - 线性探测法
    - 二次探测法
    - 再次散列法
  - 开散列表：将碰撞的结点存放在散列表外的各自的线性表中（链接法）

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

线性探测法

- 当散列发生冲突时，探测下一个单元，直到发现一个空单元
- 在一个规模为11的散列表中依次插入关键字17、12，23，60、29、38，采用的散列函数为H(key) = key MOD 11。

0	1	2	3	4	5	6	7	8	9	10
	12	23			60	17	29	38		

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

删除和查找

- 查找:

  计算 addr = H(key)

  while (addr的内容非空&& 不等于要查找的键)

  ++addr

  if (addr内容为空) 没有找到 else 找到

- 删除：一般来讲，删除某一元素，先要找到该元素，然后把该空间的内容清空。但这样就给查找带来了问题，某些元素会查不到。解决的方案是采用迟删除，即不真正删除元素，而是做一个删除标记。

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

闭散列表类的实现

- 闭散列表是用一个数组实现，数组的大小是由用户定义散列表时指定
- 由于闭散列表中的删除是用迟删除的方法实现的，为此每个数组元素除了要保存对应的数据元素之外还必须保存一个数组元素的状态。
- 必须定义数组元素的类型。

二次探测法

0	1	2	3	4	5	6	7	8	9	10
	12	23			60	17	29	38		

- 二次探测法：地址序列为

$$k+1^2, k+2^2, k+3^2 \ldots$$

- 问题：
  - 是否能保证每次探测到的是新单元？是否能保证当表没有满时一定能插入？
  - 地址的计算量是否太大？

定理：

如果采用二次探测法，并且表的大小是一个素数，那么，如果表至少有一半是空的，新的元素总能被插入。而且，在插入过程中，没有一个单元被探测两次。

证明：

设M是表的大小。假设M是一个大于3的奇素数。我们说明前$[M/2]$个替换单元（包括初始单元）是不同的。这些单元中的某两个单元是$H+i^2 \pmod{M}$和$H+j^2 \pmod{M}$，其中，用反证法，

$0 \le i,j \le [M/2]$假设这两个单元是相同的，但$i \neq j$，那么

$$H+i^2 \equiv H+j^2 \pmod{M}$$

$$i^2 \equiv j^2 \pmod{M}$$

$$i^2 - j^2 \equiv 0 \pmod{M}$$

$$(i-j)(i+j) \equiv 0 \pmod{M}$$

因为M是素数，所以$i-j$或$i+j$是可以被M整除的。因为i和j是不同的，并且它们的和小于M，因此，这些可能性都不可能出现。这样我们得到了一个矛盾。它遵从前$[M/2]$个替换单元（包括初始单元）是不同的，并且保证如果表至少有一半是空的，新的元素总能被插入。

地址的计算量问题

- 定理：不需要昂贵的乘法和除法就能实现二次探测法。

证明:

设H_{i-1}是最近计算到的探测点（H_0是原始的散列位置），H_i是要计算的新的探测点，那么，可以有

$$H_i = H_0 + i^2 (\mathrm{mod}\ M)$$

$$H_{i-1} = H_0 + (i - 1)^2 (\mathrm{mod}\ M)$$

如果把这两个公式相减，可以得到

$$H_i = H_{i-1} + 2i - 1 (\mathrm{mod}\ M)$$

乘2是1次移位，取模可以用一个减法！

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

再次散列法

- 再散列法中有两个散列表函数H1和H2。H1用来计算探测序列的起始地址，H2用来计算下一个探测位置的步长。
- 设表长为M，插入的元素为x。再散列法的探测序列为：H1(x),(H1(x)+H2(x))mod M,), (H1(x)+2*H2(x))mod M
- H2的选择是非常重要的。例如，它将永远不会计算出0，必须保证所有单元都能被探测到。

例如：关键字集合 { 19, 01, 23, 14, 55, 68, 11, 82, 36 }

设定哈希函数 *H(key) = key MOD 11* (表长=11)

若采用**线性探测再散列**处理冲突

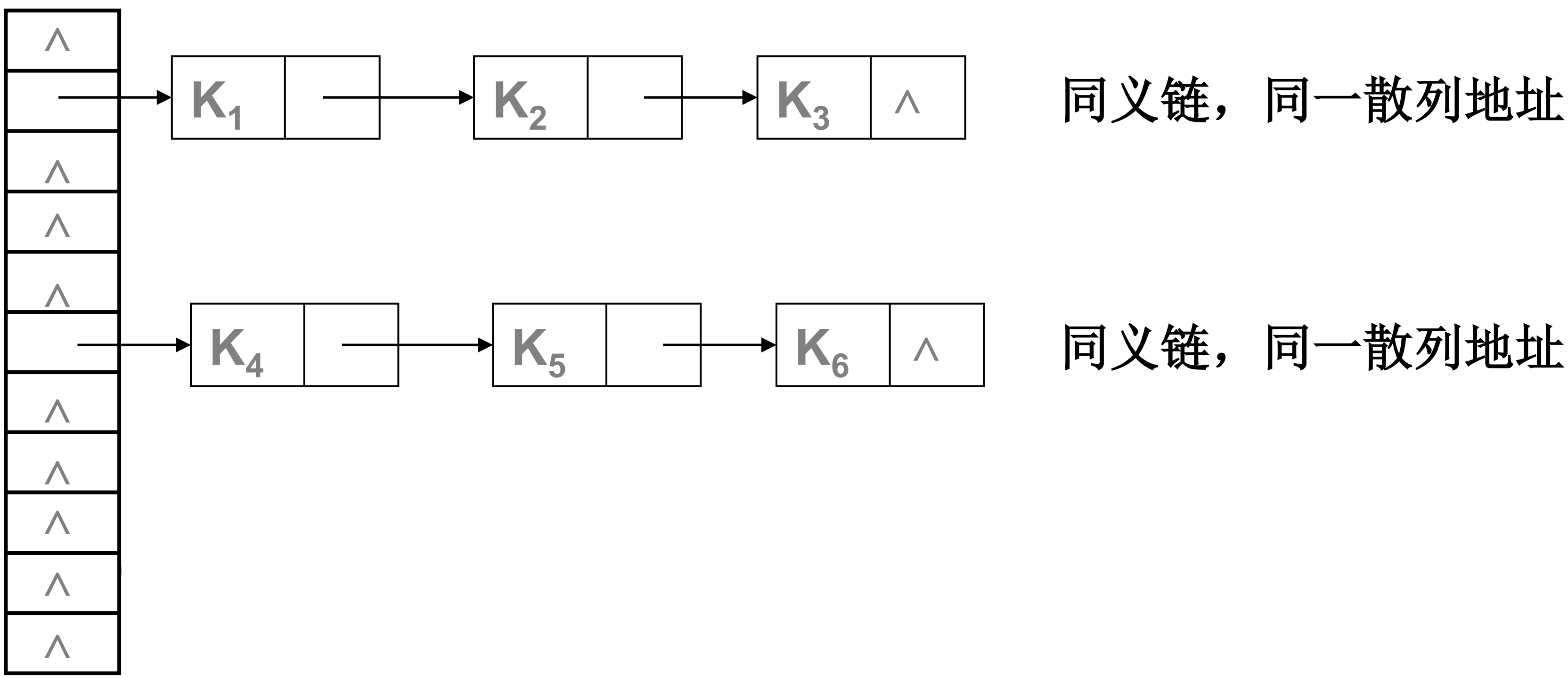
0	1	2	3	4	5	6	7	8	9	10
55	**01**	**23**	**14**	**68**	**11**	**82**	**36**	**19**		
1	1	2	1	3	6	2	5	1		

若采用**二次探测再散列**处理冲突

0	1	2	3	4	5	6	7	8	9	10
55	**01**	**23**	**14**	**11**	**82**	**68**	**36**	**19**		

开散列表

将具有同一散列地址的结点保存于 M 存区的各自的链表之中。书上介绍的是外链法，通常用于组织存在于外存设备上的数据文件。



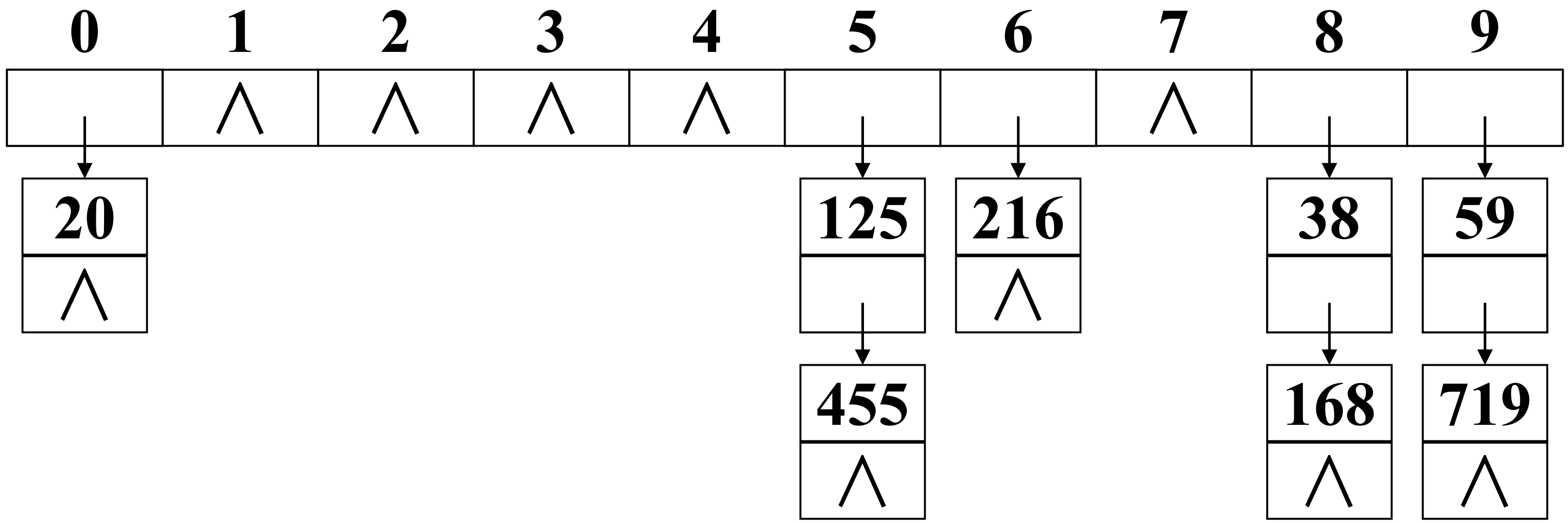
[例] 有一组关键字序列(38、59、125、168、216、719、455、20)，用函数 H(key) = key MOD 10 将其按顺序散列到哈希表 HT(0:9) 中，分别用两种方法解决冲突：线性探测法、链地址法(开散列表)，画出这两种方法建立的散列表。

1)线性再探测法

0	1	2	3	4	5	6	7	8	9
168	719	20			125	216	455	38	59
719 20	20				455	455		168	168 719

SHANGHAI JIAO TONG UNIVERSITY

2)链地址法



开散列表的实现

- 开散列表是将所有散列到同一地址的元素链成一个单链表，于是需要定义了一个单链表中的结点类。
- 采用不带头结点的单链表
- 散列表保存在一个数组中，数组的每个元素是一个指针，指向对应的单链表的首地址。
- 开散列表的行为除了构造函数和析构函数之外，还支持插入、删除和查找操作

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

总结

- 散列存储结构最“忠实”地表达了集合结构。
- 散列结构不是利用数据元素之间的关系，而是利用散列函数来查找元素，因此在理想的情况下可以在常量的时间内实现insert、remove和find操作。
- 本章讨论了散列表实现中需要解决的问题，包括散列函数的选择、碰撞的解决。在此基础上实现了开散列表类和闭散列表类。

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

1. 设某散列表的长度为100，散列函数H(k)=k mod P, 则 P 通常情况下最好选择______

A. 99　　B. 97　　C. 91
D. 93

2. （散列表）散列表长度为11，散列函数H(k) = k MOD 11, 若输入顺序为（18, 10, 21, 9, 6, 3, 16, 25, 7 ）的序列，处理冲突方法为线性探测法，请构造散列表

3 若采用链地址法构造散列表，Hash函数为H（key）=key Mod17，则需①中的（A）个链表。

这些链的链首指针构成一个指针数组，数组的下标范围为②中的（C）

① A.17 B.13 C.16 D.任意

② A.0~17 B.1～17 C.0~16 D.1~16

4 给定输入{4371，1323，6173，4199，4344，9679，1989}，散列表长度为11，散列函数为H(x)=X mod 11,写出下列结果：

（1）线性探测散列表

（2）二次探测散列表

（3）开散列表