第6章 二叉搜索树（字典）

1 字典的定义
2 用数组实现字典
3 用二叉搜索树实现字典
4 AVL树
5 字典的应用

学习要点:

- 理解以有序集为基础的抽象数据类型有序字典。
- 理解用数组实现有序字典的方法。
- 理解二叉搜索树的概念和实现方法。
- 掌握用二叉搜索树实现有序字典的方法。
- 理解AVL树的定义和性质。
- 掌握二叉搜索树的结点旋转变换及实现方法。
- 掌握AVL树的插入重新平衡运算及实现方法。
- 掌握AVL树的删除重新平衡运算及实现方法。

返回章目录

1 字典的定义

有序字典是以有序集为基础的抽象数据类型。

它支持以下运算：

(1)Member(x)，成员运算。

(2)Insert(x)，插入运算：将元素x插入集合。

(3)Delete(x)，删除运算：将元素x从当前集合中删去。

(4)Predecessor(x)，前驱运算：返回集合中小于x的最大元素。

(5)Successor(x)，后继运算：返回集合中大于x的最小元素。

(6)Range(x，y)，区间查询运算：返回集合中界于x和y之间，即x≤z≤y的所有元素z组成的集合。

(7)Min( )，最小元运算：返回当前集合中依线性序最小的元素。

返回章目录

2 用数组实现字典

用数组实现字典与用数组实现符号表的不同之处：

可以利用线性序将字典中的元素从小到大依序存储在数组中，通过数组下标来反映字典元素之间的序关系。

- 优点：
  - 可用二分法高效地实现与线性序有关的一些运算。如：**Member(x)**，**Predecessor(x)**和 **Successor(x)**可在时间O(logn)内实现。
- 缺点：
  - 插入和删除运算的效率较低。
  - 每执行一次**Insert**或**Delete**运算，需要移动部分数组元素，从而导致它们在最坏情况下的计算时间为**O(n)**。
- 考虑：能否用链表来实现字典？？？
  - **Member**运算需要**O(n)**时间，一旦找到元素在链表中插入或删除的位置后，只要用O(1)时间就可完成插入或删除操作。

➔两种实现方式均不可取！

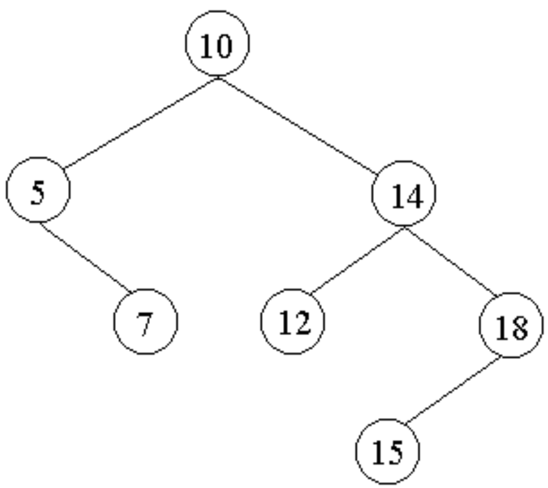
返回章目录

3 用二叉搜索树实现字典

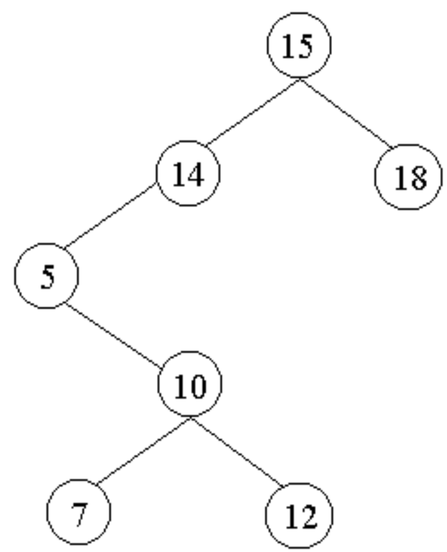
3.1 基本思想：

用二叉树来存储有序集，每一个结点存储一个元素。

满足：存储于每个结点中的元素x大于其左子树中任一结点中所存储的元素，小于其右子树中任一结点中所存储的元素。



(a)



(b)

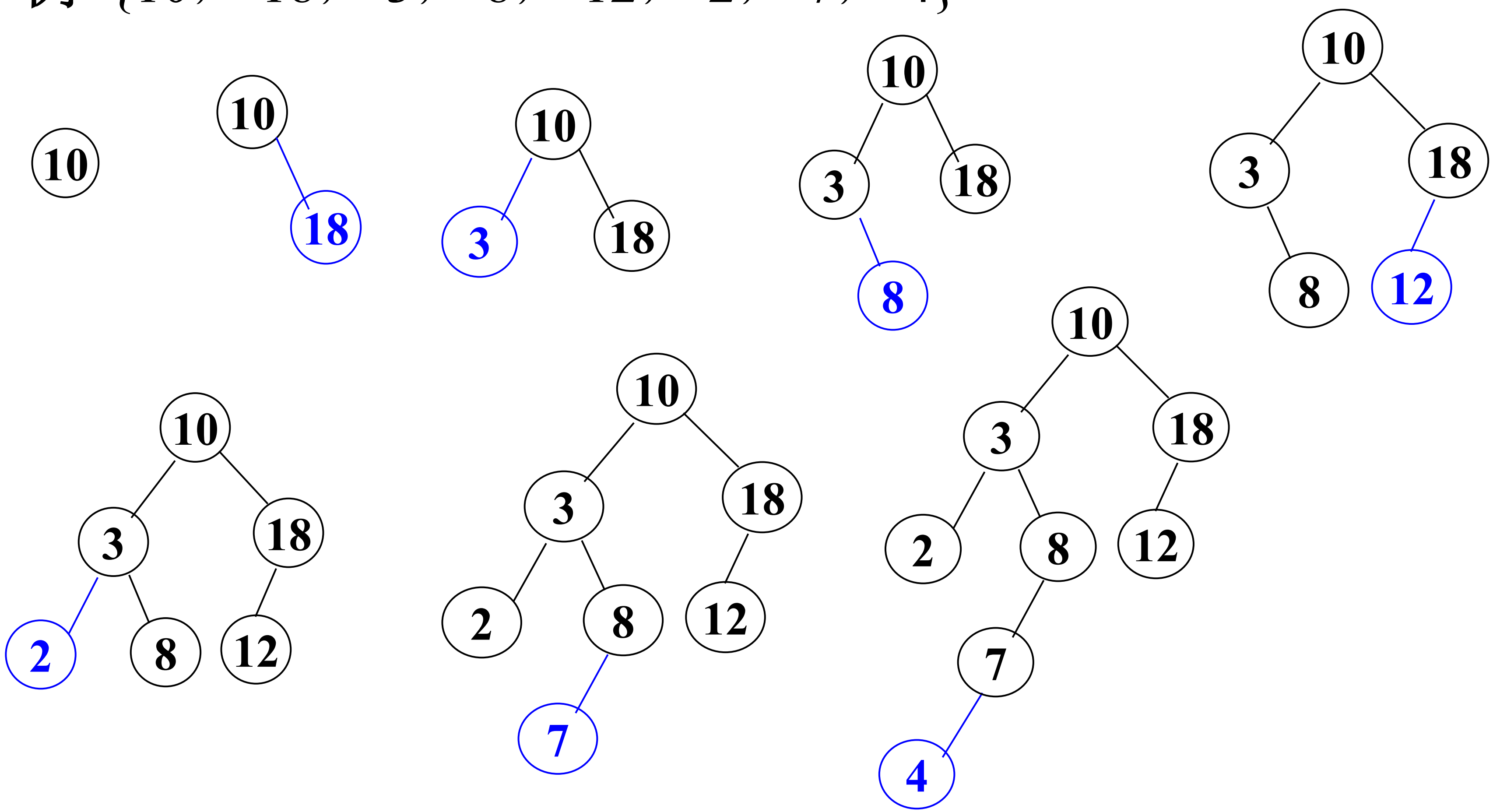
3.2 在二叉搜索树T表示的字典中搜索元素x的运算实现

```
btlink BSSearch (TreeItem x, BinaryTree T)
{
    btlink p = T->root;
      while (p)
          if (x < p->element) p = p->LeftChild;
          else if (x > p->element) p = p->RightChild;
          else break;
   return p;
}
```

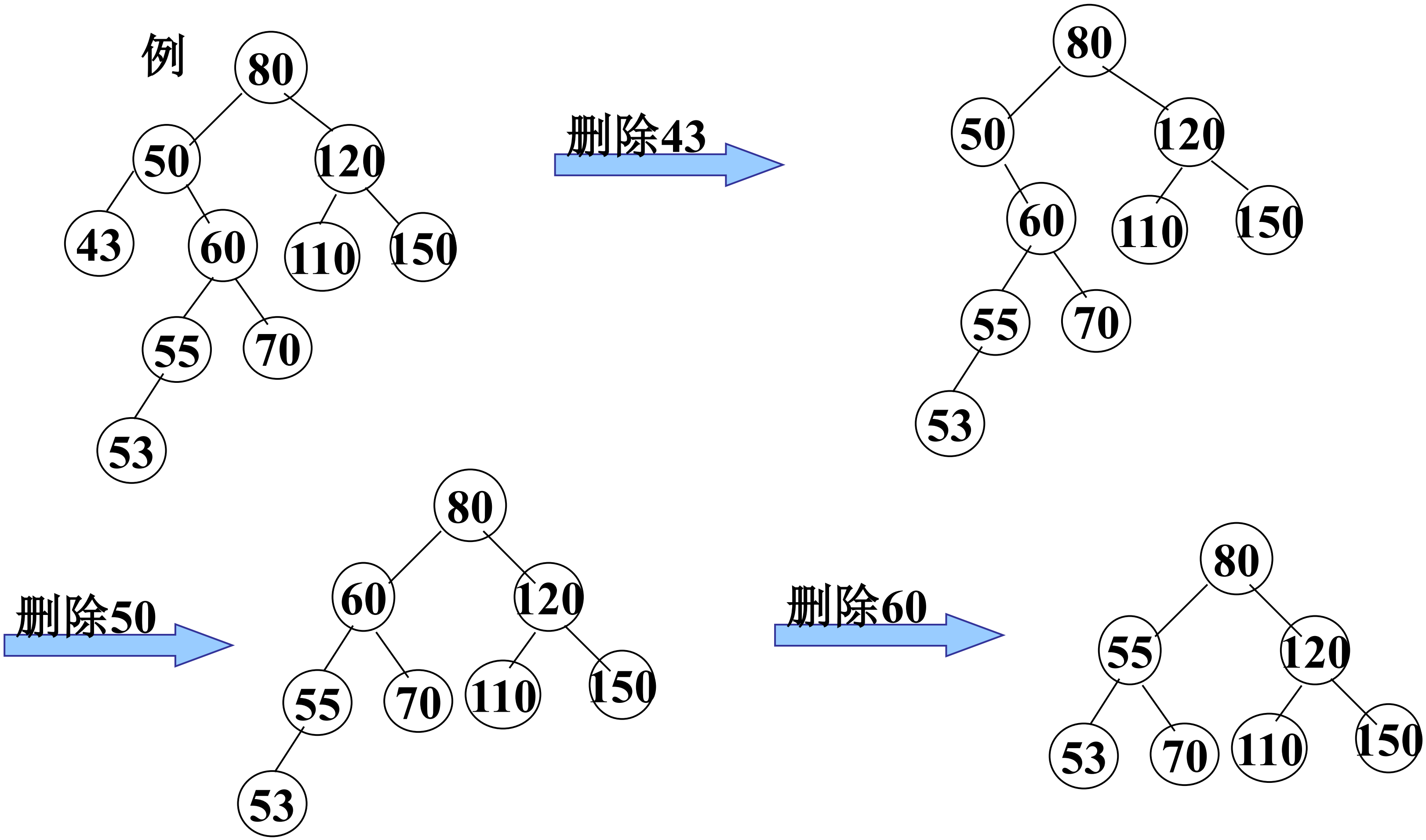
成员查询函数**Member(x,T)**只要返回**BSSearch(x, T)**搜索的结果。

3.3 二叉搜索树的建立过程—插入运算的实现

例 {10， 18， 3， 8， 12， 2， 7， 4}



3.4 在二叉搜索树T表示的字典中删除元素x的运算实现



3.4 在二叉搜索树T表示的字典中删除元素x的运算实现(续)

设要删除二叉搜索树中的结点p ，分三种情况：

- p为叶结点 ➔直接删除节点p
- p只有左子树或右子树
  - p只有左子树 ➔ 用p的左儿子代替p
  - p只有右子树 ➔ 用p的右儿子代替p
- p左、右子树均非空
  - ➔找p的左子树的最大元素结点（即**p**的前驱结点），用该结点代替结点**p**，然后删除该结点。

用二叉搜索树实现有序字典时间复杂性分析

- 最坏情况分析—member,insert,delete都需要O(n)
- 平均情况分析

引入记号:

记：p(n)为含有n个结点的二叉搜索树的平均查找长度。

显然p(0)=0,p(1)=1;

若设某二叉搜索树的左子树有i个结点，则：

p(i)+1为查找左子树中每个结点的平均查找长度;

p(n-i-1)+1为查找右子树中每个结点的平均查找长度;

➔由此构造而得的二叉搜索树在n个结点的查找概率相等的情况下，其平均查找长度为:

$$q(n,i)=\frac{1}{n}\big(1+i\big(p(i)+1\big)+\big(n-i-1\big)\big(1+p(n-i-1)\big)\big)$$

又假设当前的二叉搜索树有**n**个结点，而它是从空树开始反复调用**n**次的**Insert**运算得到的，且被插入的**n**个元素的所有可能的顺序是等概率的。则：

$$p(n)=\frac{1}{n}(\sum_{i=0}^{n-1}q(n,i))$$

$$=\frac{1}{n}(\sum_{i=0}^{n-1}\frac{1}{n}(1+i(p(i)+1)+(n-i-1)(p(n-i-1)+1)))$$

$$=1+\frac{1}{n^2}\sum_{i=0}^{n-1}\left(ip(i)+(n-i-1)p(n-i-1)\right)$$

$$=1+\frac{2}{n^2}\sum_{i=0}^{n-1}ip(i)$$

对**n**用数学归纳法可以证明：$p(n)\le 1+4\log n$

当**n=1**时显然成立。若设**i<n**时有 $p(i)\le 1+4\log i$ ，则

$$p(n) \le 1 + \frac{2}{n^2}\sum_{i=1}^{n-1} i(1 + 4\log i)$$

$$\le 1 + \frac{2}{n^2}\sum_{i=1}^{n-1} 4i\log i + \frac{2}{n^2}\sum_{i=1}^{n-1} i$$

$$\le 2 + \frac{8}{n^2}\sum_{i=1}^{n-1} i\log i$$

略去 -1/n 项

$$\le 2 + \frac{8}{n^2}\left(\sum_{i=1}^{n/2-1} i\log(n/2) + \sum_{i=n/2}^{n-1} i\log n\right)$$

$$\le 2 + \frac{8}{n^2}\left(\frac{n^2}{8}\log(n/2) + \frac{3n^2}{8}\log n\right)$$

$$= 2 + \frac{8}{n^2}\left(\frac{n^2}{2}\log n - \frac{n^2}{8}\right) \quad = 1 + 4\log n$$

➔平均情况下的时间复杂度为： $O(\log n)$

- 运算Predecessor(x)和Successor(x)的实现：
  ——类似于Search(x)算法
- 运算Range(y, z)的实现：可借助于Search(y)和Successor(y)运算
  - 首先，用Search(y)检测y是否在二叉搜索树中，是则输出y，否则不输出y；
  - 然后，从y开始，不断地用Successor找当前元素在二叉搜索树中的后继元素。当找出的后继元素x满足x ≤ z时，就输出x，并将x作为当前元素。重复这个过程，直到找出的当前元素的后继元素大于z，或二叉搜索树中已没有后继元素为止。
  - 时间复杂度：若二叉树搜索树中有r个元素x满足y ≤ x ≤ z，则在最坏情况下用 $O(rn)$ 时间，在平均情况下用 $O(r\log n)$ 时间可实现Range运算。

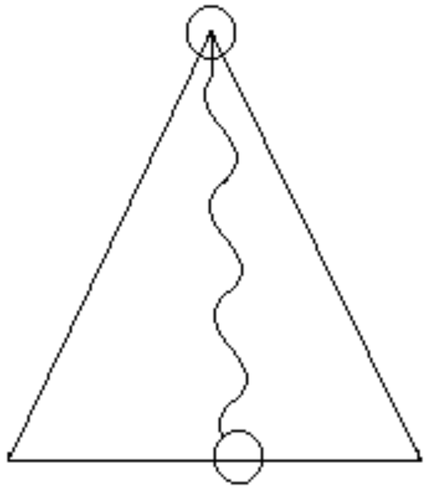
运算Range(y,z)的改进：

考虑半无限查询区域 $[y,+\infty)$ ，

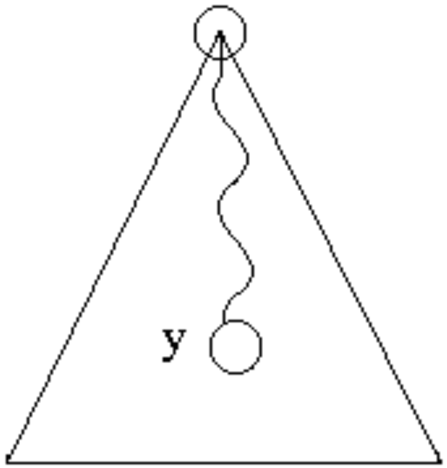
——即找出二叉搜索树中满足y ≤ x的所有元素x。

当y不在二叉搜索树中时，产生一条从根到叶的路径。如下图(a)

当y在二叉搜索树中时，产生一条从根到存储元素y的结点的路径。如下图(b)

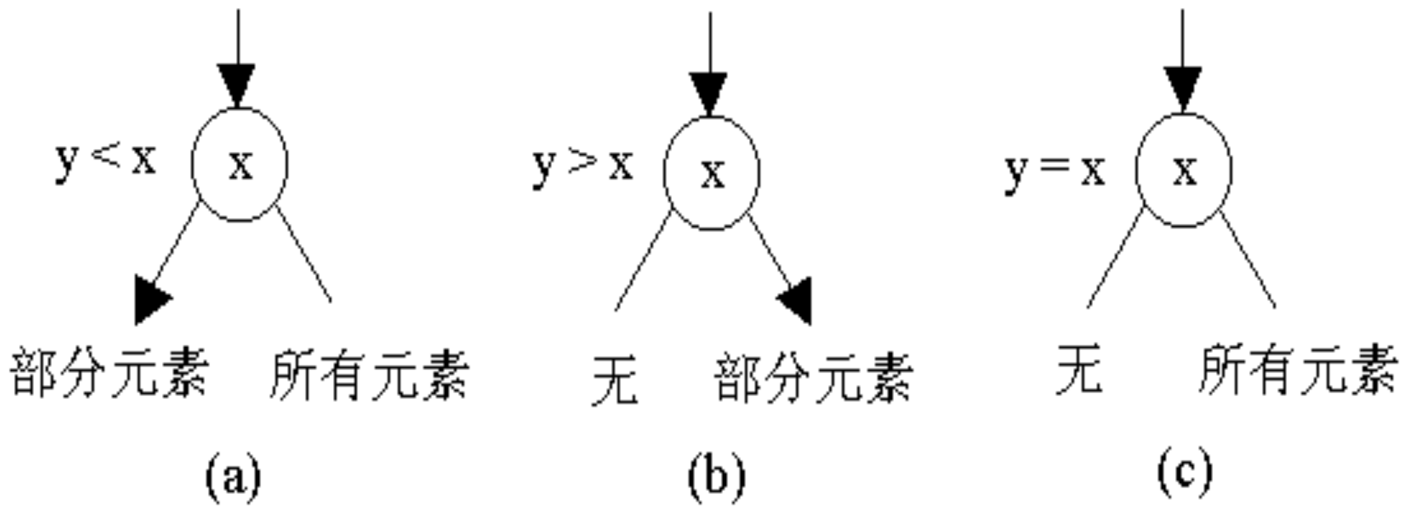


(a)



(b)

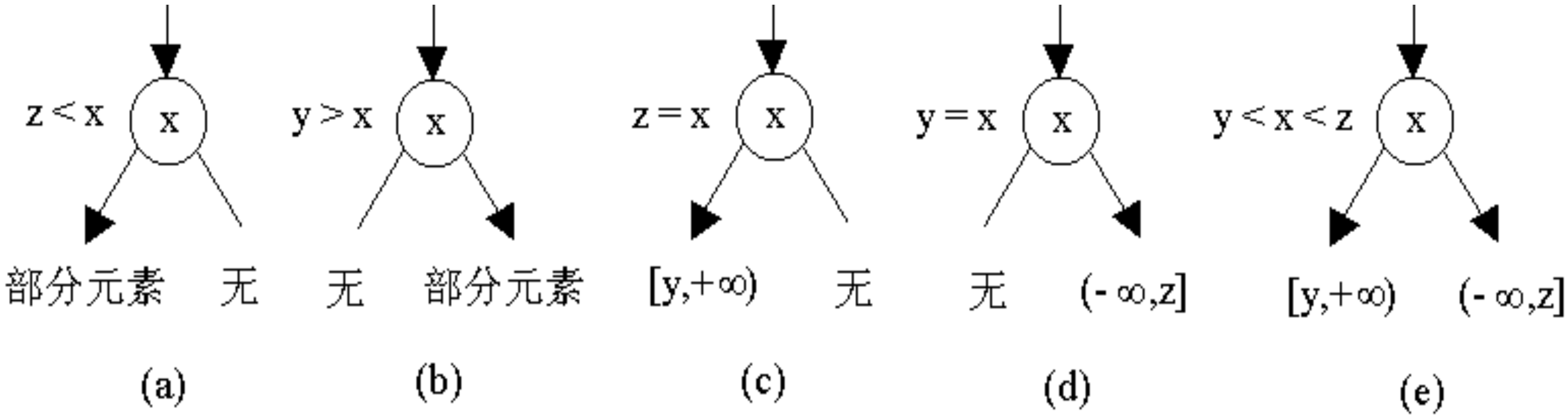
在找到的搜索路径上的所有结点可分为以下3种情况，如下图：



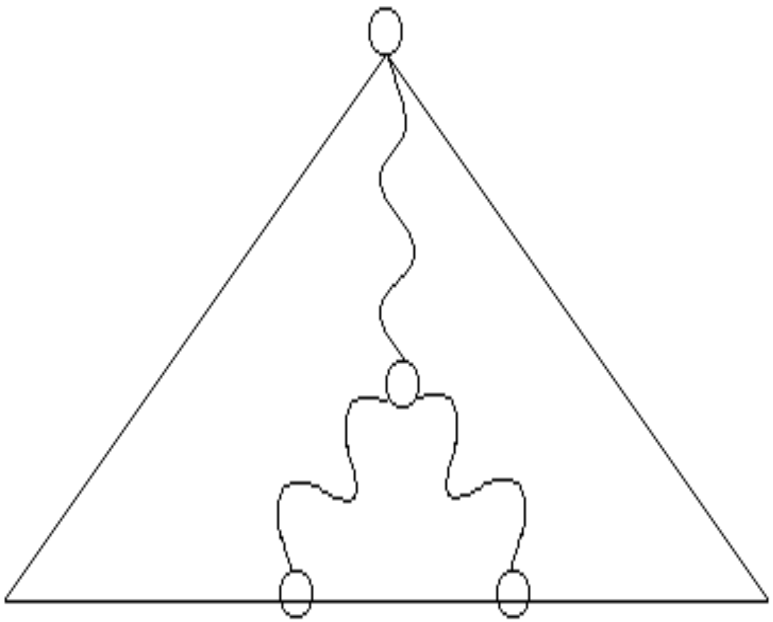
- 运算**Range(y,z)**的实现：

  ——可用类似于Range(y,∞)算法

从二叉搜索树的根结点开始，同时与y和z比较, 此时, 结点分类的情况可能有（见下图）：



- 运算Range(y,z)的搜索路径如下图：



返回章目录

4 AVL树

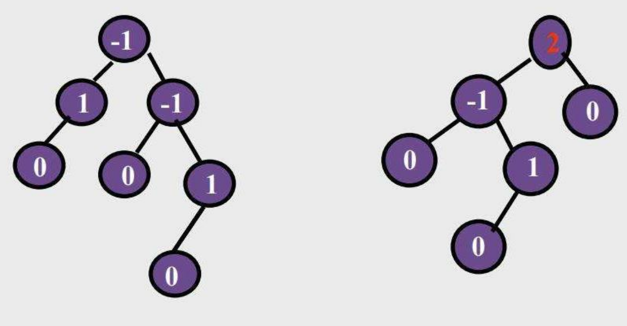
4.0 引言—AVL树产生的背景

- 问题的提出:用二叉搜索树实现有序字典在最坏情况下— member,insert,delete都需要O(n);平均情况下需要O(log n)。

问在最坏情况下能降到O(log n)吗?

前苏联科学家G.M. Adelson-Velskii 和 E.M. Landis在1962年发表的一篇名为《An algorithm for the organization of information》的文章中提出了一种自平衡二叉查找树（self-balancing binary search tree）。

4 AVL树



(a) 平衡树 (b) 不平衡树

4.0 引言—AVL树产生的背景(续)

- 解决问题的设想:
  - n个结点的二叉树最矮是近似满二叉树，其高为[log n]。若放宽此限制为每一个结点的左子树与右子树高度差的绝对值不超过1，则二叉树当然就达不到最矮，却可望接近最矮，而不超过O(log n)，目的就达到了。这正是AVL树。剩下的问题是设法找一种在insert和delete后只需O(log n)时间的维护算法。
- 设想的证实:
  - (1) n个结点的AVL树的高度为O(log n);
  - (2) insert和delete后的维护算法在最坏的情况下只需O(logn)的时间。

4 AVL树

4.1 AVL树的定义和性质

- 递归定义:
  - 空的和单结点的二叉搜索树都是AVL树;
    结点数大于1的二叉搜索树,若满足左子树和右子树都是AVL树且左、右子树高度之差的绝对值不超过1,那么,它是AVL树。
- 性质:

(1)AVL树T的结点数n与高度h的关系。

设高度h的AVL树的最少结点数N(h)。N(h)一定出现在树的左、右子树中一棵高为h-1,而另一棵高为h-2时。则N(h)满足如下递归方程:

$$N(h)=\begin{cases}1 & h=0\\ 2 & h=1\\ N(h-1)+N(h-2)+1 & h>1\end{cases}$$

4 AVL树

解上面的递归方程得：

$$N(h)=F(h+2)-1=\left(\left((1+\sqrt{5})/2\right)^{h+2}-\left((1-\sqrt{5})/2\right)^{h+2}\right)/\sqrt{5}-1$$

由于 $\left(\left((1+\sqrt{5})/2\right)^{h+2}-\left((1-\sqrt{5})/2\right)^{h+2}\right)/\sqrt{5}>\left(1+\sqrt{5}\right)^{h+2}/\sqrt{5}-1$

因此 $n\ge N(h)>\left((1+\sqrt{5})/2\right)^{h+2}/\sqrt{5}-2$

$$h+2<\log_{\frac{1+\sqrt{5}}{2}}(n+2)+\log_{\frac{1+\sqrt{5}}{2}}\sqrt{5}$$

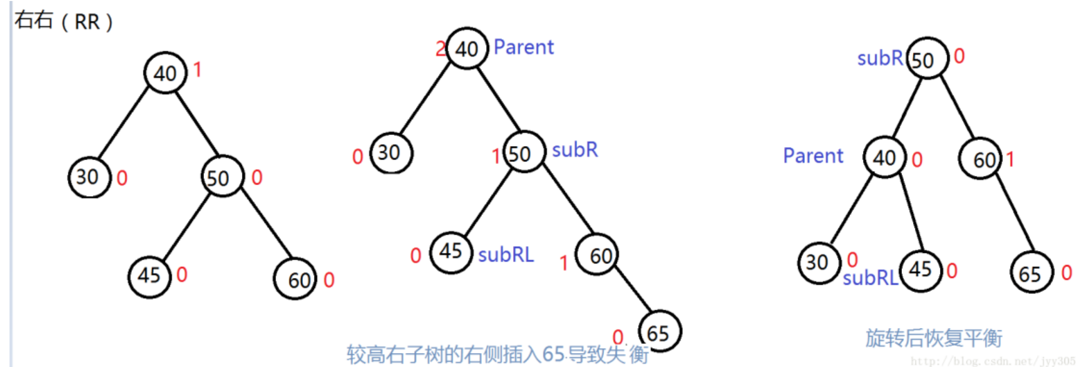
$$h<\log_{\frac{1+\sqrt{5}}{2}}(n+2)+\log_{\frac{1+\sqrt{5}}{2}}\sqrt{5}-2\le 1.4404\log(n+2)-0.328$$

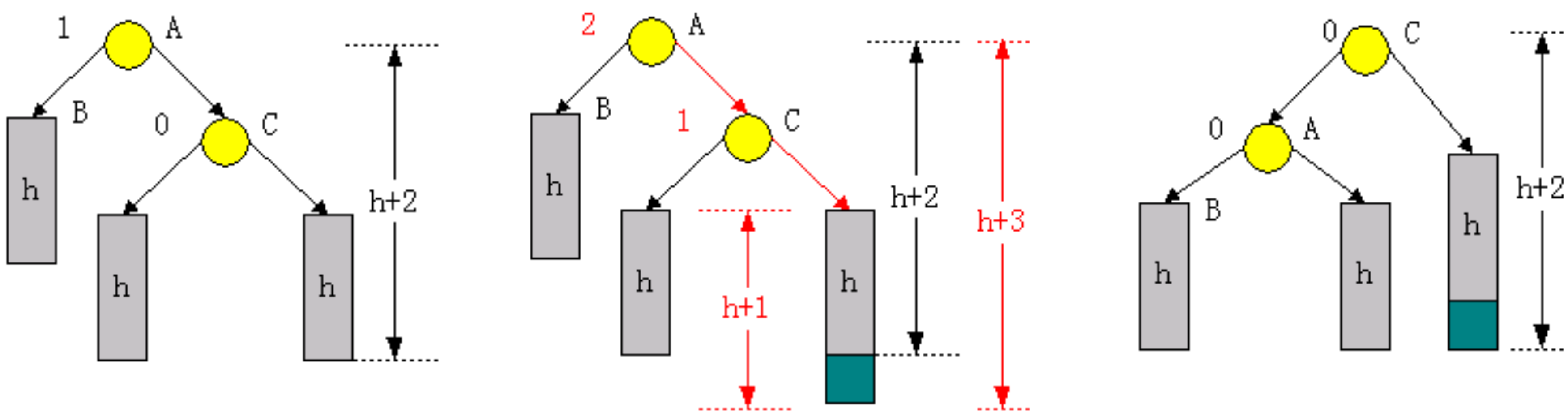
4 AVL树

4.2 旋转变换

- 旋转变换的目的：是调整结点的子树高度，并维持二叉搜索树性质，即结点中元素的中序性质。
- 旋转变换分为单旋转变换和双旋转变换2种类型。
- 单旋转变换又分为右单旋转变换和左单旋转变换。
- 双旋转变换又分为先左后右双旋转变换和先右后左双旋转变换。

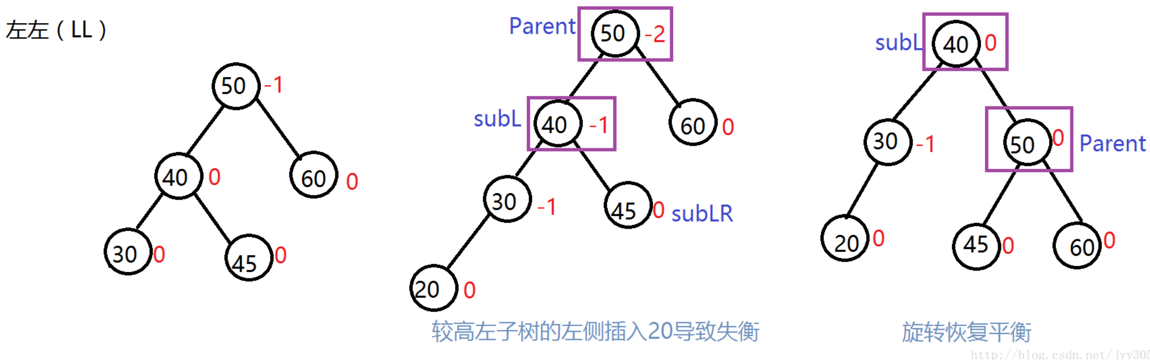
1、左单旋的情况

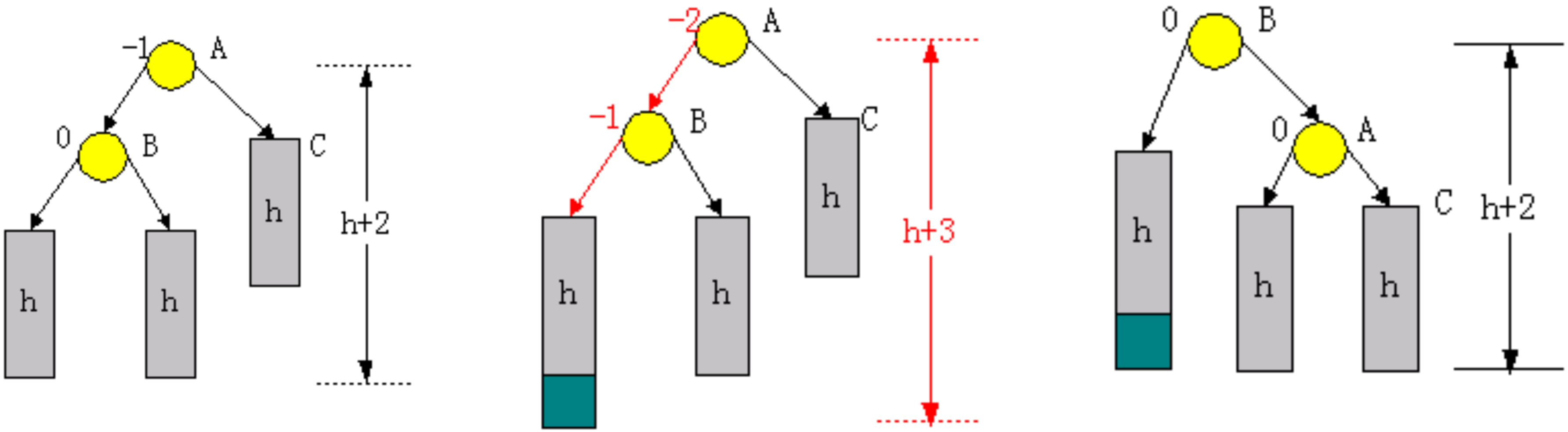




原来的AVL树　　插入一结点，A点不平衡　　左单旋的结果

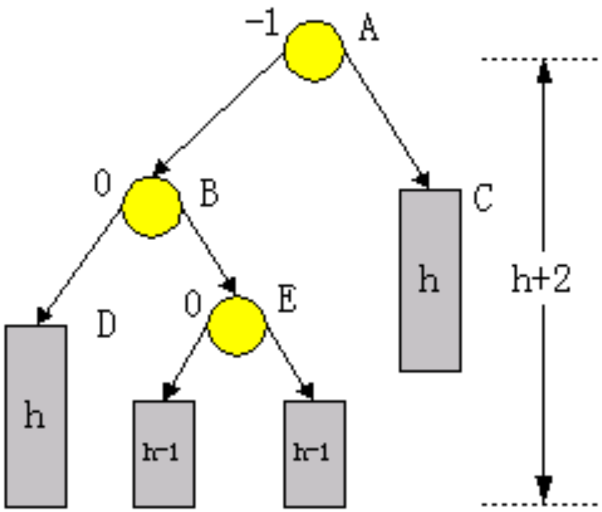
2.右单旋的情况



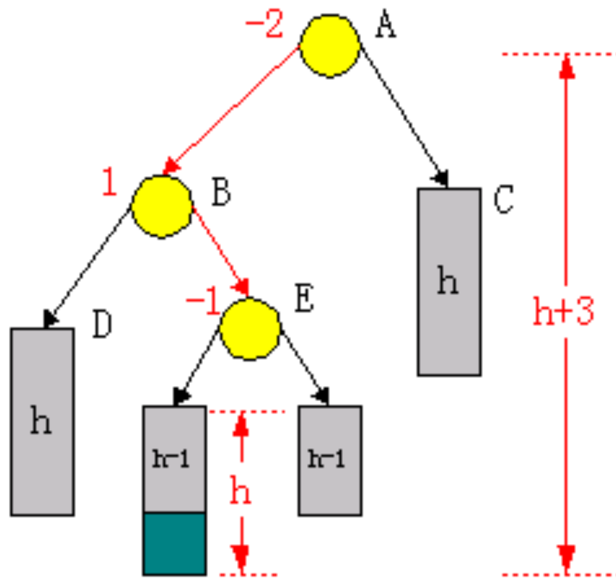


原来的AVL树　　插入一结点，A点不平衡　　右单旋的结果

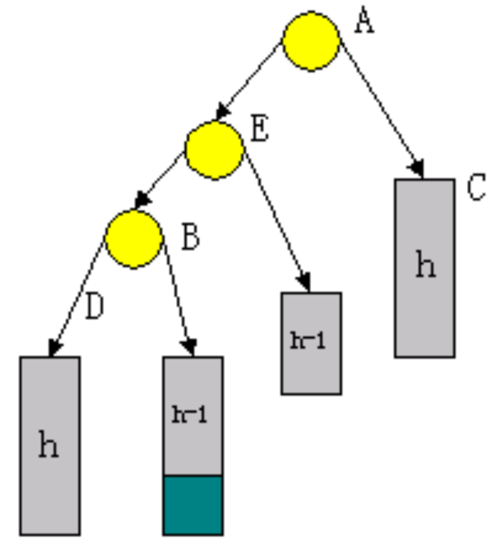
3.先左后右双旋的情况



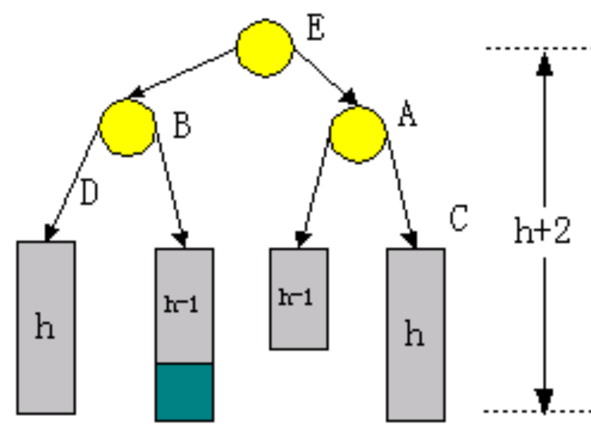
原来的AVL树



插入一结点，A点不平衡

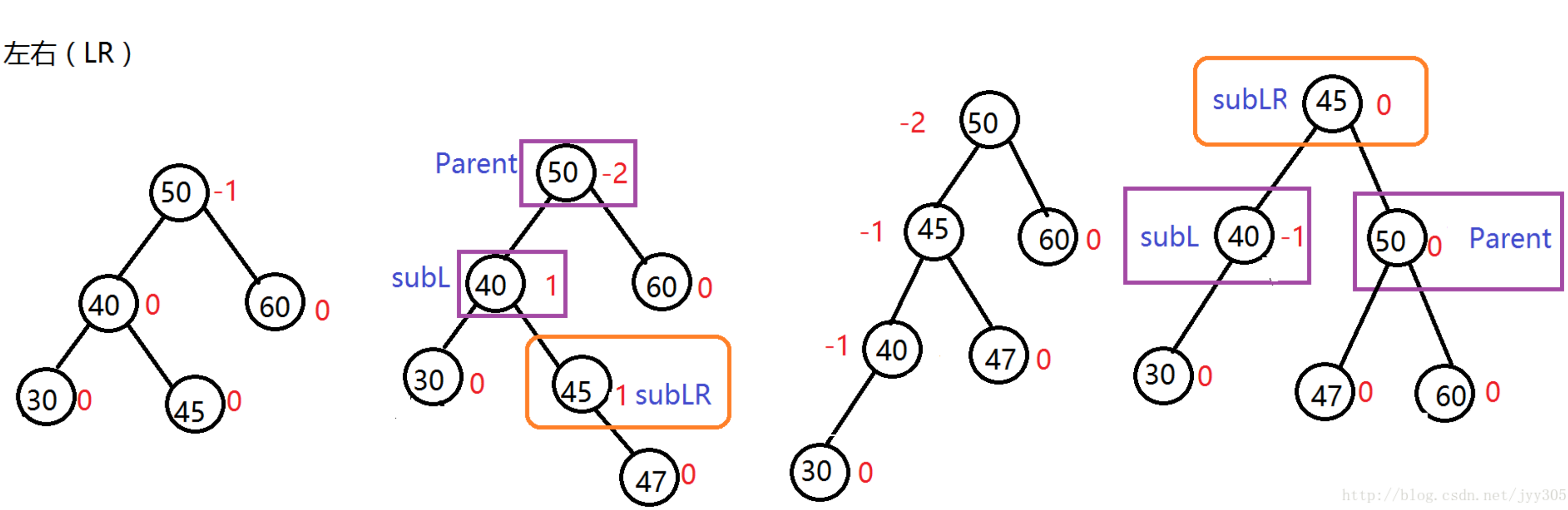


先左旋

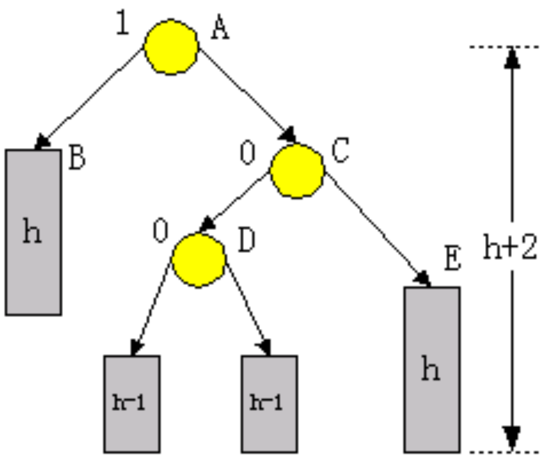


再右旋

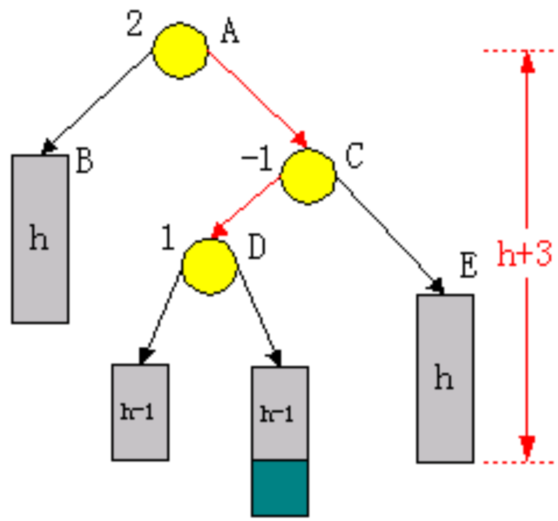
左右（LR）


50
-1
40
0
60
0
30
0
45
0
Parent
50
-2
subL
40
1
60
0
30
0
45
1 subLR
47
0
-2
50
-1
45
60
0
-1
40
47
0
30
0
subLR
45
0
subL
40
-1
50
0
Parent
30
0
47
0
60
0
http://blog.csdn.net/jyy305

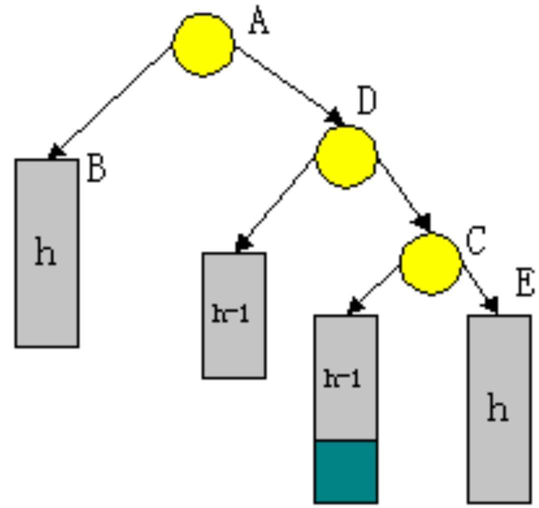
4.先右后左双旋的情况



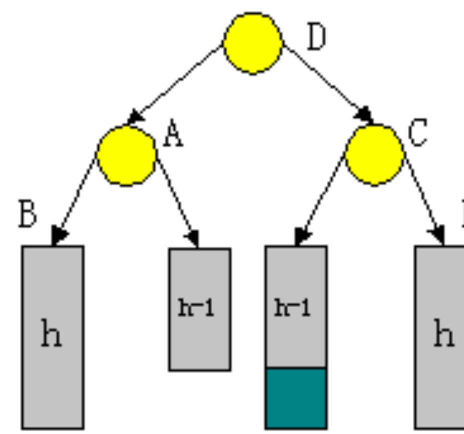
原来的AVL树



插入一结点，A点不平衡

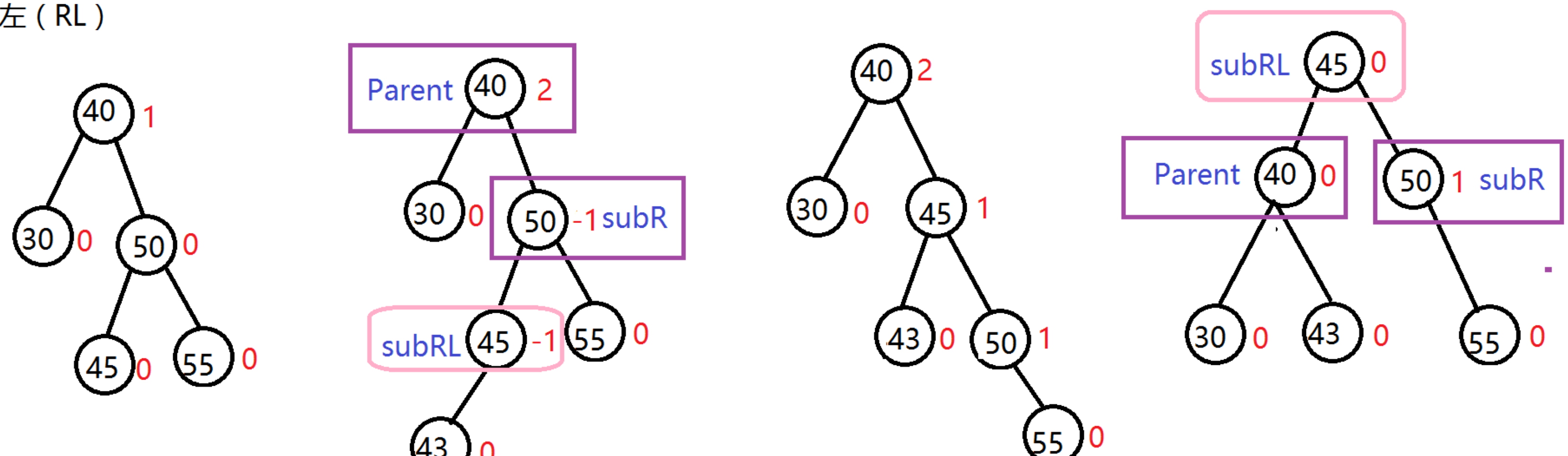


先右旋

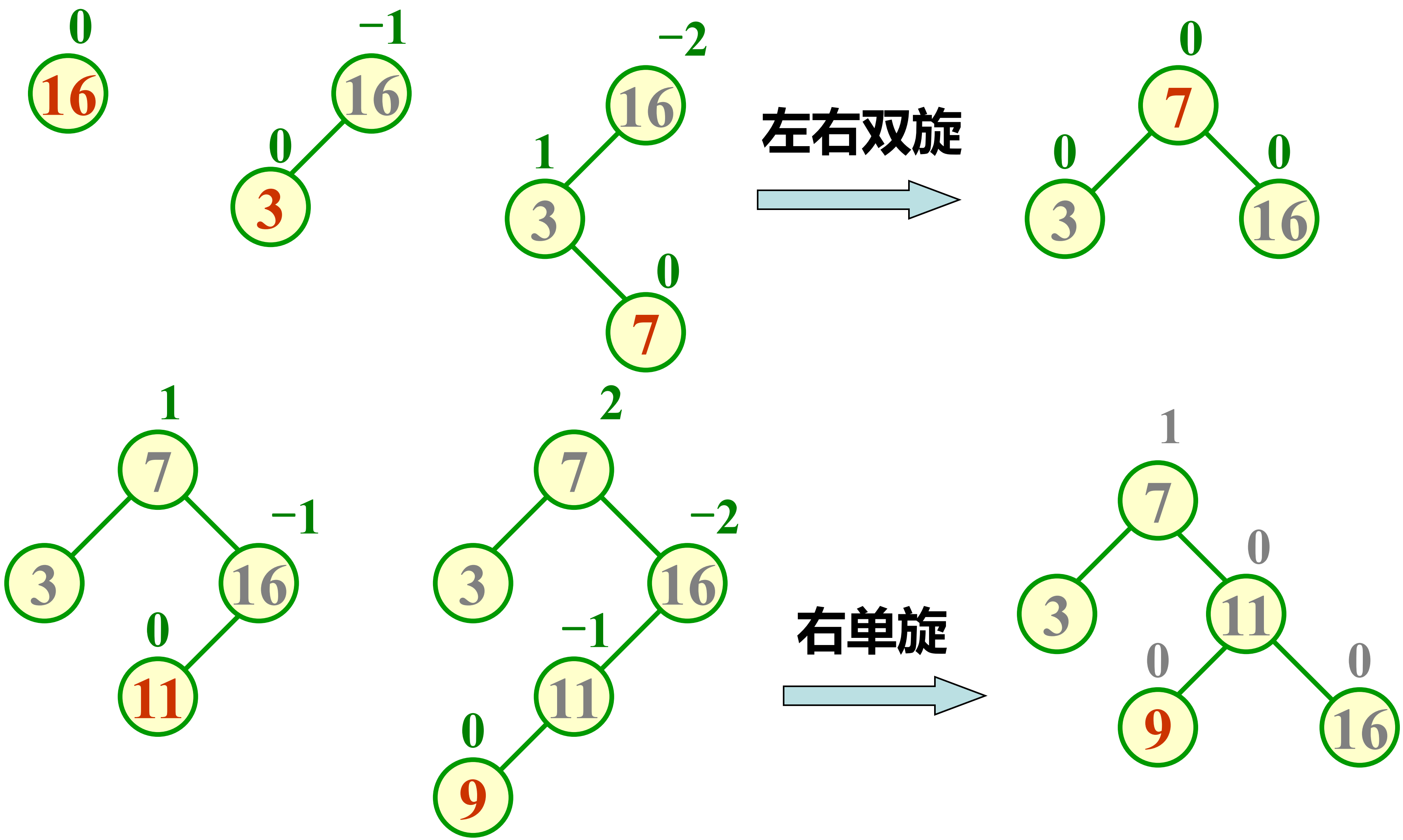


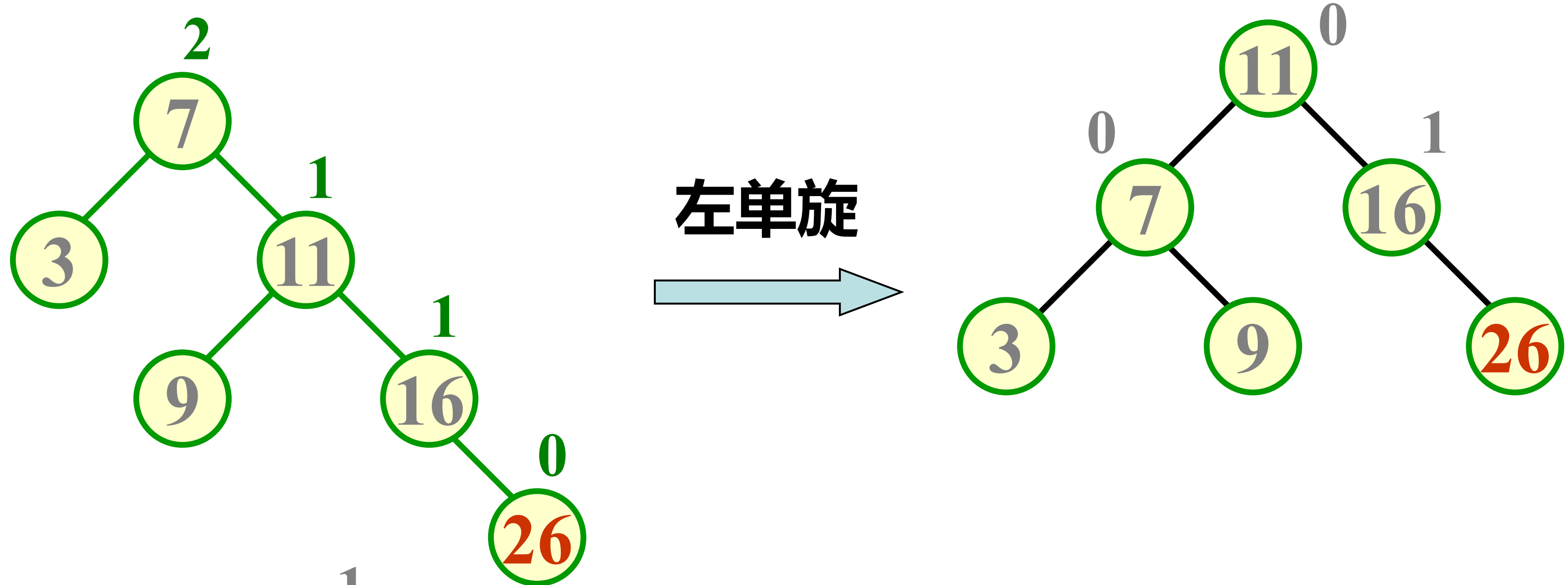
再左旋

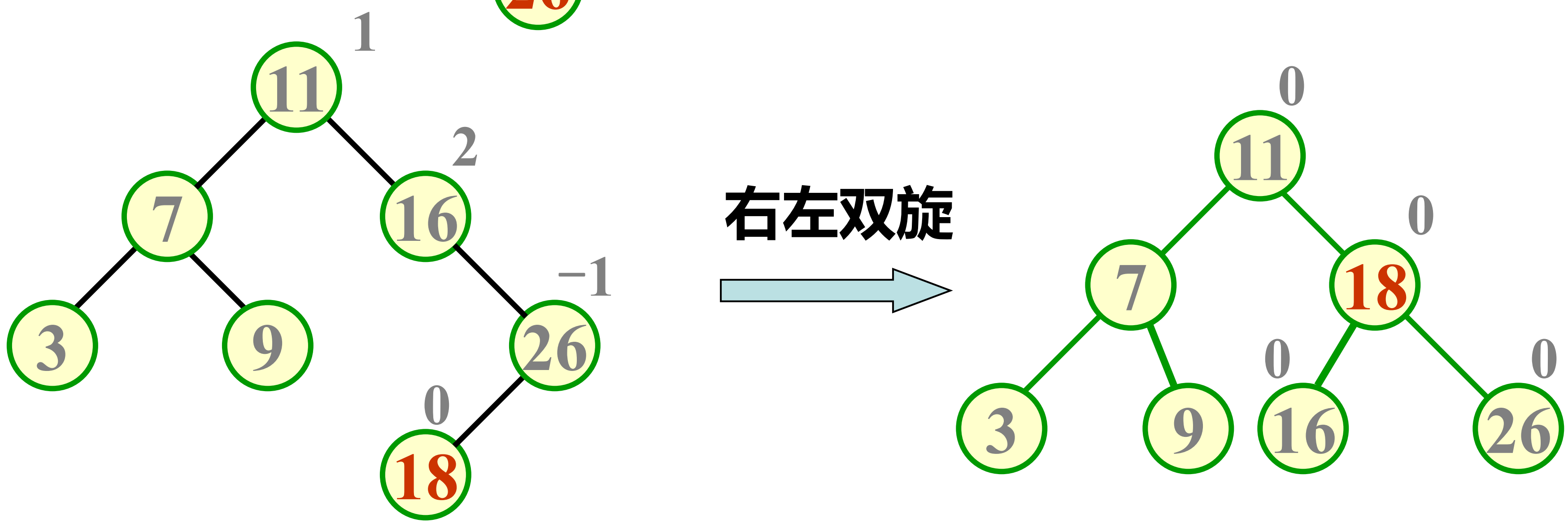
右左（RL）

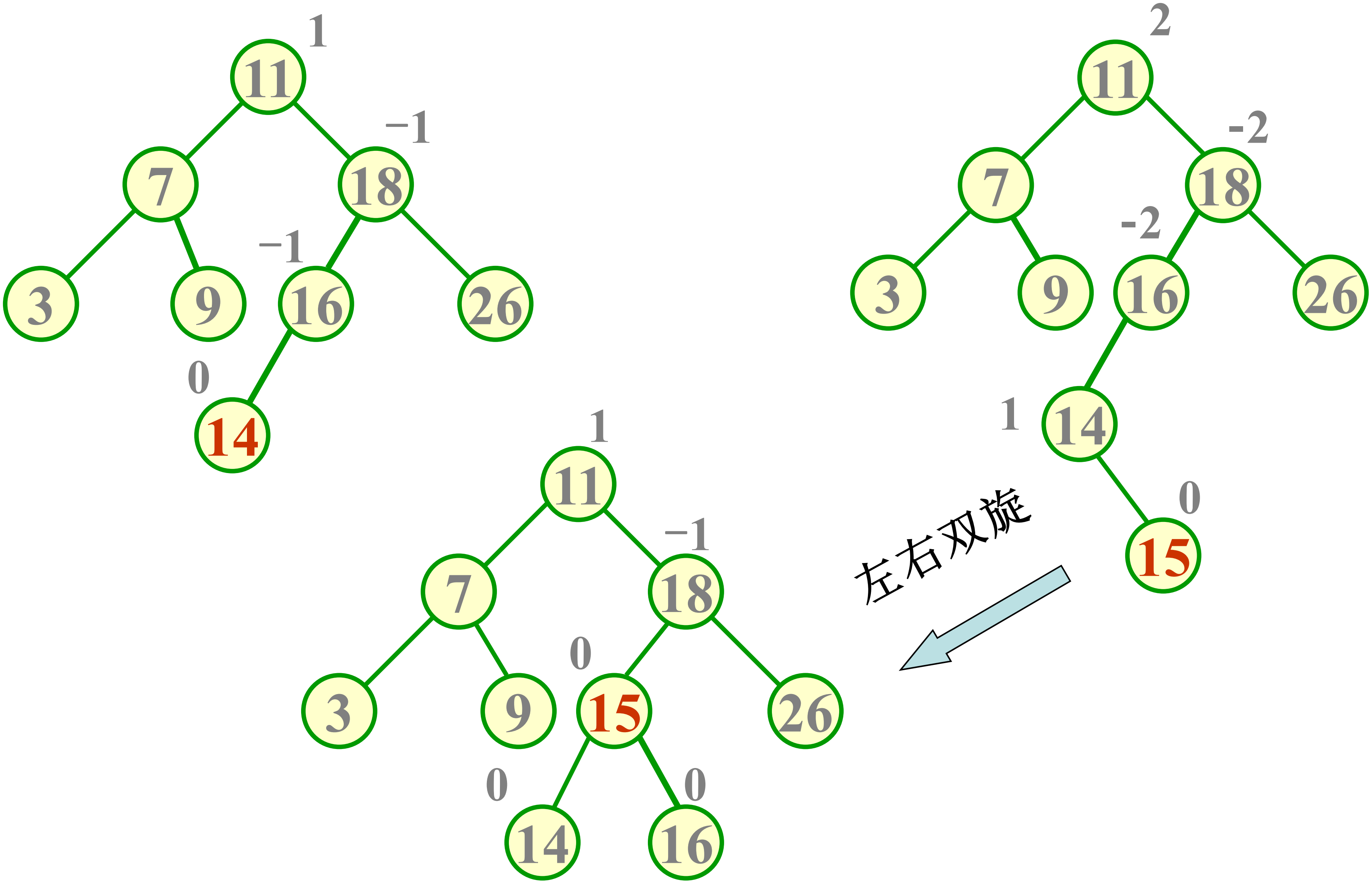

40 1
30 0
50 0
45 0
55 0
Parent 40 2
30 0
50 -1 subR
subRL 45 -1
55 0
43 0
40 2
30 0
45 1
43 0
50 1
55 0
subRL 45 0
Parent 40 0
50 1 subR
30 0
43 0
55 0

例:关键码序列为{16,3,7,11,9,26,18,14,15},插入和调整过程如下。




左单旋


右左双旋

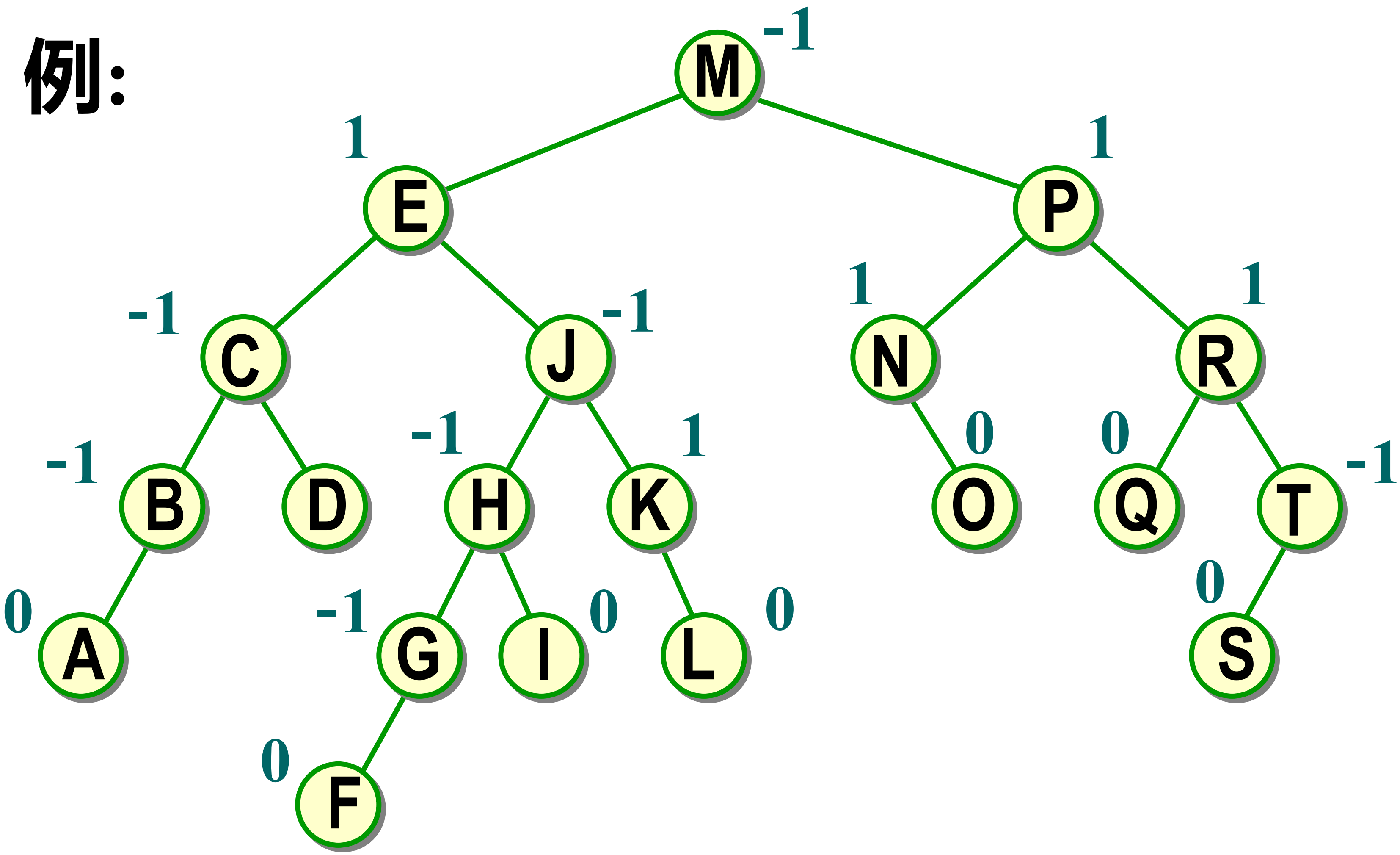

左右双旋

4 AVL树

AVL树的删除运算

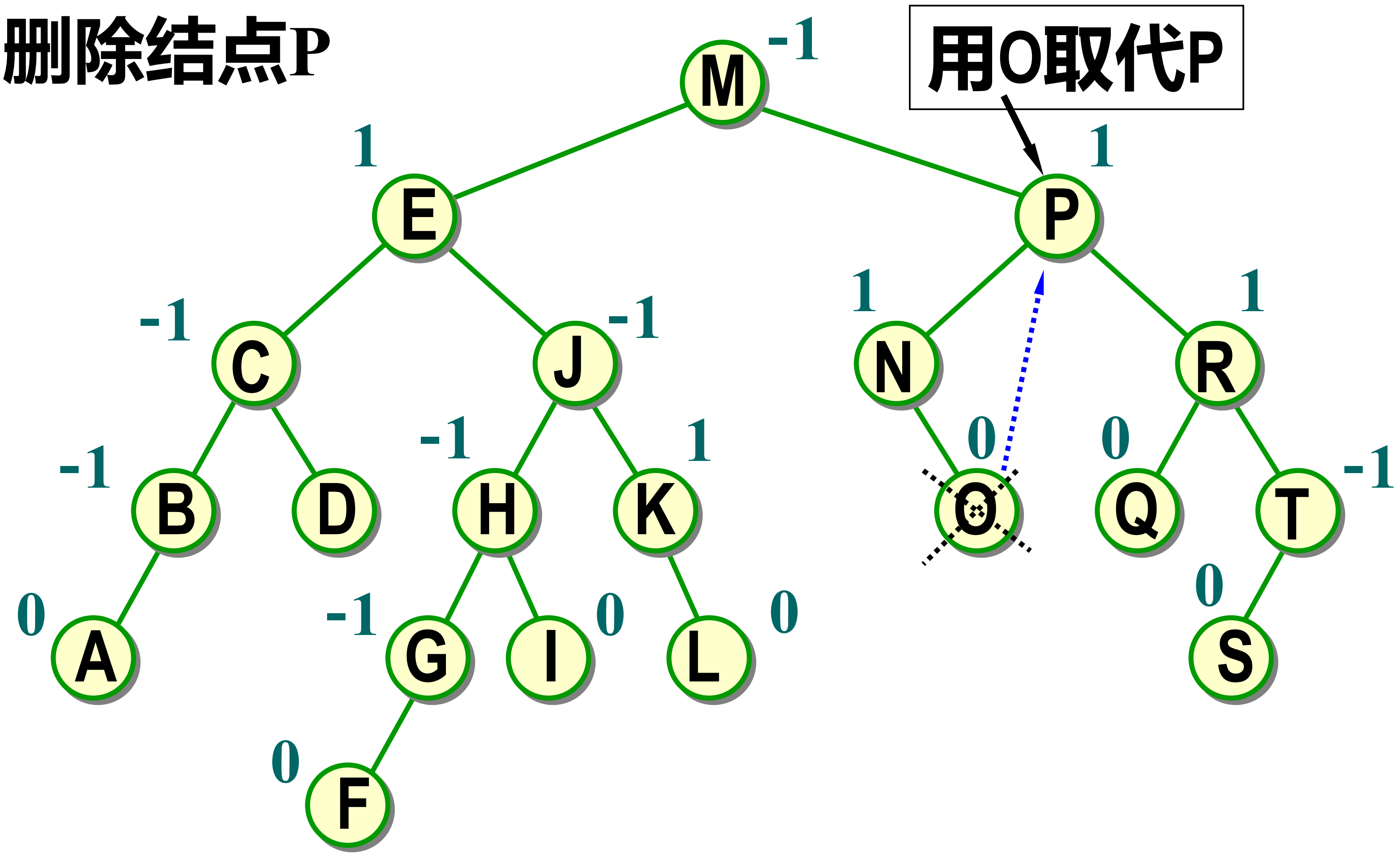
- AVL树与二叉搜索树的删除运算是类似的。惟一的不同之处是，在AVL树中执行1次二叉搜索树的删除运算，可能会破坏AVL树的高度平衡性质，因此需要重新平衡。
- 设被删除结点为p，其惟一的儿子结点为v。结点p被删除后，结点v取代了它的位置。从根结点到结点v的路径上，每个结点处删除运算所进入的子树高度可能减1。因此在执行1次二叉搜索树的删除运算后，需从结点v开始，沿此删除路径向根结点回溯，修正平衡因子，调整子树高度，恢复被破坏的平衡性质。

例:



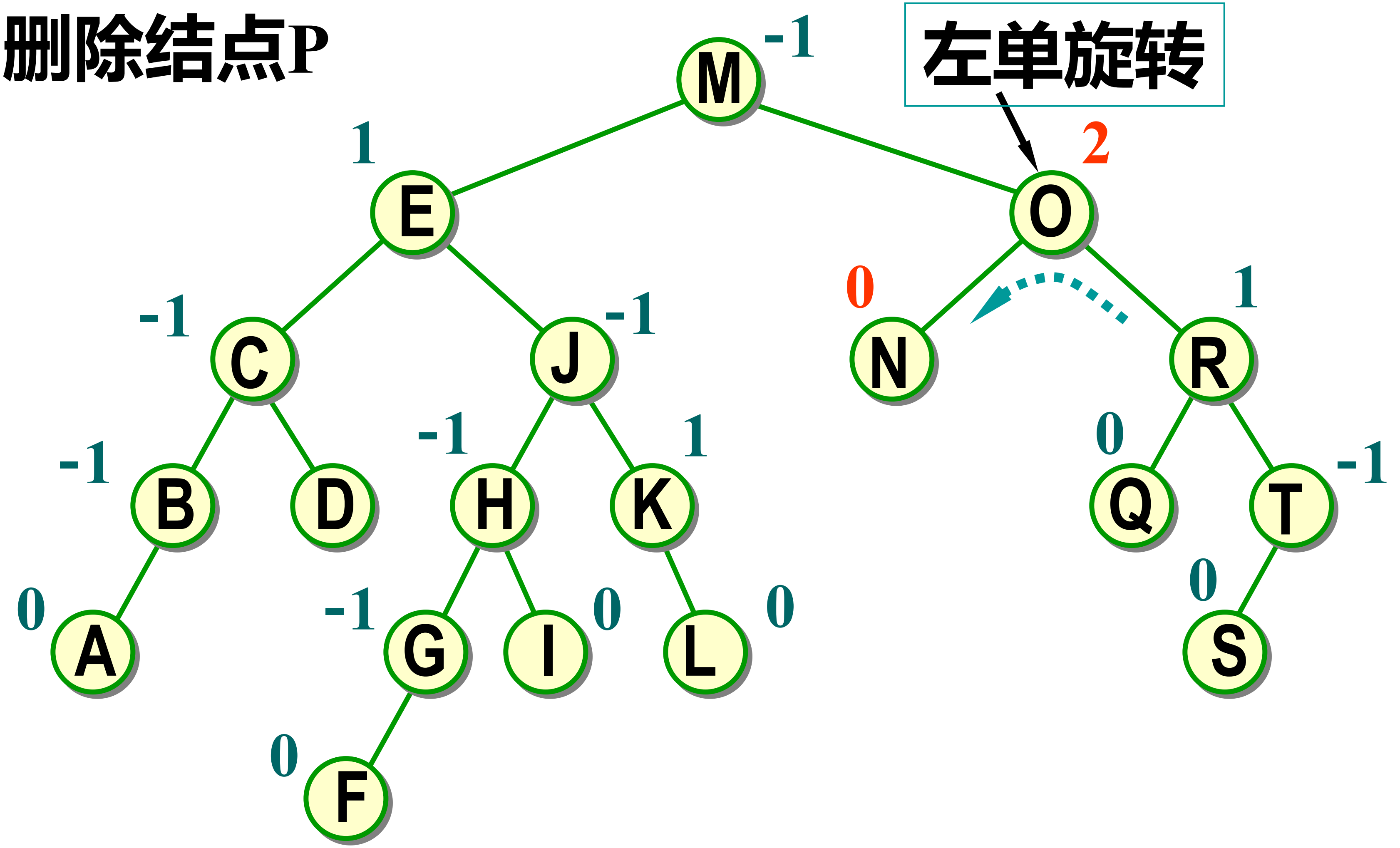
树的初始状态

删除结点P



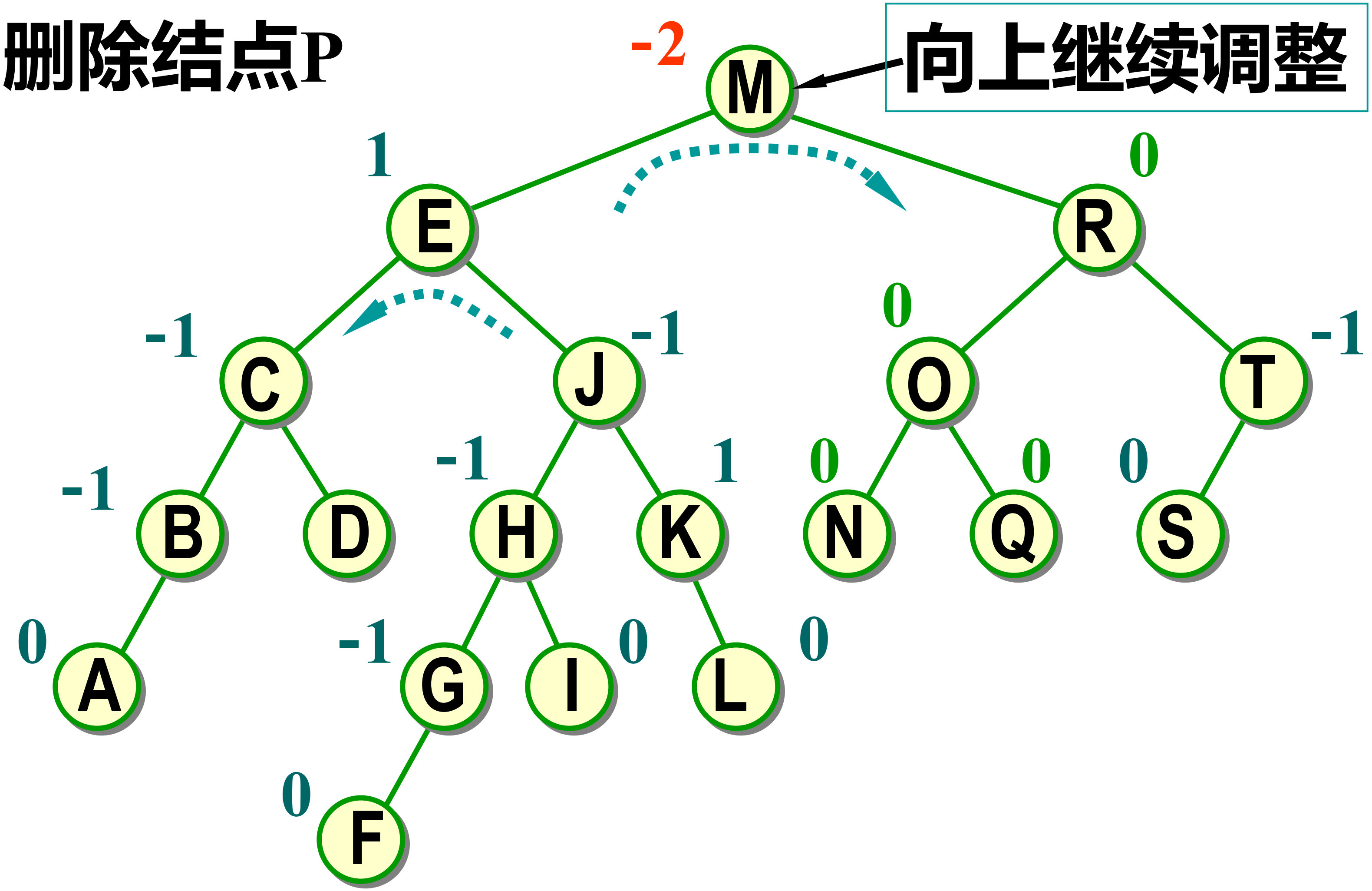
寻找结点P在中序下的直接前驱O, 用O顶替P, 删除O。

删除结点P



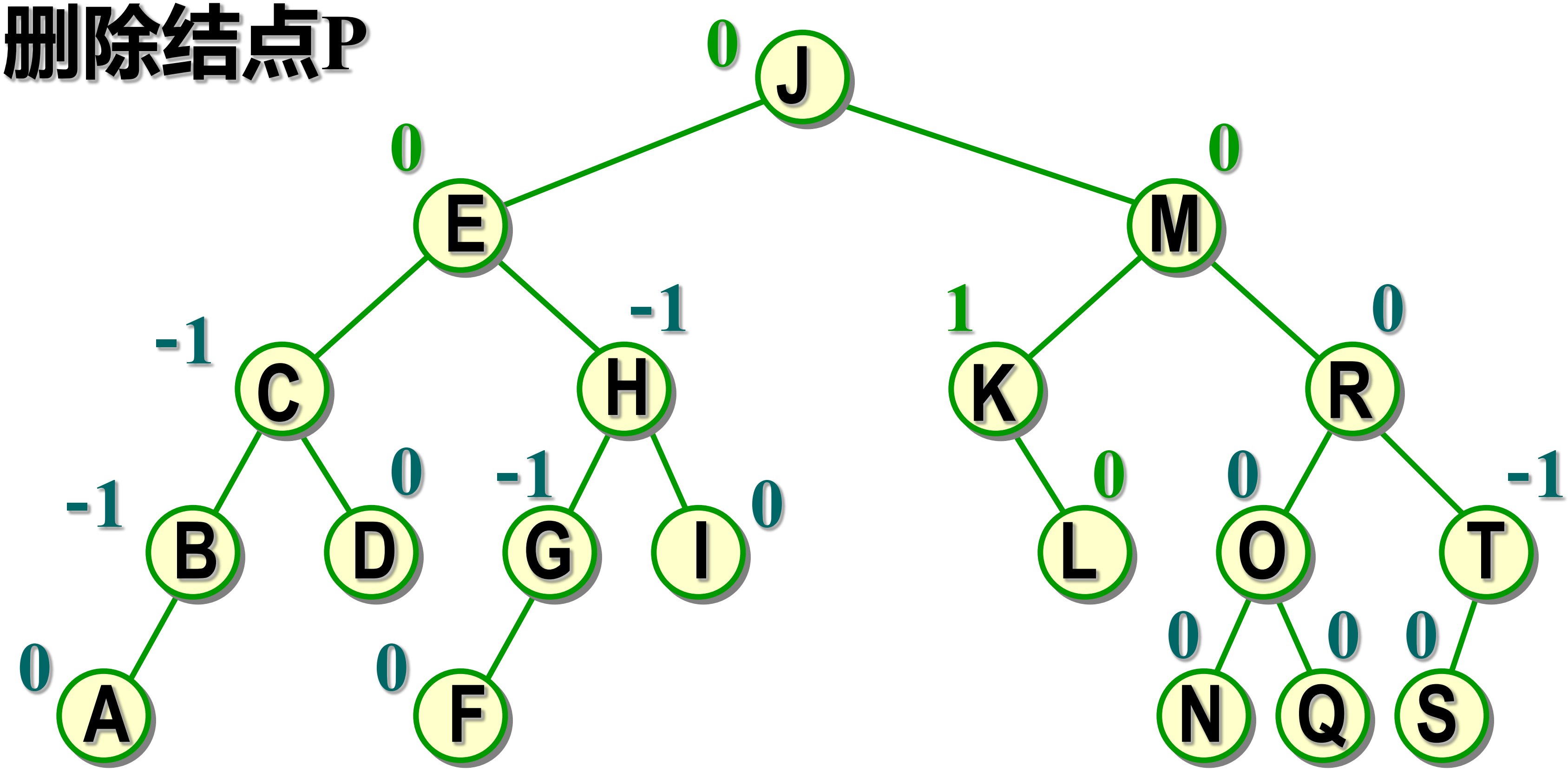
O与R的平衡因子同号, 以R为旋转轴做左单旋转, M的子树高度减 1。

删除结点P



M的子树高度减 1, M发生不平衡。 M与E的平衡因子反号, 做左右双旋转。

删除结点P



返回章目录

5 字典的应用—条形图统计问题

★问题描述：

条形图常用于表示数据分布情况。例如学生考试成绩分布；居民收入分布情况，以及图像灰度等级分布等。当给定的 n 个数据不是非负整数时，通常可以通过映射将它们转换为非负整数。例如，可以将 26 个英文字母{a,b,c,⋯,z}映射为{0,1,⋯,25}。

给定 n 个数据，条形图问题要求绘出表示这 n 个数据的条形统计图，即统计出这 n 个数据中有多少个不同的值，每个值出现的频率是多少。下图是对给定的 10 个非负整数绘制条形图的例子。其中图(a)是输入数据；图(b)是频率统计；图(c)是相应的条形统计图。

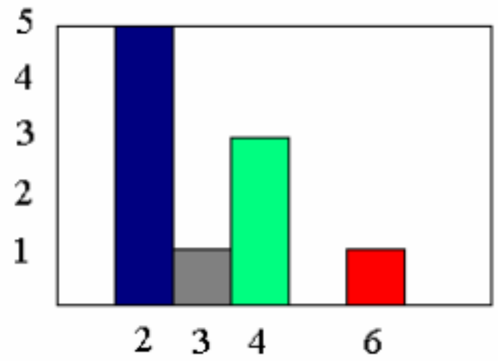
n=10; 输入数据集 s={2,4, 2,2,3,4,2,6,4,2}

(a)

数据值	2	3	4	6
频　率	5	1	3	1

(b)

5 字典的应用—条形图统计问题



(c)

★编程任务：

对于给定的 n 个正整数，设计并实现解条形图问题的 $O(n\log n)$时间算法。如果这 n 个正整数中只有 m 个互不相同的正整数，算法的计算时间为 $O(n\log m)$。

THE END