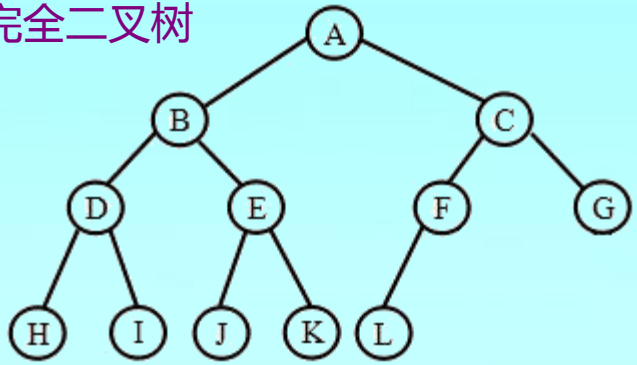
本节主题：

平衡二叉树(AVL)

排序二叉树的问题

理想：接近完全二叉树



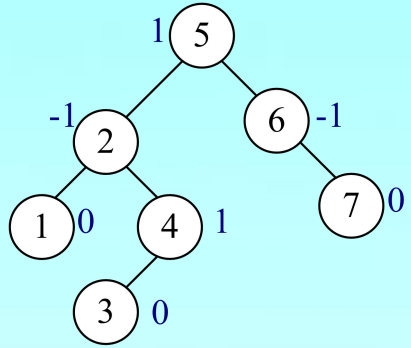
现实选择：左右子树高度不要相差太多！

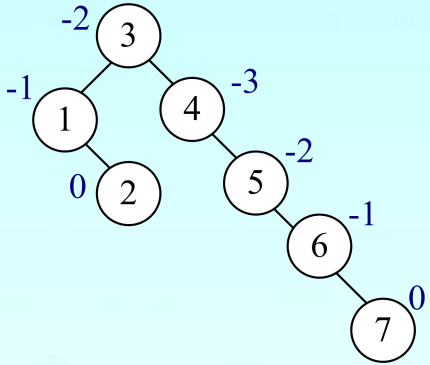
对排序二叉树的查找，
在最坏情况下，
可能会退化为顺序查找！

平衡二叉树

AVL树，平衡二叉树中的经典以发明者名字命名

- 定义
  - 若一棵二叉树中每个节点的左、右子树的高度至多相差1，则称此二叉树为**平衡二叉树**。
- 度量
  - 通过平衡因子（balancd factor，bf）来具体实现上述平衡二叉树的定义。
  - 每个节点的平衡因子是该节点左子树的高度减去右子树的高度
- 平衡二叉树的新定义（从平衡因子的角度）
  - 若一棵二叉树中所有节点的平衡因子的绝对值小于或等于1，则该二叉树称为平衡二叉树。
  - 平衡二叉树中节点的平衡因子取值为1、0或-1，

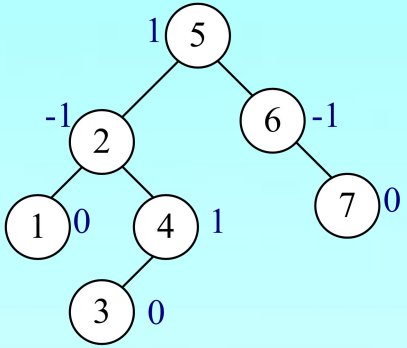




平衡二叉树的存储和操作

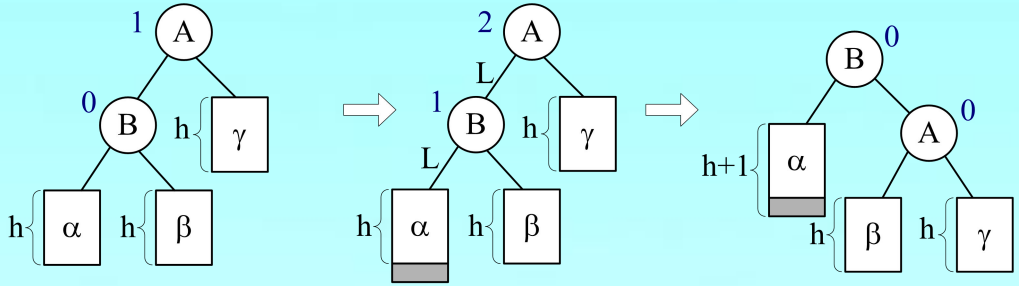
- AVL树的节点类型

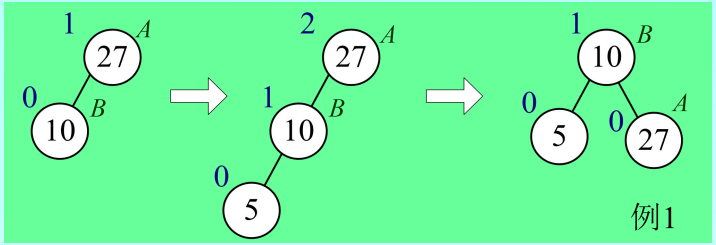
```
typedef struct node                    //记录类型
{
   KeyType key;                        //关键字项
   int bf;                             //平衡因子
   InfoType data;                      //其他数据域
   struct node *lchild,*rchild;        //左右孩子指针
} BSTNode;
```



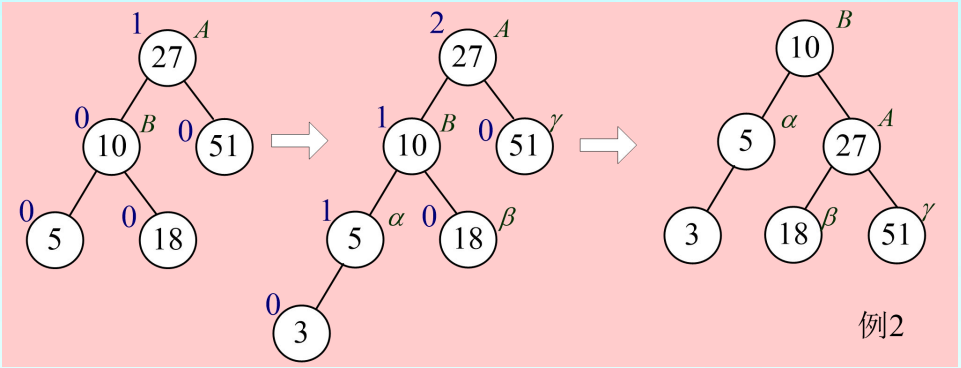
- 重要操作
  - 插入节点
    - 平衡二叉树中插入新节点方式与二叉排序树相似
    - 插入后可能破坏了平衡二叉树的平衡性 → 策略：调整
      - LL型调整
      - RR型调整
      - LR型调整
      - RL型调整
  - 删除节点

(1)LL型调整——针对左孩子的左子树上插入引起的不平衡

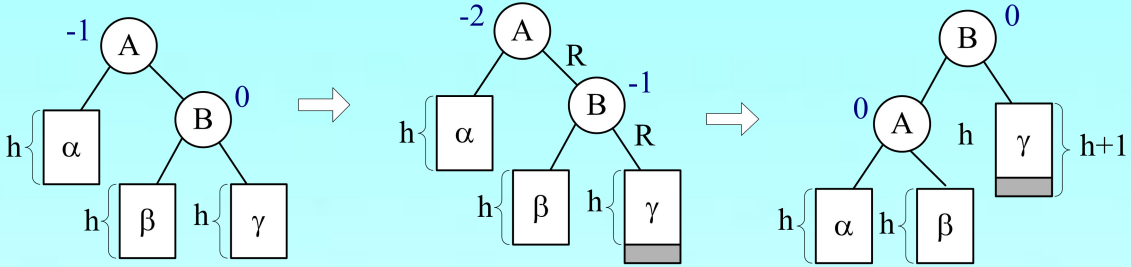




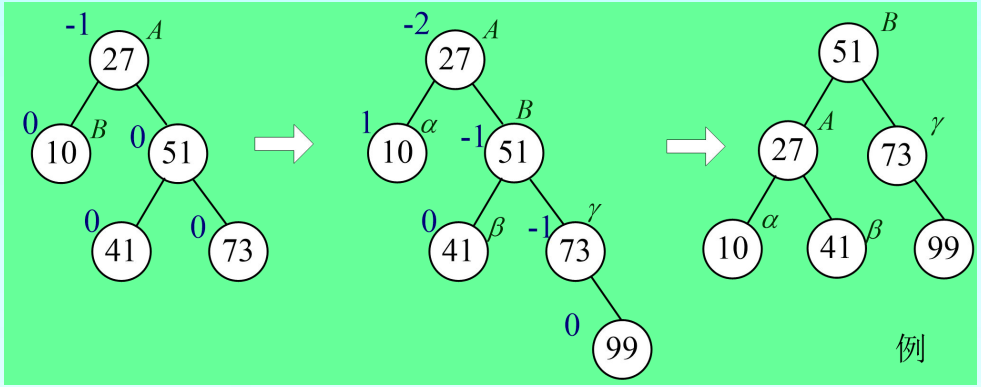
- 调整方法
  - 左孩子B右上旋做根节点
  - 原根节点A右下旋做B的右子树的根节点
  - B的原右子树成为A节点的左子树
- 原理：保持中序
  - 原平衡且有序：$(\alpha B\beta)A(\gamma)$
  - 插入后，有序不平衡：$(\alpha' B\beta)A(\gamma)$
  - 调整后，有序且平衡：$(\alpha')B(\beta A\gamma)$



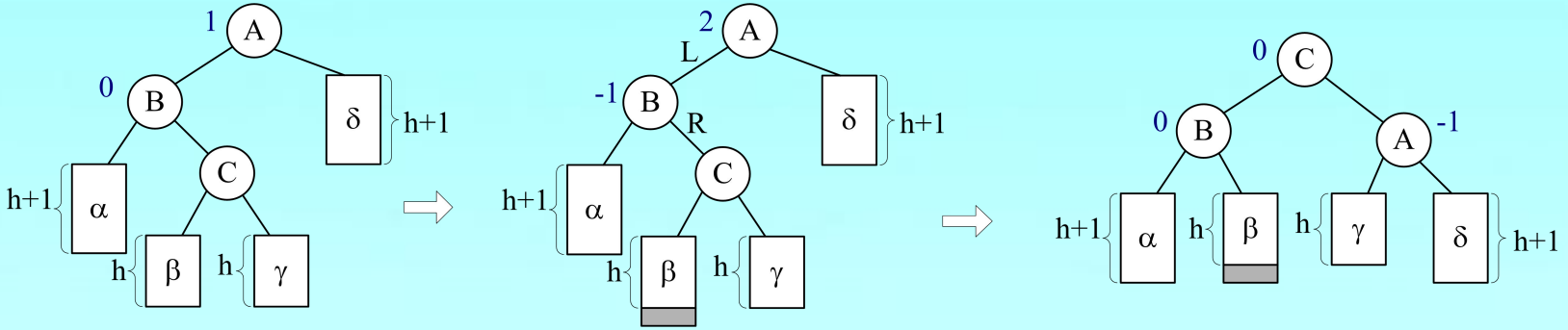
(2)RR型调整——针对右孩子的右子树上插入引起的不平衡



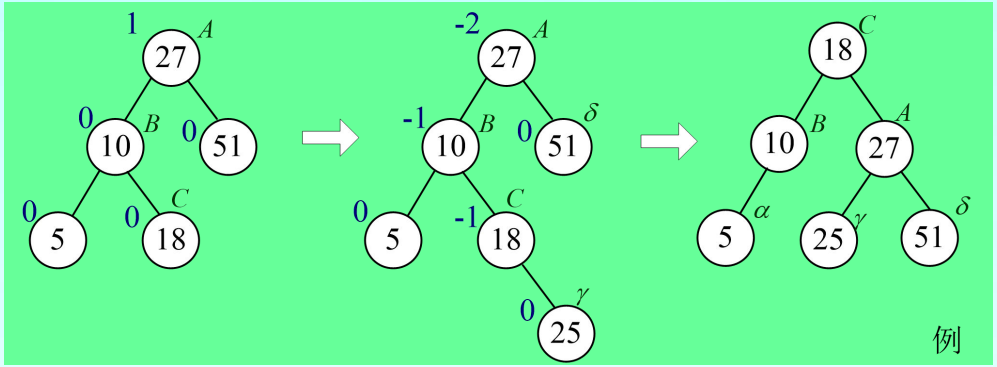
- 调整方法
  - 右孩子B左上旋做根节点
  - 原根节点A左下旋做B的左子树的根节点
  - B的原左子树成为A节点的右子树
- 原理：保持中序
  - 原平衡且有序：$(\alpha)A(\beta B\gamma)$
  - 插入后，有序不平衡：$(\alpha)A(\beta B\gamma')$
  - 调整后，有序且平衡：$(\alpha A\beta)B(\gamma')$



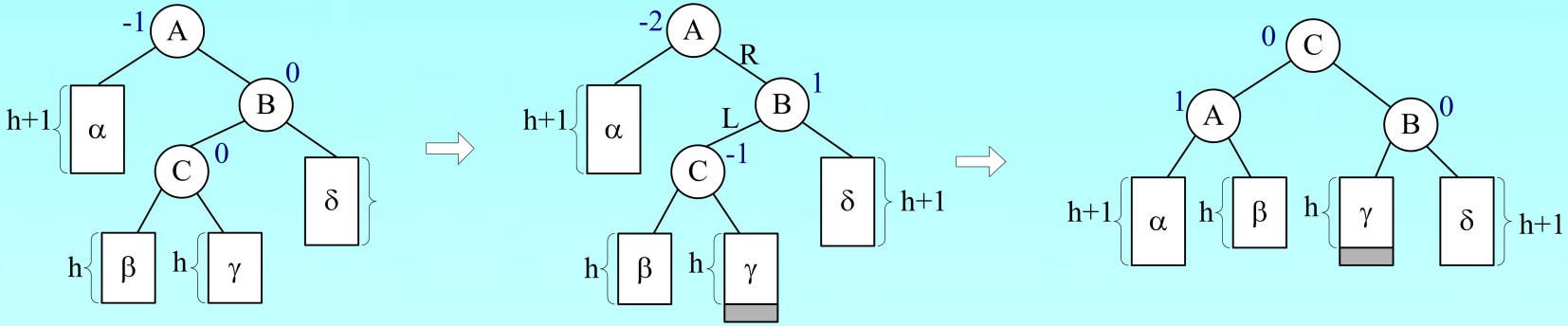
(3)LR型调整——针对左孩子的右子树上插入引起的不平衡



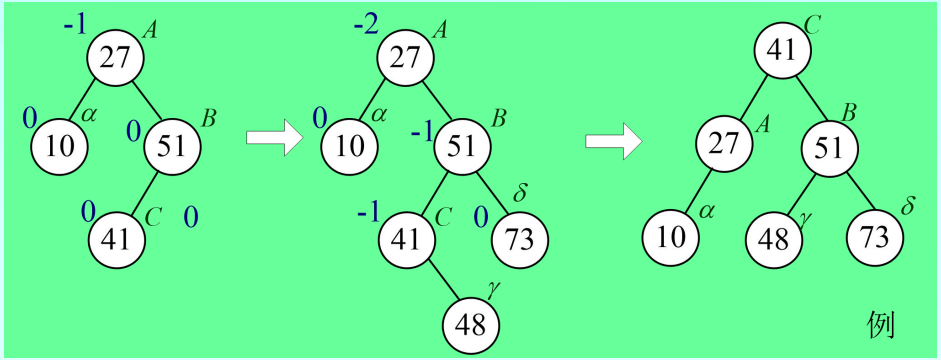
- 原理：保持中序
  - 原平衡且有序：$((\alpha)B(\beta C\gamma))A(\delta)$
  - 插入后，有序不平衡：$((\alpha)B(\beta' C\gamma))A(\delta)$
  - 调整后，有序且平衡：$(\alpha B\beta')C(\gamma A\delta)$



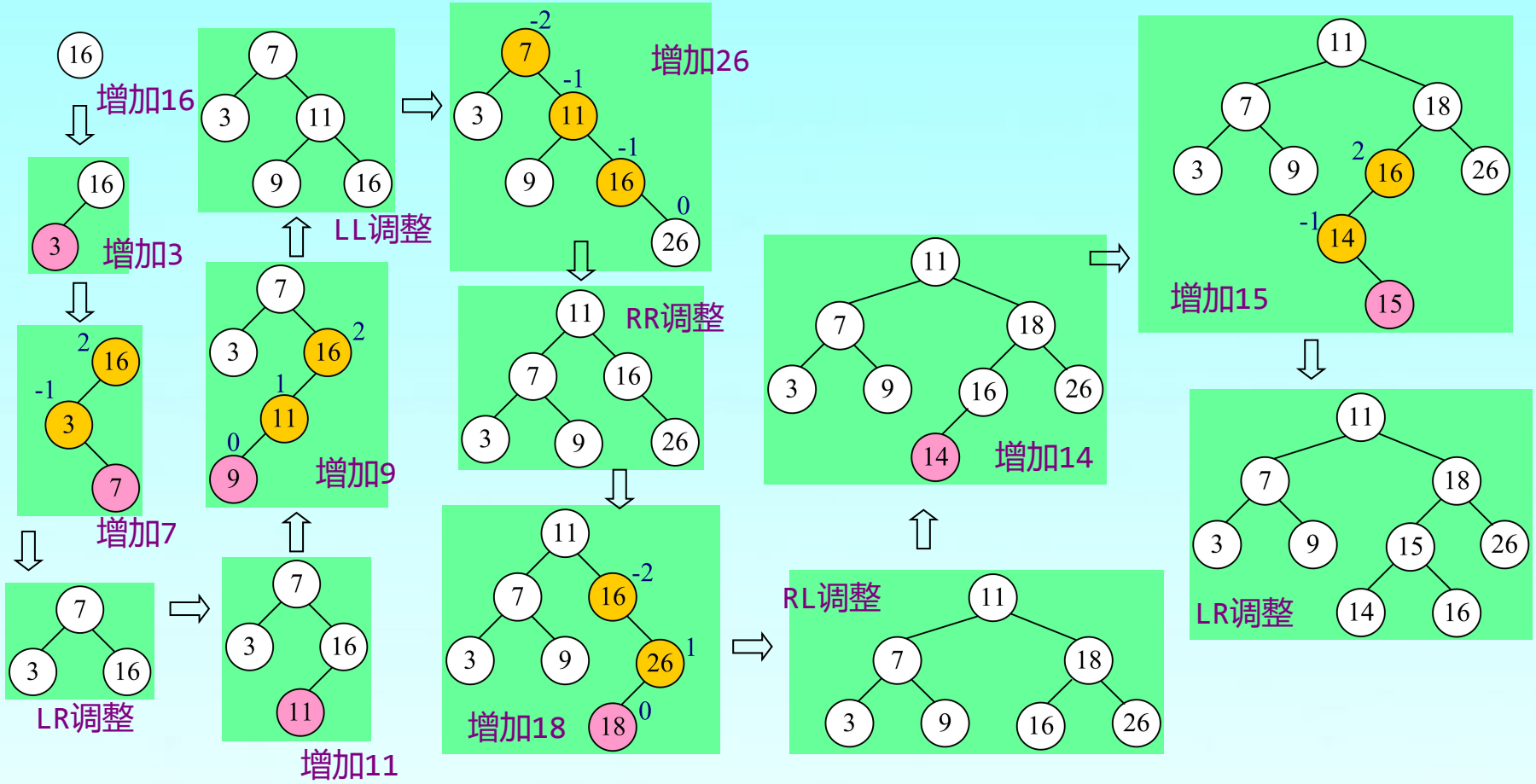
(4)RL型调整——针对右孩子的左子树上插入引起的不平衡



- 原理：保持中序
  - 原平衡且有序：$((\alpha)A(\beta C\gamma))B(\delta)$
  - 插入后，有序不平衡：$((\alpha)A(\beta C\gamma'))B(\delta)$
  - 调整后，有序且平衡：$(\alpha A\beta)C(\gamma' B\delta)$



例 关键字序列{16,3,7,11,9,26,18,14,15},请构造一棵AVL树



平衡二叉树节点的删除

(1)采用二叉排序树中删除节点的方法，找到节点x并删除

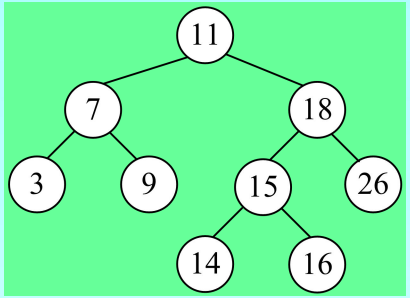
- 叶节点：直接删除
- 非叶子节点：用前驱节点代替

(2)沿被删除节点到根节点的方向，逐层向上查找，必要时修改经过的祖先节点的平衡因子；

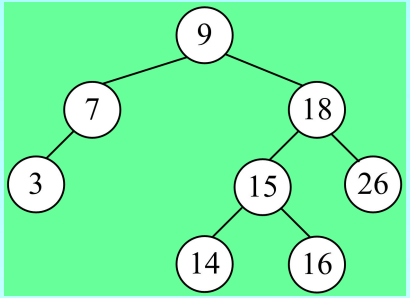
(3)发现“失衡”，立即调整

- 视失衡点的平衡因子，选择RL、RR、LL或者LR调整

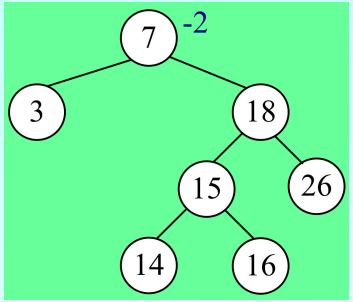
(4)继续调整，直到根节点（删除一个节点，可能需要多次调整，直到根节点）



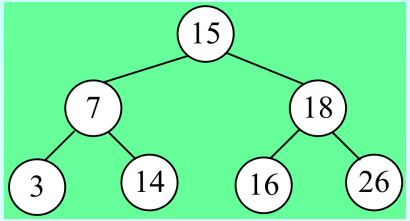
初始AVL



删除11



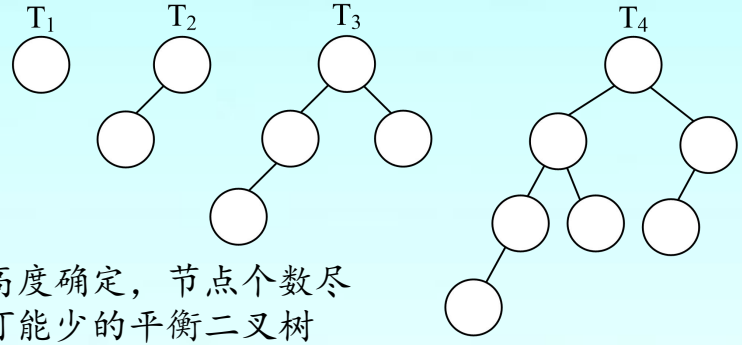
删除9，失衡



RL调整

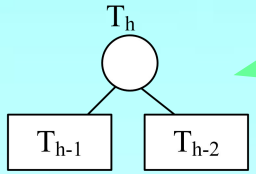
在平衡二叉树上的查找和性能

- 在平衡二叉树上进行查找的过程
  - 和在二叉排序树上进行查找的过程完全相同
  - 在平衡二叉树上进行查找关键字的比较次数不会超过平衡二叉树的高度。
- 平衡二叉树的高度h和节点个数n之间的关系？
  - 构造一系列的平衡二叉树T_1，T_2，T_3，…，其中，T_h（h=1，2，3，…）是高度为h且节点数尽可能少的平衡二叉树，如下图所示的T_1，T_2，T_3和T_4。



高度确定，节点个数尽可能少的平衡二叉树

- 为了构造T_h，先分别构造T_{h-1}和T_{h-2}。



对于每一个T_h，只要从中删去一个节点，就会失去平衡或高度不再是h

- 设N(h)为T_h的节点数，有：

  N(1)=1

  N(2)=2

  N(h)=N(h-1)+N(h-2)+1

h=f(n)?

$h \approx \log_2(N(h)+1)$

在平衡二叉树中查找的最坏复杂度

对比：普通二叉排序树在最坏的情况下，查找次数为O(n)