数据结构

深圳技术大学
大数据与互联网学院

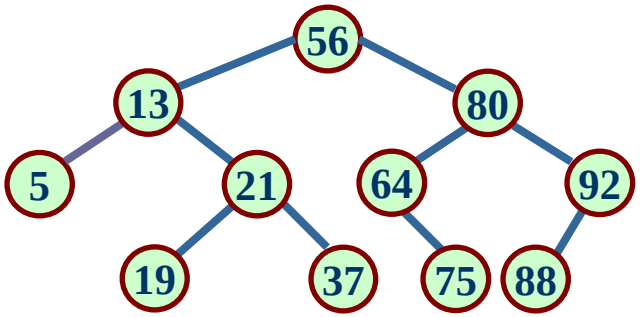
第九章 查找

9.1 静态查找表
9.2 动态查找表
9.3 哈希表

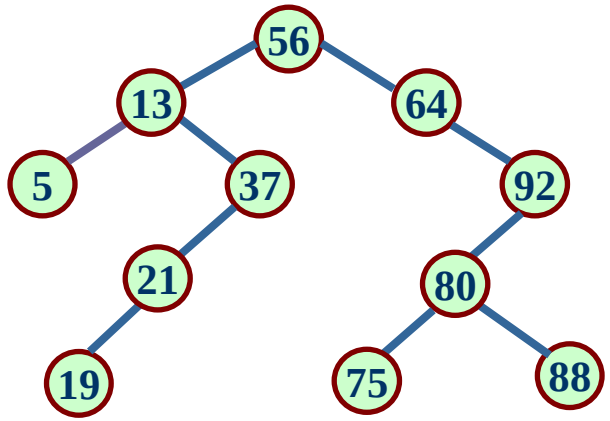
9.2 动态查找表

二. 平衡二叉树

- 平衡二叉树是二叉排序(查找)树的另一种形式
- 平衡二叉树又称AVL树(Adelsen-Velskii and Landis)
  - 其特点为：树中每个结点的左、右子树深度之差的绝对值不大于1，即$|h_L-h_R|\leqslant 1$



AVL树

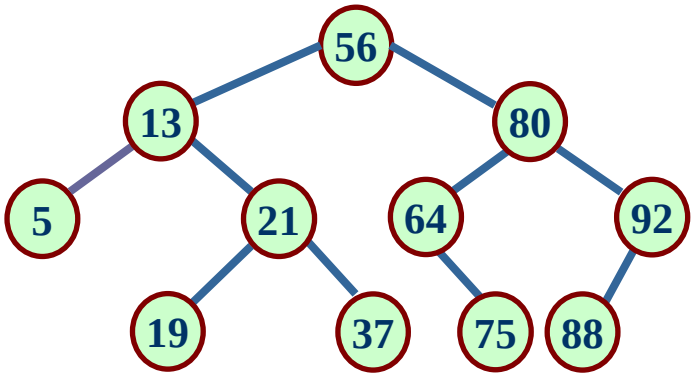


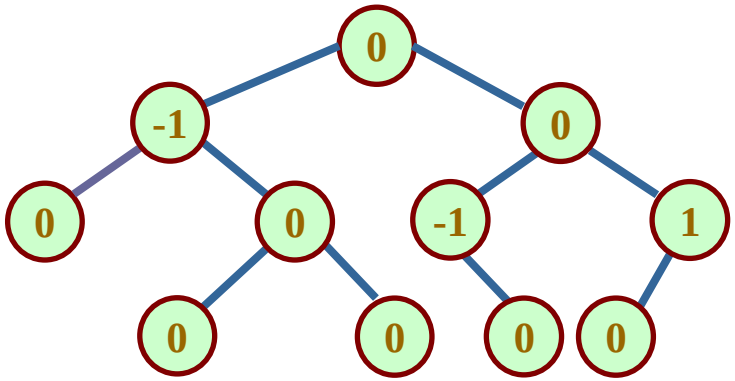
非AVL树

9.2 动态查找表

二. 平衡二叉树

- 平衡因子
  - 每个结点附加一个数字，给出该结点左子树的高度减去右子树的高度所得的高度差，这个数字即为结点的平衡因子balance
  - AVL树任一结点平衡因子只能取 -1，0，1





9.2 动态查找表

二. 平衡二叉树

- 平衡化旋转
  - 如果在一棵平衡的二叉查找树中插入一个新结点，造成了不平衡。此时必须调整树的结构，使之平衡化。
- 平衡化旋转(处理)有两类：
  - 单向旋转（单向右旋和单向左旋）
  - 双向旋转（先左后右旋转和先右后左旋转）

9.2 动态查找表

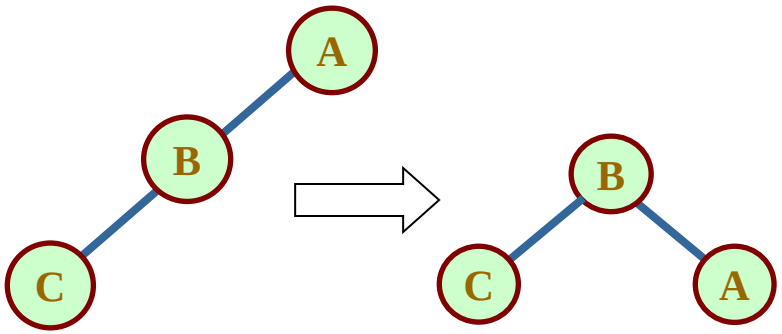
二. 平衡二叉树

- 平衡化旋转
  - 每插入一个新结点时，AVL树中相关结点的平衡状态会发生改变
  - 在插入一个新结点后，需要从插入位置沿通向根的路径回溯，检查各结点的平衡因子
  - 如果在某一结点发现高度不平衡，停止回溯。
  - 从发生不平衡的结点起，沿刚才回溯的路径取直接下两层的结点。对这三个结点进行平衡化处理
  - 平衡化处理包括：
    - 单向右旋
    - 单向左旋
    - 先左后右旋转
    - 先右后左旋转

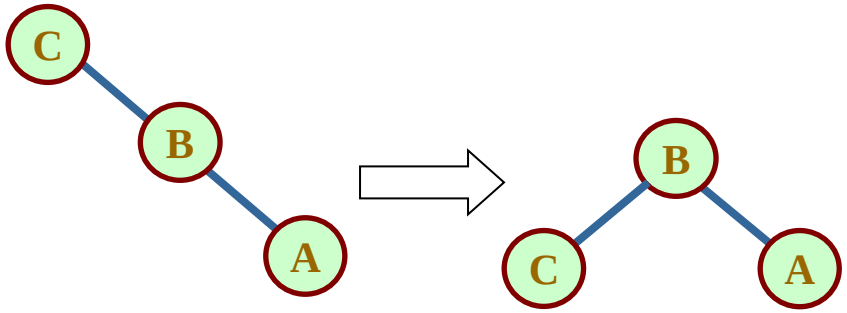
9.2 动态查找表

二. 平衡二叉树

- 平衡化单向旋转
  - 如果这三个结点处于一条直线上(“/”型或 “\”型)，则采用单向旋转进行平衡化[A>B>C]
  - 单向旋转分为单向右旋(“/”型)和单向左旋(“\”型)



单向右旋

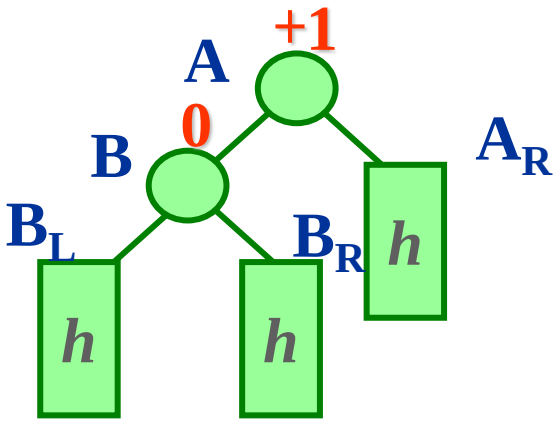


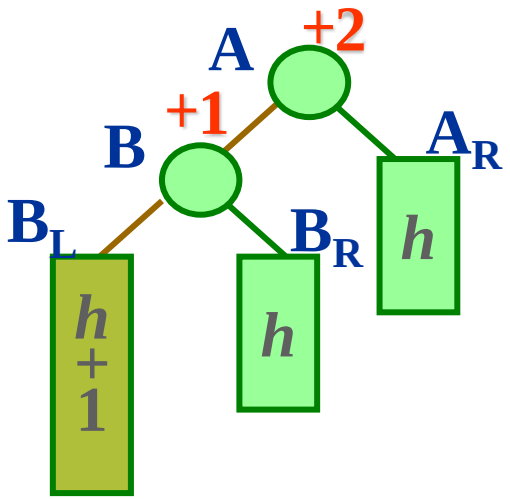
单向左旋

9.2 动态查找表

二. 平衡二叉树

- 单向右旋
  - 在B左子树B_L上插入新结点使其高度增1，导致结点A的平衡因子增到+2，造成不平衡进行平衡化[A>B>C]

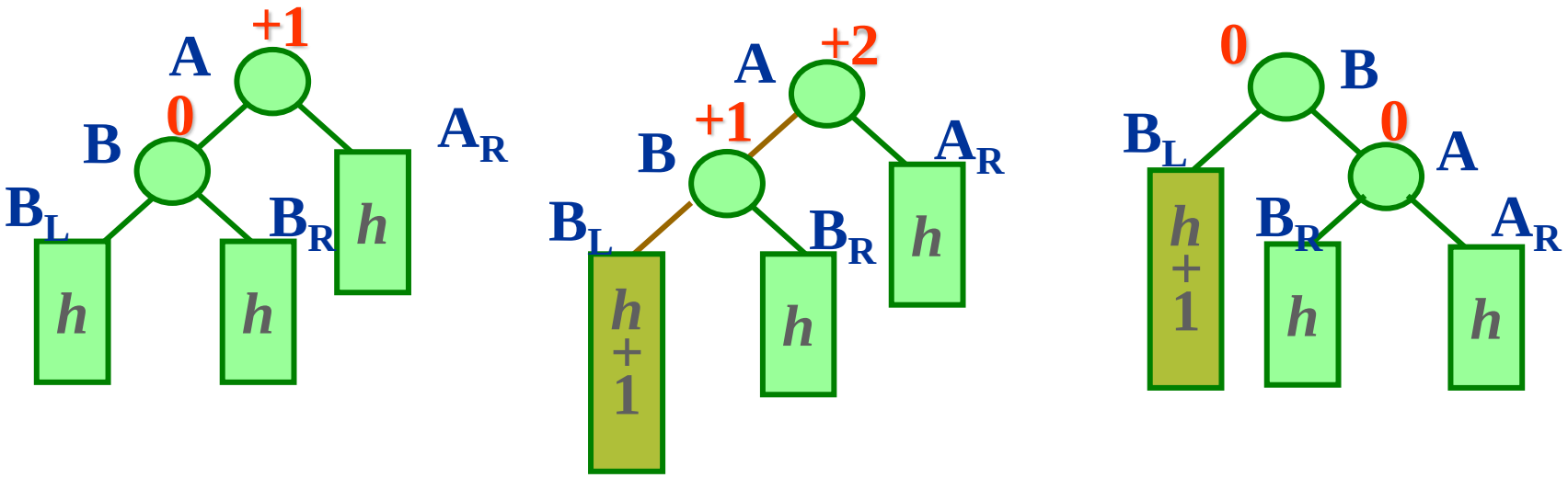




9.2 动态查找表

二. 平衡二叉树

- 单向右旋的具体操作
  - 为使树恢复平衡，从A沿插入路径连续取3个结点A、B和B_L（“/”型）
  - 以结点B为旋转轴，将结点A顺时针(右)旋转。

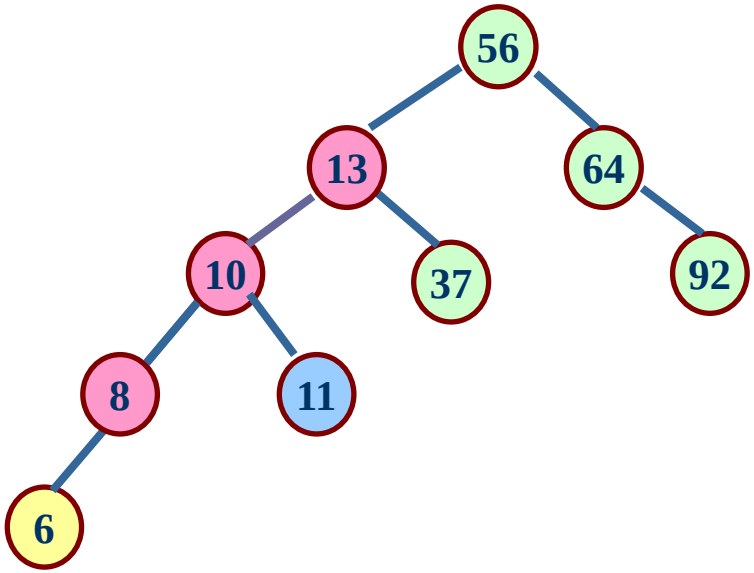


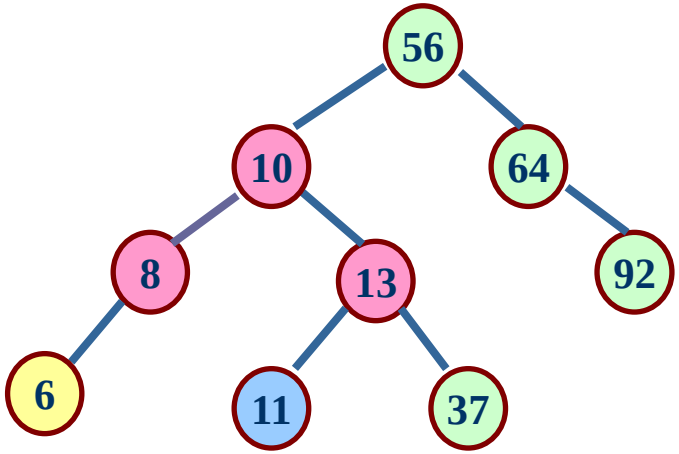
  - 实际操作，A变为B的右孩子，原B右孩子变为A左孩子

9.2 动态查找表

二. 平衡二叉树

- 举例，已知AVL树如下图，插入新数据6，求插入后的树

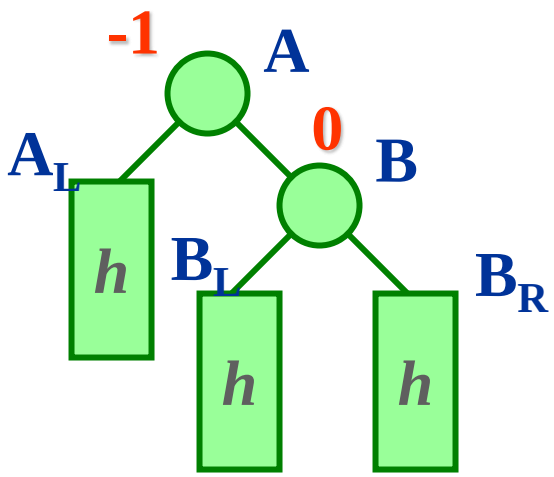


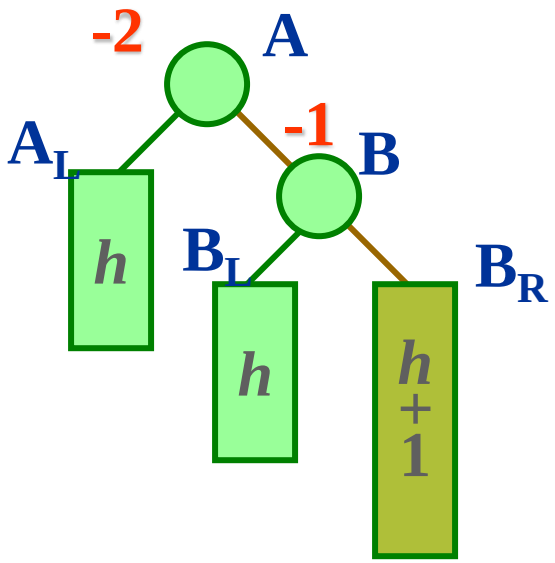


9.2 动态查找表

二. 平衡二叉树

- 单向左旋
  - 在B右子树B_R中插入新结点，该子树高度增1导致结点A的平衡因子变成-2，出现不平衡

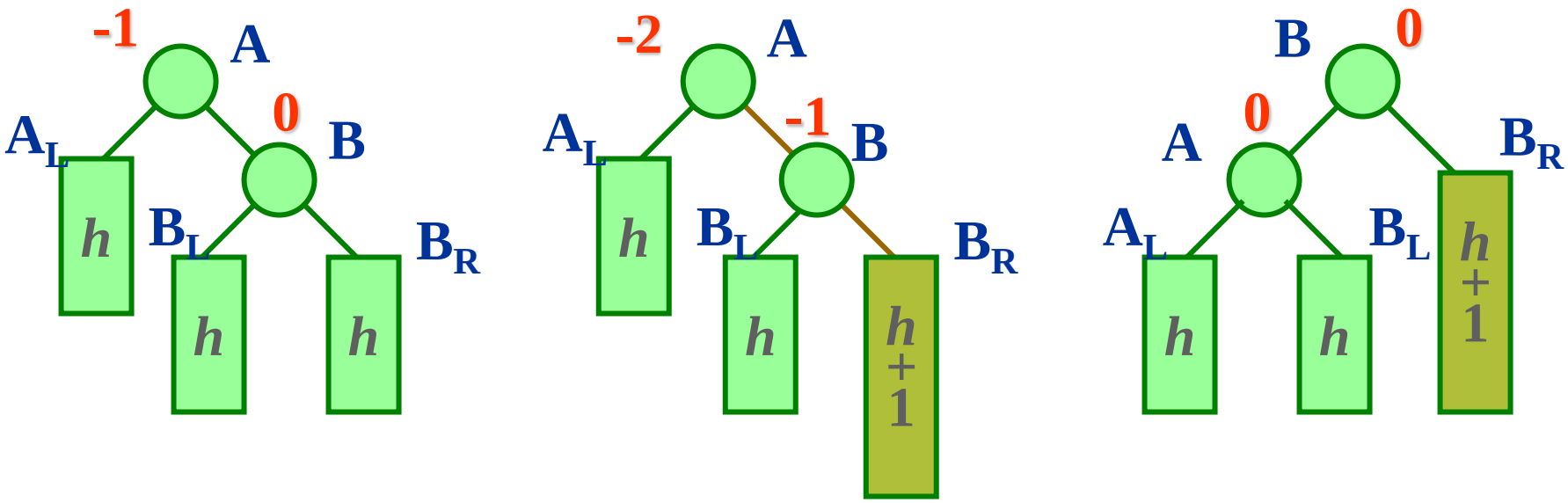




9.2 动态查找表

二. 平衡二叉树

- 单向左旋
  - 沿插入路径检查三个结点A、B和B_R（“\”型）
  - 以结点B为旋转轴，让结点A反时针(左)旋转

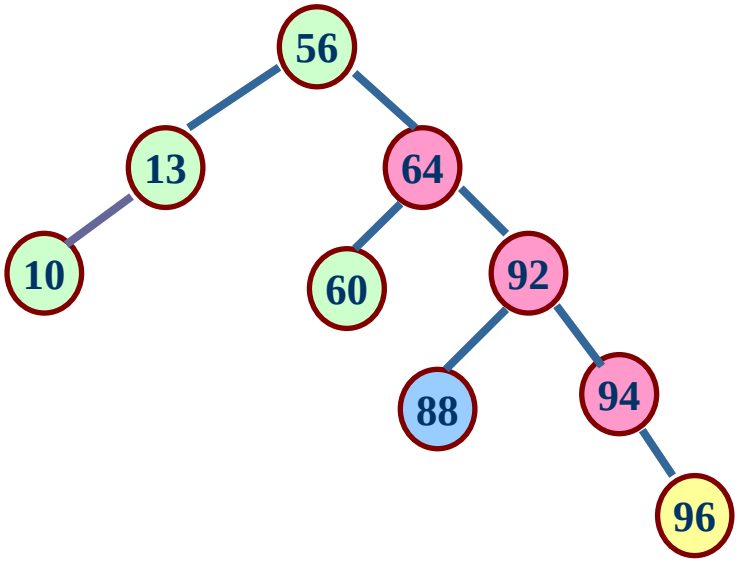


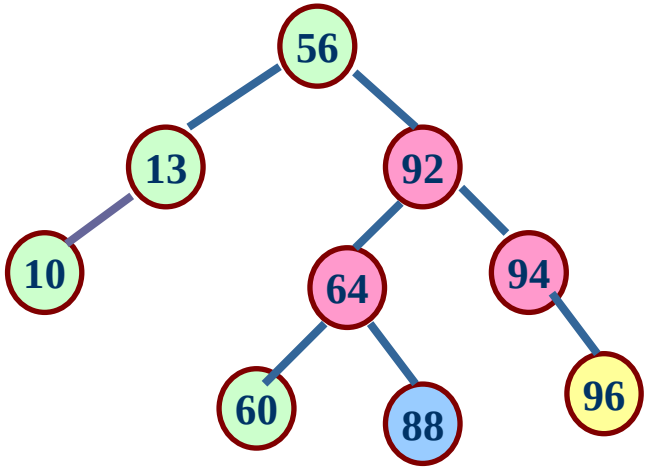
  - 实际操作，A变为B的左孩子，原B左孩子变为A右孩子

9.2 动态查找表

二. 平衡二叉树

- 举例，已知AVL树如下图，插入新数据96，求插入后的树

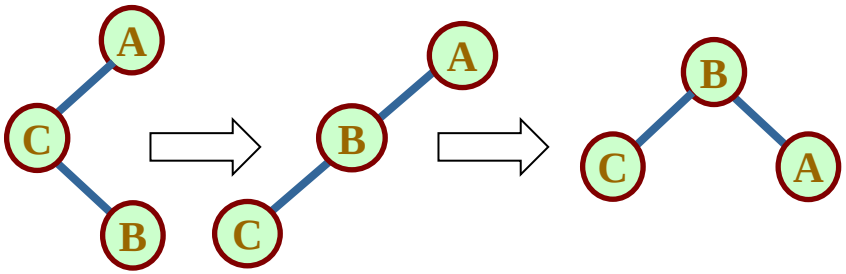




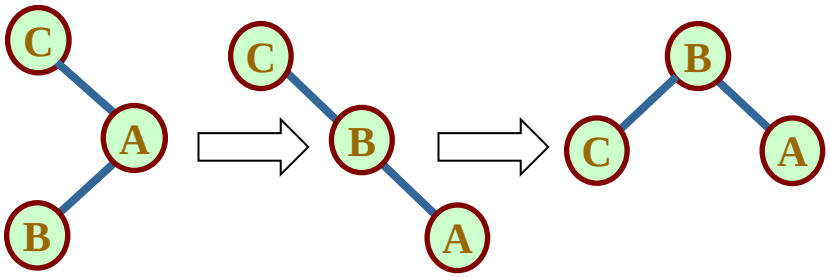
9.2 动态查找表

二. 平衡二叉树

- 平衡化双向旋转
  - 如果这三个结点处于一条折线上(“<”型或 “>”型)，则采用双向旋转进行平衡化[A>B>C]
  - 双旋转分为先左后右(“<”型)和先右后左(“>”型)



先左后右旋转

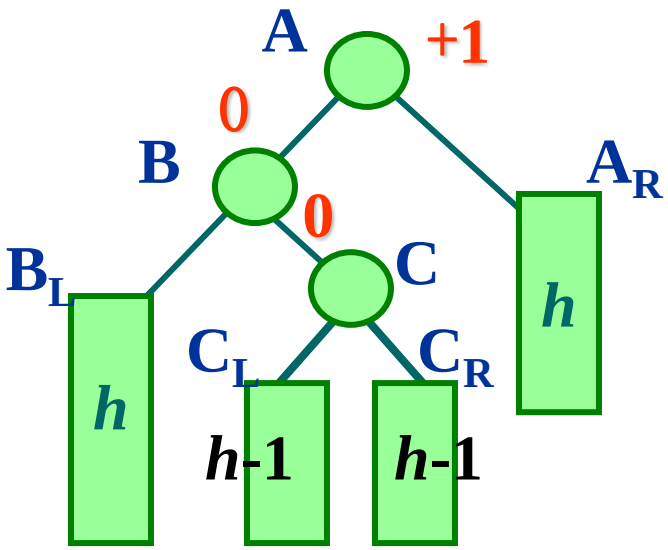


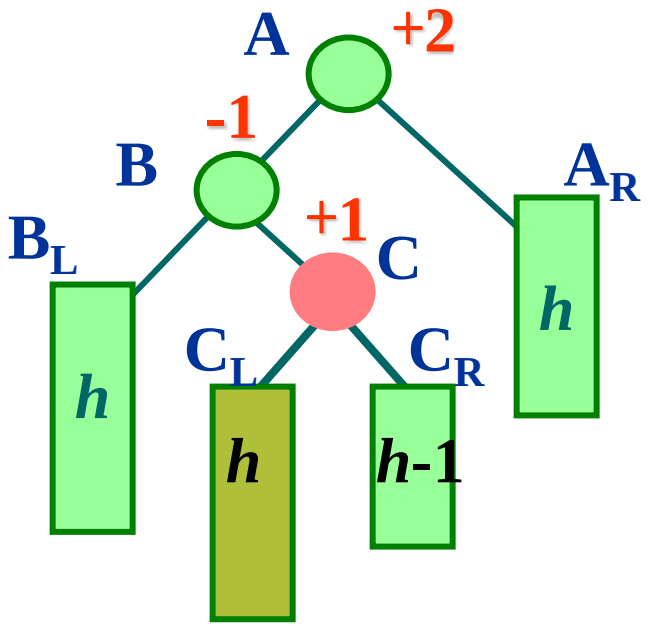
先右后左旋转

9.2 动态查找表

二. 平衡二叉树

- 先左后右双向旋转
  - 在C的子树C_L或C_R中插入新结点，该子树的高度增1。结点A的平衡因子变为+2，发生了不平衡

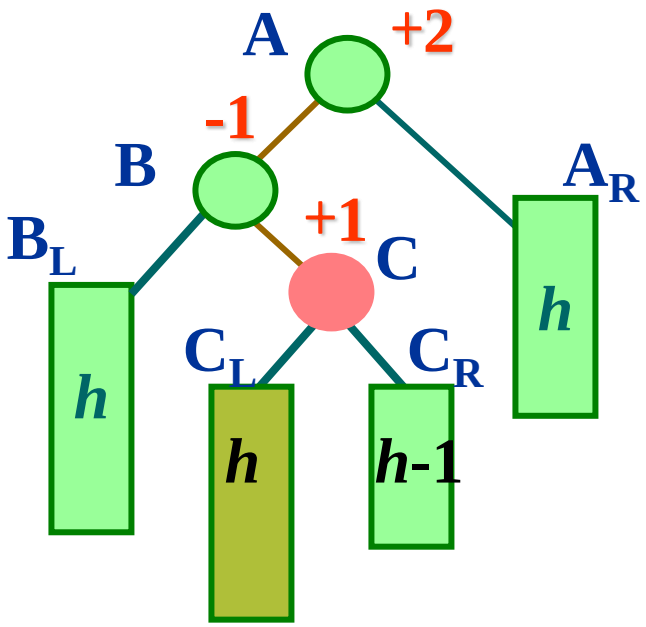


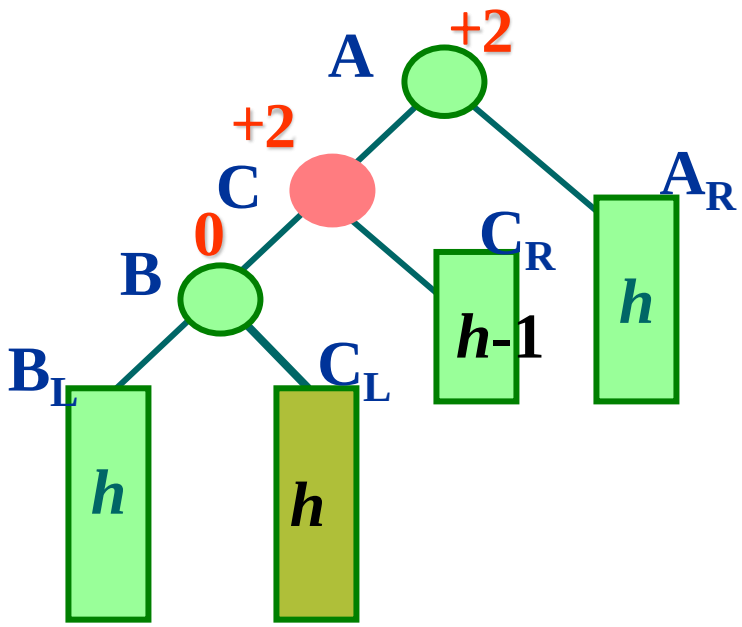


9.2 动态查找表

二. 平衡二叉树

- 先左后右双向旋转
  - 从结点A起沿插入路径选取3个结点A、B和C(“<”型)
  - 以结点B为旋转轴，做单向左旋



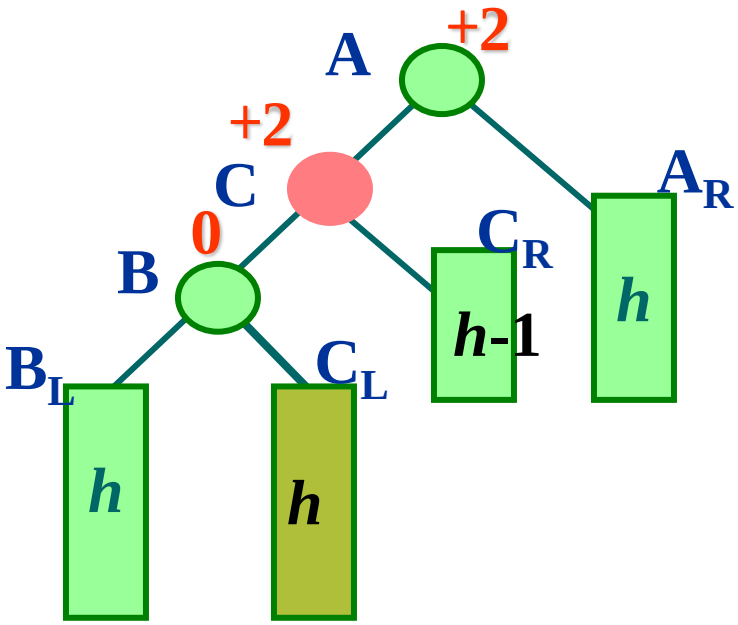


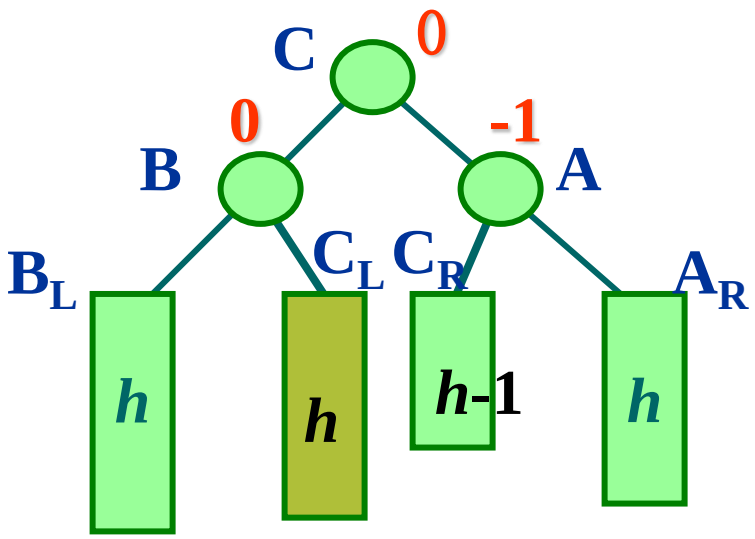
  - A左孩子变为C(C替换B)，B变为C左孩子，原C左孩子变为B右孩子

9.2 动态查找表

二. 平衡二叉树

- 先左后右双向旋转
  - 再以结点C为旋转轴，做单向右旋



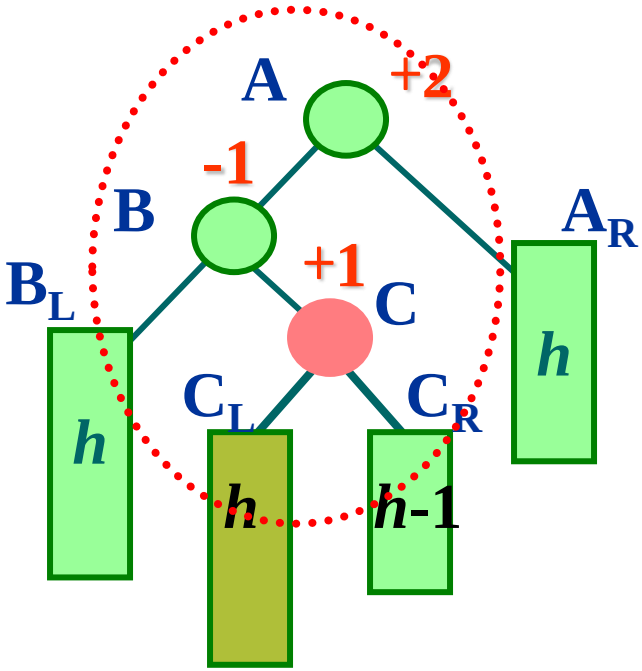


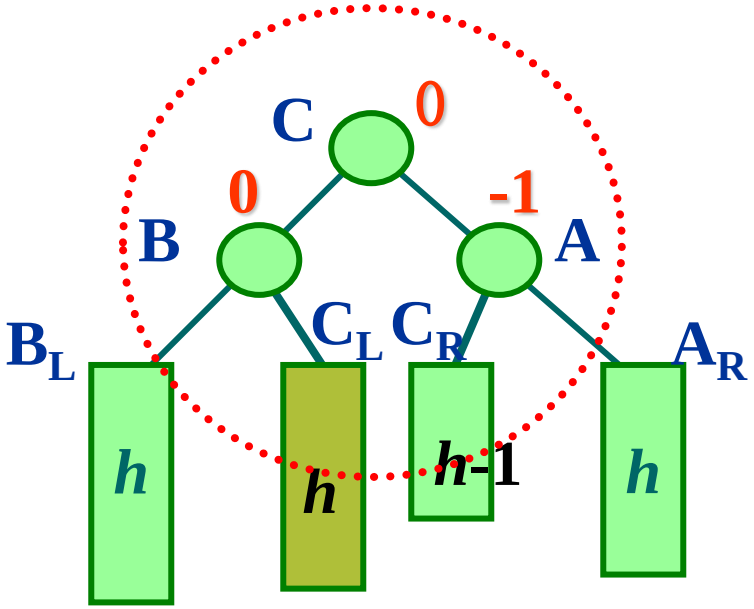
  - A变为C右孩子，原C右孩子变为A左孩子

9.2 动态查找表

二. 平衡二叉树

- 先左后右双向旋转，全部操作
  - B变为C左孩子，A变为C右孩子
  - C左孩子变为B右孩子，C右孩子变为A左孩子

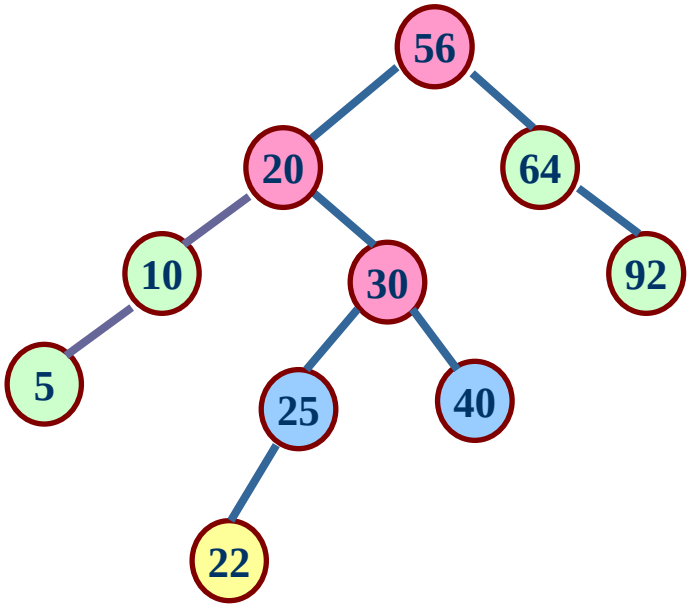


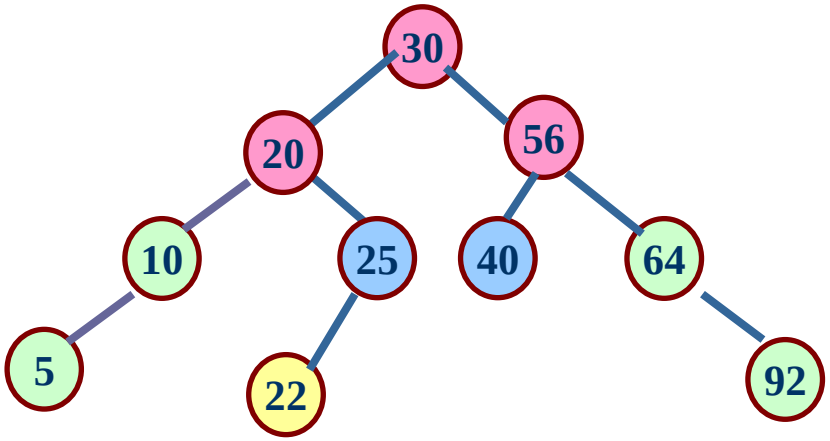


9.2 动态查找表

二. 平衡二叉树

- 举例，已知AVL树如下图，插入新数据22，求插入后的树

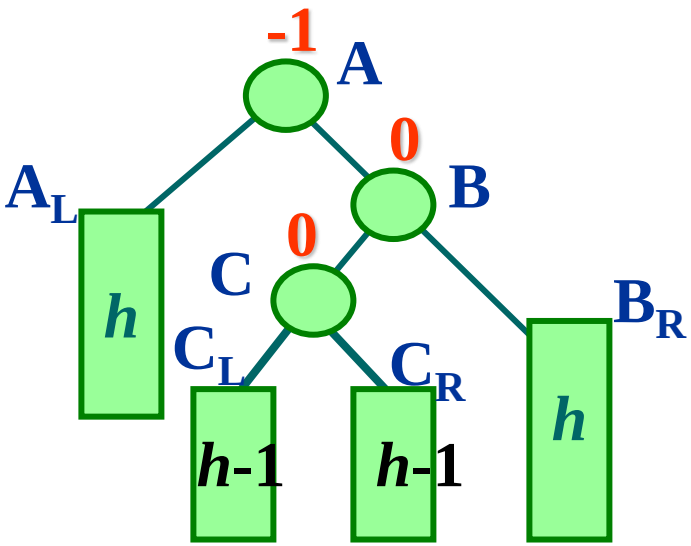


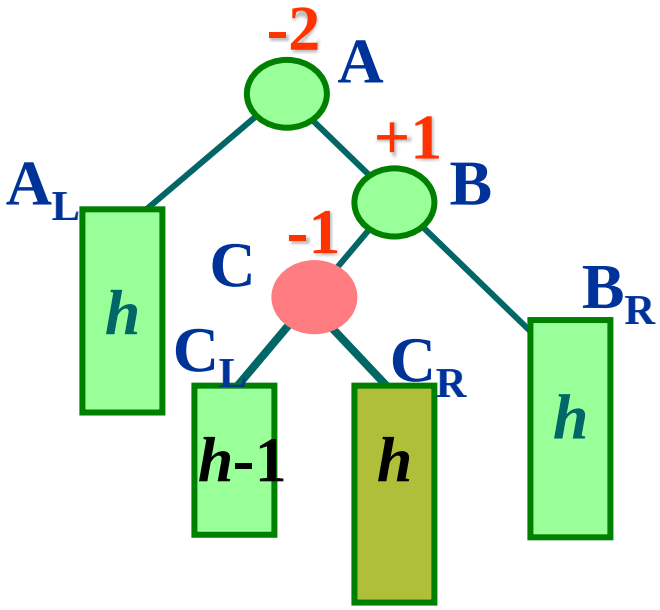


9.2 动态查找表

二. 平衡二叉树

- 先右后左双向旋转
  - 在C的子树C_L或C_R中插入新结点，该子树的高度增1。结点A的平衡因子变为-2，发生了不平衡

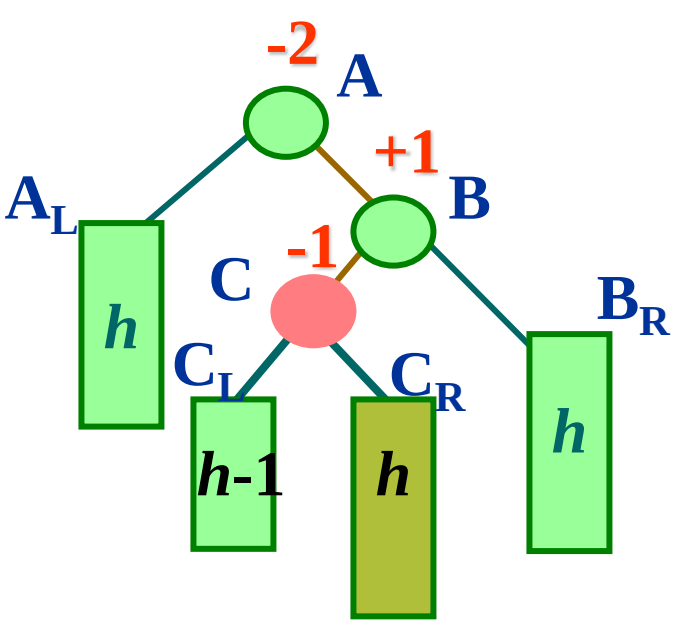


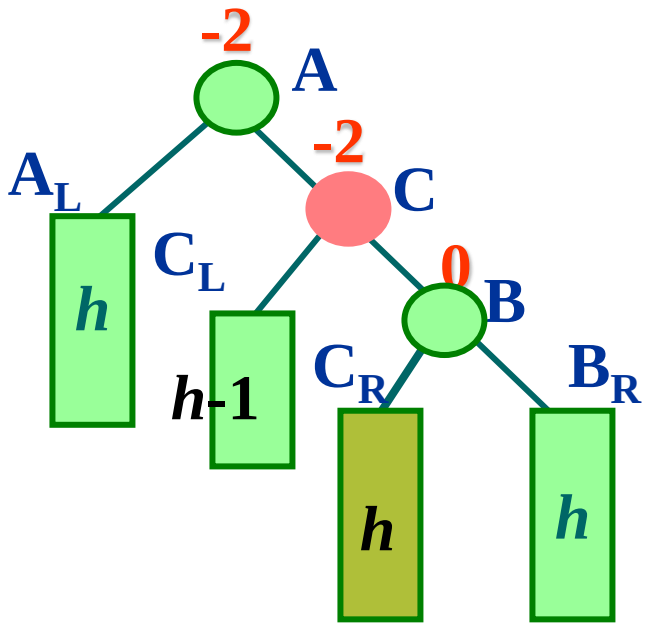


9.2 动态查找表

二. 平衡二叉树

- 先右后左双向旋转
  - 从结点A起沿插入路径选取3个结点A、C和D(“>”型)
  - 以结点B为旋转轴，作单向右旋



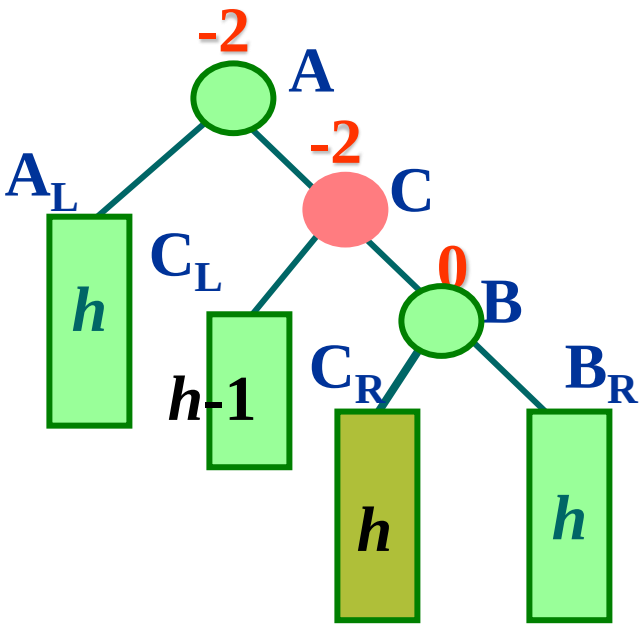


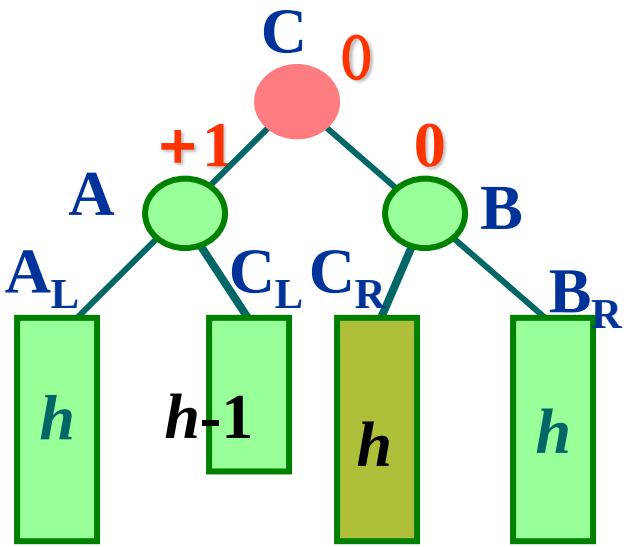
  - A右孩子变为C(C替换B)，B变为C右孩子，原C右孩子变为B左孩子

9.2 动态查找表

二. 平衡二叉树

- 先右后左双向旋转
  - 再以结点C为旋转轴，作单向左旋



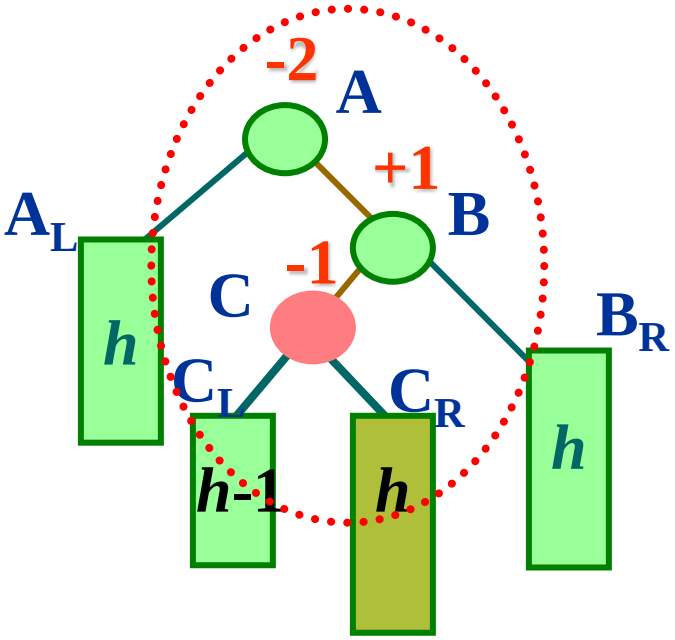


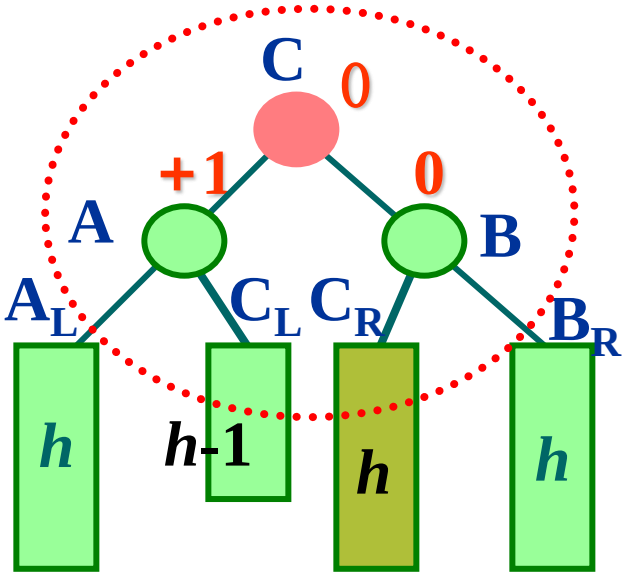
  - A变为C左孩子，原C左孩子变为A右孩子

9.2 动态查找表

二. 平衡二叉树

- 先右后左双向旋转，最终操作
  - A变为C左孩子，B变为C右孩子
  - C左孩子变为A右孩子，C右孩子变为B左孩子

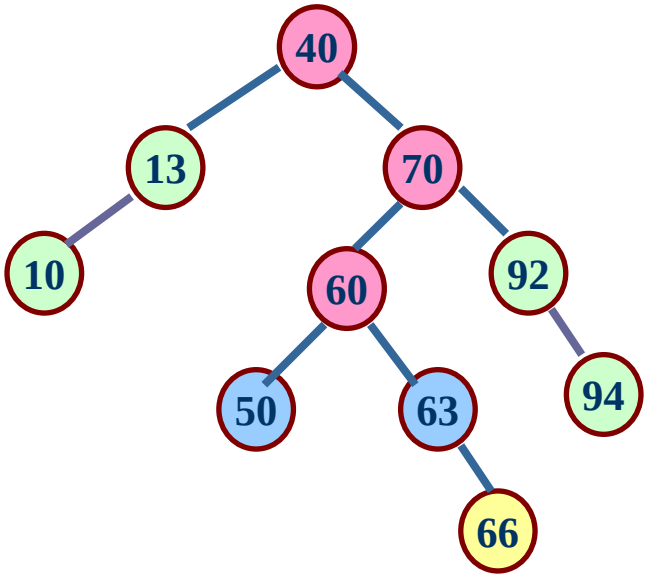


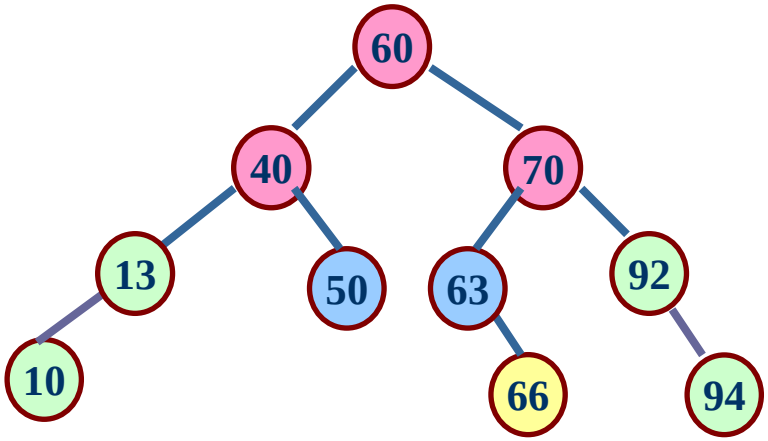


9.2 动态查找表

二. 平衡二叉树

- 举例，已知AVL树如下图，插入新数据66，求插入后的树





9.2 动态查找表

二. 平衡二叉树

平衡二叉树的删除

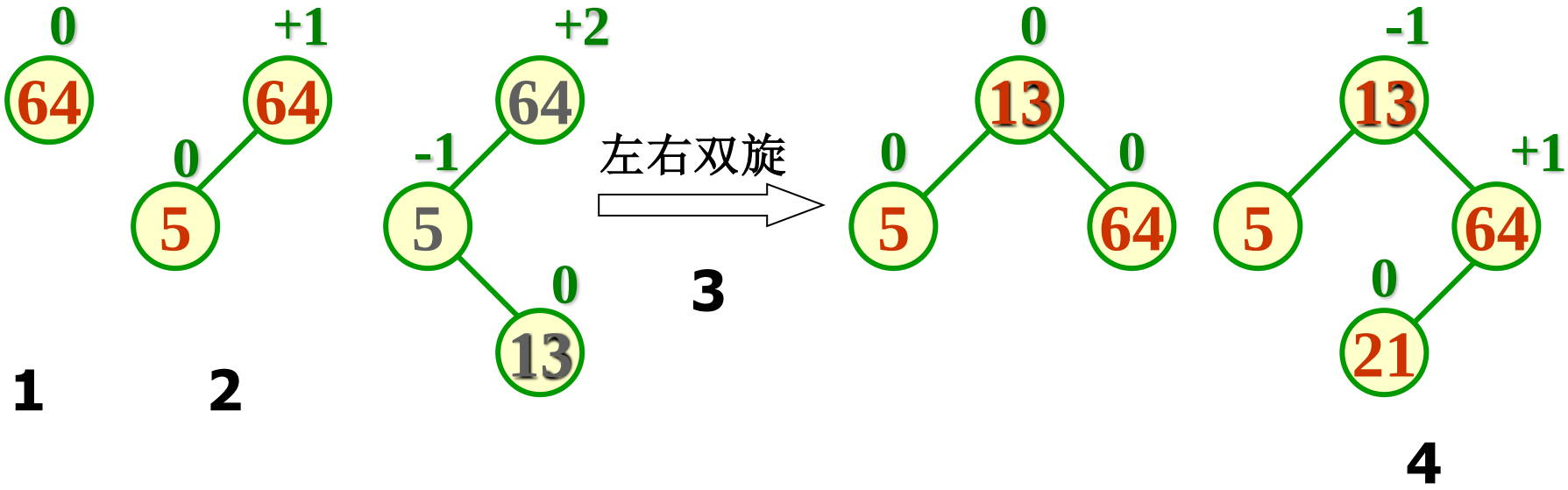
- 如果被删结点A没有孩子，则直接删除之，并作平衡化处理
- 如果被删结点A最多只有一个孩子，那么将结点A从树中删去，并将其双亲指向它的指针指向它的唯一的孩子，并作平衡化处理
- 如果被删结点A有两个子女，则用该结点的直接前驱S替代被删结点，然后对直接前驱S作删除处理(S只有一个孩子或没有孩子)

- 和二叉排序树的删除一样，增加了平衡化处理

9.2 动态查找表

二. 平衡二叉树

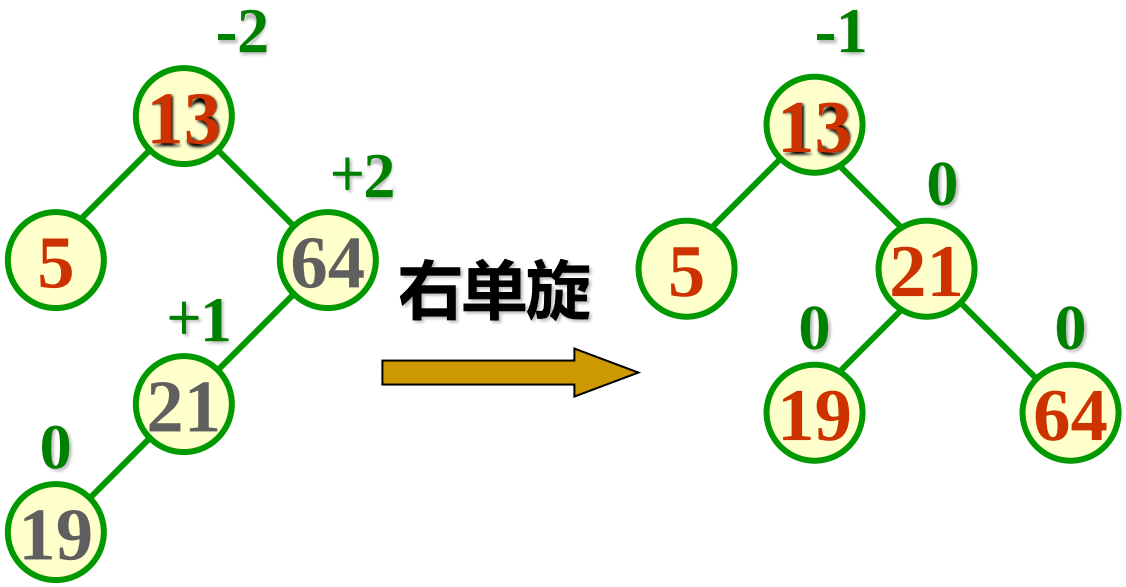
- 综合举例
  - 画出在初始为空的AVL树中依次插入 64, 5, 13, 21, 19, 80, 75, 37, 56的生长过程，并在有旋转时说出旋转的类型。



9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21*后，继续插入19, 80, 75, 37, 56时该树的生长过程，

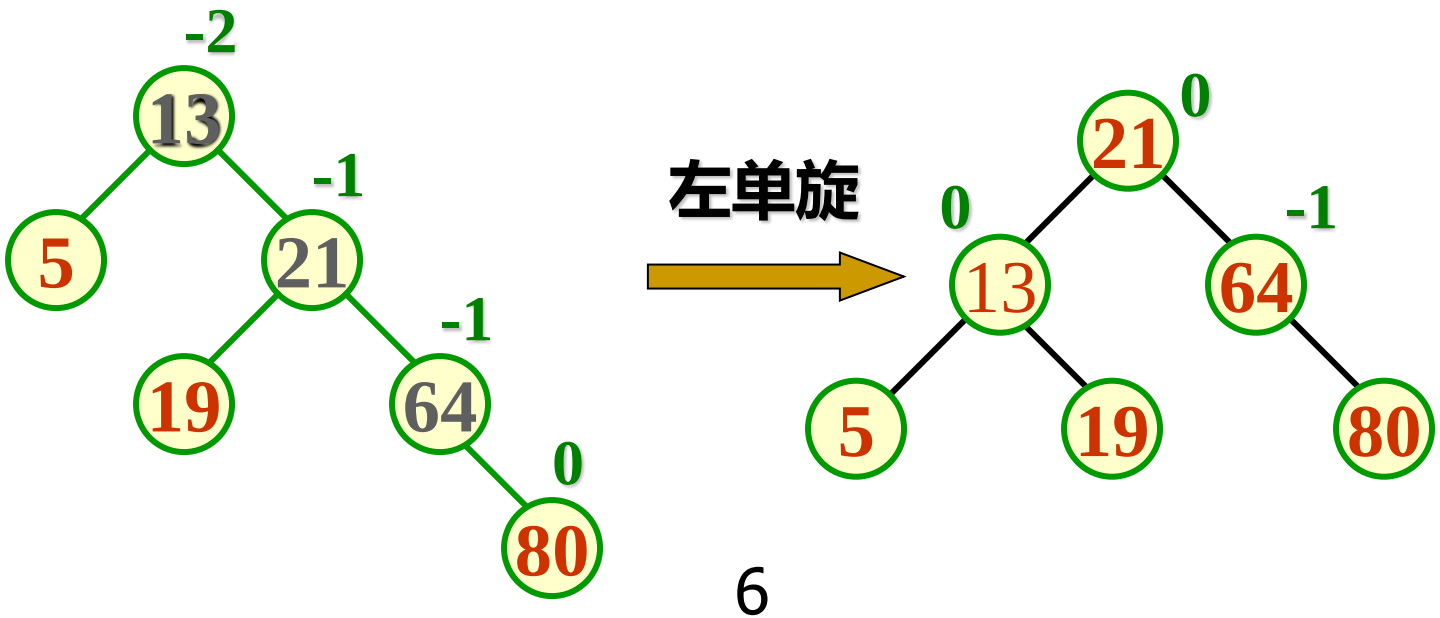


5

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19*后，继续插入80, 75, 37, 56时该树的生长过程，

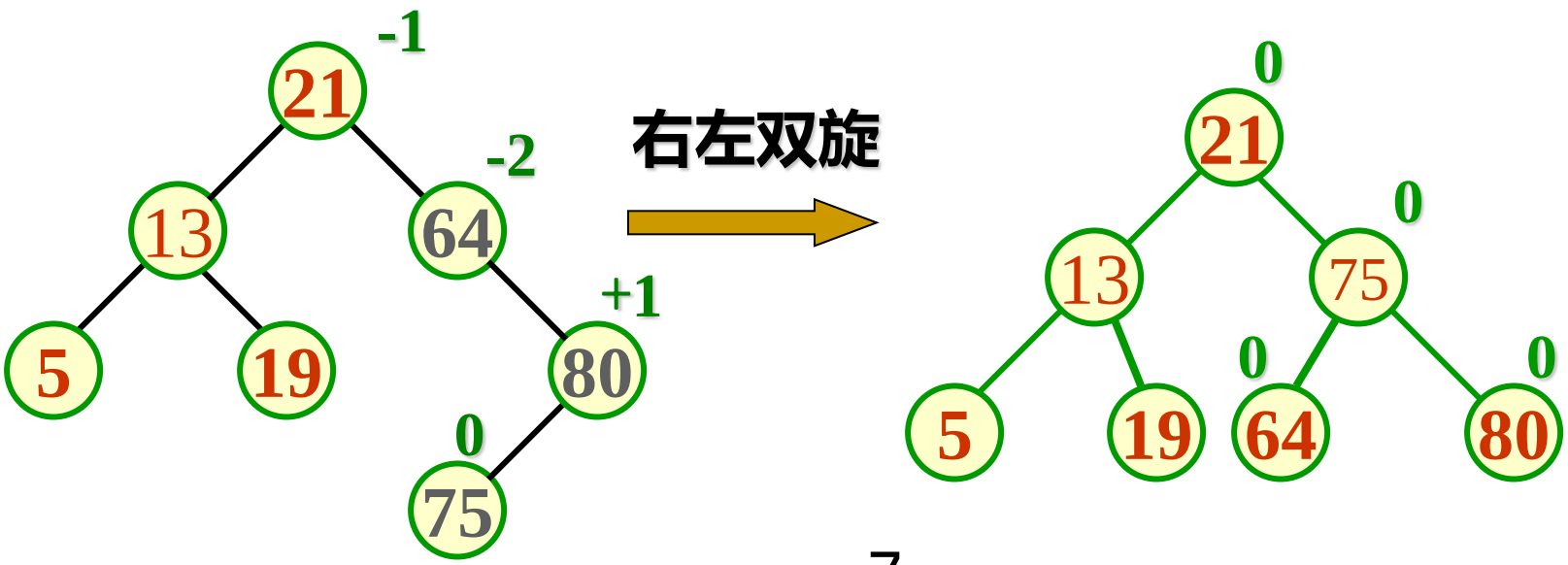


6

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80*后，继续插入75, 37, 56时该树的生长过程，

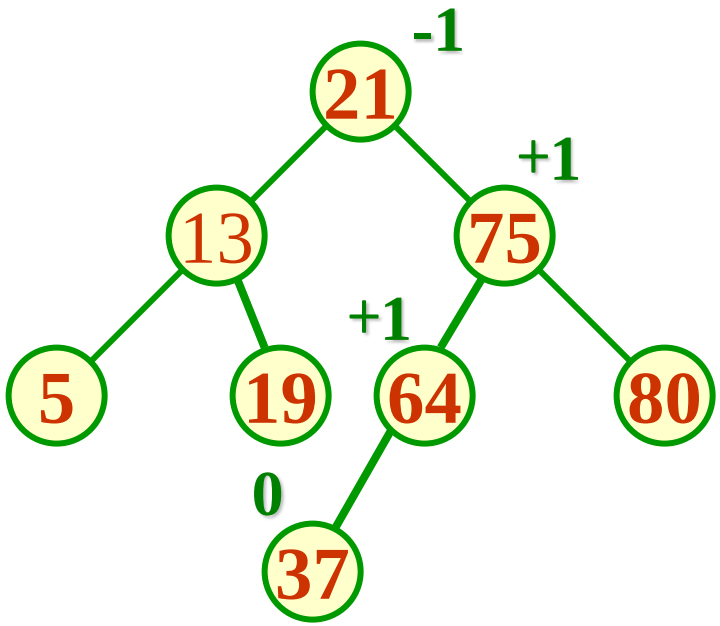


7

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80, 70*后，继续插入37, 56时该树的生长过程，

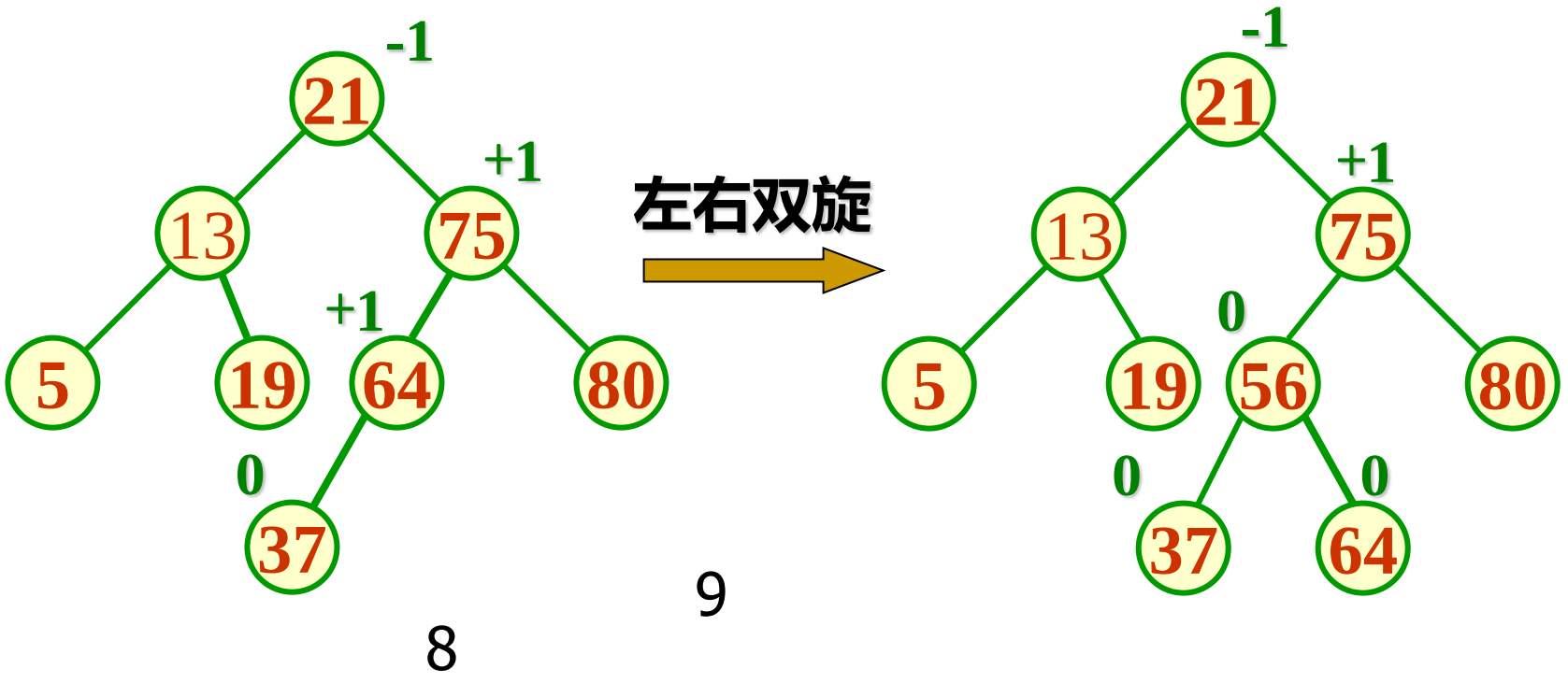


8

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80, 70, 37*后，继续插入56时该树的生长过程，



8 9

9.2 动态查找表

二. 平衡二叉树

- 平衡二叉树删除举例
  - 独立删除5、64、37，无需平衡化处理
  - 独立删除80，单向右旋处理
  - 连续删除5和13，以21-75-64做右左双旋