AVL Tree

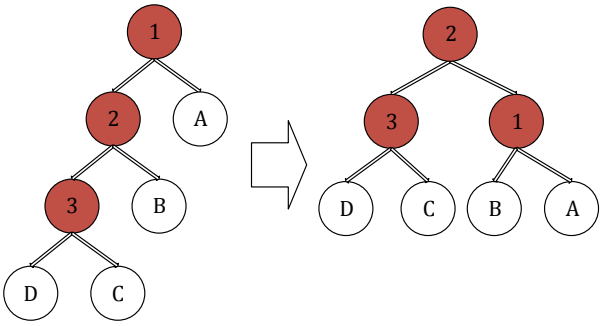
AVL 树

描述：

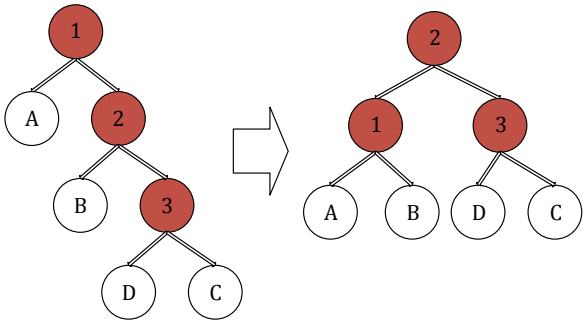
AVL 树是最早发明的一种自平衡二叉查找树，树中的任何节点的左右两个子树的高度最大差别为 1，因此也称为高度平衡树。AVL 树的查找、插入、删除操作的平均时间复杂度都是$O(log_2 N)$，AVL 树高度为$O(log_2 N)$。

为了保持树的左右子树的平衡，避免一侧过长或过短，AVL 树会对 LL（左左）、RR（右右）、LR（左右）、RL（右左）四种情况进行调整：

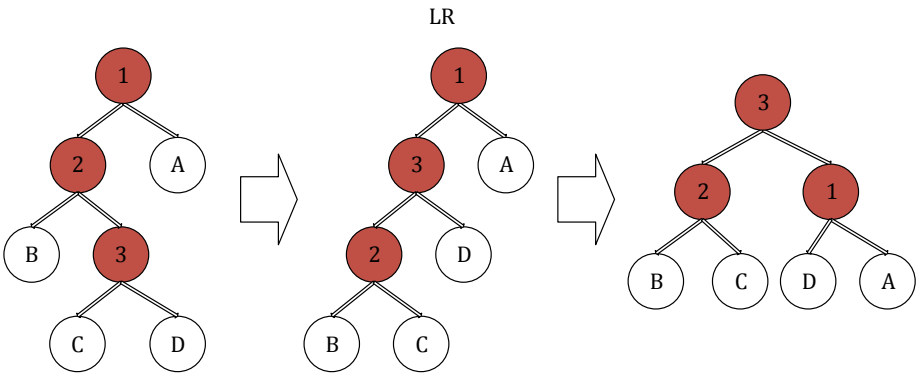
LL



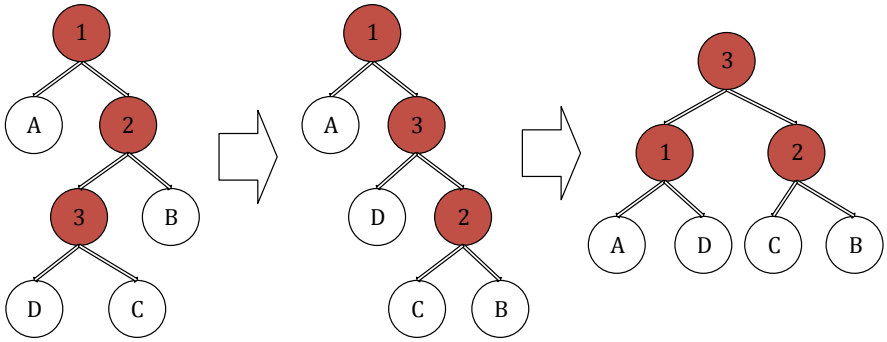
RR



LR



RL



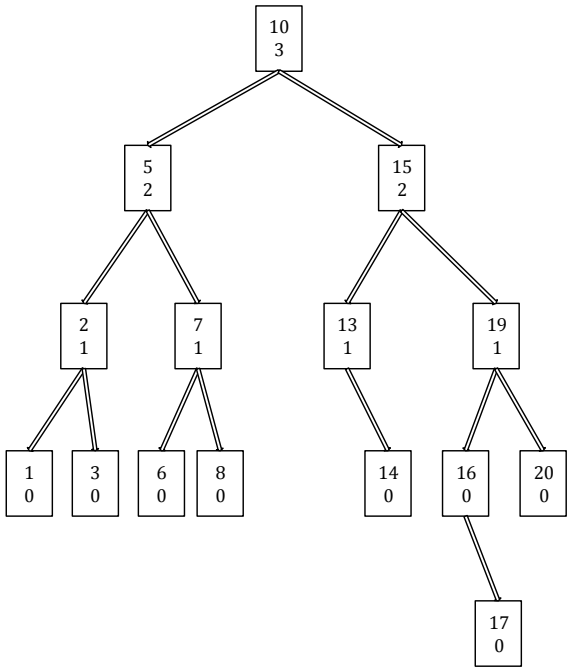
上面四种情况包含了所有从不平衡转化为平衡的步骤，其中单向右旋平衡处理 LL，单向左旋平衡处理 RR，双向旋转（先左后右）平衡处理 LR，双向旋转（先右后左）平衡处理 RL。

这四种操作既能够平衡左右子树的高度，还能够保持树的有序性。即平衡后的树的左子树中所有节点仍然小于（或大于）树的根节点，而右子树中所有节点仍然大于（或小于）树的根节点。

AVL 树的每个节点都有一个高度值$depth$，树的平衡因子为$balance_factor = left_tree.depth - right_tree.depth$，即左右子树的深度之差。当一个节点的$|balance_factor| \leq 1$时该子树平衡；当$|balance_factor| \geq 2$时该子树不平衡。

将空节点的高度值视作-1，一个节点的高度值为$node.depth = max(node.left_child.depth, node.right_child.depth) + 1$。上面 LL、RR、LR 和 RL 四种操作，都会将其节点 1 的高度值减 2，其余节点的高度值都不变。

对于下面这个 AVL 树，每个节点中上面的数字是节点下标号，下面的数字是该节点的高度值$depth$。将节点 18 插入下面的 AVL 树：



(1) 从根节点开始，将节点 18 与节点 10 比较，有$18 > 10$，因此把节点 18 插入节点 10 的右子树；

(2) 将节点 18 与节点 15 比较，有$18 > 15$，因此把节点 18 插入节点 15 的右子树；

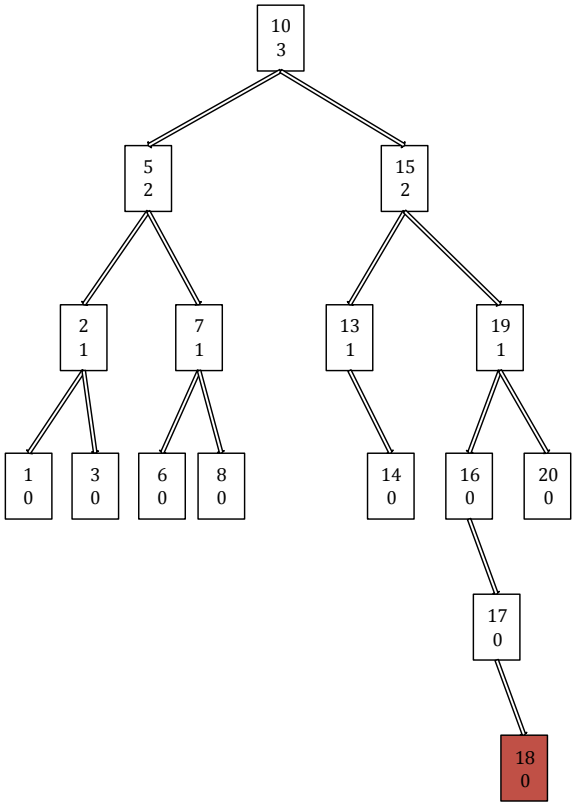
(3) 将节点 18 与节点 19 比较，有$18 < 19$，因此把节点 18 插入节点 19 的左子树；

(4) 将节点 18 与节点 16 比较，有$18 > 16$，因此把节点 18 插入节点 16 的右子树；

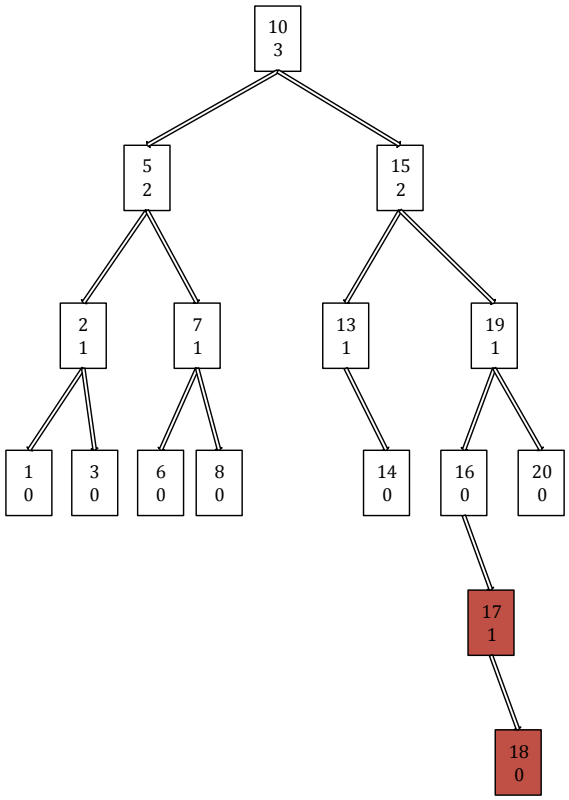
(5) 将节点 18 与节点 17 比较，有$18 > 17$，因此把节点 18 插入节点 17 的右子树，节点 17 的右孩子节点为空，因此节点 18 成为节点 17 的右孩子节点；

然后从节点 18 开始，向上依次更新所有节点的高度值，若新的高度值不满足 AVL 树的平衡性，则进行旋转操作：

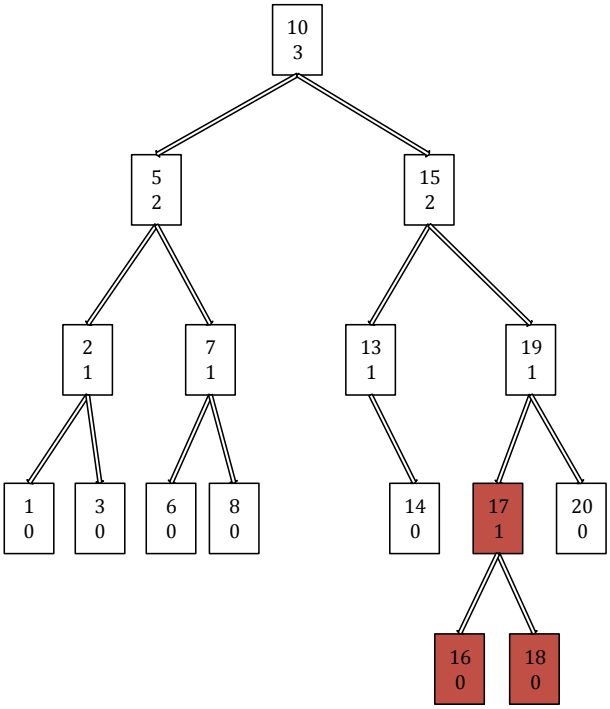
(1) 节点 18 为叶子节点，因此高度值为 0；



(2) 节点 17 的平衡因子为 $node_{17}.balance\_factor = |node_{nil}.depth - node_{18}.depth| = |-1-0| = 1$，不需要旋转，高度值更新为 $node_{17}.depth = max(node_{17}.left\_child.depth, node_{17}.right\_child.depth) + 1 = max(-1, 0) + 1 = 1$；

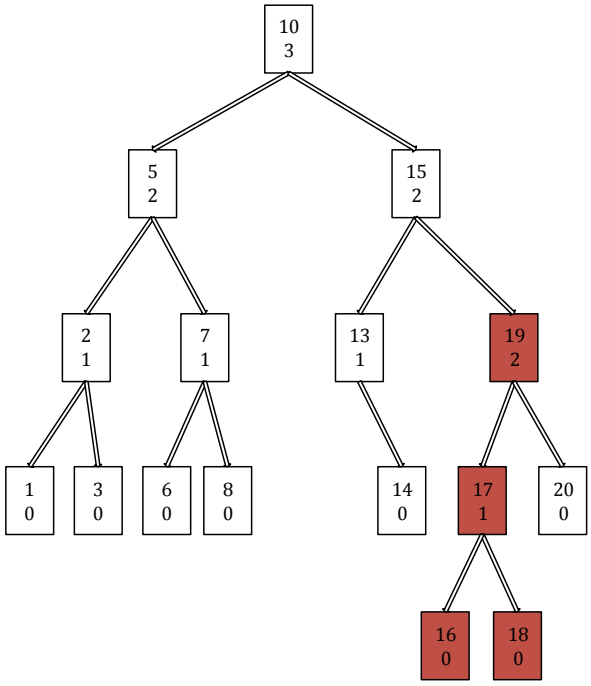


(3) 节点 16 的平衡因子为 $node_{16}.balance\_factor = |node_{nil}.depth - node_{17}.depth| = |-1-1| = 2$，高度值更新为 $node_{16}.depth = max(node_{16}.left\_child.depth, node_{16}.right\_child.depth) + 1 = max(-1,1) + 1 = 2$，由于节点 16 的平衡因子超过 1，需要进行 RR 操作，旋转后节点 16 的高度值减 2；

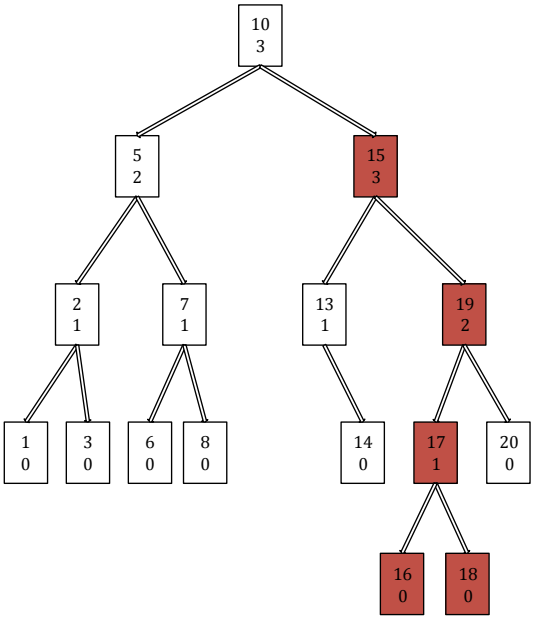


(4) 节点 19 的平衡因子为$node_{19}.balance_factor = |node_{17}.depth - node_{20}.depth| = |1-0| = 1$，高度值更新为 $node_{19}.depth =$

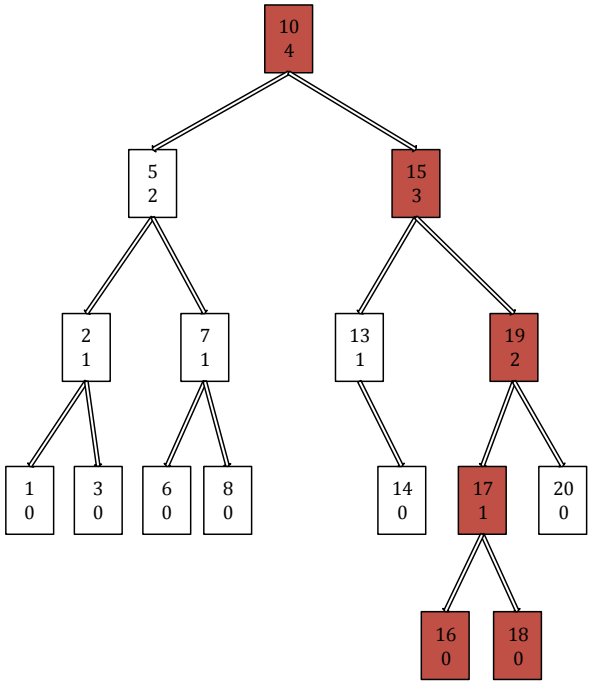
$max(node_{19}.left\_child.depth, node_{19}.right\_child.depth) + 1 = max(1,0) + 1 = 2$；



(5) 节点 15 的平衡因子为$node_{15}.balance_factor = |node_{13}.depth - node_{19}.depth| = |1 - 2| = 1$，高度值更新为$node_{15}.depth = max(node_{15}.left_child.depth, node_{15}.right_child.depth) + 1 = max(1,2) + 1 = 3$；



(6) 节点 10 的平衡因子为$node_{10}.balance_factor = |node_{5}.depth - node_{15}.depth| = |2 - 3| = 1$，高度值更新为$node_{10}.depth = max(node_{10}.left_child.depth, node_{10}.right_child.depth) + 1 = max(2,3) + 1 = 4$；


10
4
5
2
15
3
2
1
7
1
13
1
19
2
1
0
3
0
6
0
8
0
14
0
17
1
20
0
16
0
18
0