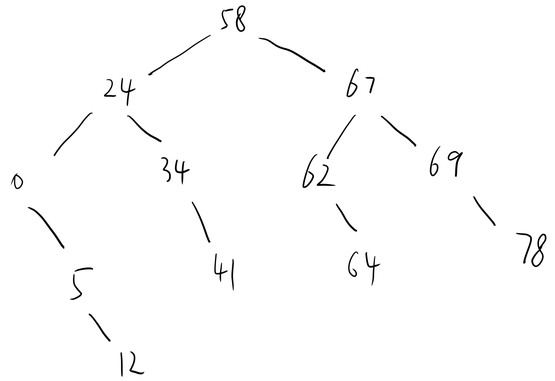
BST



插入：最坏的情况无非是长这样 / 是个数字就能插进来

删除：1. 没有孩子的节点　把父节点的地址域 nullptr　> 可以写到一个里面

2. 有一个孩子的节点　孩子写入父节点地址域

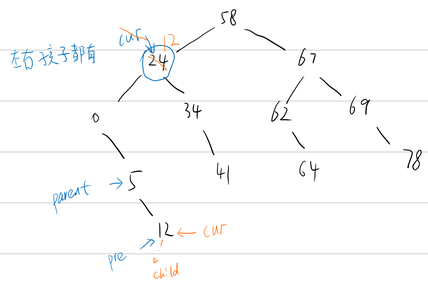
3. 有两个孩子

找待删除节点的前驱节点（或者后继节点）用前驱或者后继节点的值把待删除节点的值覆盖掉，然后直接删除前驱或者后继节点就可以了

前驱节点：当前节点左子树中值最大的节点

后继节点：当前节点右子树中值最小的节点

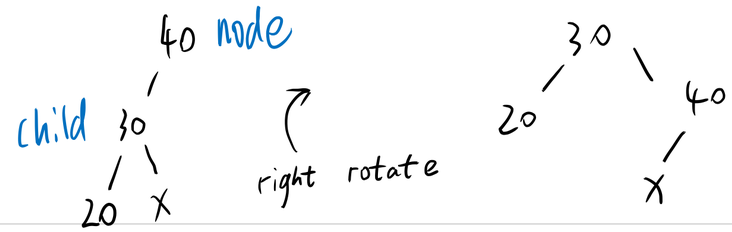
前驱/后继节点：最多有一个孩子，或者没有孩子，回到情况1/2



AVL

节点失衡的原因

1. 左孩子的左子树太高了



right rotate

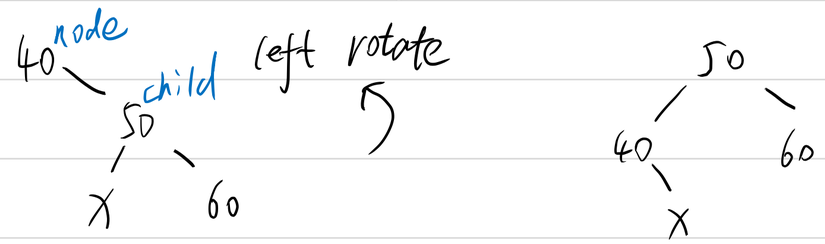
child = node -> left

node -> left = child -> right

child -> right = node

node 和 child 节点的高度值都要更新

2. 右孩子的右子树太高了



left rotate

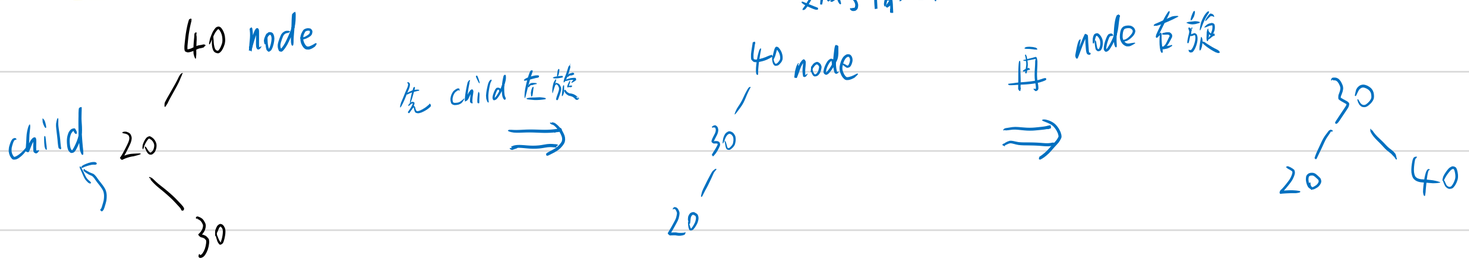
child = node -> right

node -> right = child -> left

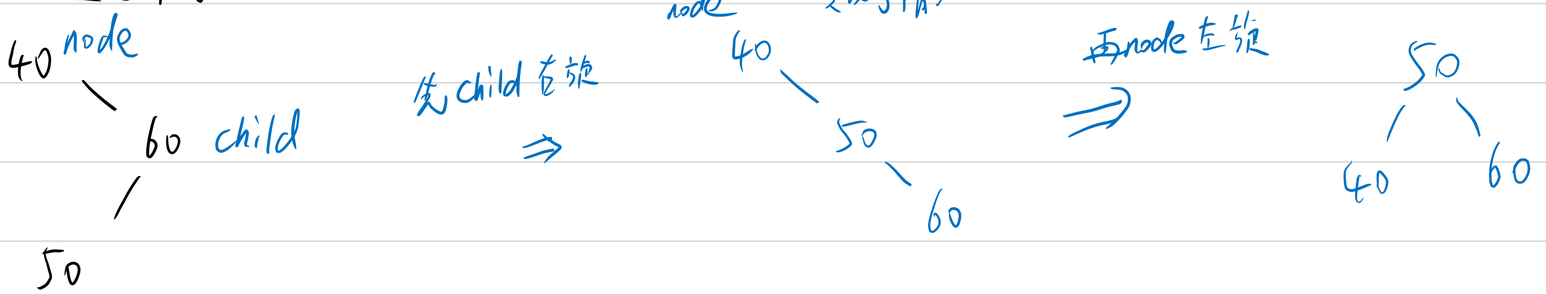
child -> left = node

node 和 child 节点的高度值都要更新

3. 左孩子的右子树太高了



4. 右孩子的左子树太高



红黑树

3 O(logn)
删除的效率比AVL高一点

不是一颗平衡二叉树，节点左右子树的高度差，长的不超过短的2倍

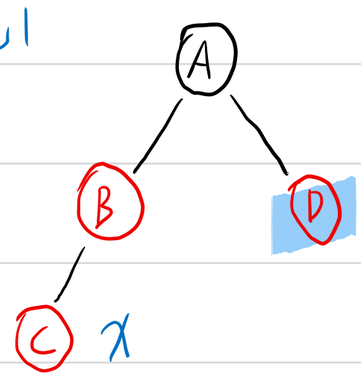
特点：

1. 树的每一个节点都有颜色，非黑即红
2. null是黑色
3. root是黑色
4. 不能出现连续的红色节点
5. 从root根节点到每一个叶子节点的路径上，黑色节点的数量是相同的

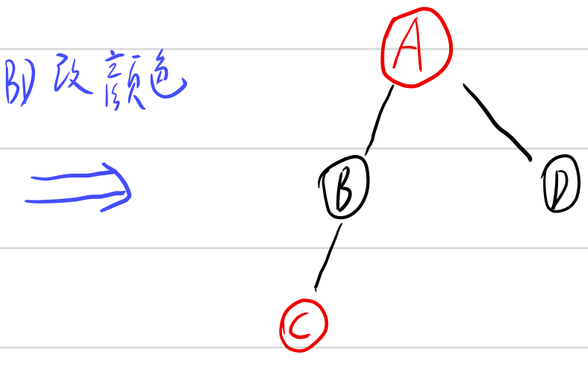
插入

1. 空树　　节点：黑色
2. 非空　　插入叶子节点：红色
   如果父节点黑色，插入完成
   否则 调整如下：

情况1



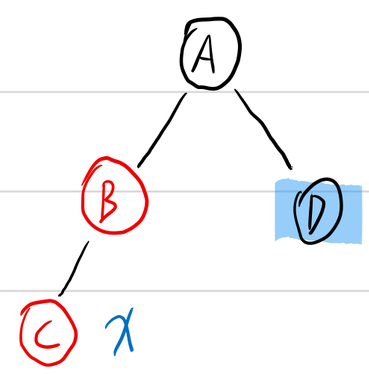
ABD改颜色



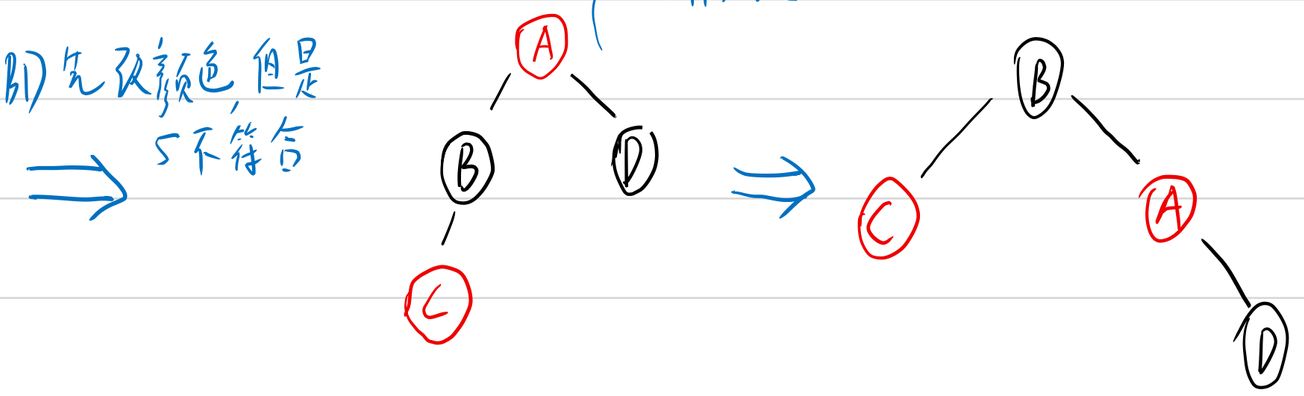
X=A　　继续调整

root.color = BLACK

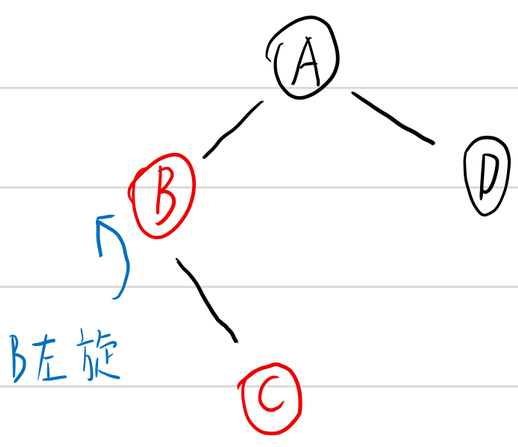
情况2

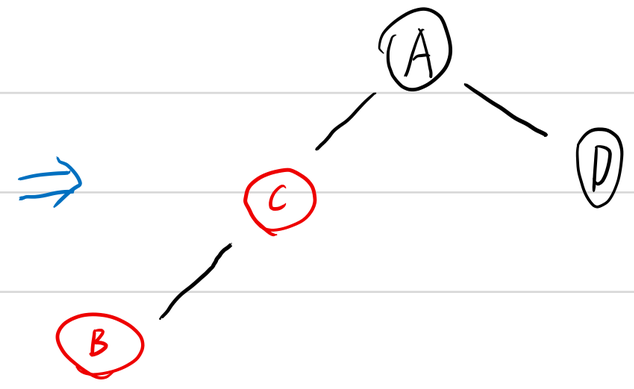


AB均先改颜色，但是5不符合



情况3





情况2

⇒ ...