红黑树 (RBT)

1. 为什么要使用红黑树?

查找
- 静态查找表
  - 顺序查找 — O(n)
  - 二分查找：
    - 时：O(log₂n)
    - 缺：有序表
  - 索引查找
- 动态查找表 ⇑
  - i> 无序序列查找
  - ii> 边查边插.
  - BST — 二叉排序树
    - — 无法控制树的高度
    - 在相同的时间复杂度下无法控制地较好水平
  - AVL — 平衡二叉树 (不实用)
  - RBT — 红黑树

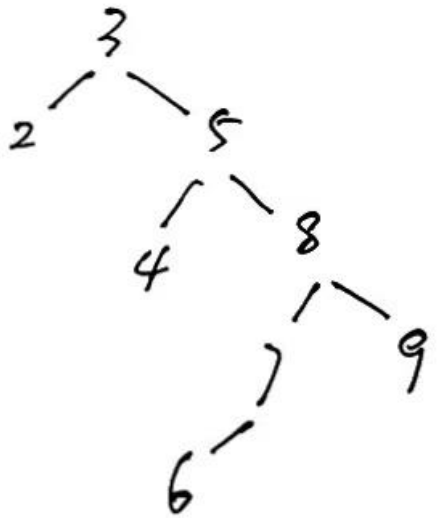
① BST 无法控制高度
— 无法控制查找效率

② AVL — 平均、最坏，都 O(log n)
\ 插入、删除 要保持 时 效率低下、(要平衡)

③ RBT — 平衡

平衡 没有 AVL 好
但插入的时候
时间复杂度 O(log₂n)
删除，
使 RBT 插入的时候简单

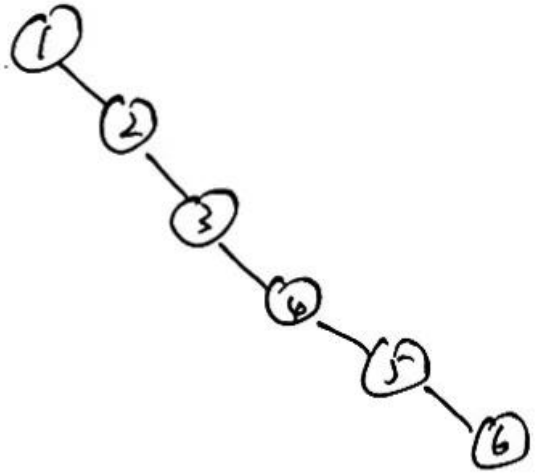
BST: {3, 2, 5, 8, 4, 7, 6, 9}.



AVG: n个节点 O(log2n)

最坏: 单枝BST. O(n).

{1, 2, 3, 4, 5, 6}. ⇒



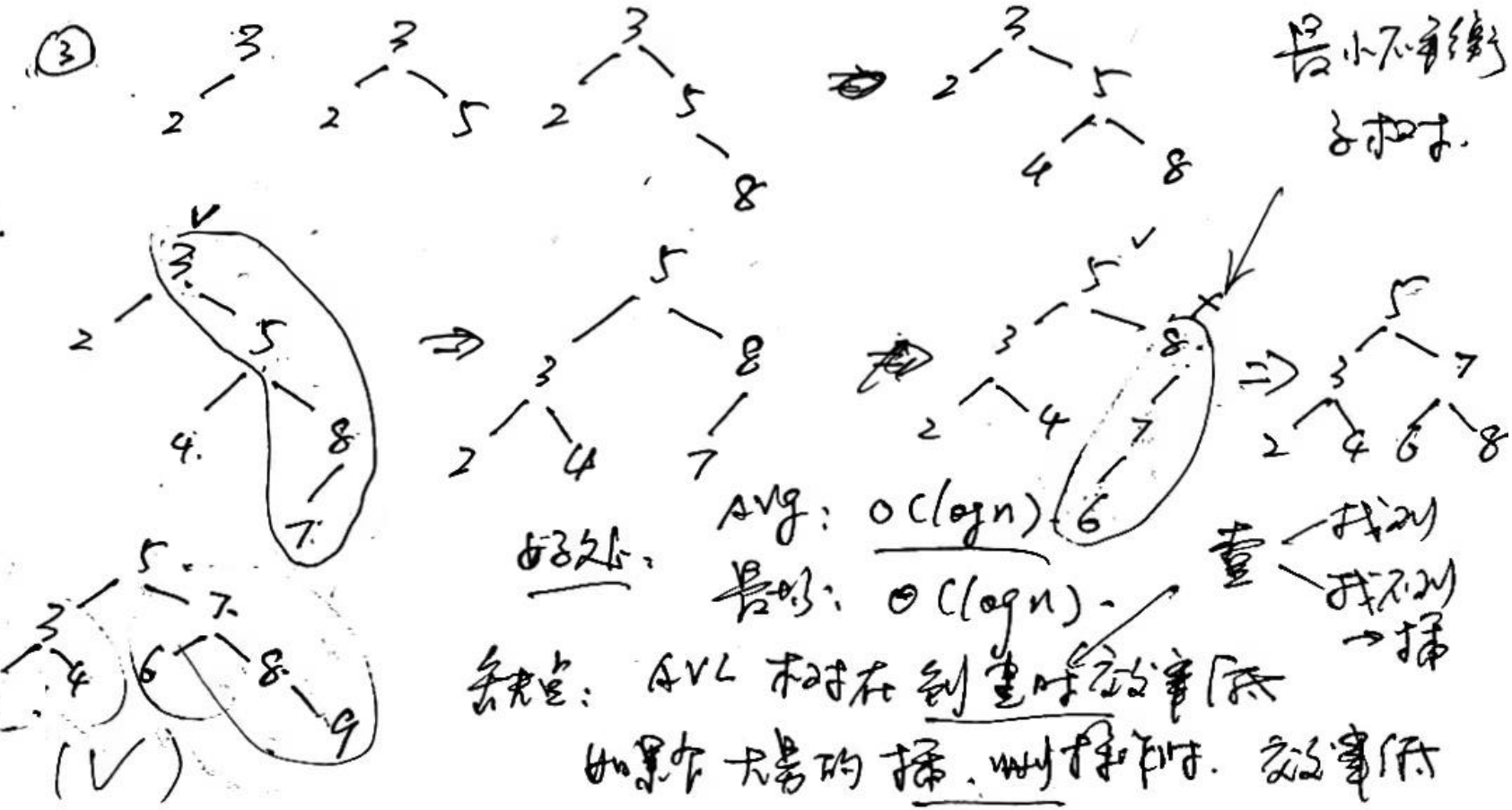
无法控制的方法

⇒ 在相同的n个节点情况下无法控制查找次数的性能.

AVL：左、右子树平衡了

平衡因子 = |左子树高度 − 右子树高度| ∈ {0, 1}

序列：{3, 2, 5, 8, 4, 7, 6, 9}



好处：

AVG：O(logn)

最坏：O(logn)

查 找到 找不到→插

缺点：AVL 树在创建时的事件

如果大量的插、删事件时，效率低

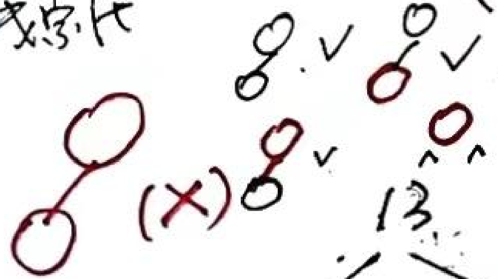
红黑树(RBT) — 自平衡二叉搜索树(BST)

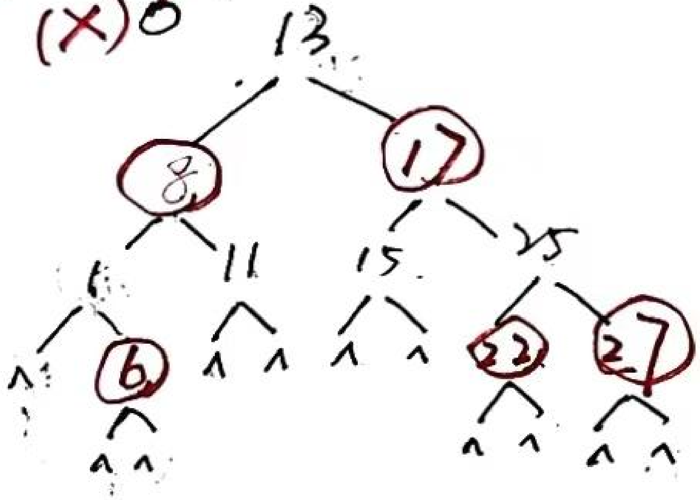
☆ 含n个结点的二叉链树 有n+1个NULL域 — 通过对结点进行着色和旋转, RBT很容易保持平衡

着色: 红色 黑色

二叉树线索化

结点:
- 颜色: 红色、黑色
- 叶子结点: 树中为NULL的结点 — 外部结点
- 内部结点: 包括根结点和叶子结点在内的非NULL结点 (除了外部结点(叶结点))





i> 每个结点要么是红色, 要么是黑色

ii> 根结点是黑色, 所有叶结点是黑色

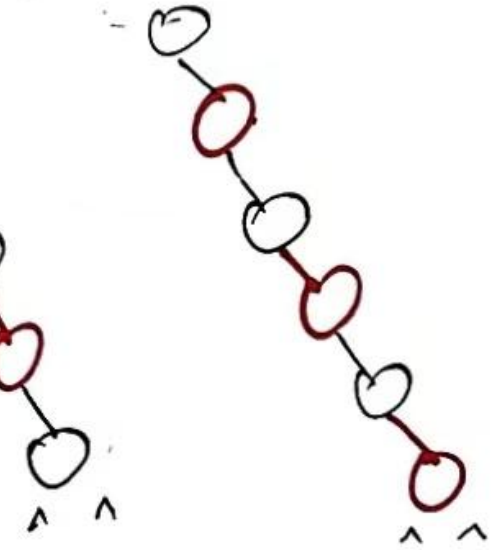
iii> 若一个结点是红色, 其子女均为黑色 — 任何路径都不会出现连续的红结点

iv>

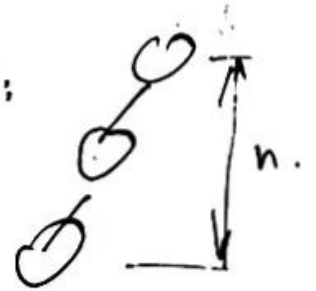
iv> 对于[每个]内部结点(x). 从它发出到所有叶结点的两个路径上包含相同的黑色结点(不包含自己)

结论: 从根结点到叶结点的所有路径中. 最长路径不会超过最短路径的二倍

最好:



最坏:

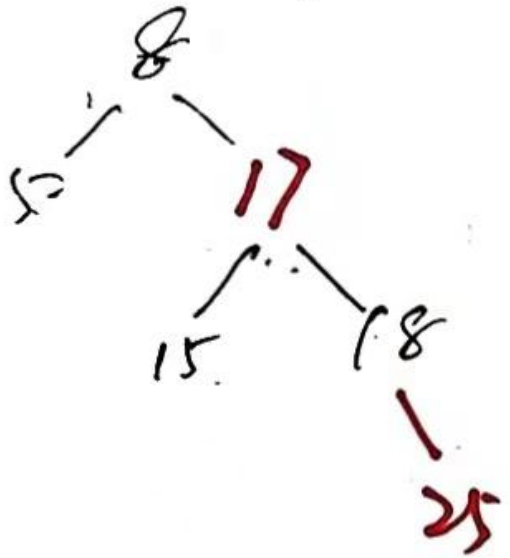


最长路径 ≤ 2d 最短路径.

① 不需要像AVL的平衡性.

只需要像操作对左右子树之差在一层以内.

☆ 红黑树最大高度：$2\log_2(n+1)$.



$h(8)=4$

$h(17)=3$

$h(18)=2$

$h(25)=1$

最好查找次数：$2\log_2(n+1)$.

平均时间复杂度：$O(\log n)$.

最坏时间复杂度：$O(\log_2 n)$

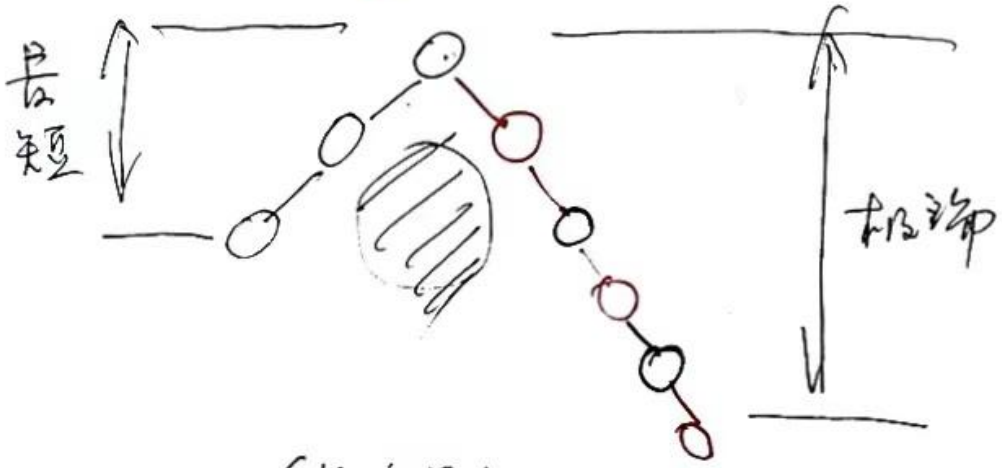
性质4 从根到叶结点的最长路径不大于最短路径的二倍

黑色高度(Black Height): 从任何一个顶点出发(不含该结点)到任一叶结点(本结点可能也是叶结点)的所经过的黑色结点个数.(要算叶结点)

bh

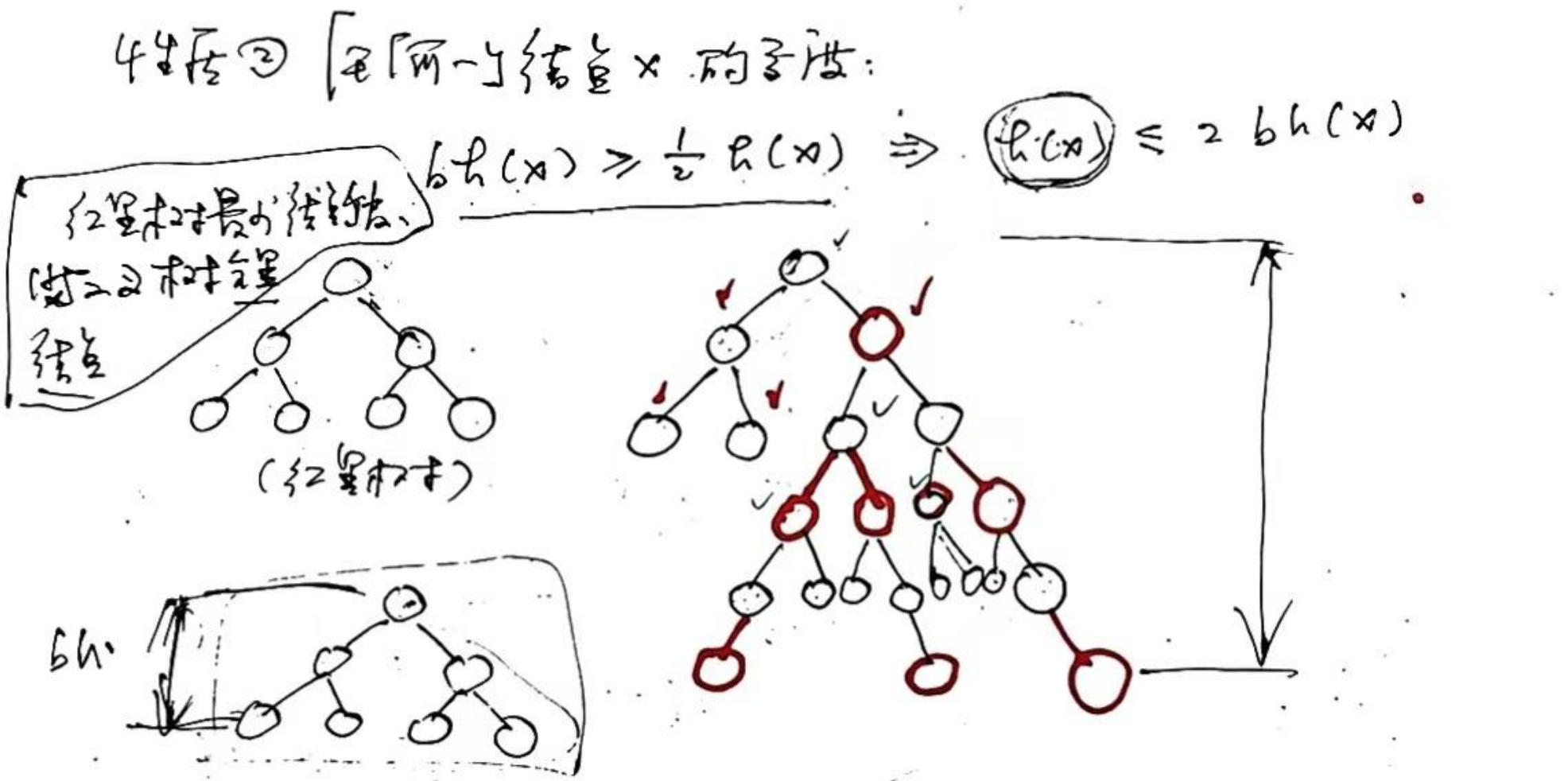
☆ 根结点的黑高即红黑树的黑高

☆ 对于每个结点,从该结点到任一叶结点的简单路径上,所含黑色结点数量相同.即黑高相同.



(非红黑树)

(某个红黑树的两条路径)


性质③ 在所有结点 x 的子孙:
bh(x) ≥ 1/2 h(x) ⇒ h(x) ≤ 2 bh(x)
红黑树最长路径就是
随2叉树结点
(红黑树)
bh

性质③ 从任何一个结点x的向叶结点不少于 $2^{bh(x)}-1$.

☆原因：从任何一个结点的子树的最少结点个数，一棵全是黑的满二叉树。(全是黑色)

$$2^{bh(x)}-1 \le n$$

$$bh(x) \le \log_2(n+1)$$

$$bh(x) \ge \frac{1}{2}h(x)$$

$$\Rightarrow \frac{1}{2}h(x) \le \log_2(n+1)$$

$$\Rightarrow h(x) \le 2\log_2(n+1)$$

最大深度.

☆有n个结点的红黑树的树高 $\le 2\log_2(n+1)$

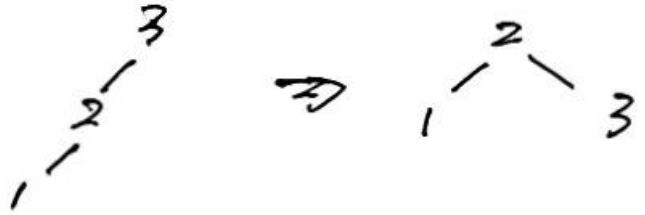
$\Rightarrow$ 时间复杂度 $O(2\log_2(n+1)) \Rightarrow O(\log_2(n))$

红黑树的插入 < 着色
旋转：LL LR RL RR

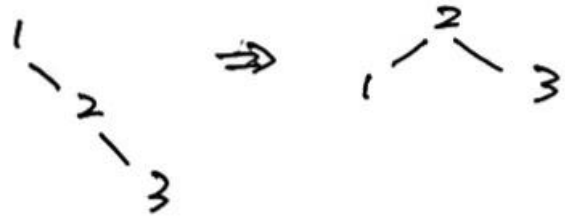
① 复习 AVL 旋转

基本结构：

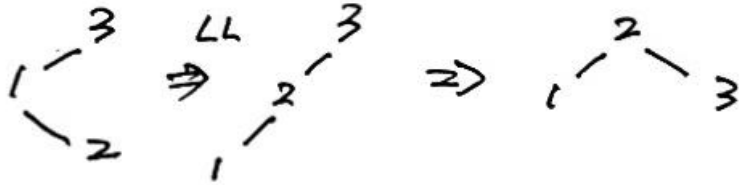
LL



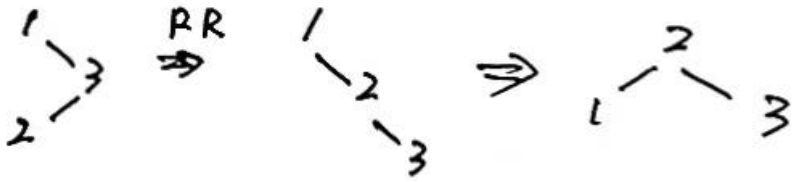
RR

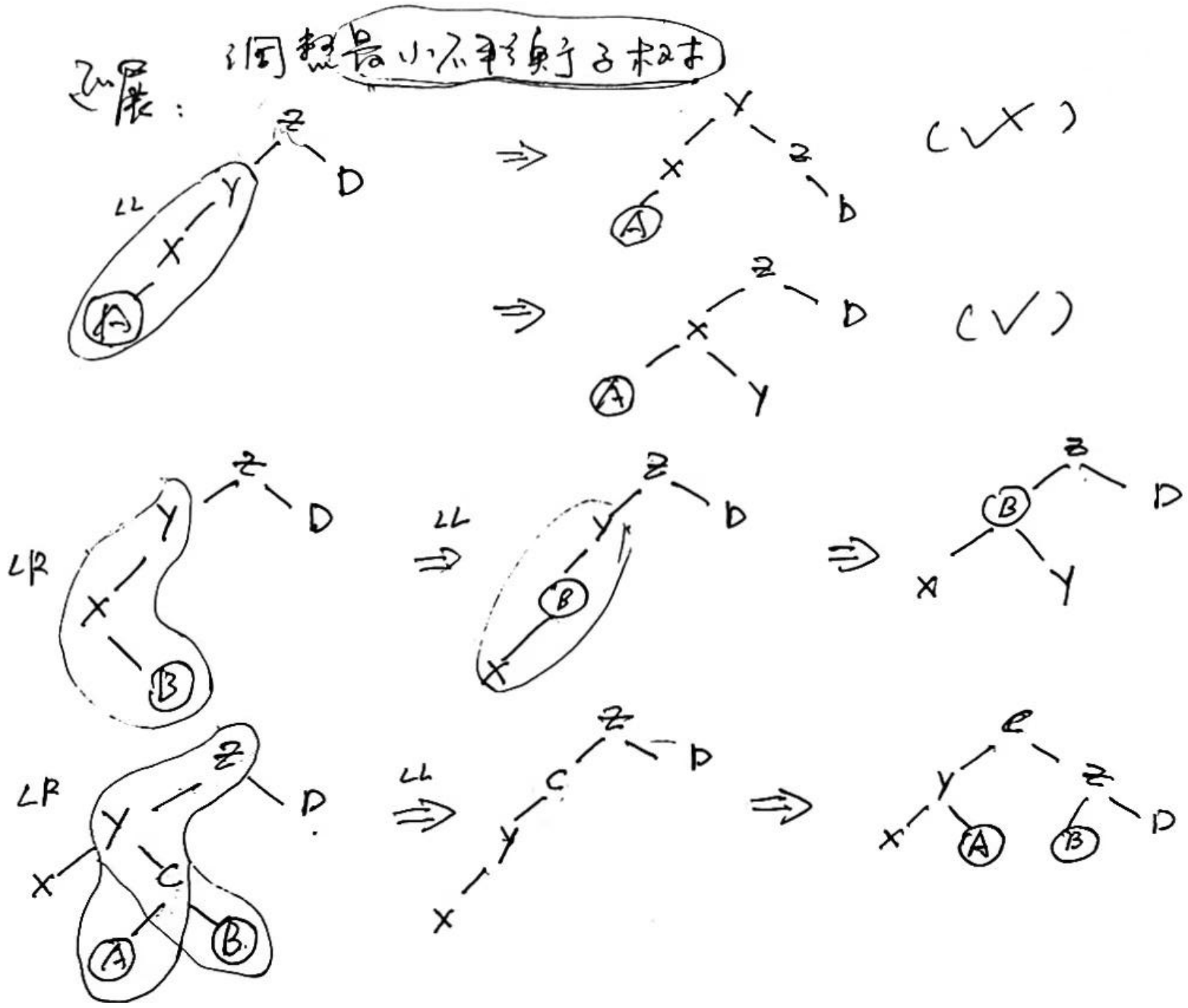


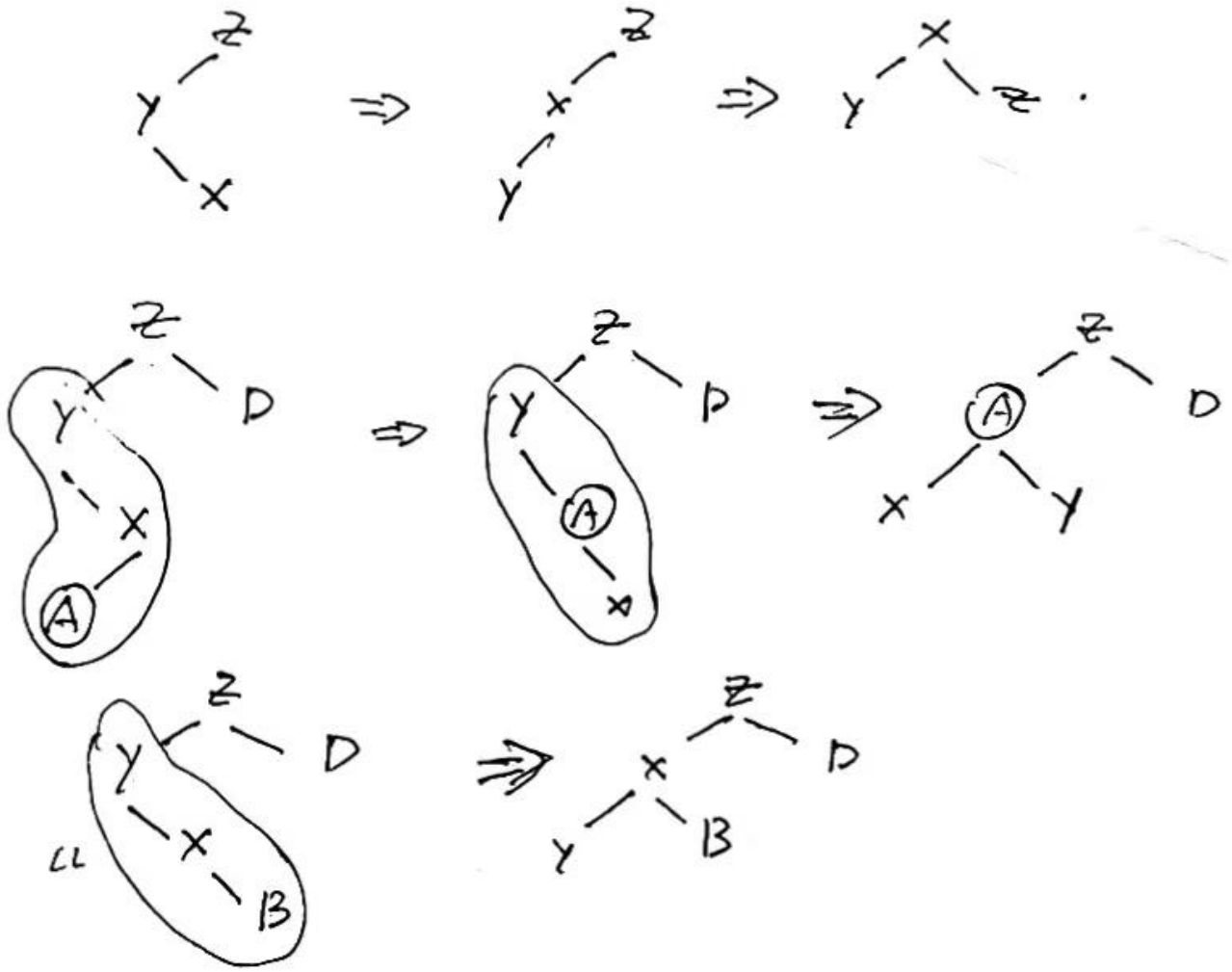
LR



RL




调整最小不平衡子树
LL
LR
LR
LL
LL
(✗)
(✓)


Z
Y
X
Z
X
Y
X
Y
Z
Z
Y
D
X
A
Z
Y
D
A
X
Z
A
D
X
Y
Z
Y
D
X
LL
B
Z
X
D
Y
B

RBT：① 根是黑色

② 待插结点都是红色 → 调整（为什么待插结点不是黑结点）

调整 < 不要出现连续的红色结点
（对是RBT的平衡）— 旋转

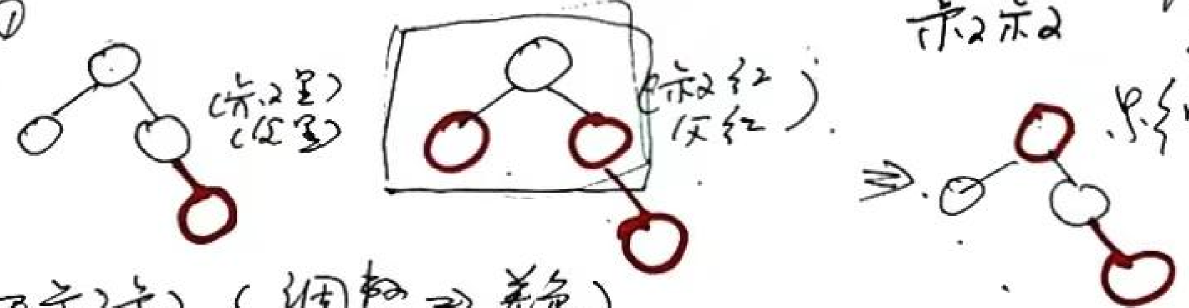
父黑 —— 黑 红
不影响黑高

☆ 着色后 若根是红的 根→黑

A> 插入根 → 变色（也可能不变色）

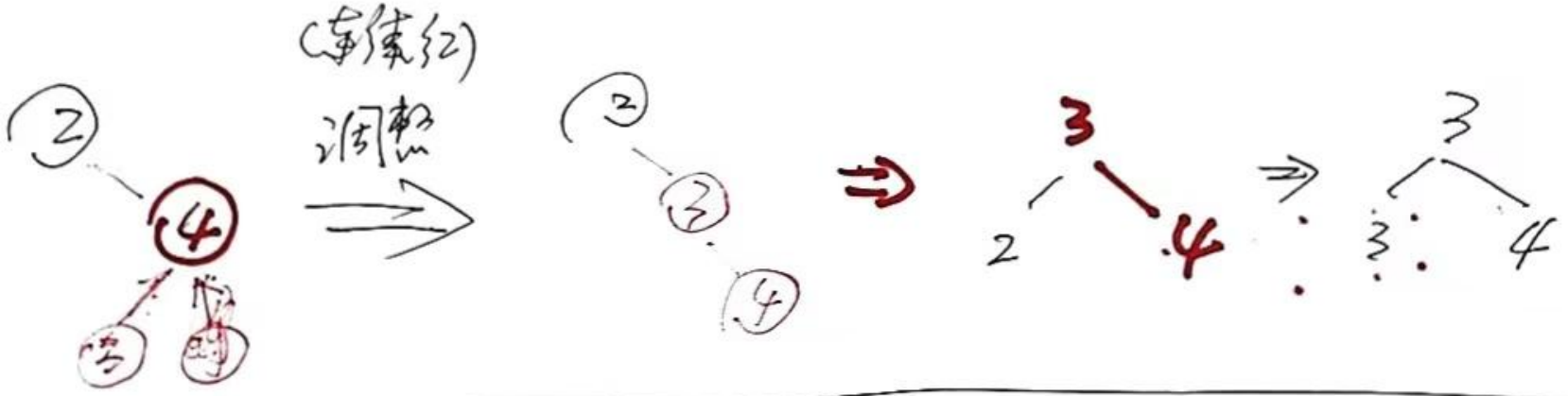
B> 插入非根 < 待插结点是红叶子
1> 看双亲结点的兄弟结点（叔叔）< 有：变换（着色）；没有：不影响黑高

情况①

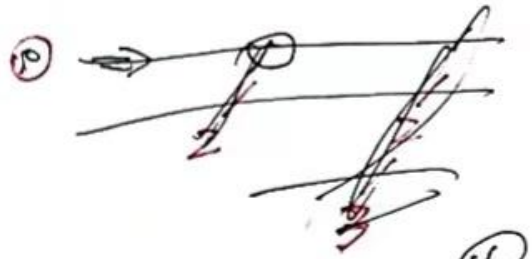


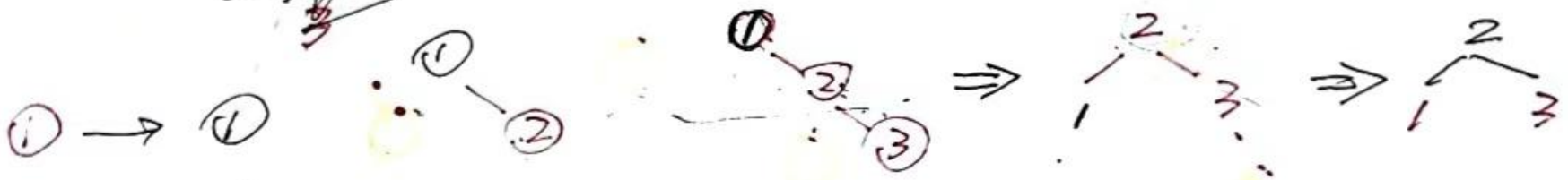
情况② 无叔叔（调整 => 着色）

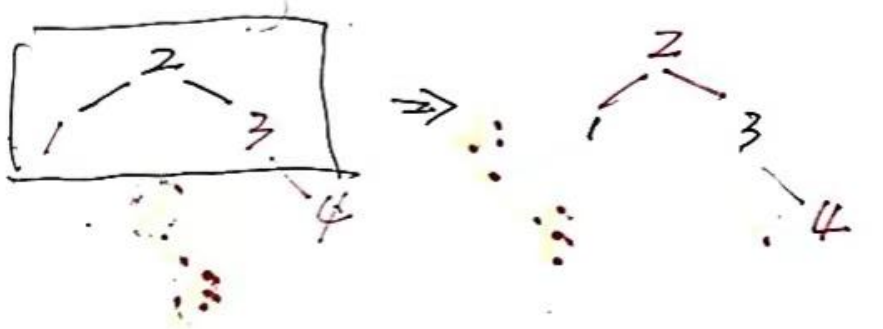
(×) 待插结点的父结点不可能为红。

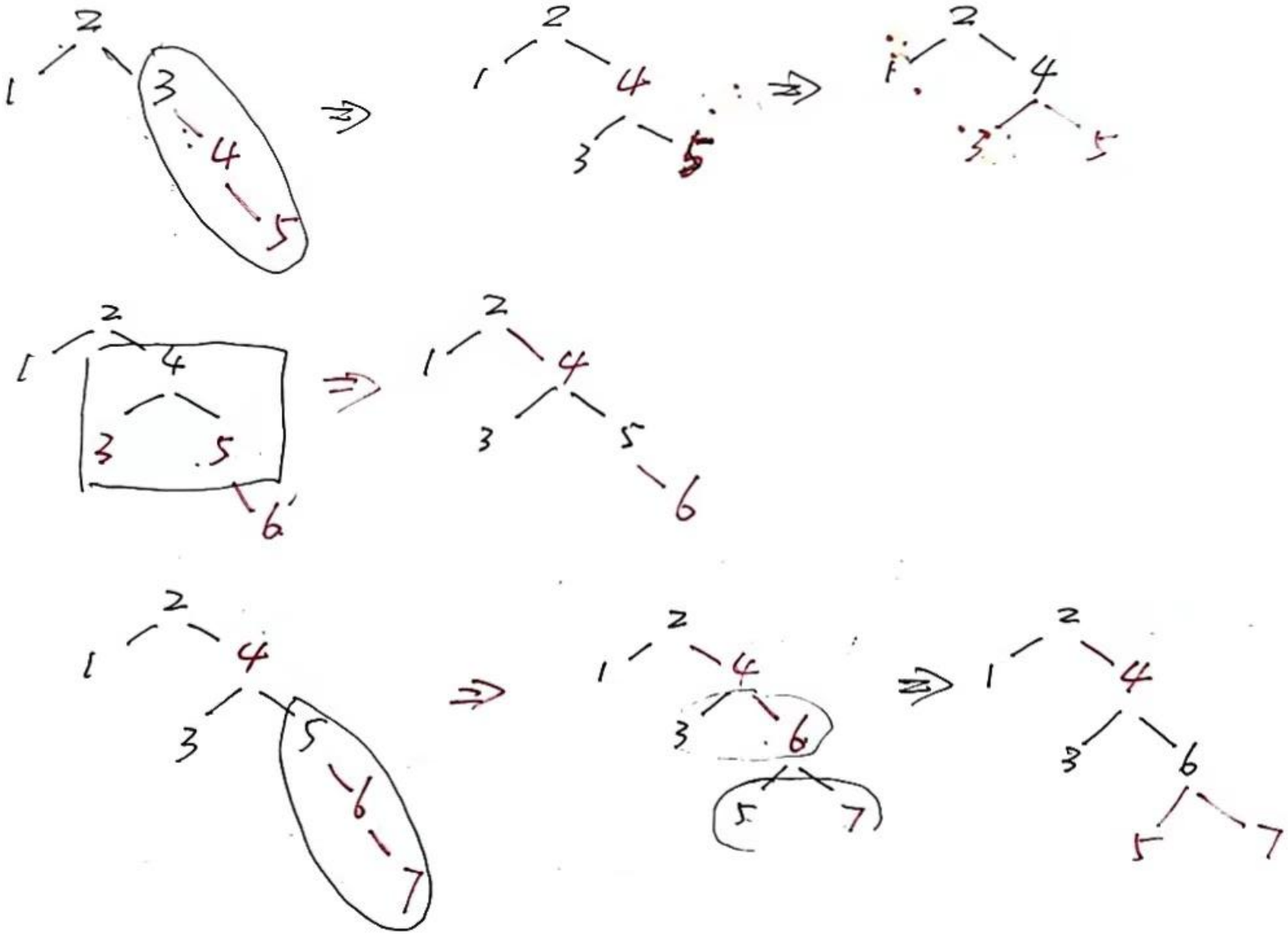


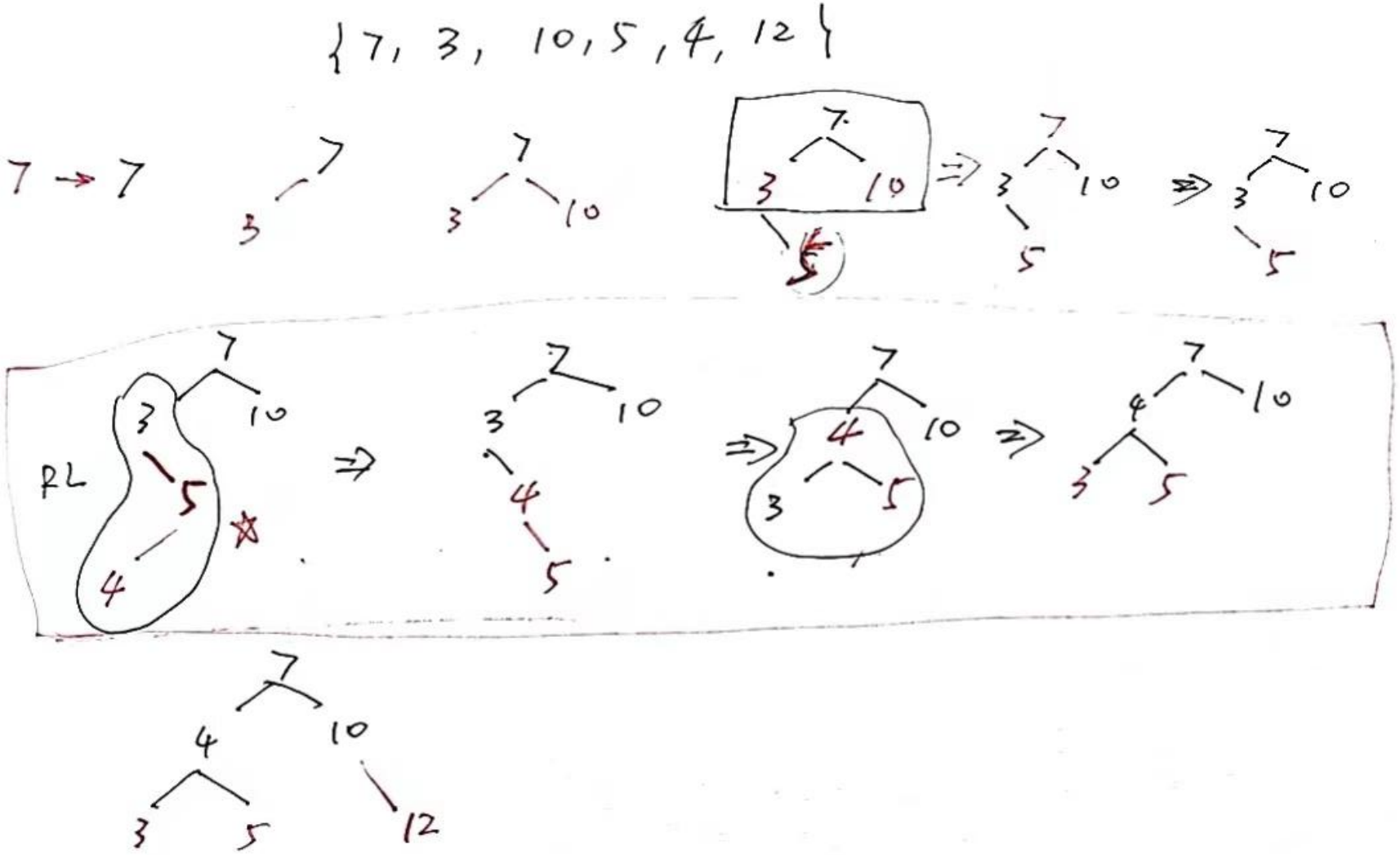
{1, 2, 3, 4, 5, 6, 7} 平不平.








2
1
3
4
5
2
1
4
3
5
2
4
3
5
2
1
4
3
5
6
2
1
4
3
5
6
2
1
4
3
6
5
7
2
1
4
3
6
5
7


{7, 3, 10, 5, 4, 12}
RL

① 根必须黑

② 非根
- 红叔叔 → 直接 调整上二层颜色
- 黑叔叔 — 调整平衡
  - LL LR RL RR
  - 自底向上查找最小不平衡子树

☆ 调整时不改色

调整之后再改色
- 一定切记要保证黑高相同
- 根一定要黑