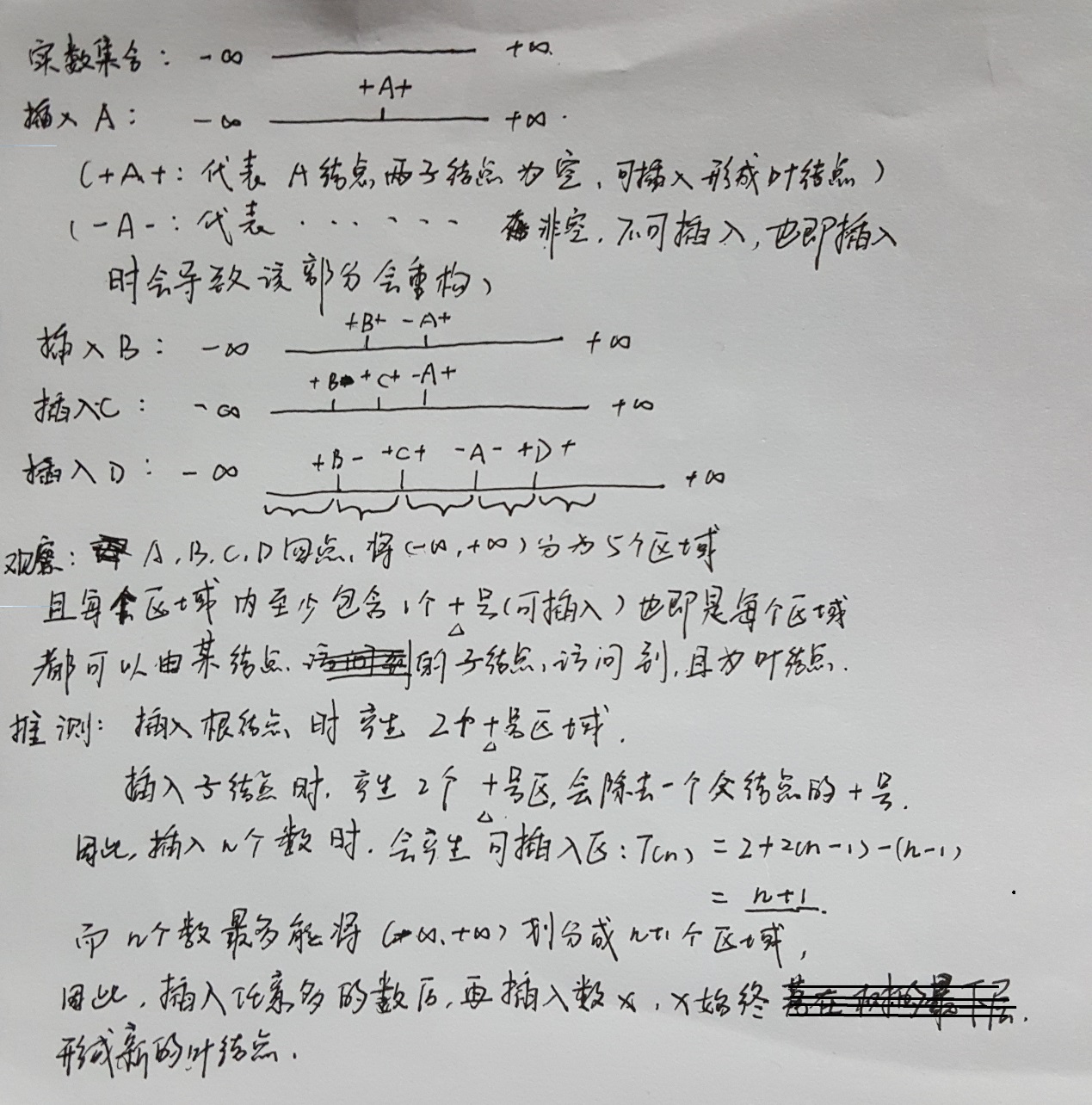
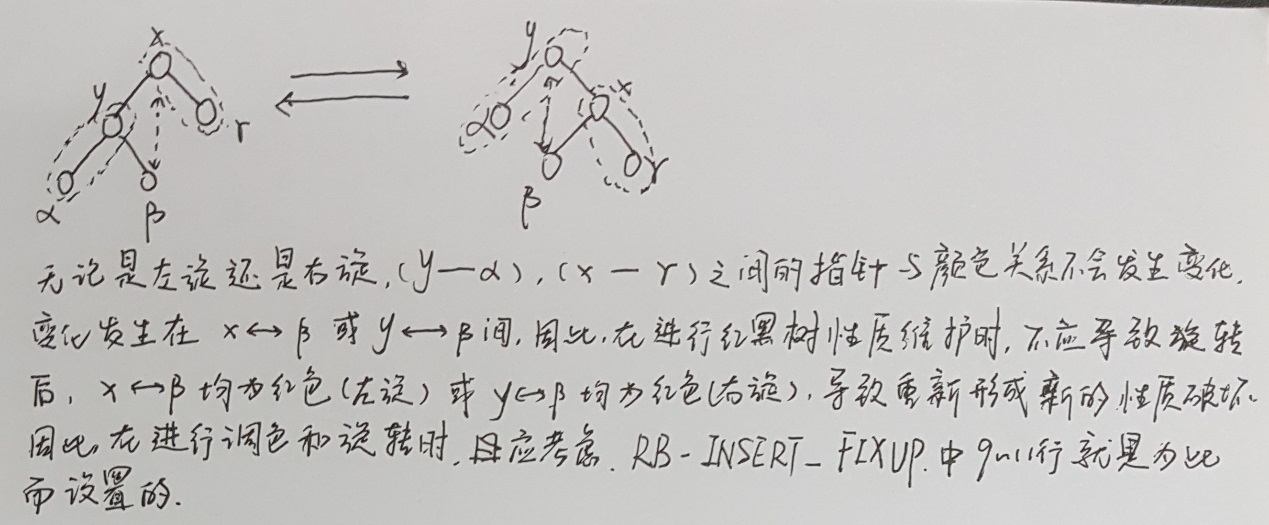
Thought1.为什么二叉搜索树Insert操作总会插入形成新的叶结点，而不会插入到两内部节点间，从而导致这部分结构发生变化？



Thoughts2.红黑树RB-INSERT插入过程中，维护红黑树性质的思考

A: 旋转：

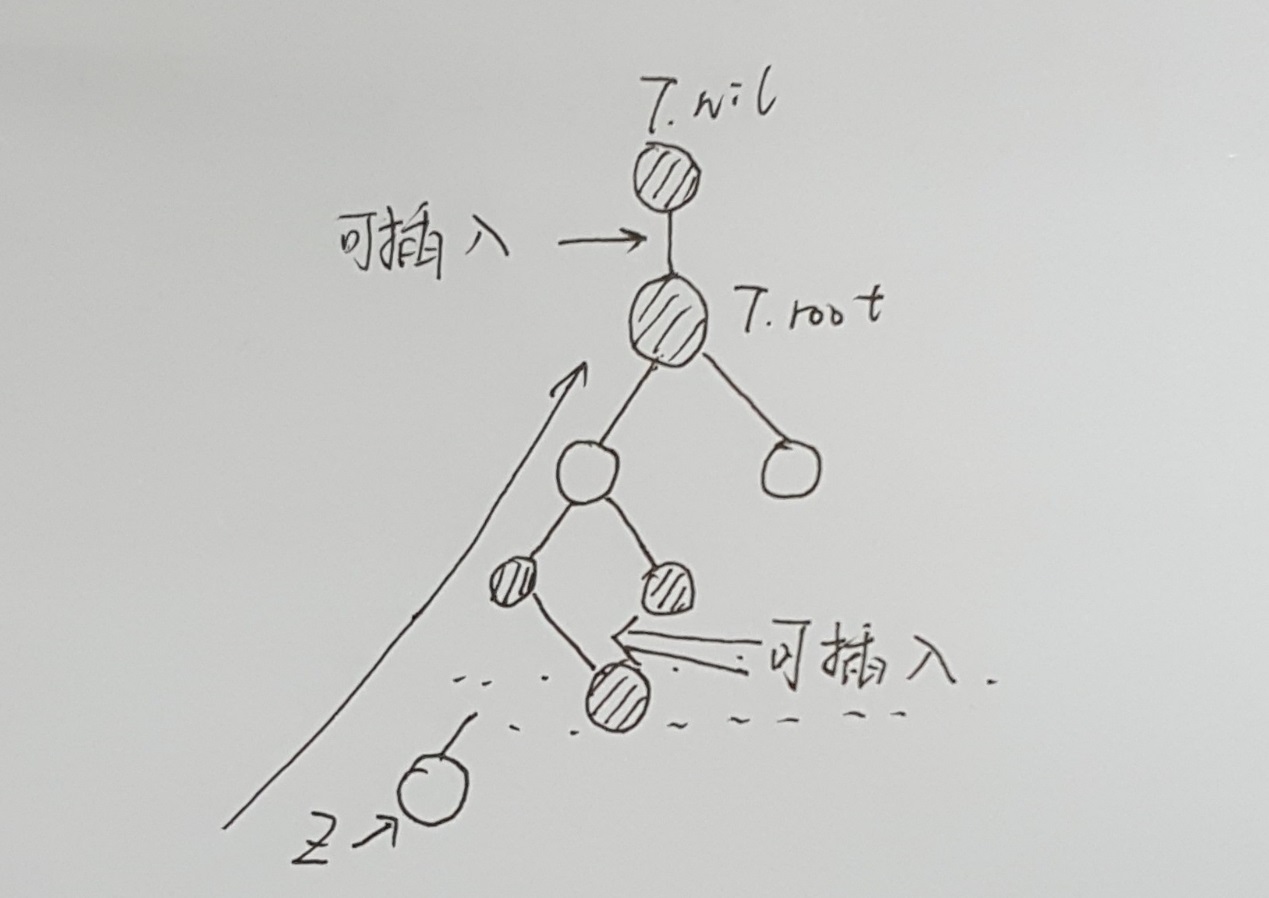


B: 什么样的情况下向红黑树中插入一个元素，其黑高不会变化？什么情况下其黑高会增加？

考虑到红黑树性质4：如果一个结点是红色的，则其两个子结点都是黑色的。因此，只有在每个结点与其相邻结点颜色不同时（包括T.NIL），其黑高最小，此时再插入一个元素，必然导致黑高增加。在黑高固定且结点达到最大值时，该树处于“黑高h的满载状态”。

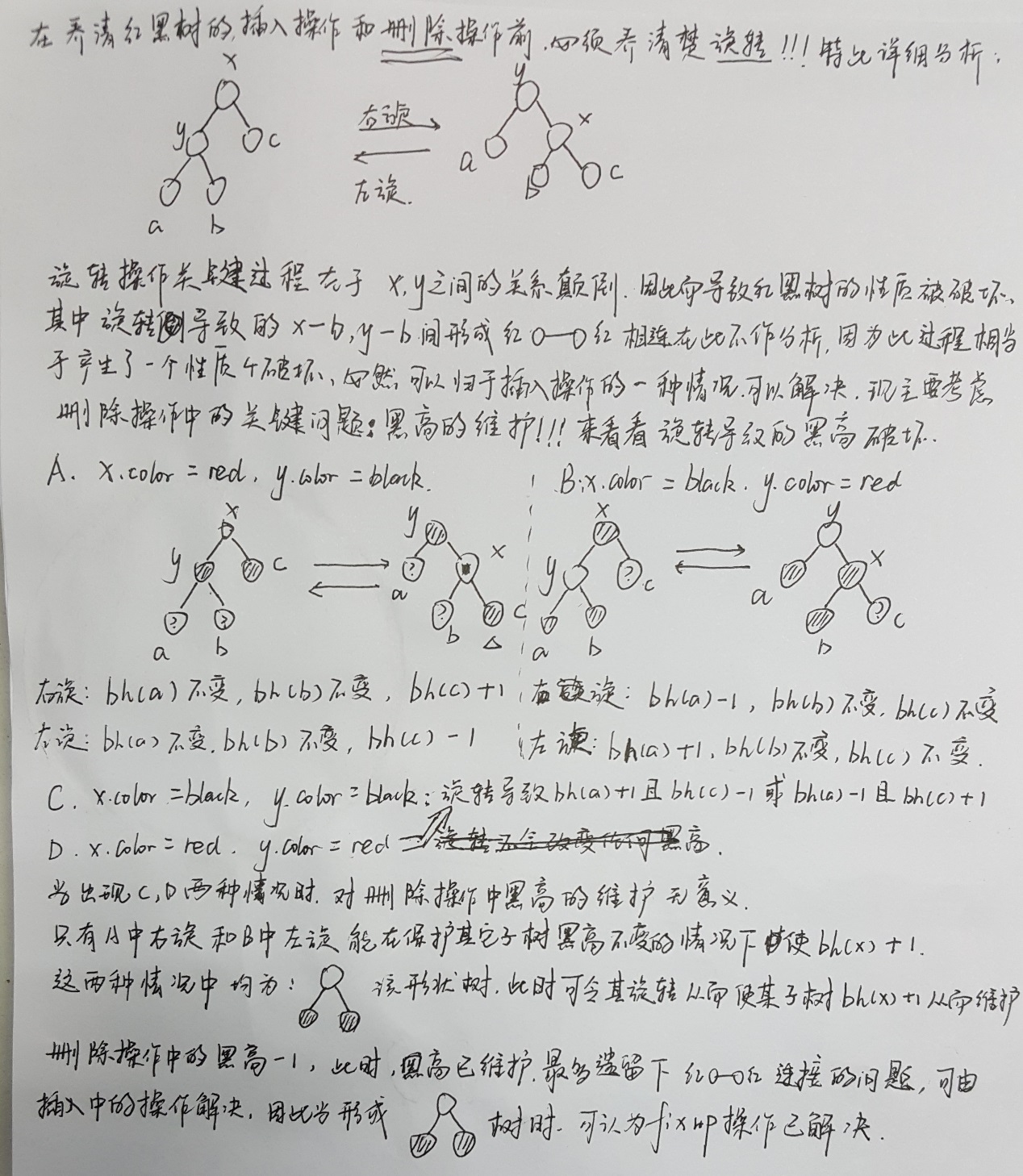
除了树的整体具备“满载状态”以外，每个根节点到叶结点的简单路径也存在“满载状态”，因为RB-INSERT的操作方式，每插入一个红色节点Z导致红黑树性质破坏时，该“冗余”的红色结点会在其简单路径上搜索最近的黑（父结点）——黑（叔结点）的相连模式，并在此插入。如果整条简单路径上，都不存在与该路径上黑结点相连的黑结点，则该路径“满载”，只能插入到T.nil（黑）——T.root（黑）中，从而导致黑高+1。

因此，综合上述，红黑树的性质维护过程，根本在于从叶结点到根节点的简单路径中，所有结点和与其直接相连的子结点是否存在黑——黑的相连模式，并且该模式至少存在一个，即T.nil——T.root。性质维护过程会将该红色插入到最近的黑——黑连线中。



Thought3. 红黑树删除过程，光看书，实在难以理解，不如自己动手分析各种情况。

A: 还是旋转！想理解好删除过程中如何将该黑高少1的子树，进行维护，必须知道旋转过程中对黑高的影响，因此旋转分析如下：



在此基础之上，分析删除过程如下：

