Len[i]: 表示第 i 层中，当前节点和右节点间的实际节点数

rand()/RAND\_MAX可表示为在[0,1]区间内的随机数

search：找到第 i 层最大的 键值小于key 的节点

插入：search可以找到插入的位置（该节点的右侧）

查询：是否找到取决于 cur 的右节点是否为待查询节点

删除：同理

由于哨兵节点 head,tail 的存在，cur 和 cur -> nxt[0] 都是一定存在的

**查询：**自然解决，cur节点的右节点

**插入：**

插入时，为防止level无限制增长，我们限定每次插入的level最多为cur\_maxlevel + 1

如何维护len？

维护两个数组：note\_to\_update 和 len\_between

每一层需要修改的节点位置，即“最大的键值小于key的节点”，也即cur

自下而上，用下一层的len去更新上一层的len，用左半部分的len去更新右半部分的len

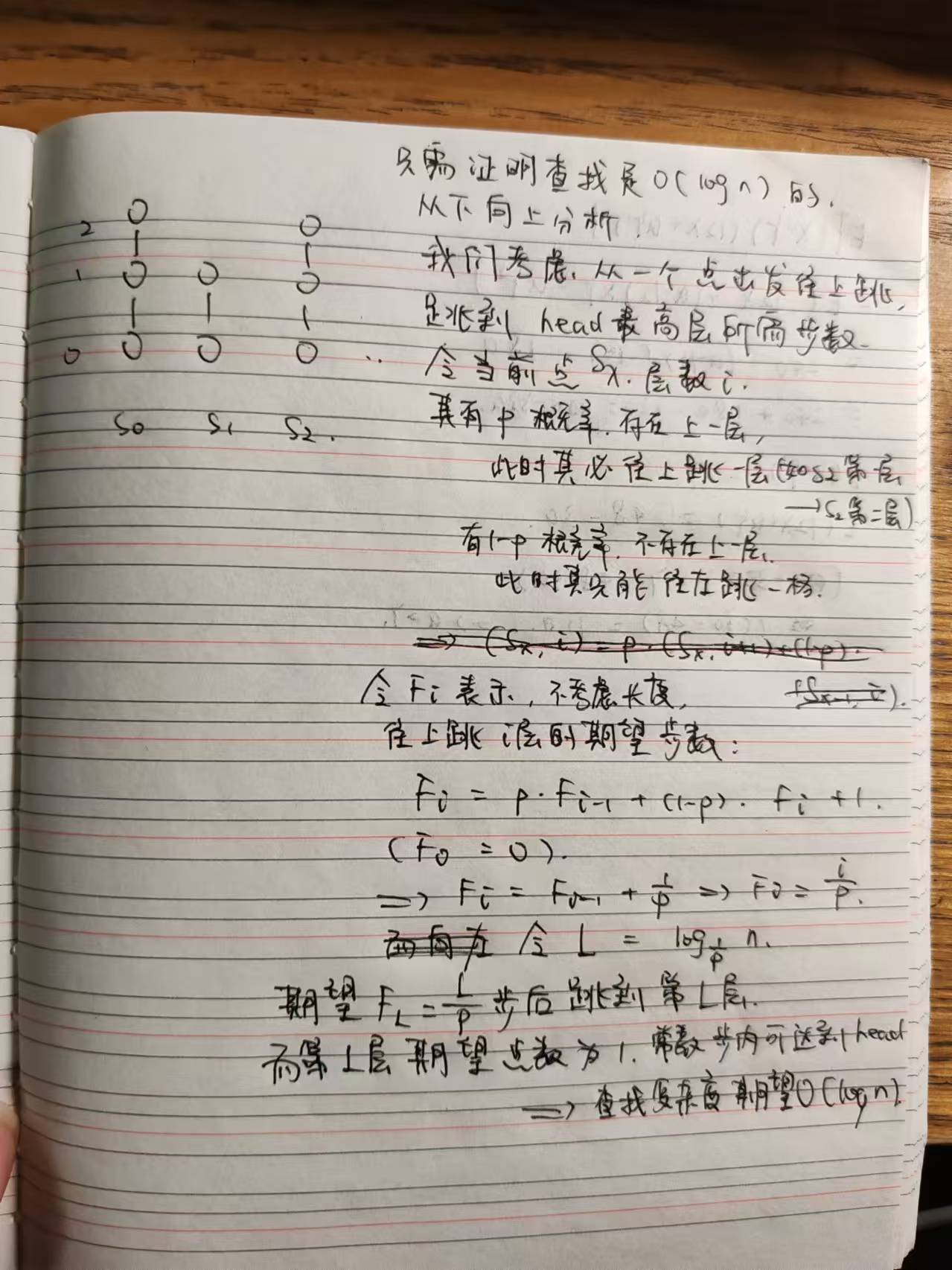
**Deletion**和insertion一样，无非是将两段len做合并，特殊情况是top层删光

**Kth：**自然的，根据len决定这部能不能走

测试：

pb\_ds 库是 GNU C++ 标准库的一部分，其中包含了红黑树

时间复杂度：



唯一的区别是要存下每层需修改的节点并依次对其操作，这一步的复杂度是O(层高)