**二叉树链式结构的实现**

**对于二叉树中求节点个数的问题，不管return顺序怎么改变都是后续。**

**因为一个节点的左子树和右子树的个数都必须等左子树和右子树访问完了之后，再加上原来的那个节点才能求出总的节点。**

**对于一个节点的个数都必须要先求这个树的左子树和右子树的总结点求完之后，再加上这个节点才能求出来。**

**对于二叉树来说，相对于以前的链表和顺序表来说，代码变简单了，但是代码的执行变复杂了。**

**高度（深度）：高度其实就是在搜索最长的那个路径的一个层次。**

**假设校长要求两个院里身高最高的，然后校长就把这个任务分配给了院长，然后院长将这个任务分配给了辅导员，辅导员将这个任务分配给了班长，然后每个班长统计出了这个班最高的人了之后，就往上返回，返回了之后，就比较左子树和右子树的分别的高度谁是最大的，最后统计给校长，这个任务就完成了**

**这就是分治算法**

**分治：**

1. **子问题**
2. **返回条件（最小子问题）**

**对于求一个树的高度，可以分为左子树的高度和右子树的高度中大的那个高度加1，**

**对于二叉树的子问题来说，二叉树中的子问题就是左子树和右子树。**

**对于求高度来说，最小子问题是空（而不是叶子）**

**如果用叶子作为最小子问题的话，就会出现空指针的问题，当一个节点的左子树和右子树只有一个节点时，有一边是空，所以当访问空的时候，就会出现空指针的访问，就会报错**

**如何查找二叉树的第k层的节点**

**二叉树第k层的节点就可以转化成第下一层的第k-1层**

**假如一个二叉树查找第三层的节点个数，将第一层的第三层转化成第二层的第2层。**

**查找x所在的节点个数**

**递归：**

**返回条件：root==NULL**

**返回NULL（没找到）**

**Root->val==x;**

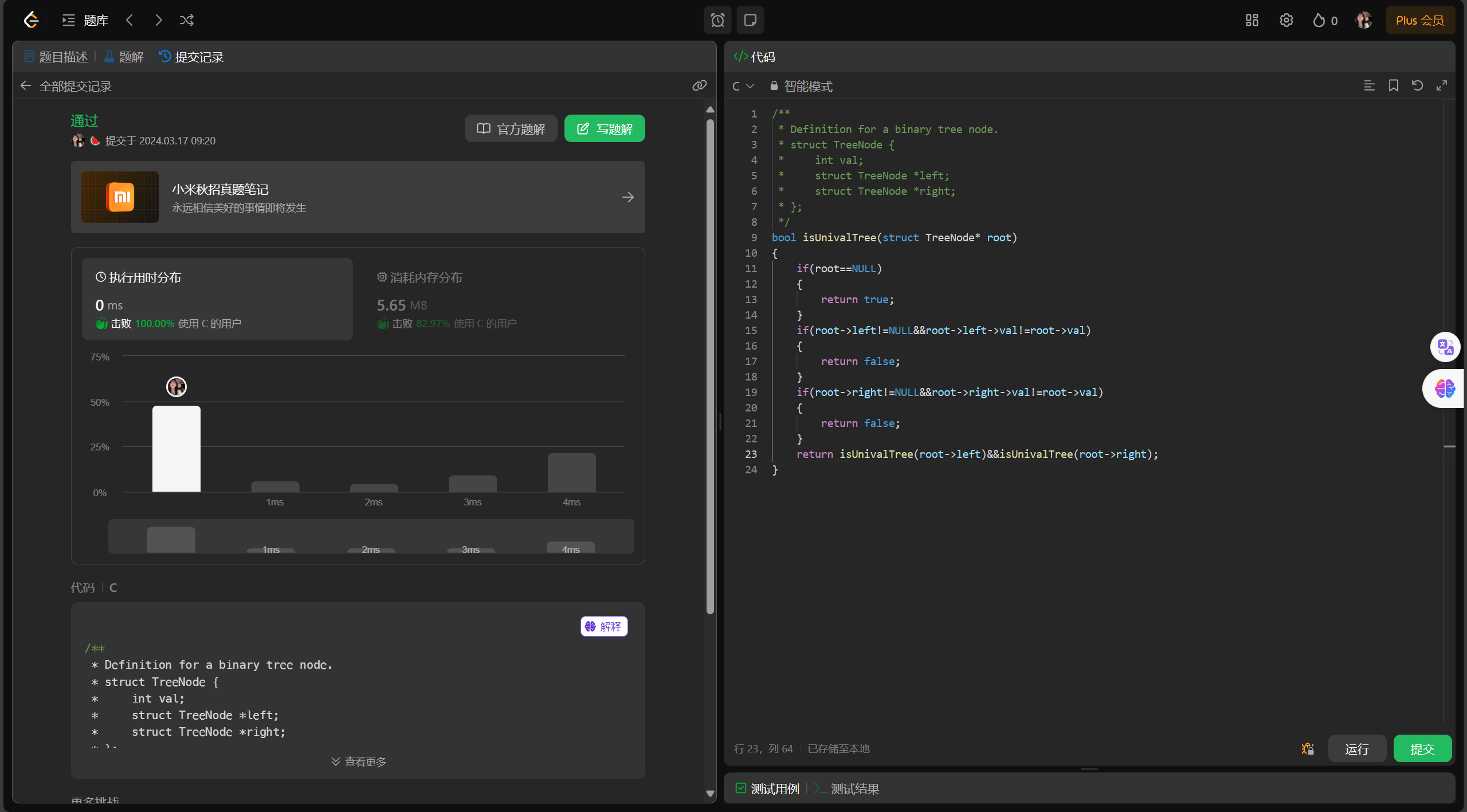
**返回我的root节点**

**当节点不是我们要的节点转化成先在左子树找，没有找到，再去右子树找。**

**单值二叉树的判断：**

**子问题：用根的值和左孩子和右孩子进行比较，如果相等则就是单值二叉树，再用递归去判断左子树的左子树和右子树的值判断相不相等，再进行递归。**

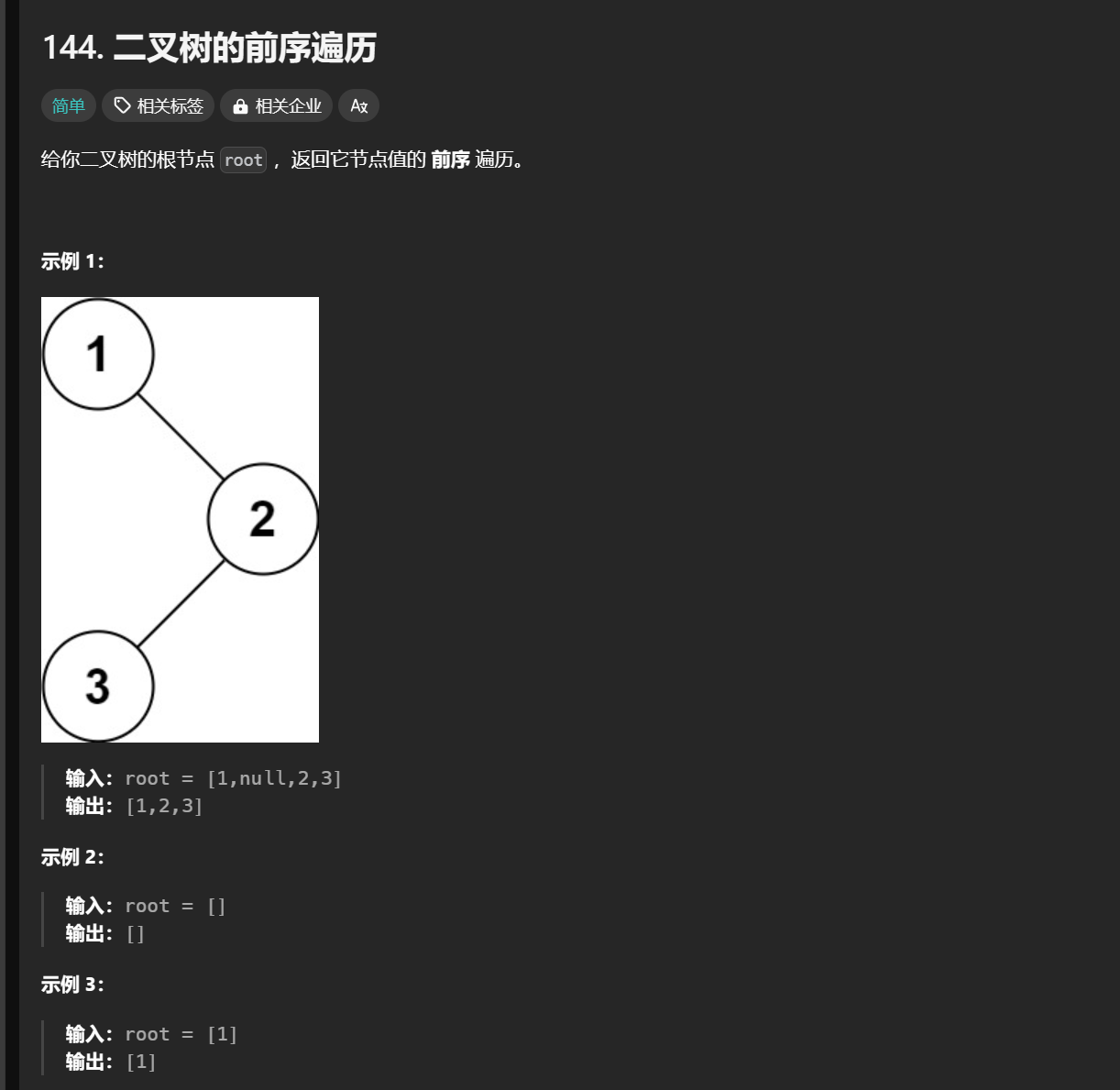
**前期调整后期是常态，后期一周四节课，周末必然有两节课，周末是满课。**

****

**先比较当前节点和下一个左孩子和右孩子是否相等，return递归**

**对称二叉树的判断，就是判断左子树的右子树和右子树的左子树是否相等，然后判断左子树的左子树和右子树的右子树是否相等，**

**对称二叉树也叫镜像二叉树，和isSameTree基本上是同一个思路，左子树和右子树比较，右子树和左子树比较，可以写一个函数传两个参数，然后再进行比较**

****

**问题：数组开多大？**

**解决方法：像顺序表一样没有空间直接malloc一个空间，不断开空间带来的不便->两个空间可能是不连续的，如果数据是连续扩容，就不会产生太大的消耗，如果是异地扩，就会带来更大的消耗。**

**还有一种办法就是：一次性开100个空间，这样存在大量的空间浪费，（最不推荐的做法）**

**二叉树+排序**

**判断完全二叉树**

**通过节点的个数来判断完全二叉树是搞不定的，只能判断满二叉树，完全二叉树的节点个数是一个返回，最后一层是确定不了的，最后一层的节点个数无法判断是否连续的问题，判断完全二叉树借助层序遍历，通过层序遍历到最后一层的时候，如果遇到了NULL就停止，然后判断二叉树是否为空，就可以了。**

**判断完全二叉树一层一层遍历最后一层的数遍历完之后后面全是空。**

**完全二叉树和非完全二叉树的特征是：**

**完全二叉树的非空和空是隔离开来的，非完全二叉树：非空和空不是隔开的，是不连续的。**

**层序遍历是最简单的思路，还有一种就是去标记遇到空了之后后面就不能有空，**

**先用层序遍历的方式进行入数据，然后取数据的时候当遇到空的时候就直接跳出循环。**

**进入下一个循环去判断后面的数如果有非空就直接return false就可以了，如果后面没有非空就直接返回true。**

**二叉树的性质**

1. **若规定根节点的层数是1，则一颗非空二叉树的第i层上最多有2^(i-1)个节点**
2. **若规定根节点的层数为1，则深度为h的二叉树的最大节点个数是2^h-1**
3. **对于任何一颗二叉树，如果度为0其叶子结点个数是n0，度为2的分支节点个数为n2，则n0=n2+1**
4. **若规定根节点的层数为1，具有n个节点的满二叉树的深度，h=log(n+1)**
5. **对于具有n个节点的完全二叉树，如果按照从上至下从左到右的数组顺序对所有节点从0开始编号，则对于序号为i的节点有下面的关系。**

**对于第三条：增加一个度为1的，就会减少一个度为0的，新增节点一定是新增度为0的，连接后：**

**第一种：新增一个度为1的，则减少一个度为0的。**

**新增一个度为2的，则减少一个度为1的**

**完全二叉树中N1要么是1要么是0.**

**排序**

**直接插入排序：**

**基本思想：直接插入排序是一种简单的插入排序法，基本思想是：把待排序的记录按其关键码值的大小逐个插入到一个已经拍好的序的有序序列中，直到所有的记录插入完为止，得到一个新的用有序序列**

**冒泡排序：**

**前后两个比较，然后进行一轮比较把最大的冒到最后面，然后再进行排序次大的………**

**数据结构需要画图**

**希尔排序：**

**时间复杂度：第一次预排序可以用精确的算，但是后面的预排序不能按最坏的情况来算，因为前一组预排序对后一组预排序有影戏，因为部分有序了，到了最后gap等于1的时候，每组n个数据，但是有1组，这次排序的时间复杂度是n，因为这个时候已经很接近有序了，希尔排序每次预排序的次数的变化大概就是：gap在不断变小，**

**快速排序：**

**六十年前，一个叫霍尔的人创造了快速排序，快速排序的价值很大，一些软件，很多软件的底层就是快速排序，比如京东的商品的筛选，可能底层都是用的快速排序。**

**目标是选一个值做关键字，第一个值和最后一个值比较方便，**

**第一遍排序达到的目的是，保证左边比右边小。左边比key要小，右边比key要大。**

**单趟排序达到了剩下的怎么走？**

**单趟排序达到的目的，把key放在了最终位置，比它小的都在左边，比他大的都放在右边，（单趟的快速排序达到的价值）**

**如果左边有序右边有序这个排序就排完了，单趟排序排完之后左区间和右区间，再进行相同的递归进行排序，对于这个递归的最小子问题就是，最后得区间只有一个值，还有一种区间就是不存在，**

**只有一个值就是l==r，**

**快速排序也叫二叉树排序。**

**在快速排序中为什么相遇位置一定比key小，右边先走保证的。**

**一种是L遇R；R先走一定是在一个比key小的位置停下来了，L没有找到比key大的，就会和R相遇，所以，相遇位置是R停的位置，所以相遇位置是比key小的位置**

**一种是R遇L：第一轮以后的，先交换了，L位置的值小于key，R位置的值是大于key的，第一轮交换之后，第二轮，R启动找小的，没有找到，跟L相遇了，相遇的位置L停下来比key小，因为这是上一轮交换的比key小的R。**

**如果是第一轮，R和L相遇了，那么只有一个可能就是R没有找到小的，一直找，直到遇到了L，L和R还有key重合了。**

**快速排序的最坏的情况是N^2**

**在比较两个算法时注意注意两个算法对照的样本是否相同，如果两个算法对照的样本不同，那两个样本没法进行比较。**

**注意对照两个算法进行比较的样本。**

**如果数据量非常大，当到最后还有十个数据的时候，因为有很多组，所以对这些数据建立一个递归的消耗是很大的。**

**因为对于一个二叉树来说，最后两层的节点个数将近占整个二叉树的节点个数的百分之八十，所以，到最后两层的时候，我可以直接将其排序完了之后，直接返回上去，这样的优化可以减去大量的空间的浪费。**

**在debug版本下这中优化的版本比realise版的优化更明显。**

**归并排序**

**如果两个子序列有序，就可以利用归并排序将两个合并。**

**归并排序：就是把两个 有序数组合并成一个 新的数组。**

**当一个数组左区间有序，右区间也有序可以在原数组上进行归并吗？显然是不能的。**

**不能在原数组上进行操作，必须取2两个元素中小的那个元素进行尾插，取小的尾插的逻辑就是每次取最小的依次往后放，如果在原数组中操作会导致覆盖的问题，交换也会导致很多问题，所以数组的归并必须借助一个临时数组，**

**归并的前提是左区间有序，右区间也有序，但是如果左右区间没有序该如何使用归并排序呢。如果左区间 没有序，右区间也没有序可以进行一个分治，将左右区间调为有序，假如一个八个数的数组，可以分治将这个数组的左区间有序改成左右四个数有序，然后 再分治改为左右四个数的左右两个数的有序，最后改为左右两个数的左右一个数的有序，最后几可以网上进行归了，一直拆下去然后归回来**