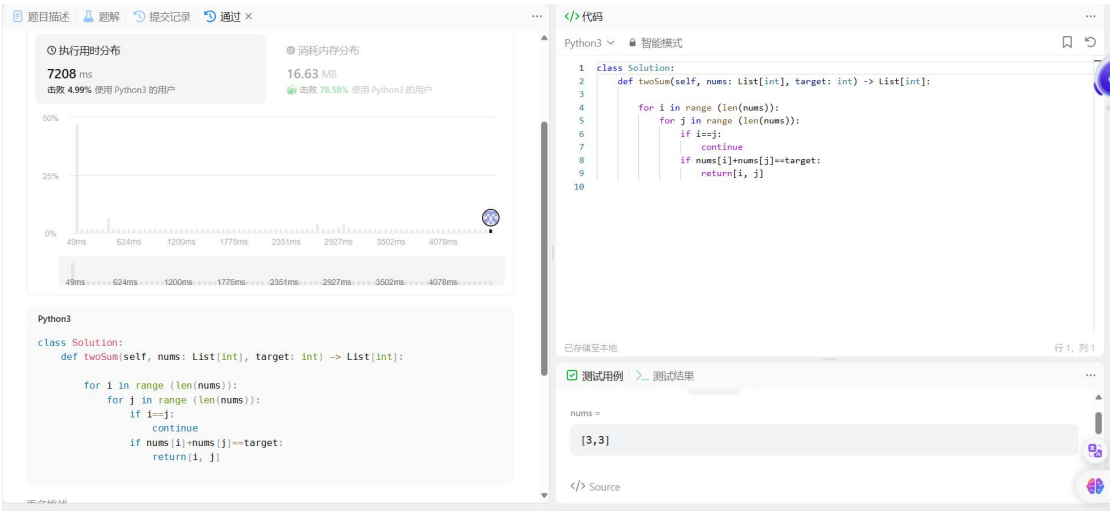
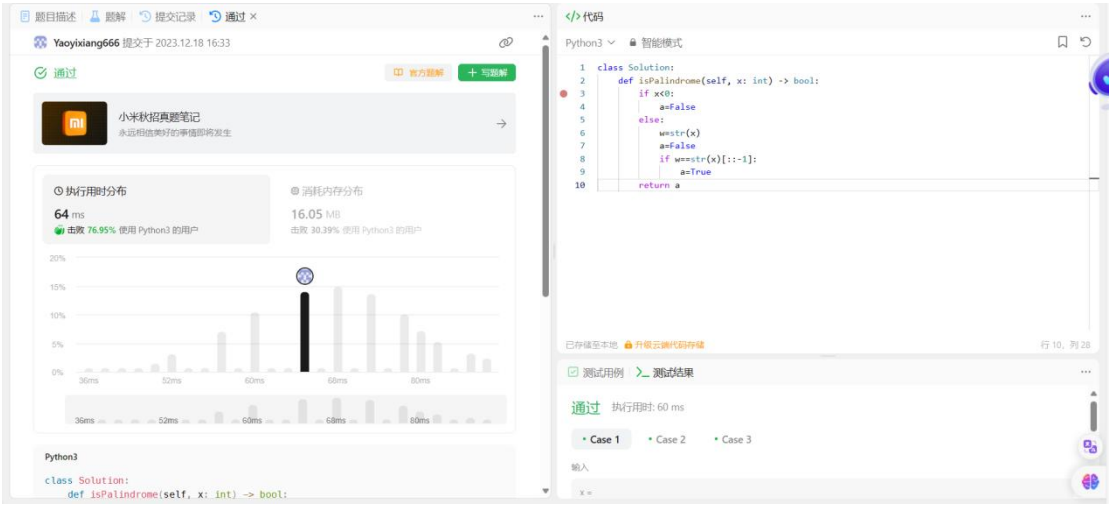
1. 两数之和



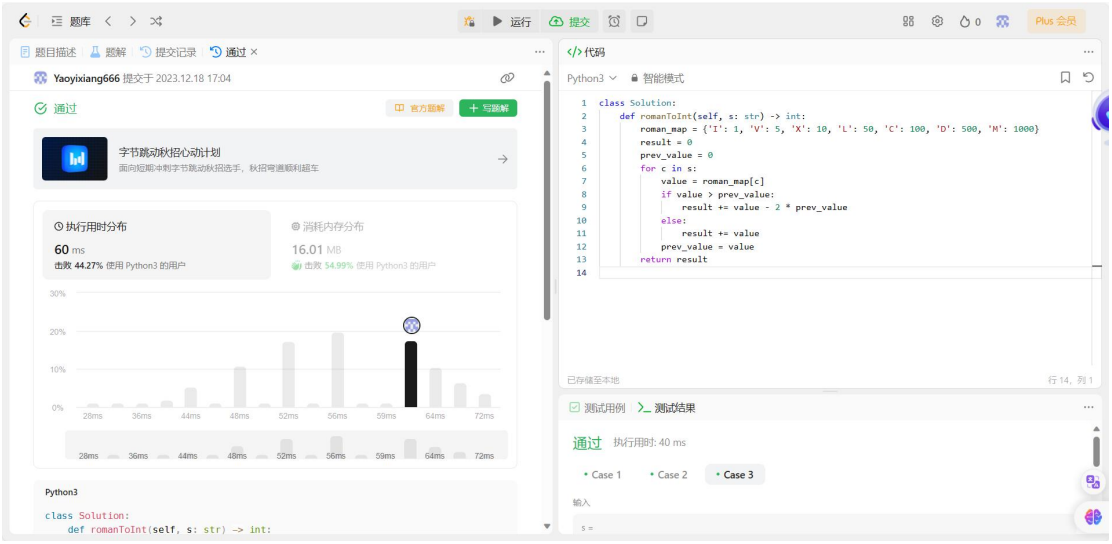
从数组第一个数开始遍历循环，若为同一个元素重复遍历则跳出并继续进行循环，若满足和为目标值则输出。

2. 回文数



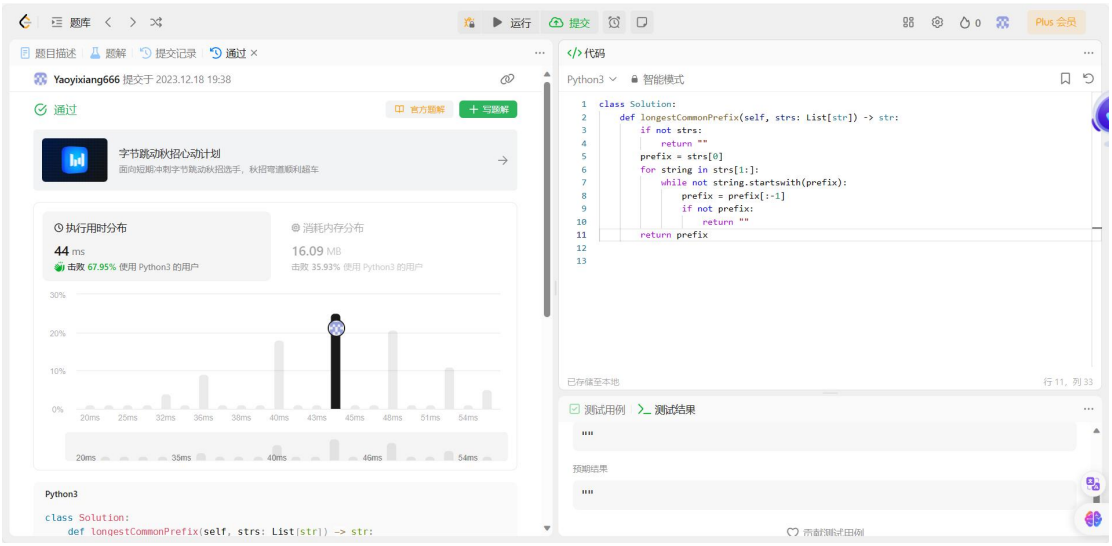
因为负数不是回文数，故先判断其是否是回文数，若不是负数，再将其转化为字符串进行倒置，若倒置后与原来值相同，则返回 Ture，否则返回 False

3. 罗马数字转整数



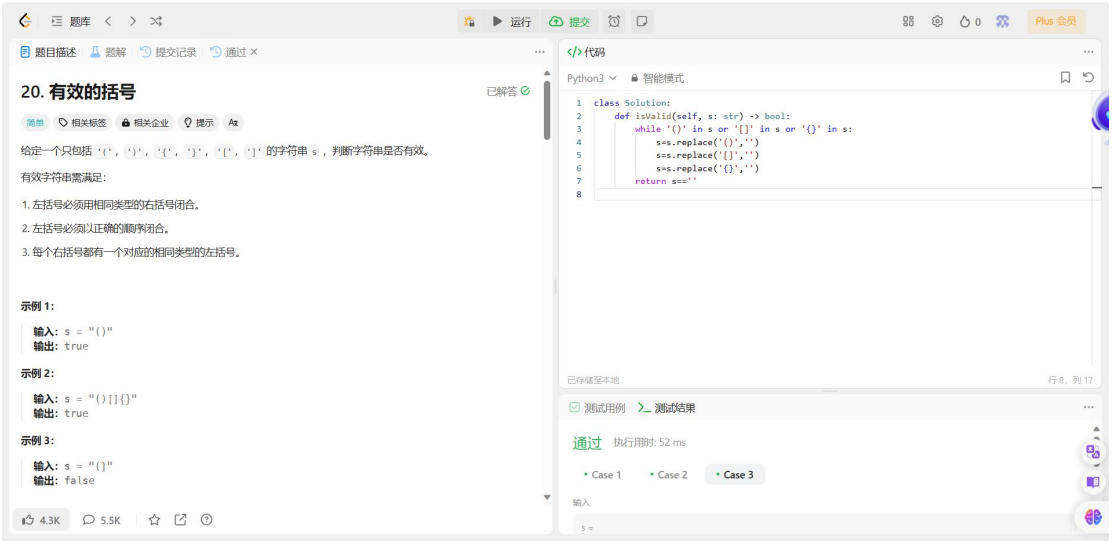
首先定义一个罗马数字字符和对应数值的映射，然后遍历输入的罗马数字字符串，将累加得到的值给予 result 变量；用 prev_value 记录上个字符的数值，并与当前字符的数值对比，若当前字符的数值大于上一个字符的数值，则减去上一个字符的两倍数值，然后加上当前字符的数值，最后返回累计的值

4. 最长公共前缀



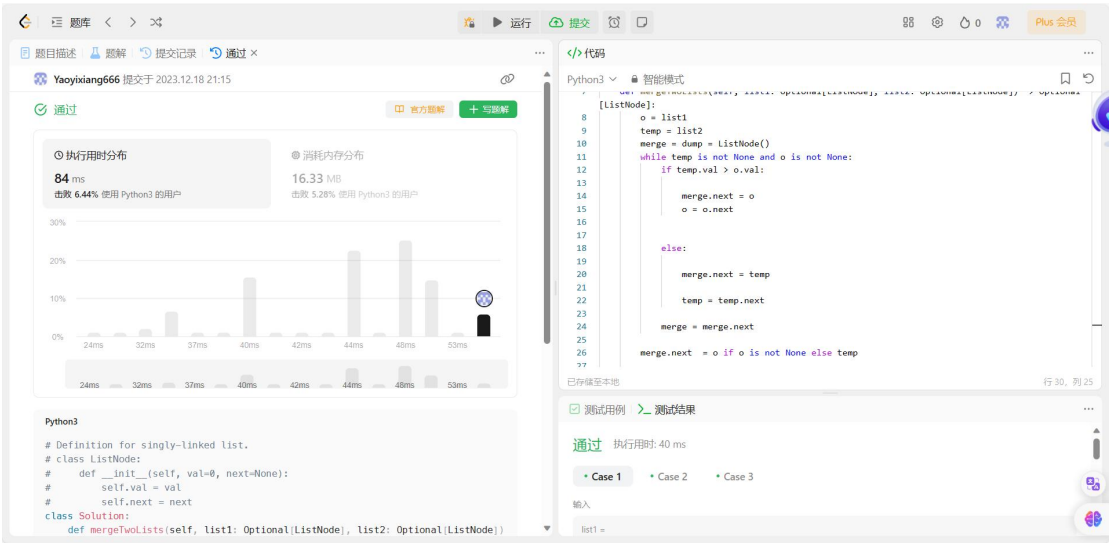
首先判断字符串数组是否为空，然后将数组中第一个字符串取出来并用 prefix 定义，用 while 循环检查当前字符串是否以当前的公共前缀 prefix 开头，如果不是则删除最后一个字符再进行判断，直到字符串为空或找到最长公共前缀，最后返回字符串。

5. 有效的括号



用 while 循环将 s 中的（），{}，[]取出直到将正常的括号全部取光，若仍有剩余则输出 False，若为空则输出 True

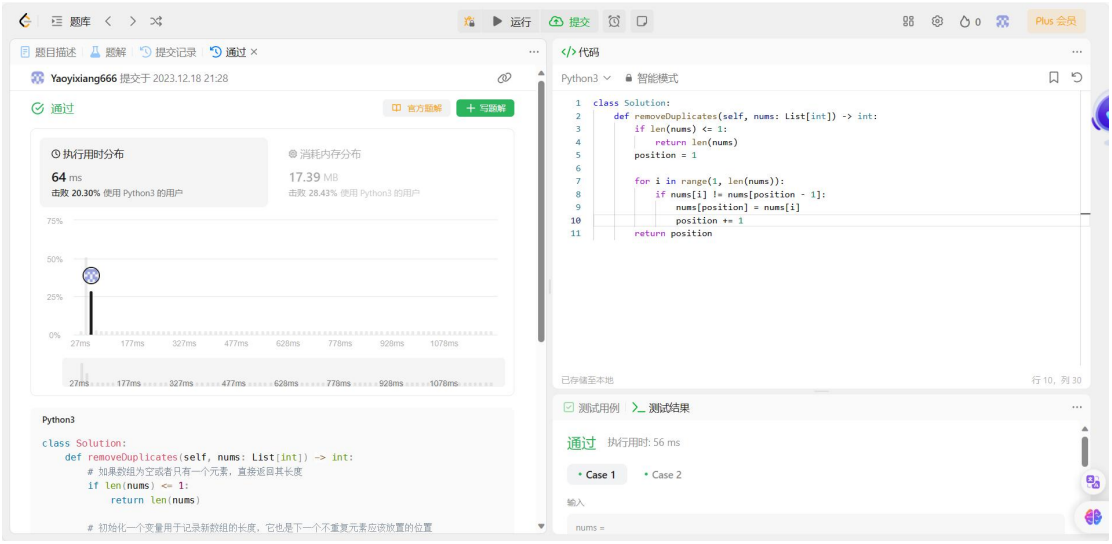
6. 合成两个有序链表



首先创建一个新的 ListNode 作为合并后的链表的头部，然后设置一个指针 merge 指向这个头部，同时设置另一个指针 dump 也指向这个头部。然后开始遍历两个输入的有序链表 list1 和 list2。在每一步中，我们比较 list1 和 list2 当前节点的值，将值较小的节点接到 merge 指针的后面，并移动指向这个值较小节点的列表的指针。如果 list1 当前节点的值小于 list2 当前节点的值，将 merge 的下一个节点指向 list1 当前节点，并将 list1 的指针移动到下一个节点。如果 list2 当前节点的值小于 list1 当前节点的值，将 merge 的下一个节点指向 list2 当前节点，并将 list2 的指针移动到下一个节点。

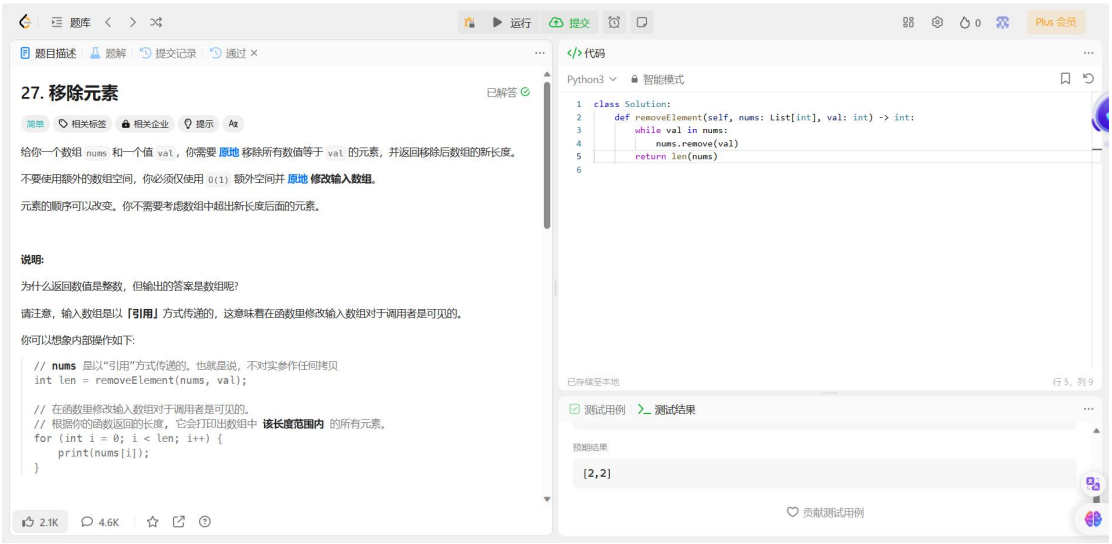
然后移动 merge 指针到下一个节点。重复以上步骤，直到 list1 或 list2 的指针为空。

7. 删除有序数组中的重复项



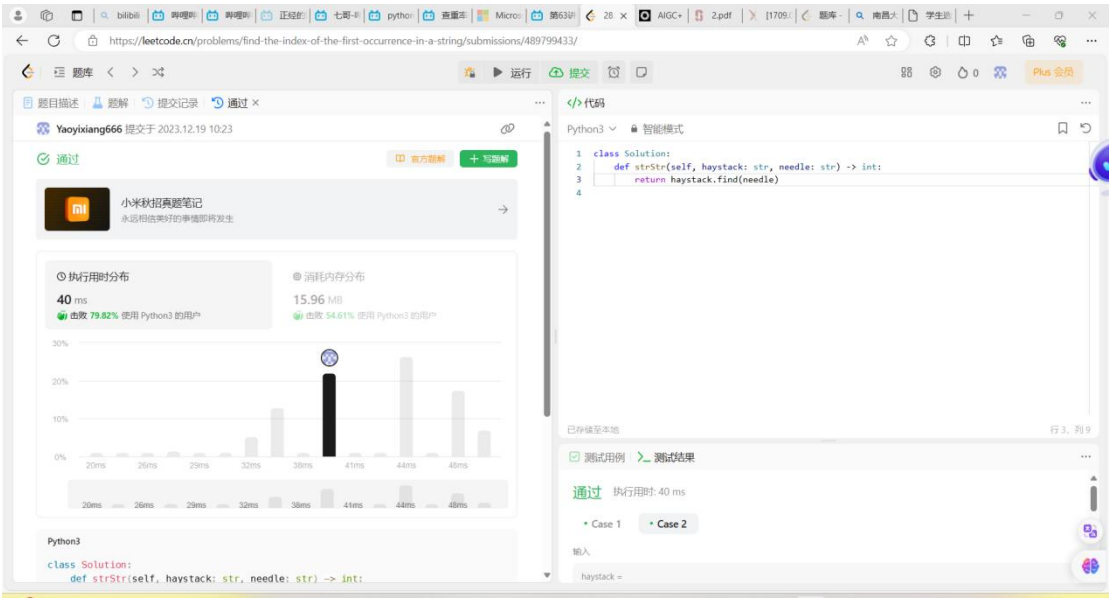
用 position 变量来记录新数组的长度，它同时也指示下一个不重复元素应该放置的位置。再从第二个元素开始遍历数组，每当遇到一个与前一个元素不同的元素时，就将其移动到 position 所指示的位置，并递增 position。最后，返回 position 作为新数组的长度。

8. 移除元素



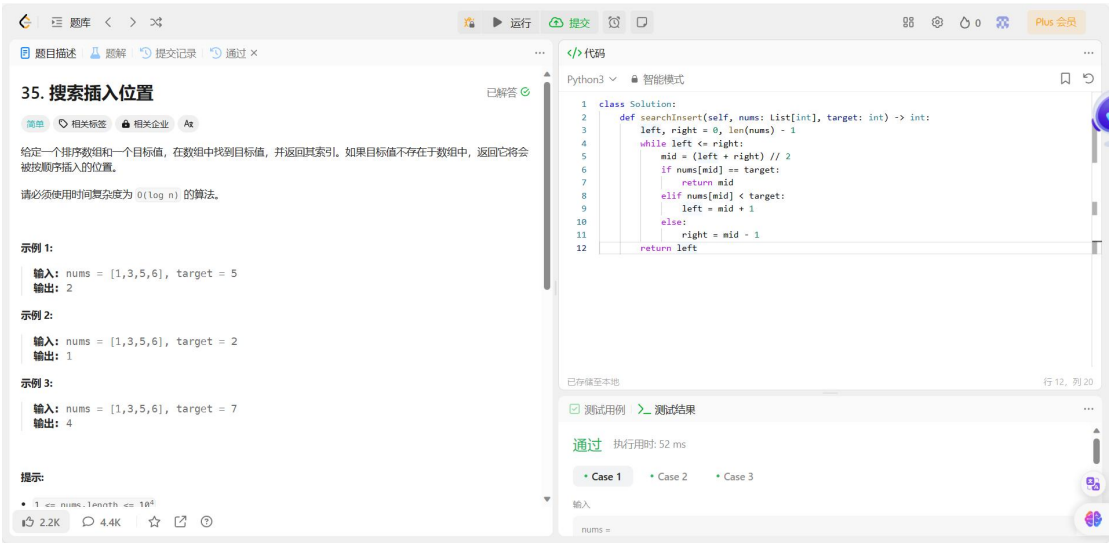
用 while 循环遍历 nums 中的 val 值，若存在则移除，最后返回一处后数组的长度

9. 找出字符串中第一个匹配项的下标



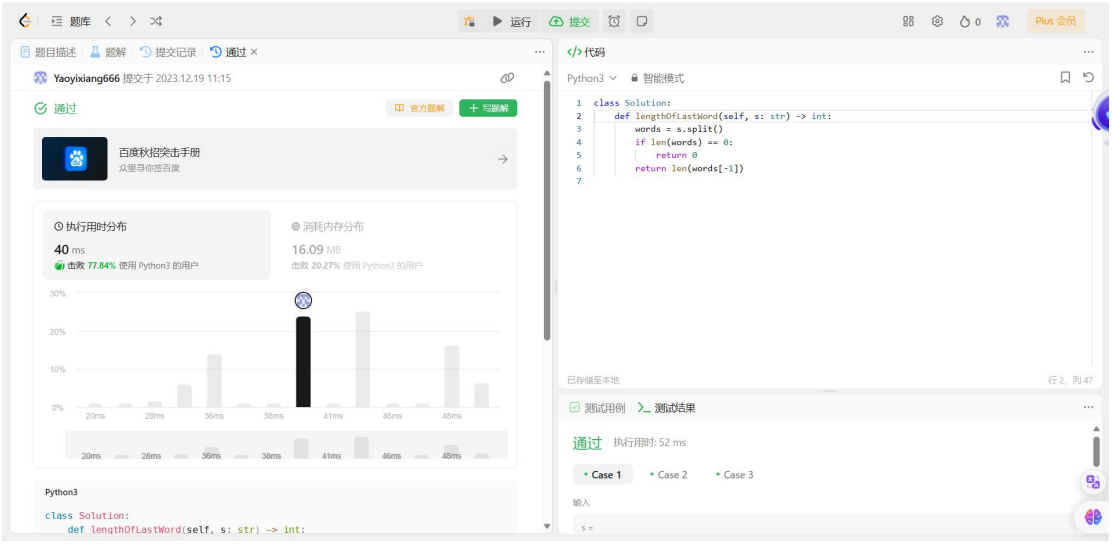
用 str 的 find 方法寻找字符串中是否包含指定的子字符串，如果找到了则返回其索引，否则返回-1

10. 搜索插入位置



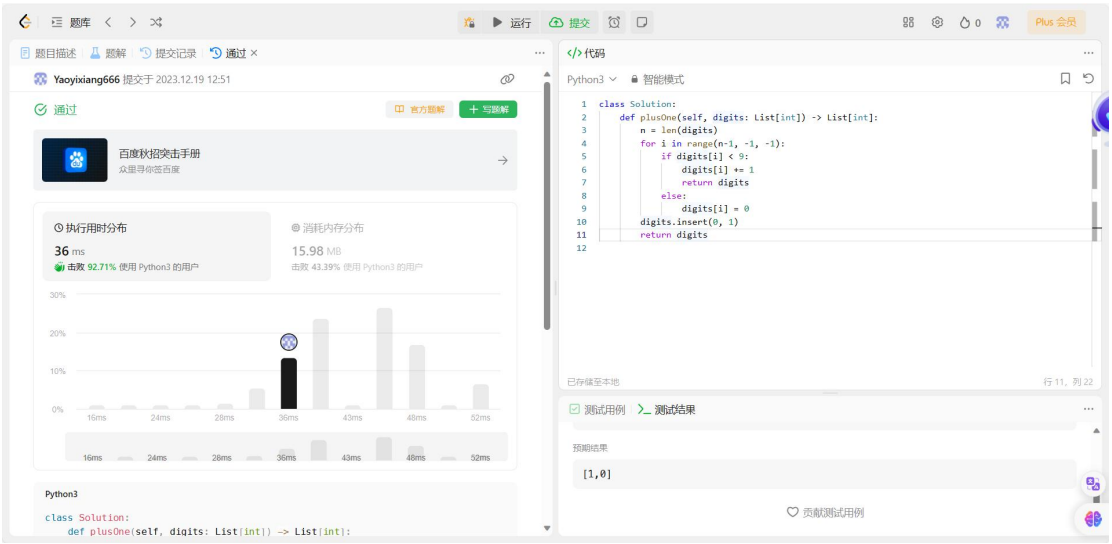
首先将目标值与数组中间元素进行比较,如果目标值等于中间元素，则返回中间元素的索引。如果目标值小于中间元素，则说明目标值可能在左半部分，反之则说明目标值可能在右半部分，然后新的搜索范围中重复上述步骤，直到找到目标值或者确定目标值应该插入的位置。

11. 最后一个单词的长度



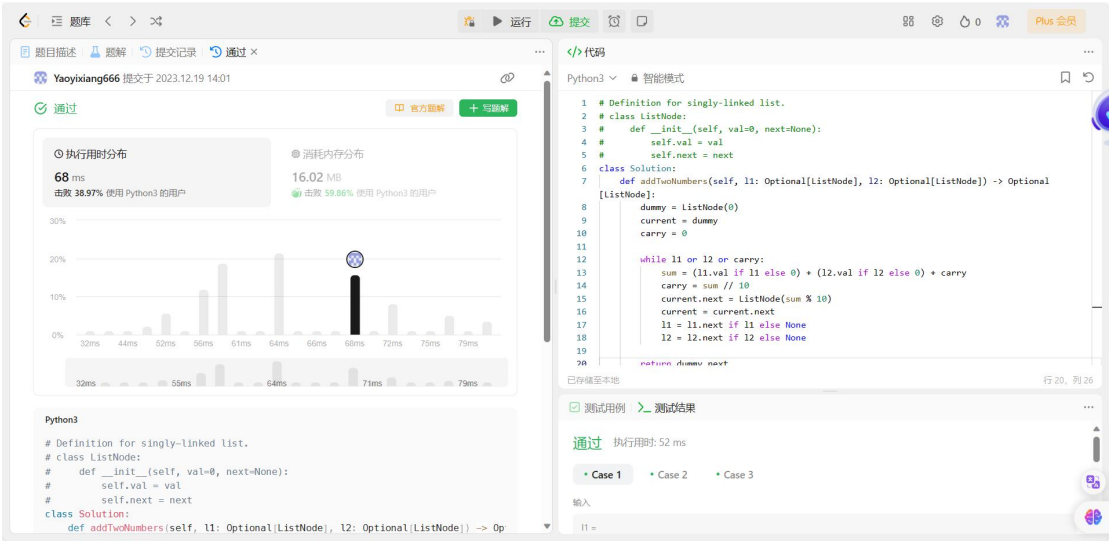
先用 spilt 函数将其按空格分割为单词，并将其保存在列表 words 中，然后检查列表是否为空，若为空则返回 0，反之则返回 word 最后一个单词的长度。

12. 加一



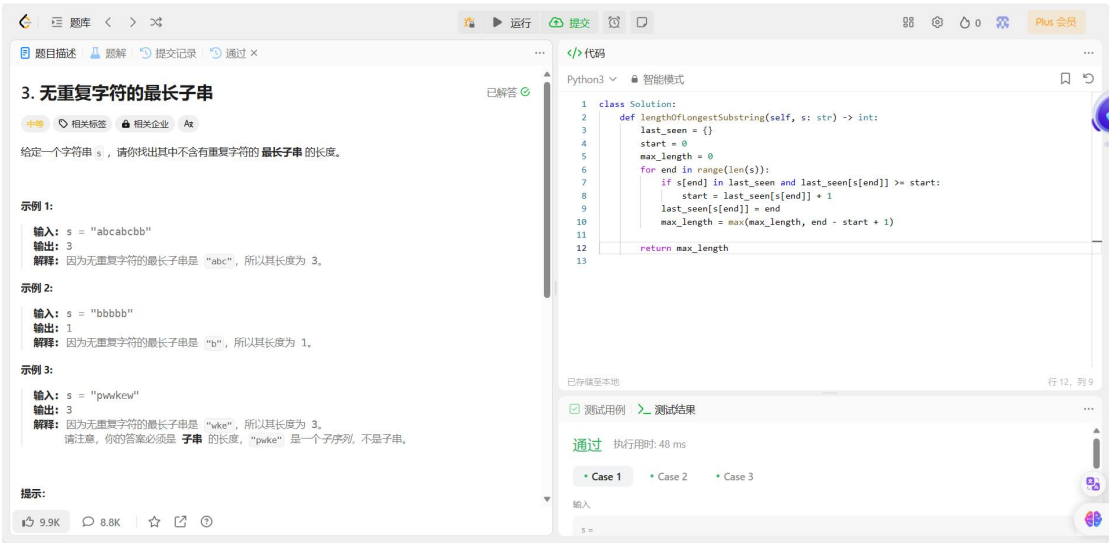
从数组的最后一个元素开始遍历，若元素小于 9，则直接加一并返回；若元素为 9，则将其设为 0。如果整个数组都是 9，则在数组的开头插入 1。

13. 两数相加



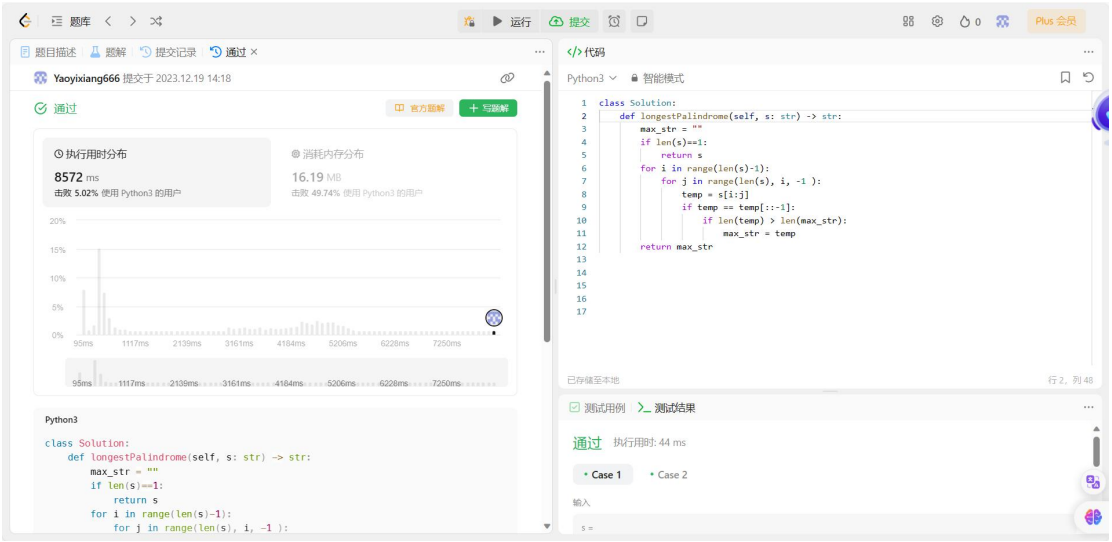
使用一个虚拟头节点 dummy 来简化链表操作。然后使用 while 循环对两个链表进行遍历，直到两个链表都遍历完并且进位为 0。在循环中计算当前位的和，并创建一个新节点存储当前位的值，然后将当前节点后移。最后返回虚拟头节点的下一个节点，即为相加后的链表。

14. 无重复字符的最长字串



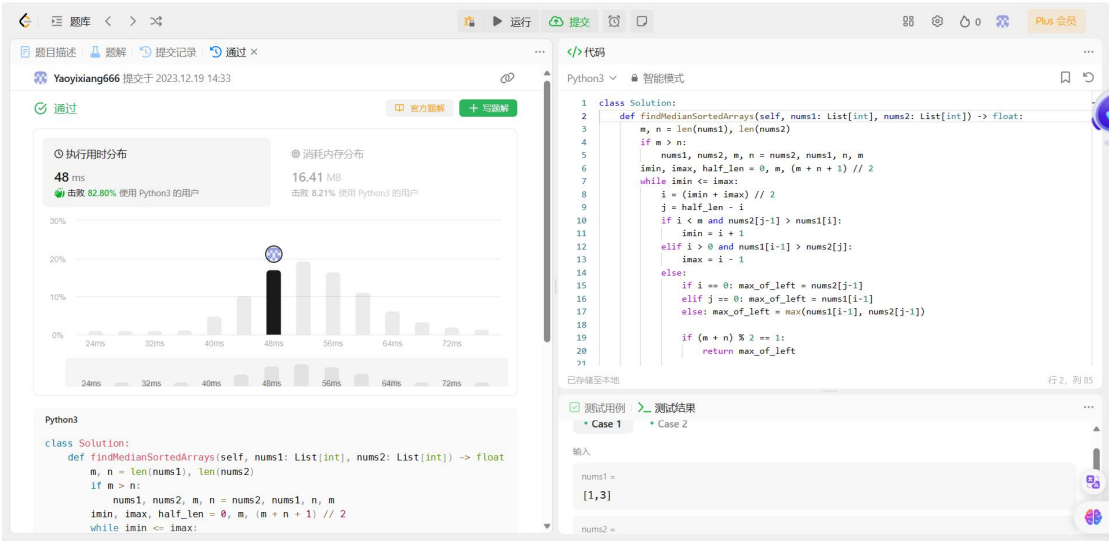
使用一个字典 last_seen 来存储每个字符最后出现的位置，并用 start 来表示当前最长子串的起始位置，max_length 来表示最长子串的长。然后用一个循环遍历字符串 s，对于每个字符检查它是否在 last_seen 中出现过，并且检查出现的位置是否在当前子串的起始位置之后。如果是，则更新起始位置为重复字符的下一个位置。在循环中更新 last_seen 中当前字符最后出现的位置，并根据当前子串的长度更新 max_length。最后返回 max_length 即为最长不含重复字符的子串的长度。

15. 最长回文字串



首先定义了一个名为 Solution 的类，其中包含了一个名为 longestPalindrome 的方法，并初始化了一个变量 max_str 为空字符串，用来存储找到的最长回文子串。然后如果输入的字符串长度为 1，则直接返回这个字符本身。接着使用两层循环遍历字符串 s，在内层循环中，获取当前遍历的子串 temp，然后判断 temp 是否为回文串，如果是回文串且长度大于 max_str，则更新 max_str 的值。最后返回找到的最长回文子串 max_str。

16. 寻找两个正序数组的中位数



首先需要确保 nums1 的长度不大于 nums2 的长度。如果不满足这个条件则交换两个数组，使得 nums1 的长度小于等于 nums2 的长度。然后定义 imin 和 imax 分别为 0 和 m，其中 m 为 nums1 的长度，定义了一个变量 half_len，其值为 (m + n + 1) // 2，其中 n 为 nums2 的长度。这里的 half_len 表示了两个数组合并后的长度的一半。接下来使用二分查找的方法来查找合适的分割点 i，使得 nums1[:i] 和 nums2[:j] 的长度之和等于 half_len。这里的 j = half_len - i。在每一次二分查找的循环中计算 i 和 j，然后根据 nums1[i-1]、nums1[i]、nums2[j-1] 和 nums2[j] 的值来调整 imin 和 imax 的值。最后找到合适的分割点 i 和 j 后，根据分割点将数组分成左右两部分，并计算出左半部分的最大值 max_of_left 和右半部分的最小值 min_of_right。如果两个数组合并后的长度为

奇数，则中位数就是 max_of_left；如果长度为偶数，则中位数就是 (max_of_left + min_of_right) / 2