* 1. 两数之和

难度：简单；知识点：数组、哈希表

给定一个整数数组和一个目标值，找出数组中和为目标值的两个数。

可以假设每个输入只对应一种答案，且同样的元素不能被重复利用。

|  |
| --- |
| 示例：给定 nums = [2, 7, 11, 15], target = 9  因为 nums[0] + nums[1] = 2 + 7 = 9  所以返回 [0, 1] |

直接的思路的使用双指针法：

|  |
| --- |
| *class* Solution:  *def* twoSum(*self*, *nums*, *target*):  n = len(nums)  for i in range(n - 1):  for j in range(1, n):  if nums[i] + nums[j] == target:  return [i, j] |

但是时间复杂度O(*n*2)，LeetCode直接超时了...

根据提示，使用哈希表，也就是Python中的字典。

将前i个数作为键，下标作为值存入字典d，如果target - nums[i]在字典中，说明找到，返回[d[target - nums[i]], i]。

如nums = [3, 2, 4, 2]，target = 4的情况，i=1时nums[i]=2, 而4-2=2的确在d中，但都是num[i]，即同一个数，返回[1,1]。这样不符合要求，所以要判断另一个数下标和当前数下标i是否相同。

|  |
| --- |
| *class* Solution:  *def* twoSum(*self*, *nums*, *target*):  d = {} # 数字为key, 下标为value  for i in range(len(nums)):  if not nums[i] in d:  d[nums[i]] = i  if target - nums[i] in d:  # 可能出现nums[i]\*2=target的情况,此时j=i,不符合;  # 所以另一个下标j要在i之前  j = d[target - nums[i]]  if j < i:  return [j, i] |

* 2. 两数相加

难度：中等；知识点：链表、数学

给定两个非空链表表示两个非负整数。位数按照逆序方式存储，它们的每个结点只存储单个数字。将两数相加返回一个新的链表。

可以假设除了数字0之外，这两个数字都不会以零开头。

|  |
| --- |
| 示例：输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)；输出：7 -> 0 -> 8  原因：342 + 465 = 807 |

如2+5=7，结果头结点为7；4+6=10，要对10求余，第2个结点为0，还要进1；3+4+1=8，所以最后个结点是8。结果是7->0->8。

用flag记录进位是0或1，第i位是两个链表i位相加+flag再对10求余。当一个链表已经结束，将另一个链表剩余添加。

|  |
| --- |
| from hikari\_tool import ListNode, SingleLinkedList  *class* Solution:  *def* addTwoNumbers(*self*, *l1*, *l2*):  if l1 is None:  return l2  if l2 is None:  return l1  s = l1.val + l2.val  head = ListNode(s % 10)  flag = s // 10 # 是否进位,0或1  cur = head # 新链表的工作指针  while l1.next and l2.next:  s = l1.next.val + l2.next.val + flag  cur.next = ListNode(s % 10)  flag = s // 10  l1, l2, cur = l1.next, l2.next, cur.next  # 其中有一个链表遍历结束,将另一个链表遍历完  while l1.next:  s = l1.next.val + flag  cur.next = ListNode(s % 10)  flag = s // 10  l1, cur = l1.next, cur.next  while l2.next:  s = l2.next.val + flag  cur.next = ListNode(s % 10)  flag = s // 10  l2, cur = l2.next, cur.next  if flag: # 可能进位  cur.next = ListNode(flag)  return head  if \_\_name\_\_ == '\_\_main\_\_':  l1 = SingleLinkedList([8, 5]) # 代表数字58  l2 = SingleLinkedList([7, 5]) # 代表数字57  l3 = SingleLinkedList()  l3.head = Solution().addTwoNumbers(l1.head, l2.head)  l3.travel() # [5, 1, 1], 代表数字115 |

LeetCode参考答案C++版本改编：

|  |
| --- |
| *def* addTwoNumbers\_2(*self*, *l1*, *l2*):  # 官方答案,好简洁...  # 结果这个用时180ms, 自己写的用时160ms...  head = ListNode(0) # 随意创建一个头结点,方便处理  p = head # 工作指针  carry = 0 # 进位  while l1 or l2 or carry:  s = (l1.val if l1 else 0)+(l2.val if l2 else 0)+carry  carry = s//10  p.next = ListNode(s % 10)  p = p.next  l1 = l1.next if l1 else l1  l2 = l2.next if l2 else l2  return head.next |

* 3. 无重复字符的最长子串

难度：中等；知识点：哈希表、双指针、字符串

给定一个字符串，找出不含有重复字符的最长子串的长度。

|  |
| --- |
| 示例：  给定"abcabcbb"，没有重复字符的最长子串是"abc"，那么长度就是3；  给定"bbbbb"，最长子串就是"b"，长度是1；  给定"pwwkew"，最长子串是"wke"，长度是3。  请注意答案必须是一个子串，"pwke"是子序列，而不是子串。 |

|  |
| --- |
| *class* Solution:  *def* lengthOfLongestSubstring(*self*, *s*):  if len(s) < 2:  return len(s)  max\_len = 0  start = 0 # 记录不重复字符串首位  d = {} # 每个字符为key, 下标为value  for i in range(len(s)):  # 如果s[i]在start~i不重复(0~i不重复或重复位置<start), 随时更新最大长度  if s[i] not in d or d[s[i]] < start:  max\_len = max(max\_len, i - start + 1)  else: # 重复则start移到与s[i]重复字符后一位  start = d[s[i]] + 1  d[s[i]] = i # 更新字符对应下标  return max\_len  if \_\_name\_\_ == '\_\_main\_\_':  s = Solution()  print(s.lengthOfLongestSubstring('abcabcbb')) # 3  print(s.lengthOfLongestSubstring('bbbbb')) # 1  print(s.lengthOfLongestSubstring('pwwkew')) # 3  print(s.lengthOfLongestSubstring('aab')) # 2  print(s.lengthOfLongestSubstring('abb')) # 2  print(s.lengthOfLongestSubstring('dedf')) # 3  print(s.lengthOfLongestSubstring('abba')) # 2 |

一开始写成判断字符重复，更新前面不重复字符串的最大长度；但遇到后面没有重复的就要出循环再判断。应该遇到不重复字符就随时更新最大长度。

还有一个坑，如果if不写d[s[i]] < start，例如'abbac'，i=2时start=d['b']+1=2；i=3时start=d['a']+1=1；i=4时i-start+1=4，则max\_len=4，然而真实结果是3。

因为start=2时，第1个a已经不在考虑的子串范围了，也就是d['a']<start，所以当d[s[i]]<start时与s[i]重复的字符已经不在考虑范围，当前字符s[i]是可以被收录到不重复子串，需要更新最大长度。