

## 双指针

```
In [ ]: 1 # 59. 螺旋矩阵 II
2 class Solution:
3     def generateMatrix(self, n: int) -> List[List[int]]:
4         maxtrix = [[1]*n for _ in range(n)]
5         left, right, up, down = 0, n-1, 0, n-1 # 上面的坐标会比下面的坐标小
6         num = 1
7         # 左闭右开
8         while left < right and up < down:
9             # 方阵, 最终结果会相等
10            for i in range(left, right):
11                maxtrix[up][i] = num
12                num += 1
13            for i in range(up, down):
14                maxtrix[i][right] = num
15                num += 1
16            for i in range(right, left, -1):
17                maxtrix[down][i] = num
18                num += 1
19            for i in range(down, up, -1):
20                maxtrix[i][left] = num
21                num += 1
22            # 走完一圈后再更新
23            left += 1; right -= 1; up += 1; down -= 1
24            if left == right:
25                # 在相等时是不会有输出的 range(left, left) == None
26                # 之后会left=right=up=down
27                maxtrix[up][left] = num
28            return maxtrix
29 # 54. 螺旋矩阵
30 class Solution:
31     def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
32         # 旋转打印数组, 不是方阵比较麻烦
33         m = len(matrix)
34         n = len(matrix[0])
35         left, right, up, down = 0, n-1, 0, m-1
36         result = []
37         while left < right and up < down:
38             # 不是方阵不会同时满足
39             for i in range(left, right):
40                 result.append(matrix[up][i])
41             for i in range(up, down):
42                 result.append(matrix[i][right])
43             for i in range(right, left, -1):
44                 result.append(matrix[down][i])
45             for i in range(down, up, -1):
46                 result.append(matrix[i][left])
47             left += 1; right -= 1; up += 1; down -= 1
48
49         # 特殊处理, 非方阵
50         if left == right: # 剩下一列, 从上到下依次添加
51             for i in range(up, down+1):
52                 result.append(matrix[i][left])
53         elif up == down:
54             for i in range(left, right+1):
55                 result.append(matrix[up][i])
56         return result
57 # 977. 有序数组的平方
58 class Solution:
59     def sortedSquares(self, nums: List[int]) -> List[int]:
60         # 双指针
61         left, right = 0, len(nums)-1
62         result = []
63         while left <= right:
64             if nums[left]**2 <= nums[right]**2:
65                 result.append(nums[right]**2)
66                 right -= 1
67             else:
68                 result.append(nums[left]**2)
69                 left += 1
70         return result[::-1]
71 # 27. 移除元素
```

```

72 class Solution:
73     def removeElement(self, nums: List[int], val: int) -> int:
74         # 判断中不含有i - 先交换再走i, 长度为i
75         i = 0
76         for j in range(len(nums)):
77             if nums[j] != val:
78                 nums[i] = nums[j]
79                 i += 1
80         return i
81 # 26. 删除有序数组中的重复项
82 class Solution:
83     def removeDuplicates(self, nums: List[int]) -> int:
84         # 判断中含有i, i先行再交换, 长度i+1
85         i = 0
86         for j in range(len(nums)):
87             if nums[j] != nums[i]:
88                 i += 1
89                 nums[i] = nums[j]
90         return i+1
91
92 # 283. 移动零
93 class Solution:
94     def moveZeroes(self, nums: List[int]) -> None:
95         """
96         Do not return anything, modify nums in-place instead.
97         """
98         # 判断中含有i, i先行再交换, 长度i+1
99         i = 0
100        for j in range(len(nums)):
101            if nums[j] != 0:
102                nums[i] = nums[j]
103                i += 1
104        for k in range(i, len(nums)):
105            nums[k] = 0
106        return nums
107
108 # 844. 比较含退格的字符串
109 class Solution:
110     def backspaceCompare(self, s: str, t: str) -> bool:
111         def backspace(s):
112             # 不含有i, 先交换, 再走i
113             s = list(s)
114             i = 0
115             for j in range(len(s)):
116                 if s[j] == '#':
117                     i -= 1
118                     i = max(i, 0)
119                 else:
120                     s[i] = s[j]
121                     i += 1
122             return ''.join(s[:i])
123         return True if backspace(s) == backspace(t) else False

```

## #反转链表

```
In [ ]: 1 class Solution:
2     def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
3         length = n-m+1 # 逆的长度
4         prehead = None # 逆段前驱
5         result = head
6         for i in range(m-1): # 当m == 1, 不会操作, prehead = None
7             prehead = head
8             head = head.next
9         # 此时head为逆置开始
10        # 接下来和整段逆置一样
11        last_head = head # 现在的头, 逆置后的尾
12        newhead = None # 转置后的头
13        pTmp = head.next
14        while length > 0 and pTmp:
15            head.next = newhead
16            newhead = head
17            head = pTmp
18            pTmp = pTmp.next
19            length -= 1
20        # 此时head为后驱的头
21        ##### 没有后驱情况
22        if length == 1:
23            head.next = newhead
24            if prehead:
25                prehead.next = head
26            return result
27            return head
28        #####
29        last_head.next = head
30        if prehead:
31            prehead.next = newhead
32        else:
33            result = newhead # 没有前驱情况
34        return result
35
36 class Solution:
37     def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
38         successor = None
39         def reverseN(head, n):
40             nonlocal successor
41             if n == 1:
42                 successor = head.next
43             return head
44             last = reverseN(head.next, n-1)
45             head.next.next = head
46             head.next = successor
47             return last
48         if left == 1:
49             return reverseN(head, right)
50         head.next = self.reverseBetween(head.next, left-1, right-1)
51         return head
```

```
In [16]: 1 from collections import defaultdict
2     def numberOfArithmeticSlices(self, nums: List[int]) -> int:
3         ans = 0
4         f = [defaultdict(int) for _ in nums]
5         for i, x in enumerate(nums):
6             for j in range(i):
7                 d = x - nums[j]
8                 cnt = f[j][d]
9                 ans += cnt
10                f[i][d] += cnt + 1
11        return ans
```

```
Out[16]: [defaultdict(int, {1: 0, 2: 0}),
defaultdict(int, {1: 1}),
defaultdict(int, {2: 1, 1: 2})]
```

## 链表

▶ In [ ]:

```
1 # 143. 重排链表
2 class Solution:
3     def reorderList(self, head: ListNode) -> None:
4         """
5         Do not return anything, modify head in-place instead.
6         """
7         slow, fast = head, head
8         while fast.next and fast.next.next:
9             slow = slow.next
10            fast = fast.next.next
11        cur = slow
12        newhead = None
13        while cur:
14            pTmp = cur.next
15            cur.next = newhead
16            newhead, cur = cur, pTmp
17        while head and newhead:
18            pTmp1 = head.next
19            pTmp2 = newhead.next
20            head.next = newhead
21            head = pTmp1
22            newhead.next = head
23            newhead = pTmp2
24
25 # 83. 删除排序链表中的重复元素
26 class Solution:
27     def deleteDuplicates(self, head: ListNode) -> ListNode:
28         cur = head
29         while cur and cur.next:
30             val = cur.val
31             if cur.next.val == val:
32                 cur.next = cur.next.next
33             else:
34                 cur = cur.next
35         return head
36
37 # 203. 移除链表元素
38 class Solution:
39     def removeElements(self, head: ListNode, val: int) -> ListNode:
40         # dummy = ListNode(next=head)
41         # cur = dummy
42         # while cur.next:
43         #     if cur.next.val == val:
44         #         cur.next = cur.next.next
45         #     else:
46         #         cur = cur.next
47         # return dummy.next
48         dummy = ListNode(next=head)
49         cur = dummy
50         while cur.next:
51             if cur.next.val == val:
52                 cur.next = cur.next.next
53             else:
54                 cur = cur.next
55         return dummy
56
57 # 206. 反转链表
58 class Solution:
59     def reverseList(self, head: ListNode) -> ListNode:
60         if not head or not head.next: return head
61         cur = head
62         newhead = None
63         while cur:
64             pTmp = cur.next
65             cur.next = newhead
66             newhead, cur = cur, pTmp
67         return newhead
68
69 # 24. 两两交换链表中的节点
70 class Solution:
71     def swapPairs(self, head: ListNode) -> ListNode:
```

```

72     # 两两交换
73     dummy = ListNode(0)
74     dummy.next = head
75     cur = dummy    # 形成一条新的链
76     while cur.next and cur.next.next:
77         temp = cur.next    # 待反转的第一个节点
78         temp1 = cur.next.next.next    # 跨越两个节点
79
80         cur.next = temp.next
81         cur.next.next = temp
82         cur.next.next.next = temp1
83         cur = cur.next.next    # 走两步，到上一次反转的结尾
84     return dummy.next
85
86 # 19. 删除链表的倒数第 N 个结点
87 class Solution:
88     def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
89         # 删除倒数第k个节点
90         # 快慢节点
91         # 加虚拟头，如果不加比较难处理只有一个头的问题
92         dummy = ListNode(next=head)
93         fastnode, slownode = dummy, dummy
94         for i in range(n+1):
95             fastnode = fastnode.next
96         while fastnode:
97             fastnode = fastnode.next
98             slownode = slownode.next
99         slownode.next = slownode.next.next
100        return dummy.next
101
102 # 707. 设计链表
103 class Node:
104
105     def __init__(self, val):
106         self.val = val
107         self.next = None
108
109 class MyLinkedList:
110
111     def __init__(self):
112         """
113         Initialize your data structure here.
114         """
115         # 一开始是一个只有一个虚拟头的链表，后面经过一系列操作生成
116         self._head = Node(0)    # 添加虚拟节点, 这里定义的头结点 是一个虚拟头结点，而不是真正的链表头结
117         self._count = 0    # 记录节点数, 链表长度（不包含虚拟头节点）, self._count会比最后一个节点的index
118
119     def get(self, index: int) -> int:
120         # 这个前提是已经 self._count != 0, 已经调用其他函数进行添加了。
121         if 0 <= index < self._count:
122             cur = self._head
123             for i in range(index+1):
124                 # 包含一个虚拟头节点，所以查找第 index+1 个节点
125                 # 解释加1: 由于第一个是虚拟节点，而index是从0开始的数，那么到节点第1(index)个话，se
126                 cur = cur.next
127             return cur.val
128         return -1
129
130     def addAtHead(self, val: int) -> None:
131         self.addAtIndex(0, val)
132
133     def addAtTail(self, val: int) -> None:
134         self.addAtIndex(self._count, val)
135
136     def addAtIndex(self, index: int, val: int) -> None:
137         # 边界条件处理
138         if index < 0: index = 0
139         if index > self._count: return    # 不做操作
140         self._count += 1
141
142         add_node = Node(val)
143         pre_node, cur_node = None, self._head

```

```

144         for _ in range(index+1):
145             pre_node, cur_node = cur_node, cur_node.next
146         else:
147             # 在pre_node 和 cur_node之间插入节点
148             pre_node.next, add_node.next = add_node, cur_node
149
150     def deleteAtIndex(self, index: int) -> None:
151         # 定义两个节点, 删除cur
152         pre_node, cur_node = None, self._head
153         if 0<=index < self._count:
154             self._count -= 1
155             for i in range(index+1):
156                 pre_node, cur_node = cur_node, cur_node.next
157             else:
158                 pre_node.next, cur_node.next = cur_node.next, None
159
160 # 141. 环形链表
161 class Solution:
162     def hasCycle(self, head: ListNode) -> bool:
163         slow = head
164         fast = head
165         while fast != None and fast.next != None:
166             slow = slow.next
167             fast = fast.next.next
168             if slow == fast :
169                 return True
170         return False
171
172 # 142. 环形链表 II
173 class Solution:
174     def detectCycle(self, head: ListNode) -> ListNode:
175         if head == None:
176             return None
177         show = head
178         fast = head
179         hasCycle = False
180         while fast != None and fast.next != None:
181             show = show.next
182             fast = fast.next.next
183             if show == fast:
184                 hasCycle = True
185                 break
186         if hasCycle:
187             slow = head
188             while slow != fast:
189                 slow = slow.next
190                 fast = fast.next
191             return fast
192         else:
193             return None
194
195 # 92. 反转链表 II
196 class Solution:
197     def __init__(self):
198         self.successor = None
199     def reverseN(self, head, n):
200         if n == 1:
201             self.successor = head.next
202             return head
203         last = self.reverseN(head.next, n-1)
204         head.next.next = head
205         head.next = self.successor
206         return last
207     def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
208         if left == 1:
209             return self.reverseN(head, right)
210         head.next = self.reverseBetween(head.next, left-1, right-1)
211         return head
212
213 # 25. K 个一组翻转链表
214 class Solution:
215     def reverse(self, a, b):

```

思路：快慢指针  
慢的走一步，快  
的走两步，如果  
有环存在，就会  
出现slow和  
fast相等

```

216         newhead = None
217         cur = a
218         while cur != b:
219             pTmp = cur.next
220             cur.next = newhead
221             newhead, cur = cur, pTmp
222         return newhead
223     def reverseKGroup(self, head, k):
224         if not head or not head.next: return head
225         a, b = head, head
226         for i in range(k):
227             if not b: return head
228             b = b.next
229         newhead = self.reverse(a, b)
230         a.next = self.reverseKGroup(b, k)
231         return newhead

```

## hash

In [ ]:

```

1 # 242. 有效的字母异位词
2 class Solution:
3     def isAnagram(self, s: str, t: str) -> bool:
4         s_dict = {i:s.count(i) for i in set(s)}
5         t_dict = {i:t.count(i) for i in set(t)}
6         return s_dict == t_dict
7         # return sorted(s) == sorted(t) # 一行代码即可
8
9 # 349. 两个数组的交集
10 class Solution:
11     def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
12         nums1 = set(nums1)
13         nums2 = set(nums2)
14         result = []
15         for i in nums1:
16             if i in nums2:
17                 result.append(i)
18         return result
19
20 # 202. 快乐数
21 class Solution:
22     def isHappy(self, n: int) -> bool:
23         _set = set() # 记录不重复的数
24         while True:
25             _sum = self.getSum(n)
26             if _sum == 1: return True
27             if _sum in _set: return False
28             else: _set.add(_sum) # set.add, 列表也可以
29             n = _sum
30     def getSum(self, n):
31         Sum = 0
32         while n > 0:
33             # 注意这个顺序, 先取//会造成520//10=52, 取了前好几位
34             Sum += (n%10)**2 #取最后一位
35             n //= 10 # 取去最后一位
36         return Sum

```

## nsum模板

```

1  def nSum(nums, n, start, target):
2      res = []
3      sz = len(nums)
4      if n < 2 or sz < n: return res
5      if n == 2:
6          lo = start; hi = sz-1
7          while lo < hi:
8              Sum = nums[lo]+nums[hi]
9              left = nums[lo];right = nums[hi]
10             if Sum < target:
11                 while lo < hi and nums[lo] == left: lo += 1
12             elif Sum > target:
13                 while lo < hi and nums[hi] == right: hi -= 1
14             else:
15                 res.append([left,right])
16                 while lo < hi and nums[lo] == left: lo += 1
17                 while lo < hi and nums[hi] == right: hi -= 1
18         else:
19             for i in range(start,sz):
20                 if i > start and nums[i] == nums[i-1]: continue
21                 sub = nSum(nums, n-1, i+1, target-nums[i])
22                 for arr in sub:
23                     arr.append(nums[i])
24                 res.append(arr)
25     return res

```



In [ ]:

```
1 # 1. 两数之和
2 class Solution:
3     def twoSum(self, nums: List[int], target: int) -> List[int]:
4         hashmap = {i:nums.index(i) for i in nums}
5         for i,num in enumerate(nums):
6             j = hashmap.get(target-num)
7             if j is not None and i != j:
8                 return [i,j]
9
10 # 15. 三数之和
11 class Solution:
12     def threeSum(self, nums: List[int]) -> List[List[int]]:
13         def nSum(n, target, start):
14             res = []
15             sz = len(nums)
16             if n < 2 and sz < n: return res
17             if n == 2:
18                 lo,hi = start,sz-1
19                 while lo < hi:
20                     left,right = nums[lo],nums[hi]
21                     Sum = nums[lo]+nums[hi]
22                     if Sum > target:
23                         while lo < hi and nums[hi] == right: hi -= 1
24                     elif Sum < target:
25                         while lo < hi and nums[lo] == left: lo += 1
26                     else:
27                         res.append([left,right])
28                         while lo < hi and nums[hi] == right: hi -= 1
29                         while lo < hi and nums[lo] == left: lo += 1
30             else:
31                 for i in range(start,sz):
32                     if i > start and nums[i] == nums[i-1]: continue
33                     sub = nSum(n-1, target-nums[i], i+1)
34                     for arr in sub:
35                         arr.append(nums[i])
36                     res.append(arr)
37             return res
38
39         nums.sort()
40         return nSum(3,0,0)
41
42 # 454. 四数相加 II
43 class Solution:
44     def fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int], nums4: List[int]) -
45         # A, B, C, D 具有相同的长度 N
46         hashmap = {}
47         for i in nums1:
48             for j in nums2:
49                 if i+j in hashmap:
50                     hashmap[i+j] += 1
51                 else:
52                     hashmap[i+j] = 1
53
54         count = 0
55         for i in nums3:
56             for j in nums4:
57                 key = -i-j
58                 if key in hashmap:
59                     count += hashmap[key] # 记所有次数
60         return count
61
62 # 18. 四数之和
63 class Solution:
64     def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
65         def nSum(nums,n, start, target):
66             res = []
67             sz = len(nums)
68             if n < 2 or sz < n: return res
69             if n == 2:
70                 lo = start; hi = sz-1
71                 while lo < hi:
72                     Sum = nums[lo]+nums[hi]
```

```
72         left = nums[lo];right = nums[hi]
73         if Sum < target:
74             while lo < hi and nums[lo] == left: lo += 1
75         elif Sum > target:
76             while lo < hi and nums[hi] == right: hi -= 1
77         else:
78             res.append([left,right])
79             while lo < hi and nums[lo] == left: lo += 1
80             while lo < hi and nums[hi] == right: hi -= 1
81     else:
82         for i in range(start,sz):
83             if i > start and nums[i] == nums[i-1]: continue
84             sub = nSum(nums,n-1,i+1,target-nums[i])
85             for arr in sub:
86                 arr.append(nums[i])
87                 res.append(arr)
88     return res
89 nums.sort()
90 return nSum(nums,4,0,target)
```