双指针

In [ ]:

```python
# 59. 螺旋矩阵 II
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        maxtrix = [[1]*n for _ in range(n)]
        left,right,up,down = 0,n-1,0,n-1  # 上面的坐标会比下面的坐标小
        num = 1
        # 左闭右开
        while left < right and up < down:
            # 方阵，最终结果会相等
            for i in range(left,right):
                maxtrix[up][i] = num
                num += 1
            for i in range(up,down):
                maxtrix[i][right] = num
                num += 1
            for i in range(right,left,-1):
                maxtrix[down][i] = num
                num += 1
            for i in range(down,up,-1):
                maxtrix[i][left] = num
                num += 1
            # 走完一圈后再更新
            left += 1; right -= 1; up += 1; down -= 1
            if left == right:
                # 在相等时是不会有输出的 range(left,left) == None
                # 之后会left=right=up=down
                maxtrix[up][left] = num
        return maxtrix
# 54. 螺旋矩阵
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        # 旋转打印数组,不是方阵比较麻烦
        m = len(matrix)
        n = len(matrix[0])
        left,right,up,down = 0,n-1,0,m-1
        result = []
        while left < right and up < down:
            # 不是方阵不会同时满足
            for i in range(left,right):
                result.append(matrix[up][i])
            for i in range(up,down):
                result.append(matrix[i][right])
            for i in range(right,left,-1):
                result.append(matrix[down][i])
            for i in range(down,up,-1):
                result.append(matrix[i][left])
            left += 1; right -= 1; up += 1; down -=1

        # 特殊处理，非方阵
        if left == right:  # 剩下一列，从上到下依次添加
            for i in range(up,down+1):
                result.append(matrix[i][left])
        elif up == down:
            for i in range(left,right+1):
                    result.append(matrix[up][i])
        return result
# 977. 有序数组的平方
class Solution:
    def sortedSquares(self, nums: List[int]) -> List[int]:
        # 双指针
        left,right = 0,len(nums)-1
        result = []
        while left <= right:
            if nums[left]**2 <= nums[right]**2:
                result.append(nums[right]**2)
                right -= 1
            else:
                result.append(nums[left]**2)
                left += 1
        return result[::-1]
# 27. 移除元素
```

```python
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        # 判断中不含有i - 先交换再走i,长度为i
        i = 0
        for j in range(len(nums)):
            if nums[j] != val:
                nums[i] = nums[j]
                i += 1
        return i
# 26. 删除有序数组中的重复项
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        # 判断中含有i，i先行再交换，长度i+1
        i = 0
        for j in range(len(nums)):
            if nums[j] != nums[i]:
                i += 1
                nums[i] = nums[j]
        return i+1

# 283. 移动零
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        # 判断中含有i，i先行再交换，长度i+1
        i = 0
        for j in range(len(nums)):
            if nums[j] != 0:
                nums[i] = nums[j]
                i += 1
        for k in range(i,len(nums)):
            nums[k] = 0
        return nums

# 844. 比较含退格的字符串
class Solution:
    def backspaceCompare(self, s: str, t: str) -> bool:
        def backspace(s):
            # 不含有i,先交换，再走i
            s = list(s)
            i = 0
            for j in range(len(s)):
                if s[j] == '#':
                    i -= 1
                    i = max(i,0)
                else:
                    s[i] = s[j]
                    i += 1
            return ''.join(s[:i])
        return True if backspace(s) == backspace(t) else False
```

#反转链表

In [ ]:

```python
class Solution:
    def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
        length = n-m+1  # 逆的长度
        prehead = None  # 逆段前驱
        result = head
        for i in range(m-1):  # 当m == 1,不会操作,prehead = None
            prehead = head
            head = head.next
        # 此时head为逆置开始
        # 接下来和整段逆置一样
        last_head = head  # 现在的头,逆置后的尾
        newhead = None  # 转置后的头
        pTmp = head.next
        while length > 0 and pTmp:
            head.next = newhead
            newhead = head
            head = pTmp
            pTmp = pTmp.next
            length -= 1
        # 此时head为后驱的头
        ############################################################ 没有后驱情况
        if length == 1:
                head.next = newhead
                if prehead:
                    prehead.next = head
                    return result
                 return head
        ############################################################
        last_head.next = head
        if prehead:
             prehead.next = newhead
        else:
            result = newhead  # 没有前驱情况
        return result

class Solution:
    def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
        successor = None
        def reverseN(head,n):
            nonlocal successor
            if n == 1:
                successor = head.next
                return head
            last = reverseN(head.next,n-1)
            head.next.next = head
            head.next = successor
            return last
        if left == 1:
            return reverseN(head,right)
        head.next = self.reverseBetween(head.next,left-1,right-1)
        return head
```

In [16]:

```python
from collections import defaultdict
def numberOfArithmeticSlices(self, nums: List[int]) -> int:
    ans = 0
    f = [defaultdict(int) for _ in nums]
    for i, x in enumerate(nums):
        for j in range(i):
            d = x - nums[j]
            cnt = f[j][d]
            ans += cnt
            f[i][d] += cnt + 1
    return ans
```

Out[16]:

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

链表

In [ ]:

```python
# 143. 重排链表
class Solution:
    def reorderList(self, head: ListNode) -> None:
        """
        Do not return anything, modify head in-place instead.
        """
        slow,fast = head,head
        while fast.next and fast.next.next:
            slow = slow.next
            fast = fast.next.next
        cur = slow
        newhead = None
        while cur:
            pTmp = cur.next
            cur.next = newhead
            newhead,cur = cur,pTmp
        while head and newhead:
            pTmp1 = head.next
            pTmp2 = newhead.next
            head.next = newhead
            head = pTmp1
            newhead.next = head
            newhead = pTmp2

# 83. 删除排序链表中的重复元素
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        cur = head
        while cur and cur.next:
            val = cur.val
            if cur.next.val == val:
                cur.next = cur.next.next
            else:
                cur = cur.next
        return head

# 203. 移除链表元素
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        # dummy = ListNode(next=head)
        # cur = dummy
        # while cur.next:
        #     if cur.next.val == val:
        #         cur.next = cur.next.next
        #     else:
        #         cur = cur.next
        # return dummy.next
        dummy = ListNode(next=head)
        cur = dummy
        while cur.next:
            if cur.next.val == val:
                cur.next = cur.next.next
            else:
                cur = cur.next
        return dummy

# 206. 反转链表
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        if not head or not head.next: return head
        cur = head
        newhead = None
        while cur:
            pTmp = cur.next
            cur.next = newhead
            newhead,cur = cur,pTmp
        return newhead

# 24. 两两交换链表中的节点
class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
```

```python
        # 两两交换
        dummy = ListNode(0)
        dummy.next = head
        cur = dummy    # 形成一条新的链
        while cur.next and cur.next.next:
            temp = cur.next  # 待反转的第一个节点
            temp1 = cur.next.next.next   # 跨越两个节点

            cur.next = temp.next
            cur.next.next = temp
            cur.next.next.next = temp1
            cur = cur.next.next  # 走两步，到上一次反转的结尾
        return dummy.next

# 19. 删除链表的倒数第 N 个结点
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        # 删除倒数第k个节点
        # 快慢节点
        # 加虚拟头，如果不加比较难处理只有一个头的问题
        dummy = ListNode(next=head)
        fastnode,slownode = dummy,dummy
        for i in range(n+1):
                fastnode = fastnode.next
        while fastnode:
            fastnode = fastnode.next
            slownode = slownode.next
        slownode.next = slownode.next.next
        return dummy.next

# 707. 设计链表
class Node:

    def __init__(self, val):
        self.val = val
        self.next = None

class MyLinkedList:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        # 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
        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
        self._count = 0  # 记录节点数,链表长度（不包含虚拟头节点）,self._count会比最后一个几点的inde

    def get(self, index: int) -> int:
        # 这个前提是已经 self._count != 0，  已经调用其他函数进行添加了。
        if 0<= index < self._count:
            cur = self._head
            for i in range(index+1):
                # 包含一个虚拟头节点，所以查找第 index+1 个节点
                # 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
                cur = cur.next
            return cur.val
        return -1

    def addAtHead(self, val: int) -> None:
        self.addAtIndex(0,val)

    def addAtTail(self, val: int) -> None:
        self.addAtIndex(self._count,val)

    def addAtIndex(self, index: int, val: int) -> None:
        # 边界条件处理
        if index < 0:  index = 0
        if index > self._count: return  # 不做操作
        self._count += 1

        add_node = Node(val)
        pre_node,cur_node = None,self._head
```

```
        for _ in range(index+1):
            pre_node,cur_node = cur_node,cur_node.next
        else:
            # 在pre_node 和 cur_node之间插入节点
            pre_node.next,add_node.next = add_node,cur_node

    def deleteAtIndex(self, index: int) -> None:
        # 定义两个节点，删除cur
        pre_node,cur_node = None,self._head
        if 0<=index < self._count:
            self._count -= 1
            for i in range(index+1):
                pre_node,cur_node = cur_node,cur_node.next
            else:
                pre_node.next,cur_node.next = cur_node.next,None

# 141. 环形链表
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        slow = head
        fast = head                                          思路：快慢指针
        while fast != None and fast.next != None:            慢的走一步，快
            slow = slow.next                                 的走两步，如果
            fast = fast.next.next                            有环存在，就会
            if slow == fast :                                出现slow和
                return True                                  fast相等
        return False

# 142. 环形链表 II
class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        if head == None:
            return None
        show = head
        fast = head
        hasCycle = False
        while fast != None and fast.next != None:
            show = show.next
            fast = fast.next.next
            if show == fast:
                hasCycle = True
                break
        if hasCycle:
            slow = head
            while slow != fast:
                slow = slow.next
                fast = fast.next
            return fast
        else:
            return None

# 92. 反转链表 II
class Solution:
    def __init__(self):
        self.successor = None
    def reverseN(self,head,n):
        if  n == 1:
            self.successor = head.next
            return head
        last = self.reverseN(head.next,n-1)
        head.next.next = head
        head.next = self.successor
        return last
    def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
        if left == 1:
            return self.reverseN(head,right)
        head.next = self.reverseBetween(head.next,left-1,right-1)
        return head

# 25. K 个一组翻转链表
class Solution:
    def reverse(self,a,b):
```

```
        newhead = None
        cur = a
        while cur != b:
            pTmp = cur.next
            cur.next = newhead
            newhead,cur = cur,pTmp
        return newhead
    def reverseKGroup(self,head,k):
        if not head or not head.next: return head
        a,b = head,head
        for i in range(k):
            if not b: return head
            b = b.next
        newhead = self.reverse(a,b)
        a.next = self.reverseKGroup(b,k)
        return newhead
```

hash

In [ ]:

```
# 242. 有效的字母异位词
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        s_dict = {i:s.count(i) for i in set(s)}
        t_dict = {i:t.count(i) for i in set(t)}
        return s_dict == t_dict
        # return sorted(s) == sorted(t)  # 一行代码即可

# 349. 两个数组的交集
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        nums1 = set(nums1)
        nums2 = set(nums2)
        result = []
        for i in nums1:
            if i in nums2:
                result.append(i)
        return result

# 202. 快乐数
class Solution:
    def isHappy(self, n: int) -> bool:
        _set = set() # 记录不重复的数
        while True:
            _sum = self.getSum(n)
            if _sum == 1: return True
            if _sum in _set: return False
            else: _set.add(_sum)  # set.add,列表也可以
            n = _sum
    def getSum(self,n):
            Sum = 0
            while n > 0:
                # 注意这个顺序，先取//会造成520//10=52，取了前好几位
                Sum += (n%10)**2  #取最后一位
                n //= 10  # 取去最后一位
            return Sum
```

nsum模板

In [ ]:

```python
def nSum(nums,n,start,target):
    res = []
    sz = len(nums)
    if n < 2 or sz < n: return res
    if n == 2:
        lo = start; hi = sz-1
        while lo < hi:
            Sum = nums[lo]+nums[hi]
            left = nums[lo];right = nums[hi]
            if Sum < target:
                while lo < hi and nums[lo] == left: lo += 1
            elif Sum > target:
                while lo < hi and nums[hi] == right: hi -= 1
            else:
                res.append([left,right])
                while lo < hi and nums[lo] == left: lo += 1
                while lo < hi and nums[hi] == right: hi -= 1
    else:
        for i in range(start,sz):
            if i > start and nums[i] == nums[i-1]: continue
            sub = nSum(nums,n-1,i+1,target-nums[i])
            for arr in sub:
                arr.append(nums[i])
                res.append(arr)
    return res
```

In [ ]:

```python
# 1. 两数之和
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap = {i:nums.index(i) for i in nums}
        for i,num in enumerate(nums):
            j = hashmap.get(target-num)
            if j is not None and i != j:
                return [i,j]
# 15. 三数之和
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        def nSum(n,target,start):
            res = []
            sz = len(nums)
            if n < 2 and sz < n: return res
            if n == 2:
                lo,hi = start,sz-1
                while lo < hi:
                    left,right = nums[lo],nums[hi]
                    Sum = nums[lo]+nums[hi]
                    if Sum > target:
                        while lo < hi and nums[hi] == right: hi -= 1
                    elif Sum < target:
                        while lo < hi and nums[lo] == left: lo += 1
                    else:
                        res.append([left,right])
                        while lo < hi and nums[hi] == right: hi -= 1
                        while lo < hi and nums[lo] == left: lo += 1
            else:
                for i in range(start,sz):
                    if i > start and nums[i] == nums[i-1]: continue
                    sub = nSum(n-1,target-nums[i],i+1)
                    for arr in sub:
                        arr.append(nums[i])
                        res.append(arr)
            return res

        nums.sort()
        return nSum(3,0,0)

# 454. 四数相加 II
class Solution:
    def fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int], nums4: List[int]) -
        # A, B, C, D 具有相同的长度 N
        hashmap = {}
        for i in nums1:
            for j in nums2:
                if i+j in hashmap:
                    hashmap[i+j] += 1
                else:
                    hashmap[i+j] = 1

        count = 0
        for i in nums3:
            for j in nums4:
                key = -i-j
                if key in hashmap:
                    count += hashmap[key]  # 记所有次数
        return count

# 18. 四数之和
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        def nSum(nums,n,start,target):
            res = []
            sz = len(nums)
            if n < 2 or sz < n: return res
            if n == 2:
                lo = start; hi = sz-1
                while lo < hi:
                    Sum = nums[lo]+nums[hi]
```

第一种

第二种

为了去重

```
                    left = nums[lo];right = nums[hi]
                    if Sum < target:
                        while lo < hi and nums[lo] == left: lo += 1
                    elif Sum > target:
                        while lo < hi and nums[hi] == right: hi -= 1
                    else:
                        res.append([left,right])
                        while lo < hi and nums[lo] == left: lo += 1
                        while lo < hi and nums[hi] == right: hi -= 1
            else:
                for i in range(start,sz):
                    if i > start and nums[i] == nums[i-1]: continue
                    sub = nSum(nums,n-1,i+1,target-nums[i])
                    for arr in sub:
                        arr.append(nums[i])
                        res.append(arr)
            return res
        nums.sort()
        return nSum(nums,4,0,target)
```