LeetCode 题解 (Python 版本)

胡欣毅 (icedomain_hu@qq.com)
https://github.com/Icedomain/LeetCode

最后更新 April 23, 2020

本文档一共统计了 283 道题

```
#
# @lc app=leetcode.cn id=1 lang=python3
#
# [1] 两数之和
#
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dic = {}
        for i in range(len(nums)):
            if target - nums[i] in dic :
                return [dic[target-nums[i]] , i ]
            dic[nums[i]] = i
```

```
#
# @lc app=leetcode.cn id=2 lang=python3
#
# [2] 两数相加
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        jingwei = 0
        # 两个空指针 n后面要被覆盖的
        head = n = ListNode(0)
        while l1 or l2 or jingwei:
            v1 = v2 = 0
            if l1:
                v1 = l1.val
                l1 = l1.next
            if l2:
                v2 = l2.val
                l2 = l2.next
            # 除数、余数
            val = (v1+v2+jingwei) % 10
            jingwei = (v1+v2+jingwei) // 10
            n.next = ListNode(val)
            # 指向下一个
            n = n.next
        return head.next # 记得把第一个0去掉
```

```
#
# @lc app=leetcode.cn id=3 lang=python3
#
# [3] 无重复字符的最长子串
#
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # 记录表 256个字符
        dic = {}

        start = maxlen = 0
        # 遍历 滑动窗 [start,j] j往右边移动 若遇到重复的 start又移一位
        for j in range(len(s)):
            # 如果这个字符出现过了，又移动 最左边那个踢出滑动窗
            if s[j] in dic and dic[s[j]] >= start:
                start = dic[s[j]] + 1
            # 如果这个字符在滑动窗中没出现过，位置给它(出现过也要给它)
            dic[s[j]] = j
            maxlen = max(maxlen, j-start +1)
        return maxlen
```

```
#
# @lc app=leetcode.cn id=4 lang=python3
#
# [4] 寻找两个有序数组的中位数
#
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
        leng = len(nums1)+len(nums2)
        if leng%2 == 1:#奇数
            return self.findk(nums1,nums2,leng//2)
        else:
            return (self.findk(nums1,nums2,leng//2-1)+self.findk(nums1,nums2,leng//2))/2.0
    # 找k大的数
    def findk(self,nums1,nums2,k):
        if not nums1:
            return nums2[k]
        if not nums2:
            return nums1[k]
        l1 , l2 = len(nums1)//2,len(nums2)//2
        val1 ,val2 = nums1[l1],nums2[l2]

        if l1+l2<k:# 往右找
            if val1 > val2:
                return self.findk(nums1, nums2[l2 + 1:], k - l2 - 1)
            else:
                return self.findk(nums1[l1 + 1:],nums2, k - l1 - 1)
        else: # 往左找
            if val1 > val2:
                return self.findk(nums1[:l1],nums2, k)

            else:
                return self.findk(nums1, nums2[:l2], k)
```

```
#
# @lc app=leetcode.cn id=5 lang=python3
#
# [5] 最长回文子串
#
class Solution:
    def longestPalindrome(self, s: str) -> str:
        if s is None:
            return None

        # 动态规划
        dp = [[0 for _ in range(len(s))] for _ in range(len(s))]
        left , right , max_len = 0, 0, 0

        for j in range(len(s)):
            # 对角线置1
            dp[j ][ j ] = 1
            for i in range(j-1,-1,-1):
                if s[ i ] == s[j] and (j-i < 2 or dp[i+1][j-1]):
                    dp[i ][ j ] = 1
                if dp[i ][ j ] and max_len < j-i+1:
                    max_len = j - i + 1
                    left , right = i, j
        return s[left : right+1]
```

```
#
# @lc app=leetcode.cn id=6 lang=python3
#
```

```
# [6] Z 字形变换
#
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows == 1 or numRows >= len(s):
            return s
        # z前半个(|/)个数两行减2
        p = 2 * (numRows - 1)

        result = [""] * numRows
        for i in range(len(s)):
            floor = i % p # 一个形状轮回的位置
            if floor >= p//2: # 在/上
                floor = p - floor
            result[floor] += s[i]
        return "".join(result)
```

```
#
# @lc app=leetcode.cn id=7 lang=python3
#
# [7] 整数反转
#
class Solution:
    def reverse(self, x: int) -> int:
        sign = 1 if x > 0 else -1
        res = 0
        x = abs(x)
        while x :
            res = res*10 + x%10
            if res > 2**31 - 1:
                return 0
            x = x//10

        return sign * res
```

```
#
# @lc app=leetcode.cn id=8 lang=python3
#
# [8] 字符串转换整数 (atoi)
#
class Solution:
    def myAtoi(self, str: str) -> int:
        # 去空格
        str = str.strip()
        if len(str) == 0:
            return 0
        sign = 1
        if str[0] == ’+’ or str[0] == ’−’:
            if str[0] == ’−’:
                sign = −1
            str = str[1:]
        res = 0
        for char in str:
            if char >= ’0’ and char <= ’9’:
                res = res ∗ 10 + ord(char) − ord(’0’)
            if char < ’0’ or char > ’9’:
                break
        return max(−2∗∗31, min(sign ∗ res,2∗∗31−1))
```

```
#
# @lc app=leetcode.cn id=9 lang=python3
#
# [9] 回文数
#
class Solution:
    def isPalindrome(self, x: int) −> bool:
        if x < 0 :
            return False
        # 最高位的位数
        d = 1
        while x // d >= 10:
            d ∗= 10
        while x > 0:
            # p q 对应最高位和最低位
            p = x //d
            q = x % 10
            if p != q :
                return False
            # x 去掉最高位,去掉最低位
            x = x % d // 10
            # x 去掉了两位,d也减两位
            d //= 100
        return True
```

```
#
# @lc app=leetcode.cn id=10 lang=python3
#
# [10] 正则表达式匹配
#
class Solution:
    def isMatch(self, s: str, p: str) −> bool:
        ’’’
        # 递归写法
        # s已被匹配且p已耗完
        if not s and not p:
            return True
        # p已耗完但s未被完全匹配
        if len(s) > 0 and len(p) == 0:
            return False

        # 如果模式第二个字符是*
        if len(p) > 1 and p[1] == '*':
            if len(s) > 0 and (s[0] == p[0] or p[0] == '.'): # ax a* or ax .*
                # 如果第一个字符匹配，三种可能1、p后移两位；2、字符串移1位
                return self.isMatch(s, p[2:]) or self.isMatch(s[1:], p)
            else:
                # 如果第一个字符不匹配，p往后移2位，相当于忽略x*
                return self.isMatch(s, p[2:])
        # 如果模式第二个字符不是*
        if len(s) > 0 and (s[0] == p[0] or p[0] == '.'):
            return self.isMatch(s[1:], p[1:])
        else:
            return False
         '''
        # 动态规划
        # 初始化dp表，初始化表的第一列和第一行
        # p对应列 s对应行
        dp = [[False for j in range(len(p) + 1)] for i in range(len(s) + 1)]
        dp[0][0] = True # s 和 p 都为空时
        # 若 s 为空时
        # 处理第一行
        # p 与 dp 有一位的错位(多了一个空导致的)
        for j in range(1, len(p) + 1):
            # dp[0][j]= (p[j-1]=="*")and(j>=2)and(dp[0][j-2])
            # 等同于下列语句
            if p[j - 1] == '*':
                if j >= 2:
                    dp[0][j] = dp[0][j - 2]
        # 第一列就第一个是 True,下面都是 False
        # 不用处理 pass

        for i in range(1, len(s) + 1):
            for j in range(1, len(p) + 1):
                # j-1才为正常字符串中的索引
                # p当前位置为"*"时
                # 代表空串--dp[i][j-2]
                # 一个或者多个前一个字符--( dp[i-1][j] and (p[j-2]==s[i-1] or p[j-2]=='.')
                if p[j - 1] == '*':
                    dp[i][j] = dp[i][j - 2] or (
                                dp[i - 1][j] and (p[j - 2] == s[i - 1] or p[j - 2] == '.')
                                )
                # p当前位置为"."时或者与s相同时，传递dp[i-1][j-1]的真值
                else:
                    dp[i][j] = (p[j - 1] == '.' or p[j - 1] == s[i - 1]) and dp[i - 1][j - 1]
        return dp[-1][-1]
```

```
#
# @lc app=leetcode.cn id=11 lang=python3
#
# [11] 盛最多水的容器
#
class Solution:
    def maxArea(self, height: List[int]) -> int:
        max_area = 0
        left, right = 0,len(height) - 1
        while left < right :
            # 高取左边和右边的高当中的最小值，下标right-left为宽，两者相乘为面积
            temp = min(height[left], height[right]) * (right - left)
            max_area = max(max_area, temp)
            # 判断哪条高小，小的那边下标进行操作
            if height[right] > height[left]:
                left += 1
            else:
                right -= 1
        return max_area
```

```
#
# @lc app=leetcode.cn id=12 lang=python3
#
# [12] 整数转罗马数字
#
class Solution:
    def intToRoman(self, num: int) -> str:
        # 贪心算法
        dic = {
            'M': 1000,
            'CM': 900, 'D': 500, 'CD': 400, 'C': 100,
            'XC': 90, 'L': 50, 'XL': 40, 'X': 10,
            'IX': 9, 'V': 5, 'IV': 4, 'I': 1,
        }
        result = ""
        for letter,number in dic.items():
            if num >= number:
                result += letter*(num//number)
                num %= number
        return result
```

```
#
# @lc app=leetcode.cn id=13 lang=python3
#
# [13] 罗马数字转整数
#
class Solution:
    def romanToInt(self, s: str) -> int:
        dicts = {
            ”I”: 1,
            ”V”: 5,
            ”X”: 10,
            ”L”: 50,
            ”C”: 100,
            ”D”: 500,
            ”M”: 1000
        }
        s = s.replace(”IV”, ”IIII”).replace(”IX”, ”VIIII”)
        s = s.replace(”XL”, ”XXXX”).replace(”XC”, ”LXXXX”)
        s = s.replace(”CD”, ”CCCC”).replace(”CM”, ”DCCCC”)
        data = 0
        for item in s:
            data += dicts[item]
        return data
```

```
#
# @lc app=leetcode.cn id=14 lang=python3
#
# [14] 最长公共前缀
#
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        ’’’
        sz = zip(*strs)
        ret = ””
        for char in sz:
            if len(set(char)) > 1:
                break
            ret +=char[0]
        return ret
        ’’’
        if not strs:
            return ’’
        strs.sort(key = lambda x : len(x))
        for idx in range(len(strs[0])):
            # 最大的可能长度就是第一个的长度
            for i in range(1,len(strs)):
                # 对每个元素都要遍历
                if strs[i][idx] != strs[0][idx]:
                    return strs[0][:idx]
        return strs[0]
```

```
#
# @lc app=leetcode.cn id=15 lang=python3
#
# [15] 三数之和
#
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        res = []
        for i in range(len(nums)-2):
            if i >0 and nums[i] == nums[i-1]:
                continue
            l,r = i+1,len(nums) - 1
            while l < r:
                s = nums[i]+nums[l]+nums[r]
                if s < 0:
                    l+=1
                elif s > 0:
                    r -= 1
                else:
                    res.append((nums[i], nums[l], nums[r]))
                    # 避免一样的加进去
                    while l < r and nums[l] == nums[l+1]:
                        l += 1
                    while l < r and nums[r] == nums[r-1]:
                        r -= 1
                    l += 1
                    r -= 1
        return res
```

```
#
# @lc app=leetcode.cn id=16 lang=python3
#
# [16] 最接近的三数之和
#
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
        nums.sort()
        res = sum(nums[0:3])

        for i in range(len(nums)-2):
            l,r = i+1,len(nums)-1
            while l < r:
                sum_val = nums[i]+nums[l]+nums[r]
                if sum_val == target:
                    return sum_val
                if abs(res-target)>abs(sum_val-target):
                    res = sum_val
                if sum_val < target:
                    l+=1
                else:
                    r -= 1
        return res
```

```
#
# @lc app=leetcode.cn id=17 lang=python3
#
# [17] 电话号码的字母组合
#
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        dmap = {
        ’2’: ’abc’,
        ’3’: ’def’,
        ’4’: ’ghi’,
        ’5’: ’jkl’,
        ’6’: ’mno’,
        ’7’: ’pqrs’,
        ’8’: ’tuv’,
        ’9’: ’wxyz’
        }
        if len(digits) == 0:
            return []
        if len(digits) == 1:
            return list(dmap[digits])
        prev = self.letterCombinations(digits[:-1])
        additional = dmap[digits[-1]]
        return [s + c for s in prev for c in additional]
```

```
#
# @lc app=leetcode.cn id=18 lang=python3
#
# [18] 四数之和
#
```

```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        res = []
        # 去除异常
        if not nums or len(nums) < 4:
            return res
        nums.sort()
        # 第一个数遍历
        for i in range(len(nums) - 3):
            if i > 0 and nums[i] == nums[i - 1]:
                continue
            # 第二个数遍历
            for j in range(i + 1, len(nums) - 2):
                if j > i + 1 and nums[j] == nums[j - 1]:
                    continue
                # 双指针
                L, R = j + 1, len(nums) - 1
                while L < R:
                    if nums[i] + nums[j] + nums[L] + nums[R] == target:
                        res.append([nums[i], nums[j], nums[L], nums[R]])
                        while L < R and nums[L] == nums[L + 1]:
                            L += 1
                        while L < R and nums[R] == nums[R - 1]:
                            R -= 1
                        L += 1
                        R -= 1
                    elif nums[i] + nums[j] + nums[L] + nums[R] < target:
                        L += 1
                    else:
                        R -= 1
        return res
```

```python
#
# @lc app=leetcode.cn id=19 lang=python3
#
# [19] 删除链表的倒数第N个节点
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        if head is None:
            return None
        dummy = ListNode(-1)
        dummy.next = head
        slow = fast = dummy
        # 先走n步
        for i in range(n):
            fast = fast.next

        # slow 少走n步
        while fast.next :
            fast = fast.next
            slow = slow.next
        # 删除
        slow.next = slow.next.next
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=20 lang=python3
#
# [20] 有效的括号
#
class Solution:
    def isValid(self, s: str) -> bool:
        # 判断是否是奇数或空字符
        if s=='':
            return True
        stack = []
        match = {')': '(', ']': '[', '}': '{'}
        for ch in s:
            if ch in match:
                if not (stack and stack.pop() == match[ch]):
                    return False
            else:
                stack.append(ch)
        return not stack

        '''
        if len(s) %2 != 0:
            return False
        count = 0
        leng = len(s)
        # 将其中的(){}[] 都换掉，然后判断是否有剩余
        while(count < leng/2 ):
            s = s.replace("{}","").replace ("[]","") .replace ("()","")
            count+=1

        if len(s)>0:
            return False
        else:
            return True
        '''
```

```
#
# @lc app=leetcode.cn id=21 lang=python3
#
# [21] 合并两个有序链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummy = now = ListNode(-1)
        while l1 and l2:
            if l1.val <= l2.val:
                now.next = l1
                l1 = l1.next
            else:
                now.next = l2
                l2 = l2.next
            now = now.next
        now.next = l1 or l2
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=22 lang=python3
#
# [22] 括号生成
#
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        if n > 0 :
            self.dfs(n,'',res,0,0)
        return res

    def dfs(self,n,path,res, left, right):
        # 终止条件
        if len(path) == 2 * n:
            res.append(path)
            return
        # 左括号(够了没
        if left < n:
            self.dfs(n,path+'(',res, left+1, right)
        # 右括号补成和左括号一样多
        if left > right:
            self.dfs(n,path+')',res, left, right+1)
```

```
#
# @lc app=leetcode.cn id=23 lang=python3
#
# [23] 合并K个排序链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        if not lists:
            return None
        return self.mergeK(lists, 0, len(lists) - 1)

    def mergeK(self, lists, low, high):
        if low == high:
            return lists[low]
        elif low + 1 == high:
            return self.mergeTwolists(lists[low], lists[high])
        mid = (low + high) // 2
        return self.mergeTwolists(self.mergeK(lists, low, mid), self.mergeK(lists, mid + 1, high))

    def mergeTwolists(self, l1, l2):
        if l1 is None:
            return l2
        if l2 is None:
            return l1
        head = curr = ListNode(-1)
        while l1 and l2:
            if l1.val <= l2.val:
                curr.next = l1
                l1 = l1.next
            else:
                curr.next = l2
                l2 = l2.next
            curr = curr.next
        curr.next = l1 or l2
        return head.next
```

```
#
# @lc app=leetcode.cn id=24 lang=python3
#
# [24] 两两交换链表中的节点
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
        prev = dummy = ListNode(-1)
        dummy.next = head
        while prev.next and prev.next.next :
            # prev a b -> prev b a (交换a,b)
            a = prev.next
            b = prev.next.next
            prev.next, b.next, a.next = b, a, b.next
            prev = a
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=25 lang=python3
#
# [25] K 个一组翻转链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
        if head is None or k < 2:
            return head
        dummy = ListNode(0)
        dummy.next = head
        start = dummy
        end = start.next

        count = 0
        while end:
            count += 1
            if count % k == 0:
                # 返回为新一轮的头
                start = self.reverse(start, end.next)
                end = start.next
            else:
                end = end.next
        return dummy.next

    def reverse(self, start, end):
        # 输入一个是前驱,一个后驱
        prev, curr = start, start.next
        first = curr
        while curr != end:
            temp = curr.next
            curr.next = prev
            prev = curr
            curr = temp
        start.next = prev
        first.next = end
        return first
```

```
#
# @lc app=leetcode.cn id=26 lang=python3
#
# [26] 删除排序数组中的重复项
#
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        idx = 0
        while idx < len(nums) -1 :
            if nums[idx] == nums[idx+1]:
                nums.pop(idx)
                idx -= 1
            idx += 1
        return len(nums)
```

```
#
# @lc app=leetcode.cn id=27 lang=python3
#
# [27] 移除元素
#
```

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        left = 0
        right = len(nums) - 1
        while left <= right :
            if nums[left] == val:
                nums[left] , nums[right] = nums[right] ,nums[left]
                right -= 1
            else:
                left += 1
        return left
```

```
#
# @lc app=leetcode.cn id=28 lang=python3
#
# [28] 实现 strStr()
#
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not needle or haystack == needle:
            return 0
        elif len(haystack)<= len(needle):
            return -1

        leng = len(needle)
        for i in range(len(haystack)-leng +1):
            if needle == haystack[i:i+leng]:
                return i
        return -1
```

```
#
# @lc app=leetcode.cn id=29 lang=python3
#
# [29] 两数相除
#
class Solution:
    def divide(self, dividend: int, divisor: int) -> int:
        if (dividend < 0 and divisor < 0) or (dividend > 0 and divisor > 0):
            positive = 1
        else:
            positive = -1

        dividend, divisor = abs(dividend), abs(divisor)
        res = 0
        while dividend >= divisor:
            temp, i = divisor, 1
            while dividend >= temp:
                dividend -= temp
                res += i
                # 除数乘以2 商一下子也多2
                i <<= 1
                temp <<= 1

        # 防止溢出
        return min(max(positive * res,-2**31) , 2**31-1)
```

```
#
# @lc app=leetcode.cn id=31 lang=python3
#
# [31] 下一个排列
#
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        # i为数组倒数第二个值，j为倒数第一个值
        i = len(nums) - 2
        j = len(nums) - 1
        # 从右到左找到第一次断崖
        # 第一次非逆序的地方
        while i >= 0 and nums[i] >= nums[i+1]:
            i -= 1

        # 从右到左找到比崖底水平面高的第一个元素
        if i >= 0:
            while j >= 0 and nums[i] >= nums[j]:
                j -= 1
            nums[i], nums[j] = nums[j], nums[i]

        self.reverse(nums, i+1)

    # 用于原地反转nums中从start之后的所有元素
    def reverse(self, nums, start):
        i, j = start, len(nums) - 1
        while i < j:
            nums[i], nums[j] = nums[j], nums[i]
            i += 1
            j -= 1
        return
```

```
#
# @lc app=leetcode.cn id=32 lang=python3
#
# [32] 最长有效括号
```

```
#
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        '''
        # 栈法
        res = []
        stack = []
        for i in range(len(s)):
            if(stack and s[i]==")"):
                res.append(stack.pop())
                res.append(i)
            if(s[i]=="("):
                stack.append(i)

        res.sort()
        max_len = 0
        i=0
        while i < len(res)-1:
            tmp = i
            # 最长连续值
            while( i < len(res)-1 and res[i+1]-res[i] == 1):
                i += 1
            max_len = max(max_len,i-tmp+1)
            i += 1
        return max_len
        '''

        # 动态规划
        if not s:
            return 0
        dp = [0] * len(s)
        for i in range(1,len(s)):
            if s[i]==")":
                # ()对
                if s[i-1]=="(":
                    dp[i] = dp[i-2] + 2
                # 连着两个))
                if s[i-1]==")" and i-1-dp[i-1]>=0 and s[i-1-dp[i-1]]=="(":
                    dp[i] = dp[i-dp[i-1]-2] + dp[i-1] + 2
        return max(dp)
```

```
#
# @lc app=leetcode.cn id=33 lang=python3
#
# [33] 搜索旋转排序数组
#
```

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        if not nums:
            return -1
        l,r = 0 , len(nums) -1

        while l <= r:
            mid = (l+r)//2
            if nums[mid] == target:
                return mid
            # mid在前半段 或者l mid r 都在右边
            if nums[l] <= nums[mid]:
                if nums[l] <= target <= nums[mid]:
                    r = mid -1
                else:
                    l = mid +1
            # l 在左半段 、mid 在后半段
            else:
                if nums[mid] <= target <= nums[r]:
                    l = mid +1
                else:
                    r = mid -1
        return -1
```

```
#
# @lc app=leetcode.cn id=34 lang=python3
#
# [34] 在排序数组中查找元素的第一个和最后一个位置
#
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        if len(nums) == 0:
            return [-1, -1]
        min = 0
        max = len(nums) - 1
        while min <= max:
            pos = (min + max) // 2
            if nums[pos] > target:
                max = pos - 1
            elif nums[pos] < target:
                min = pos + 1
            else:
                # when nums[pos] == target
                # find the min and max
                for i in range(pos, max + 1) :
                    if nums[i] == target:
```

```
                    max = i
            for i in range(pos, min -1 , -1) :
                if nums[i] == target:
                    min = i
            return [min, max]
        return [-1, -1]
```

```
#
# @lc app=leetcode.cn id=35 lang=python3
#
# [35] 搜索插入位置
#
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = (left + right)//2
            if nums[mid] == target:
                return mid
            elif target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        return left
```

```
#
# @lc app=leetcode.cn id=36 lang=python3
#
# [36] 有效的数独
#
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        return (self.is_row_valid(board) and
                self.is_col_valid(board) and
                self.is_square_valid(board))

    def is_row_valid(self, board):
        for row in board:
            if not self.is_unit_valid(row):
                return False
        return True

    def is_col_valid(self, board):
        # 列转化成行
        for col in zip(*board):
```

```python
            if not self.is_unit_valid(col):
                return False
        return True

    def is_square_valid(self, board):
        for i in (0, 3, 6):
            for j in (0, 3, 6):
                square = [board[x][y] for x in range(i, i + 3) for y in range(j, j + 3)]
                if not self.is_unit_valid(square):
                    return False
        return True

    def is_unit_valid(self, unit):
        unit = [i for i in unit if i != '.']
        return len(set(unit)) == len(unit)
```

```python
#
# @lc app=leetcode.cn id=37 lang=python3
#
# [37] 解数独
#
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
        self.dfs(board)

    def dfs(self,board):
        for i in range(9):
            for j in range(9):
                if board[i][j] == '.':
                    for k in '123456789':
                        board[i][j] = k
                        # 修改一个值判断是不是合法的
                        # 如果这个递归可以返回true并且当前填入的数字也没毛病
                        # 则证明我们解完了数独
                        if self.isOK(board,i,j) and self.dfs(board):
                            return True
                        board[i][j] = '.'
                    return False
        # 全部填完之后返回True
        return True

    def isOK(self,board,x,y):
        #列符合
        for i in range(9):
            if i != x and board[i][y] == board[x][y]:
                return False
        #检查行是否符合
        for j in range(9):
            if j != y and board[x][j] == board[x][y]:
                return False
        row_start = 3*(x // 3)
        col_start = 3*(y // 3)
        for i in range(row_start,row_start+3):
            for j in range(col_start,col_start+3):
                if (i!= x or j!= y) and board[i][j] == board[x][y]:
                    return False
        return True
```

```
#
# @lc app=leetcode.cn id=38 lang=python3
#
# [38] 外观数列
#
class Solution:
    def countAndSay(self, n: int) -> str:
        s = '1'
        for _ in range(n-1):
            s = self.count(s)
        return s

    def count(self ,s):
        m = list(s)
        # 加一个后面不会溢出(随便加一个就行)
        m.append(5)
        res = ()
        i,j = 0,0
        while i < len(m)-1:
            j+= 1
            if m[j] != m[i]:
                res += (str(j-i) , m[i] )
                i = j
        # 用空元素链接res
        return ''.join(res)
```

```
#
# @lc app=leetcode.cn id=39 lang=python3
#
# [39] 组合总和
#
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        candidates.sort()
        res = []
        self.dfs(candidates, target, 0, [], res)
        return res

    def dfs(self, nums, target, index, path, res):
        if target < 0:
            return
        if target == 0:
            res.append(path)
            return
        for i in range(index, len(nums)):
            self.dfs(nums, target-nums[i], i, path+[nums[i]], res)
```

```
#
# @lc app=leetcode.cn id=40 lang=python3
#
# [40] 组合总和 II
#
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        candidates.sort()
        res = []
        self.combine_sum_2(candidates,target, 0, [], res)
        return res

    def combine_sum_2(self, nums,target, start, path, res):
        # 超过了
        if target < 0:
            return
        if target == 0 :
            res.append(path)
            return

        for i in range(start, len(nums)):
            # 解集不重复
            if i > start and nums[i] == nums[i - 1]:
                continue
            self.combine_sum_2(nums,target - nums[i],
                    i + 1, path + [nums[i],], res)
```

```
#
# @lc app=leetcode.cn id=41 lang=python3
#
# [41] 缺失的第一个正数
#
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        self.bucket_sort(nums)

        for i in range(len(nums)):
            if nums[i] != (i+1):
                return i+1
        return len(nums)+1

    def bucket_sort(self,nums):
        for i in range(len(nums)):
            while 0 <= nums[i] < len(nums) and nums[i] != nums[nums[i]-1]:
                temp = nums[i]-1
                nums[i] = nums[temp]
                nums[temp] = temp +1
        return
```

```
#
# @lc app=leetcode.cn id=42 lang=python3
#
# [42] 接雨水
#
class Solution:
    def trap(self, height: List[int]) -> int:
        if not height:  # 边界检查
            return 0
        l ,r = 0 , len(height) - 1

        res = 0
        l_max , r_max = 0 , 0
        while l < r :
            if height[l] < height[r]:
                if height[l] >= l_max:
                    l_max = height[l]
                else:
                    res += l_max - height[l]
                l += 1
            else:
                if height[r] >= r_max:
                    r_max = height[r]
                else:
                    res += r_max - height[r]

                r -= 1
        return res
```

```
#
```

```
# @lc app=leetcode.cn id=43 lang=python3
#
# [43] 字符串相乘
#
class Solution:
    def multiply(self, num1: str, num2: str) -> str:

        #把num1,num2翻转方便计算
        num1 = num1[::-1]; num2 = num2[::-1]
        #每一位互相乘的结果用一维数组去储存
        arr = [0 for i in range(len(num1)+len(num2))]
        #填充这个一维数组
        for i in range(len(num1)):
            for j in range(len(num2)):
                arr[i+j] += int(num1[i]) * int(num2[j])

        res = []
        # arr是反的
        #计算每一位的终极结果
        for i in range(len(arr)):
            #digit表示这一位的数字
            digit = arr[i] % 10
            #carry表示加给下一位的量
            carry = arr[i] // 10
            if i < len(arr)-1:
                #下一位加上
                arr[i+1] += carry
            #更新答案
            res.insert(0, str(digit))
        #去除首位为0的情况
        while res[0] == ’0’ and len(res) > 1:
            res.pop(0)
        #连接成字符串
        return ’’.join(res)
```

```
#
# @lc app=leetcode.cn id=45 lang=python3
#
# [45] 跳跃游戏 II
#
class Solution:
    def jump(self, nums: List[int]) -> int:
        if len(nums) <= 1:
            return 0
        # (start -> end )
        end = nums[0]
        start = 0
        step = 1
        maxDis = nums[0]
        while end < len(nums) - 1:
            # 看一步最远能走到哪
            for i in range(start + 1, end + 1):
                maxDis = max(maxDis, nums[i] + i)
            start = end
            end = maxDis
            step += 1
        return step
```

```
#
# @lc app=leetcode.cn id=46 lang=python3
#
# [46] 全排列
#
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        #nums.sort()
        res = []
        self.dfs(nums, [], res)
        return res

    def dfs(self, nums, path, res):
        if not nums :
            # nums已经全部压入到path里面了
            res.append(path)
            return
        for i in range(len(nums)):
            self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```
#
# @lc app=leetcode.cn id=47 lang=python3
#
# [47] 全排列 II
#
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        res = []
        self.dfs(nums, [], res)
        return res

    def dfs(self, nums, path, res):
        if not nums and path not in res:
            # nums已经全部压入到path里面了
```

```python
            res.append(path)
            return
        for i in range(len(nums)):
            self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```python
#
# @lc app=leetcode.cn id=48 lang=python3
#
# [48] 旋转图像
#
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        if matrix is None or len(matrix) == 1:
            return
        ls = len(matrix)

        for i in range(ls // 2):
            # 那一圈的半行
            begin, end = i, ls - 1 - i # 左右都往内部i个单位
            for k in range(ls- 1 - 2 * i ): # 减两个i的单位
                # 顺着转
                temp = matrix[end - k][begin] # 左下角
                matrix[end - k][begin] = matrix[end][end - k] # 右下角给左下角
                matrix[end][end - k] = matrix[begin + k][end] # 右上角给右下角
                matrix[begin + k][end] = matrix[begin][begin + k] # 左上角给右上角
                matrix[begin][begin + k] = temp # 左下角给左上角
        return
```

```python
#
# @lc app=leetcode.cn id=49 lang=python3
#
# [49] 字母异位词分组
#
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        dic = {}
        # key是单词对应的元素
        # value是字符串
        for word in strs:
            key = ''.join(sorted(word))
            if key not in dic :
                dic[key] = []
            dic[key].append(word)
        res = []
        for i in dic:
            res.append(dic[i])
        return res
```

```
#
# @lc app=leetcode.cn id=50 lang=python3
#
# [50] Pow(x, n)
#
class Solution:
    def myPow(self, x: float, n: int) -> float:
        if n == 0:
            return 1
        if n < 0:
            return 1 / self.myPow(x, -n)
        if n % 2:
            return x * self.myPow(x, n-1)
        return self.myPow(x*x, n // 2)
```

```
#
# @lc app=leetcode.cn id=51 lang=python3
#
# [51] N皇后
#
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        result = []
        # C[i]表示第i行皇后在哪一列
        C = [-1 for _ in range(n)]
        self.dfs(C,result,0)
        return result

    def dfs(self,C,res,row):
        N = len(C)
        # 终止条件
        if N == row:
            path = [[”.” for _ in range(N)] for _ in range(N)]
            for i in range(N):
                # (i,C[i])位置对应皇后
                path[i][C[i]] = ”Q”
            path = [””.join(r) for r in path]
            # if path not in res:
            # 不用排除
            res.append(path)
            return
        # 对该行每一列都进行尝试,可以的话下一行
        for j in range(N):
            if j not in C and self.isOK(C,row,j):
                C[row] = j
                self.dfs(C,res,row+1)
                C[row] = -1

    # 对该行之前的都进行判断,返回合理与否
    def isOK(self,C,row,col):
        for i in range(row):
            # 同一列
            # 同一对角线
            if C[i] == col or abs(i-row) == abs(C[i]-col):
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=52 lang=python3
#
# [52] N皇后 II
#
class Solution:
    def totalNQueens(self, n: int) -> int:
        self.res = 0
        # C[i]表示第i行皇后在哪一列
        C = [-1 for _ in range(n)]
        self.dfs(C,0)
        return self.res

    def dfs(self,C,row):
        N = len(C)
        # 终止条件
        if N == row:
            # 不用排除
            self.res += 1
        # 对该行每一列都进行尝试,可以的话下一行
        for j in range(N):
            if j not in C and self.isOK(C,row,j):
                C[row] = j
                self.dfs(C,row+1)
                C[row] = -1

    # 对该行之前的都进行判断,返回合理与否
    def isOK(self,C,row,col):
        for i in range(row):
            # 同一列
            # 同一对角线
            if C[i] == col or abs(i-row) == abs(C[i]-col):
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=53 lang=python3
#
# [53] 最大子序和
#
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        temp = maxsum = nums[0]
        for num in nums[1:]:
            # num 要么单独一个子列,要么归入别的子列
            temp = max(num,temp+num)
            maxsum = max(temp,maxsum)
        return maxsum
```

```
#
# @lc app=leetcode.cn id=54 lang=python3
#
# [54] 螺旋矩阵
#
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix:
            return []

        '''
        # 常规方法太烦了
        res = []
        xbegin = ybegin = 0
        xend = len(matrix[0]) - 1
        yend = len(matrix) - 1
        while True :
            # 横
            for j in range(xbegin,xend+1):
                res.append(matrix[ybegin][j])
            ybegin += 1
            if ybegin > yend :
                break
            # 竖
            for j in range(ybegin,yend+1):
                res.append(matrix[j][xend])
            xend -= 1
            if xbegin > xend:
                break
            # 横
            for j in range(xend,xbegin-1,-1):
                res.append(matrix[yend][j])
            yend -= 1
            if ybegin > yend :
                break
            # 竖
            for j in range(yend,ybegin-1,-1):
                res.append(matrix[j][xbegin])
            xbegin += 1
            if xbegin > xend:
                break
        return res
        '''

        m,n = len(matrix),len(matrix[0])
        x = y = di = 0
        dx = [0,1,0,-1]
        dy = [1,0,-1,0]
        res = []
        visited = set()

        for i in range(m*n):
            res.append(matrix[x][y])
            visited.add((x,y))
            nx,ny = x+dx[di],y+dy[di]
            if 0<=nx<m and 0<=ny<n and (nx,ny) not in visited:
                x,y = nx,ny
            else:
                di = (di+1)%4 # 如果不满足条件，换一个方向进行遍历
                x,y = x+dx[di],y+dy[di]
        return res
```

```
#
# @lc app=leetcode.cn id=55 lang=python3
#
# [55] 跳跃游戏
#
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        start = end = 0
        while start <= end < len(nums) - 1:
            end = max(end, nums[start] + start)
            start += 1
        return end >= len(nums) - 1
```

```
#
```

```
# @lc app=leetcode.cn id=56 lang=python3
#
# [56] 合并区间
#
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        if len(intervals) <= 1:
            return intervals
        res = []
        intervals.sort(key = lambda x: x[0])
        s , e = intervals [0][0] , intervals [0][1]

        for i in range(1,len(intervals)):
            # 后边跟着的区间和[s,e]的交叉,相当于合并
            if e >= intervals[i][0] :
                e = max( e , intervals[i][1] )
            # 紧跟着的区间在[s,e]后面
            else:
                res.append([s,e])
                s ,e = intervals[i][0] , intervals [i][1]
        res.append([s,e])
        return res
```

```
#
# @lc app=leetcode.cn id=57 lang=python3
#
# [57] 插入区间
#
class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]) -> List[List[int]]:
        s, e = newInterval[0], newInterval[1]
        left, right = [], []
        for inter in intervals:
            # 左边部分
            if s > inter[1]:
                left.append(inter)
            # 右边部分
            elif e < inter[0]:
                right.append(inter)
            # 和区间交叉部分,合并
            else:
                s = min(s, inter[0])
                e = max(e, inter[1])
        return left + [[s, e]] + right
```

```
#
```

```
# @lc app=leetcode.cn id=58 lang=python3
#
# [58] 最后一个单词的长度
#
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        if not s :
            return 0
        tmp = s.split(' ')
        tmp = [t for t in tmp if len(t) > 0]
        if len(tmp) == 0:
            return 0
        else:
            return len(tmp[-1])
```

```
#
# @lc app=leetcode.cn id=59 lang=python3
#
# [59] 螺旋矩阵 II
#
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        mat = [[0 for _ in range(n)] for _ in range(n)]

        b,e = 0 , n - 1
        val = 1
        while b < e :
            # 横
            for i in range(b,e):
                mat[b][i] = val
                val += 1
            # 竖
            for i in range(b,e):
                mat[i][e] = val
                val += 1
            # 横
            for i in range(e,b,-1):
                mat[e][i] = val
                val += 1
            # 竖
            for i in range(e,b,-1):
                mat[i][b] = val
                val += 1
            b += 1
            e -= 1

        # n为奇数,中间还有一个值
        if n % 2 :
            mat[b][e] = val
        return mat
```

```
#
# @lc app=leetcode.cn id=60 lang=python3
#
# [60] 第k个排列
#
class Solution:
    def getPermutation(self, n: int, k: int) -> str:
        # 待选择的字符串
        nums = [str(i) for i in range(1,n+1)]
        # 0!, 1!, ..., (n - 1)!
        factorials = [1]
        for i in range(1, n):
            factorials.append(factorials[i - 1] * i)

        # 第几个转化为第几个的索引(减1)
        k -= 1

        res = []
        for i in range(n - 1, -1, -1):
            # 计算第几个区间,首位所在的区间 k//(n-1)!
            # 第一个区间首位是1,第二个区间首位是2
            idx = k // factorials[i]
            # 减去多个区间对应的值
            k -= idx * factorials[i]
            # 结果值添加对应的数字
            res.append(nums[idx])
            # 因为排列不重复,nums需要去掉对应元素
            nums.pop(idx)

        return ”.join(res)
```

```
#
# @lc app=leetcode.cn id=61 lang=python3
#
# [61] 旋转链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
```

```

class Solution:
    def rotateRight(self, head: ListNode, k: int) -> ListNode:
        if head is None or k == 0:
            return head

        pointer = head
        length = 1
        while pointer.next:
            pointer = pointer.next
            length += 1

        # 左部分多少个
        k = length - k%length

        # 连成一个环
        pointer.next = head

        for i in range(k):
            pointer = pointer.next

        # 断开
        head = pointer.next
        pointer.next = None
        return head
```

```
#
# @lc app=leetcode.cn id=62 lang=python3
#
# [62] 不同路径
#
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        mat = [[0 for _ in range(n)] for _ in range(m)]
        for r in range(m):
            mat[r][0] = 1
        for c in range(n):
            mat[0][c] = 1
        for r in range(1,m):
            for c in range(1,n):
                mat[r][c] = mat[r-1][c] + mat[r][c-1]
        return mat[-1][-1]
```

```
#
# @lc app=leetcode.cn id=63 lang=python3
#
```

```
# [63] 不同路径 II
#
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
        if not obstacleGrid:
            return
        r, c = len(obstacleGrid), len(obstacleGrid[0])
        mat = [[0 for _ in range(c)] for _ in range(r)]
        # 到起点看这里有没有问题
        mat[0][0] = 1 - obstacleGrid[0][0]

        for i in range(1, r):
            mat[i][0] = mat[i-1][0] * (1 - obstacleGrid[i][0])
        for i in range(1, c):
            mat[0][i] = mat[0][i-1] * (1 - obstacleGrid[0][i])

        for i in range(1, r):
            for j in range(1, c):
                mat[i][j] = (mat[i][j-1] + mat[i-1][j]) * (1 - obstacleGrid[i][j])
        return mat[-1][-1]
```

```
#
# @lc app=leetcode.cn id=64 lang=python3
#
# [64] 最小路径和
#
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        m,n = len(grid),len(grid[0])
        dp = [[0 for _ in range(n)] for _ in range(m)]
        dp[0][0] = grid[0][0]
        for r in range(1,m):
            dp[r][0] = dp[r-1][0] + grid[r][0]
        for c in range(1,n):
            dp[0][c] = dp[0][c-1] + grid[0][c]
        for r in range(1,m):
            for c in range(1,n):
                dp[r][c] = min(dp[r-1][c] , dp[r][c-1])+grid[r][c]
        return dp[m-1][n-1]
```

```
#
# @lc app=leetcode.cn id=66 lang=python3
#
# [66] 加一
#
class Solution:
```

```python
    def plusOne(self, digits: List[int]) -> List[int]:
        '''
        # 数值操作
        num = 0
        for i in range(len(digits)):
            num = num * 10 + digits[i]
        num = num + 1
        res = []
        while num > 0:
            res.append(num%10)
            num //= 10
        res.reverse()
        return res
        '''

        # 列表操作
        digits[-1] += 1
        digits.insert(0, 0)
        for i in range(len(digits)-1,0,-1):
            carry = digits[i] // 10
            digits[i] %= 10
            digits[i-1] += carry

        if digits[0] == 0:
            digits.pop(0)

        return digits
```

```python
#
# @lc app=leetcode.cn id=67 lang=python3
#
# [67] 二进制求和
#
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        if not a:
            return b
        if not b:
            return a
        # 最后都是1 前面的相加 再加1 补0
        if a[-1] == '1' and b[-1] == '1':
            return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'
        # 最后都是0 补0
        if a[-1] == '0' and b[-1] == '0':
            return self.addBinary(a[0:-1],b[0:-1])+'0'
        # 最后一个1 一个0 补1
        else:
            return self.addBinary(a[0:-1],b[0:-1])+'1'
```

```
#
# @lc app=leetcode.cn id=69 lang=python3
#
# [69] x 的平方根
#
class Solution:
    def mySqrt(self, x: int) -> int:
        l ,r = 0 , x
        while l <= r:
            mid = (l+r)//2
            if mid**2 <= x < (mid+1)**2:
                return mid
            elif x < mid**2:
                r = mid
            else:
                l = mid+1
```

```
#
# @lc app=leetcode.cn id=70 lang=python3
#
# [70] 爬楼梯
#
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        # 初始的两个  输入1 or 2
        a, b = 1, 2
        # 从n大于3开始
        for i in range(2, n):
            b ,a = a+b , b
        return b
```

```
#
# @lc app=leetcode.cn id=71 lang=python3
#
# [71] 简化路径
#
class Solution:
    def simplifyPath(self, path: str) -> str:
        res = []
        for child in path.split('/'):
            if child in ('' , '.'):
                pass
            elif child == '..':
                if res:res.pop()
            else:
                res.append(child)
        return '/' + '/'.join(res)
```

```
#
# @lc app=leetcode.cn id=72 lang=python3
#
# [72] 编辑距离
#
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        l1, l2 = len(word1)+1, len(word2)+1
        dp = [[0 for _ in range(l2)] for _ in range(l1)]
        # 行列处理 对应从空到一个字符串 或 一个字符串到空
        for i in range(l1):
            dp[i][0] = i
        for j in range(l2):
            dp[0][j] = j
        for i in range(1, l1):
            for j in range(1, l2):
                if word1[i-1]==word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    # 三个分别对应于加、减、替换
                    dp[i][j] = min(dp[i-1][j],
                                   dp[i][j-1],
                                   dp[i-1][j-1]
                                   )+1
        return dp[-1][-1]
```

```
#
# @lc app=leetcode.cn id=73 lang=python3
#
# [73] 矩阵置零
#
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        '''
        # 直接法
        row = []
        col = []
        m = len(matrix)
        n = len(matrix[0])
        for i in range(m):
            for j in range(n):
                if matrix[i][j] == 0:
                    row.append(i)
                    col.append(j)
        row = set(row)
        col = set(col)

        for i in row:
            for j in range(n):
                matrix[i][j] = 0
        for j in col :
            for i in range(m):
                matrix[i][j] =0

        return matrix
        '''
        # 第一行出现一个0
        firstRowHasZero = not all(matrix[0])
        m = len(matrix)
        n = len(matrix[0])
        # 第一行第一列做标记
        for i in range(1,m):
            for j in range(n):
                if matrix[i][j] == 0:
                    matrix[0][j] = matrix[i][0] = 0
        # 置0
        for i in range(1,m):
            for j in range(n-1,-1,-1):
                if matrix[i][0] == 0 or matrix[0][j] == 0:
                    matrix[i][j] = 0
        # 补一下第一行的

        if firstRowHasZero:
            matrix[0] = [0] * n

        return matrix
```

```
#
# @lc app=leetcode.cn id=74 lang=python3
#
# [74] 搜索二维矩阵
#
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        if len(matrix)==0 or len(matrix[0])==0 or target < matrix[0][0] or target > matrix[-1][-1]:
            return False
        row = 0
        col = len(matrix[0]) -1
        while row < len(matrix) and col >= 0 :
            if matrix[row][col] > target:
                col -= 1
            elif matrix[row][col] < target:
                row += 1
            else :
                return True
        return False
```

```
#
# @lc app=leetcode.cn id=75 lang=python3
#
# [75] 颜色分类
#
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        count = [0,0,0]
        for num in nums:
            count[num] += 1
        idx = 0
        for i in range(3):
            for j in range(count[i]):
                nums[idx] = i
                idx += 1
```

```
#
# @lc app=leetcode.cn id=76 lang=python3
#
# [76] 最小覆盖子串
#
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if s is None or len(s) < len(t):
            return ””
        res = ””
        left = 0
        right = 0
        min_len = len(s)
        count = 0

        m = {}
        # 统计t中字符数目
        for i in t:
```

```python
            m[i] = m.get(i,0) + 1

        while right < len(s):
            if s[right] in m:
                # 先找到一个区间能包含t,但长度不一定是最短的
                m[s[right]] -= 1
                if m[s[right]] >= 0:
                    count += 1
                # 找到了一个区间
                while (count == len(t)):
                    # 选择更短的子串
                    if (right - left + 1 < min_len):
                        min_len = right-left+1
                        res = s[left:right+1]

                    if s[left] in m:
                        m[s[left]] += 1
                        if m[s[left]] > 0:
                            count -= 1
                    left += 1
            right += 1

        return res
```

```python
#
# @lc app=leetcode.cn id=77 lang=python3
#
# [77] 组合
#
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []
        self.dfs(n,k,1,[], res)
        return res

    def dfs(self,n,k,start,path,res):
        if 0 == k and path not in res:
            res.append(path)
        for i in range(start,n+1):
            self.dfs(n,k-1,i+1,path+[i],res)
```

```python
#
# @lc app=leetcode.cn id=78 lang=python3
#
# [78] 子集
#
```

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        self.dfs(nums, 0, [], res)
        return res

    def dfs(self, nums, index, path, res):
        res.append(path)
        for i in range(index, len(nums)):
            self.dfs(nums, i+1, path+[nums[i]], res)
```

```
#
# @lc app=leetcode.cn id=79 lang=python3
#
# [79] 单词搜索
#
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m , n = len(board) , len(board[0])
        visited = [[False for i in range(n)] for i in range(m)]
        # 遍历寻找开头
        for i in range(m):
            for j in range(n):
                if self.dfs(board,word,visited,i,j,0):
                    return True
        return False

    def dfs(self,board,word,visited,i,j,start):
        # 终止条件
        if start == len(word):
            return True
        # 溢出 剪枝 or 已经访问过了
        if i < 0 or j < 0 or i >= len(board) or j >= len(board[0]) or visited[i][j] or board[i][j] != word[start]:
            return False

        if board[i][j] == word[start]:
            visited[i][j] = True
            ret = self.dfs(board,word,visited,i+1,j,start+1) or \
                  self.dfs(board,word,visited,i-1,j,start+1) or \
                  self.dfs(board,word,visited,i,j+1,start+1) or \
                  self.dfs(board,word,visited,i,j-1,start+1)
            visited[i][j] = False

            return ret
```

```
#
# @lc app=leetcode.cn id=80 lang=python3
#
# [80] 删除排序数组中的重复项 II
#
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        if not nums:
            return 0
        # 初始化第一个
        i, count = 1, 1

        while i < len(nums) :
            if nums[i] == nums[i - 1]:
                count += 1
                if count > 2:
                    nums.pop(i)
                    # 这里的减一和后面对消
                    i-= 1
            else:
                count = 1
            i += 1
        return len(nums)
```

```
#
# @lc app=leetcode.cn id=81 lang=python3
#
# [81] 搜索旋转排序数组 II
#
class Solution:
    def search(self, nums: List[int], target: int) -> bool:
        if not nums:
            return False
        l,r = 0 , len(nums) -1

        while l <= r:
            mid = (l+r)//2
            if nums[mid] == target:
                return True
            # mid在前半段 或者l mid r 都在右边
            if nums[l] < nums[mid]:
                if nums[l] <= target < nums[mid]:
                    r = mid -1
                else:
                    l = mid +1
            # l 在左半段 、mid 在后半段
            elif nums[mid] < nums[l]:
                if nums[mid] < target <= nums[r]:
                    l = mid +1
                else:
                    r = mid -1
            else:
                l += 1
        return False
```

```
#
# @lc app=leetcode.cn id=82 lang=python3
#
# [82] 删除排序链表中的重复元素 II
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        prev = dummy

        while head and head.next:
            if head.val == head.next.val:
                while head and head.next and head.val == head.next.val:
                    head = head.next
                head = head.next
                prev.next = head
            # 两个指针都往后走
            else:
                prev = prev.next
                head = head.next
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=83 lang=python3
#
# [83] 删除排序链表中的重复元素
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
```

```
#           self.val = x
#           self.next = None

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        point = head
        while point:
            while point.next and point.val == point.next.val:
                point.next = point.next.next
            point = point.next
        return head
```

```
#
# @lc app=leetcode.cn id=84 lang=python3
#
# [84] 柱状图中最大的矩形
#
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        # 此处较为巧妙。若heights数组中元素都是单增序列，则最后无法出栈stack，也就无法计算最大面积，所以补个0，使之最后可以出栈
        heights.append(0)
        stack = [-1]
        res = 0

        for idx, val in enumerate(heights):
            # 不是递增栈
            while heights[stack[-1]] > val:
                h = heights[stack.pop()]
                w = idx - stack[-1] -1
                res = max(res, h*w)
            stack.append(idx)
        return res
```

```
#
# @lc app=leetcode.cn id=85 lang=python3
#
# [85] 最大矩形
#
class Solution:
    def maximalRectangle(self, matrix: List[List[str]]) -> int:
        '''
        if not matrix or not matrix[0]:
            return 0
        m , n = len(matrix),len(matrix[0])
        # height 的尾部多了一个0,防止递增错误
        height = [0] * (n+1)
        max_area = 0
        for i in range(m):
            # 计算h
            for j in range(n):
                # 遍历到的每行的h
                height[j] = height[j]+1 if matrix[i][j]=='1' else 0
            # 找出所有h和w的组合
            # 同84题
            stack = [-1]
            for k in range(n + 1):
                while height[k] < height[stack[-1]]:
                    h = height[stack.pop()]
                    w = k - stack[-1] - 1
                    max_area = max(max_area, h * w)
                stack.append(k)
        return max_area
        '''
        if not matrix or not matrix[0]:
            return 0
        m , n = len(matrix),len(matrix[0])
        # 申请辅助数组并初始化
        # 向上、向左、向右能延伸到的最远的地方
        left, right, height = [0]*n, [n]*n, [0]*n
        max_A = 0
        # 从第一行开始遍历
        for i in range(m):
            # 用来记录下标
            cur_left, cur_right = 0, n
            # 从第一个元素开始遍历
            for j in range(n):
                # 如果矩阵中当前坐标为1时， 我们将height对应的下标加一
                # left取cur_left和 left[i]中取最大的
                if matrix[i][j] == "1":
                    height[j] = height[j] + 1
                    left[j] = max(left[j], cur_left)
                else: # 否则赋值位0
                    height[j], left[j] = 0, 0
                    cur_left = j+1
            # right数组从末尾开始遍历
            for j in range(n-1, -1, -1):
                if matrix[i][j] == "1":
                    right[j] = min(right[j], cur_right)
                else:
                    right[j] = n
                    cur_right = j
            for j in range(n):
                # 计算到前行为止最大的面积
                max_A = max(max_A,(right[j]-left[j])*height[j])
        return max_A
```

```
#
# @lc app=leetcode.cn id=86 lang=python3
#
# [86] 分隔链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def partition(self, head: ListNode, x: int) -> ListNode:
        h1 = l1 = ListNode(0)
        h2 = l2 = ListNode(0)

        while head:
            if head.val < x:
                l1.next = head
                l1 = l1.next
            else:
                l2.next = head
                l2 = l2.next
            head = head.next
        # l1 l2都在各自的尾部了
        l2.next = None
        l1.next = h2.next

        return h1.next
```

```
#
# @lc app=leetcode.cn id=88 lang=python3
#
# [88] 合并两个有序数组
#
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        # 从后往前
        p1 = m - 1
        p2 = n - 1
        p = m + n -1
        # 两个都没放完
        while p1 >= 0 and p2 >= 0:
            if nums1[p1] >= nums2[p2]:
                nums1[p] = nums1[p1]
                p1 -= 1
            else:
                nums1[p] = nums2[p2]
                p2 -= 1
            p -= 1
        # p1没放完，那就不用再操作了
        # p2没放完
        while p2 >= 0:
            nums1[p] = nums2[p2]
            p -= 1
            p2 -= 1
```

```
#
# @lc app=leetcode.cn id=89 lang=python3
#
# [89] 格雷编码
#
class Solution:
    def grayCode(self, n: int) -> List[int]:
        res = [0]
        for i in range(n):
            for j in range(len(res)-1,-1,-1):
                res.append(res[j] + (1 << i))
        return res
```

```
#
# @lc app=leetcode.cn id=90 lang=python3
#
# [90] 子集 II
#
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        # self.dfs(nums, 0, [], res)
        self.dfs2(nums, 0, [], res)
        return res

    def dfs(self, nums, index, path, res):
        if path not in res:
            res.append(path)
        for i in range(index, len(nums)):
            self.dfs(nums, i+1, path+[nums[i]], res)

    def dfs2(self, nums, index, path, res):
        res.append(path)
        for i in range(index, len(nums)):
            if i > index and nums[i] == nums[i-1]:
                continue
            self.dfs2(nums, i+1, path+[nums[i]], res)
```

```
#
# @lc app=leetcode.cn id=91 lang=python3
#
# [91] 解码方法
#
class Solution:
    def numDecodings(self, s: str) -> int:
        if s is None or s[0] == '0':
            return 0
        # dp[i] 表示s中前i个字符组成的子串的解码方法的个数，长度比输入数组长多多1，并将 dp[0] 初始化为1
        dp = [0] * (len(s)+1)
        dp[0] = dp[1] = 1
        for i in range(2,len(s)+1):
            if s[i - 1] >= '1' and s[i - 1] <= '9':
                dp[i] += dp[i - 1]
            if s[i-2]=='1' or (s[i-2] == '2' and s[i-1] <= '6'):
                dp[i] += dp[i - 2]
        return dp[-1]
```

```
#
# @lc app=leetcode.cn id=92 lang=python3
#
# [92] 反转链表 II
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        prev = dummy
        # 走m-1个
```

```python
        for i in range(m-1):
            prev = prev.next
        # 反转
        temp = None
        cur = prev.next
        for i in range(n-m+1):
            next = cur.next
            # reverse
            cur.next = temp
            temp = cur
            # 下一个
            cur = next
        # cur指向的是最后部分,中间已经没有了
        # None 的下一个
        # 最后面一段
        prev.next.next = cur
        '''
        wi = temp
        while wi.next :
            wi = wi.next
        wi.next = cur
        '''
        # 中间一段
        prev.next = temp

        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=93 lang=python3
#
# [93] 复原IP地址
#
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        res = []
        self.dfs(s,[], res,0)
        return res

    def dfs(self,s,ip,res,start):
        # 终止条件
        if len(ip) == 4 and start == len(s):
            address = '.'.join(ip)
            res.append(address)
            return

        # 特殊场景下可以剪枝
        # 剩下的子串太长(剩下的ip位都超过了3位)或太短(剩下的ip位都小于1位了)
        if len(s) -start > 3*(4-len(ip)) or len(s) -start < (4-len(ip)):
            return

        # 最多三位(+0,+1,+2)
        for i in range(0,3):
            substr = s[start:start+i+1]
            # 允许单个0,但是不允许0开头的一串,比如025
            if i != 0 and substr[0] == '0':
                continue
            if int(substr) >= 0 and int(substr) <= 255:
                self.dfs(s,ip+[substr],res,start + i + 1)
```

```
#
# @lc app=leetcode.cn id=94 lang=python3
#
# [94] 二叉树的中序遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        '''
        result = []
        stack = []
        p = root
        while stack or p:
            # 先把左边的压进去
            if p:
                stack.append(p)
                p = p.left
            else:
                p = stack.pop()
                result.append(p.val)
                p = p.right

        return result
        '''

        return self.inorder(root)

    def inorder(self,r):
        if r:
            return self.inorder(r.left) + [r.val] + self.inorder(r.right)
        else:
            return []
```

```
#
# @lc app=leetcode.cn id=95 lang=python3
#
# [95] 不同的二叉搜索树 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def generateTrees(self, n: int) -> List[TreeNode]:
        if n == 0:
            return []
        return self.get_trees(1,n)

    def get_trees(self,start,end):
        res = []
        if start > end:
            # 空子树情况
            return [None]
        for i in range(start,end+1):
            lefts = self.get_trees(start,i-1)
            rights = self.get_trees(i+1,end)
            # lefts 和 rights 有可能是空的[None]
            for l in lefts:
                for r in rights:
                    root = TreeNode(i)
                    root.left = l
                    root.right = r
                    res.append(root)
        return res
```

```
#
# @lc app=leetcode.cn id=96 lang=python3
#
```

```
# [96] 不同的二叉搜索树
#
class Solution:
    def numTrees(self, n: int) −> int:
        f = [0 for _ in range(n+1)]
        f[0] = f[1] = 1
        for k in range(2,n+1):
            for i in range(k+1):
                f[k] += f[i−1]∗f[k−i]
        return f[n]
```

```
#
# @lc app=leetcode.cn id=97 lang=python3
#
# [97] 交错字符串
#
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) −> bool:
        l1, l2, l3 = len(s1), len(s2), len(s3)
        if l1+l2 != l3:
            return False

        dp = [[True for _ in range(l2+1)] for _ in range(l1+1)]
        # 边界条件
        # 用s1去填
        for i in range(1, l1+1):
            dp[i][0] = dp[i−1][0] and s1[i−1] == s3[i−1]
        # 用s2去填
        for j in range(1, l2+1):
            dp[0][j] = dp[0][j−1] and s2[j−1] == s3[j−1]

        for i in range(1, l1+1):
            for j in range(1, l2+1):
                dp[i][j] = (dp[i−1][j] and s1[i−1] == s3[i+j−1]) or \
                (dp[i][j−1] and s2[j−1] == s3[i+j−1])

        return dp[l1][l2]
```

```
#
# @lc app=leetcode.cn id=98 lang=python3
#
# [98] 验证二叉搜索树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
```

```
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        return self.isOK(root,-float('inf'),float('inf'))

    def isOK(self,root,low,upper):
        if root is None:
            return True
        elif root.val > low and root.val < upper :
            return self.isOK(root.left,low,root.val) and self.isOK(root.right,root.val,upper)
        else:
            return False
```

```
#
# @lc app=leetcode.cn id=99 lang=python3
#
# [99] 恢复二叉搜索树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def recoverTree(self, root: TreeNode) -> None:
        cur, pre = root, None
        first, second = None, None
        stack = []

        while cur or stack:
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                node = stack.pop()
                if pre and pre.val >= node.val:
                    if not first:
                        first = pre
                    second = node

                pre = node
                cur = node.right

        first.val, second.val = second.val, first.val
        '''
        # 定义
        self.pre = None
        self.m1, self.m2 = None, None

        self.inorderTraversal(root)
        self.m1.val, self.m2.val = self.m2.val, self.m1.val
        '''

    # 中序遍历
    def inorderTraversal(self, root):
        if root :
            self.inorderTraversal(root.left)
            if self.pre and self.pre.val > root.val:
                if self.m1 == None:
                    self.m1 = self.pre
                self.m2 = root
            self.pre = root
            self.inorderTraversal(root.right)
```

```
#
# @lc app=leetcode.cn id=100 lang=python3
#
# [100] 相同的树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if p is None and q is None:
            return True
        elif p and q and p.val == q.val:
            return self.isSameTree(p.left,q.left) and self.isSameTree(p.right, q.right)
        elif p or q :
            return False
```

```
#
# @lc app=leetcode.cn id=101 lang=python3
```

```python
#
# [101] 对称二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if root is None:
            return True
        return self.yes(root.left ,root.right)

    def yes(self, left, right):
        if left is None and right is None:
            return True
        if left and right and left.val == right.val:
            if self.yes(left.left, right.right) and self.yes(left.right, right.left):
                return True
        return False
```

```python
#
# @lc app=leetcode.cn id=102 lang=python3
#
# [102] 二叉树的层次遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root :
            return []
        result = [[]]
        self.traverse(root,0, result)
        return result

    def traverse(self,root, level, result):
        if not root:
            return
        if level >= len(result):
            result.append([])
        result[level].append(root.val)
        self.traverse(root.left, level+1,result)
        self.traverse(root.right, level+1,result)
```

```
#
# @lc app=leetcode.cn id=103 lang=python3
#
# [103] 二叉树的锯齿形层次遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root :
            return []
        result = [[]]
        self.traverse(root,0, result,True)
        return result

    def traverse(self,root, level, result, flag):
        if root is None:
            return
        if level >= len(result):
            result.append([])

        if flag:
            result[level].append(root.val)
        else:
            result[level].insert(0,root.val)
        self.traverse(root.left, level+1,result, not flag)
        self.traverse(root.right, level+1,result, not flag)
```

```
#
# @lc app=leetcode.cn id=104 lang=python3
#
# [104] 二叉树的最大深度
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        elif root.left and root.right:
            return 1 + max(self.maxDepth(root.left),self.maxDepth(root.right))
        elif root.left:
            return 1 + self.maxDepth(root.left)
        elif root.right:
            return 1 + self.maxDepth(root.right)
        else:
            return 1
```

```
#
# @lc app=leetcode.cn id=105 lang=python3
#
# [105] 从前序与中序遍历序列构造二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if inorder:
            # 前序的头就是root
            # 中序中,root左边就是左子树,右边是右子树
            idx = inorder.index(preorder.pop(0))
            root = TreeNode(inorder[idx])
            # 递归构造子树先left后right
            root.left = self.buildTree(preorder, inorder[0:idx])
            root.right = self.buildTree(preorder, inorder[idx+1:])
            return root
        else:
            return None
```

```
#
```

```
# @lc app=leetcode.cn id=106 lang=python3
#
# [106] 从中序与后序遍历序列构造二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:
        if inorder :
            # 后序的尾部就是root
            # 中序中,root值左边就是左子树,右边是右子树
            idx = inorder.index(postorder.pop())
            root = TreeNode(inorder[idx])
            # 递归构造子树先right后left
            root.right = self.buildTree(inorder[idx+1:],postorder)
            root.left  = self.buildTree(inorder[0:idx],postorder)
            return root
        else:
            return None
```

```
#
# @lc app=leetcode.cn id=107 lang=python3
#
# [107] 二叉树的层次遍历 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:

        if not root:
            return []
        # use stack
        stack = [[root]]
        res = []
        while stack:
            # 取出最新装入的list
            top = stack.pop()
            # 一直在头部插入以达到倒序
            res.insert(0, [t.val for t in top])
            # 向下新一轮扫描
            temp = []
            for node in top:
                if node.left :
                    temp.append(node.left)
                if node.right :
                    temp.append(node.right)
            if temp:
                stack.append(temp)
        return res
        '''
        # 递归法
        if not root:
            return []
        result = [[]]
        self.traverse(root,0, result)
        result.reverse()
        return result
        '''

    def traverse(self,root,level,result):
        if root is None:
            return
        if level >= len(result):
            result.append([])
        result[level].append(root.val)
        self.traverse(root.left,level+1,result)
        self.traverse(root.right,level+1,result)
```

```
#
# @lc app=leetcode.cn id=108 lang=python3
#
# [108] 将有序数组转换为二叉搜索树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
```

```python
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        if not nums :
            return None
        mid = len(nums)//2

        root = TreeNode(nums[mid])
        root. left = self.sortedArrayToBST(nums[:mid])
        root. right = self.sortedArrayToBST(nums[mid+1:])

        return root
```

```python
#
# @lc app=leetcode.cn id=109 lang=python3
#
# [109] 有序链表转换二叉搜索树
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self. left = None
#         self. right = None

class Solution:
    def sortedListToBST(self, head: ListNode) -> TreeNode:
        '''
        if not head :
            return None
        if not head.next:
            return TreeNode(head.val)

        slow = head
        fast = head.next.next
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        head2 = slow.next
        slow.next = None
        root = TreeNode(head2.val)
        root. left = self.sortedListToBST(head)
        root.right = self.sortedListToBST(head2.next)
        return root
        '''

        if not head :
            return None
        nums = []
        while head:
            nums.append(head.val)
            head = head.next
        return self.sortedArrayToBST(nums)

    def sortedArrayToBST(self, nums):
        if not nums :
            return None
        mid = len(nums)//2

        root = TreeNode(nums[mid])
        root.left = self.sortedArrayToBST(nums[:mid])
        root.right = self.sortedArrayToBST(nums[mid+1:])

        return root
```

```python
#
# @lc app=leetcode.cn id=110 lang=python3
#
# [110] 平衡二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        return self.check(root) != -1

    def check(self,root):
        if root is None:
            return 0
        l = self.check(root.left)
        r = self.check(root.right)
        if l == -1 or r == -1 or abs(l-r)>1:
            return -1
        return 1 + max(l,r)
```

```
#
# @lc app=leetcode.cn id=111 lang=python3
#
# [111] 二叉树的最小深度
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def minDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        if root.left is None or root.right is None:
            return self.minDepth(root.left) + self.minDepth(root.right) + 1
        return min(self.minDepth(root.left) , self.minDepth(root.right)) + 1
```

```
#
# @lc app=leetcode.cn id=112 lang=python3
#
# [112] 路径总和
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if root is None:
            return False

        sum -= root.val
        if sum == 0 and root.left is None and root.right is None:
            return True
        left = self.hasPathSum(root.left,sum)
        right = self.hasPathSum(root.right,sum)
        return left or right
```

```
#
# @lc app=leetcode.cn id=113 lang=python3
#
# [113] 路径总和 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        if root is None:
            return []
        result = []
        self.dfs(root, sum, [], result)
        return result

    def dfs(self,root,sum,path,result):
        if root is None:
            return
        if root.left is None and root.right is None and sum == root.val:
            path.append(root.val)
            result.append(path)

        self.dfs(root.left, sum - root.val, path + [root.val], result)
        self.dfs(root.right, sum - root.val, path + [root.val], result)
```

```
#
# @lc app=leetcode.cn id=114 lang=python3
#
# [114] 二叉树展开为链表
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def flatten(self, root: TreeNode) -> None:
        if root is None:
            return

        self.flatten(root.left)
        self.flatten(root.right)

        if root.left is None:
            return

        # 左子树插到root和root.right之间
        p = root.left
        # 左子链的最后一个节点
        while p.right:
            p = p.right
        p.right = root.right
        root.right = root.left
        root.left = None
```

```
#
# @lc app=leetcode.cn id=115 lang=python3
#
# [115] 不同的子序列
#
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        if s is None or t is None :
            return 0
        ls = len(s)
        lt = len(t)
        dp = [ [0 for _ in range(lt+1) ] for _ in range(ls+1)]

        # init
        # 当子串长度为0时，所有次数都是1
        # 当母串长度为0时，所有次数都是0 (默认是0,不用重复了)
        for i in range(ls+1):
            dp[i][0] = 1

        for i in range(1,ls+1):
            for j in range(1,lt+1):
                # 跳过上一个字符串匹配过程
                dp[i][j] = dp[i-1][j]
                # 要匹配的话
                if s[i-1] == t[j-1]:
                    dp[i][j] += dp[i - 1][j - 1]

        return dp[-1][-1]
```

```
#
```

```
# @lc app=leetcode.cn id=116 lang=python3
#
# [116] 填充每个节点的下一个右侧节点指针
#
”””
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: ’Node’ = None, right: ’Node’ = None, next: ’Node’ =
        None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
”””
class Solution:
    def connect(self, root: ’Node’) −> ’Node’:
        if root is None or root.left is None:
            return root
        # 左右链接
        root.left.next = root.right
        if root.next:
            root.right.next = root.next.left
        else:
            root.right.next = None

        self.connect(root.left)
        self.connect(root.right)

        return root
```

```
#
# @lc app=leetcode.cn id=117 lang=python3
#
# [117] 填充每个节点的下一个右侧节点指针 II
#
”””
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: ’Node’ = None, right: ’Node’ = None, next: ’Node’ =
        None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
”””
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        head = root
        dummy = Node(-1)
        prev = dummy
        # dummy 当前行的最左端节点
        while root :
            if root.left :
                prev.next = root.left
                prev = prev.next
            if root.right :
                prev.next = root.right
                prev = prev.next
            root = root.next
            # 行的尾部
            if root is None:
                # dummy.next为前面prev.next 第一次赋值的节点
                root = dummy.next
                #前面链接断开,开始新的一行
                dummy.next = None
                # prev值新的
                prev = dummy
        return head
```

```
#
# @lc app=leetcode.cn id=118 lang=python3
#
# [118] 杨辉三角
#
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        # 全部都用1先填充
        out = [[1]*(i+1) for i in range(numRows)]
        for r in range(numRows):
            for col in range(1,r):
                out[r][col] = out[r-1][col-1] + out[r-1][col]
        return out
```

```
#
# @lc app=leetcode.cn id=119 lang=python3
#
# [119] 杨辉三角 II
#
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        '''
        if rowIndex == 0:
            return [1]
        rowIndex += 1
        # 全部都用1先填充
        out = [[1]*(i+1) for i in range(rowIndex)]
        for r in range(rowIndex):
            for col in range(1,r):
                out[r][col] = out[r-1][col-1] + out[r-1][col]
        return out[-1]
        '''
        # 先用1填充
        res = [1]*(rowIndex+1)
        # 从后往前,从上往下覆盖
        for r in range(2,rowIndex+1):
            for col in range(r-1,0,-1):# 逆序
                res[col] += res[col-1]
        return res
```

```
#
# @lc app=leetcode.cn id=120 lang=python3
#
# [120] 三角形最小路径和
#
class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        if not triangle:
            return
        # 倒数第二行到最上面一行
        for i in range( len(triangle)-2, -1, -1):
            # 每行的第一列到最后一列
            for j in range(len(triangle[i])):
                triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1])
        return triangle[0][0]
```

```
#
# @lc app=leetcode.cn id=121 lang=python3
#
# [121] 买卖股票的最佳时机
#
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices:
            return 0
        minelement = float('inf')
        profit = 0
        for i in range(len(prices)):
            minelement = min(minelement, prices[i])
            profit = max(profit, prices[i] - minelement)
        return profit
```

```
#
# @lc app=leetcode.cn id=122 lang=python3
#
# [122] 买卖股票的最佳时机 II
#
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices:
            return 0
        profit = 0
        for i in range(1,len(prices)):
            if prices[i]>prices[i-1]:
                profit += (prices[i]-prices[i-1])
        return profit
```

```
#
# @lc app=leetcode.cn id=123 lang=python3
#
# [123] 买卖股票的最佳时机 III
#
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        '''
        """
        对于任意一天考虑四个变量:
        fstBuy: 在该天第一次买入股票可获得的最大收益
        fstSell: 在该天第一次卖出股票可获得的最大收益
        secBuy: 在该天第二次买入股票可获得的最大收益
        secSell: 在该天第二次卖出股票可获得的最大收益
        分别对四个变量进行相应的更新, 最后secSell就是最大
        收益值(secSell >= fstSell)
        """
        fstBuy, fstSell = -float('inf'),0
        secBuy,secSell= -float('inf'),0
        for i in prices:
            fstBuy = max(fstBuy, -i)
            fstSell = max(fstSell, fstBuy + i)
            secBuy = max(secBuy, fstSell - i)
            secSell = max(secSell, secBuy + i)
        return secSell
        '''
        if not prices:
            return 0
        num = len(prices)

        forward  = [0]*num
        backward = [0]*num
        # 前向
        current_min = prices[0]
        for i in range(1,len(prices)):
            current_min = min(current_min, prices[i])
            forward[i] = max(forward[i-1] , prices[i]-current_min )
        # 后向
        total_max = 0
        current_max = prices[-1]
        for i in range(len(prices) - 2, -1, -1):
            current_max = max(current_max, prices[i])
            backward[i] = max(backward[i+1], current_max - prices[i])
            total_max = max(total_max, backward[i] + forward[i])
        return total_max
```

```
#
# @lc app=leetcode.cn id=124 lang=python3
#
# [124] 二叉树中的最大路径和
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def maxPathSum(self, root: TreeNode) -> int:
        self.res = -float('inf')
        self.maxend(root)
        return self.res

    def maxend(self,root):
        if root is None:
            return 0
        left = self.maxend(root.left)
        right = self.maxend(root.right)
        self.res = max(self.res, left + root.val + right)
        return max(root.val + max(left, right), 0)
```

```
#
# @lc app=leetcode.cn id=125 lang=python3
```

```
#
# [125] 验证回文串
#
class Solution:
    def isPalindrome(self, s: str) -> bool:
        # 检测字符串是否由字母和数字组成
        alnum = [t.lower() for t in s if t.isalnum()]
        leng = len(alnum)
        mid = leng//2
        if leng < 2 :
            return True
        for i in range(mid):
            if alnum[i] != alnum[leng - i -1]:
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=126 lang=python3
#
# [126] 单词接龙 II
#
class Solution:
    def findLadders(self, beginWord: str, endWord: str, wordList: List[str]) -> List[List[str]]:
        import collections
        wordset = set(wordList)

        level = {beginWord}
        parents = collections.defaultdict(set)

        while level and endWord not in parents:
            next_level = collections.defaultdict(set)
            for word in level:
                # 不同位置都可以插入不同字母进行新单词重构
                for i in range(len(beginWord)):
                    for c in 'abcdefghijklmnopqrstuvwxyz':
                        newWord = word[:i] + c + word[i+1:]
                        if newWord in wordset and newWord not in parents:
                            next_level[newWord].add(word)

            level = next_level
            parents.update(next_level)
        res = [[endWord]]
        # parents相当于是逆向
        while res and res[0][0] != beginWord:
            # 确定是等长的
            res = [[p]+r for r in res for p in parents[r[0]]]
        return res
```

```
#
# @lc app=leetcode.cn id=127 lang=python3
#
# [127] 单词接龙
#
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) −> int:
        # 防止时间超出
        wordset = set(wordList)
        # 初始化
        bfs = [(beginWord, 1)]
        while bfs:
            word,length = bfs.pop(0) # 左边弹出
            if word == endWord:
                return length
            for i in range(len(word)):
                for c in ”abcdefghijklmnopqrstuvwxyz”:
                    # 不同位置都可以插入不同字母进行新单词重构
                    newWord = word[:i] + c + word[i + 1:]
                    if newWord in wordset and newWord != word:
                        wordset.remove(newWord)
                        bfs.append((newWord, length + 1))
        return 0
```

```
#
# @lc app=leetcode.cn id=128 lang=python3
#
# [128] 最长连续序列
#
class Solution:
    def longestConsecutive(self, nums: List[int]) −> int:
        maxLen = 0
        while nums:
            n = nums.pop()
            # 往大处搜索
            i1 = n + 1
            while i1 in nums:
                nums.remove(i1)
                i1 += 1
            # 往小处搜索
            i2 = n − 1
            while i2 in nums:
                nums.remove(i2)
                i2 −= 1
            maxLen = max(maxLen, i1 - i2 - 1)
        return maxLen
```

```
#
# @lc app=leetcode.cn id=129 lang=python3
#
# [129] 求根到叶子节点数字之和
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def sumNumbers(self, root: TreeNode) -> int:
        return self.sum_tree(root,0)

    def sum_tree(self,root,sum):
        if root is None:
            return 0
        if root.left is None and root.right is None:
            return sum*10+root.val

        return self.sum_tree(root.left,sum*10+root.val)+ self.sum_tree(root.right,sum*10+root.val)
```

```
#
# @lc app=leetcode.cn id=130 lang=python3
#
# [130] 被围绕的区域
#
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        if len(board) <= 2 or len(board[0])<=2:
            return
        row , col = len(board) , len(board[0])
        # 对边界上的所有点分别进行深度遍历
        # 第一列和最后一列
        for i in range(row):
            self.dfs(board,i,0,      row,col)
            self.dfs(board,i,col-1,row,col)
        # 第一行和最后一行
        for j in range(1,col-1):
            self.dfs(board,0,     j,row,col)
            self.dfs(board,row-1,j,row,col)

        for i in range(row):
            for j in range(col):
                if board[i][j] == ”O”:
                    board[i][j] = ”X”
                if board[i][j] == ”T”:
                    board[i][j] = ”O”
        return

    def dfs(self,board,i,j,row,col):
        if i < 0 or j < 0 or i >= row or j >= col or board[i][j] != ”O”:
            return
        else:
            board[i][j] = ”T”
            self.dfs(board,i-1,j,row,col)
            self.dfs(board,i,j-1,row,col)
            self.dfs(board,i+1,j,row,col)
            self.dfs(board,i,j+1,row,col)
        return
```

```
#
# @lc app=leetcode.cn id=131 lang=python3
#
# [131] 分割回文串
#
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        res = []
        self.dfs(s,res, [], 0)
        return res

    def dfs(self,s, res, path, start):
        if start == len(s):
            res.append(path)
            return
        for i in range(start, len(s)):
            if self.isPalindrome(s, start, i):
                self.dfs(s,res,path +[s[start:i+1]],i + 1)
    # 判断回文
    def isPalindrome(self, s, begin, end):
        while begin < end :
            if s[begin] != s[end]:
                return False
            begin += 1
            end -= 1
        return True
```

```
#
# @lc app=leetcode.cn id=132 lang=python3
#
# [132] 分割回文串 II
#
class Solution:
    def minCut(self, s: str) -> int:
        n = len(s)
        dp = [[False for _ in range(n)] for _ in range(n)]
        # f[0->n](共n+1个) f[n]=-1
        # f(i) [i, n-1]最小裁剪数
        f = [n] *(n+1)
        f[-1] = -1
        # f 从右往左更新
        # dp (i 往左更新,j往右更新)
        for i in range(n-1,-1,-1):
            for j in range(i,n):
                if (s[i] == s[j] and (j - i < 2 or dp[i + 1][j - 1])) :
                    dp[i][j] = True
                    # 如果满足回文的条件
                    # f 选取裁剪更少的方案
                    f[i] = min(f[i], f[j + 1] + 1)
        return f[0]
```

```
#
# @lc app=leetcode.cn id=133 lang=python3
#
# [133] 克隆图
#
"""
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = []):
        self.val = val
        self.neighbors = neighbors
"""
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        if not node:
            return None
        '''
        # BFS
        queue = [node]
        copy_node = Node(node.val)
        visited = {node: copy_node}
        while queue:
            node = queue.pop(0)
            for i in node.neighbors:
                if i in visited:
                    visited[node].neighbors.append(visited[i])
                else:
                    copy_node_ne = Node(i.val)
                    visited[node].neighbors.append(copy_node_ne)
                    visited[i] = copy_node_ne
                    queue.append(i)

        return copy_node
        '''
        # DFS
        stack = [node]
        copy_node = Node(node.val)
        visited = {node: copy_node}
        while stack:
            node = stack.pop()
            for i in node.neighbors:
                if i in visited:
                    visited[node].neighbors.append(visited[i])
                else:
                    copy_node_ne = Node(i.val)
                    visited[node].neighbors.append(copy_node_ne)
                    visited[i] = copy_node_ne
                    stack.append(i)

        return copy_node
```

```
#
# @lc app=leetcode.cn id=134 lang=python3
#
# [134] 加油站
#
class Solution:
    def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
        sumGas = sumCost = 0
        start = 0
        diff = 0
        for i in range(len(gas)):
            sumGas += gas[i]
            sumCost += cost[i]
            diff += gas[i] - cost[i]
            if diff < 0:
                start = i + 1 ## 下一个开始
                diff = 0
        return start if sumGas - sumCost >= 0 else -1
```

```
#
# @lc app=leetcode.cn id=135 lang=python3
#
# [135] 分发糖果
#
class Solution:
    def candy(self, ratings: List[int]) -> int:
        if not ratings:
            return 0
        leng = len(ratings)
        res = [1 for _ in range(leng)]
        for i in range(1,leng):
            # 右边大
            if ratings[i] > ratings[i-1]:
                res[i] = res[i-1] + 1
        for i in range(leng-1 , 0 , -1 ):
            # 左边大
            if ratings[i-1] > ratings[i]:
                res[i-1] = max(res[i]+1 ,res[i-1])
        return sum(res)
```

```
#
# @lc app=leetcode.cn id=136 lang=python3
#
# [136] 只出现一次的数字
#
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        '''
        return 2*sum(set(nums)) - sum(nums)
        '''
        res = 0
        for i in range(len(nums)):
            res = res ^ nums[i]
        return res
```

```
#
# @lc app=leetcode.cn id=137 lang=python3
#
# [137] 只出现一次的数字 II
#
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        return (3 * sum(set(nums)) − sum(nums) ) //2
```

```
#
# @lc app=leetcode.cn id=138 lang=python3
#
# [138] 复制带随机指针的链表
#
”””
# Definition for a Node.
class Node:
    def __init__(self, x: int, next: ’Node’ = None, random: ’Node’ = None):
        self.val = int(x)
        self.next = next
        self.random = random
”””
class Solution:
    def copyRandomList(self, head: ’Node’) −> ’Node’:
        if head is None:
            return None
        # 复制next部分
        headcopy = head
        while headcopy:
            node = Node(headcopy.val)
            node.next = headcopy.next
            headcopy.next = node
            headcopy = node.next
        # 复制random部分
        headcopy = head
        while headcopy:
            if headcopy.random:
                headcopy.next.random = headcopy.random.next
            headcopy = headcopy.next.next

        # 拆分两个单链表
        src = head
        pnew = res = head.next

        while pnew.next:
            src.next = pnew.next
            src = src.next
            pnew.next = src.next
            pnew = pnew.next
        src.next = None
        pnew.next = None

        return res
```

```
#
# @lc app=leetcode.cn id=139 lang=python3
#
# [139] 单词拆分
#
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        n = len(s)
        dp = [False for _ in range(n+1)]
        dp[0] = True

        for i in range(n+1):
            for j in range(i-1,-1,-1):
                if dp[j] and s[j:i] in wordDict:
                    dp[i] = True
                    break

        return dp[-1]
```

```
#
# @lc app=leetcode.cn id=140 lang=python3
#
# [140] 单词拆分 II
#
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        n = len(s)
        dp = [False for _ in range(n+1)]
        dp[0] = True
        # prev true 表示s[j,i)是一个合法单词,从j处切开
        prev = [[False for _ in range(n)] for _ in range(n+1) ]

        for i in range(n+1):
            for j in range(i-1,-1,-1):
                if dp[j] and s[j:i] in wordDict:
                    dp[i] = True
                    prev[i][j] = True

        res = []
        self.dfs(s,prev,n,[], res)
        return res

    def dfs(self,s,prev,cur,path,res):
        if cur == 0:
            # 终止条件
            temp = ”␣”.join(list(reversed(path)))
            res.append(temp)
            return

        for i in range(cur-1,-1,-1):
            if prev[cur][i]:
                self.dfs(s,prev,i,path+[s[i:cur]], res)
```

```
#
# @lc app=leetcode.cn id=141 lang=python3
#
# [141] 环形链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        '''
        try:
            slow = head
            fast = head.next
            while slow is not fast:
                slow = slow.next
                fast = fast.next.next
            return True
        except:
            return False
        '''
        fast = slow = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                return True
        return False
```

```
#
# @lc app=leetcode.cn id=142 lang=python3
#
# [142] 环形链表 II
#
# Definition for singly-linked list.
# class ListNode:
```

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

class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        fast  = slow = head
        while fast and fast.next:
            slow = slow.next
            fast  = fast.next.next
            if slow == fast:
                #相遇了
                res  = head
                while res != slow:
                    slow  = slow.next
                    res  = res.next
                return res
        return None
```

```
#
# @lc app=leetcode.cn id=143 lang=python3
#
# [143] 重排链表
#
# Definition for  singly-linked  list .
# class ListNode:
#     def __init__(self, x):
#         self . val  = x
#         self . next  = None

class Solution:
    def reorderList( self , head: ListNode) -> None:
        if  head is None or head.next is None:
            return head
        p1 ,p2 = head , head
        while p2 and p2.next :
            p1 = p1.next
            p2 = p2.next.next
        # head2 是后面半部分
        head2 = p1.next
        p1.next = None
        # head head2 对应前后两部分

        cur  = head2
        rever  = None
        # 反转
```

```python
        while cur:
            temp = cur.next
            cur.next = rever
            rever = cur
            cur = temp

        # head rever 两个合并
        p1 = head
        while rever:
            # 两个链的下一个
            temp = p1.next
            temp2 = rever.next
            # 链接好
            p1.next = rever
            rever.next = temp
            # 下一个循环
            p1 = temp
            rever = temp2
        return head
```

```python
#
# @lc app=leetcode.cn id=144 lang=python3
#
# [144] 二叉树的前序遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        result = []
        stack = []
        stack.append(root)

        while stack:
            p = stack.pop()
            result.append(p.val)
            if p.right:
                stack.append(p.right)
            if p.left:
```

```
            stack.append(p.left)
        return result
```

```
#
# @lc app=leetcode.cn id=145 lang=python3
#
# [145] 二叉树的后序遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        result = []
        stack = []
        stack.append(root)
        while stack:
            p = stack.pop()
            result.append(p.val)
            if p.left :
                stack.append(p.left)
            if p.right:
                stack.append(p.right)
        return result[::-1]
```

```
#
# @lc app=leetcode.cn id=146 lang=python3
#
# [146] LRU缓存机制
#
class LRUCache:
    def __init__(self, capacity: int):
        self.capacity = capacity
        self.cache = {}
        self.queue = []

    def update(self,key):
        # 移到头部去
        self.queue.remove(key)
        self.queue.insert(0,key)
```

```python

    def get(self, key: int) -> int:
        if key in self.cache:
            self.update(key)
            return self.cache[key]
        else:
            return -1

    def put(self, key: int, value: int) -> None:
        if not key or not value:
            return None
        if key in self.cache: # 已经在了
            self.queue.remove(key)
        elif len(self.queue) == self.capacity: # 满了
            del self.cache[self.queue.pop()]

        self.cache[key] = value
        self.queue.insert(0,key)

# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

```python
#
# @lc app=leetcode.cn id=147 lang=python3
#
# [147] 对链表进行插入排序
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        dummy = ListNode(-1000)
        dummy.next = head
        p = dummy
        cur = head
        while cur and cur.next:
            val = cur.next.val
            # 顺序的
            if cur.val < val:
                cur = cur.next
                continue
            # 找到p(小于的最后一个节点)
            # 这个相当于p重新初始化
            if p.next.val > val:
                p = dummy
            while p.next.val < val:
                p = p.next
            # 右边的节点插入到左边去
            next_step = cur.next
            cur.next = cur.next.next
            next_step.next = p.next
            p.next = next_step
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=148 lang=python3
#
# [148] 排序链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if head is None or head.next is None:
            return head
        fast = slow = head
        pre = None
        while fast and fast.next:
            fast = fast.next.next
            pre = slow
            slow = slow.next
        pre.next = None
        return self.mergeTwoLists(self.sortList(head), self.sortList(slow))

    def mergeTwoLists(self, l1, l2):
        res = now = ListNode(-1000)
        while l1 and l2:
            if l1.val <= l2.val:
                now.next = l1
                l1 = l1.next
            else:
                now.next = l2
                l2 = l2.next
            now = now.next
        now.next = l1 or l2
        return res.next
```

```
#
# @lc app=leetcode.cn id=149 lang=python3
#
# [149] 直线上最多的点数
#
class Solution:
    def maxPoints(self, points: List[List[int]]) -> int:
        if points is None :
            return 0
        res = 0
        # 两重循环
        # 双重字典
        for i in range(len(points)):
            line_map = {}
            same = max_point_num = 0
            for j in range(i + 1, len(points)):
                dx, dy = points[j][0] - points[i][0], points[j][1] - points[i][1]
                # 同一个点
                if dx == 0 and dy == 0:
                    same += 1
                    continue
                # 去除最大公约数部分
                gcd = self.generateGCD(dx, dy)
                if gcd != 0:
                    dx //= gcd
                    dy //= gcd

                if dx in line_map:
                    if dy in line_map[dx]:
                        line_map[dx][dy] += 1
                    else:
                        line_map[dx][dy] = 1
                else:
                    line_map[dx] = {}
                    line_map[dx][dy] = 1
                max_point_num = max(max_point_num, line_map[dx][dy])
            res = max(res, max_point_num + same + 1)
        return res

    # 辗转相除法求最大公约数
    def generateGCD(self, x, y):
        if y == 0:
            return x
        else:
            return self.generateGCD(y, x % y)
```

```
#
# @lc app=leetcode.cn id=150 lang=python3
#
# [150] 逆波兰表达式求值
#
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        nums = []
        for t in tokens:
            if t not in ['+','-','*','/']:
                nums.append(int(t))
            else:
                r = nums.pop()
                l = nums.pop()
                if t == '+':
                    temp = l+r
                elif t == '-':
                    temp = l-r
                elif t == '*':
                    temp = l*r
                elif t == '/':
                    if l*r < 0 and l%r != 0:
                        temp = l//r + 1
                    else:
                        temp = l//r
                nums.append(temp)
        return nums.pop()
```

```
#
# @lc app=leetcode.cn id=151 lang=python3
#
# [151] 翻转字符串里的单词
#
class Solution:
    def reverseWords(self, s: str) -> str:
        if not s:
            return s

        '''
        temp = s.split(' ')
        temp = [t for t in temp if len(t) > 0]
        temp.reverse()
        return ' '.join(temp)
        '''
        s = s + "␣"
        l = 0
        res = []
        for i in range(l,len(s)):
            if s[i] == "␣":
                if l != i:
                    res.append(s[l:i])
                l = i + 1

        res.reverse()
        return "␣".join(res)
```

```
#
# @lc app=leetcode.cn id=152 lang=python3
#
# [152] 乘积最大子序列
#
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        if not nums:
            return 0
        maxtmp = mintmp = res = nums[0]
        for i in range(1,len(nums)):
            maxtmp ,mintmp =max(nums[i] , nums[i]*maxtmp ,nums[i]*mintmp) ,\
                            min(nums[i] , nums[i]*maxtmp ,nums[i]*mintmp)
            res = max(maxtmp,res)
        return res
```

```
#
# @lc app=leetcode.cn id=153 lang=python3
#
# [153] 寻找旋转排序数组中的最小值
#
class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) == 1 or nums[0] < nums[-1]: # 升序
            return nums[0]
        l , r = 0, len(nums)-1
        while l < r :
            mid = (l+r)//2
            # 左边
            if nums[0] <= nums[mid]:
                l = mid + 1
            # 在右边
            else:
                r = mid
        return nums[l]
```

```
#
# @lc app=leetcode.cn id=154 lang=python3
#
# [154] 寻找旋转排序数组中的最小值 II
#

class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) == 1 or nums[0] < nums[-1]: # 升序
            return nums[0]

        l , r = 0, len(nums)-1
        while l < r :
            mid = (l+r)//2
            # 左边
            if nums[mid] > nums[r]:
                l = mid + 1
            # 在右边
            elif nums[mid] < nums[r]:
                r = mid
            # nums[mid] == nums[r]情况
            else:
                r -= 1
        return nums[l]
```

```
#
# @lc app=leetcode.cn id=155 lang=python3
#
# [155] 最小栈
#
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, x: int) -> None:
        self.stack.append(x)
        if len(self.min_stack) == 0:
            self.min_stack.append(x)
            return
        # x 和栈尾 哪个小压哪个
        if x <= self.min_stack[−1]:
            self .min_stack.append(x)
        else:
            self .min_stack.append(self.min_stack[−1])

    def pop(self) −> None:
        if len( self .stack)>0:
            self .min_stack.pop()
            self .stack.pop()

    def top( self ) −> int:
        if len( self .stack)>0:
            return self.stack[−1]
        return None

    def getMin(self) −> int:
        if len( self .min_stack)>0:
            return self.min_stack[−1]
        return None

# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(x)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

```
#
# @lc app=leetcode.cn id=160 lang=python3
#
# [160] 相交链表
#
# Definition for singly−linked list .
# class ListNode:
#     def ___init___(self, x):
#         self . val = x
#         self . next = None

class Solution:
    def getIntersectionNode(self , headA: ListNode, headB: ListNode) −> ListNode:
        p1, p2 = headA, headB
        # 初始化两个运动结点p1和p2
        while p1 != p2:
            # 只要两个结点还未相遇
            p1 = headB if p1 is None else p1.next
            # 如果p1走到了链表A的末尾，则换到链表B上
            p2 = headA if p2 is None else p2.next
            # 如果p2走到了链表B的末尾，则换到链表A上

        return p1
        # 当p1和p2都换到对方的链表上，再次相遇后第一个结点即为首个公共结点，否则为None
```

```
#
# @lc app=leetcode.cn id=162 lang=python3
#
# [162] 寻找峰值
#

class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 1:
            return 0

        l,r = 0,len(nums)-1
        while l <= r:
            mid = (l+r) //2
            if (mid == 0 or nums[mid]>nums[mid-1]) and (mid == n - 1 or nums[mid]>nums[mid+1]):
                return mid
            elif mid > 0 and nums[mid-1] > nums[mid]:
                r = mid -1
            else:
                l = mid + 1
```

```
#
# @lc app=leetcode.cn id=165 lang=python3
#
# [165] 比较版本号
#
class Solution:
    def compareVersion(self, version1: str, version2: str) -> int:
        vs1 = version1.split('.')
        vs2 = version2.split('.')
        l1 , l2 = len(vs1) , len(vs2)
        if (l1 > l2):
            vs2 += [0] *(l1-l2)
        elif l1 < l2:
            vs1 += [0] *(l2-l1)
        n = max(l1,l2)
        for i in range(n):
            if int(vs1[i]) > int(vs2[i]):
                return 1
            elif int(vs1[i]) < int(vs2[i]):
                return -1
        return 0
```

```
#
# @lc app=leetcode.cn id=167 lang=python3
#
# [167] 两数之和 II - 输入有序数组
#
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        l = 0
        r = len(numbers)-1
        while l <= r:
            temp = numbers[l]+numbers[r]
            if temp == target:
                return [l+1 , r+1]
            elif temp < target :
                l += 1
            elif temp > target:
                r -= 1
```

```
#
# @lc app=leetcode.cn id=168 lang=python3
#
# [168] Excel表列名称
#
class Solution:
    def convertToTitle(self, n: int) -> str:
        capitals = [chr(x) for x in range(ord('A'), ord('Z')+1)]
        result = []

        while n > 0:
            result.append(capitals[(n-1)%26])
            n = (n-1) // 26
        result.reverse()
        return ''.join(result)
```

```
#
# @lc app=leetcode.cn id=169 lang=python3
#
# [169] 多数元素
#
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        nums.sort()
        return nums[len(nums)//2]
```

```
#
# @lc app=leetcode.cn id=171 lang=python3
#
# [171] Excel表列序号
#
class Solution:
    def titleToNumber(self, s: str) -> int:
        res = 0
        for i in s:
            res = res*26 + ord(i)-ord('A')+1
        return res
```

```
#
# @lc app=leetcode.cn id=172 lang=python3
#
# [172] 阶乘后的零
#
class Solution:
    def trailingZeroes(self, n: int) -> int:
        count = 0
        while n >0:
            n //= 5
            count += n
        return count
```

```
#
# @lc app=leetcode.cn id=173 lang=python3
#
# [173] 二叉搜索树迭代器
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class BSTIterator:
    def __init__(self, root: TreeNode):
        # 包含按排序顺序的所有节点的数组
        self.nodes_sorted = []
        self.index = -1
        self._inorder(root)

    def _inorder(self, root):
        if not root:
            return
        self._inorder(root.left)
        self.nodes_sorted.append(root.val)
        self._inorder(root.right)

    def next(self) -> int:
        """
        @return the next smallest number
        """
        self.index += 1
        return self.nodes_sorted[self.index]

    def hasNext(self) -> bool:
        """
        @return whether we have a next smallest number
        """
        return self.index + 1 < len(self.nodes_sorted)

# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

```
#
# @lc app=leetcode.cn id=174 lang=python3
#
# [174] 地下城游戏
#
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        m,n = len(dungeon),len(dungeon[0])
        res = [[0 for _ in range(n)] for _ in range(m)]

        # 逆序遍历
        # 逆序初始化
        res[m-1][n-1] = max(-dungeon[m-1][n-1],0)+1
        for r in range(m-2,-1,-1):
            res[r][n-1] = max(res[r+1][n-1] -dungeon[r][n-1] ,1)
        for c in range(n-2,-1,-1):
            res[m-1][c] = max(res[m-1][c+1] -dungeon[m-1][c] ,1)
        # 从下往上从右往左遍历
        for r in range(m-2,-1,-1):
            for c in range(n-2,-1,-1):
                res[r][c] = max(
                    min(res[r][c+1],res[r+1][c]) - dungeon[r][c],
                    1)
        return res[0][0]
```

```
#
# @lc app=leetcode.cn id=179 lang=python3
#
# [179] 最大数
#
# Python的富比较方法包括__lt__、__gt__分别表示:小于、大于，对应的操作运算符为:“<”、“>”
class LargerNumKey(str):
    def __lt__(x, y):
        return x+y < y+x

class Solution:
    def largestNumber(self, nums: List[int]) -> str:
        '''
        if set(nums) == {0}:
            return '0'
        str_nums = sorted([str(i) for i in nums], key=LargerNumKey,reverse = True)
        largest = "".join(str_nums)
        return largest
        '''

        if set(nums) == {0}:
            return '0'
        for i in range(len(nums), 0, -1):
            tmp = 0
            for j in range(i):
                if not self.compare(nums[j], nums[tmp]):
                    tmp = j
            nums[tmp], nums[i-1] = nums[i-1], nums[tmp]
        return "".join(map(str, nums))

    def compare(self, n1, n2):
        return str(n1) + str(n2) > str(n2) + str(n1)
```

```
#
# @lc app=leetcode.cn id=187 lang=python3
#
# [187] 重复的DNA序列
#
class Solution:
    def findRepeatedDnaSequences(self, s: str) -> List[str]:
        dic,res = {}, set()
        for i in range(len(s)-9):
            dic[s[i:i+10]] = dic.get(s[i:i+10], 0)+1
            if dic[s[i:i+10]] > 1:
                res.add(s[i:i+10])
        return list(res)
```

```
#
# @lc app=leetcode.cn id=188 lang=python3
#
# [188] 买卖股票的最佳时机IV
#
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        #交易次数太多，用贪心
        if k >= len(prices)//2:
            return self.greedy(prices)

        # k=0的时候此时sell为空
        # k小，动态规划
        buy, sell = [-prices[0]]*k , [0]*(k+1)
        for p in prices [1:]:
            for i in range(k):
                # 买的收益 = max(买、买了再买)
                buy[i] = max(buy[i], sell[i-1]-p)
                # 卖的收益 = (卖/买)
                sell[i] = max(sell[i], buy[i]+p)

        return max(sell)

    def greedy(self, prices):
        res = 0
        for i in range(1, len(prices)):
            if prices[i] > prices[i-1]:
                res += prices[i] - prices[i-1]
        return res
```

```
#
# @lc app=leetcode.cn id=189 lang=python3
#
# [189] 旋转数组
#
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        tmp = [0] * len(nums)
        for i in range(len(nums)):
            tmp[(i+k)%len(nums)] = nums[i] #recycle

        for i in range(len(nums)):
            nums[i] = tmp[i]
```

```
#
# @lc app=leetcode.cn id=190 lang=python3
#
# [190] 颠倒二进制位
#
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0
        bitsSize = 31
        while bitsSize > -1 and n :
            res += ((n%2) << bitsSize)
            n = n >> 1
            bitsSize -= 1
        return res
```

```
#
# @lc app=leetcode.cn id=198 lang=python3
#
# [198] 打家劫舍
#
class Solution:
    def rob(self, nums: List[int]) -> int:
        if not nums :
            return 0
        f1 = 0
        f2 = 0
        for i in nums:
            fi = max(f2+i,f1)
            f1 ,f2 = fi ,f1
        return f1
```

```
#
# @lc app=leetcode.cn id=199 lang=python3
#
# [199] 二叉树的右视图
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
```

```

class Solution:
    def rightSideView(self, root: TreeNode) −> List[int]:
        res = []
        self.dfs(root, 0,res)
        return res

    def dfs( self ,root, depth,res):
        if not root:
            return
        if depth >= len(res) :
            res.append(0)
        res[depth] = root.val
        # 先进行左子树的迭代,右子树迭代出来的值会覆盖到之前的上面去
        self.dfs(root.left , depth + 1,res)
        self.dfs(root.right, depth + 1,res)
```

```
#
# @lc app=leetcode.cn id=200 lang=python3
#
# [200] 岛屿数量
#
class Solution:
    def numIslands(self, grid: List[List[str]]) −> int:
        if not grid:
            return 0
        m,n = len(grid),len(grid[0])

        res = 0
        for r in range(m):
            for c in range(n):
                if grid[r][c] == ”1”:
                    res += 1
                    self.dfs(grid,r,c,m,n)
        return res

    def dfs( self ,grid,i,j,row,col):
        # 终止条件
        if i < 0 or j < 0 or i >= row or j >= col or grid[i][j] == ”0”:
            return
        # 合法的话置位
        grid[i][j] = ”0”
        self.dfs(grid,i−1,j,row,col)
        self.dfs(grid,i,j−1,row,col)
        self.dfs(grid,i+1,j,row,col)
        self.dfs(grid,i,j+1,row,col)
```

```python
#
# @lc app=leetcode.cn id=201 lang=python3
#
# [201] 数字范围按位与
#
class Solution:
    def rangeBitwiseAnd(self, m: int, n: int) -> int:
        '''
        # 时间溢出
        res = m
        for i in range(m+1,n+1):
            res = res & i
            if res == 0 :
                break
        return res
        '''

        i = 0
        while m != n:
            m >>= 1
            n >>= 1
            i += 1
        return m << i
```

```python
#
# @lc app=leetcode.cn id=202 lang=python3
#
# [202] 快乐数
#
class Solution:
    def isHappy(self, n: int) -> bool:
        mem = set()
        while n != 1:
            # 求和
            n = sum([int(i) ** 2 for i in str(n)])
            if n in mem:
                # 陷入死循环了
                return False
            else:
                mem.add(n)
        else:
            return True
```

```python
#
# @lc app=leetcode.cn id=203 lang=python3
#
```

```
# [203] 移除链表元素
#

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        dummy = ListNode(-1)
        dummy.next = head
        prev, curr = dummy, head
        while curr :
            if curr.val == val:
                # prev 跟上了 curr
                prev.next = curr.next
            else:
                prev = curr
            curr = curr.next
        return dummy.next
```

```
#
# @lc app=leetcode.cn id=204 lang=python3
#
# [204] 计数质数
#
class Solution:
    def countPrimes(self, n: int) -> int:
        if n <= 2:
            return 0
        res = [0,0]+ [1]*(n-2)
        for i in range(2,n):
            # 这些没改过
            if res[i] == 1:
                for j in range(2,(n-1)//i+1):
                    res[i*j] = 0
        return sum(res)
```

```
#
# @lc app=leetcode.cn id=205 lang=python3
#
# [205] 同构字符串
#
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False

        mapStoT = [0] * 128
        mapTtoS = [0] * 128
        for i in range(len(s)):
            s_num, t_num = ord(s[i]), ord(t[i])
            if mapStoT[s_num] == 0 and mapTtoS[t_num] == 0:
                mapStoT[s_num] = t_num
                mapTtoS[t_num] = s_num
            elif mapTtoS[t_num] != s_num or mapStoT[s_num] != t_num:
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=206 lang=python3
#
# [206] 反转链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        '''
        if head is None or head.next is None:
            return head
        curr = head # 他来往后走
        prev = None # 新的反转的
        while curr:
            # 下一步先保存下来
            nextcurr = curr.next
            # 反转的接上去
            curr.next = prev
            prev  = curr
            # 下一步
            curr = nextcurr
        return prev
        '''
        # 递归方法
        if not head:
            return None
        if not head.next:
            return head
        headNode = self.reverseList(head.next)
        # head headNode 顺序(环)
        head.next.next = head
        # head headNode head(断开)
        head.next = None
        return headNode
```

```
#
# @lc app=leetcode.cn id=207 lang=python3
#
# [207] 课程表
#
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        adjacency = [[] for _ in range(numCourses)]

        flags = [0 for _ in range(numCourses)]
        #(cur,pre)对
        for cur, pre in prerequisites:
            adjacency[pre].append(cur)
        for i in range(numCourses):
            if not self.dfs(i, adjacency, flags):
                return False
        return True

    def dfs(self,i, adjacency, flags):
        # flag标志
        # 0:未访问
        # 1:已被当前节点启动的访问
        #-1:已被其他节点启动的访问
        if flags[i] == -1:
            return True
        if flags[i] == 1:
            return False
        flags[i] = 1
        for j in adjacency[i]:
            if not self.dfs(j, adjacency, flags):
                return False
        flags[i] = -1
        return True
```

```
#
# @lc app=leetcode.cn id=209 lang=python3
#
```

```
# [209] 长度最小的子数组
#
class Solution:
    def minSubArrayLen(self, s: int, nums: List[int]) −> int:
        res = float('inf')
        left = 0
        sumval = 0

        for i in range(len(nums)):
            sumval += nums[i]
            while sumval >= s:
                res = min(res, i−left+1)
                sumval −= nums[left]
                left += 1

        if res != float('inf'):
            return res
        else:
            return 0
```

```
#
# @lc app=leetcode.cn id=210 lang=python3
#
# [210] 课程表 II
#
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[int]]) −> List[int]:
        if not prerequisites:
            return [i for i in range(numCourses)]

        flags = [0 for _ in range(numCourses)]
        inverse_adj = [[] for _ in range(numCourses)]
        for second, first in prerequisites:
            inverse_adj[second].append(first)

        res = []
        for i in range(numCourses):
            if self.dfs(i,inverse_adj, flags, res):
                return []
        return res

    def dfs(self, i, inverse_adj, flags, res):
        """
        :param i: 结点的索引
        :param inverse_adj: 逆邻接表，记录的是当前结点的前驱结点的集合
        :param flags: 记录了结点是否被访问过，2 表示当前正在 DFS 这个结点
        :return: 是否有环
        """
        if flags[i] == 2:
            return True
        if flags[i] == 1:
            return False

        flags[i] = 2
        for precursor in inverse_adj[i]:
            if self.dfs(precursor, inverse_adj, flags, res):
                return True

        flags[i] = 1
        res.append(i)
        return False
```

```
#
# @lc app=leetcode.cn id=213 lang=python3
#
# [213] 打家劫舍 II
#
class Solution:
    def rob(self, nums: List[int]) -> int:
        if not nums:
            return 0
        if len(nums) == 1:
            return nums[0]
        # 奇偶串
        return max(
            self.robb(nums[0:-1]),
            self.robb(nums[1:])
        )

    def robb(self ,nums):
        f1 = 0
        f2 = 0
        for i in nums:
            fi = max(f2+i,f1)
            f1 ,f2 = fi ,f1
        return f1
```

```
#
# @lc app=leetcode.cn id=214 lang=python3
#
# [214] 最短回文串
#
```

```
class Solution:
    def shortestPalindrome(self, s: str) -> str:
        '''
        # 暴力法
        r = s[::-1]
        for i in range(len(s)) :
            if s[0: len(s)-i ] == r[i:] :
                return r[: i] + s
        return ""
        '''
        '''
        # 双指针法
        i = 0
        # 找到从头开始，最长的回文子串
        for j in range(len(s) - 1, -1, -1):
            if s[i] == s[j]:
                i += 1
        if i == len(s):
            return s
        # 后缀
        suffix = s[i :]
        return suffix [::-1] + self.shortestPalindrome(s[: i]) + suffix
        '''

        # kmp算法
        table = self.kmp(s + "#" + s[::-1])
        return s[table [-1]:][::-1] + s


    def kmp(self,p):
        table = [0] * len(p)
        i = 1
        j = 0
        while i < len(p):
            if p[i] == p[j]:
                j += 1
                table[i] = j
                i += 1
            else :
                if j > 0:
                    j = table[j - 1]
                else:
                    i += 1
                    j = 0
        return table
```

```
#
# @lc app=leetcode.cn id=215 lang=python3
#
# [215] 数组中的第K个最大元素
#

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        '''
        nums.sort()
        return nums[-k]
        '''
        return self.qSelect(nums, 0, len(nums) - 1, k)

    def qSelect(self, nums, start, end, k):
        '''
        if start > end:
            return float('inf')
        '''
        # 找一个参照值
        pivot = nums[end]
        left = start
        for i in range(start, end):
            # 比参照大的都移到左边去
            if nums[i] >= pivot:
                nums[left], nums[i] = nums[i], nums[left]
                left += 1
        # 参照值也拉倒左边去
        nums[left], nums[end] = nums[end], nums[left]
        # 左边的个数够没(从0开始到k-1,共k个)
        if left == k-1:
            return nums[left]
        # 还不够
        elif left < k-1:
            return self.qSelect(nums, left + 1, end, k)
        # 太多了
        else:
            return self.qSelect(nums, start, left - 1, k)
```

```
#
# @lc app=leetcode.cn id=216 lang=python3
#
# [216] 组合总和 III
#
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        res = []
        self.dfs(k,n ,1,[], res)
        return res

    def dfs(self,k,target,start,path,res):
        # 终止条件
        if target == 0 and len(path) == k:
            res.append(path)
            return
        elif target < 0 or len(path) > k or start > 9 :
            return

        for i in range(start,10):
            self.dfs(k,target-i,i+1,path+[i],res)
```

```
#
# @lc app=leetcode.cn id=217 lang=python3
#
# [217] 存在重复元素
#
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        return len(nums) != len(set(nums))
```

```
#
# @lc app=leetcode.cn id=219 lang=python3
#
# [219] 存在重复元素 II
#
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        dic = {}
        for key ,val in enumerate(nums):
            if val in dic and key - dic[val]<=k :
                return True
            dic[val] = key
        return False
```

```
#
# @lc app=leetcode.cn id=220 lang=python3
#
# [220] 存在重复元素 III
#
class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) -> bool:
        if t < 0 or k < 0:
            return False
        all_buckets = {}
        # 桶的大小设成t+1更加方便
        bucket_size = t + 1
        for i in range(len(nums)):
            # 放入哪个桶
            bucket_num = nums[i] // bucket_size
            # 桶中已经有元素了
            if bucket_num in all_buckets:
                return True
            # 把nums[i]放入桶中
            all_buckets[bucket_num] = nums[i]
            # 检查前一个桶
            if (bucket_num - 1) in all_buckets and abs(all_buckets[bucket_num - 1] - nums[i]) <= t:
                return True
            # 检查后一个桶
            if (bucket_num + 1) in all_buckets and abs(all_buckets[bucket_num + 1] - nums[i]) <= t:
                return True

            # 如果不构成返回条件，那么当i >= k 的时候就要删除旧桶了，以维持桶中的元素索引跟下一个i+1索引只差不超过k
            if i >= k:
                all_buckets.pop(nums[i-k]//bucket_size)

        return False
```

```
#
# @lc app=leetcode.cn id=221 lang=python3
#
# [221] 最大正方形
#
class Solution:
    def maximalSquare(self, matrix: List[List[str]]) -> int:
        if not matrix:
            return 0
        row ,col = len(matrix) ,len(matrix[0])

        # 多了一行一列
        dp = [ [0 for _ in range(col + 1)] for _ in range(row + 1)]
        res = 0
        for i in range(1, row +1):
            for j in range(1, col + 1):
                if matrix[i - 1][j - 1] == "1":
                    # 否则dp为0，不用操作
                    dp[i][j] = min(dp[i-1][j - 1], dp[i - 1][j], dp[i][j - 1]) + 1
                    res = max(res, dp[i][j] ** 2)
        return res
```

```
#
# @lc app=leetcode.cn id=222 lang=python3
#
# [222] 完全二叉树的节点个数
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def countNodes(self, root: TreeNode) -> int:
        if not root:
            return 0

        # return 1 + self.countNodes(root.left) + self.countNodes(root.right)

        h_l, h_r = 0, 0
        # 计算当前节点左子树的最大高度
        curRoot = root
        while curRoot.left:
            h_l += 1
            curRoot = curRoot.left
        # 计算当前节点右子树的最大高度
        curRoot = root
        if curRoot.right:
            h_r += 1
            curRoot = curRoot.right
            while curRoot.left:
                h_r += 1
                curRoot = curRoot.left

        # 左右子树最大高度相同，说明左子树为满二叉树，在右子树继续递归求解
        if h_l == h_r:
            sumNodes_r = self.countNodes(root.right)
            sumNodes_l = 2**h_l - 1
        # 左子树高度更高，说明右子树为满二叉树，在左子树继续递归求解
        if h_l == h_r + 1:
            sumNodes_l = self.countNodes(root.left)
            sumNodes_r = 2**h_r - 1

        # 返回左子节点个数+右子节点个数+当前根节点
        return sumNodes_l + sumNodes_r + 1
```

```
#
# @lc app=leetcode.cn id=223 lang=python3
#
# [223] 矩形面积
#
class Solution:
    def computeArea(self, A: int, B: int, C: int, D: int, E: int, F: int, G: int, H: int) -> int:
        x = min( C,G )- max(A,E)
        y = min( D,H )- max(B,F)
        return (A-C)*(B-D) + (E-G)*(F-H) - max(x,0)*max(y,0)
```

```
#
# @lc app=leetcode.cn id=224 lang=python3
#
# [224] 基本计算器
#
class Solution:
    def calculate(self, s: str) -> int:
        res = 0
        sign = 1
        stack = []
        i = 0
        while i < len(s):
            c = s[i]
            if c.isdigit():
                start = i
                while i < len(s) and s[i].isdigit():
                    i += 1
                res += sign * int(s[start:i])
                # 因为后加1,不满足while的时候此时的i已经不是数字,需要回退一步,和后边加1对冲
                i -= 1
            elif c == '+':
                sign = 1
            elif c == '-':
                sign = -1
            elif c == "(":
                stack.append(res)
                stack.append(sign)
                res = 0
                sign = 1
            elif c == ")":
                # 现在的res是括号里面的计算结果
                # 需要乘以对应的符号
                res *= stack.pop()
                res += stack.pop()
            i += 1
        return res
```

```
#
# @lc app=leetcode.cn id=225 lang=python3
#
# [225] 用队列实现栈
#
class MyStack:
    def __init__(self):
        self . list = []

    def push(self, x: int) -> None:
        # 尾部压入
        self . list .append(x)

    def pop(self) -> int:
        # 尾部弹出
        if len( self . list ) == 0:
            return
        else:
            temp = self. list [−1]
            del self . list [−1]
            return temp

    def top( self ) -> int:
        if len( self . list ) == 0:
            return
        else:
            return self. list [−1]

    def empty(self) -> bool:
        return len(self. list ) == 0


# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

```
#
```

```
# @lc app=leetcode.cn id=226 lang=python3
#
# [226] 翻转二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        if not root:
            return None
        root.left ,root.right = self.invertTree(root.right) , self.invertTree(root.left)
        return root
```

```
#
# @lc app=leetcode.cn id=228 lang=python3
#
# [228] 汇总区间
#
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:
        if not nums:
            return []
        res = []
        i = 0
        while i < len(nums):
            j = i
            while j+1<len(nums) and (nums[j + 1] - nums[j] <= 1):
                j += 1

            if i == j:
                res.append( str(nums[i]) )
            else:
                res.append( str(nums[i]) + "->" + str(nums[j]) )
            i = j+1
        return res
```

```
#
# @lc app=leetcode.cn id=229 lang=python3
#
# [229] 求众数 II
#
```

```
class Solution:
    def majorityElement(self, nums: List[int]) -> List[int]:
        # 摩尔投票法得到两个大多数
        result1 , result2 = -1, -1
        score1 , score2 = 0 , 0
        for i in range(len(nums)):
            if (result1==nums[i]):
                score1+=1
            elif (result2==nums[i]):
                score2+=1
            elif (score1==0):
                result1=nums[i]
                score1=1
            elif (score2==0):
                result2=nums[i]
                score2=1
            else :
                score1 -= 1
                score2 -= 1

        # 统计两个大多数的出现次数
        time1,time2 = 0 , 0
        for i in range(len(nums)):
            if   (nums[i]==result1): time1+=1
            elif (nums[i]==result2): time2+= 1

        # 得到结果
        result = []
        if (time1>len(nums)/3): result.append(result1)
        if (time2>len(nums)/3): result.append(result2)
        return result
```

```
#
# @lc app=leetcode.cn id=230 lang=python3
#
# [230] 二叉搜索树中第K小的元素
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        '''
        # 方法一
        reslist = self.inorder(root)
        return reslist [k - 1]
        '''
        # 方法二
        # 左子树有多少个点
        n = self.count(root. left )
        if n == k -1 :
            return root.val
        # 递归到左子树
        elif n > k - 1:
            return self.kthSmallest(root. left ,k)
        # 递归到右子树
        else:
            return self.kthSmallest(root.right ,k-1-n)

    def inorder( self ,r):
        if r:
            return self.inorder(r. left ) + [r.val] + self.inorder(r.right)
        else:
            return []

    def count(self ,root):
        if not root :
            return 0
        return self.count(root. left ) + self.count(root.right) + 1
```

```
#
# @lc app=leetcode.cn id=231 lang=python3
#
# [231] 2的幂
#
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        while n > 1:
            n /= 2
        if n == 1:
            return True
        else:
            return False
```

```
#
# @lc app=leetcode.cn id=232 lang=python3
#
```

```python
# [232] 用栈实现队列
#
class MyQueue:
    def __init__(self):
        self.stack = []

    def push(self, x: int) -> None:
        # 尾部加入
        self.stack.append(x)

    def pop(self) -> int:
        temp = self.stack[0]
        self.stack.pop(0)
        return temp

    def peek(self) -> int:
        return self.stack[0]

    def empty(self) -> bool:
        return len(self.stack) == 0

# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

```python
#
# @lc app=leetcode.cn id=233 lang=python3
#
# [233] 数字 1 的个数
#
class Solution:
    def countDigitOne(self, n: int) -> int:
        res = 0
        a = 1
        b = 1
        while n >= 1:
            # 用(x+8)//10来判断一个数是否大于等于2
            # 从低位到高位
            res += (n + 8)//10*a
            if n % 10 == 1:
                res += b
            b += n % 10 * a
            a *= 10
            n //= 10
        return res
```

```
#
# @lc app=leetcode.cn id=234 lang=python3
#
# [234] 回文链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        if head is None:
            return True
        rev = None
        slow = fast = head
        # fast 到尾部
        # slow 到中部
        # rev 前半部分的反向
        while fast and fast.next:
            fast = fast.next.next
            rev, rev.next ,slow = slow, rev , slow.next
        # 奇
        if fast:
            slow = slow.next
        # 一个向左,一个向右
        while rev :
            if rev.val != slow.val:
                return False
            slow = slow.next
            rev = rev.next
        return True
```

```
#
# @lc app=leetcode.cn id=235 lang=python3
#
# [235] 二叉搜索树的最近公共祖先
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        if p is None or q is None or root is None:
            return None
        if p.val < root.val and q.val < root.val:
            return self.lowestCommonAncestor(root.left , p ,q)
        elif p.val > root.val and q.val > root.val:
            return self.lowestCommonAncestor(root.right , p ,q)
        else:
            return root
```

```
#
# @lc app=leetcode.cn id=236 lang=python3
#
# [236] 二叉树的最近公共祖先
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        #若root为空或者root为p或者root为q,说明找到了p或q其中一个
        if(root is None or root== p or root== q):
            return root

        left  = self.lowestCommonAncestor(root.left,p,q)
        right = self.lowestCommonAncestor(root.right,p,q)

        #若左子树找到了p,右子树找到了q,说明此时的root就是公共祖先
        if left and right :
            return root
        # 若左子树是none右子树不是，说明右子树找到了p或q
        if not left :
            return right
        # 同理
        if not right :
            return left
        return None
```

```
#
# @lc app=leetcode.cn id=237 lang=python3
#
# [237] 删除链表中的节点
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteNode(self, node):
        node.val = node.next.val
        node.next = node.next.next
```

```
#
# @lc app=leetcode.cn id=238 lang=python3
#
# [238] 除自身以外数组的乘积
#
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        res = [1] * len(nums)
        right = 1
        for i in range(1, len(nums)):
            res[i] = res[i - 1] * nums[i - 1]

        for i in range(len(nums) - 1, -1, -1):
            res[i] *= right
            right *= nums[i]
        return res
```

```
#
# @lc app=leetcode.cn id=240 lang=python3
#
# [240] 搜索二维矩阵 II
#
class Solution:
    def searchMatrix(self, matrix, target):
        if not len(matrix) or not len(matrix[0]):
            return False
        # 左下角
        r , c = len(matrix) - 1 , 0
        while r >= 0 and c < len(matrix[0]):
            if matrix[r][c] > target :
                # 往上
                r -= 1
            elif matrix[r][c] < target :
                # 往右
                c += 1
            else:
                return True
        return False
```

```
#
# @lc app=leetcode.cn id=242 lang=python3
#
# [242] 有效的字母异位词
#
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        dic1, dic2 = {}, {}
        for item in s:
            dic1[item] = dic1.get(item, 0) + 1
        for item in t:
            dic2[item] = dic2.get(item, 0) + 1
        return dic1 == dic2
```

```
#
# @lc app=leetcode.cn id=257 lang=python3
#
# [257] 二叉树的所有路径
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        if not root:
            return []
        res = []
        self.dfs(root, [] ,res )
        paths = ['->'.join(path) for path in res ]
        return paths

    def dfs(self , node , path ,res):
        # 终止条件 没有子节点
        if not node.left and not node.right:
            res.append(path+[str(node.val)])
            return
        path = path + [str(node.val)]
        if node.left:
            self.dfs(node.left , path , res )
        if node.right:
            self.dfs(node.right , path, res )
```

```
#
# @lc app=leetcode.cn id=258 lang=python3
#
# [258] 各位相加
#
class Solution:
    def addDigits(self, num: int) -> int:
        t = num
        while t >= 10 :
            t = sum([int(char) for char in str(t)])
        return t
```

```
#
# @lc app=leetcode.cn id=260 lang=python3
#
# [260] 只出现一次的数字 III
#
class Solution:
    def singleNumber(self, nums: List[int]) -> List[int]:
        if not nums:
            return []
        # 异或的结果
        diff = 0
        # 得到 x^y
        for num in nums:
            diff ^= num
        # 区分x和y, n&(-n)得到x和y不同的最低位
        diff &= -diff
        res = [0, 0]
        for num in nums:
            # 除了x外，其他&=0的数成对出现
            if num & diff:
                res[0] ^= num
            # 除了y外，其他&=1的数成对出现
            else:
                res[1] ^= num
        return res
```

```
#
# @lc app=leetcode.cn id=263 lang=python3
#
# [263] 丑数
#
class Solution:
    def isUgly(self, num: int) -> bool:
        if num <= 0 :
            return False

        divisors = [2, 3, 5]
        for d in divisors:
            while num % d == 0:
                num /= d
        return num == 1
```

```
#
# @lc app=leetcode.cn id=264 lang=python3
#
# [264] 丑数 II
#
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        ugly = [1]
        i2,i3,i5 = 0,0,0
        idx = 1
        while idx < n :
            newugly = min([ugly[i2]*2 , ugly[i3]*3  ,ugly[i5]*5])
            ugly.append(newugly)

            while ugly[i2]*2<= newugly:
                i2 += 1
            while ugly[i3]*3<= newugly:
                i3 += 1
            while ugly[i5]*5<= newugly:
                i5 += 1
            idx += 1
        return ugly[-1]
```

```
#
# @lc app=leetcode.cn id=268 lang=python3
#
# [268] 缺失数字
#
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        return len(nums)*(len(nums)+1)//2 - sum(nums)
```

```
#
# @lc app=leetcode.cn id=274 lang=python3
#
# [274] H指数
#
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        citations.sort()
        i = 0
        while i<len(citations) and citations[len(citations)-1-i]>i:
            i += 1
        return i
```

```
#
# @lc app=leetcode.cn id=275 lang=python3
#
# [275] H指数 II
#
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        i = 0
        while i<len(citations) and citations[len(citations)-1-i]>i:
            i += 1
        return i
```

```
#
# @lc app=leetcode.cn id=278 lang=python3
#
# [278] 第一个错误的版本
#
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution:
    def firstBadVersion(self, n):
        l ,r = 0 , n-1
        while l <= r:
            mid = (l+r)//2
            if isBadVersion(0) == isBadVersion(mid):
                l = mid + 1
            elif isBadVersion(n) == isBadVersion(mid):
                r = mid -1
        return l
```

```
#
# @lc app=leetcode.cn id=279 lang=python3
#
# [279] 完全平方数
#
class Solution:
    def numSquares(self, n: int) -> int:
        dp = list(range(n+1))
        for i in range(2,n+1):
            for j in range(1,int(i**(0.5))+1):
                dp[i]=min(dp[i],dp[i-j*j]+1)
        return dp[-1]
```

```
#
# @lc app=leetcode.cn id=283 lang=python3
#
# [283] 移动零
#
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        '''
        zeros = []
        for i in range(len(nums)):
            if nums[i] == 0 :
                zeros.append(i)

        for i in zeros[::-1]:
            nums.pop(i)
            nums.append(0)
        return nums
        '''
        j = 0
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[j] = nums[i]
                j += 1
        for i in range(j,len(nums)):
            nums[i] = 0
```

```
#
# @lc app=leetcode.cn id=287 lang=python3
#
# [287] 寻找重复数
#
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        l,r = 0,len(nums)-1
        while l < r :
            mid = (l+r)//2
            cnt = 0
            for num in nums:
                if num <= mid :
                    cnt += 1

            if cnt > mid :
                r = mid
            else:
                l = mid + 1
        return l
```

```
#
# @lc app=leetcode.cn id=289 lang=python3
#
# [289] 生命游戏
#
class Solution:
    def gameOfLife(self, board: List[List[int]]) -> None:
        '''
        # 卷积的思想
        import numpy as np
        r,c = len(board),len(board[0])
        #下面两行做zero padding
        board_exp=np.zeros((r+2,c+2))
        board_exp[1:-1,1:-1]=np.array(board)
        #设置卷积核
        kernel=np.array  ([[1,1,1],[1,0,1],[1,1,1]])
        #开始卷积
        for i in range(1,r+1):
            for j in range(1,c+1):
                #统计细胞周围8个位置的状态
                temp_sum=np.sum(kernel*board_exp[i-1:i+2,j-1:j+2])
                #按照题目规则进行判断
                if board_exp[i,j]==1:
                    if temp_sum<2 or temp_sum>3:
                        board[i-1][j-1]=0
                else :
                    if temp_sum==3:
                        board[i-1][j-1]=1
        return
        '''

        """
        方法二:两次遍历
        第一次遍历时也是分两种情况:
            若活细胞变成了死细胞,由1->-1
            若死细胞变成了活细胞,由0->2
        第二次遍历则是将2(活)->1, -1(死)->0
        """
        row_len, col_len = len(board), len(board[0])
        for row in range(row_len):
            for col in range(col_len):
                lives = self.count(board,row, col,row_len ,col_len )
                if board[row][col] == 1:
                    if lives < 2 or lives > 3:
                        board[row][col] = -1
                else:
                    if lives == 3:
                        board[row][col] = 2
        # 第二次遍历,恢复更改的值
        for row in range(row_len):
            for col in range(col_len):
                if board[row][col] == 2:
                    board[row][col] = 1
                elif board[row][col] == -1:
                    board[row][col] = 0
        return

    def count(self,board,row, col ,row_len ,col_len ):
        lives = 0
        start_row ,end_row = max(0, row - 1) ,min(row_len-1, row+1)
        start_col, end_col = max(0, col - 1), min(col_len-1, col+1)
        for r in range(start_row, end_row+1):
            for c in range(start_col, end_col+1):
                if board[r][c] in [-1, 1] and not (r == row and c == col):
                    lives += 1
        return lives
```

```
#
# @lc app=leetcode.cn id=290 lang=python3
#
# [290] 单词规律
#
class Solution:
    def wordPattern(self, pattern: str, str: str) -> bool:
        '''
        word_list = str.split(' ')
        pattern_list = list(pattern)
        if len(word_list) != len(pattern_list):
            return False
        for i, word in enumerate(word_list):
            idx = word_list.index(word)
            idx2 = pattern_list.index(pattern[i])
            if idx != idx2:
                return False
        return True
        '''

        words = str.split(" ")
        hash_table_pattern = {}
        hash_table_words = {}

        if len(words) != len(pattern):
            return False
        #第一步
        for i, letter in enumerate(pattern):
            if letter in hash_table_pattern:
                if hash_table_pattern[letter] != words[i]:
                    return False
            else:
                hash_table_pattern[letter] = words[i]
        #第二步
        for i, word in enumerate(words):
            if word in hash_table_words:
                if hash_table_words[word] != pattern[i]:
                    return False
            else:
                hash_table_words[word] = pattern[i]
        return True
```

```
#
# @lc app=leetcode.cn id=292 lang=python3
#
# [292] Nim 游戏
#
class Solution:
    def canWinNim(self, n: int) -> bool:
        return n%4 != 0
```

```
#
# @lc app=leetcode.cn id=299 lang=python3
#
# [299] 猜数字游戏
#
```

```
class Solution:
    def getHint(self, secret: str, guess: str) -> str:
        a = b = 0
        dic = {}
        for i in range(len(secret)):
            if secret[i] == guess[i]:
                a += 1
            dic[secret[i]] = dic.get(secret[i],0) + 1
        for i in range(len(guess)):
            if guess[i] in dic and dic[guess[i]] > 0 :
                b += 1
                dic[guess[i]] -= 1
        b -= a
        return f"{a}A{b}B"
```

```
#
# @lc app=leetcode.cn id=300 lang=python3
#
# [300] 最长上升子序列
#
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        if not nums:
            return 0

        '''
        dp = [1] * len(nums)
        for i in range(1,len(nums)):
            for j in range(i):
                # 如果要求非严格递增,将此行 '<' 改为 '<=' 即可
                if(nums[j] < nums[i]):
                    dp[i] = max(dp[i] , dp[j] + 1 )
        return max(dp)
        '''

        up_list = []
        for i in range(len(nums)):
            # 二分查找
            left , right = 0, len(up_list)-1
            while left <= right:
                mid = (left+right)//2
                if up_list[mid] < nums[i]:
                    left = mid+1
                else:
                    right = mid-1
            # 若 left 等于数组长度，则需要添加新值；否则，在 left 位置的值覆盖为新值
```

```python
            if left == len(up_list):
                up_list.append(nums[i])
            else:
                up_list[left] = nums[i]
        return len(up_list)
```

```python
#
# @lc app=leetcode.cn id=303 lang=python3
#
# [303] 区域和检索 - 数组不可变
#
class NumArray:

    def __init__(self, nums: List[int]):
        self.list = [0] *(len(nums)+1)
        for i in range(len(nums)):
            self.list[i+1] = self.list[i] + nums[i]

    def sumRange(self, i: int, j: int) -> int:
        return self.list[j+1] - self.list[i]


# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(i,j)
```

```python
#
# @lc app=leetcode.cn id=304 lang=python3
#
# [304] 二维区域和检索 - 矩阵不可变
#
class NumMatrix:
    def __init__(self, matrix: List[List[int]]):
        if not matrix:
            return
        n, m = len(matrix), len(matrix[0])
        self.sums = [ [0 for j in range(m+1)] for i in range(n+1) ]
        for i in range(1, n+1):
            for j in range(1, m+1):
                self.sums[i][j] = matrix[i-1][j-1] + self.sums[i][j-1] + self.sums[i-1][j] - self.sums[i-1][j-1]

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
        row1, col1, row2, col2 = row1+1, col1+1, row2+1, col2+1
        return self.sums[row2][col2] - self.sums[row2][col1-1] - self.sums[row1-1][col2] + self.sums[row1-1][col1-1]
```

```

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)
```

```
#
# @lc app=leetcode.cn id=306 lang=python3
#
# [306] 累加数
#
class Solution:
    def isAdditiveNumber(self, num: str) -> bool:
        # 题意解读：确认前两个数字，后面即被确认
        # 思路：遍历前两个数字，优化是遍历不超过num_str的一半即可
        # 限制:开头不可为0--->但有'000'的情况，len(num)至少为3
        # 0可以作为一个数字，但不能有以0开头的数字
        len_num = len(num)
        if len_num < 3:
            return False

        for i in range(len_num//2 + 1):
            num1 = num[:i+1]
            # 若num1是以0开头的数字，return Fasle
            if num1[0] =='0' and i >= 1 :
                return False

            for j in range(i+1, len_num//2+i+1):
                num2 = num[i+1:j+1]
                # 若num2以0开头，break
                if num2[0] =='0' and j >= i + 2 :
                    break
                num3 = num[j+1:]
                if self.isValid(num1, num2, num3) and num3:
                    return True
        return False

    def isValid(self, num1, num2, num3):
        # 已确定前两个数字，判断是否合法
        while num3:
            sum_num = str(int(num1) + int(num2))
            if num3.startswith(sum_num):
                num1 = num2
                num2 = sum_num
                num3 = num3[len(sum_num):]
            else:
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=309 lang=python3
#
# [309] 最佳买卖股票时机含冷冻期
#
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if len(prices) < 2:
            return 0
        sale = [0 for _ in range(len(prices))]
        buy = [0 for _ in range(len(prices))]
        cool = [0 for _ in range(len(prices))]

        buy[0] = -prices[0]

        for i in range(1, len(prices)):
            cool[i] = sale[i-1]
            buy[i] = max(buy[i-1], cool[i-1] - prices[i])
            sale[i] = max(sale[i-1], buy[i] + prices[i])

        return max(sale[-1], cool[-1])
```

```
#
# @lc app=leetcode.cn id=313 lang=python3
#
# [313] 超级丑数
#
class Solution:
    def nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
        ugly = [1]
        ls = len(primes)
        ix = [0]*ls
        idx = 1
        while idx < n :
            newugly = min([ugly[ix[i]]*primes[i] for i in range(ls)])
            ugly.append(newugly)
            for i in range(ls):
                while ugly[ix[i]]*primes[i]<= newugly:
                    ix[i] += 1
            idx += 1
        return ugly[-1]
```

```
#
# @lc app=leetcode.cn id=319 lang=python3
#
```

```
# [319] 灯泡开关
#
class Solution:
    def bulbSwitch(self, n: int) -> int:
        return int(math.sqrt(n))
```

```
#
# @lc app=leetcode.cn id=322 lang=python3
#
# [322] 零钱兑换
#
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        if amount == 0:
            return 0
        if not coins :
            return -1

        coins.sort()
        dp = [float('inf')] * (amount + 1)
        dp[0] = 0

        for coin in coins:
            for j in range(coin, amount+1):
                dp[j] = min(dp[j], dp[j - coin] + 1)

        if dp[-1] > amount:
            return -1
        else:
            return dp[-1]
```

```
#
# @lc app=leetcode.cn id=324 lang=python3
#
# [324] 摆动排序 II
#
class Solution:
    def wiggleSort(self, nums: List[int]) -> None:
        # 降序
        nums.sort(reverse=True)
        nums[1::2], nums[0::2] = nums[:len(nums) // 2], nums[len(nums) // 2:]
```

```
#
# @lc app=leetcode.cn id=326 lang=python3
#
# [326] 3的幂
#
```

```
class Solution:
    def isPowerOfThree(self, n: int) -> bool:
        while n > 1:
            n /= 3
        if n == 1:
            return True
        else:
            return False
```

```
#
# @lc app=leetcode.cn id=329 lang=python3
#
# [329] 矩阵中的最长递增路径
#
class Solution:
    def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
        if not matrix :
            return 0

        m, n, res = len(matrix), len(matrix[0]), 0
        # 用于记录每个点的最长递增路径的长度
        cache = [[-1 for _ in range(n)] for _ in range(m)]

        for i in range(m):
            for j in range(n):
                # 每次寻找该点的最长递增路径的长度，并且更新全局的长度
                cur_len = self.dfs(matrix,i, j, cache)
                res = max(res, cur_len)
        return res

    def dfs(self,matrix,i,j,cache):
        if cache[i][j] != -1:
            return cache[i][j]

        m, n, res = len(matrix), len(matrix[0]), 1
        for x_offset, y_offset in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            x, y = i + x_offset, j + y_offset

            if x < 0 or y < 0 or x >= m or y >= n or matrix[x][y] <= matrix[i][j]:
                continue
            # x,y比i,j位置值大
            length = 1 + self.dfs(matrix,x, y, cache)
            res = max(length, res)
        # 记录当前这个点的最长递增路径长度
        cache[i][j] = res
        return res
```

```
#
# @lc app=leetcode.cn id=335 lang=python3
#
# [335] 路径交叉
#
class Solution:
    def isSelfCrossing(self, x: List[int]) -> bool:
        for i in range(len(x)):
            if i + 3 < len(x) and x[i] >= x[i + 2] \
                and x[i + 1] <= x[i + 3]:
                return True
            if i + 4 < len(x) and x[i + 1] == x[i + 3] \
                and x[i] + x[i + 4] >= x[i + 2]:
                return True
            if i + 5 < len(x) and x[i] < x[i + 2] \
                and x[i + 4] < x[i + 2]      \
                and x[i + 2] <= x[i] + x[i + 4]  \
                and x[i + 1] < x[i + 3] \
                and x[i + 3] <= x[i + 1] + x[i + 5]:
                return True
        return False
```

```
#
# @lc app=leetcode.cn id=342 lang=python3
#
# [342] 4的幂
#
class Solution:
    def isPowerOfFour(self, num: int) -> bool:
        # bin(4**0) '0b1'
        # bin(4**1) '0b100'
        # bin(4**2) '0b10000'
        # bin(4**3) '0b1000000'
        return num > 0 and num & (num-1) == 0 and len(bin(num)[3:]) % 2 == 0
```

```
#
# @lc app=leetcode.cn id=343 lang=python3
#
# [343] 整数拆分
#
class Solution:
    def integerBreak(self, n: int) -> int:
        dp = [1]*(n+1)
        # dp[0] = 0
        # dp[1] = 1
        # dp[2] = 1
        for i in range(3,n+1):
            for j in range(1,i):
                dp[i] = max(dp[i],j*(i-j),j*dp[i-j])

        return dp[-1]
```

```
#
# @lc app=leetcode.cn id=344 lang=python3
#
# [344] 反转字符串
#
class Solution:
    def reverseString(self, s: List[str]) -> None:
        n =len(s)
        for i in range(n//2):
            s[i],s[n-i-1] = s[n-i-1],s[i]
```

```
#
# @lc app=leetcode.cn id=345 lang=python3
#
# [345] 反转字符串中的元音字母
#
class Solution:
    def reverseVowels(self, s: str) -> str:
        s = list(s)
        n =len(s)
        l ,r = 0,n-1
        while l < r:
            if s[l] not in 'aeiouAEIOU':
                l += 1
            elif s[r] not in 'aeiouAEIOU':
                r -= 1
            else:
                s[l],s[r] = s[r],s[l]
                l += 1
                r -= 1
        return ''.join(s)
```

```
#
# @lc app=leetcode.cn id=349 lang=python3
#
# [349] 两个数组的交集
#
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        # return list(set(nums1) & set(nums2))
        '''
        res = []
        for i in nums1:
            if i not in res and i in nums2:
                res.append(i)

        return res
        '''
        nums1.sort()
        nums2.sort()
        if not nums1 or not nums2:
            return []
        if nums1[0] == nums2[0]:
            foo = self.intersection(nums1[1:],nums2[1:])
            if foo and foo[0] == nums1[0]:
                return foo
            else:
                return [nums1[0]]+foo
        elif nums1[0]<nums2[0]:
            return self.intersection(nums1[1:],nums2)
        else:
            return self.intersection(nums1,nums2[1:])
```

```
#
# @lc app=leetcode.cn id=350 lang=python3
#
# [350] 两个数组的交集 II
#
class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        nums1.sort()
        nums2.sort()
        res = []
        pos1 = pos2 = 0
        while pos1 < len(nums1) and pos2 < len(nums2):
            if nums1[pos1] == nums2[pos2]:
                res.append(nums1[pos1])
                pos1 += 1
                pos2 += 1
            elif nums1[pos1] < nums2[pos2]:
                pos1 += 1
            else:
                pos2 += 1
        return res
```

```
#
# @lc app=leetcode.cn id=354 lang=python3
```

```
#
# [354] 俄罗斯套娃信封问题
#
class Solution:
    def maxEnvelopes(self, envelopes: List[List[int]]) -> int:
        if not envelopes:
            return 0
        '''
        # 超时
        envelopes.sort(key=lambda x:x[0])
        dp = [1] * len(envelopes)
        for i in range(len(envelopes)):
            for j in range(i):
                if envelopes[i][0] > envelopes[j][0] and envelopes[i][1] > envelopes[j][1]:
                    dp[i] = max(dp[i],dp[j]+1)
        return max(dp)
        '''

        from bisect import bisect_left
        # 在L中查找x,x存在时返回x左侧的位置,x不存在返回应该插入的位置
        # 按w升序,h降序排列
        envelopes.sort(key=lambda x:(x[0], -x[1]))
        up_list = []
        for e in envelopes:
            index = bisect_left(up_list, e[1])
            if index == len(up_list):
                up_list.append(e[1])
            else:
                up_list[index] = e[1]
        return len(up_list)
```

```
#
# @lc app=leetcode.cn id=367 lang=python3
#
# [367] 有效的完全平方数
#
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        '''
        l,r = 1,num
        while l <= r:
            mid = (l+r)//2
            if mid ** 2 == num:
                return True
            elif mid ** 2 < num:
                l = mid +1
            else:
                r = mid -1
        return False
        '''
        x = num
        while x ** 2 > num:
            x = (x+num//x)//2
        return x ** 2 == num
```

```
#
# @lc app=leetcode.cn id=368 lang=python3
#
# [368] 最大整除子集
#
class Solution:
    def largestDivisibleSubset(self, nums: List[int]) -> List[int]:
        nums.sort()
        dp = [[x] for x in nums]
        res = []
        for i in range(len(nums)):
            for j in range(i):
                if nums[i]%nums[j] == 0 and len(dp[j])+1 > len(dp[i]):
                    dp[i] = dp[j] + [nums[i]]
            if len(dp[i])>len(res):
                res = dp[i]
        return res
```

```
#
# @lc app=leetcode.cn id=371 lang=python3
#
# [371] 两整数之和
#
class Solution:
    def getSum(self, a: int, b: int) -> int:
        MAX_INT = 0x7FFFFFFF
        MIN_INT = 0x80000000
        MASK = 0x100000000
        while b:
            a, b = (a ^ b) % MASK, ((a & b) << 1) % MASK
        return a if a <= MAX_INT else ~((a % MIN_INT) ^ MAX_INT)
```

```
#
# @lc app=leetcode.cn id=374 lang=python3
#
# [374] 猜数字大小
#
# The guess API is already defined for you.
```

```python
# @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
# def guess(num: int) -> int:

class Solution:
    def guessNumber(self, n: int) -> int:
        start, end = 1, n
        while start <= end:
            mid = (start + end)//2
            if guess(mid) == 0:
                return mid
            elif guess(mid) == 1:
                start = mid + 1
            else:
                end = mid
```

```python
#
# @lc app=leetcode.cn id=383 lang=python3
#
# [383] 赎金信
#
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        letter_map = {}
        for i in magazine:
            letter_map[i] = letter_map.get(i, 0) + 1
        for i in ransomNote:
            letter_map[i] = letter_map.get(i, 0) - 1
            if letter_map[i]<0:
                return False
        return True
```

```python
#
# @lc app=leetcode.cn id=387 lang=python3
#
# [387] 字符串中的第一个唯一字符
#
class Solution:
    def firstUniqChar(self, s: str) -> int:
        letter_map = {}
        for i in s:
            letter_map[i] = letter_map.get(i, 0) + 1
        for i in range(len(s)):
            if letter_map[s[i]] == 1:
                return i
        return -1
```

```python
#
# @lc app=leetcode.cn id=393 lang=python3
#
# [393] UTF-8 编码验证
#
class Solution:
    def validUtf8(self, data: List[int]) -> bool:
        # cnt表示后面接几个字节字符
        # cnt 从0到0表示一个字符
        cnt = 0
        for d in data:
            if cnt == 0:
                if (d >> 5) == 0b110:
                    cnt = 1
                elif (d >> 4) == 0b1110:
                    cnt = 2
                elif (d >> 3) == 0b11110:
                    cnt = 3
                # 0xxxxxxx 后面不接
                # 这种情况首位不是0就错
                elif (d >> 7):
                    return False
            else:
                # 如果不接10xxxxxx
                if (d >> 6) != 0b10:
                    return False
                cnt -= 1
        return cnt == 0
```

```python
#
# @lc app=leetcode.cn id=410 lang=python3
#
# [410] 分割数组的最大值
#
class Solution:
    def splitArray(self, nums: List[int], m: int) -> int:
        # 最大值最小的范围(单个最大,整体和)
        left = max(nums)
        right = sum(nums)

        while left < right:
            mid = (right + left) // 2
            count = self.count(nums,mid)
            if count > m:
                left = mid + 1
            else:
                right = mid
        return left

    def count(self,nums,mid):
        tmpsum = 0
        count = 1
        for num in nums:
            tmpsum += num
            if tmpsum > mid:
                tmpsum = num
                count += 1
        return count
```

```
#
# @lc app=leetcode.cn id=414 lang=python3
#
# [414] 第三大的数
#
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        nums = list(set(nums))
        if len(nums)<3:
            return max(nums)
        nums.sort()
        return nums[-3]
```

```
#
# @lc app=leetcode.cn id=416 lang=python3
#
# [416] 分割等和子集
#
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        # 背包问题+动态规划
        target = sum(nums)
        if target % 2 == 1:
            return False
        target //= 2

        # 行nums 列对应 目标值
        dp = [[False]*(target+1) for _ in range(len(nums))]
        # 每一列赋值
        if nums[0] <= target:
            dp[0][nums[0]] = True

        for i in range(1,len(nums)):
            for j in range(1,target+1):
                if j >= nums[i]:
                    dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i]]
                else:
                    dp[i][j] = dp[i-1][j]
        return dp[-1][-1]
```

```
#
# @lc app=leetcode.cn id=432 lang=python3
#
# [432] 全 O(1) 的数据结构
#
class AllOne:
    def __init__(self):
        self.lookup = {}

    def inc(self, key: str) -> None:
        if key in self.lookup:
            self.lookup[key] += 1
        else:
            self.lookup[key] = 1

    def dec(self, key: str) -> None:
        if key in self.lookup:
            if self.lookup[key] == 1:
                self.lookup.pop(key)
            else:
                self.lookup[key] -= 1

    def getMaxKey(self) -> str:
        return max(self.lookup.items(), key=lambda x: x[1], default=[""])[0]

    def getMinKey(self) -> str:
        return min(self.lookup.items(), key=lambda x: x[1], default=[""])[0]

# Your AllOne object will be instantiated and called as such:
# obj = AllOne()
# obj.inc(key)
# obj.dec(key)
# param_3 = obj.getMaxKey()
# param_4 = obj.getMinKey()
```

```
#
# @lc app=leetcode.cn id=434 lang=python3
#
# [434] 字符串中的单词数
```

```
#
class Solution:
    def countSegments(self, s: str) -> int:
        if not s:
            return 0
        '''
        segment_count = 0
        for i in range(len(s)):
            if i == 0 and s[i] != ' ':
                segment_count = 1
            elif s[i-1] == ' ' and s[i] != ' ':
                segment_count += 1

        return segment_count
        '''
        s_list = list(s.split("␣"))
        s_list = [i for i in s_list if i != "␣" and i != "" ]
        return len(s_list)
```

```
#
# @lc app=leetcode.cn id=442 lang=python3
#
# [442] 数组中重复的数据
#
class Solution:
    def findDuplicates(self, nums: List[int]) -> List[int]:
        res = []
        for x in nums:
            x = abs(x)
            # 若x出现过了,x-1对应位置的值是负的(减一是为了超出范围)
            if nums[x-1] < 0:
                res.append(x)
            else:
                nums[x-1] *= -1
        return res
```

```
#
# @lc app=leetcode.cn id=443 lang=python3
#
# [443] 压缩字符串
#
class Solution:
    def compress(self, chars: List[str]) -> int:
        # count 几个一样
        # walker 写入的位置
        # runner 往后跑的
        walker, runner = 0, 0

        while runner < len(chars):
            # 写字符
            chars[walker] = chars[runner]
            count = 1

            while runner + 1 < len(chars) and \
            chars[runner] == chars[runner+1] :
                runner += 1
                count += 1

            if count > 1:
                for c in str(count):
                    # 写数字
                    walker += 1
                    chars[walker] = c

            runner += 1
            walker += 1

        return walker
```

```
#
# @lc app=leetcode.cn id=448 lang=python3
#
# [448] 找到所有数组中消失的数字
#
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        '''
        # time Limit Exceeded
        res = []
        leng = len(nums)
        for i in range(leng):
            if i+1 not in nums:
                res.append(i+1)
        return res
        '''
        for num in nums:
            index = abs(num)-1
            if nums[index]>0:
                nums[index] *= -1

        res = []
        for i in range(len(nums)):
            if nums[i] > 0:
                res.append(i+1)
        return res
```

```
#
# @lc app=leetcode.cn id=474 lang=python3
#
# [474] 一和零
#
class Solution:
    def findMaxForm(self, strs: List[str], m: int, n: int) -> int:
        if not strs:
            return 0
        # 准备很多个背包
        dp = [ [0]*(n+1) for _ in range(m+1)]

        for str in strs:
            count0 = str.count('0')
            count1 = str.count('1')

            # 遍历可容纳的背包
            for zeroes in range(m, count0 - 1, -1):
                for ones in range(n, count1 - 1, -1):
                    dp[zeroes][ones] = max(dp[zeroes][ones] ,
                    1 + dp[zeroes - count0][ones - count1])
        return dp[m][n]
```

```
#
# @lc app=leetcode.cn id=485 lang=python3
#
# [485] 最大连续1的个数
#
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        maxval = 0
        tmp = 0
        for i in range(len(nums)):
            if nums[i] != 0 :
                tmp += 1
            else:
                maxval = max(maxval,tmp)
                tmp = 0
        maxval = max(maxval,tmp)
        return maxval
```

```
#
# @lc app=leetcode.cn id=494 lang=python3
```

```
#
# [494] 目标和
#
class Solution:
    def findTargetSumWays(self, nums: List[int], S: int) -> int:
        sum_nums = sum(nums)
        if sum_nums < S or (S + sum_nums)%2 != 0:
            return 0

        target = (S + sum_nums) // 2
        dp = [0]*(target + 1)
        dp[0] = 1
        for num in nums:
            for i in range(target, num-1, -1):
                dp[i] += dp[i - num]
        return dp[-1]
```

```
#
# @lc app=leetcode.cn id=518 lang=python3
#
# [518] 零钱兑换 II
#
class Solution:
    def change(self, amount: int, coins: List[int]) -> int:
        dp = [0] * (amount + 1)
        dp[0] = 1

        for coin in coins:
            for x in range(coin, amount + 1):
                dp[x] += dp[x - coin]
        return dp[amount]
```

```
#
# @lc app=leetcode.cn id=532 lang=python3
#
# [532] 数组中的K-diff数对
#
class Solution:
    def findPairs(self, nums: List[int], k: int) -> int:
        dic = {}
        if k < 0 :
            return 0
        res = 0
        for num in nums:
            dic[num] = dic.get(num,0) + 1
        for num in nums:
            # 值在里面 且 k 不为0
            if dic.get(num-k,0) > 0 and k != 0:
                res += 1
                dic[num-k] = 0
            # k 为0,值有多个
            elif k == 0 and dic.get(num,0) > 1:
                res += 1
                dic[num-k] = 0
        return res
```

```
#
# @lc app=leetcode.cn id=541 lang=python3
#
# [541] 反转字符串 II
#
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        if len(s)<k:
            return s[::-1]
        if len(s)<2*k:
            return s[:k][::-1]+s[k:]
        return s[:k][::-1]+s[k:2*k] + self.reverseStr(s[2*k:], k)
```

```
#
# @lc app=leetcode.cn id=547 lang=python3
#
# [547] 朋友圈
#
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        # 遍历每个人,遍历到过置1
        visited = [0 for _ in range(len(M))]
        # 圈数
        count = 0
        for i in range(len(M)):
            # 等于1表示被别的圈包进去了,等于0表示再开一个圈
            if visited[i] == 0:
                visited[i] = 1
                self.dfs(M, visited, i)
                count += 1
        return count

    # 判断和i认识的都是哪些人
    def dfs(self, M, visited, i):
        # 全1了
        # if sum(visited) == len(M):
```

```
        if all(visited):
            return
        for j in range(len(M)):
            if j != i and visited[j] == 0 and M[i][j] == 1 :
                visited[j] = 1
                self.dfs(M, visited, j)
```

```
#
# @lc app=leetcode.cn id=551 lang=python3
#
# [551] 学生出勤记录 I
#
class Solution:
    def checkRecord(self, s: str) -> bool:
        count = 0
        for i in range(len(s)):
            if s[i] =='A':
                # 大于1个A
                count += 1
                if count > 1:
                    return False
            elif s[i] == 'L' and 0 < i < len(s)-1 \
                and s[i-1] == 'L' == s[i+1]:
                return False
        return True
```

```
#
# @lc app=leetcode.cn id=557 lang=python3
#
# [557] 反转字符串中的单词 III
#
class Solution:
    def reverseWords(self, s: str) -> str:
        return '␣'.join([word[::-1] for word in s.split('␣')])
```

```
#
# @lc app=leetcode.cn id=560 lang=python3
#
# [560] 和为K的子数组
#
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        '''
        # 超时
        same_length = 0
        for start in range(len(nums)):
            sums = 0
            for end in range(start, len(nums)):
                sums += nums[end]
                if sums == k:
                    same_length += 1
        return same_length
        '''

        count = 0
        sums = 0
        dic = {0:1}

        for num in nums:
            sums += num
            count += dic.get(sums-k,0)
            dic[sums] = dic.get(sums,0) + 1

        return count
```

```
#
# @lc app=leetcode.cn id=561 lang=python3
#
# [561] 数组拆分 I
#
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        return sum(nums[::2])
```

```
#
# @lc app=leetcode.cn id=566 lang=python3
#
# [566] 重塑矩阵
#
class Solution:
    def matrixReshape(self, nums: List[List[int]], r: int, c: int) -> List[List[int]]:
        row = len(nums)
        col = len(nums[0])
        if row * col != r*c:
            return nums
        res = [[]]
        for i in range(row):
            for j in range(col):
                k = nums[i][j]
                if len(res[-1]) < c:
                    res[-1].append(k)
                else:
                    res.append([k])
        return res
```

```
#
# @lc app=leetcode.cn id=567 lang=python3
#
# [567] 字符串的排列
#
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        if len(s1) > len(s2):
            return False
        dic = [0] * 26
        for i in range(len(s1)):
            dic[ord(s1[i]) - ord('a')] -= 1
            dic[ord(s2[i]) - ord('a')] += 1

        for i in range(len(s2)-len(s1)):
            if sum(list(map(abs,dic))) == 0 :
                return True
            else:
                # 滑动窗往右滑动
                dic[ord(s2[i+len(s1)]) - ord('a')] += 1
                dic[ord(s2[i]) - ord('a')] -= 1
        return sum(list(map(abs,dic))) == 0
```

```
#
# @lc app=leetcode.cn id=575 lang=python3
#
# [575] 分糖果
#
class Solution:
    def distributeCandies(self, candies: List[int]) -> int:
        return int(min( len(set(candies)) , len(candies)//2))
```

```
#
# @lc app=leetcode.cn id=581 lang=python3
#
# [581] 最短无序连续子数组
#
class Solution:
    def findUnsortedSubarray(self, nums: List[int]) -> int:
        num_sort = nums[:] # 浅拷贝和深拷贝
        num_sort.sort()
        n=len(nums)
        i,j=0,n-1
        while i<n and nums[i]==num_sort[i]:
            i += 1
        while j>i+1 and nums[j]==num_sort[j]:
            j -= 1
        return j-i+1
```

```
#
# @lc app=leetcode.cn id=605 lang=python3
#
# [605] 种花问题
#
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        # 前后补零解决边界问题
        nums=[0]+flowerbed+[0]
        cnt=0
        i=1
        while i<len(flowerbed)+1:
            if nums[i-1]==0 and nums[i]==0 and nums[i+1]==0:
                cnt += 1
                # 可以种花，则需要间隔一个位置，所以+2
                i += 2
            else:
                i+=1
        return cnt>=n
```

```
#
# @lc app=leetcode.cn id=628 lang=python3
#
# [628] 三个数的最大乘积
#
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        nums.sort()
        res1 = nums[-1]*nums[-2]*nums[-3]
        res2 = nums[-1]*nums[0]*nums[1]
        return max(res1,res2)
```

```
#
# @lc app=leetcode.cn id=638 lang=python3
#
# [638] 大礼包
#
class Solution:
    def shoppingOffers(self, price: List[int], special: List[List[int]], needs: List[int]) -> int:
        self.dic = {}
        return self.dfs(price , special , needs)

    def dfs(self,price, special, needs):
        # 买完了
        if sum(needs) == 0:
            return 0
        # 避免重复
        if tuple(needs) in self.dic:
            return self.dic[tuple(needs)]

        res = 0
        # 没有优惠的价格
        # 单个买
        for i in range(len(needs)):
            res += needs[i]*price[i]

        # 买套装
        for sp in special:
            for i in range(len(needs)):
                needs[i] -= sp[i]
            if all(needs[i] >= 0 for i in range(len(needs))):
                res = min(res, self.dfs(price, special, needs) + sp[-1])
            for i in range(len(needs)):
                needs[i] += sp[i]

        self.dic[tuple(needs)] = res
        return res
```

```
#
# @lc app=leetcode.cn id=643 lang=python3
#
# [643] 子数组最大平均数 I
#
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        tmp = maxmean = sum(nums[:k])
        for i in range(k,len(nums)):
            tmp += (nums[i]-nums[i-k])
            maxmean = max(maxmean,tmp)
        return maxmean/k
```

```
#
# @lc app=leetcode.cn id=661 lang=python3
#
# [661] 图片平滑器
#
class Solution:
    def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:
        R, C = len(M), len(M[0])
        res = [[0] * C for _ in range(R)]

        for r in range(R):
            for c in range(C):
                count = 0
                for nr in (r-1, r, r+1):
                    for nc in (c-1, c, c+1):
                        if 0 <= nr < R and 0 <= nc < C:
                            res[r][c] += M[nr][nc]
                            count += 1
                res[r][c] //= count
        return res
```

```
#
# @lc app=leetcode.cn id=665 lang=python3
#
# [665] 非递减数列
#
class Solution:
    def checkPossibility(self, nums: List[int]) -> bool:
        count = 0
        for i in range(len(nums)-1):
            if nums[i]>nums[i+1]:
                count +=1
                #变相去掉nums[i]
                if i <1 or nums[i-1] <= nums[i+1]:
                    nums[i] = nums[i+1]
                else:
                    # 变相去掉nums[i+1]
                    nums[i+1]=nums[i]
        return count <= 1
```

```
#
# @lc app=leetcode.cn id=674 lang=python3
#
# [674] 最长连续递增序列
#
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        if not nums:
            return 0
        count = 1
        res = 0
        for i in range(len(nums)-1):
            if nums[i] < nums[i+1]:
```

```python
                count += 1
            else:
                res = max(res,count)
                count = 1
        return max(res,count)
```

```python
#
# @lc app=leetcode.cn id=680 lang=python3
#
# [680] 验证回文字符串
#
class Solution:
    def validPalindrome(self, s: str) -> bool:
        count = 0
        for i in range(len(s)//2):
            if s[i] != s[-1-i]:
                t, u = s[:i]+s[i+1:], s[:-1-i]+s[len(s)-i:]
                return t == t[::-1] or u == u[::-1]
        return True
```

```python
#
# @lc app=leetcode.cn id=695 lang=python3
#
# [695] 岛屿的最大面积
#
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        res = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    temp = self.dfs(grid, i, j)
                    res = max(res,temp)
        return res

    def dfs(self, grid, i, j):
        # 终止条件
        if i < 0 or j < 0 or i >= len(grid) or j >= len(grid[0]) or grid[i][j] == 0:
            return 0

        # 四个方向搜索
        grid[i][j] = 0
        res = 1
        res += self.dfs(grid, i-1, j)
        res += self.dfs(grid, i, j-1)
        res += self.dfs(grid, i+1, j)
        res += self.dfs(grid, i, j+1)

        return res
```

```
#
# @lc app=leetcode.cn id=754 lang=python3
#
# [754] 到达终点数字
#
class Solution:
    def reachNumber(self, target: int) -> int:
        target = abs(target)
        p, n = 0, 0
        # 和比目标值还小 或者不同奇偶
        while p < target or (p + target) % 2 != 0:
            n += 1
            p += n
        return n
```

```
#
# @lc app=leetcode.cn id=836 lang=python3
#
# [836] 矩形重叠
#
class Solution:
    def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
        return not (rec1[2] <= rec2[0] or  # rec1的右边在rec2的左边
                    rec1[3] <= rec2[1] or  # rec1的上边在rec2的下边
                    rec1[0] >= rec2[2] or  # rec1的左边在rec2的右边
                    rec1[1] >= rec2[3])    # rec1的下边在rec2的上边
```

```
#
# @lc app=leetcode.cn id=974 lang=python3
#
# [974] 和可被 K 整除的子数组
#
class Solution:
    def subarraysDivByK(self, A: List[int], K: int) -> int:
        sums = [0]
        for x in A:
            sums.append((sums[-1]+x)%K)

        dic = {}
        for i in sums:
            dic[i] = dic.get(i,0)+1
        res = 0
        for _,val in dic.items():
            res += val*(val-1)//2
        return res
```

```
#
# @lc app=leetcode.cn id=1015 lang=python3
#
# [1015] 可被 K 整除的最小整数
#
class Solution:
    def smallestRepunitDivByK(self, K: int) -> int:
        if K%2 == 0 or K%5 == 0:
            return -1
        temp = 1
        len = 1
        while temp % K :
            temp = (temp % K) * 10 + 1
            len += 1
        return len
```

```
#
# @lc app=leetcode.cn id=1147 lang=python3
#
# [1147] 段式回文
#
class Solution:
    def longestDecomposition(self, text: str) -> int:
        n = len(text)
        i, j = 0, n - 1
        str1, str2, ans = '', '', 0
        while i < j:
            str1 = str1 + text[i]
            str2 = text[j] + str2
            if str1 == str2:
                ans += 2
                str1, str2 = '', ''
            i += 1
            j -= 1
        if n % 2 == 1 or str1 != '':
            ans += 1
        return ans
```

```
#
# @lc app=leetcode.cn id=1293 lang=python3
#
# [1293] 网格中的最短路径
#
class Solution:
    def shortestPath(self, grid: List[List[int]], k: int) -> int:
        m, n = len(grid), len(grid[0])
        if m == 1 and n == 1:
            return 0

        k = min(k, m + n - 3)
        visited = set((0, 0, k))
        q = [(0, 0, k)]

        step = 0
        while q:
            step += 1
            cnt = len(q)
            for _ in range(cnt):
                x, y, rest = q.pop(0)
                for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < m and 0 <= ny < n:
                        if grid[nx][ny] == 0 and (nx, ny, rest) not in visited:
                            if nx == m - 1 and ny == n - 1:
                                return step
                            q.append((nx, ny, rest))
                            visited.add((nx, ny, rest))
                        elif grid[nx][ny] == 1 and rest > 0 and (nx, ny, rest - 1) not in visited:
                            q.append((nx, ny, rest - 1))
                            visited.add((nx, ny, rest - 1))
        return -1
```