

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

```
1 #
2 # @lc app=leetcode.cn id=1 lang=python3
3 #
4 # [1] 两数之和
5 #
6 class Solution:
7     def twoSum(self, nums: List[int], target: int) -> List[int]:
8         dic = {}
9         for i in range(len(nums)):
10             if target - nums[i] in dic :
11                 return [dic[target-nums[i]], i ]
12             dic[nums[i]] = i
```

```
1 #
2 # @lc app=leetcode.cn id=2 lang=python3
3 #
4 # [2] 两数相加
5 #
6 # Definition for singly-linked list .
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
14         jingwei = 0
15         # 两个空指针 n后面要被覆盖的
16         head = n = ListNode(0)
17         while l1 or l2 or jingwei:
18             v1 = v2 = 0
19             if l1:
20                 v1 = l1.val
21                 l1 = l1.next
22             if l2:
```

```

23         v2 = l2.val
24         l2 = l2.next
25     # 除数、余数
26     val = (v1+v2+jingwei) % 10
27     jingwei = (v1+v2+jingwei) // 10
28     n.next = ListNode(val)
29     # 指向下一个
30     n = n.next
31     return head.next # 记得把第一个0去掉

```

```

1  #
2  # @lc app=leetcode.cn id=3 lang=python3
3  #
4  # [3] 无重复字符的最长子串
5  #
6  class Solution:
7      def lengthOfLongestSubstring(self, s: str) -> int:
8          # 记录表 256个字符
9          dic = {}
10
11         start = maxlen = 0
12         # 遍历 滑动窗 [start,j] j往右边移动 若遇到重复的 start又移一位
13         for j in range(len(s)):
14             # 如果这个字符出现过了, 又移动 最左边那个踢出滑动窗
15             if s[j] in dic and dic[s[j]] >= start:
16                 start = dic[s[j]] + 1
17             # 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)
18             dic[s[j]] = j
19             maxlen = max(maxlen, j-start + 1)
20         return maxlen

```

```

1  #
2  # @lc app=leetcode.cn id=4 lang=python3
3  #
4  # [4] 寻找两个有序数组的中位数
5  #
6  class Solution:
7      def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
8          leng = len(nums1)+len(nums2)
9          if leng%2 == 1:#奇数
10             return self.findk(nums1,nums2,leng//2)
11         else:
12             return (self.findk(nums1,nums2,leng//2-1)+self.findk(nums1,nums2,leng//2))/2.0
13         # 找k大的数
14         def findk(self, nums1, nums2, k):
15             if not nums1:

```

```

16         return nums2[k]
17     if not nums2:
18         return nums1[k]
19     l1 , l2 = len(nums1)//2,len(nums2)//2
20     val1 , val2 = nums1[l1],nums2[l2]
21
22     if l1+l2<k:# 往右找
23         if val1 > val2:
24             return self.findk(nums1, nums2[l2 + 1:], k - l2 - 1)
25         else:
26             return self.findk(nums1[l1 + 1:],nums2, k - l1 - 1)
27     else: # 往左找
28         if val1 > val2:
29             return self.findk(nums1[:l1],nums2, k)
30
31         else:
32             return self.findk(nums1, nums2[:l2], k)

```

```

1  #
2  # @lc app=leetcode.cn id=5 lang=python3
3  #
4  # [5] 最长回文子串
5  #
6  class Solution:
7      def longestPalindrome(self, s: str) -> str:
8          if s is None:
9              return None
10
11         # 动态规划
12         dp = [[0 for __ in range(len(s))] for __ in range(len(s))]
13         left , right , max_len = 0, 0, 0
14
15         for j in range(len(s)):
16             # 对角线置1
17             dp[j][j] = 1
18             for i in range(j-1,-1,-1):
19                 if s[i] == s[j] and (j-i < 2 or dp[i+1][j-1]):
20                     dp[i][j] = 1
21                 if dp[i][j] and max_len < j-i+1:
22                     max_len = j - i + 1
23                     left , right = i, j
24         return s[left : right+1]

```

```

1  #
2  # @lc app=leetcode.cn id=6 lang=python3
3  #

```

```

4 # [6] Z 字形变换
5 #
6 class Solution:
7     def convert(self, s: str, numRows: int) -> str:
8         if numRows == 1 or numRows >= len(s):
9             return s
10        # z前半个(|/)个数两行减2
11        p = 2 * (numRows - 1)
12
13        result = [""] * numRows
14        for i in range(len(s)):
15            floor = i % p # 一个形状轮回的位置
16            if floor >= p//2: # 在/上
17                floor = p - floor
18            result[floor] += s[i]
19        return "".join(result)

```

```

1 #
2 # @lc app=leetcode.cn id=7 lang=python3
3 #
4 # [7] 整数反转
5 #
6 class Solution:
7     def reverse(self, x: int) -> int:
8         sign = 1 if x > 0 else -1
9         res = 0
10        x = abs(x)
11        while x:
12            res = res*10 + x%10
13            if res > 2**31 - 1:
14                return 0
15            x = x//10
16
17        return sign * res

```

```

1 #
2 # @lc app=leetcode.cn id=8 lang=python3
3 #
4 # [8] 字符串转换整数 (atoi)
5 #
6 class Solution:
7     def myAtoi(self, str: str) -> int:
8         # 去空格
9         str = str.strip()
10        if len(str) == 0:
11            return 0

```

```

12     sign = 1
13     if str[0] == '+' or str[0] == '-':
14         if str[0] == '-':
15             sign = -1
16         str = str[1:]
17     res = 0
18     for char in str:
19         if char >= '0' and char <= '9':
20             res = res * 10 + ord(char) - ord('0')
21         if char < '0' or char > '9':
22             break
23     return max(-2**31, min(sign * res, 2**31-1))

```

```

1  #
2  # @lc app=leetcode.cn id=9 lang=python3
3  #
4  # [9] 回文数
5  #
6  class Solution:
7      def isPalindrome(self, x: int) -> bool:
8          if x < 0 :
9              return False
10         # 最高位的位数
11         d = 1
12         while x // d >= 10:
13             d *= 10
14         while x > 0:
15             # p q 对应最高位和最低位
16             p = x // d
17             q = x % 10
18             if p != q :
19                 return False
20             # x 去掉最高位,去掉最低位
21             x = x % d // 10
22             # x 去掉了两位,d也减两位
23             d //= 100
24         return True

```

```

1  #
2  # @lc app=leetcode.cn id=10 lang=python3
3  #
4  # [10] 正则表达式匹配
5  #
6  class Solution:
7      def isMatch(self, s: str, p: str) -> bool:
8          """

```

```

9      # 递归写法
10     # s已被匹配且p已耗完
11     if not s and not p:
12         return True
13     # p已耗完但s未被完全匹配
14     if len(s) > 0 and len(p) == 0:
15         return False
16
17     # 如果模式第二个字符是*
18     if len(p) > 1 and p[1] == '*':
19         if len(s) > 0 and (s[0] == p[0] or p[0] == '.'): # ax a* or ax .*
20             # 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
21             return self.isMatch(s, p[2:]) or self.isMatch(s[1:], p)
22         else:
23             # 如果第一个字符不匹配, p往后移2位, 相当于忽略x*
24             return self.isMatch(s, p[2:])
25     # 如果模式第二个字符不是*
26     if len(s) > 0 and (s[0] == p[0] or p[0] == '.'):
27         return self.isMatch(s[1:], p[1:])
28     else:
29         return False
30     """
31
32     # 动态规划
33     # 初始化dp表, 初始化表的第一列和第一行
34     # p对应列 s对应行
35     dp = [[False for j in range(len(p) + 1)] for i in range(len(s) + 1)]
36     dp[0][0] = True # s 和 p 都为空时
37     # 若 s 为空时
38     # 处理第一行
39     # p 与 dp 有一位的错位(多了一个空导致的)
40     for j in range(1, len(p) + 1):
41         # dp[0][j] = (p[j-1] == "*" and (j >= 2 and dp[0][j-2]))
42         # 等同于下列语句
43         if p[j - 1] == '*':
44             if j >= 2:
45                 dp[0][j] = dp[0][j - 2]
46     # 第一列就第一个是 True, 下面都是 False
47     # 不用处理 pass
48
49     for i in range(1, len(s) + 1):
50         for j in range(1, len(p) + 1):
51             # j-1才为正常字符串中的索引
52             # p当前位置为"*"时
53             # 代表空串--dp[i][j-2]
54             # 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] == '.')
55             if p[j - 1] == '*':

```

```

55         dp[i][j] = dp[i][j - 2] or (
56             dp[i - 1][j] and (p[j - 2] == s[i - 1] or p[j - 2] == '.')
57         )
58         # p当前位置为"."时或者与s相同时，传递dp[i-1][j-1]的真值
59         else:
60             dp[i][j] = (p[j - 1] == '.' or p[j - 1] == s[i - 1]) and dp[i - 1][j - 1]
61     return dp[-1][-1]

```

```

1  #
2  # @lc app=leetcode.cn id=11 lang=python3
3  #
4  # [11] 盛最多水的容器
5  #
6  class Solution:
7      def maxArea(self, height: List[int]) -> int:
8          max_area = 0
9          left, right = 0, len(height) - 1
10         while left < right:
11             # 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
12             temp = min(height[left], height[right]) * (right - left)
13             max_area = max(max_area, temp)
14             # 判断哪条高小，小的那边下标进行操作
15             if height[right] > height[left]:
16                 left += 1
17             else:
18                 right -= 1
19         return max_area

```

```

1  #
2  # @lc app=leetcode.cn id=12 lang=python3
3  #
4  # [12] 整数转罗马数字
5  #
6  class Solution:
7      def intToRoman(self, num: int) -> str:
8          # 贪心算法
9          dic = {
10              'M': 1000,
11              'CM': 900, 'D': 500, 'CD': 400, 'C': 100,
12              'XC': 90, 'L': 50, 'XL': 40, 'X': 10,
13              'IX': 9, 'V': 5, 'IV': 4, 'I': 1,
14          }
15          result = ""
16          for letter, number in dic.items():
17              if num >= number:
18                  result += letter * (num // number)

```

```
19         num %= number
20     return result
```

```
1  #
2  # @lc app=leetcode.cn id=13 lang=python3
3  #
4  # [13] 罗马数字转整数
5  #
6  class Solution:
7      def romanToInt(self, s: str) -> int:
8          dicts = {
9              "I": 1,
10             "V": 5,
11             "X": 10,
12             "L": 50,
13             "C": 100,
14             "D": 500,
15             "M": 1000
16         }
17         s = s.replace("IV", "IIII").replace("IX", "VIIII")
18         s = s.replace("XL", "XXXX").replace("XC", "LXXXX")
19         s = s.replace("CD", "CCCC").replace("CM", "DCCCC")
20         data = 0
21         for item in s:
22             data += dicts[item]
23     return data
```

```
1  #
2  # @lc app=leetcode.cn id=14 lang=python3
3  #
4  # [14] 最长公共前缀
5  #
6  class Solution:
7      def longestCommonPrefix(self, strs: List[str]) -> str:
8          """
9          sz = zip(*strs)
10         ret = ""
11         for char in sz:
12             if len(set(char)) > 1:
13                 break
14             ret +=char[0]
15         return ret
16         """
17         if not strs:
18             return ''
19         strs.sort(key = lambda x : len(x))
```



```

20     for idx in range(len(strs[0])):
21         # 最大的可能长度就是第一个的长度
22         for i in range(1, len(strs)):
23             # 对每个元素都要遍历
24             if strs[i][idx] != strs[0][idx]:
25                 return strs[0][:idx]
26     return strs[0]

```

```

1  #
2  # @lc app=leetcode.cn id=15 lang=python3
3  #
4  # [15] 三数之和
5  #
6  class Solution:
7      def threeSum(self, nums: List[int]) -> List[List[int]]:
8          nums.sort()
9          res = []
10         for i in range(len(nums)-2):
11             if i > 0 and nums[i] == nums[i-1]:
12                 continue
13             l, r = i+1, len(nums) - 1
14             while l < r:
15                 s = nums[i] + nums[l] + nums[r]
16                 if s < 0:
17                     l += 1
18                 elif s > 0:
19                     r -= 1
20                 else:
21                     res.append((nums[i], nums[l], nums[r]))
22                     # 避免一样的加进去
23                     while l < r and nums[l] == nums[l+1]:
24                         l += 1
25                     while l < r and nums[r] == nums[r-1]:
26                         r -= 1
27                     l += 1
28                     r -= 1
29     return res

```

```

1  #
2  # @lc app=leetcode.cn id=16 lang=python3
3  #
4  # [16] 最接近的三数之和
5  #
6  class Solution:
7      def threeSumClosest(self, nums: List[int], target: int) -> int:
8          nums.sort()

```

```

9     res = sum(nums[0:3])
10
11     for i in range(len(nums)-2):
12         l,r = i+1,len(nums)-1
13         while l < r:
14             sum_val = nums[i]+nums[l]+nums[r]
15             if sum_val == target:
16                 return sum_val
17             if abs(res-target)>abs(sum_val-target):
18                 res = sum_val
19             if sum_val < target:
20                 l+=1
21             else:
22                 r -= 1
23     return res

```

```

1  #
2  # @lc app=leetcode.cn id=17 lang=python3
3  #
4  # [17] 电话号码的字母组合
5  #
6  class Solution:
7      def letterCombinations(self, digits: str) -> List[str]:
8          dmap = {
9              '2': 'abc',
10             '3': 'def',
11             '4': 'ghi',
12             '5': 'jkl',
13             '6': 'mno',
14             '7': 'pqrs',
15             '8': 'tuv',
16             '9': 'wxyz'
17         }
18         if len(digits) == 0:
19             return []
20         if len(digits) == 1:
21             return list(dmap[digits])
22         prev = self.letterCombinations(digits[:-1])
23         additional = dmap[digits[-1]]
24         return [s + c for s in prev for c in additional]

```

```

1  #
2  # @lc app=leetcode.cn id=18 lang=python3
3  #
4  # [18] 四数之和
5  #

```

```

6 class Solution:
7     def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
8         res = []
9         # 去除异常
10        if not nums or len(nums) < 4:
11            return res
12        nums.sort()
13        # 第一个数遍历
14        for i in range(len(nums) - 3):
15            if i > 0 and nums[i] == nums[i - 1]:
16                continue
17            # 第二个数遍历
18            for j in range(i + 1, len(nums) - 2):
19                if j > i + 1 and nums[j] == nums[j - 1]:
20                    continue
21            # 双指针
22            L, R = j + 1, len(nums) - 1
23            while L < R:
24                if nums[i] + nums[j] + nums[L] + nums[R] == target:
25                    res.append([nums[i], nums[j], nums[L], nums[R]])
26                    while L < R and nums[L] == nums[L + 1]:
27                        L += 1
28                    while L < R and nums[R] == nums[R - 1]:
29                        R -= 1
30                    L += 1
31                    R -= 1
32                elif nums[i] + nums[j] + nums[L] + nums[R] < target:
33                    L += 1
34                else:
35                    R -= 1
36            return res

```

```

1 #
2 # @lc app=leetcode.cn id=19 lang=python3
3 #
4 # [19] 删除链表的倒数第N个节点
5 #
6 # Definition for singly-linked list .
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
14         if head is None:

```

```

15         return None
16     dummy = ListNode(-1)
17     dummy.next = head
18     slow = fast = dummy
19     # 先走n步
20     for i in range(n):
21         fast = fast.next
22
23     # slow 少走n步
24     while fast.next :
25         fast = fast.next
26         slow = slow.next
27     # 删除
28     slow.next = slow.next.next
29     return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=20 lang=python3
3  #
4  # [20] 有效的括号
5  #
6  class Solution:
7      def isValid(self, s: str) -> bool:
8          # 判断是否是奇数或空字符
9          if s=="":
10             return True
11         stack = []
12         match = {')': '(', ']' : '[' , '}' : '{'}
13         for ch in s:
14             if ch in match:
15                 if not (stack and stack.pop() == match[ch]):
16                     return False
17             else :
18                 stack.append(ch)
19         return not stack
20
21         """
22         if len(s) %2 != 0:
23             return False
24         count = 0
25         leng = len(s)
26         # 将其中的(){}[] 都换掉，然后判断是否有剩余
27         while(count < leng/2 ):
28             s = s.replace("{}","").replace ("[]","") .replace ("()", "")
29             count+=1
30

```

```

31         if len(s)>0:
32             return False
33         else:
34             return True
35     """

```

```

1  #
2  # @lc app=leetcode.cn id=21 lang=python3
3  #
4  # [21] 合并两个有序链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
14         dummy = now = ListNode(-1)
15         while l1 and l2:
16             if l1.val <= l2.val:
17                 now.next = l1
18                 l1 = l1.next
19             else:
20                 now.next = l2
21                 l2 = l2.next
22             now = now.next
23         now.next = l1 or l2
24         return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=22 lang=python3
3  #
4  # [22] 括号生成
5  #
6  class Solution:
7     def generateParenthesis(self, n: int) -> List[str]:
8         res = []
9         if n > 0:
10             self.dfs(n, '', res, 0, 0)
11         return res
12
13     def dfs(self, n, path, res, left, right):
14         # 终止条件
15         if len(path) == 2 * n:

```

```

16         res.append(path)
17         return
18     # 左括号(够了没
19     if left < n:
20         self.dfs(n,path+'(',res, left +1, right)
21     # 右括号补成和左括号一样多
22     if left > right:
23         self.dfs(n,path+')',res, left , right+1)

```

```

1  #
2  # @lc app=leetcode.cn id=23 lang=python3
3  #
4  # [23] 合并K个排序链表
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def mergeKLists(self, lists : List[ListNode]) -> ListNode:
14         if not lists :
15             return None
16         return self.mergeK(lists, 0, len( lists ) - 1)
17
18     def mergeK(self, lists , low, high):
19         if low == high:
20             return lists [low]
21         elif low + 1 == high:
22             return self.mergeTwolists(lists [low], lists [high])
23         mid = (low + high) // 2
24         return self.mergeTwolists(self.mergeK(lists, low, mid), self.mergeK(lists, mid + 1, high))
25
26     def mergeTwolists(self, l1 , l2):
27         if l1 is None:
28             return l2
29         if l2 is None:
30             return l1
31         head = curr = ListNode(-1)
32         while l1 and l2:
33             if l1.val <= l2.val:
34                 curr.next = l1
35                 l1 = l1.next
36             else :
37                 curr.next = l2

```

```

38         l2 = l2.next
39         curr = curr.next
40         curr.next = l1 or l2
41         return head.next

```

```

1  #
2  # @lc app=leetcode.cn id=24 lang=python3
3  #
4  # [24] 两两交换链表中的节点
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def swapPairs(self, head: ListNode) -> ListNode:
14         prev = dummy = ListNode(-1)
15         dummy.next = head
16         while prev.next and prev.next.next :
17             # prev a b -> prev b a (交换a,b)
18             a = prev.next
19             b = prev.next.next
20             prev.next, b.next, a.next = b, a, b.next
21             prev = a
22         return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=25 lang=python3
3  #
4  # [25] K 个一组翻转链表
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
14         if head is None or k < 2:
15             return head
16         dummy = ListNode(0)
17         dummy.next = head
18         start = dummy

```

```

19     end = start.next
20
21     count = 0
22     while end:
23         count += 1
24         if count % k == 0:
25             # 返回为新一轮的头
26             start = self.reverse(start, end.next)
27             end = start.next
28         else:
29             end = end.next
30     return dummy.next
31
32     def reverse(self, start, end):
33         # 输入一个是前驱,一个后驱
34         prev, curr = start, start.next
35         first = curr
36         while curr != end:
37             temp = curr.next
38             curr.next = prev
39             prev = curr
40             curr = temp
41         start.next = prev
42         first.next = end
43         return first

```

```

1 #
2 # @lc app=leetcode.cn id=26 lang=python3
3 #
4 # [26] 删除排序数组中的重复项
5 #
6 class Solution:
7     def removeDuplicates(self, nums: List[int]) -> int:
8         idx = 0
9         while idx < len(nums) - 1:
10             if nums[idx] == nums[idx+1]:
11                 nums.pop(idx)
12                 idx -= 1
13             idx += 1
14         return len(nums)

```

```

1 #
2 # @lc app=leetcode.cn id=27 lang=python3
3 #
4 # [27] 移除元素
5 #

```



```

6 class Solution:
7     def removeElement(self, nums: List[int], val: int) -> int:
8         left = 0
9         right = len(nums) - 1
10        while left <= right :
11            if nums[left] == val:
12                nums[left] , nums[right] = nums[right] ,nums[left]
13                right -= 1
14            else :
15                left += 1
16        return left

```

```

1 #
2 # @lc app=leetcode.cn id=28 lang=python3
3 #
4 # [28] 实现 strStr()
5 #
6 class Solution:
7     def strStr(self, haystack: str, needle: str) -> int:
8         if not needle or haystack == needle:
9             return 0
10        elif len(haystack)<= len(needle):
11            return -1
12
13        leng = len(needle)
14        for i in range(len(haystack)-leng+1):
15            if needle == haystack[i:i+leng]:
16                return i
17        return -1

```

```

1 #
2 # @lc app=leetcode.cn id=29 lang=python3
3 #
4 # [29] 两数相除
5 #
6 class Solution:
7     def divide(self, dividend: int, divisor: int) -> int:
8         if (dividend < 0 and divisor < 0) or (dividend > 0 and divisor > 0):
9             positive = 1
10        else :
11            positive = -1
12
13        dividend, divisor = abs(dividend), abs(divisor)
14        res = 0
15        while dividend >= divisor:
16            temp, i = divisor, 1

```

```

17         while dividend >= temp:
18             dividend -= temp
19             res += i
20             # 除数乘以2 商一下子也多2
21             i <<= 1
22             temp <<= 1
23
24         # 防止溢出
25         return min(max(positive * res, -2**31), 2**31-1)

```

```

1  #
2  # @lc app=leetcode.cn id=31 lang=python3
3  #
4  # [31] 下一个排列
5  #
6  class Solution:
7      def nextPermutation(self, nums: List[int]) -> None:
8          # i为数组倒数第二个值, j为倒数第一个值
9          i = len(nums) - 2
10         j = len(nums) - 1
11         # 从右到左找到第一次断崖
12         # 第一次非逆序的地方
13         while i >= 0 and nums[i] >= nums[i+1]:
14             i -= 1
15
16         # 从右到左找到比崖底水平面高的第一个元素
17         if i >= 0:
18             while j >= 0 and nums[i] >= nums[j]:
19                 j -= 1
20             nums[i], nums[j] = nums[j], nums[i]
21
22         self.reverse(nums, i+1)
23
24         # 用于原地反转nums中从start之后的所有元素
25         def reverse(self, nums, start):
26             i, j = start, len(nums) - 1
27             while i < j:
28                 nums[i], nums[j] = nums[j], nums[i]
29                 i += 1
30                 j -= 1
31         return

```

```

1  #
2  # @lc app=leetcode.cn id=32 lang=python3
3  #
4  # [32] 最长有效括号

```

```

5  #
6  class Solution:
7      def longestValidParentheses(self, s: str) -> int:
8          """
9          # 栈法
10         res = []
11         stack = []
12         for i in range(len(s)):
13             if (stack and s[i]==")"):
14                 res.append(stack.pop())
15                 res.append(i)
16             if (s[i]=="("):
17                 stack.append(i)
18
19         res.sort()
20         max_len = 0
21         i=0
22         while i < len(res)-1:
23             tmp = i
24             # 最长连续值
25             while( i < len(res)-1 and res[i+1]-res[i] == 1):
26                 i += 1
27             max_len = max(max_len,i-tmp+1)
28             i += 1
29         return max_len
30         """
31
32         # 动态规划
33         if not s:
34             return 0
35         dp = [0] * len(s)
36         for i in range(1,len(s)):
37             if s[i]==")":
38                 # ()对
39                 if s[i-1]=="(":
40                     dp[i] = dp[i-2] + 2
41                 # 连着两个))
42                 if s[i-1]==")" and i-1-dp[i-1]>=0 and s[i-1-dp[i-1]]=="(":
43                     dp[i] = dp[i-dp[i-1]-2] + dp[i-1] + 2
44         return max(dp)

```

```

1  #
2  # @lc app=leetcode.cn id=33 lang=python3
3  #
4  # [33] 搜索旋转排序数组
5  #

```

```

6 class Solution:
7     def search(self, nums: List[int], target: int) -> int:
8         if not nums:
9             return -1
10        l, r = 0, len(nums) - 1
11
12        while l <= r:
13            mid = (l+r)//2
14            if nums[mid] == target:
15                return mid
16            # mid在前半段 或者l mid r 都在右边
17            if nums[l] <= nums[mid]:
18                if nums[l] <= target <= nums[mid]:
19                    r = mid - 1
20                else:
21                    l = mid + 1
22            # l 在左半段 、 mid 在后半段
23            else:
24                if nums[mid] <= target <= nums[r]:
25                    l = mid + 1
26                else:
27                    r = mid - 1
28        return -1

```

```

1 #
2 # @lc app=leetcode.cn id=34 lang=python3
3 #
4 # [34] 在排序数组中查找元素的第一个和最后一个位置
5 #
6 class Solution:
7     def searchRange(self, nums: List[int], target: int) -> List[int]:
8         if len(nums) == 0:
9             return [-1, -1]
10        min = 0
11        max = len(nums) - 1
12        while min <= max:
13            pos = (min + max) // 2
14            if nums[pos] > target:
15                max = pos - 1
16            elif nums[pos] < target:
17                min = pos + 1
18            else:
19                # when nums[pos] == target
20                # find the min and max
21                for i in range(pos, max + 1):
22                    if nums[i] == target:

```

```

23         max = i
24     for i in range(pos, min - 1, -1):
25         if nums[i] == target:
26             min = i
27     return [min, max]
28 return [-1, -1]

```

```

1  #
2  # @lc app=leetcode.cn id=35 lang=python3
3  #
4  # [35] 搜索插入位置
5  #
6  class Solution:
7      def searchInsert(self, nums: List[int], target: int) -> int:
8          left = 0
9          right = len(nums) - 1
10         while left <= right:
11             mid = (left + right) // 2
12             if nums[mid] == target:
13                 return mid
14             elif target < nums[mid]:
15                 right = mid - 1
16             else:
17                 left = mid + 1
18         return left

```

```

1  #
2  # @lc app=leetcode.cn id=36 lang=python3
3  #
4  # [36] 有效的数独
5  #
6  class Solution:
7      def isValidSudoku(self, board: List[List[str]]) -> bool:
8          return (self.is_row_valid(board) and
9                  self.is_col_valid(board) and
10                 self.is_square_valid(board))
11
12     def is_row_valid(self, board):
13         for row in board:
14             if not self.is_unit_valid(row):
15                 return False
16         return True
17
18     def is_col_valid(self, board):
19         # 列转化成行
20         for col in zip(*board):

```

```

21         if not self.is_unit_valid(col):
22             return False
23     return True
24
25     def is_square_valid(self, board):
26         for i in (0, 3, 6):
27             for j in (0, 3, 6):
28                 square = [board[x][y] for x in range(i, i + 3) for y in range(j, j + 3)]
29                 if not self.is_unit_valid(square):
30                     return False
31         return True
32
33     def is_unit_valid(self, unit):
34         unit = [i for i in unit if i != '.']
35         return len(set(unit)) == len(unit)

```

```

1  #
2  # @lc app=leetcode.cn id=37 lang=python3
3  #
4  # [37] 解数独
5  #
6  class Solution:
7      def solveSudoku(self, board: List[List[str]]) -> None:
8          self.dfs(board)
9
10     def dfs(self, board):
11         for i in range(9):
12             for j in range(9):
13                 if board[i][j] == '.':
14                     for k in '123456789':
15                         board[i][j] = k
16                         # 修改一个值判断是不是合法的
17                         # 如果这个递归可以返回true并且当前填入的数字也没毛病
18                         # 则证明我们解完了数独
19                         if self.isOK(board, i, j) and self.dfs(board):
20                             return True
21                         board[i][j] = '.'
22             return False
23     # 全部填完之后返回True
24     return True
25
26     def isOK(self, board, x, y):
27         # 列符合
28         for i in range(9):
29             if i != x and board[i][y] == board[x][y]:
30                 return False

```

```

31     #检查行是否符合
32     for j in range(9):
33         if j != y and board[x][j] == board[x][y]:
34             return False
35     row_start = 3*(x // 3)
36     col_start = 3*(y // 3)
37     for i in range(row_start,row_start+3):
38         for j in range(col_start,col_start+3):
39             if (i!= x or j!= y) and board[i][j] == board[x][y]:
40                 return False
41     return True

```

```

1  #
2  # @lc app=leetcode.cn id=38 lang=python3
3  #
4  # [38] 外观数列
5  #
6  class Solution:
7      def countAndSay(self, n: int) -> str:
8          s = '1'
9          for _ in range(n-1):
10             s = self.count(s)
11         return s
12
13     def count(self, s):
14         m = list(s)
15         # 加一个后面不会溢出(随便加一个就行)
16         m.append(5)
17         res = ()
18         i,j = 0,0
19         while i < len(m)-1:
20             j+= 1
21             if m[j] != m[i]:
22                 res += (str(j-i) , m[i] )
23                 i = j
24         # 用空元素链接res
25         return ''.join(res)

```

```

1  #
2  # @lc app=leetcode.cn id=39 lang=python3
3  #
4  # [39] 组合总和
5  #
6  class Solution:
7      def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
8          candidates.sort()

```

```

9     res = []
10    self.dfs(candidates, target, 0, [], res)
11    return res
12
13    def dfs(self, nums, target, index, path, res):
14        if target < 0:
15            return
16        if target == 0:
17            res.append(path)
18            return
19        for i in range(index, len(nums)):
20            self.dfs(nums, target-nums[i], i, path+[nums[i]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=40 lang=python3
3  #
4  # [40] 组合总和 II
5  #
6  class Solution:
7      def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
8          candidates.sort()
9          res = []
10         self.combine_sum_2(candidates,target, 0, [], res)
11         return res
12
13         def combine_sum_2(self, nums,target, start, path, res):
14             # 超过了
15             if target < 0:
16                 return
17             if target == 0 :
18                 res.append(path)
19                 return
20
21             for i in range(start, len(nums)):
22                 # 解集不重复
23                 if i > start and nums[i] == nums[i - 1]:
24                     continue
25                 self.combine_sum_2(nums,target - nums[i],
26                                     i + 1, path + [nums[i]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=41 lang=python3
3  #
4  # [41] 缺失的第一个正数
5  #
6  class Solution:

```



```

7  def firstMissingPositive ( self , nums: List[int] ) -> int:
8      self.bucket_sort(nums)
9
10     for i in range(len(nums)):
11         if nums[i] != (i+1):
12             return i+1
13     return len(nums)+1
14
15     def bucket_sort(self,nums):
16         for i in range(len(nums)):
17             while 0 <= nums[i] < len(nums) and nums[i] != nums[nums[i]-1]:
18                 temp = nums[i]-1
19                 nums[i] = nums[temp]
20                 nums[temp] = temp +1
21     return

```

```

1  #
2  # @lc app=leetcode.cn id=42 lang=python3
3  #
4  # [42] 接雨水
5  #
6  class Solution:
7      def trap(self , height: List[int] ) -> int:
8          if not height: # 边界检查
9              return 0
10         l ,r = 0 , len(height) - 1
11
12         res = 0
13         l_max , r_max = 0 , 0
14         while l < r :
15             if height[l] < height[r]:
16                 if height[l] >= l_max:
17                     l_max = height[l]
18                 else :
19                     res += l_max - height[l]
20                 l += 1
21             else :
22                 if height[r] >= r_max:
23                     r_max = height[r]
24                 else :
25                     res += r_max - height[r]
26
27             r -= 1
28         return res

```

```

1  #

```

```

2  # @lc app=leetcode.cn id=43 lang=python3
3  #
4  # [43] 字符串相乘
5  #
6  class Solution:
7      def multiply(self, num1: str, num2: str) -> str:
8
9          #把num1,num2翻转方便计算
10         num1 = num1[::-1]; num2 = num2[::-1]
11         #每一位互相乘的结果用一维数组去储存
12         arr = [0 for i in range(len(num1)+len(num2))]
13         #填充这个一维数组
14         for i in range(len(num1)):
15             for j in range(len(num2)):
16                 arr[i+j] += int(num1[i]) * int(num2[j])
17
18         res = []
19         # arr是反的
20         #计算每一位的终极结果
21         for i in range(len(arr)):
22             #digit表示这一位的数字
23             digit = arr[i] % 10
24             #carry表示加给下一位的量
25             carry = arr[i] // 10
26             if i < len(arr)-1:
27                 #下一位加上
28                 arr[i+1] += carry
29             #更新答案
30             res.insert(0, str(digit))
31         #去除首位为0的情况
32         while res[0] == '0' and len(res) > 1:
33             res.pop(0)
34         #连接成字符串
35         return ''.join(res)

```

```

1  #
2  # @lc app=leetcode.cn id=45 lang=python3
3  #
4  # [45] 跳跃游戏 II
5  #
6  class Solution:
7      def jump(self, nums: List[int]) -> int:
8          if len(nums) <= 1:
9              return 0
10         # (start -> end )
11         end = nums[0]

```

```

12     start = 0
13     step = 1
14     maxDis = nums[0]
15     while end < len(nums) - 1:
16         # 看一步最远能走到哪
17         for i in range(start + 1, end + 1):
18             maxDis = max(maxDis, nums[i] + i)
19         start = end
20         end = maxDis
21         step += 1
22     return step

```

```

1  #
2  # @lc app=leetcode.cn id=46 lang=python3
3  #
4  # [46] 全排列
5  #
6  class Solution:
7      def permute(self, nums: List[int]) -> List[List[int]]:
8          #nums.sort()
9          res = []
10         self.dfs(nums, [], res)
11         return res
12
13     def dfs(self, nums, path, res):
14         if not nums:
15             # nums已经全部压入到path里面了
16             res.append(path)
17             return
18         for i in range(len(nums)):
19             self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=47 lang=python3
3  #
4  # [47] 全排列 II
5  #
6  class Solution:
7      def permuteUnique(self, nums: List[int]) -> List[List[int]]:
8          res = []
9          self.dfs(nums, [], res)
10         return res
11
12     def dfs(self, nums, path, res):
13         if not nums and path not in res:
14             # nums已经全部压入到path里面了

```

```

15         res.append(path)
16         return
17     for i in range(len(nums)):
18         self.dfs(nums[i]+nums[i+1:], path+[nums[i]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=48 lang=python3
3  #
4  # [48] 旋转图像
5  #
6  class Solution:
7      def rotate(self, matrix: List[List[int]]) -> None:
8          if matrix is None or len(matrix) == 1:
9              return
10         ls = len(matrix)
11
12         for i in range(ls // 2):
13             # 那一圈的半行
14             begin, end = i, ls - 1 - i # 左右都往内部i个单位
15             for k in range(ls - 1 - 2 * i): # 减两个i的单位
16                 # 顺着转
17                 temp = matrix[end - k][begin] # 左下角
18                 matrix[end - k][begin] = matrix[end][end - k] # 右下角给左下角
19                 matrix[end][end - k] = matrix[begin + k][end] # 右上角给右下角
20                 matrix[begin + k][end] = matrix[begin][begin + k] # 左上角给右上角
21                 matrix[begin][begin + k] = temp # 左下角给左上角
22         return

```

```

1  #
2  # @lc app=leetcode.cn id=49 lang=python3
3  #
4  # [49] 字母异位词分组
5  #
6  class Solution:
7      def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
8          dic = {}
9          # key是单词对应的元素
10         # value是字符串
11         for word in strs:
12             key = ''.join(sorted(word))
13             if key not in dic:
14                 dic[key] = []
15             dic[key].append(word)
16         res = []
17         for i in dic:
18             res.append(dic[i])

```

```
19     return res
```

```
1  #
2  # @lc app=leetcode.cn id=50 lang=python3
3  #
4  # [50] Pow(x, n)
5  #
6  class Solution:
7      def myPow(self, x: float , n: int) -> float:
8          if n == 0:
9              return 1
10         if n < 0:
11             return 1 / self.myPow(x, -n)
12         if n % 2:
13             return x * self.myPow(x, n-1)
14         return self.myPow(x*x, n // 2)
```

```
1  #
2  # @lc app=leetcode.cn id=51 lang=python3
3  #
4  # [51] N皇后
5  #
6  class Solution:
7      def solveNQueens(self, n: int) -> List[List[str ]]:
8          result = []
9          # C[i]表示第i行皇后在哪一列
10         C = [-1 for _ in range(n)]
11         self.dfs(C,result,0)
12         return result
13
14     def dfs( self ,C,res,row):
15         N = len(C)
16         # 终止条件
17         if N == row :
18             path = [[ "." for _ in range(N)] for _ in range(N)]
19             for i in range(N):
20                 # (i,C[i]) 位置对应皇后
21                 path[i][ C[i]] = "Q"
22             path = ["".join(r) for r in path]
23             # if path not in res:
24             # 不用排除
25             res.append(path)
26             return
27         # 对该行每一列都进行尝试,可以的话下一行
28         for j in range(N):
29             if j not in C and self.isOK(C,row,j):
```

```

30         C[row] = j
31         self.dfs(C,res,row+1)
32         C[row] = -1
33
34     # 对该行之前的都进行判断,返回合理与否
35     def isOK(self,C,row,col):
36         for i in range(row):
37             # 同一列
38             # 同一对角线
39             if C[i] == col or abs(i-row) == abs(C[i]-col):
40                 return False
41         return True

```

```

1  #
2  # @lc app=leetcode.cn id=52 lang=python3
3  #
4  # [52] N皇后 II
5  #
6  class Solution:
7      def totalNQueens(self, n: int) -> int:
8          self.res = 0
9          # C[i]表示第i行皇后在哪一列
10         C = [-1 for _ in range(n)]
11         self.dfs(C,0)
12         return self.res
13
14     def dfs( self ,C,row):
15         N = len(C)
16         # 终止条件
17         if N == row :
18             # 不用排除
19             self.res += 1
20             # 对该行每一列都进行尝试,可以的话下一行
21             for j in range(N):
22                 if j not in C and self.isOK(C,row,j):
23                     C[row] = j
24                     self.dfs(C,row+1)
25                     C[row] = -1
26
27     # 对该行之前的都进行判断,返回合理与否
28     def isOK(self,C,row,col):
29         for i in range(row):
30             # 同一列
31             # 同一对角线
32             if C[i] == col or abs(i-row) == abs(C[i]-col):
33                 return False

```

```
34     return True
```

```
1  #
2  # @lc app=leetcode.cn id=53 lang=python3
3  #
4  # [53] 最大子序和
5  #
6  class Solution:
7      def maxSubArray(self, nums: List[int]) -> int:
8          temp = maxsum = nums[0]
9          for num in nums[1:]:
10             # num 要么单独一个子列,要么归入别的子列
11             temp = max(num,temp+num)
12             maxsum = max(temp,maxsum)
13     return maxsum
```

```
1  #
2  # @lc app=leetcode.cn id=54 lang=python3
3  #
4  # [54] 螺旋矩阵
5  #
6  class Solution:
7      def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
8          if not matrix:
9              return []
10
11          """
12          # 常规方法太烦了
13          res = []
14          xbegin = ybegin = 0
15          xend = len(matrix[0]) - 1
16          yend = len(matrix) - 1
17          while True :
18              # 横
19              for j in range(xbegin,xend+1):
20                  res.append(matrix[ybegin][j])
21              ybegin += 1
22              if ybegin > yend :
23                  break
24              # 竖
25              for j in range(ybegin,yend+1):
26                  res.append(matrix[j][xend])
27              xend -= 1
28              if xbegin > xend:
29                  break
30              # 横
```

```

31         for j in range(xend,xbegin-1,-1):
32             res.append(matrix[yend][j])
33         yend -= 1
34         if ybegin > yend :
35             break
36         # 竖
37         for j in range(yend,ybegin-1,-1):
38             res.append(matrix[j][xbegin])
39         xbegin += 1
40         if xbegin > xend:
41             break
42     return res
43     """
44
45     m,n = len(matrix),len(matrix[0])
46     x = y = di = 0
47     dx = [0,1,0,-1]
48     dy = [1,0,-1,0]
49     res = []
50     visited = set()
51
52     for i in range(m*n):
53         res.append(matrix[x][y])
54         visited.add((x,y))
55         nx,ny = x+dx[di],y+dy[di]
56         if 0<=nx<m and 0<=ny<n and (nx,ny) not in visited:
57             x,y = nx,ny
58         else :
59             di = (di+1)%4 # 如果不满足条件，换一个方向进行遍历
60             x,y = x+dx[di],y+dy[di]
61     return res

```

```

1  #
2  # @lc app=leetcode.cn id=55 lang=python3
3  #
4  # [55] 跳跃游戏
5  #
6  class Solution:
7      def canJump(self, nums: List[int]) -> bool:
8          start = end = 0
9          while start <= end < len(nums) - 1:
10             end = max(end, nums[start] + start)
11             start += 1
12         return end >= len(nums) - 1

```

```

1  #

```



```

2  # @lc app=leetcode.cn id=56 lang=python3
3  #
4  # [56] 合并区间
5  #
6  class Solution:
7      def merge(self, intervals : List[List[int]]) -> List[List[int]]:
8          if len(intervals) <= 1:
9              return intervals
10         res = []
11         intervals.sort(key = lambda x: x[0])
12         s , e = intervals[0][0] , intervals[0][1]
13
14         for i in range(1,len(intervals)):
15             # 后边跟着的区间和[s,e]的交叉,相当于合并
16             if e >= intervals[i][0] :
17                 e = max(e , intervals[i][1] )
18             # 紧跟着的区间在[s,e]后面
19             else :
20                 res.append([s,e])
21                 s ,e = intervals[i][0] , intervals[i][1]
22         res.append([s,e])
23         return res

```

```

1  #
2  # @lc app=leetcode.cn id=57 lang=python3
3  #
4  # [57] 插入区间
5  #
6  class Solution:
7      def insert(self, intervals : List[List[int]], newInterval: List[int]) -> List[List[int]]:
8          s, e = newInterval[0], newInterval[1]
9          left , right = [], []
10         for inter in intervals :
11             # 左边部分
12             if s > inter[1]:
13                 left.append(inter)
14             # 右边部分
15             elif e < inter[0]:
16                 right.append(inter)
17             # 和区间交叉部分,合并
18             else :
19                 s = min(s, inter[0])
20                 e = max(e, inter[1])
21         return left + [[s, e]] + right

```

```

1  #

```

```

2 # @lc app=leetcode.cn id=58 lang=python3
3 #
4 # [58] 最后一个单词的长度
5 #
6 class Solution:
7     def lengthOfLastWord(self, s: str) -> int:
8         if not s :
9             return 0
10        tmp = s.split(' ')
11        tmp = [t for t in tmp if len(t) > 0]
12        if len(tmp) == 0:
13            return 0
14        else:
15            return len(tmp[-1])

```

```

1 #
2 # @lc app=leetcode.cn id=59 lang=python3
3 #
4 # [59] 螺旋矩阵 II
5 #
6 class Solution:
7     def generateMatrix(self, n: int) -> List[List[int]]:
8         mat = [[0 for _ in range(n)] for _ in range(n)]
9
10        b,e = 0 , n - 1
11        val = 1
12        while b < e :
13            # 横
14            for i in range(b,e):
15                mat[b][i] = val
16                val += 1
17            # 竖
18            for i in range(b,e):
19                mat[i][e] = val
20                val += 1
21            # 横
22            for i in range(e,b,-1):
23                mat[e][i] = val
24                val += 1
25            # 竖
26            for i in range(e,b,-1):
27                mat[i][b] = val
28                val += 1
29            b += 1
30            e -= 1
31

```

```

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

```

```

1  #
2  # @lc app=leetcode.cn id=60 lang=python3
3  #
4  # [60] 第k个排列
5  #
6  class Solution:
7      def getPermutation(self, n: int, k: int) -> str:
8          # 待选择的字符串
9          nums = [str(i) for i in range(1,n+1)]
10         # 0!, 1!, ..., (n - 1)!
11         factorials = [1]
12         for i in range(1, n):
13             factorials.append(factorials[i - 1] * i)
14
15         # 第几个转化为第几个的索引(减1)
16         k -= 1
17
18         res = []
19         for i in range(n - 1, -1, -1):
20             # 计算第几个区间,首位所在的区间 k/(n-1)!
21             # 第一个区间首位是1,第二个区间首位是2
22             idx = k // factorials[i]
23             # 减去多个区间对应的值
24             k -= idx * factorials[i]
25             # 结果值添加对应的数字
26             res.append(nums[idx])
27             # 因为排列不重复,nums需要去掉对应元素
28             nums.pop(idx)
29
30         return ''.join(res)

```

```

1  #
2  # @lc app=leetcode.cn id=61 lang=python3
3  #
4  # [61] 旋转链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10         self.next = None

```

```

11
12 class Solution:
13     def rotateRight(self, head: ListNode, k: int) -> ListNode:
14         if head is None or k == 0:
15             return head
16
17         pointer = head
18         length = 1
19         while pointer.next:
20             pointer = pointer.next
21             length += 1
22
23         # 左部分多少个
24         k = length - k%length
25
26         # 连成一个环
27         pointer.next = head
28
29         for i in range(k):
30             pointer = pointer.next
31
32         # 断开
33         head = pointer.next
34         pointer.next = None
35         return head

```

```

1 #
2 # @lc app=leetcode.cn id=62 lang=python3
3 #
4 # [62] 不同路径
5 #
6 class Solution:
7     def uniquePaths(self, m: int, n: int) -> int:
8         mat = [[0 for _ in range(n)] for _ in range(m)]
9         for r in range(m):
10             mat[r][0] = 1
11         for c in range(n):
12             mat[0][c] = 1
13         for r in range(1,m):
14             for c in range(1,n):
15                 mat[r][c] = mat[r-1][c] + mat[r][c-1]
16         return mat[-1][-1]

```

```

1 #
2 # @lc app=leetcode.cn id=63 lang=python3
3 #

```

```

4 # [63] 不同路径 II
5 #
6 class Solution:
7     def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
8         if not obstacleGrid:
9             return
10        r, c = len(obstacleGrid), len(obstacleGrid[0])
11        mat = [[0 for _ in range(c)] for _ in range(r)]
12        # 到起点看这里有没有问题
13        mat[0][0] = 1 - obstacleGrid[0][0]
14
15        for i in range(1, r):
16            mat[i][0] = mat[i-1][0] * (1 - obstacleGrid[i][0])
17        for i in range(1, c):
18            mat[0][i] = mat[0][i-1] * (1 - obstacleGrid[0][i])
19
20        for i in range(1, r):
21            for j in range(1, c):
22                mat[i][j] = (mat[i][j-1] + mat[i-1][j]) * (1 - obstacleGrid[i][j])
23        return mat[-1][-1]

```

```

1 #
2 # @lc app=leetcode.cn id=64 lang=python3
3 #
4 # [64] 最小路径和
5 #
6 class Solution:
7     def minPathSum(self, grid: List[List[int]]) -> int:
8         m,n = len(grid),len(grid[0])
9         dp = [[0 for _ in range(n)] for _ in range(m)]
10        dp[0][0] = grid[0][0]
11        for r in range(1,m):
12            dp[r][0] = dp[r-1][0] + grid[r][0]
13        for c in range(1,n):
14            dp[0][c] = dp[0][c-1] + grid[0][c]
15        for r in range(1,m):
16            for c in range(1,n):
17                dp[r][c] = min(dp[r-1][c], dp[r][c-1]) + grid[r][c]
18        return dp[m-1][n-1]

```

```

1 #
2 # @lc app=leetcode.cn id=66 lang=python3
3 #
4 # [66] 加一
5 #
6 class Solution:

```

```

7  def plusOne(self, digits: List[int]) -> List[int]:
8      """
9      # 数值操作
10     num = 0
11     for i in range(len(digits)):
12         num = num * 10 + digits[i]
13     num = num + 1
14     res = []
15     while num > 0:
16         res.append(num%10)
17         num //= 10
18     res.reverse()
19     return res
20     """
21
22     # 列表操作
23     digits[-1] += 1
24     digits.insert(0, 0)
25     for i in range(len(digits)-1,0,-1):
26         carry = digits[i] // 10
27         digits[i] %= 10
28         digits[i-1] += carry
29
30     if digits[0] == 0:
31         digits.pop(0)
32
33     return digits

```

```

1  #
2  # @lc app=leetcode.cn id=67 lang=python3
3  #
4  # [67] 二进制求和
5  #
6  class Solution:
7      def addBinary(self, a: str, b: str) -> str:
8          if not a:
9              return b
10         if not b:
11             return a
12         # 最后都是1 前面的相加 再加1 补0
13         if a[-1] == '1' and b[-1] == '1':
14             return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'
15         # 最后都是0 补0
16         if a[-1] == '0' and b[-1] == '0':
17             return self.addBinary(a[0:-1],b[0:-1])+'0'
18         # 最后一个1 一个0 补1

```

```

19         else :
20             return self.addBinary(a[0:-1],b[0:-1])+'1'

```

```

1  #
2  # @lc app=leetcode.cn id=69 lang=python3
3  #
4  # [69] x 的平方根
5  #
6  class Solution:
7      def mySqrt(self, x: int) -> int:
8          l ,r = 0 , x
9          while l <= r:
10             mid = (l+r)//2
11             if mid**2 <= x < (mid+1)**2:
12                 return mid
13             elif x < mid**2:
14                 r = mid
15             else :
16                 l = mid+1

```

```

1  #
2  # @lc app=leetcode.cn id=70 lang=python3
3  #
4  # [70] 爬楼梯
5  #
6  class Solution:
7      def climbStairs( self , n: int) -> int:
8          if n == 1:
9              return 1
10             # 初始的两个 输入1 or 2
11             a, b = 1, 2
12             # 从n大于3开始
13             for i in range(2, n):
14                 b ,a = a+b , b
15             return b

```

```

1  #
2  # @lc app=leetcode.cn id=71 lang=python3
3  #
4  # [71] 简化路径
5  #
6  class Solution:
7      def simplifyPath(self, path: str) -> str:
8          res = []
9          for child in path.split('/'):
10             if child in ('', '.'):
11                 pass

```

```

12         elif child == '..':
13             if res: res.pop()
14         else:
15             res.append(child)
16     return '/' + '/'.join(res)

```

```

1  #
2  # @lc app=leetcode.cn id=72 lang=python3
3  #
4  # [72] 编辑距离
5  #
6  class Solution:
7      def minDistance(self, word1: str, word2: str) -> int:
8          l1, l2 = len(word1)+1, len(word2)+1
9          dp = [[0 for _ in range(l2)] for _ in range(l1)]
10         # 行列处理 对应从空到一个字符串 或 一个字符串到空
11         for i in range(l1):
12             dp[i][0] = i
13         for j in range(l2):
14             dp[0][j] = j
15         for i in range(1, l1):
16             for j in range(1, l2):
17                 if word1[i-1]==word2[j-1]:
18                     dp[i][j] = dp[i-1][j-1]
19                 else:
20                     # 三个分别对应于加、减、替换
21                     dp[i][j] = min(dp[i-1][j],
22                                     dp[i][j-1],
23                                     dp[i-1][j-1]
24                                     )+1
25         return dp[-1][-1]

```

```

1  #
2  # @lc app=leetcode.cn id=73 lang=python3
3  #
4  # [73] 矩阵置零
5  #
6  class Solution:
7      def setZeroes(self, matrix: List[List[int]]) -> None:
8          """
9          # 直接法
10         row = []
11         col = []
12         m = len(matrix)
13         n = len(matrix[0])
14         for i in range(m):

```



```

15         for j in range(n):
16             if matrix[i][j] == 0:
17                 row.append(i)
18                 col.append(j)
19     row = set(row)
20     col = set(col)
21
22     for i in row:
23         for j in range(n):
24             matrix[i][j] = 0
25     for j in col :
26         for i in range(m):
27             matrix[i][j] = 0
28
29     return matrix
30     """
31     # 第一行出现一个0
32     firstRowHasZero = not all(matrix[0])
33     m = len(matrix)
34     n = len(matrix[0])
35     # 第一行第一列做标记
36     for i in range(1,m):
37         for j in range(n):
38             if matrix[i][j] == 0:
39                 matrix[0][j] = matrix[i][0] = 0
40     # 置0
41     for i in range(1,m):
42         for j in range(n-1,-1,-1):
43             if matrix[i][0] == 0 or matrix[0][j] == 0:
44                 matrix[i][j] = 0
45     # 补一下第一行的
46
47     if firstRowHasZero:
48         matrix[0] = [0] * n
49
50     return matrix

```

```

1  #
2  # @lc app=leetcode.cn id=74 lang=python3
3  #
4  # [74] 搜索二维矩阵
5  #
6  class Solution:
7      def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
8          if len(matrix)==0 or len(matrix[0])==0 or target < matrix[0][0] or target > matrix
              [-1][-1]:

```

```

9         return False
10    row = 0
11    col = len(matrix[0]) - 1
12    while row < len(matrix) and col >= 0 :
13        if matrix[row][col] > target:
14            col -= 1
15        elif matrix[row][col] < target:
16            row += 1
17        else :
18            return True
19    return False

```

```

1  #
2  # @lc app=leetcode.cn id=75 lang=python3
3  #
4  # [75] 颜色分类
5  #
6  class Solution:
7      def sortColors( self , nums: List[int]) -> None:
8          count = [0,0,0]
9          for num in nums:
10              count[num] += 1
11          idx = 0
12          for i in range(3):
13              for j in range(count[i]):
14                  nums[idx] = i
15                  idx += 1

```

```

1  #
2  # @lc app=leetcode.cn id=76 lang=python3
3  #
4  # [76] 最小覆盖子串
5  #
6  class Solution:
7      def minWindow(self, s: str , t: str) -> str:
8          if s is None or len(s) < len(t):
9              return ""
10         res = ""
11         left = 0
12         right = 0
13         min_len = len(s)
14         count = 0
15
16         m = {}
17         # 统计t中字符数目
18         for i in t:

```

```

19         m[i] = m.get(i,0) + 1
20
21     while right < len(s):
22         if s[right] in m:
23             # 先找到一个区间能包含t,但长度不一定是最短的
24             m[s[right]] -= 1
25             if m[s[right]] >= 0:
26                 count += 1
27             # 找到了一个区间
28             while (count == len(t)):
29                 # 选择更短的子串
30                 if (right - left + 1 < min_len):
31                     min_len = right - left + 1
32                     res = s[left:right+1]
33
34             if s[left] in m:
35                 m[s[left]] += 1
36                 if m[s[left]] > 0:
37                     count -= 1
38             left += 1
39         right += 1
40
41     return res

```

```

1  #
2  # @lc app=leetcode.cn id=77 lang=python3
3  #
4  # [77] 组合
5  #
6  class Solution:
7      def combine(self, n: int, k: int) -> List[List[int]]:
8          res = []
9          self.dfs(n,k,1,[], res)
10         return res
11
12     def dfs(self, n, k, start, path, res):
13         if 0 == k and path not in res:
14             res.append(path)
15         for i in range(start, n+1):
16             self.dfs(n, k-1, i+1, path+[i], res)

```

```

1  #
2  # @lc app=leetcode.cn id=78 lang=python3
3  #
4  # [78] 子集
5  #

```

```

6 class Solution:
7     def subsets( self , nums: List[int]) -> List[List[int ]]:
8         res = []
9         nums.sort()
10        self.dfs(nums, 0, [], res)
11        return res
12
13    def dfs( self , nums, index, path, res):
14        res.append(path)
15        for i in range(index, len(nums)):
16            self.dfs(nums, i+1, path+[nums[i]], res)

```

```

1 #
2 # @lc app=leetcode.cn id=79 lang=python3
3 #
4 # [79] 单词搜索
5 #
6 class Solution:
7     def exist( self , board: List[List[str]], word: str) -> bool:
8         m , n = len(board) , len(board[0])
9         visited = [[False for i in range(n)] for i in range(m)]
10        # 遍历寻找开头
11        for i in range(m):
12            for j in range(n):
13                if self.dfs(board,word,visited,i,j,0):
14                    return True
15        return False
16
17    def dfs( self ,board,word,visited,i,j,start):
18        # 终止条件
19        if start == len(word):
20            return True
21        # 溢出 剪枝 or 已经访问过了
22        if i < 0 or j < 0 or i >= len(board) or j >= len(board[0]) or visited[i][j] or board[i][j]
23            != word[start]:
24            return False
25
26        if board[i][j] == word[start]:
27            visited[i][j] = True
28            ret = self.dfs(board,word,visited,i+1,j,start+1) or \
29                self.dfs(board,word,visited,i-1,j,start+1) or \
30                self.dfs(board,word,visited,i,j+1,start+1) or \
31                self.dfs(board,word,visited,i,j-1,start+1)
32            visited[i][j] = False
33        return ret

```

```

1 #
2 # @lc app=leetcode.cn id=80 lang=python3
3 #
4 # [80] 删除排序数组中的重复项 II
5 #
6 class Solution:
7     def removeDuplicates(self, nums: List[int]) -> int:
8         if not nums:
9             return 0
10        # 初始化第一个
11        i, count = 1, 1
12
13        while i < len(nums) :
14            if nums[i] == nums[i - 1]:
15                count += 1
16                if count > 2:
17                    nums.pop(i)
18                    # 这里的减一和后面对消
19                    i -= 1
20            else :
21                count = 1
22            i += 1
23        return len(nums)

```

```

1 #
2 # @lc app=leetcode.cn id=81 lang=python3
3 #
4 # [81] 搜索旋转排序数组 II
5 #
6 class Solution:
7     def search(self, nums: List[int], target: int) -> bool:
8         if not nums:
9             return False
10        l, r = 0, len(nums) - 1
11
12        while l <= r:
13            mid = (l+r)//2
14            if nums[mid] == target:
15                return True
16            # mid在前半段 或者l mid r 都在右边
17            if nums[l] < nums[mid]:
18                if nums[l] <= target < nums[mid]:
19                    r = mid - 1
20            else :
21                l = mid + 1
22        # l 在左半段 、 mid 在右半段

```

```

23         elif nums[mid] < nums[l]:
24             if nums[mid] < target <= nums[r]:
25                 l = mid + 1
26             else:
27                 r = mid - 1
28         else:
29             l += 1
30     return False

```

```

1  #
2  # @lc app=leetcode.cn id=82 lang=python3
3  #
4  # [82] 删除排序链表中的重复元素 II
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def deleteDuplicates(self, head: ListNode) -> ListNode:
14         dummy = ListNode(0)
15         dummy.next = head
16         prev = dummy
17
18         while head and head.next:
19             if head.val == head.next.val:
20                 while head and head.next and head.val == head.next.val:
21                     head = head.next
22                 head = head.next
23                 prev.next = head
24             # 两个指针都往后走
25             else:
26                 prev = prev.next
27                 head = head.next
28     return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=83 lang=python3
3  #
4  # [83] 删除排序链表中的重复元素
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):

```

```

9      self.val = x
10     self.next = None
11
12     class Solution:
13         def deleteDuplicates(self, head: ListNode) -> ListNode:
14             point = head
15             while point:
16                 while point.next and point.val == point.next.val:
17                     point.next = point.next.next
18                 point = point.next
19             return head

```

```

1  #
2  # @lc app=leetcode.cn id=84 lang=python3
3  #
4  # [84] 柱状图中最大的矩形
5  #
6  class Solution:
7      def largestRectangleArea(self, heights: List[int]) -> int:
8          # 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
9          # 最大面积，所以补个0，使之最后可以出栈
10         heights.append(0)
11         stack = [-1]
12         res = 0
13
14         for idx, val in enumerate(heights):
15             # 不是递增栈
16             while heights[stack[-1]] > val:
17                 h = heights[stack.pop()]
18                 w = idx - stack[-1] - 1
19                 res = max(res, h*w)
20             stack.append(idx)
21         return res

```

```

1  #
2  # @lc app=leetcode.cn id=85 lang=python3
3  #
4  # [85] 最大矩形
5  #
6  class Solution:
7      def maximalRectangle(self, matrix: List[List[str]]) -> int:
8          """
9          if not matrix or not matrix[0]:
10             return 0
11             m, n = len(matrix), len(matrix[0])
12             # height 的尾部多了一个0,防止递增错误

```

```

13     height = [0] * (n+1)
14     max_area = 0
15     for i in range(m):
16         # 计算h
17         for j in range(n):
18             # 遍历到的每行的h
19             height[j] = height[j]+1 if matrix[i][j]=='1' else 0
20         # 找出所有h和w的组合
21         # 同84题
22         stack = [-1]
23         for k in range(n + 1):
24             while height[k] < height[stack[-1]]:
25                 h = height[stack.pop()]
26                 w = k - stack[-1] - 1
27                 max_area = max(max_area, h * w)
28             stack.append(k)
29     return max_area
30     """
31     if not matrix or not matrix[0]:
32         return 0
33     m , n = len(matrix),len(matrix[0])
34     # 申请辅助数组并初始化
35     # 向上、向左、向右能延伸到的最远的地方
36     left , right , height = [0]*n, [n]*n, [0]*n
37     max_A = 0
38     # 从第一行开始遍历
39     for i in range(m):
40         # 用来记录下标
41         cur_left, cur_right = 0, n
42         # 从第一个元素开始遍历
43         for j in range(n):
44             # 如果矩阵中当前坐标为1时， 我们将height对应的下标加一
45             # left取cur_left和left [i] 中取最大的
46             if matrix[i][j] == "1":
47                 height[j] = height[j] + 1
48                 left [j] = max(left[j], cur_left)
49             else: # 否则赋值位0
50                 height[j], left [j] = 0, 0
51                 cur_left = j+1
52         # right数组从末尾开始遍历
53         for j in range(n-1, -1, -1):
54             if matrix[i][j] == "1":
55                 right [j] = min(right[j], cur_right)
56             else:
57                 right [j] = n
58                 cur_right = j

```



```

59         for j in range(n):
60             # 计算到前行为止最大的面积
61             max_A = max(max_A, (right[j] - left[j]) * height[j])
62     return max_A

```

```

1  #
2  # @lc app=leetcode.cn id=86 lang=python3
3  #
4  # [86] 分隔链表
5  #
6  # Definition for singly-linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10     #         self.next = None
11
12     class Solution:
13         def partition(self, head: ListNode, x: int) -> ListNode:
14             h1 = l1 = ListNode(0)
15             h2 = l2 = ListNode(0)
16
17             while head:
18                 if head.val < x:
19                     l1.next = head
20                     l1 = l1.next
21                 else :
22                     l2.next = head
23                     l2 = l2.next
24                 head = head.next
25             # l1 l2都在各自的尾部了
26             l2.next = None
27             l1.next = h2.next
28
29             return h1.next

```

```

1  #
2  # @lc app=leetcode.cn id=88 lang=python3
3  #
4  # [88] 合并两个有序数组
5  #
6  class Solution:
7     def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
8         # 从后往前
9         p1 = m - 1
10        p2 = n - 1
11        p = m + n - 1

```

```

12     # 两个都没放完
13     while p1 >= 0 and p2 >= 0:
14         if nums1[p1] >= nums2[p2]:
15             nums1[p] = nums1[p1]
16             p1 -= 1
17         else:
18             nums1[p] = nums2[p2]
19             p2 -= 1
20         p += 1
21     # p1没放完, 那就不用再操作了
22     # p2没放完
23     while p2 >= 0:
24         nums1[p] = nums2[p2]
25         p += 1
26         p2 -= 1

```

```

1  #
2  # @lc app=leetcode.cn id=89 lang=python3
3  #
4  # [89] 格雷编码
5  #
6  class Solution:
7      def grayCode(self, n: int) -> List[int]:
8          res = [0]
9          for i in range(n):
10             for j in range(len(res)-1, -1, -1):
11                 res.append(res[j] + (1 << i))
12         return res

```

```

1  #
2  # @lc app=leetcode.cn id=90 lang=python3
3  #
4  # [90] 子集 II
5  #
6  class Solution:
7      def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
8          res = []
9          nums.sort()
10         # self.dfs(nums, 0, [], res)
11         self.dfs2(nums, 0, [], res)
12         return res
13
14     def dfs(self, nums, index, path, res):
15         if path not in res:
16             res.append(path)
17         for i in range(index, len(nums)):

```

```

18         self.dfs(nums, i+1, path+[nums[i]], res)
19
20     def dfs2(self, nums, index, path, res):
21         res.append(path)
22         for i in range(index, len(nums)):
23             if i > index and nums[i] == nums[i-1]:
24                 continue
25             self.dfs2(nums, i+1, path+[nums[i]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=91 lang=python3
3  #
4  # [91] 解码方法
5  #
6  class Solution:
7      def numDecodings(self, s: str) -> int:
8          if s is None or s[0] == '0':
9              return 0
10         # 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
11         # [0] 初始化为1
12         dp = [0] * (len(s)+1)
13         dp[0] = dp[1] = 1
14         for i in range(2, len(s)+1):
15             if s[i-1] >= '1' and s[i-1] <= '9':
16                 dp[i] += dp[i-1]
17             if s[i-2] == '1' or (s[i-2] == '2' and s[i-1] <= '6'):
18                 dp[i] += dp[i-2]
19         return dp[-1]

```

```

1  #
2  # @lc app=leetcode.cn id=92 lang=python3
3  #
4  # [92] 反转链表 II
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10         self.next = None
11
12 class Solution:
13     def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
14         dummy = ListNode(0)
15         dummy.next = head
16         prev = dummy
17         # 走m-1个

```

```

18     for i in range(m-1):
19         prev = prev.next
20     # 反转
21     temp = None
22     cur = prev.next
23     for i in range(n-m+1):
24         next = cur.next
25         # reverse
26         cur.next = temp
27         temp = cur
28         # 下一个
29         cur = next
30     # cur指向的是最后部分,中间已经没有了
31     # None 的下一个
32     # 最后面一段
33     prev.next.next = cur
34     '''
35     wi = temp
36     while wi.next :
37         wi = wi.next
38     wi.next = cur
39     '''
40     # 中间一段
41     prev.next = temp
42
43     return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=93 lang=python3
3  #
4  # [93] 复原IP地址
5  #
6  class Solution:
7      def restoreIpAddresses(self, s: str) -> List[str]:
8          res = []
9          self.dfs(s, [], res, 0)
10         return res
11
12     def dfs(self, s, ip, res, start):
13         # 终止条件
14         if len(ip) == 4 and start == len(s):
15             address = '.'.join(ip)
16             res.append(address)
17             return
18
19         # 特殊场景下可以剪枝

```

```

20 # 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)
21 if len(s) - start > 3*(4-len(ip)) or len(s) - start < (4-len(ip)):
22     return
23
24 # 最多三位(+0,+1,+2)
25 for i in range(0,3):
26     substr = s[start:start+i+1]
27     # 允许单个0,但是不允许0开头的一串,比如025
28     if i != 0 and substr[0] == '0':
29         continue
30     if int(substr) >= 0 and int(substr) <= 255:
31         self.dfs(s,ip+[substr],res,start + i + 1)

```

```

1 #
2 # @lc app=leetcode.cn id=94 lang=python3
3 #
4 # [94] 二叉树的中序遍历
5 #
6 # Definition for a binary tree node.
7 # class TreeNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def inorderTraversal(self, root: TreeNode) -> List[int]:
15         if root is None:
16             return []
17         """
18         result = []
19         stack = []
20         p = root
21         while stack or p:
22             # 先把左边的压进去
23             if p:
24                 stack.append(p)
25                 p = p.left
26             else:
27                 p = stack.pop()
28                 result.append(p.val)
29                 p = p.right
30
31         return result
32         """
33

```

```

34         return self.inorder(root)
35
36     def inorder(self, r):
37         if r:
38             return self.inorder(r.left) + [r.val] + self.inorder(r.right)
39         else:
40             return []

```

```

1  #
2  # @lc app=leetcode.cn id=95 lang=python3
3  #
4  # [95] 不同的二叉搜索树 II
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def generateTrees(self, n: int) -> List[TreeNode]:
15         if n == 0:
16             return []
17         return self.get_trees(1, n)
18
19     def get_trees(self, start, end):
20         res = []
21         if start > end:
22             # 空子树情况
23             return [None]
24         for i in range(start, end+1):
25             lefts = self.get_trees(start, i-1)
26             rights = self.get_trees(i+1, end)
27             # lefts 和 rights 有可能是空的[None]
28             for l in lefts:
29                 for r in rights:
30                     root = TreeNode(i)
31                     root.left = l
32                     root.right = r
33                     res.append(root)
34         return res

```

```

1  #
2  # @lc app=leetcode.cn id=96 lang=python3
3  #

```

```

4 # [96] 不同的二叉搜索树
5 #
6 class Solution:
7     def numTrees(self, n: int) -> int:
8         f = [0 for _ in range(n+1)]
9         f[0] = f[1] = 1
10        for k in range(2,n+1):
11            for i in range(k+1):
12                f[k] += f[i-1]*f[k-i]
13        return f[n]

```

```

1 #
2 # @lc app=leetcode.cn id=97 lang=python3
3 #
4 # [97] 交错字符串
5 #
6 class Solution:
7     def isInterleave ( self , s1: str , s2: str , s3: str ) -> bool:
8         l1, l2, l3 = len(s1), len(s2), len(s3)
9         if l1+l2 != l3:
10            return False
11
12        dp = [[True for _ in range(l2+1)] for _ in range(l1+1)]
13        # 边界条件
14        # 用s1去填
15        for i in range(1, l1+1):
16            dp[i][0] = dp[i-1][0] and s1[i-1] == s3[i-1]
17        # 用s2去填
18        for j in range(1, l2+1):
19            dp[0][j] = dp[0][j-1] and s2[j-1] == s3[j-1]
20
21        for i in range(1, l1+1):
22            for j in range(1, l2+1):
23                dp[i][j] = (dp[i-1][j] and s1[i-1] == s3[i+j-1]) or \
24                    (dp[i][j-1] and s2[j-1] == s3[i+j-1])
25
26        return dp[l1][l2]

```

```

1 #
2 # @lc app=leetcode.cn id=98 lang=python3
3 #
4 # [98] 验证二叉搜索树
5 #
6 # Definition for a binary tree node.
7 # class TreeNode:
8 #     def __init__(self, x):

```

```

9      #         self.val = x
10     #         self.left = None
11     #         self.right = None
12
13     class Solution:
14         def isValidBST(self, root: TreeNode) -> bool:
15             return self.isOK(root, -float('inf'), float('inf'))
16
17         def isOK(self, root, low, upper):
18             if root is None:
19                 return True
20             elif root.val > low and root.val < upper :
21                 return self.isOK(root.left, low, root.val) and self.isOK(root.right, root.val, upper)
22             else:
23                 return False

```

```

1  #
2  # @lc app=leetcode.cn id=99 lang=python3
3  #
4  # [99] 恢复二叉搜索树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10     #         self.left = None
11     #         self.right = None
12
13     class Solution:
14         def recoverTree(self, root: TreeNode) -> None:
15             cur, pre = root, None
16             first, second = None, None
17             stack = []
18
19             while cur or stack:
20                 if cur:
21                     stack.append(cur)
22                     cur = cur.left
23                 else:
24                     node = stack.pop()
25                     if pre and pre.val >= node.val:
26                         if not first:
27                             first = pre
28                             second = node
29
30                     pre = node

```



```

31         cur = node.right
32
33     first.val, second.val = second.val, first.val
34     """
35     # 定义
36     self.pre = None
37     self.m1, self.m2 = None, None
38
39     self.inorderTraversal(root)
40     self.m1.val, self.m2.val = self.m2.val, self.m1.val
41     """
42
43     # 中序遍历
44     def inorderTraversal( self , root):
45         if root :
46             self.inorderTraversal(root.left )
47             if self.pre and self.pre.val > root.val:
48                 if self.m1 == None:
49                     self.m1 = self.pre
50                     self.m2 = root
51             self.pre = root
52             self.inorderTraversal(root.right )

```

```

1  #
2  # @lc app=leetcode.cn id=100 lang=python3
3  #
4  # [100] 相同的树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
15         if p is None and q is None:
16             return True
17         elif p and q and p.val == q.val:
18             return self.isSameTree(p.left, q.left ) and self.isSameTree(p.right, q.right )
19         elif p or q :
20             return False

```

```

1  #
2  # @lc app=leetcode.cn id=101 lang=python3

```

```

3  #
4  # [101] 对称二叉树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def isSymmetric(self, root: TreeNode) -> bool:
15         if root is None:
16             return True
17         return self.yes(root.left, root.right)
18
19     def yes(self, left, right):
20         if left is None and right is None:
21             return True
22         if left and right and left.val == right.val:
23             if self.yes(left.left, right.right) and self.yes(left.right, right.left):
24                 return True
25         return False

```

```

1  #
2  # @lc app=leetcode.cn id=102 lang=python3
3  #
4  # [102] 二叉树的层次遍历
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def levelOrder(self, root: TreeNode) -> List[List[int]]:
15         if not root:
16             return []
17         result = [[]]
18         self.traverse(root, 0, result)
19         return result
20
21     def traverse(self, root, level, result):
22         if not root:

```

```

23         return
24     if level >= len(result):
25         result.append([])
26     result[level].append(root.val)
27     self.traverse(root.left, level+1, result)
28     self.traverse(root.right, level+1, result)

```

```

1  #
2  # @lc app=leetcode.cn id=103 lang=python3
3  #
4  # [103] 二叉树的锯齿形层次遍历
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
15         if not root:
16             return []
17         result = []
18         self.traverse(root, 0, result, True)
19         return result
20
21     def traverse(self, root, level, result, flag):
22         if root is None:
23             return
24         if level >= len(result):
25             result.append([])
26
27         if flag:
28             result[level].append(root.val)
29         else:
30             result[level].insert(0, root.val)
31         self.traverse(root.left, level+1, result, not flag)
32         self.traverse(root.right, level+1, result, not flag)

```

```

1  #
2  # @lc app=leetcode.cn id=104 lang=python3
3  #
4  # [104] 二叉树的最大深度
5  #
6  # Definition for a binary tree node.

```

```

7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def maxDepth(self, root: TreeNode) -> int:
15         if root is None:
16             return 0
17         elif root.left and root.right:
18             return 1 + max(self.maxDepth(root.left), self.maxDepth(root.right))
19         elif root.left:
20             return 1 + self.maxDepth(root.left)
21         elif root.right:
22             return 1 + self.maxDepth(root.right)
23         else:
24             return 1

```

```

1  #
2  # @lc app=leetcode.cn id=105 lang=python3
3  #
4  # [105] 从前序与中序遍历序列构造二叉树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
15         if inorder:
16             # 前序的头就是root
17             # 中序中,root左边就是左子树,右边是右子树
18             idx = inorder.index(preorder.pop(0))
19             root = TreeNode(inorder[idx])
20             # 递归构造子树先left后right
21             root.left = self.buildTree(preorder, inorder[0:idx])
22             root.right = self.buildTree(preorder, inorder[idx+1:])
23             return root
24         else:
25             return None

```

```

1  #

```

```

2  # @lc app=leetcode.cn id=106 lang=python3
3  #
4  # [106] 从中序与后序遍历序列构造二叉树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:
15         if inorder:
16             # 后序的尾部就是root
17             # 中序中,root值左边就是左子树,右边是右子树
18             idx = inorder.index(postorder.pop())
19             root = TreeNode(inorder[idx])
20             # 递归构造子树先right后left
21             root.right = self.buildTree(inorder[idx+1:], postorder)
22             root.left = self.buildTree(inorder[0:idx], postorder)
23             return root
24         else:
25             return None

```

```

1  #
2  # @lc app=leetcode.cn id=107 lang=python3
3  #
4  # [107] 二叉树的层次遍历 II
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
15
16         if not root:
17             return []
18         # use stack
19         stack = [[root]]
20         res = []
21         while stack:

```

```

22         # 取出最新装入的list
23         top = stack.pop()
24         # 一直在头部插入以达到倒序
25         res.insert(0, [t.val for t in top])
26         # 向下新一轮扫描
27         temp = []
28         for node in top:
29             if node.left :
30                 temp.append(node.left)
31             if node.right :
32                 temp.append(node.right)
33         if temp:
34             stack.append(temp)
35     return res
36     """
37     # 递归法
38     if not root:
39         return []
40     result = [[]]
41     self.traverse(root,0, result)
42     result.reverse()
43     return result
44     """
45
46     def traverse(self, root, level, result):
47         if root is None:
48             return
49         if level >= len(result):
50             result.append([])
51         result[level].append(root.val)
52         self.traverse(root.left, level+1, result)
53         self.traverse(root.right, level+1, result)

```

```

1  #
2  # @lc app=leetcode.cn id=108 lang=python3
3  #
4  # [108] 将有序数组转换为二叉搜索树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:

```

```

14 def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
15     if not nums :
16         return None
17     mid = len(nums)//2
18
19     root = TreeNode(nums[mid])
20     root.left = self.sortedArrayToBST(nums[:mid])
21     root.right = self.sortedArrayToBST(nums[mid+1:])
22
23     return root

```

```

1 #
2 # @lc app=leetcode.cn id=109 lang=python3
3 #
4 # [109] 有序链表转换二叉搜索树
5 #
6 # Definition for singly-linked list .
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 # Definition for a binary tree node.
13 # class TreeNode:
14 #     def __init__(self, x):
15 #         self.val = x
16 #         self.left = None
17 #         self.right = None
18
19 class Solution:
20     def sortedListToBST(self, head: ListNode) -> TreeNode:
21         """
22         if not head :
23             return None
24         if not head.next:
25             return TreeNode(head.val)
26
27         slow = head
28         fast = head.next.next
29         while fast and fast.next:
30             fast = fast.next.next
31             slow = slow.next
32         head2 = slow.next
33         slow.next = None
34         root = TreeNode(head2.val)
35         root.left = self.sortedListToBST(head)

```

```

36     root.right = self.sortedListToBST(head2.next)
37     return root
38     """
39
40     if not head :
41         return None
42     nums = []
43     while head:
44         nums.append(head.val)
45         head = head.next
46     return self.sortedArrayToBST(nums)
47
48     def sortedArrayToBST(self, nums):
49         if not nums :
50             return None
51         mid = len(nums)//2
52
53         root = TreeNode(nums[mid])
54         root.left = self.sortedArrayToBST(nums[:mid])
55         root.right = self.sortedArrayToBST(nums[mid+1:])
56
57         return root

```

```

1  #
2  # @lc app=leetcode.cn id=110 lang=python3
3  #
4  # [110] 平衡二叉树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10     #         self.left = None
11     #         self.right = None
12
13 class Solution:
14     def isBalanced(self, root: TreeNode) -> bool:
15         return self.check(root) != -1
16
17     def check(self, root):
18         if root is None:
19             return 0
20         l = self.check(root.left)
21         r = self.check(root.right)
22         if l == -1 or r == -1 or abs(l-r)>1:
23             return -1

```



```
24     return 1 + max(l,r)
```

```
1  #
2  # @lc app=leetcode.cn id=111 lang=python3
3  #
4  # [111] 二叉树的最小深度
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def minDepth(self, root: TreeNode) -> int:
15         if root is None:
16             return 0
17         if root.left is None or root.right is None:
18             return self.minDepth(root.left) + self.minDepth(root.right) + 1
19         return min(self.minDepth(root.left) , self.minDepth(root.right)) + 1
```

```
1  #
2  # @lc app=leetcode.cn id=112 lang=python3
3  #
4  # [112] 路径总和
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def hasPathSum(self, root: TreeNode, sum: int) -> bool:
15         if root is None:
16             return False
17
18         sum -= root.val
19         if sum == 0 and root.left is None and root.right is None:
20             return True
21         left = self.hasPathSum(root.left,sum)
22         right = self.hasPathSum(root.right,sum)
23         return left or right
```

```

1  #
2  # @lc app=leetcode.cn id=113 lang=python3
3  #
4  # [113] 路径总和 II
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
15         if root is None:
16             return []
17         result = []
18         self.dfs(root, sum, [], result)
19         return result
20
21     def dfs(self, root, sum, path, result):
22         if root is None:
23             return
24         if root.left is None and root.right is None and sum == root.val:
25             path.append(root.val)
26             result.append(path)
27
28         self.dfs(root.left, sum - root.val, path + [root.val], result)
29         self.dfs(root.right, sum - root.val, path + [root.val], result)

```

```

1  #
2  # @lc app=leetcode.cn id=114 lang=python3
3  #
4  # [114] 二叉树展开为链表
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def flatten(self, root: TreeNode) -> None:
15         if root is None:
16             return

```

```

17
18     self.flatten(root.left)
19     self.flatten(root.right)
20
21     if root.left is None:
22         return
23
24     # 左子树插到root和root.right之间
25     p = root.left
26     # 左子链的最后一个节点
27     while p.right:
28         p = p.right
29     p.right = root.right
30     root.right = root.left
31     root.left = None

```

```

1  #
2  # @lc app=leetcode.cn id=115 lang=python3
3  #
4  # [115] 不同的子序列
5  #
6  class Solution:
7      def numDistinct(self, s: str, t: str) -> int:
8          if s is None or t is None:
9              return 0
10         ls = len(s)
11         lt = len(t)
12         dp = [ [0 for _ in range(lt+1) ] for _ in range(ls+1)]
13
14         # init
15         # 当子串长度为0时, 所有次数都是1
16         # 当母串长度为0时, 所有次数都是0 (默认是0,不用重复了)
17         for i in range(ls+1):
18             dp[i][0] = 1
19
20         for i in range(1, ls+1):
21             for j in range(1, lt+1):
22                 # 跳过上一个字符串匹配过程
23                 dp[i][j] = dp[i-1][j]
24                 # 要匹配的话
25                 if s[i-1] == t[j-1]:
26                     dp[i][j] += dp[i-1][j-1]
27
28         return dp[-1][-1]

```

```

1  #

```

```

2  # @lc app=leetcode.cn id=116 lang=python3
3  #
4  # [116] 填充每个节点的下一个右侧节点指针
5  #
6  """
7  # Definition for a Node.
8  class Node:
9      def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' =
        None):
10         self.val = val
11         self.left = left
12         self.right = right
13         self.next = next
14 """
15 class Solution:
16     def connect(self, root: 'Node') -> 'Node':
17         if root is None or root.left is None:
18             return root
19         # 左右链接
20         root.left.next = root.right
21         if root.next:
22             root.right.next = root.next.left
23         else:
24             root.right.next = None
25
26         self.connect(root.left)
27         self.connect(root.right)
28
29         return root

```

```

1  #
2  # @lc app=leetcode.cn id=117 lang=python3
3  #
4  # [117] 填充每个节点的下一个右侧节点指针 II
5  #
6  """
7  # Definition for a Node.
8  class Node:
9      def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' =
        None):
10         self.val = val
11         self.left = left
12         self.right = right
13         self.next = next
14 """
15 class Solution:

```

```

16 def connect(self, root: 'Node') -> 'Node':
17     head = root
18     dummy = Node(-1)
19     prev = dummy
20     # dummy 当前行的最左端节点
21     while root :
22         if root.left :
23             prev.next = root.left
24             prev = prev.next
25         if root.right :
26             prev.next = root.right
27             prev = prev.next
28         root = root.next
29         # 行的尾部
30         if root is None:
31             # dummy.next为前面prev.next 第一次赋值的节点
32             root = dummy.next
33             #前面链接断开,开始新的一行
34             dummy.next = None
35             # prev值新的
36             prev = dummy
37     return head

```

```

1 #
2 # @lc app=leetcode.cn id=118 lang=python3
3 #
4 # [118] 杨辉三角
5 #
6 class Solution:
7     def generate(self, numRows: int) -> List[List[int]]:
8         # 全部都用1先填充
9         out = [[1]*(i+1) for i in range(numRows)]
10        for r in range(numRows):
11            for col in range(1,r):
12                out[r][col] = out[r-1][col-1] + out[r-1][col]
13        return out

```

```

1 #
2 # @lc app=leetcode.cn id=119 lang=python3
3 #
4 # [119] 杨辉三角 II
5 #
6 class Solution:
7     def getRow(self, rowIndex: int) -> List[int]:
8         """
9         if rowIndex == 0:

```

```

10         return [1]
11         rowIndex += 1
12         # 全部都用1先填充
13         out = [[1]*(i+1) for i in range(rowIndex)]
14         for r in range(rowIndex):
15             for col in range(1,r):
16                 out[r][col] = out[r-1][col-1] + out[r-1][col]
17         return out[-1]
18     """
19     # 先用1填充
20     res = [1]*(rowIndex+1)
21     # 从后往前,从上往下覆盖
22     for r in range(2,rowIndex+1):
23         for col in range(r-1,0,-1):# 逆序
24             res[col] += res[col-1]
25     return res

```

```

1  #
2  # @lc app=leetcode.cn id=120 lang=python3
3  #
4  # [120] 三角形最小路径和
5  #
6  class Solution:
7      def minimumTotal(self, triangle: List[List[int]]) -> int:
8          if not triangle:
9              return
10         # 倒数第二行到最上面一行
11         for i in range(len(triangle)-2, -1, -1):
12             # 每行的第一列到最后一列
13             for j in range(len(triangle[i])):
14                 triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1])
15         return triangle[0][0]

```

```

1  #
2  # @lc app=leetcode.cn id=121 lang=python3
3  #
4  # [121] 买卖股票的最佳时机
5  #
6  class Solution:
7      def maxProfit(self, prices: List[int]) -> int:
8          if not prices:
9              return 0
10         minelement = float('inf')
11         profit = 0
12         for i in range(len(prices)):
13             minelement = min(minelement, prices[i])

```

```
14         profit = max(profit, prices[i] - minelement)
15     return profit
```

```
1 #
2 # @lc app=leetcode.cn id=122 lang=python3
3 #
4 # [122] 买卖股票的最佳时机 II
5 #
6 class Solution:
7     def maxProfit(self, prices: List[int]) -> int:
8         if not prices:
9             return 0
10        profit = 0
11        for i in range(1, len(prices)):
12            if prices[i] > prices[i-1]:
13                profit += (prices[i] - prices[i-1])
14        return profit
```

```
1 #
2 # @lc app=leetcode.cn id=123 lang=python3
3 #
4 # [123] 买卖股票的最佳时机 III
5 #
6 class Solution:
7     def maxProfit(self, prices: List[int]) -> int:
8         """
9         """
10        对于任意一天考虑四个变量:
11        fstBuy: 在该天第一次买入股票可获得的最大收益
12        fstSell: 在该天第一次卖出股票可获得的最大收益
13        secBuy: 在该天第二次买入股票可获得的最大收益
14        secSell: 在该天第二次卖出股票可获得的最大收益
15        分别对四个变量进行相应的更新, 最后secSell就是最大
16        收益值(secSell >= fstSell)
17        """
18        fstBuy, fstSell = -float('inf'), 0
19        secBuy, secSell = -float('inf'), 0
20        for i in prices:
21            fstBuy = max(fstBuy, -i)
22            fstSell = max(fstSell, fstBuy + i)
23            secBuy = max(secBuy, fstSell - i)
24            secSell = max(secSell, secBuy + i)
25        return secSell
26        """
27        if not prices:
28            return 0
```

```

29     num = len(prices)
30
31     forward = [0]*num
32     backward = [0]*num
33     # 前向
34     current_min = prices[0]
35     for i in range(1,len(prices)):
36         current_min = min(current_min, prices[i])
37         forward[i] = max(forward[i-1] , prices[i]-current_min )
38     # 后向
39     total_max = 0
40     current_max = prices[-1]
41     for i in range(len(prices) - 2, -1, -1):
42         current_max = max(current_max, prices[i])
43         backward[i] = max(backward[i+1], current_max - prices[i])
44         total_max = max(total_max, backward[i] + forward[i])
45     return total_max

```

```

1  #
2  # @lc app=leetcode.cn id=124 lang=python3
3  #
4  # [124] 二叉树中的最大路径和
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def maxPathSum(self, root: TreeNode) -> int:
15         self.res = -float('inf')
16         self.maxend(root)
17         return self.res
18
19     def maxend(self, root):
20         if root is None:
21             return 0
22         left = self.maxend(root.left)
23         right = self.maxend(root.right)
24         self.res = max(self.res, left + root.val + right)
25         return max(root.val + max(left, right), 0)

```

```

1  #
2  # @lc app=leetcode.cn id=125 lang=python3

```



```

3  #
4  # [125] 验证回文串
5  #
6  class Solution:
7      def isPalindrome(self, s: str) -> bool:
8          # 检测字符串是否由字母和数字组成
9          alnum = [t.lower() for t in s if t.isalnum()]
10         leng = len(alnum)
11         mid = leng//2
12         if leng < 2:
13             return True
14         for i in range(mid):
15             if alnum[i] != alnum[leng - i - 1]:
16                 return False
17         return True

```

```

1  #
2  # @lc app=leetcode.cn id=126 lang=python3
3  #
4  # [126] 单词接龙 II
5  #
6  class Solution:
7      def findLadders(self, beginWord: str, endWord: str, wordList: List[str]) -> List[List[str]]:
8          import collections
9          wordset = set(wordList)
10
11         level = {beginWord}
12         parents = collections.defaultdict(set)
13
14         while level and endWord not in parents:
15             next_level = collections.defaultdict(set)
16             for word in level:
17                 # 不同位置都可以插入不同字母进行新单词重构
18                 for i in range(len(beginWord)):
19                     for c in 'abcdefghijklmnopqrstuvwxyz':
20                         newWord = word[:i] + c + word[i+1:]
21                         if newWord in wordset and newWord not in parents:
22                             next_level[newWord].add(word)
23
24             level = next_level
25             parents.update(next_level)
26         res = [[endWord]]
27         # parents相当于是逆向
28         while res and res[0][0] != beginWord:
29             # 确定是等长的
30             res = [[p]+r for r in res for p in parents[r[0]]]

```

```
31 return res
```

```
1 #
2 # @lc app=leetcode.cn id=127 lang=python3
3 #
4 # [127] 单词接龙
5 #
6 class Solution:
7     def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
8         # 防止时间超出
9         wordset = set(wordList)
10        # 初始化
11        bfs = [(beginWord, 1)]
12        while bfs:
13            word,length = bfs.pop(0) # 左边弹出
14            if word == endWord:
15                return length
16            for i in range(len(word)):
17                for c in "abcdefghijklmnopqrstuvwxyz":
18                    # 不同位置都可以插入不同字母进行新单词重构
19                    newWord = word[:i] + c + word[i + 1:]
20                    if newWord in wordset and newWord != word:
21                        wordset.remove(newWord)
22                        bfs.append((newWord, length + 1))
23        return 0
```

```
1 #
2 # @lc app=leetcode.cn id=128 lang=python3
3 #
4 # [128] 最长连续序列
5 #
6 class Solution:
7     def longestConsecutive(self, nums: List[int]) -> int:
8         maxLen = 0
9         while nums:
10            n = nums.pop()
11            # 往大处搜索
12            i1 = n + 1
13            while i1 in nums:
14                nums.remove(i1)
15                i1 += 1
16            # 往小处搜索
17            i2 = n - 1
18            while i2 in nums:
19                nums.remove(i2)
20                i2 -= 1
```

```

21         maxLen = max(maxLen, i1 - i2 - 1)
22     return maxLen

```

```

1  #
2  # @lc app=leetcode.cn id=129 lang=python3
3  #
4  # [129] 求根到叶子节点数字之和
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def sumNumbers(self, root: TreeNode) -> int:
15         return self.sum_tree(root, 0)
16
17     def sum_tree(self, root, sum):
18         if root is None:
19             return 0
20         if root.left is None and root.right is None:
21             return sum*10+root.val
22
23         return self.sum_tree(root.left, sum*10+root.val) + self.sum_tree(root.right, sum*10+root.
            val)

```

```

1  #
2  # @lc app=leetcode.cn id=130 lang=python3
3  #
4  # [130] 被围绕的区域
5  #
6  class Solution:
7     def solve(self, board: List[List[str]]) -> None:
8         if len(board) <= 2 or len(board[0]) <= 2:
9             return
10        row, col = len(board), len(board[0])
11        # 对边界上的所有点分别进行深度遍历
12        # 第一列和最后一列
13        for i in range(row):
14            self.dfs(board, i, 0, row, col)
15            self.dfs(board, i, col-1, row, col)
16        # 第一行和最后一行
17        for j in range(1, col-1):
18            self.dfs(board, 0, j, row, col)

```

```

19         self.dfs(board,row-1,j,row,col)
20
21     for i in range(row):
22         for j in range(col):
23             if board[i][j] == "O":
24                 board[i][j] = "X"
25             if board[i][j] == "T":
26                 board[i][j] = "O"
27     return
28
29 def dfs( self ,board,i,j,row,col):
30     if i < 0 or j < 0 or i >= row or j >= col or board[i][j] != "O":
31         return
32     else :
33         board[i][j] = "T"
34         self.dfs(board,i-1,j,row,col)
35         self.dfs(board,i,j-1,row,col)
36         self.dfs(board,i+1,j,row,col)
37         self.dfs(board,i,j+1,row,col)
38     return

```

```

1  #
2  # @lc app=leetcode.cn id=131 lang=python3
3  #
4  # [131] 分割回文串
5  #
6  class Solution:
7      def partition( self , s: str ) -> List[List[str ]]:
8          res = []
9          self.dfs(s,res, [], 0)
10         return res
11
12     def dfs( self ,s, res , path, start):
13         if start == len(s):
14             res.append(path)
15             return
16         for i in range(start, len(s)):
17             if self.isPalindrome(s, start , i):
18                 self.dfs(s,res,path+[s[start:i+1]],i + 1)
19
20     # 判断回文
21     def isPalindrome(self, s, begin, end):
22         while begin < end :
23             if s[begin] != s[end]:
24                 return False
25             begin += 1
26             end -= 1

```

```
26         return True
```

```
1  #
2  # @lc app=leetcode.cn id=132 lang=python3
3  #
4  # [132] 分割回文串 II
5  #
6  class Solution:
7      def minCut(self, s: str) -> int:
8          n = len(s)
9          dp = [[False for _ in range(n)] for _ in range(n)]
10         # f[0->n](共n+1个) f[n]=-1
11         # f(i) [i, n-1]最小裁剪数
12         f = [n] *(n+1)
13         f[-1] = -1
14         # f 从右往左更新
15         # dp (i 往左更新,j往右更新)
16         for i in range(n-1,-1,-1):
17             for j in range(i,n):
18                 if (s[i] == s[j] and (j - i < 2 or dp[i + 1][j - 1])) :
19                     dp[i][j] = True
20                     # 如果满足回文的条件
21                     # f 选取裁剪更少的方案
22                     f[i] = min(f[i], f[j + 1] + 1)
23         return f[0]
```

```
1  #
2  # @lc app=leetcode.cn id=133 lang=python3
3  #
4  # [133] 克隆图
5  #
6  """
7  # Definition for a Node.
8  class Node:
9      def __init__(self, val = 0, neighbors = []):
10         self.val = val
11         self.neighbors = neighbors
12 """
13 class Solution:
14     def cloneGraph(self, node: 'Node') -> 'Node':
15         if not node:
16             return None
17         """
18         # BFS
19         queue = [node]
20         copy_node = Node(node.val)
```

```

21     visited = {node: copy_node}
22     while queue:
23         node = queue.pop(0)
24         for i in node.neighbors:
25             if i in visited:
26                 visited[node].neighbors.append(visited[i])
27             else:
28                 copy_node_ne = Node(i.val)
29                 visited[node].neighbors.append(copy_node_ne)
30                 visited[i] = copy_node_ne
31                 queue.append(i)
32
33     return copy_node
34     """
35     # DFS
36     stack = [node]
37     copy_node = Node(node.val)
38     visited = {node: copy_node}
39     while stack:
40         node = stack.pop()
41         for i in node.neighbors:
42             if i in visited:
43                 visited[node].neighbors.append(visited[i])
44             else:
45                 copy_node_ne = Node(i.val)
46                 visited[node].neighbors.append(copy_node_ne)
47                 visited[i] = copy_node_ne
48                 stack.append(i)
49
50     return copy_node

```

```

1  #
2  # @lc app=leetcode.cn id=134 lang=python3
3  #
4  # [134] 加油站
5  #
6  class Solution:
7      def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
8          sumGas = sumCost = 0
9          start = 0
10         diff = 0
11         for i in range(len(gas)):
12             sumGas += gas[i]
13             sumCost += cost[i]
14             diff += gas[i] - cost[i]
15             if diff < 0:

```

```

16         start = i + 1 ## 下一个开始
17         diff = 0
18     return start if sumGas - sumCost >= 0 else -1

```

```

1  #
2  # @lc app=leetcode.cn id=135 lang=python3
3  #
4  # [135] 分发糖果
5  #
6  class Solution:
7      def candy(self, ratings: List[int]) -> int:
8          if not ratings:
9              return 0
10         leng = len(ratings)
11         res = [1 for _ in range(leng)]
12         for i in range(1,leng):
13             # 右边大
14             if ratings[i] > ratings[i-1]:
15                 res[i] = res[i-1] + 1
16         for i in range(leng-1, 0, -1):
17             # 左边大
18             if ratings[i-1] > ratings[i]:
19                 res[i-1] = max(res[i]+1, res[i-1])
20         return sum(res)

```

```

1  #
2  # @lc app=leetcode.cn id=136 lang=python3
3  #
4  # [136] 只出现一次的数字
5  #
6  class Solution:
7      def singleNumber(self, nums: List[int]) -> int:
8          """
9          return 2*sum(set(nums)) - sum(nums)
10         """
11         res = 0
12         for i in range(len(nums)):
13             res = res ^ nums[i]
14         return res

```

```

1  #
2  # @lc app=leetcode.cn id=137 lang=python3
3  #
4  # [137] 只出现一次的数字 II
5  #
6  class Solution:
7      def singleNumber(self, nums: List[int]) -> int:

```

```
8 return (3 * sum(set(nums)) - sum(nums)) // 2
```

```
1 #
2 # @lc app=leetcode.cn id=138 lang=python3
3 #
4 # [138] 复制带随机指针的链表
5 #
6 """
7 # Definition for a Node.
8 class Node:
9     def __init__(self, x: int, next: 'Node' = None, random: 'Node' = None):
10         self.val = int(x)
11         self.next = next
12         self.random = random
13 """
14 class Solution:
15     def copyRandomList(self, head: 'Node') -> 'Node':
16         if head is None:
17             return None
18         # 复制next部分
19         headcopy = head
20         while headcopy:
21             node = Node(headcopy.val)
22             node.next = headcopy.next
23             headcopy.next = node
24             headcopy = node.next
25         # 复制random部分
26         headcopy = head
27         while headcopy:
28             if headcopy.random:
29                 headcopy.next.random = headcopy.random.next
30             headcopy = headcopy.next.next
31
32         # 拆分两个单链表
33         src = head
34         pnew = res = head.next
35
36         while pnew.next:
37             src.next = pnew.next
38             src = src.next
39             pnew.next = src.next
40             pnew = pnew.next
41         src.next = None
42         pnew.next = None
43
44         return res
```



```

1 #
2 # @lc app=leetcode.cn id=139 lang=python3
3 #
4 # [139] 单词拆分
5 #
6 class Solution:
7     def wordBreak(self, s: str, wordDict: List[str]) -> bool:
8         n = len(s)
9         dp = [False for _ in range(n+1)]
10        dp[0] = True
11
12        for i in range(n+1):
13            for j in range(i-1,-1,-1):
14                if dp[j] and s[j:i] in wordDict:
15                    dp[i] = True
16                    break
17
18        return dp[-1]

```

```

1 #
2 # @lc app=leetcode.cn id=140 lang=python3
3 #
4 # [140] 单词拆分 II
5 #
6 class Solution:
7     def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
8         n = len(s)
9         dp = [False for _ in range(n+1)]
10        dp[0] = True
11        # prev true 表示s[j,i)是一个合法单词,从j处切开
12        prev = [[False for _ in range(n)] for _ in range(n+1)]
13
14        for i in range(n+1):
15            for j in range(i-1,-1,-1):
16                if dp[j] and s[j:i] in wordDict:
17                    dp[i] = True
18                    prev[i][j] = True
19
20        res = []
21        self.dfs(s,prev,n,[], res)
22        return res
23
24    def dfs(self,s,prev,cur,path,res):
25        if cur == 0:
26            # 终止条件
27            temp = "".join(list(reversed(path)))

```

```

28         res.append(temp)
29         return
30
31     for i in range(cur-1,-1,-1):
32         if prev[cur][i]:
33             self.dfs(s,prev,i,path+[s[i:cur]], res)

```

```

1  #
2  # @lc app=leetcode.cn id=141 lang=python3
3  #
4  # [141] 环形链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def hasCycle(self, head: ListNode) -> bool:
14         """
15         try:
16             slow = head
17             fast = head.next
18             while slow is not fast:
19                 slow = slow.next
20                 fast = fast.next.next
21             return True
22         except:
23             return False
24         """
25         fast = slow = head
26         while fast and fast.next:
27             fast = fast.next.next
28             slow = slow.next
29             if slow == fast:
30                 return True
31         return False

```

```

1  #
2  # @lc app=leetcode.cn id=142 lang=python3
3  #
4  # [142] 环形链表 II
5  #
6  # Definition for singly-linked list.
7  # class ListNode:

```

```

8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def detectCycle(self, head: ListNode) -> ListNode:
14         fast = slow = head
15         while fast and fast.next:
16             slow = slow.next
17             fast = fast.next.next
18             if slow == fast:
19                 #相遇了
20                 res = head
21                 while res != slow:
22                     slow = slow.next
23                     res = res.next
24                 return res
25         return None

```

```

1  #
2  # @lc app=leetcode.cn id=143 lang=python3
3  #
4  # [143] 重排链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def reorderList(self, head: ListNode) -> None:
14         if head is None or head.next is None:
15             return head
16         p1, p2 = head, head
17         while p2 and p2.next:
18             p1 = p1.next
19             p2 = p2.next.next
20         # head2 是后面半部分
21         head2 = p1.next
22         p1.next = None
23         # head head2 对应前后两部分
24
25         cur = head2
26         rever = None
27         # 反转

```

```

28     while cur:
29         temp = cur.next
30         cur.next = rever
31         rever = cur
32         cur = temp
33
34     # head rever 两个合并
35     p1 = head
36     while rever:
37         # 两个链的下一个
38         temp = p1.next
39         temp2 = rever.next
40         # 链接好
41         p1.next = rever
42         rever.next = temp
43         # 下一个循环
44         p1 = temp
45         rever = temp2
46     return head

```

```

1  #
2  # @lc app=leetcode.cn id=144 lang=python3
3  #
4  # [144] 二叉树的前序遍历
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10     #         self.left = None
11     #         self.right = None
12
13 class Solution:
14     def preorderTraversal(self, root: TreeNode) -> List[int]:
15         if root is None:
16             return []
17         result = []
18         stack = []
19         stack.append(root)
20
21         while stack:
22             p = stack.pop()
23             result.append(p.val)
24             if p.right:
25                 stack.append(p.right)
26             if p.left:

```

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

```
1  #
2  # @lc app=leetcode.cn id=145 lang=python3
3  #
4  # [145] 二叉树的后序遍历
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def postorderTraversal(self, root: TreeNode) -> List[int]:
15         if root is None:
16             return []
17         result = []
18         stack = []
19         stack.append(root)
20         while stack:
21             p = stack.pop()
22             result.append(p.val)
23             if p.left:
24                 stack.append(p.left)
25             if p.right:
26                 stack.append(p.right)
27         return result[::-1]
```

```
1  #
2  # @lc app=leetcode.cn id=146 lang=python3
3  #
4  # [146] LRU缓存机制
5  #
6  class LRUCache:
7     def __init__(self, capacity: int):
8         self.capacity = capacity
9         self.cache = {}
10        self.queue = []
11
12    def update(self, key):
13        # 移到头部去
14        self.queue.remove(key)
15        self.queue.insert(0, key)
```

```

16
17 def get( self , key: int ) -> int:
18     if key in self .cache:
19         self .update(key)
20         return self .cache[key]
21     else :
22         return -1
23
24 def put(self , key: int , value: int ) -> None:
25     if not key or not value:
26         return None
27     if key in self .cache: # 已经在了
28         self .queue.remove(key)
29     elif len( self .queue) == self .capacity: # 满了
30         del self .cache[ self .queue.pop()]
31
32     self .cache[key] = value
33     self .queue.insert (0,key)
34
35 # Your LRUCache object will be instantiated and called as such:
36 # obj = LRUCache(capacity)
37 # param_1 = obj.get(key)
38 # obj.put(key,value)

```

```

1 #
2 # @lc app=leetcode.cn id=147 lang=python3
3 #
4 # [147] 对链表进行插入排序
5 #
6 # Definition for singly-linked list .
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def insertionSortList ( self , head: ListNode) -> ListNode:
14         dummy = ListNode(-1000)
15         dummy.next = head
16         p = dummy
17         cur = head
18         while cur and cur.next:
19             val = cur.next.val
20             # 顺序的
21             if cur.val < val:
22                 cur = cur.next

```

```

23         continue
24     # 找到p(小于的最后一个节点)
25     # 这个相当于p重新初始化
26     if p.next.val > val:
27         p = dummy
28     while p.next.val < val:
29         p = p.next
30     # 右边的节点插入到左边去
31     next_step = cur.next
32     cur.next = cur.next.next
33     next_step.next = p.next
34     p.next = next_step
35     return dummy.next

```

```

1  #
2  # @lc app=leetcode.cn id=148 lang=python3
3  #
4  # [148] 排序链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def sortList(self, head: ListNode) -> ListNode:
14         if head is None or head.next is None:
15             return head
16         fast = slow = head
17         pre = None
18         while fast and fast.next:
19             fast = fast.next.next
20             pre = slow
21             slow = slow.next
22         pre.next = None
23         return self.mergeTwoLists(self.sortList(head), self.sortList(slow))
24
25     def mergeTwoLists(self, l1, l2):
26         res = now = ListNode(-1000)
27         while l1 and l2:
28             if l1.val <= l2.val:
29                 now.next = l1
30                 l1 = l1.next
31             else:
32                 now.next = l2

```

```

33         l2 = l2.next
34         now = now.next
35         now.next = l1 or l2
36         return res.next

```

```

1  #
2  # @lc app=leetcode.cn id=149 lang=python3
3  #
4  # [149] 直线上最多的点数
5  #
6  class Solution:
7      def maxPoints(self, points: List[List[int]]) -> int:
8          if points is None :
9              return 0
10         res = 0
11         # 两重循环
12         # 双重字典
13         for i in range(len(points)):
14             line_map = {}
15             same = max_point_num = 0
16             for j in range(i + 1, len(points)):
17                 dx, dy = points[j][0] - points[i][0], points[j][1] - points[i][1]
18                 # 同一个点
19                 if dx == 0 and dy == 0:
20                     same += 1
21                     continue
22                 # 去除最大公约数部分
23                 gcd = self.generateGCD(dx, dy)
24                 if gcd != 0:
25                     dx //= gcd
26                     dy //= gcd
27
28                 if dx in line_map:
29                     if dy in line_map[dx]:
30                         line_map[dx][dy] += 1
31                     else:
32                         line_map[dx][dy] = 1
33                 else:
34                     line_map[dx] = {}
35                     line_map[dx][dy] = 1
36                 max_point_num = max(max_point_num, line_map[dx][dy])
37             res = max(res, max_point_num + same + 1)
38         return res
39
40     # 辗转相除法求最大公约数
41     def generateGCD(self, x, y):

```



```

42     if y == 0:
43         return x
44     else:
45         return self.generateGCD(y, x % y)

```

```

1  #
2  # @lc app=leetcode.cn id=150 lang=python3
3  #
4  # [150] 逆波兰表达式求值
5  #
6  class Solution:
7      def evalRPN(self, tokens: List[str]) -> int:
8          nums = []
9          for t in tokens:
10             if t not in ['+', '-', '*', '/']:
11                 nums.append(int(t))
12             else:
13                 r = nums.pop()
14                 l = nums.pop()
15                 if t == '+':
16                     temp = l+r
17                 elif t == '-':
18                     temp = l-r
19                 elif t == '*':
20                     temp = l*r
21                 elif t == '/':
22                     if l*r < 0 and l%r != 0:
23                         temp = l//r + 1
24                     else:
25                         temp = l//r
26                 nums.append(temp)
27         return nums.pop()

```

```

1  #
2  # @lc app=leetcode.cn id=151 lang=python3
3  #
4  # [151] 翻转字符串里的单词
5  #
6  class Solution:
7      def reverseWords(self, s: str) -> str:
8          if not s:
9              return s
10
11         """
12         temp = s.split(' ')
13         temp = [t for t in temp if len(t) > 0]

```

```

14     temp.reverse()
15     return ' '.join(temp)
16     """
17     s = s + " "
18     l = 0
19     res = []
20     for i in range(1, len(s)):
21         if s[i] == " ":
22             if l != i:
23                 res.append(s[l:i])
24                 l = i + 1
25
26     res.reverse()
27     return " ".join(res)

```

```

1  #
2  # @lc app=leetcode.cn id=152 lang=python3
3  #
4  # [152] 乘积最大子序列
5  #
6  class Solution:
7      def maxProduct(self, nums: List[int]) -> int:
8          if not nums:
9              return 0
10         maxtmp = mintmp = res = nums[0]
11         for i in range(1, len(nums)):
12             maxtmp, mintmp = max(nums[i], nums[i]*maxtmp, nums[i]*mintmp), \
13                             min(nums[i], nums[i]*maxtmp, nums[i]*mintmp)
14         res = max(maxtmp, res)
15     return res

```

```

1  #
2  # @lc app=leetcode.cn id=153 lang=python3
3  #
4  # [153] 寻找旋转排序数组中的最小值
5  #
6  class Solution:
7      def findMin(self, nums: List[int]) -> int:
8          if len(nums) == 1 or nums[0] < nums[-1]: # 升序
9              return nums[0]
10         l, r = 0, len(nums)-1
11         while l < r:
12             mid = (l+r)//2
13             # 左边
14             if nums[0] <= nums[mid]:
15                 l = mid + 1

```

```

16         # 在右边
17     else :
18         r = mid
19     return nums[l]

```

```

1  #
2  # @lc app=leetcode.cn id=154 lang=python3
3  #
4  # [154] 寻找旋转排序数组中的最小值 II
5  #
6
7  class Solution:
8      def findMin(self, nums: List[int]) -> int:
9          if len(nums) == 1 or nums[0] < nums[-1]: # 升序
10             return nums[0]
11
12         l , r = 0, len(nums)-1
13         while l < r :
14             mid = (l+r)//2
15             # 左边
16             if nums[mid] > nums[r]:
17                 l = mid + 1
18             # 在右边
19             elif nums[mid] < nums[r]:
20                 r = mid
21             # nums[mid] == nums[r]情况
22             else :
23                 r -= 1
24         return nums[l]

```

```

1  #
2  # @lc app=leetcode.cn id=155 lang=python3
3  #
4  # [155] 最小栈
5  #
6  class MinStack:
7      def __init__(self):
8          self.stack = []
9          self.min_stack = []
10
11      def push(self, x: int) -> None:
12          self.stack.append(x)
13          if len(self.min_stack) == 0:
14              self.min_stack.append(x)
15          return
16          # x 和栈尾 哪个小压哪个

```

```

17         if x <= self.min_stack[-1]:
18             self.min_stack.append(x)
19         else:
20             self.min_stack.append(self.min_stack[-1])
21
22     def pop(self) -> None:
23         if len(self.stack)>0:
24             self.min_stack.pop()
25             self.stack.pop()
26
27     def top(self) -> int:
28         if len(self.stack)>0:
29             return self.stack[-1]
30         return None
31
32     def getMin(self) -> int:
33         if len(self.min_stack)>0:
34             return self.min_stack[-1]
35         return None
36
37 # Your MinStack object will be instantiated and called as such:
38 # obj = MinStack()
39 # obj.push(x)
40 # obj.pop()
41 # param_3 = obj.top()
42 # param_4 = obj.getMin()

```

```

1 #
2 # @lc app=leetcode.cn id=160 lang=python3
3 #
4 # [160] 相交链表
5 #
6 # Definition for singly-linked list.
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
14         p1, p2 = headA, headB
15         # 初始化两个运动结点p1和p2
16         while p1 != p2:
17             # 只要两个结点还未相遇
18             p1 = headB if p1 is None else p1.next
19             # 如果p1走到了链表A的末尾，则换到链表B上

```

```

20         p2 = headA if p2 is None else p2.next
21         # 如果p2走到了链表B的末尾，则换到链表A上
22
23     return p1
24     # 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

```

```

1  #
2  # @lc app=leetcode.cn id=162 lang=python3
3  #
4  # [162] 寻找峰值
5  #
6
7  class Solution:
8      def findPeakElement(self, nums: List[int]) -> int:
9          n = len(nums)
10         if n == 1:
11             return 0
12
13         l, r = 0, len(nums) - 1
14         while l <= r:
15             mid = (l + r) // 2
16             if (mid == 0 or nums[mid] > nums[mid - 1]) and (mid == n - 1 or nums[mid] > nums[
17                 mid + 1]):
18                 return mid
19             elif mid > 0 and nums[mid - 1] > nums[mid]:
20                 r = mid - 1
21             else:
22                 l = mid + 1

```

```

1  #
2  # @lc app=leetcode.cn id=165 lang=python3
3  #
4  # [165] 比较版本号
5  #
6  class Solution:
7      def compareVersion(self, version1: str, version2: str) -> int:
8          vs1 = version1.split('.')
9          vs2 = version2.split('.')
10         l1, l2 = len(vs1), len(vs2)
11         if l1 > l2:
12             vs2 += [0] * (l1 - l2)
13         elif l1 < l2:
14             vs1 += [0] * (l2 - l1)
15         n = max(l1, l2)
16         for i in range(n):
17             if int(vs1[i]) > int(vs2[i]):

```

```

18         return 1
19     elif int(vs1[i]) < int(vs2[i]):
20         return -1
21     return 0

```

```

1  #
2  # @lc app=leetcode.cn id=167 lang=python3
3  #
4  # [167] 两数之和 II - 输入有序数组
5  #
6  class Solution:
7      def twoSum(self, numbers: List[int], target: int) -> List[int]:
8          l = 0
9          r = len(numbers)-1
10         while l <= r:
11             temp = numbers[l]+numbers[r]
12             if temp == target:
13                 return [l+1, r+1]
14             elif temp < target :
15                 l += 1
16             elif temp > target:
17                 r -= 1

```

```

1  #
2  # @lc app=leetcode.cn id=168 lang=python3
3  #
4  # [168] Excel表列名称
5  #
6  class Solution:
7      def convertToTitle(self, n: int) -> str:
8          capitals = [chr(x) for x in range(ord('A'), ord('Z')+1)]
9          result = []
10
11         while n > 0:
12             result.append(capitals[(n-1)%26])
13             n = (n-1) // 26
14         result.reverse()
15         return ''.join(result)

```

```

1  #
2  # @lc app=leetcode.cn id=169 lang=python3
3  #
4  # [169] 多数元素
5  #
6  class Solution:
7      def majorityElement(self, nums: List[int]) -> int:
8          nums.sort()

```

```

9     return nums[len(nums)//2]

1  #
2  # @lc app=leetcode.cn id=171 lang=python3
3  #
4  # [171] Excel表列序号
5  #
6  class Solution:
7      def titleToNumber(self, s: str) -> int:
8          res = 0
9          for i in s:
10             res = res*26 + ord(i)-ord('A')+1
11     return res

```

```

1  #
2  # @lc app=leetcode.cn id=172 lang=python3
3  #
4  # [172] 阶乘后的零
5  #
6  class Solution:
7      def trailingZeroes(self, n: int) -> int:
8          count = 0
9          while n > 0:
10             n //= 5
11             count += n
12     return count

```

```

1  #
2  # @lc app=leetcode.cn id=173 lang=python3
3  #
4  # [173] 二叉搜索树迭代器
5  #
6
7  # Definition for a binary tree node.
8  # class TreeNode:
9  #     def __init__(self, x):
10 #         self.val = x
11 #         self.left = None
12 #         self.right = None
13
14 class BSTIterator:
15     def __init__(self, root: TreeNode):
16         # 包含按排序顺序的所有节点的数组
17         self.nodes_sorted = []
18         self.index = -1
19         self._inorder(root)
20

```

```

21 def __inorder(self, root):
22     if not root:
23         return
24     self.__inorder(root.left)
25     self.nodes__sorted.append(root.val)
26     self.__inorder(root.right)
27
28 def next(self) -> int:
29     """
30     @return the next smallest number
31     """
32     self.index += 1
33     return self.nodes__sorted[self.index]
34
35 def hasNext(self) -> bool:
36     """
37     @return whether we have a next smallest number
38     """
39     return self.index + 1 < len(self.nodes__sorted)
40
41 # Your BSTIterator object will be instantiated and called as such:
42 # obj = BSTIterator(root)
43 # param_1 = obj.next()
44 # param_2 = obj.hasNext()

```

```

1 #
2 # @lc app=leetcode.cn id=174 lang=python3
3 #
4 # [174] 地下城游戏
5 #
6 class Solution:
7     def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
8         m,n = len(dungeon),len(dungeon[0])
9         res = [[0 for _ in range(n)] for _ in range(m)]
10
11         # 逆序遍历
12         # 逆序初始化
13         res[m-1][n-1] = max(-dungeon[m-1][n-1],0)+1
14         for r in range(m-2,-1,-1):
15             res[r][n-1] = max(res[r+1][n-1] -dungeon[r][n-1] ,1)
16         for c in range(n-2,-1,-1):
17             res[m-1][c] = max(res[m-1][c+1] -dungeon[m-1][c] ,1)
18         # 从下往上从右往左遍历
19         for r in range(m-2,-1,-1):
20             for c in range(n-2,-1,-1):
21                 res[r][c] = max(

```



```

22         min(res[r][c+1],res[r+1][c]) - dungeon[r][c],
23         1)
24     return res[0][0]

```

```

1  #
2  # @lc app=leetcode.cn id=179 lang=python3
3  #
4  # [179] 最大数
5  #
6  # 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: "<
   " 、 ">"
7  class LargerNumKey(str):
8      def __lt__(x, y):
9          return x+y < y+x
10
11 class Solution:
12     def largestNumber(self, nums: List[int]) -> str:
13         """
14         if set(nums) == {0}:
15             return '0'
16         str_nums = sorted([str(i) for i in nums], key=LargerNumKey,reverse = True)
17         largest = "".join(str_nums)
18         return largest
19         """
20
21         if set(nums) == {0}:
22             return '0'
23         for i in range(len(nums), 0, -1):
24             tmp = 0
25             for j in range(i):
26                 if not self.compare(nums[j], nums[tmp]):
27                     tmp = j
28             nums[tmp], nums[i-1] = nums[i-1], nums[tmp]
29         return "".join(map(str, nums))
30
31     def compare(self, n1, n2):
32         return str(n1) + str(n2) > str(n2) + str(n1)

```

```

1  #
2  # @lc app=leetcode.cn id=187 lang=python3
3  #
4  # [187] 重复的DNA序列
5  #
6  class Solution:
7      def findRepeatedDnaSequences(self, s: str) -> List[str]:
8          dic, res = {}, set()

```

```

9     for i in range(len(s)-9):
10         dic[s[i:i+10]] = dic.get(s[i:i+10], 0)+1
11         if dic[s[i:i+10]] > 1:
12             res.add(s[i:i+10])
13     return list(res)

```

```

1  #
2  # @lc app=leetcode.cn id=188 lang=python3
3  #
4  # [188] 买卖股票的最佳时机IV
5  #
6  class Solution:
7      def maxProfit(self, k: int, prices: List[int]) -> int:
8          #交易次数太多，用贪心
9          if k >= len(prices)//2:
10             return self.greedy(prices)
11
12         # k=0的时候此时sell为空
13         # k小，动态规划
14         buy, sell = [-prices[0]]*k, [0]*(k+1)
15         for p in prices[1:]:
16             for i in range(k):
17                 # 买的收益 = max(买、买了再买)
18                 buy[i] = max(buy[i], sell[i-1]-p)
19                 # 卖的收益 = (卖/买)
20                 sell[i] = max(sell[i], buy[i]+p)
21
22         return max(sell)
23
24     def greedy(self, prices):
25         res = 0
26         for i in range(1, len(prices)):
27             if prices[i] > prices[i-1]:
28                 res += prices[i] - prices[i-1]
29         return res

```

```

1  #
2  # @lc app=leetcode.cn id=189 lang=python3
3  #
4  # [189] 旋转数组
5  #
6  class Solution:
7      def rotate(self, nums: List[int], k: int) -> None:
8          tmp = [0] * len(nums)
9          for i in range(len(nums)):
10             tmp[(i+k)%len(nums)] = nums[i] #recycle

```

```
11
12     for i in range(len(nums)):
13         nums[i] = tmp[i]
```

```
1 #
2 # @lc app=leetcode.cn id=190 lang=python3
3 #
4 # [190] 颠倒二进制位
5 #
6 class Solution:
7     def reverseBits(self, n: int) -> int:
8         res = 0
9         bitsSize = 31
10        while bitsSize > -1 and n :
11            res += ((n%2) << bitsSize)
12            n = n >> 1
13            bitsSize -= 1
14        return res
```

```
1 #
2 # @lc app=leetcode.cn id=198 lang=python3
3 #
4 # [198] 打家劫舍
5 #
6 class Solution:
7     def rob(self, nums: List[int]) -> int:
8         if not nums :
9             return 0
10        f1 = 0
11        f2 = 0
12        for i in nums:
13            fi = max(f2+i,f1)
14            f1 ,f2 = fi ,f1
15        return f1
```

```
1 #
2 # @lc app=leetcode.cn id=199 lang=python3
3 #
4 # [199] 二叉树的右视图
5 #
6
7 # Definition for a binary tree node.
8 # class TreeNode:
9 #     def __init__(self, x):
10 #         self.val = x
11 #         self.left = None
12 #         self.right = None
```

```

13
14 class Solution:
15     def rightSideView(self, root: TreeNode) -> List[int]:
16         res = []
17         self.dfs(root, 0, res)
18         return res
19
20     def dfs(self, root, depth, res):
21         if not root:
22             return
23         if depth >= len(res):
24             res.append(0)
25         res[depth] = root.val
26         # 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
27         self.dfs(root.left, depth + 1, res)
28         self.dfs(root.right, depth + 1, res)

```

```

1 #
2 # @lc app=leetcode.cn id=200 lang=python3
3 #
4 # [200] 岛屿数量
5 #
6 class Solution:
7     def numIslands(self, grid: List[List[str]]) -> int:
8         if not grid:
9             return 0
10        m, n = len(grid), len(grid[0])
11
12        res = 0
13        for r in range(m):
14            for c in range(n):
15                if grid[r][c] == "1":
16                    res += 1
17                    self.dfs(grid, r, c, m, n)
18        return res
19
20    def dfs(self, grid, i, j, row, col):
21        # 终止条件
22        if i < 0 or j < 0 or i >= row or j >= col or grid[i][j] == "0":
23            return
24        # 合法的话置位
25        grid[i][j] = "0"
26        self.dfs(grid, i-1, j, row, col)
27        self.dfs(grid, i, j-1, row, col)
28        self.dfs(grid, i+1, j, row, col)
29        self.dfs(grid, i, j+1, row, col)

```

```

1 #
2 # @lc app=leetcode.cn id=201 lang=python3
3 #
4 # [201] 数字范围按位与
5 #
6 class Solution:
7     def rangeBitwiseAnd(self, m: int, n: int) -> int:
8         """
9         # 时间溢出
10        res = m
11        for i in range(m+1,n+1):
12            res = res & i
13            if res == 0 :
14                break
15        return res
16        """
17
18        i = 0
19        while m != n:
20            m >>= 1
21            n >>= 1
22            i += 1
23        return m << i

```

```

1 #
2 # @lc app=leetcode.cn id=202 lang=python3
3 #
4 # [202] 快乐数
5 #
6 class Solution:
7     def isHappy(self, n: int) -> bool:
8         mem = set()
9         while n != 1:
10            # 求和
11            n = sum([int(i) ** 2 for i in str(n)])
12            if n in mem:
13                # 陷入死循环了
14                return False
15            else :
16                mem.add(n)
17        else :
18            return True

```

```

1 #
2 # @lc app=leetcode.cn id=203 lang=python3
3 #

```

```

4 # [203] 移除链表元素
5 #
6
7 # Definition for singly-linked list .
8 # class ListNode:
9 #     def __init__(self, x):
10 #         self.val = x
11 #         self.next = None
12
13 class Solution:
14     def removeElements(self, head: ListNode, val: int) -> ListNode:
15         dummy = ListNode(-1)
16         dummy.next = head
17         prev, curr = dummy, head
18         while curr :
19             if curr.val == val:
20                 # prev 跟上了curr
21                 prev.next = curr.next
22             else :
23                 prev = curr
24             curr = curr.next
25         return dummy.next

```

```

1 #
2 # @lc app=leetcode.cn id=204 lang=python3
3 #
4 # [204] 计数质数
5 #
6 class Solution:
7     def countPrimes(self, n: int) -> int:
8         if n <= 2:
9             return 0
10         res = [0,0] + [1]*(n-2)
11         for i in range(2,n):
12             # 这些没改过
13             if res[i] == 1:
14                 for j in range(2,(n-1)//i+1):
15                     res[i*j] = 0
16         return sum(res)

```

```

1 #
2 # @lc app=leetcode.cn id=205 lang=python3
3 #
4 # [205] 同构字符串
5 #
6 class Solution:

```

```

7     def isIsomorphic(self, s: str, t: str) -> bool:
8         if len(s) != len(t):
9             return False
10
11         mapStoT = [0] * 128
12         mapTtoS = [0] * 128
13         for i in range(len(s)):
14             s_num, t_num = ord(s[i]), ord(t[i])
15             if mapStoT[s_num] == 0 and mapTtoS[t_num] == 0:
16                 mapStoT[s_num] = t_num
17                 mapTtoS[t_num] = s_num
18             elif mapTtoS[t_num] != s_num or mapStoT[s_num] != t_num:
19                 return False
20         return True

```

```

1  #
2  # @lc app=leetcode.cn id=206 lang=python3
3  #
4  # [206] 反转链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def reverseList(self, head: ListNode) -> ListNode:
14         """
15         if head is None or head.next is None:
16             return head
17         curr = head # 他来往后走
18         prev = None # 新的反转的
19         while curr:
20             # 下一步先保存下来
21             nextcurr = curr.next
22             # 反转的接上去
23             curr.next = prev
24             prev = curr
25             # 下一步
26             curr = nextcurr
27         return prev
28         """
29         # 递归方法
30         if not head:
31             return None

```

```

32     if not head.next:
33         return head
34     headNode = self.reverseList(head.next)
35     # head headNode 顺序(环)
36     head.next.next = head
37     # head headNode head(断开)
38     head.next = None
39     return headNode

```

```

1  #
2  # @lc app=leetcode.cn id=207 lang=python3
3  #
4  # [207] 课程表
5  #
6  class Solution:
7      def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
8          adjacency = [[] for _ in range(numCourses)]
9
10         flags = [0 for _ in range(numCourses)]
11         # (cur, pre) 对
12         for cur, pre in prerequisites:
13             adjacency[pre].append(cur)
14         for i in range(numCourses):
15             if not self.dfs(i, adjacency, flags):
16                 return False
17         return True
18
19     def dfs(self, i, adjacency, flags):
20         # flag 标志
21         # 0: 未访问
22         # 1: 已被当前节点启动的访问
23         # -1: 已被其他节点启动的访问
24         if flags[i] == -1:
25             return True
26         if flags[i] == 1:
27             return False
28         flags[i] = 1
29         for j in adjacency[i]:
30             if not self.dfs(j, adjacency, flags):
31                 return False
32         flags[i] = -1
33         return True

```

```

1  #
2  # @lc app=leetcode.cn id=209 lang=python3
3  #

```



```

4 # [209] 长度最小的子数组
5 #
6 class Solution:
7     def minSubArrayLen(self, s: int, nums: List[int]) -> int:
8         res = float('inf')
9         left = 0
10        sumval = 0
11
12        for i in range(len(nums)):
13            sumval += nums[i]
14            while sumval >= s:
15                res = min(res, i-left+1)
16                sumval -= nums[left]
17                left += 1
18
19        if res != float('inf'):
20            return res
21        else:
22            return 0

```

```

1 #
2 # @lc app=leetcode.cn id=210 lang=python3
3 #
4 # [210] 课程表 II
5 #
6 class Solution:
7     def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> List[int]:
8         if not prerequisites:
9             return [i for i in range(numCourses)]
10
11        flags = [0 for _ in range(numCourses)]
12        inverse_adj = [[] for _ in range(numCourses)]
13        for second, first in prerequisites:
14            inverse_adj[second].append(first)
15
16        res = []
17        for i in range(numCourses):
18            if self.dfs(i, inverse_adj, flags, res):
19                return []
20        return res
21
22    def dfs(self, i, inverse_adj, flags, res):
23        """
24        :param i: 结点的索引
25        :param inverse_adj: 逆邻接表，记录的是当前结点的前驱结点的集合
26        :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

```

```

27         :return: 是否有环
28         """
29         if flags[i] == 2:
30             return True
31         if flags[i] == 1:
32             return False
33
34         flags[i] = 2
35         for precursor in inverse_adj[i]:
36             if self.dfs(precursor, inverse_adj, flags, res):
37                 return True
38
39         flags[i] = 1
40         res.append(i)
41         return False

```

```

1  #
2  # @lc app=leetcode.cn id=213 lang=python3
3  #
4  # [213] 打家劫舍 II
5  #
6  class Solution:
7      def rob(self, nums: List[int]) -> int:
8          if not nums:
9              return 0
10         if len(nums) == 1:
11             return nums[0]
12         # 奇偶串
13         return max(
14             self.rob(nums[0:-1]),
15             self.rob(nums[1:])
16         )
17
18     def robb(self, nums):
19         f1 = 0
20         f2 = 0
21         for i in nums:
22             fi = max(f2+i, f1)
23             f1, f2 = fi, f1
24         return f1

```

```

1  #
2  # @lc app=leetcode.cn id=214 lang=python3
3  #
4  # [214] 最短回文串
5  #

```

```

6 class Solution:
7     def shortestPalindrome(self, s: str) -> str:
8         """
9         # 暴力法
10        r = s[::-1]
11        for i in range(len(s)) :
12            if s[0: len(s)-i ] == r[i:] :
13                return r[:i] + s
14        return ""
15        """
16        """
17        # 双指针法
18        i = 0
19        # 找到从头开始，最长的回文子串
20        for j in range(len(s) - 1, -1, -1):
21            if s[i] == s[j]:
22                i += 1
23        if i == len(s):
24            return s
25        # 后缀
26        suffix = s[i:]
27        return suffix[::-1] + self.shortestPalindrome(s[:i]) + suffix
28        """
29
30        # kmp算法
31        table = self.kmp(s + "#" + s[::-1])
32        return s[table[-1]:][::-1] + s
33
34
35    def kmp(self,p):
36        table = [0] * len(p)
37        i = 1
38        j = 0
39        while i < len(p):
40            if p[i] == p[j]:
41                j += 1
42                table[i] = j
43                i += 1
44            else :
45                if j > 0:
46                    j = table[j - 1]
47                else :
48                    i += 1
49                j = 0
50        return table

```

```

1  #
2  # @lc app=leetcode.cn id=215 lang=python3
3  #
4  # [215] 数组中的第K个最大元素
5  #
6
7  class Solution:
8      def findKthLargest(self, nums: List[int], k: int) -> int:
9          """
10             nums.sort()
11             return nums[-k]
12             """
13         return self.qSelect(nums, 0, len(nums) - 1, k)
14
15     def qSelect(self, nums, start, end, k):
16         """
17             if start > end:
18                 return float('inf')
19             """
20         # 找一个参照值
21         pivot = nums[end]
22         left = start
23         for i in range(start, end):
24             # 比参照大的都移到左边去
25             if nums[i] >= pivot:
26                 nums[left], nums[i] = nums[i], nums[left]
27                 left += 1
28         # 参照值也拉倒左边去
29         nums[left], nums[end] = nums[end], nums[left]
30         # 左边的个数够没(从0开始到k-1,共k个)
31         if left == k-1:
32             return nums[left]
33         # 还不够
34         elif left < k-1:
35             return self.qSelect(nums, left + 1, end, k)
36         # 太多了
37         else:
38             return self.qSelect(nums, start, left - 1, k)

```

```

1  #
2  # @lc app=leetcode.cn id=216 lang=python3
3  #
4  # [216] 组合总和 III
5  #
6  class Solution:
7      def combinationSum3(self, k: int, n: int) -> List[List[int]]:

```

```

8     res = []
9     self.dfs(k,n ,1,[], res)
10    return res
11
12    def dfs( self ,k,target ,start ,path,res):
13        # 终止条件
14        if target == 0 and len(path) == k:
15            res.append(path)
16            return
17        elif target < 0 or len(path) > k or start > 9 :
18            return
19
20        for i in range(start,10):
21            self.dfs(k,target-i,i+1,path+[i],res)

```

```

1 #
2 # @lc app=leetcode.cn id=217 lang=python3
3 #
4 # [217] 存在重复元素
5 #
6 class Solution:
7     def containsDuplicate(self, nums: List[int]) -> bool:
8         return len(nums) != len(set(nums))

```

```

1 #
2 # @lc app=leetcode.cn id=219 lang=python3
3 #
4 # [219] 存在重复元素 II
5 #
6 class Solution:
7     def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
8         dic = {}
9         for key ,val in enumerate(nums):
10             if val in dic and key - dic[val]<=k :
11                 return True
12             dic[val] = key
13         return False

```

```

1 #
2 # @lc app=leetcode.cn id=220 lang=python3
3 #
4 # [220] 存在重复元素 III
5 #
6 class Solution:
7     def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) -> bool:
8         if t < 0 or k < 0:
9             return False

```

```

10     all_buckets = {}
11     # 桶的大小设成t+1更加方便
12     bucket_size = t + 1
13     for i in range(len(nums)):
14         # 放入哪个桶
15         bucket_num = nums[i] // bucket_size
16         # 桶中已经有元素了
17         if bucket_num in all_buckets:
18             return True
19         # 把nums[i]放入桶中
20         all_buckets[bucket_num] = nums[i]
21         # 检查前一个桶
22         if (bucket_num - 1) in all_buckets and abs(all_buckets[bucket_num - 1] - nums[i])
            <= t:
23             return True
24         # 检查后一个桶
25         if (bucket_num + 1) in all_buckets and abs(all_buckets[bucket_num + 1] - nums[i])
            <= t:
26             return True
27
28         # 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
29         if i >= k:
30             all_buckets.pop(nums[i-k]//bucket_size)
31
32     return False

```

```

1  #
2  # @lc app=leetcode.cn id=221 lang=python3
3  #
4  # [221] 最大正方形
5  #
6  class Solution:
7      def maximalSquare(self, matrix: List[List[str]]) -> int:
8          if not matrix:
9              return 0
10         row, col = len(matrix), len(matrix[0])
11
12         # 多了一行一列
13         dp = [ [0 for _ in range(col + 1)] for _ in range(row + 1)]
14         res = 0
15         for i in range(1, row + 1):
16             for j in range(1, col + 1):
17                 if matrix[i - 1][j - 1] == "1":
18                     # 否则dp为0, 不用操作
19                     dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]) + 1

```

```

20         res = max(res, dp[i][j] ** 2)
21     return res

```

```

1  #
2  # @lc app=leetcode.cn id=222 lang=python3
3  #
4  # [222] 完全二叉树的节点个数
5  #
6
7  # Definition for a binary tree node.
8  # class TreeNode:
9  #     def __init__(self, x):
10 #         self.val = x
11 #         self.left = None
12 #         self.right = None
13
14 class Solution:
15     def countNodes(self, root: TreeNode) -> int:
16         if not root:
17             return 0
18
19         # return 1 + self.countNodes(root.left) + self.countNodes(root.right)
20
21         h_l, h_r = 0, 0
22         # 计算当前节点左子树的最大高度
23         curRoot = root
24         while curRoot.left:
25             h_l += 1
26             curRoot = curRoot.left
27         # 计算当前节点右子树的最大高度
28         curRoot = root
29         if curRoot.right:
30             h_r += 1
31             curRoot = curRoot.right
32         while curRoot.left:
33             h_r += 1
34             curRoot = curRoot.left
35
36         # 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
37         if h_l == h_r:
38             sumNodes_r = self.countNodes(root.right)
39             sumNodes_l = 2**h_l - 1
40         # 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
41         if h_l == h_r + 1:
42             sumNodes_l = self.countNodes(root.left)
43             sumNodes_r = 2**h_r - 1

```

```

44
45     # 返回左子节点个数+右子节点个数+当前根节点
46     return sumNodes_l + sumNodes_r + 1

```

```

1  #
2  # @lc app=leetcode.cn id=223 lang=python3
3  #
4  # [223] 矩形面积
5  #
6  class Solution:
7      def computeArea(self, A: int, B: int, C: int, D: int, E: int, F: int, G: int, H: int) -> int:
8          x = min( C,G )- max(A,E)
9          y = min( D,H )- max(B,F)
10         return (A-C)*(B-D) + (E-G)*(F-H) - max(x,0)*max(y,0)

```

```

1  #
2  # @lc app=leetcode.cn id=224 lang=python3
3  #
4  # [224] 基本计算器
5  #
6  class Solution:
7      def calculate( self , s: str ) -> int:
8          res = 0
9          sign = 1
10         stack = []
11         i = 0
12         while i < len(s):
13             c = s[i]
14             if c.isdigit():
15                 start = i
16                 while i < len(s) and s[i].isdigit():
17                     i += 1
18                 res += sign * int(s[start:i])
19                 # 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
20                 i -= 1
21             elif c == '+':
22                 sign = 1
23             elif c == '-':
24                 sign = -1
25             elif c == "(":
26                 stack.append(res)
27                 stack.append(sign)
28                 res = 0
29                 sign = 1
30             elif c == ")":
31                 # 现在的res是括号里面的计算结果

```



```

32         # 需要乘以对应的符号
33         res *= stack.pop()
34         res += stack.pop()
35         i += 1
36     return res

```

```

1  #
2  # @lc app=leetcode.cn id=225 lang=python3
3  #
4  # [225] 用队列实现栈
5  #
6  class MyStack:
7      def __init__(self):
8          self.list = []
9
10     def push(self, x: int) -> None:
11         # 尾部压入
12         self.list.append(x)
13
14     def pop(self) -> int:
15         # 尾部弹出
16         if len(self.list) == 0:
17             return
18         else:
19             temp = self.list[-1]
20             del self.list[-1]
21             return temp
22
23     def top(self) -> int:
24         if len(self.list) == 0:
25             return
26         else:
27             return self.list[-1]
28
29     def empty(self) -> bool:
30         return len(self.list) == 0
31
32
33     # Your MyStack object will be instantiated and called as such:
34     # obj = MyStack()
35     # obj.push(x)
36     # param_2 = obj.pop()
37     # param_3 = obj.top()
38     # param_4 = obj.empty()

```

```

1  #

```

```

2  # @lc app=leetcode.cn id=226 lang=python3
3  #
4  # [226] 翻转二叉树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def invertTree( self , root: TreeNode) -> TreeNode:
15         if not root:
16             return None
17         root.left ,root.right = self.invertTree(root.right) , self.invertTree(root.left )
18         return root

```

```

1  #
2  # @lc app=leetcode.cn id=228 lang=python3
3  #
4  # [228] 汇总区间
5  #
6  class Solution:
7     def summaryRanges(self, nums: List[int]) -> List[str]:
8         if not nums:
9             return []
10        res = []
11        i = 0
12        while i < len(nums):
13            j = i
14            while j+1<len(nums) and (nums[j + 1] - nums[j] <= 1):
15                j += 1
16
17            if i == j:
18                res.append( str(nums[i]) )
19            else :
20                res.append( str(nums[i]) + "<-->" + str(nums[j]) )
21            i = j+1
22        return res

```

```

1  #
2  # @lc app=leetcode.cn id=229 lang=python3
3  #
4  # [229] 求众数 II
5  #

```

```

6 class Solution:
7     def majorityElement(self, nums: List[int]) -> List[int]:
8         # 摩尔投票法得到两个大多数
9         result1 , result2 = -1, -1
10        score1 , score2 = 0 , 0
11        for i in range(len(nums)):
12            if (result1==nums[i]):
13                score1+=1
14            elif (result2==nums[i]):
15                score2+=1
16            elif (score1==0):
17                result1=nums[i]
18                score1=1
19            elif (score2==0):
20                result2=nums[i]
21                score2=1
22        else :
23            score1 -= 1
24            score2 -= 1
25
26        # 统计两个大多数的出现次数
27        time1,time2 = 0 , 0
28        for i in range(len(nums)):
29            if (nums[i]==result1): time1+=1
30            elif (nums[i]==result2): time2+= 1
31
32        # 得到结果
33        result = []
34        if (time1>len(nums)/3): result.append(result1)
35        if (time2>len(nums)/3): result.append(result2)
36        return result

```

```

1 #
2 # @lc app=leetcode.cn id=230 lang=python3
3 #
4 # [230] 二叉搜索树中第K小的元素
5 #
6
7 # Definition for a binary tree node.
8 # class TreeNode:
9 #     def __init__(self, x):
10 #         self.val = x
11 #         self.left = None
12 #         self.right = None
13
14 class Solution:

```

```

15 def kthSmallest(self, root: TreeNode, k: int) -> int:
16     """
17     # 方法一
18     reslist = self.inorder(root)
19     return reslist[k - 1]
20     """
21     # 方法二
22     # 左子树有多少个点
23     n = self.count(root.left)
24     if n == k - 1:
25         return root.val
26     # 递归到左子树
27     elif n > k - 1:
28         return self.kthSmallest(root.left, k)
29     # 递归到右子树
30     else:
31         return self.kthSmallest(root.right, k - 1 - n)
32
33 def inorder(self, r):
34     if r:
35         return self.inorder(r.left) + [r.val] + self.inorder(r.right)
36     else:
37         return []
38
39 def count(self, root):
40     if not root:
41         return 0
42     return self.count(root.left) + self.count(root.right) + 1

```

```

1 #
2 # @lc app=leetcode.cn id=231 lang=python3
3 #
4 # [231] 2的幂
5 #
6 class Solution:
7     def isPowerOfTwo(self, n: int) -> bool:
8         while n > 1:
9             n /= 2
10        if n == 1:
11            return True
12        else:
13            return False

```

```

1 #
2 # @lc app=leetcode.cn id=232 lang=python3
3 #

```

```

4 # [232] 用栈实现队列
5 #
6 class MyQueue:
7     def __init__(self):
8         self.stack = []
9
10    def push(self, x: int) -> None:
11        # 尾部加入
12        self.stack.append(x)
13
14    def pop(self) -> int:
15        temp = self.stack[0]
16        self.stack.pop(0)
17        return temp
18
19    def peek(self) -> int:
20        return self.stack[0]
21
22    def empty(self) -> bool:
23        return len(self.stack) == 0
24
25 # Your MyQueue object will be instantiated and called as such:
26 # obj = MyQueue()
27 # obj.push(x)
28 # param_2 = obj.pop()
29 # param_3 = obj.peek()
30 # param_4 = obj.empty()

```

```

1 #
2 # @lc app=leetcode.cn id=233 lang=python3
3 #
4 # [233] 数字 1 的个数
5 #
6 class Solution:
7     def countDigitOne(self, n: int) -> int:
8         res = 0
9         a = 1
10        b = 1
11        while n >= 1:
12            # 用(x+8)//10来判断一个数是否大于等于2
13            # 从低位到高位
14            res += (n + 8) // 10 * a
15            if n % 10 == 1:
16                res += b
17            b += n % 10 * a
18            a *= 10

```

```
19         n //= 10
20     return res
```

```
1  #
2  # @lc app=leetcode.cn id=234 lang=python3
3  #
4  # [234] 回文链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def isPalindrome(self, head: ListNode) -> bool:
14         if head is None:
15             return True
16         rev = None
17         slow = fast = head
18         # fast 到尾部
19         # slow 到中部
20         # rev 前半部分的反向
21         while fast and fast.next:
22             fast = fast.next.next
23             rev, rev.next, slow = slow, rev, slow.next
24         # 奇
25         if fast:
26             slow = slow.next
27         # 一个向左,一个向右
28         while rev:
29             if rev.val != slow.val:
30                 return False
31             slow = slow.next
32             rev = rev.next
33         return True
```

```
1  #
2  # @lc app=leetcode.cn id=235 lang=python3
3  #
4  # [235] 二叉搜索树的最近公共祖先
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
```

```

10 #         self . left  = None
11 #         self . right = None
12
13 class Solution:
14     def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
15         if p is None or q is None or root is None:
16             return None
17         if p.val < root.val and q.val < root.val:
18             return self .lowestCommonAncestor(root.left , p ,q)
19         elif p.val > root.val and q.val > root.val:
20             return self .lowestCommonAncestor(root.right , p ,q)
21         else :
22             return root

```

```

1 #
2 # @lc app=leetcode.cn id=236 lang=python3
3 #
4 # [236] 二叉树的最近公共祖先
5 #
6
7 # Definition for a binary tree node.
8 # class TreeNode:
9 #     def __init__(self, x):
10 #         self.val = x
11 #         self.left = None
12 #         self.right = None
13
14 class Solution:
15     def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
16         #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
17         if (root is None or root== p or root== q):
18             return root
19
20         left  = self .lowestCommonAncestor(root.left,p,q)
21         right = self .lowestCommonAncestor(root.right,p,q)
22
23         #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
24         if left and right :
25             return root
26         # 若左子树是none右子树不是,说明右子树找到了p或q
27         if not left :
28             return right
29         # 同理
30         if not right :
31             return left
32         return None

```

```

1 #
2 # @lc app=leetcode.cn id=237 lang=python3
3 #
4 # [237] 删除链表中的节点
5 #
6 # Definition for singly-linked list .
7 # class ListNode:
8 #     def __init__(self, x):
9 #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def deleteNode(self, node):
14         node.val = node.next.val
15         node.next = node.next.next

```

```

1 #
2 # @lc app=leetcode.cn id=238 lang=python3
3 #
4 # [238] 除自身以外数组的乘积
5 #
6 class Solution:
7     def productExceptSelf(self, nums: List[int]) -> List[int]:
8         res = [1] * len(nums)
9         right = 1
10        for i in range(1, len(nums)):
11            res[i] = res[i - 1] * nums[i - 1]
12
13        for i in range(len(nums) - 1, -1, -1):
14            res[i] *= right
15            right *= nums[i]
16        return res

```

```

1 #
2 # @lc app=leetcode.cn id=240 lang=python3
3 #
4 # [240] 搜索二维矩阵 II
5 #
6 class Solution:
7     def searchMatrix(self, matrix, target):
8         if not len(matrix) or not len(matrix[0]):
9             return False
10        # 左下角
11        r, c = len(matrix) - 1, 0
12        while r >= 0 and c < len(matrix[0]):
13            if matrix[r][c] > target :

```



```

14         # 往上
15         r -= 1
16     elif matrix[r][c] < target :
17         # 往右
18         c += 1
19     else :
20         return True
21     return False

```

```

1  #
2  # @lc app=leetcode.cn id=242 lang=python3
3  #
4  # [242] 有效的字母异位词
5  #
6  class Solution:
7      def isAnagram(self, s: str, t: str) -> bool:
8          dic1, dic2 = {}, {}
9          for item in s:
10             dic1[item] = dic1.get(item, 0) + 1
11          for item in t:
12             dic2[item] = dic2.get(item, 0) + 1
13          return dic1 == dic2

```

```

1  #
2  # @lc app=leetcode.cn id=257 lang=python3
3  #
4  # [257] 二叉树的所有路径
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10         self.left = None
11         self.right = None
12
13 class Solution:
14     def binaryTreePaths(self, root: TreeNode) -> List[str]:
15         if not root:
16             return []
17         res = []
18         self.dfs(root, [], res)
19         paths = ['->'.join(path) for path in res]
20         return paths
21
22     def dfs(self, node, path, res):
23         # 终止条件 没有子节点

```

```

24         if not node.left and not node.right:
25             res.append(path+[str(node.val)])
26             return
27         path = path + [str(node.val)]
28         if node.left:
29             self.dfs(node.left , path , res )
30         if node.right:
31             self.dfs(node.right , path, res )

```

```

1  #
2  # @lc app=leetcode.cn id=258 lang=python3
3  #
4  # [258] 各位相加
5  #
6  class Solution:
7      def addDigits(self, num: int) -> int:
8          t = num
9          while t >= 10 :
10             t = sum([int(char) for char in str(t)])
11         return t

```

```

1  #
2  # @lc app=leetcode.cn id=260 lang=python3
3  #
4  # [260] 只出现一次的数字 III
5  #
6  class Solution:
7      def singleNumber(self, nums: List[int]) -> List[int]:
8          if not nums:
9              return []
10         # 异或的结果
11         diff = 0
12         # 得到  $x^y$ 
13         for num in nums:
14             diff ^= num
15         # 区分x和y,  $n \& (-n)$  得到x和y不同的最低位
16         diff &= -diff
17         res = [0, 0]
18         for num in nums:
19             # 除了x外, 其他 $\&=0$ 的数成对出现
20             if num & diff:
21                 res[0] ^= num
22             # 除了y外, 其他 $\&=1$ 的数成对出现
23             else:
24                 res[1] ^= num
25         return res

```

```

1 #
2 # @lc app=leetcode.cn id=263 lang=python3
3 #
4 # [263] 丑数
5 #
6 class Solution:
7     def isUgly(self, num: int) -> bool:
8         if num <= 0 :
9             return False
10
11         divisors = [2, 3, 5]
12         for d in divisors :
13             while num % d == 0:
14                 num /= d
15         return num == 1

```

```

1 #
2 # @lc app=leetcode.cn id=264 lang=python3
3 #
4 # [264] 丑数 II
5 #
6 class Solution:
7     def nthUglyNumber(self, n: int) -> int:
8         ugly = [1]
9         i2,i3,i5 = 0,0,0
10        idx = 1
11        while idx < n :
12            newugly = min([ugly[i2]*2 , ugly[i3]*3 ,ugly[i5]*5])
13            ugly.append(newugly)
14
15            while ugly[i2]*2<= newugly:
16                i2 += 1
17            while ugly[i3]*3<= newugly:
18                i3 += 1
19            while ugly[i5]*5<= newugly:
20                i5 += 1
21            idx += 1
22        return ugly[-1]

```

```

1 #
2 # @lc app=leetcode.cn id=268 lang=python3
3 #
4 # [268] 缺失数字
5 #
6 class Solution:
7     def missingNumber(self, nums: List[int]) -> int:

```

```
8     return len(nums)*(len(nums)+1)//2 - sum(nums)
```

```
1 #
2 # @lc app=leetcode.cn id=274 lang=python3
3 #
4 # [274] H指数
5 #
6 class Solution:
7     def hIndex(self, citations: List[int]) -> int:
8         citations.sort()
9         i = 0
10        while i<len(citations) and citations[len(citations)-1-i]>i:
11            i += 1
12        return i
```

```
1 #
2 # @lc app=leetcode.cn id=275 lang=python3
3 #
4 # [275] H指数 II
5 #
6 class Solution:
7     def hIndex(self, citations: List[int]) -> int:
8         i = 0
9         while i<len(citations) and citations[len(citations)-1-i]>i:
10            i += 1
11        return i
```

```
1 #
2 # @lc app=leetcode.cn id=278 lang=python3
3 #
4 # [278] 第一个错误的版本
5 #
6 # The isBadVersion API is already defined for you.
7 # @param version, an integer
8 # @return a bool
9 # def isBadVersion(version):
10
11 class Solution:
12     def firstBadVersion(self, n):
13         l, r = 0, n-1
14         while l <= r:
15             mid = (l+r)//2
16             if isBadVersion(0) == isBadVersion(mid):
17                 l = mid + 1
18             elif isBadVersion(n) == isBadVersion(mid):
19                 r = mid - 1
20         return l
```

```

1  #
2  # @lc app=leetcode.cn id=279 lang=python3
3  #
4  # [279] 完全平方数
5  #
6  class Solution:
7      def numSquares(self, n: int) -> int:
8          dp = list(range(n+1))
9          for i in range(2,n+1):
10             for j in range(1,int(i**(0.5))+1):
11                 dp[i]=min(dp[i],dp[i-j*j]+1)
12         return dp[-1]

```

```

1  #
2  # @lc app=leetcode.cn id=283 lang=python3
3  #
4  # [283] 移动零
5  #
6  class Solution:
7      def moveZeroes(self, nums: List[int]) -> None:
8          """
9          zeros = []
10         for i in range(len(nums)):
11             if nums[i] == 0 :
12                 zeros.append(i)
13
14         for i in zeros[::-1]:
15             nums.pop(i)
16             nums.append(0)
17         return nums
18         """
19         j = 0
20         for i in range(len(nums)):
21             if nums[i] != 0:
22                 nums[j] = nums[i]
23                 j += 1
24         for i in range(j,len(nums)):
25             nums[i] = 0

```

```

1  #
2  # @lc app=leetcode.cn id=287 lang=python3
3  #
4  # [287] 寻找重复数
5  #
6  class Solution:

```

```

7 def findDuplicate(self, nums: List[int]) -> int:
8     l, r = 0, len(nums) - 1
9     while l < r:
10         mid = (l + r) // 2
11         cnt = 0
12         for num in nums:
13             if num <= mid:
14                 cnt += 1
15
16         if cnt > mid:
17             r = mid
18         else:
19             l = mid + 1
20     return l

```

```

1 #
2 # @lc app=leetcode.cn id=289 lang=python3
3 #
4 # [289] 生命游戏
5 #
6 class Solution:
7     def gameOfLife(self, board: List[List[int]]) -> None:
8         """
9         # 卷积的思想
10        import numpy as np
11        r, c = len(board), len(board[0])
12        # 下面两行做 zero padding
13        board_exp = np.zeros((r + 2, c + 2))
14        board_exp[1:-1, 1:-1] = np.array(board)
15        # 设置卷积核
16        kernel = np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1]])
17        # 开始卷积
18        for i in range(1, r + 1):
19            for j in range(1, c + 1):
20                # 统计细胞周围8个位置的状态
21                temp_sum = np.sum(kernel * board_exp[i - 1:i + 2, j - 1:j + 2])
22                # 按照题目规则进行判断
23                if board_exp[i, j] == 1:
24                    if temp_sum < 2 or temp_sum > 3:
25                        board[i - 1][j - 1] = 0
26                else:
27                    if temp_sum == 3:
28                        board[i - 1][j - 1] = 1
29        return
30        """
31

```

```

32     """
33     方法二:两次遍历
34     第一次遍历时也是分两种情况:
35         若活细胞变成了死细胞,由1->-1
36         若死细胞变成了活细胞,由0->2
37     第二次遍历则是将2(活)->1, -1(死)->0
38     """
39     row_len, col_len = len(board), len(board[0])
40     for row in range(row_len):
41         for col in range(col_len):
42             lives = self.count(board, row, col, row_len, col_len)
43             if board[row][col] == 1:
44                 if lives < 2 or lives > 3:
45                     board[row][col] = -1
46             else:
47                 if lives == 3:
48                     board[row][col] = 2
49     # 第二次遍历,恢复更改的值
50     for row in range(row_len):
51         for col in range(col_len):
52             if board[row][col] == 2:
53                 board[row][col] = 1
54             elif board[row][col] == -1:
55                 board[row][col] = 0
56     return
57
58     def count(self, board, row, col, row_len, col_len):
59         lives = 0
60         start_row, end_row = max(0, row - 1), min(row_len - 1, row + 1)
61         start_col, end_col = max(0, col - 1), min(col_len - 1, col + 1)
62         for r in range(start_row, end_row + 1):
63             for c in range(start_col, end_col + 1):
64                 if board[r][c] in [-1, 1] and not (r == row and c == col):
65                     lives += 1
66     return lives

```

```

1  #
2  # @lc app=leetcode.cn id=290 lang=python3
3  #
4  # [290] 单词规律
5  #
6  class Solution:
7      def wordPattern(self, pattern: str, str: str) -> bool:
8          """
9          word_list = str.split(' ')
10         pattern_list = list(pattern)

```

```

11     if len(word_list) != len(pattern_list):
12         return False
13     for i, word in enumerate(word_list):
14         idx = word_list.index(word)
15         idx2 = pattern_list.index(pattern[i])
16         if idx != idx2:
17             return False
18     return True
19     """
20
21     words = str.split(" ")
22     hash_table_pattern = {}
23     hash_table_words = {}
24
25     if len(words) != len(pattern):
26         return False
27     #第一步
28     for i, letter in enumerate(pattern):
29         if letter in hash_table_pattern:
30             if hash_table_pattern[letter] != words[i]:
31                 return False
32         else:
33             hash_table_pattern[letter] = words[i]
34     #第二步
35     for i, word in enumerate(words):
36         if word in hash_table_words:
37             if hash_table_words[word] != pattern[i]:
38                 return False
39         else:
40             hash_table_words[word] = pattern[i]
41     return True

```

```

1  #
2  # @lc app=leetcode.cn id=292 lang=python3
3  #
4  # [292] Nim 游戏
5  #
6  class Solution:
7      def canWinNim(self, n: int) -> bool:
8          return n%4 != 0

```

```

1  #
2  # @lc app=leetcode.cn id=299 lang=python3
3  #
4  # [299] 猜数字游戏
5  #

```



```

6 class Solution:
7     def getHint(self, secret: str, guess: str) -> str:
8         a = b = 0
9         dic = {}
10        for i in range(len(secret)):
11            if secret[i] == guess[i]:
12                a += 1
13            dic[secret[i]] = dic.get(secret[i],0) + 1
14        for i in range(len(guess)):
15            if guess[i] in dic and dic[guess[i]] > 0:
16                b += 1
17                dic[guess[i]] -= 1
18        b -= a
19        return f"{a}A{b}B"

```

```

1 #
2 # @lc app=leetcode.cn id=300 lang=python3
3 #
4 # [300] 最长上升子序列
5 #
6 class Solution:
7     def lengthOfLIS(self, nums: List[int]) -> int:
8         if not nums:
9             return 0
10
11        """
12        dp = [1] * len(nums)
13        for i in range(1,len(nums)):
14            for j in range(i):
15                # 如果要求非严格递增,将此行 '<' 改为 '<=' 即可
16                if (nums[j] < nums[i]):
17                    dp[i] = max(dp[i], dp[j] + 1)
18        return max(dp)
19        """
20
21        up_list = []
22        for i in range(len(nums)):
23            # 二分查找
24            left, right = 0, len(up_list)-1
25            while left <= right:
26                mid = (left+right)//2
27                if up_list[mid] < nums[i]:
28                    left = mid+1
29                else:
30                    right = mid-1
31            # 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

```

```

32         if left == len(up_list):
33             up_list.append(nums[i])
34         else:
35             up_list[ left ] = nums[i]
36     return len(up_list)

```

```

1  #
2  # @lc app=leetcode.cn id=303 lang=python3
3  #
4  # [303] 区域和检索 - 数组不可变
5  #
6  class NumArray:
7
8      def __init__(self, nums: List[int]):
9          self.list = [0] *(len(nums)+1)
10         for i in range(len(nums)):
11             self.list [i+1] = self.list [i] + nums[i]
12
13         def sumRange(self, i: int, j: int) -> int:
14             return self.list [j+1] - self.list [i]
15
16
17 # Your NumArray object will be instantiated and called as such:
18 # obj = NumArray(nums)
19 # param_1 = obj.sumRange(i,j)

```

```

1  #
2  # @lc app=leetcode.cn id=304 lang=python3
3  #
4  # [304] 二维区域和检索 - 矩阵不可变
5  #
6  class NumMatrix:
7      def __init__(self, matrix: List[List[int]]):
8          if not matrix:
9              return
10         n, m = len(matrix), len(matrix[0])
11         self.sums = [ [0 for j in range(m+1)] for i in range(n+1) ]
12         for i in range(1, n+1):
13             for j in range(1, m+1):
14                 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]
15
16         def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
17             row1, col1, row2, col2 = row1+1, col1+1, row2+1, col2+1
18             return self.sums[row2][col2] - self.sums[row2][col1-1] - self.sums[row1-1][col2] + self.sums[row1-1][col1-1]

```

```

19
20 # Your NumMatrix object will be instantiated and called as such:
21 # obj = NumMatrix(matrix)
22 # param_1 = obj.sumRegion(row1,col1,row2,col2)

1 #
2 # @lc app=leetcode.cn id=306 lang=python3
3 #
4 # [306] 累加数
5 #
6 class Solution:
7     def isAdditiveNumber(self, num: str) -> bool:
8         # 题意解读：确认前两个数字，后面即被确认
9         # 思路：遍历前两个数字，优化是遍历不超过num_str的一半即可
10        # 限制：开头不可为0--->但有'000'的情况，len(num)至少为3
11        # 0可以作为一个数字，但不能有以0开头的数字
12        len_num = len(num)
13        if len_num < 3:
14            return False
15
16        for i in range(len_num//2 + 1):
17            num1 = num[:i+1]
18            # 若num1是以0开头的数字，return False
19            if num1[0] == '0' and i >= 1 :
20                return False
21
22            for j in range(i+1, len_num//2+i+1):
23                num2 = num[i+1:j+1]
24                # 若num2以0开头，break
25                if num2[0] == '0' and j >= i + 2 :
26                    break
27                num3 = num[j+1:]
28                if self.isValid(num1, num2, num3) and num3:
29                    return True
30        return False
31
32    def isValid(self, num1, num2, num3):
33        # 已确定前两个数字，判断是否合法
34        while num3:
35            sum_num = str(int(num1) + int(num2))
36            if num3.startswith(sum_num):
37                num1 = num2
38                num2 = sum_num
39                num3 = num3[len(sum_num):]
40            else :
41                return False

```

```
42     return True
```

```
1  #
2  # @lc app=leetcode.cn id=309 lang=python3
3  #
4  # [309] 最佳买卖股票时机含冷冻期
5  #
6  class Solution:
7      def maxProfit(self, prices: List[int]) -> int:
8          if len(prices) < 2:
9              return 0
10         sale = [0 for _ in range(len(prices))]
11         buy = [0 for _ in range(len(prices))]
12         cool = [0 for _ in range(len(prices))]
13
14         buy[0] = -prices[0]
15
16         for i in range(1, len(prices)):
17             cool[i] = sale[i-1]
18             buy[i] = max(buy[i-1], cool[i-1] - prices[i])
19             sale[i] = max(sale[i-1], buy[i] + prices[i])
20
21         return max(sale[-1], cool[-1])
```

```
1  #
2  # @lc app=leetcode.cn id=313 lang=python3
3  #
4  # [313] 超级丑数
5  #
6  class Solution:
7      def nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
8          ugly = [1]
9          ls = len(primes)
10         ix = [0]*ls
11         idx = 1
12         while idx < n:
13             newugly = min([ugly[ix[i]]*primes[i] for i in range(ls)])
14             ugly.append(newugly)
15             for i in range(ls):
16                 while ugly[ix[i]]*primes[i]<= newugly:
17                     ix[i] += 1
18             idx += 1
19         return ugly[-1]
```

```
1  #
2  # @lc app=leetcode.cn id=319 lang=python3
3  #
```

```

4 # [319] 灯泡开关
5 #
6 class Solution:
7     def bulbSwitch(self, n: int) -> int:
8         return int(math.sqrt(n))

```

```

1 #
2 # @lc app=leetcode.cn id=322 lang=python3
3 #
4 # [322] 零钱兑换
5 #
6 class Solution:
7     def coinChange(self, coins: List[int], amount: int) -> int:
8         if amount == 0:
9             return 0
10        if not coins :
11            return -1
12
13        coins.sort()
14        dp = [float('inf')] * (amount + 1)
15        dp[0] = 0
16
17        for coin in coins:
18            for j in range(coin, amount+1):
19                dp[j] = min(dp[j], dp[j - coin] + 1)
20
21        if dp[-1] > amount:
22            return -1
23        else:
24            return dp[-1]

```

```

1 #
2 # @lc app=leetcode.cn id=324 lang=python3
3 #
4 # [324] 摆动排序 II
5 #
6 class Solution:
7     def wiggleSort(self, nums: List[int]) -> None:
8         # 降序
9         nums.sort(reverse=True)
10        nums[1::2], nums[0::2] = nums[len(nums) // 2:], nums[len(nums) // 2:]

```

```

1 #
2 # @lc app=leetcode.cn id=326 lang=python3
3 #
4 # [326] 3的幂
5 #

```

```

6 class Solution:
7     def isPowerOfThree(self, n: int) -> bool:
8         while n > 1:
9             n /= 3
10        if n == 1:
11            return True
12        else:
13            return False

```

```

1 #
2 # @lc app=leetcode.cn id=329 lang=python3
3 #
4 # [329] 矩阵中的最长递增路径
5 #
6 class Solution:
7     def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
8         if not matrix:
9             return 0
10
11        m, n, res = len(matrix), len(matrix[0]), 0
12        # 用于记录每个点的最长递增路径的长度
13        cache = [[-1 for _ in range(n)] for _ in range(m)]
14
15        for i in range(m):
16            for j in range(n):
17                # 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
18                cur_len = self.dfs(matrix, i, j, cache)
19                res = max(res, cur_len)
20        return res
21
22    def dfs(self, matrix, i, j, cache):
23        if cache[i][j] != -1:
24            return cache[i][j]
25
26        m, n, res = len(matrix), len(matrix[0]), 1
27        for x_offset, y_offset in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
28            x, y = i + x_offset, j + y_offset
29
30            if x < 0 or y < 0 or x >= m or y >= n or matrix[x][y] <= matrix[i][j]:
31                continue
32            # x,y比i,j位置值大
33            length = 1 + self.dfs(matrix, x, y, cache)
34            res = max(length, res)
35        # 记录当前这个点的最长递增路径长度
36        cache[i][j] = res
37        return res

```

```

1 #
2 # @lc app=leetcode.cn id=335 lang=python3
3 #
4 # [335] 路径交叉
5 #
6 class Solution:
7     def isSelfCrossing(self, x: List[int]) -> bool:
8         for i in range(len(x)):
9             if i + 3 < len(x) and x[i] >= x[i + 2] \
10                and x[i + 1] <= x[i + 3]:
11                 return True
12             if i + 4 < len(x) and x[i + 1] == x[i + 3] \
13                and x[i] + x[i + 4] >= x[i + 2]:
14                 return True
15             if i + 5 < len(x) and x[i] < x[i + 2] \
16                and x[i + 4] < x[i + 2] \
17                and x[i + 2] <= x[i] + x[i + 4] \
18                and x[i + 1] < x[i + 3] \
19                and x[i + 3] <= x[i + 1] + x[i + 5]:
20                 return True
21         return False

```

```

1 #
2 # @lc app=leetcode.cn id=342 lang=python3
3 #
4 # [342] 4的幂
5 #
6 class Solution:
7     def isPowerOfFour(self, num: int) -> bool:
8         # bin(4**0) '0b1'
9         # bin(4**1) '0b100'
10        # bin(4**2) '0b10000'
11        # bin(4**3) '0b1000000'
12        return num > 0 and num & (num-1) == 0 and len(bin(num)[3:]) % 2 == 0

```

```

1 #
2 # @lc app=leetcode.cn id=343 lang=python3
3 #
4 # [343] 整数拆分
5 #
6 class Solution:
7     def integerBreak(self, n: int) -> int:
8         dp = [1]*(n+1)
9         # dp[0] = 0
10        # dp[1] = 1
11        # dp[2] = 1

```

```

12     for i in range(3,n+1):
13         for j in range(1,i):
14             dp[i] = max(dp[i],j*(i-j),j*dp[i-j])
15
16     return dp[-1]

```

```

1  #
2  # @lc app=leetcode.cn id=344 lang=python3
3  #
4  # [344] 反转字符串
5  #
6  class Solution:
7      def reverseString( self , s: List[str] ) -> None:
8          n=len(s)
9          for i in range(n//2):
10             s[i],s[n-i-1] = s[n-i-1],s[i]

```

```

1  #
2  # @lc app=leetcode.cn id=345 lang=python3
3  #
4  # [345] 反转字符串中的元音字母
5  #
6  class Solution:
7      def reverseVowels( self , s: str ) -> str:
8          s = list(s)
9          n=len(s)
10         l ,r = 0,n-1
11         while l < r:
12             if s[l] not in 'aeiouAEIOU':
13                 l += 1
14             elif s[r] not in 'aeiouAEIOU':
15                 r -= 1
16             else :
17                 s[l],s[r] = s[r],s[l]
18                 l += 1
19                 r -= 1
20         return ''.join(s)

```

```

1  #
2  # @lc app=leetcode.cn id=349 lang=python3
3  #
4  # [349] 两个数组的交集
5  #
6  class Solution:
7      def intersection ( self , nums1: List[int] , nums2: List[int] ) -> List[int]:
8          # return list (set (nums1) & set(nums2))
9          """

```



```

10     res = []
11     for i in nums1:
12         if i not in res and i in nums2:
13             res.append(i)
14
15     return res
16     """
17     nums1.sort()
18     nums2.sort()
19     if not nums1 or not nums2:
20         return []
21     if nums1[0] == nums2[0]:
22         foo = self.intersection(nums1[1:], nums2[1:])
23         if foo and foo[0] == nums1[0]:
24             return foo
25         else:
26             return [nums1[0]] + foo
27     elif nums1[0] < nums2[0]:
28         return self.intersection(nums1[1:], nums2)
29     else:
30         return self.intersection(nums1, nums2[1:])

```

```

1  #
2  # @lc app=leetcode.cn id=350 lang=python3
3  #
4  # [350] 两个数组的交集 II
5  #
6  class Solution:
7      def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
8          nums1.sort()
9          nums2.sort()
10         res = []
11         pos1 = pos2 = 0
12         while pos1 < len(nums1) and pos2 < len(nums2):
13             if nums1[pos1] == nums2[pos2]:
14                 res.append(nums1[pos1])
15                 pos1 += 1
16                 pos2 += 1
17             elif nums1[pos1] < nums2[pos2]:
18                 pos1 += 1
19             else:
20                 pos2 += 1
21         return res

```

```

1  #
2  # @lc app=leetcode.cn id=354 lang=python3

```

```

3  #
4  # [354] 俄罗斯套娃信封问题
5  #
6  class Solution:
7      def maxEnvelopes(self, envelopes: List[List[int]]) -> int:
8          if not envelopes:
9              return 0
10         """
11         # 超时
12         envelopes.sort(key=lambda x:x[0])
13         dp = [1] * len(envelopes)
14         for i in range(len(envelopes)):
15             for j in range(i):
16                 if envelopes[i][0] > envelopes[j][0] and envelopes[i][1] > envelopes[j][1]:
17                     dp[i] = max(dp[i], dp[j]+1)
18         return max(dp)
19         """
20
21         from bisect import bisect_left
22         # 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
23         # 按w升序,h降序排列
24         envelopes.sort(key=lambda x:(x[0], -x[1]))
25         up_list = []
26         for e in envelopes:
27             index = bisect_left(up_list, e[1])
28             if index == len(up_list):
29                 up_list.append(e[1])
30             else:
31                 up_list[index] = e[1]
32         return len(up_list)

```

```

1  #
2  # @lc app=leetcode.cn id=367 lang=python3
3  #
4  # [367] 有效的完全平方数
5  #
6  class Solution:
7      def isPerfectSquare(self, num: int) -> bool:
8          """
9          l,r = 1,num
10         while l <= r:
11             mid = (l+r)//2
12             if mid ** 2 == num:
13                 return True
14             elif mid ** 2 < num:
15                 l = mid +1

```

```

16         else :
17             r = mid - 1
18         return False
19     """
20     x = num
21     while x ** 2 > num:
22         x = (x+num//x)//2
23     return x ** 2 == num

```

```

1  #
2  # @lc app=leetcode.cn id=368 lang=python3
3  #
4  # [368] 最大整除子集
5  #
6  class Solution:
7      def largestDivisibleSubset ( self , nums: List[int] ) -> List[int]:
8          nums.sort()
9          dp = [[x] for x in nums]
10         res = []
11         for i in range(len(nums)):
12             for j in range(i):
13                 if nums[i]%nums[j] == 0 and len(dp[j])+1 > len(dp[i]):
14                     dp[i] = dp[j] + [nums[i]]
15             if len(dp[i]) > len(res):
16                 res = dp[i]
17         return res

```

```

1  #
2  # @lc app=leetcode.cn id=371 lang=python3
3  #
4  # [371] 两整数之和
5  #
6  class Solution:
7      def getSum(self, a: int , b: int ) -> int:
8          MAX_INT = 0x7FFFFFFF
9          MIN_INT = 0x80000000
10         MASK = 0x100000000
11         while b:
12             a, b = (a ^ b) % MASK, ((a & b) << 1) % MASK
13         return a if a <= MAX_INT else ~((a % MIN_INT) ^ MAX_INT)

```

```

1  #
2  # @lc app=leetcode.cn id=374 lang=python3
3  #
4  # [374] 猜数字大小
5  #
6  # The guess API is already defined for you.

```

```

7  # @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
8  # def guess(num: int) -> int:
9
10 class Solution:
11     def guessNumber(self, n: int) -> int:
12         start, end = 1, n
13         while start <= end:
14             mid = (start + end) // 2
15             if guess(mid) == 0:
16                 return mid
17             elif guess(mid) == 1:
18                 start = mid + 1
19             else:
20                 end = mid

```

```

1  #
2  # @lc app=leetcode.cn id=383 lang=python3
3  #
4  # [383] 赎金信
5  #
6  class Solution:
7     def canConstruct(self, ransomNote: str, magazine: str) -> bool:
8         letter_map = {}
9         for i in magazine:
10             letter_map[i] = letter_map.get(i, 0) + 1
11         for i in ransomNote:
12             letter_map[i] = letter_map.get(i, 0) - 1
13             if letter_map[i] < 0:
14                 return False
15         return True

```

```

1  #
2  # @lc app=leetcode.cn id=387 lang=python3
3  #
4  # [387] 字符串中的第一个唯一字符
5  #
6  class Solution:
7     def firstUniqChar(self, s: str) -> int:
8         letter_map = {}
9         for i in s:
10             letter_map[i] = letter_map.get(i, 0) + 1
11         for i in range(len(s)):
12             if letter_map[s[i]] == 1:
13                 return i
14         return -1

```

```

1  #
2  # @lc app=leetcode.cn id=393 lang=python3
3  #
4  # [393] UTF-8 编码验证
5  #
6  class Solution:
7      def validUtf8(self, data: List[int]) -> bool:
8          # cnt表示后面接几个字节字符
9          # cnt 从0到0表示一个字符
10         cnt = 0
11         for d in data:
12             if cnt == 0:
13                 if (d >> 5) == 0b110:
14                     cnt = 1
15                 elif (d >> 4) == 0b1110:
16                     cnt = 2
17                 elif (d >> 3) == 0b11110:
18                     cnt = 3
19                 # 0xxxxxxx 后面不接
20                 # 这种情况首位不是0就错
21                 elif (d >> 7):
22                     return False
23             else:
24                 # 如果不接10xxxxxx
25                 if (d >> 6) != 0b10:
26                     return False
27             cnt -= 1
28         return cnt == 0

```

```

1  #
2  # @lc app=leetcode.cn id=410 lang=python3
3  #
4  # [410] 分割数组的最大值
5  #
6  class Solution:
7      def splitArray(self, nums: List[int], m: int) -> int:
8          # 最大值最小的范围(单个最大,整体和)
9          left = max(nums)
10         right = sum(nums)
11
12         while left < right:
13             mid = (right + left) // 2
14             count = self.count(nums, mid)
15             if count > m:
16                 left = mid + 1
17             else:

```

```

18         right = mid
19         return left
20
21     def count(self, nums, mid):
22         tmpsum = 0
23         count = 1
24         for num in nums:
25             tmpsum += num
26             if tmpsum > mid:
27                 tmpsum = num
28                 count += 1
29         return count

```

```

1  #
2  # @lc app=leetcode.cn id=414 lang=python3
3  #
4  # [414] 第三大的数
5  #
6  class Solution:
7      def thirdMax(self, nums: List[int]) -> int:
8          nums = list(set(nums))
9          if len(nums) < 3:
10             return max(nums)
11          nums.sort()
12          return nums[-3]

```

```

1  #
2  # @lc app=leetcode.cn id=416 lang=python3
3  #
4  # [416] 分割等和子集
5  #
6  class Solution:
7      def canPartition(self, nums: List[int]) -> bool:
8          # 背包问题+动态规划
9          target = sum(nums)
10         if target % 2 == 1:
11             return False
12         target //= 2
13
14         # 行nums 列对应 目标值
15         dp = [[False] * (target + 1) for _ in range(len(nums))]
16         # 每一列赋值
17         if nums[0] <= target:
18             dp[0][nums[0]] = True
19
20         for i in range(1, len(nums)):

```

```

21         for j in range(1,target+1):
22             if j >= nums[i]:
23                 dp[i][j] = dp[i-1][j] or dp[i-1][j-nums[i]]
24             else:
25                 dp[i][j] = dp[i-1][j]
26         return dp[-1][-1]

```

```

1  #
2  # @lc app=leetcode.cn id=432 lang=python3
3  #
4  # [432] 全 O(1) 的数据结构
5  #
6  class AllOne:
7      def __init__(self):
8          self.lookup = {}
9
10     def inc(self, key: str) -> None:
11         if key in self.lookup:
12             self.lookup[key] += 1
13         else:
14             self.lookup[key] = 1
15
16     def dec(self, key: str) -> None:
17         if key in self.lookup:
18             if self.lookup[key] == 1:
19                 self.lookup.pop(key)
20             else:
21                 self.lookup[key] -= 1
22
23     def getMaxKey(self) -> str:
24         return max(self.lookup.items(), key=lambda x: x[1], default="")[0]
25
26     def getMinKey(self) -> str:
27         return min(self.lookup.items(), key=lambda x: x[1], default="")[0]
28
29 # Your AllOne object will be instantiated and called as such:
30 # obj = AllOne()
31 # obj.inc(key)
32 # obj.dec(key)
33 # param_3 = obj.getMaxKey()
34 # param_4 = obj.getMinKey()

```

```

1  #
2  # @lc app=leetcode.cn id=434 lang=python3
3  #
4  # [434] 字符串中的单词数

```

```

5 #
6 class Solution:
7     def countSegments(self, s: str) -> int:
8         if not s:
9             return 0
10        """
11        segment_count = 0
12        for i in range(len(s)):
13            if i == 0 and s[i] != ' ':
14                segment_count = 1
15            elif s[i-1] == ' ' and s[i] != ' ':
16                segment_count += 1
17
18        return segment_count
19        """
20        s_list = list(s.split(" "))
21        s_list = [i for i in s_list if i != "" and i != " "]
22        return len(s_list)

```

```

1 #
2 # @lc app=leetcode.cn id=442 lang=python3
3 #
4 # [442] 数组中重复的数据
5 #
6 class Solution:
7     def findDuplicates(self, nums: List[int]) -> List[int]:
8         res = []
9         for x in nums:
10            x = abs(x)
11            # 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)
12            if nums[x-1] < 0:
13                res.append(x)
14            else:
15                nums[x-1] *= -1
16        return res

```

```

1 #
2 # @lc app=leetcode.cn id=443 lang=python3
3 #
4 # [443] 压缩字符串
5 #
6 class Solution:
7     def compress(self, chars: List[str]) -> int:
8         # count 几个一样
9         # walker 写入的位置
10        # runner 往后跑的

```



```

11     walker, runner = 0, 0
12
13     while runner < len(chars):
14         # 写字符
15         chars[walker] = chars[runner]
16         count = 1
17
18         while runner + 1 < len(chars) and \
19             chars[runner] == chars[runner+1] :
20             runner += 1
21             count += 1
22
23         if count > 1:
24             for c in str(count):
25                 # 写数字
26                 walker += 1
27                 chars[walker] = c
28
29         runner += 1
30         walker += 1
31
32     return walker

```

```

1  #
2  # @lc app=leetcode.cn id=448 lang=python3
3  #
4  # [448] 找到所有数组中消失的数字
5  #
6  class Solution:
7      def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
8          """
9          # time Limit Exceeded
10         res = []
11         leng = len(nums)
12         for i in range(leng):
13             if i+1 not in nums:
14                 res.append(i+1)
15         return res
16         """
17         for num in nums:
18             index = abs(num)-1
19             if nums[index]>0:
20                 nums[index] *= -1
21
22         res = []
23         for i in range(len(nums)):

```

```

24         if nums[i] > 0:
25             res.append(i+1)
26     return res

```

```

1  #
2  # @lc app=leetcode.cn id=474 lang=python3
3  #
4  # [474] 一和零
5  #
6  class Solution:
7      def findMaxForm(self, strs: List[str], m: int, n: int) -> int:
8          if not strs:
9              return 0
10         # 准备很多个背包
11         dp = [ [0]*(n+1) for _ in range(m+1)]
12
13         for str in strs:
14             count0 = str.count('0')
15             count1 = str.count('1')
16
17             # 遍历可容纳的背包
18             for zeroes in range(m, count0 - 1, -1):
19                 for ones in range(n, count1 - 1, -1):
20                     dp[zeroes][ones] = max(dp[zeroes][ones] ,
21                                             1 + dp[zeroes - count0][ones - count1])
22     return dp[m][n]

```

```

1  #
2  # @lc app=leetcode.cn id=485 lang=python3
3  #
4  # [485] 最大连续1的个数
5  #
6  class Solution:
7      def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
8          maxval = 0
9          tmp = 0
10         for i in range(len(nums)):
11             if nums[i] != 0 :
12                 tmp += 1
13             else :
14                 maxval = max(maxval,tmp)
15                 tmp = 0
16         maxval = max(maxval,tmp)
17     return maxval

```

```

1  #
2  # @lc app=leetcode.cn id=494 lang=python3

```

```

3 #
4 # [494] 目标和
5 #
6 class Solution:
7     def findTargetSumWays(self, nums: List[int], S: int) -> int:
8         sum_nums = sum(nums)
9         if sum_nums < S or (S + sum_nums)%2 != 0:
10             return 0
11
12         target = (S + sum_nums) // 2
13         dp = [0]*(target + 1)
14         dp[0] = 1
15         for num in nums:
16             for i in range(target, num-1, -1):
17                 dp[i] += dp[i - num]
18         return dp[-1]

```

```

1 #
2 # @lc app=leetcode.cn id=518 lang=python3
3 #
4 # [518] 零钱兑换 II
5 #
6 class Solution:
7     def change(self, amount: int, coins: List[int]) -> int:
8         dp = [0] * (amount + 1)
9         dp[0] = 1
10
11         for coin in coins:
12             for x in range(coin, amount + 1):
13                 dp[x] += dp[x - coin]
14         return dp[amount]

```

```

1 #
2 # @lc app=leetcode.cn id=532 lang=python3
3 #
4 # [532] 数组中的K-diff数对
5 #
6 class Solution:
7     def findPairs(self, nums: List[int], k: int) -> int:
8         dic = {}
9         if k < 0:
10             return 0
11         res = 0
12         for num in nums:
13             dic[num] = dic.get(num,0) + 1
14         for num in nums:

```

```

15         # 值在里面 且 k 不为0
16         if dic.get(num-k,0) > 0 and k != 0:
17             res += 1
18             dic[num-k] = 0
19         # k 为0,值有多个
20         elif k == 0 and dic.get(num,0) > 1:
21             res += 1
22             dic[num-k] = 0
23     return res

```

```

1  #
2  # @lc app=leetcode.cn id=541 lang=python3
3  #
4  # [541] 反转字符串 II
5  #
6  class Solution:
7      def reverseStr(self, s: str, k: int) -> str:
8          if len(s)<k:
9              return s[::-1]
10         if len(s)<2*k:
11             return s[:k][::-1]+s[k:]
12         return s[:k][::-1]+s[k:2*k] + self.reverseStr(s[2*k:], k)

```

```

1  #
2  # @lc app=leetcode.cn id=547 lang=python3
3  #
4  # [547] 朋友圈
5  #
6  class Solution:
7      def findCircleNum(self, M: List[List[int]]) -> int:
8          # 遍历每个人,遍历到过置1
9          visited = [0 for _ in range(len(M))]
10         # 圈数
11         count = 0
12         for i in range(len(M)):
13             # 等于1表示被别的圈包进去了,等于0表示再开一个圈
14             if visited[i] == 0:
15                 visited[i] = 1
16                 self.dfs(M, visited, i)
17                 count += 1
18         return count
19
20     # 判断和i认识的都是哪些人
21     def dfs(self, M, visited, i):
22         # 全1了
23         # if sum(visited) == len(M):

```

```

24         if all(visited):
25             return
26         for j in range(len(M)):
27             if j != i and visited[j] == 0 and M[i][j] == 1 :
28                 visited[j] = 1
29                 self.dfs(M, visited, j)

```

```

1  #
2  # @lc app=leetcode.cn id=551 lang=python3
3  #
4  # [551] 学生出勤记录 I
5  #
6  class Solution:
7      def checkRecord(self, s: str) -> bool:
8          count = 0
9          for i in range(len(s)):
10             if s[i] == 'A':
11                 # 大于1个A
12                 count += 1
13                 if count > 1:
14                     return False
15             elif s[i] == 'L' and 0 < i < len(s)-1 \
16                  and s[i-1] == 'L' == s[i+1]:
17                 return False
18         return True

```

```

1  #
2  # @lc app=leetcode.cn id=557 lang=python3
3  #
4  # [557] 反转字符串中的单词 III
5  #
6  class Solution:
7      def reverseWords(self, s: str) -> str:
8          return ' '.join([word[::-1] for word in s.split(' ')])

```

```

1  #
2  # @lc app=leetcode.cn id=560 lang=python3
3  #
4  # [560] 和为K的子数组
5  #
6  class Solution:
7      def subarraySum(self, nums: List[int], k: int) -> int:
8          """
9          # 超时
10         same_length = 0
11         for start in range(len(nums)):
12             sums = 0

```

```

13         for end in range(start, len(nums)):
14             sums += nums[end]
15             if sums == k:
16                 same_length += 1
17         return same_length
18     """
19
20     count = 0
21     sums = 0
22     dic = {0:1}
23
24     for num in nums:
25         sums += num
26         count += dic.get(sums-k,0)
27         dic[sums] = dic.get(sums,0) + 1
28
29     return count

```

```

1  #
2  # @lc app=leetcode.cn id=561 lang=python3
3  #
4  # [561] 数组拆分 I
5  #
6  class Solution:
7      def arrayPairSum(self, nums: List[int]) -> int:
8          nums.sort()
9          return sum(nums[::2])

```

```

1  #
2  # @lc app=leetcode.cn id=566 lang=python3
3  #
4  # [566] 重塑矩阵
5  #
6  class Solution:
7      def matrixReshape(self, nums: List[List[int]], r: int, c: int) -> List[List[int]]:
8          row = len(nums)
9          col = len(nums[0])
10         if row * col != r * c:
11             return nums
12         res = []
13         for i in range(row):
14             for j in range(col):
15                 k = nums[i][j]
16                 if len(res[-1]) < c:
17                     res[-1].append(k)
18                 else:

```

```

19         res.append([k])
20     return res

```

```

1  #
2  # @lc app=leetcode.cn id=567 lang=python3
3  #
4  # [567] 字符串的排列
5  #
6  class Solution:
7      def checkInclusion( self , s1: str , s2: str ) -> bool:
8          if len(s1) > len(s2):
9              return False
10         dic = [0] * 26
11         for i in range(len(s1)):
12             dic[ord(s1[i]) - ord('a')] -= 1
13             dic[ord(s2[i]) - ord('a')] += 1
14
15         for i in range(len(s2)-len(s1)):
16             if sum(list(map(abs,dic))) == 0 :
17                 return True
18             else :
19                 # 滑动窗往右滑动
20                 dic[ord(s2[i+len(s1)]) - ord('a')] += 1
21                 dic[ord(s2[i]) - ord('a')] -= 1
22         return sum(list(map(abs,dic))) == 0

```

```

1  #
2  # @lc app=leetcode.cn id=575 lang=python3
3  #
4  # [575] 分糖果
5  #
6  class Solution:
7      def distributeCandies( self , candies: List[int] ) -> int:
8          return int( min( len(set(candies)) , len(candies)//2 ) )

```

```

1  #
2  # @lc app=leetcode.cn id=581 lang=python3
3  #
4  # [581] 最短无序连续子数组
5  #
6  class Solution:
7      def findUnsortedSubarray( self , nums: List[int] ) -> int:
8          num_sort = nums[:] # 浅拷贝和深拷贝
9          num_sort.sort()
10         n=len(nums)
11         i,j=0,n-1
12         while i<n and nums[i]==num_sort[i]:

```

```

13         i += 1
14         while j>i+1 and nums[j]==num_sort[j]:
15             j -= 1
16         return j-i+1

```

```

1  #
2  # @lc app=leetcode.cn id=605 lang=python3
3  #
4  # [605] 种花问题
5  #
6  class Solution:
7      def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
8          # 前后补零解决边界问题
9          nums=[0]+flowerbed+[0]
10         cnt=0
11         i=1
12         while i<len(flowerbed)+1:
13             if nums[i-1]==0 and nums[i]==0 and nums[i+1]==0:
14                 cnt += 1
15                 # 可以种花，则需要间隔一个位置，所以+2
16                 i += 2
17             else:
18                 i+=1
19         return cnt>=n

```

```

1  #
2  # @lc app=leetcode.cn id=628 lang=python3
3  #
4  # [628] 三个数的最大乘积
5  #
6  class Solution:
7      def maximumProduct(self, nums: List[int]) -> int:
8          nums.sort()
9          res1 = nums[-1]*nums[-2]*nums[-3]
10         res2 = nums[-1]*nums[0]*nums[1]
11         return max(res1,res2)

```

```

1  #
2  # @lc app=leetcode.cn id=638 lang=python3
3  #
4  # [638] 大礼包
5  #
6  class Solution:
7      def shoppingOffers(self, price: List[int], special: List[List[int]], needs: List[int]) -> int:
8          self.dic = {}
9          return self.dfs(price, special, needs)
10

```



```

11 def dfs(self, price, special, needs):
12     # 买完了
13     if sum(needs) == 0:
14         return 0
15     # 避免重复
16     if tuple(needs) in self.dic:
17         return self.dic[tuple(needs)]
18
19     res = 0
20     # 没有优惠的价格
21     # 单个买
22     for i in range(len(needs)):
23         res += needs[i]*price[i]
24
25     # 买套装
26     for sp in special:
27         for i in range(len(needs)):
28             needs[i] -= sp[i]
29             if all(needs[i] >= 0 for i in range(len(needs))):
30                 res = min(res, self.dfs(price, special, needs) + sp[-1])
31             for i in range(len(needs)):
32                 needs[i] += sp[i]
33
34     self.dic[tuple(needs)] = res
35     return res

```

```

1 #
2 # @lc app=leetcode.cn id=643 lang=python3
3 #
4 # [643] 子数组最大平均数 I
5 #
6 class Solution:
7     def findMaxAverage(self, nums: List[int], k: int) -> float:
8         tmp = maxmean = sum(nums[:k])
9         for i in range(k, len(nums)):
10             tmp += (nums[i] - nums[i-k])
11             maxmean = max(maxmean, tmp)
12         return maxmean/k

```

```

1 #
2 # @lc app=leetcode.cn id=661 lang=python3
3 #
4 # [661] 图片平滑器
5 #
6 class Solution:
7     def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:

```

```

8     R, C = len(M), len(M[0])
9     res = [[0] * C for _ in range(R)]
10
11     for r in range(R):
12         for c in range(C):
13             count = 0
14             for nr in (r-1, r, r+1):
15                 for nc in (c-1, c, c+1):
16                     if 0 <= nr < R and 0 <= nc < C:
17                         res[r][c] += M[nr][nc]
18                         count += 1
19             res[r][c] //= count
20     return res

```

```

1  #
2  # @lc app=leetcode.cn id=665 lang=python3
3  #
4  # [665] 非递减数列
5  #
6  class Solution:
7      def checkPossibility ( self , nums: List[int]) -> bool:
8          count = 0
9          for i in range(len(nums)-1):
10             if nums[i]>nums[i+1]:
11                 count +=1
12                 #变相去掉nums[i]
13                 if i <1 or nums[i-1] <= nums[i+1]:
14                     nums[i] = nums[i+1]
15                 else :
16                     # 变相去掉nums[i+1]
17                     nums[i+1]=nums[i]
18     return count <= 1

```

```

1  #
2  # @lc app=leetcode.cn id=674 lang=python3
3  #
4  # [674] 最长连续递增序列
5  #
6  class Solution:
7      def findLengthOfLCIS(self, nums: List[int]) -> int:
8          if not nums:
9              return 0
10             count = 1
11             res = 0
12             for i in range(len(nums)-1):
13                 if nums[i] < nums[i+1]:

```

```

14         count += 1
15     else :
16         res = max(res,count)
17         count = 1
18     return max(res,count)

```

```

1  #
2  # @lc app=leetcode.cn id=680 lang=python3
3  #
4  # [680] 验证回文字符串
5  #
6  class Solution:
7      def validPalindrome(self, s: str) -> bool:
8          count = 0
9          for i in range(len(s)//2):
10             if s[i] != s[-1-i]:
11                 t, u = s[:i]+s[i+1:], s[:-1-i]+s[len(s)-i:]
12                 return t == t[::-1] or u == u[::-1]
13     return True

```

```

1  #
2  # @lc app=leetcode.cn id=695 lang=python3
3  #
4  # [695] 岛屿的最大面积
5  #
6  class Solution:
7      def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
8          res = 0
9          for i in range(len(grid)):
10             for j in range(len(grid[0])):
11                 if grid[i][j] == 1:
12                     temp = self.dfs(grid, i, j)
13                     res = max(res,temp)
14     return res
15
16     def dfs(self, grid, i, j):
17         # 终止条件
18         if i < 0 or j < 0 or i >= len(grid) or j >= len(grid[0]) or grid[i][j] == 0:
19             return 0
20
21         # 四个方向搜索
22         grid[i][j] = 0
23         res = 1
24         res += self.dfs(grid, i-1, j)
25         res += self.dfs(grid, i, j-1)
26         res += self.dfs(grid, i+1, j)

```

```

27         res += self.dfs(grid, i, j+1)
28
29     return res

```

```

1  #
2  # @lc app=leetcode.cn id=754 lang=python3
3  #
4  # [754] 到达终点数字
5  #
6  class Solution:
7      def reachNumber(self, target: int) -> int:
8          target = abs(target)
9          p, n = 0, 0
10         # 和比目标值还小 或者不同奇偶
11         while p < target or (p + target) % 2 != 0:
12             n += 1
13             p += n
14         return n

```

```

1  #
2  # @lc app=leetcode.cn id=836 lang=python3
3  #
4  # [836] 矩形重叠
5  #
6  class Solution:
7      def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
8          return not (rec1[2] <= rec2[0] or # rec1的右边在rec2的左边
9                      rec1[3] <= rec2[1] or # rec1的上边在rec2的下边
10                     rec1[0] >= rec2[2] or # rec1的左边在rec2的右边
11                     rec1[1] >= rec2[3]) # rec1的下边在rec2的上边

```

```

1  #
2  # @lc app=leetcode.cn id=974 lang=python3
3  #
4  # [974] 和可被 K 整除的子数组
5  #
6  class Solution:
7      def subarraysDivByK(self, A: List[int], K: int) -> int:
8          sums = [0]
9          for x in A:
10             sums.append((sums[-1]+x)%K)
11
12         dic = {}
13         for i in sums:
14             dic[i] = dic.get(i,0)+1
15         res = 0
16         for __,val in dic.items():

```

```
17         res += val*(val-1)//2
18     return res
```

```
1 #
2 # @lc app=leetcode.cn id=1015 lang=python3
3 #
4 # [1015] 可被 K 整除的最小整数
5 #
6 class Solution:
7     def smallestRepunitDivByK(self, K: int) -> int:
8         if K%2 == 0 or K%5 == 0:
9             return -1
10        temp = 1
11        len = 1
12        while temp % K :
13            temp = (temp % K) * 10 + 1
14            len += 1
15        return len
```

```
1 #
2 # @lc app=leetcode.cn id=1147 lang=python3
3 #
4 # [1147] 段式回文
5 #
6 class Solution:
7     def longestDecomposition(self, text: str) -> int:
8         n = len(text)
9         i, j = 0, n - 1
10        str1, str2, ans = '', '', 0
11        while i < j:
12            str1 = str1 + text[i]
13            str2 = text[j] + str2
14            if str1 == str2:
15                ans += 2
16                str1, str2 = '', ''
17                i += 1
18                j -= 1
19        if n % 2 == 1 or str1 != '':
20            ans += 1
21        return ans
```

```
1 #
2 # @lc app=leetcode.cn id=1293 lang=python3
3 #
4 # [1293] 网格中的最短路径
5 #
6 class Solution:
```

```

7  def shortestPath(self, grid: List[List[int]], k: int) -> int:
8      m, n = len(grid), len(grid[0])
9      if m == 1 and n == 1:
10         return 0
11
12     k = min(k, m + n - 3)
13     visited = set((0, 0, k))
14     q = [(0, 0, k)]
15
16     step = 0
17     while q:
18         step += 1
19         cnt = len(q)
20         for _ in range(cnt):
21             x, y, rest = q.pop(0)
22             for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
23                 nx, ny = x + dx, y + dy
24                 if 0 <= nx < m and 0 <= ny < n:
25                     if grid[nx][ny] == 0 and (nx, ny, rest) not in visited:
26                         if nx == m - 1 and ny == n - 1:
27                             return step
28                         q.append((nx, ny, rest))
29                         visited.add((nx, ny, rest))
30                     elif grid[nx][ny] == 1 and rest > 0 and (nx, ny, rest - 1) not in visited:
31                         q.append((nx, ny, rest - 1))
32                         visited.add((nx, ny, rest - 1))
33     return -1

```