1、反转链表

2、**设计LRU缓存结构**

3、二叉树前中后排序

4、寻找第K大

5、两数之和

6、**合并两个排序的链表**

7、跳台阶

8、**连续子数组的最大和**

9、**最长无重复子数组**

10、**有效括号序列**

11、**按之字形顺序打印二叉树**

12、**最长公共子串**

13、两个链表的第一个公共结点

14、!!!链表相加(二)

15、反转字符串

16、斐波那契数列

17、!最长回文子串

18、!!!最长上升子序列(三)**

19、!!!字符串的排列**

20、 **接雨水问题**

21、!!! **输出二叉树的右视图**

22、岛屿数量

23、**二叉树的最大深度**

24、**判断是否为回文字符串**

25、**数组中出现次数超过一半的数字**

26、矩阵的最小路径和

27、表达式求值

28、!!!**字符串出现次数的TopK问题**

29、**判断一个链表是否为回文结构**

30、!!**不同路径的数目(一)**

31、合并区间

32、矩阵元素查找

33、链表的奇偶排序

34、**顺时针旋转矩阵**

35、!!!加起来和为目标值的组合(二)**

36、验证IP地址

37、二进制中1的个数

38、最大正方形

39、kmp算法

40、将列表中的元素转为字符串

41、质数因子

42、斗地主之顺子

43、猜密码

44、IPv4地址转换成整数

45、英文输入法

46、玩牌高手

47、找单词

48、两数之和绝对值最小

49、找朋友

50、流水线

51、磁盘容量排序

52、【We are a team】

53、目录删除

54、服务器失效判断

55、判断字符串子序列

56、组成最大值

57、称砝码

58、计算字符串的编辑距离

59、24点游戏算法

60、扑克24点运算

61、记负均正II

62、**人民币转换**

1、反转链表

temp

cur.next

prev

cur

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param head ListNode类
# @return ListNode类
#
class Solution:
    def ReverseList(self , head: ListNode) -> ListNode:
        cur = head
        prev = None
        while cur:
            temp = cur.next
            cur.next = prev
            prev = cur
            cur = temp
        return prev
```

2、设计LRU缓存结构

1. len可以得出字典的长度
2. pop(index)方法可以索引弹出
3. setdefault(1,2)后接两个参数比如1,2可以在字典里添加键值对，而且这个键值对是有顺序的
4. get(index)可以索引得到key的值

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# lru design
# @param operators int整型二维数组 the ops
# @param k int整型 the k
# @return int整型一维数组
#
class Solution:
    def LRU(self , operators: List[List[int]], k: int) -> List[int]:
        dic = {}
        res = []
        for comm in operators:
            if comm[0] == 1:
                if len(dic)>=k:
                    dic.pop(list(dic)[0])
                    dic.setdefault(comm[1],comm[2])
                    continue
                if comm[1] in list(dic):
                    dic.pop(comm[1])
                    dic.setdefault(comm[1],comm[2])
                    continue
                dic.setdefault(comm[1],comm[2])
            if comm[0] == 2:
                if dic.get(comm[1]):
                    res.append(dic.get(comm[1]))
```

```
                    key = comm[1]
                    val = dic.get(comm[1])
                    dic.pop(key)
                    dic.setdefault(key,val)
                else:
                    res.append(-1)
        return res
```

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# lru design
# @param operators int整型二维数组 the ops
# @param k int整型 the k
# @return int整型一维数组
#
class Solution:
    def LRU(self , operators: List[List[int]], k: int) -> List[int]:
        dic = {}
        res = []
        for i in operators:
            if i[0]==1:
                if dic.get(i[1]):
                    dic.pop(i[1])
                    dic.setdefault(i[1],i[2])
                elif len(dic)==k:
                    dic.pop(list(dic)[0])
                    dic.setdefault(i[1],i[2])
                else:
                    dic.setdefault(i[1],i[2])
            if i[0]==2:
                if not dic.get(i[1]):
                    res.append(-1)
                else:
                    tmp = dic.get(i[1])
                    res.append(tmp)
                    dic.pop(i[1])
                    dic.setdefault(i[1],tmp)
        return res
```

3、二叉树前中后排序

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param root TreeNode类 the root of binary tree
# @return int整型二维数组
#
```

```python
class Solution:
    def threeOrders(self , root: TreeNode) -> List[List[int]]:
        # write code here
        if root == None:
            return [[],[],[]]
        list1 = []
        list2 = []
        list3 = []
        def xian(root):
            list1.append(root.val)
            if root.left != None:
                xian(root.left)
            if root.right != None:
                xian(root.right)
        def zhong(root):
            if root.left != None:
                zhong(root.left)
            list2.append(root.val)
            if root.right != None:
                zhong(root.right)
        def hou(root):
            if root.left != None:
                hou(root.left)
            if root.right != None:
                hou(root.right)
            list3.append(root.val)
        xian(root)
        zhong(root)
        hou(root)
        list_ = [list1,list2,list3]
        return list_
```

4、寻找第K大

寻找第K大是降序找地K大的，别升序找了

```python
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param a int整型一维数组
# @param n int整型
# @param K int整型
# @return int整型
#
class Solution:
    def findKth(self , a: List[int], n: int, K: int) -> int:
        # write code here
        a.sort()
        return a[n-K]
```

5、两数之和

不能以[2,1,3].sort()在创建列表的同时排序，会变成None的

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param numbers int整型一维数组
# @param target int整型
# @return int整型一维数组
#
class Solution:
    def twoSum(self , numbers: List[int], target: int) -> List[int]:
        for i in range(1,len(numbers)):
             if numbers[i]-10>target:
                    continue
             for j in range(i):
                if numbers[i]+numbers[j]==target:
                    return [j+1,i+1]
```

6、!合并两个排序的链表

调用主函数要加self

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param pHead1 ListNode类
# @param pHead2 ListNode类
# @return ListNode类
#
class Solution:
    def Merge(self , pHead1: ListNode, pHead2: ListNode) -> ListNode:
        # write code here
        if pHead1 == None or pHead2 == None:
            return pHead1 or pHead2
        if pHead1.val < pHead2.val:
            pHead1.next = self.Merge(pHead1.next,pHead2)
            return pHead1
        else:
            pHead2.next = self.Merge(pHead1,pHead2.next)
            return pHead2
```

7、跳台阶

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param number int整型
# @return int整型
```

```
#
class Solution:
    def jumpFloor(self , number: int) -> int:
        # write code here
        a = 1
        b = 2
        if number <= 2:
            return number
        for i  in range(number-2):
            sum_ = a + b
            a = b
            b = sum_
        return sum_
```

8、连续子数组的最大和

1. 考虑列表第一个数为0的情况
2. 考虑列表全为0的情况

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param array int整型一维数组
# @return int整型
#
class Solution:
    def FindGreatestSumOfSubArray(self , array: List[int]) -> int:
        # write code here
        if len(array) == 1:
            return array[0]
        list_index = [array[0]]
        res = array[0]
        for i in range(1,len(array)):
            if list_index[i-1]>0:
                tmp = list_index[i-1]+array[i]
                list_index.append(tmp)
                res = max(tmp,res)
            else:
                list_index.append(array[i])
                res = max(list_index[i],res)
        return res
```

9、最长无重复子数组

1. 注意33334这种情况，排除前面的情况用while

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param arr int整型一维数组 the array
# @return int整型
#
class Solution:
```

```
    def maxLength(self , arr: List[int]) -> int:
        if not arr:
            return []
        dic = {}
        res = 0
        left = 0
        for index,val in enumerate(arr):
            if val not in dic:
                dic.setdefault(val,1)
                res = max(res,index-left+1)
            else:
                while dic.get(val) == 1:
                    dic.pop(list(dic)[0])
                    left += 1
                dic.setdefault(val,1)
        return res
```

10、有效括号序列

1. 没有if "()" or "[]" or "{}" in s: 这种写法

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param s string字符串
# @return bool布尔型
#
class Solution:
    def isValid(self , s: str) -> bool:
        # write code here
        #str1,str2,str3 = "()","[]","{}"
        flag = True
        while flag:
            flag = False
            if "()" in s:
                s = s.replace("()","")
                flag = True
            if "[]" in s:
                s = s.replace("[]","")
                flag = True
            if "{}" in s:
                s = s.replace("{}","")
                flag = True
        if s == "":
            return True
        else:
            return False
```

11、按之字形顺序打印二叉树

1. reverse会改变列表本身，返回值为None
2. 使用队列保存队列的长度，再pop(0)

```
# class TreeNode:
```

```
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param pRoot TreeNode类
# @return int整型二维数组
#

#p root.val
#p root.right.val root.left.val
#p root.left.left.val root.left.right.val root.right.left.val
root.right.right.val

class Solution:
    def Print(self , pRoot: TreeNode) -> List[List[int]]:
        # write code here
        queue = [pRoot]
        left_to_right = True
        result = []
        while queue:
            length = len(queue)
            tmp = []
            for i in range(length):
                node = queue.pop(0)
                if node != None:
                    tmp.append(node.val)
                    queue.append(node.left)
                    queue.append(node.right)
            if tmp:
                if left_to_right:
                    result.append(tmp)
                else:
                    tmp.reverse()
                    result.append(tmp)
            left_to_right = not left_to_right
        return result
```

12、!最长公共子串

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# longest common substring
# @param str1 string字符串 the string
# @param str2 string字符串 the string
# @return string字符串
#
class Solution:
    def LCS(self , str1: str, str2: str) -> str:
        left = 0
        res = ""
        for i in range(1,len(str1)+1):
```

```
            if str1[left:i] in str2:
                res = str1[left:i]
            else:
                left += 1
        return res
```

13、!两个链表的第一个公共结点

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

#
#
# @param pHead1 ListNode类
# @param pHead2 ListNode类
# @return ListNode类
#
class Solution:
    def FindFirstCommonNode(self , pHead1 , pHead2 ):
            if pHead1 is None or pHead2 is None:
                return None
            set_A = set()
            node1,node2 = pHead1,pHead2
            while node1:
                set_A.add(node1)
                node1 = node1.next
            while node2:
                if node2 in set_A:
                    return node2
                node2 = node2.next
            return None
```

14、链表相加(二)

1. 超时，但是可学
2.

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param head1 ListNode类
# @param head2 ListNode类
# @return ListNode类
#
class Solution:
    def addInList(self , head1: ListNode, head2: ListNode) -> ListNode:
            s1 = ""
            s2 = ""
```

```
            while head1:
                s1 = s1 + str(head1.val)
                head1 = head1.next
            while head2:
                s2 = s2 + str(head2.val)
                head2 = head2.next
            he = list(str(int(s1)+int(s2)))
            res = tmp = ListNode(int(he.pop(0)))
            while he:
                node = ListNode(int(he.pop(0)))
                tmp.next = node
                tmp = node
            return res
```

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param head1 ListNode类
# @param head2 ListNode类
# @return ListNode类
#
class Solution:
    def addInList(self , head1: ListNode, head2: ListNode) -> ListNode:
        # write code here
        def reverse_List(head:ListNode):
            cur = head
            pre = None
            while cur:
                tmp = cur.next
                cur.next = pre
                pre = cur
                cur = tmp
            return pre
        head1 = reverse_List(head1)
        head2 = reverse_List(head2)
        cur_val = head1.val+head2.val
        e_val = 0
        if cur_val>=10:
            cur_val -= 10
            e_val += 1
        head3 = pre = ListNode(cur_val)
        cur1 = head1.next
        cur2 = head2.next
        while cur1 and cur2:
            cur_val = cur1.val + cur2.val + e_val
            e_val = 0
            cur1 = cur1.next
            cur2 = cur2.next
            if cur_val>=10:
                cur_val -= 10
                e_val += 1
            node = ListNode(cur_val)
```

```
            pre.next = node
            pre = node
        cur4 = ListNode(0)
        if cur1:
            cur4 = cur1
        elif cur2:
            cur4 = cur2
        else:
            cur4 = None
        while cur4:
            if e_val:
                if cur4.val == 9:
                    node = ListNode(0)
                    e_val = 1
                    pre.next = node
                    pre = node
                    cur4 = cur4.next
                else:
                    cur4.val = cur4.val + 1
                    pre.next = cur4
                    e_val = 0
                    break
            else:
                pre.next = cur4
                break
        if e_val:
            node = ListNode(1)
            pre.next = node
        head3 = reverse_List(head3)
        return head3
```

15、反转字符串

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 反转字符串
# @param str string字符串
# @return string字符串
#
class Solution:
    def solve(self , str: str) -> str:
        length = len(str)
        list_ = list(str)
        res = ""
        for i in range(length):
            res += list_.pop()
        return res
```

16、斐波那契数列

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
```

```
# @param n int整型
# @return int整型
#
class Solution:
    def Fibonacci(self , n: int) -> int:
        # write code here
        if n == 1 or n == 2:
            return 1
        a = 1
        b = 1
        for i in range(n-2):
            sum_ = a+b
            a = b
            b = sum_
        return sum_
```

17、!最长回文子串

1. 创建二维数组要这样创建dp = [[0]*length for i in range(length)]

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param A string字符串
# @return int整型
#
class Solution:
    def getLongestPalindrome(self , A: str) -> int:
        if A == "":
            return 0
        length = len(A)
        dp = [[0]*length for i in range(length)]
        res = 1
        for i in range(length):
            dp[i][i] = 1
        for i in range(length-1):
            if A[i] == A[i+1]:
                dp[i][i+1] = 1
                res = 2
        for leng in range(3,length+1):
            for i in range(length-leng+1):
                if A[i] == A[i+leng-1] and dp[i+1][i+leng-2] == 1:
                    dp[i][i+leng-1] = 1
                    res = leng
        return res
```

18、!最长上升子序列(三)**

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# retrun the longest increasing subsequence
# @param arr int整型一维数组 the array
# @return int整型一维数组
```

```
#
import bisect
class Solution:
    def LIS(self , arr: List[int]) -> List[int]:
        # write code here
        arrLen = len(arr)
        if arrLen < 2:
            return arr
        ansArr = [arr[0]]  # 记录某个数字结尾时最长的最长递增子序列，初始化第一个数字
        maxLen = [1]  # 下标i时，最长的递增子序列长度，初始化1
        for a in arr[1:]:
            if a > ansArr[-1]:  # 当前数字大于ansArr最后一个数字，子数组保持递增
                ansArr.append(a)
                maxLen.append(len(ansArr))
            # 当前数字小于等于ansArr最后一个数字，二分查找ansArr中第一个比当前数字大的下标
pos
            # 替换ansArr中下标pos的数字为当前数字，更新maxLen，记录当前最长递增子序列长度
为：pos + 1(下标+1)
            else:
                pos = bisect.bisect_left(ansArr,a)
                ansArr[pos] = a
                maxLen.append(pos + 1)
        # 找到的ansArr不一定是最终结果，[2,1,5,3,6,4,8,9,7] - > [1, 3, 4, 7, 9] (不是
最终结果)
        # [1, 1, 2, 2, 3, 3, 4, 5, 4] 从后往前遍历maxLen,依次找到等于len(arrLen)对应的
arr[i]
        ansLen = len(ansArr)
        for i in range(arrLen - 1, -1, -1):
            if maxLen[i] == ansLen:
                ansArr[ansLen - 1] = arr[i]
                ansLen -= 1
        return ansArr
```

19、!字符串的排列**

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param str string字符串
# @return string字符串一维数组
#
class Solution:
    def Permutation(self , str: str) -> List[str]:
        length = len(str)
        if length<2:
            return str
        res = []
        for i in range(length):
            for j in self.Permutation(str[:i]+str[i+1:]):
                temp = str[i] + j
                if temp not in res:
                    res.append(temp)
        return res
```

```
#法2
class Solution:
    def Permutation(self , str: str) -> List[str]:
        from itertools import permutations
        a = list(str)
        b = set()
        for i in permutations(a):
            b.add("".join(i))
        return list(b)
```

20、 接雨水问题

1. 以最高点为基准点，最左边与最右边向最高点聚合

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# retrun the longest increasing subsequence
# @param arr int整型一维数组 the array
# @return int整型一维数组
#
import bisect
class Solution:
    def LIS(self , arr: List[int]) -> List[int]:
        # write code here
        arrLen = len(arr)
        if arrLen < 2:
            return arr
        ansArr = [arr[0]]  # 记录某个数字结尾时最长的最长递增子序列，初始化第一个数字
        maxLen = [1]  # 下标i时，最长的递增子序列长度，初始化1
        for a in arr[1:]:
            if a > ansArr[-1]:  # 当前数字大于ansArr最后一个数字，子数组保持递增
                ansArr.append(a)
                maxLen.append(len(ansArr))
            # 当前数字小于等于ansArr最后一个数字，二分查找ansArr中第一个比当前数字大的下标
pos
            # 替换ansArr中下标pos的数字为当前数字，更新maxLen，记录当前最长递增子序列长度
为：pos + 1(下标+1)
            else:
                pos = bisect.bisect_left(ansArr, a)
                ansArr[pos] = a
                maxLen.append(pos + 1)
        # 找到的ansArr不一定是最终结果，[2,1,5,3,6,4,8,9,7] - > [1, 3, 4, 7, 9] (不是
最终结果)
        # [1, 1, 2, 2, 3, 3, 4, 5, 4] 从后往前遍历maxLen,依次找到等于len(arrLen)对应的
arr[i]
        ansLen = len(ansArr)
        for i in range(arrLen - 1, -1, -1):
            if maxLen[i] == ansLen:
                ansArr[ansLen - 1] = arr[i]
                ansLen -= 1
        return ansArr
```

21、！输出二叉树的右视图

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 求二叉树的右视图
# @param xianxu int整型一维数组 先序遍历
# @param zhongxu int整型一维数组 中序遍历
# @return int整型一维数组
#
class Solution:
    def solve(self , xianxu: List[int], zhongxu: List[int]) -> List[int]:
        result = []
        def dfs(p,i,level):
            if not p:
                return
            if level >= len(result):
                result.append(p[0])
            else:
                result[level] = p[0]
            tmp = i.index(p[0])
            dfs(p[1:tmp+1],i[0:tmp],level+1)
            dfs(p[tmp+1:], i[tmp+1:], level+1)
        dfs(xianxu, zhongxu, 0)
        return result
```

22、岛屿数量

1. 注意上下左右都得考虑
2. 小心[[0],[1],[0]]这种单个岛屿的情况

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 判断岛屿数量
# @param grid char字符型二维数组
# @return int整型
#
class Solution:
    def solve(self , grid: List[List[str]]) -> int:
        # write code here
        #判断周围是否有1
        #把1都变为0
        op = [(0,1),(1,0),(-1,0),(0,-1)]
        def f(row,col,grid):
            for op_row,op_col in op:
                if row+op_row >= 0 and row+op_row<len(grid) and col + op_col >=
0 and col +op_col < len(grid[0]) and grid[row+op_row][col+op_col] == "1":
                    grid[row+op_row][col+op_col] = "0"
                    f(row+op_row,col+op_col, grid)
        res = 0
        length_col = len(grid[0])
        length_row = len(grid)
        for i in range(length_row):
            for j in range(length_col):
```

```
                if grid[i][j] == "1":
                    res += 1
                    f(i,j,grid)
        return res
```

23、二叉树的最大深度

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param root TreeNode类
# @return int整型
#
class Solution:
    def maxDepth(self , root: TreeNode) -> int:
        # write code here
        if root is None:
            return 0
        queue = [root]
        res = 0
        while queue:
            length = len(queue)
            res += 1
            for i in range(length):
                node = queue.pop(0)
                if node.left is not None:
                    queue.append(node.left)
                if node.right is not None:
                    queue.append(node.right)
        return res
```

24、判断是否为回文字符串

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param str string字符串 待判断的字符串
# @return bool布尔型
#
class Solution:
    def judge(self , str: str) -> bool:
        # write code here
        length = len(str)
        n = length//2
        flag = True
        for i in range(n):
            if str[i] != str[-(i+1)]:
                flag = False
```

```
                break
        return flag
```

25、数组中出现次数超过一半的数字

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param numbers int整型一维数组
# @return int整型
#
class Solution:
    def MoreThanHalfNum_Solution(self , numbers: List[int]) -> int:
        # write code here
        set_ = set()
        length = len(numbers)
        req = length//2
        for i in numbers:
            if i not in set_:
                set_.add(i)
                if numbers.count(i)>req:
                    return i
            else:
                continue
```

26、矩阵的最小路径和

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param matrix int整型二维数组 the matrix
# @return int整型
#
class Solution:
    def minPathSum(self , matrix: List[List[int]]) -> int:
        row = len(matrix)
        col = len(matrix[0])
        dp = [[0]*col for i in range(row)]
        dp[0][0] = matrix[0][0]
        for i in range(1,col):
            dp[0][i] = matrix[0][i] + dp[0][i-1]
        for i in range(1,row):
            dp[i][0] = matrix[i][0] + dp[i-1][0]
        for i in range(1,col):
            for j in range(1,row):
                tmp = min(dp[i-1][j],dp[i][j-1])
                dp[i][j] = tmp+matrix[i][j]
        return dp[row-1][col-1]
```

27、表达式求值

```python
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 返回表达式的值
# @param s string字符串 待计算的表达式
# @return int整型
#
class Solution:
    def solve(self , s: str) -> int:
        # write code here
        def cal(num1,num2,sign):
            if sign == "+":
                return num1 + num2
            if sign == "-":
                return num2 - num1
            if sign == "*":
                return num2 * num1
        def judge_is_d(tmp):
            if tmp == "+" or tmp == "-" or tmp == "*" or tmp == "(" or tmp ==
")":
                return False
            else:
                return True
        #数字栈
        stack1 = []
        #符号栈
        stack2 = []
        tmp_sum = ""
        list_s = []
        s = list(s)
        for i in range(len(s)):
            if judge_is_d(s[i]):
                tmp_sum += s[i]
                if i == len(s)-1 or not judge_is_d(s[i+1]):
                    list_s.append(tmp_sum)
                    tmp_sum = ""
                    continue
            if not judge_is_d(s[i]):
                list_s.append(s[i])
        s = list_s
        while s:
            tmp = s.pop(0)
            if tmp == "*" or tmp == "(" or tmp == "+" or tmp == "-":
                stack2.append(tmp)
            elif tmp == ")":
                while stack2[-1] != "(":
                    num1 = int(stack1.pop())
                    num2 = int(stack1.pop())
                    sign = stack2.pop()
                    res = cal(num1, num2, sign)
                    stack1.append(res)
                stack2.pop()
            elif judge_is_d(tmp) and stack2 and stack2[-1] == "*":
                num1 = int(tmp)
                num2 = int(stack1.pop())
                sign = stack2.pop()
                res =  cal(num1, num2, sign)
                stack1.append(res)
```

```
            elif judge_is_d(tmp) and stack2 and stack2[-1] == "-":
                num1 = int(tmp)
                num2 = int(stack1.pop())
                sign = stack2.pop()
                res =  cal(num1, num2, sign)
                stack1.append(res)
            else:
                stack1.append(tmp)
        while stack2:
            num1 = int(stack1.pop())
            num2 = int(stack1.pop())
            sign = stack2.pop()
            res = cal(num1, num2, sign)
            stack1.append(res)
        return stack1.pop()
```

28、!字符串出现次数的TopK问题

1. 字典二次排序
2. lambda函数

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# return topK string
# @param strings string字符串一维数组 strings
# @param k int整型 the k
# @return string字符串二维数组
#
class Solution:
    def topKstrings(self , strings: List[str], k: int) -> List[List[str]]:
        dic = {}
        for i in strings:
            if i not in dic:
                dic.setdefault(i,0)
            if i in dic:
                dic.update({i:dic.get(i)+1})
        return sorted(dic.items(),key=lambda x:(-x[1],x[0]))[:k]
```

29、判断一个链表是否为回文结构

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param head ListNode类 the head
# @return bool布尔型
#
class Solution:
    def isPail(self , head: ListNode) -> bool:
        # write code here
```

```
        cur = head
        list_ = [head.val]
        while cur:
            cur = cur.next
            if cur:
                list_.append(cur.val)
        length = len(list_)
        n = length//2
        flag = True
        for i in range(n):
            a = list_[i]
            b = list_[-i-1]
            if a != b:
                return not flag
        return flag
```

30、!!不同路径的数目(一)

1. python用除号`/`会变成小数

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param m int整型
# @param n int整型
# @return int整型
#
from functools import reduce
class Solution:
    def uniquePaths(self , m: int, n: int) -> int:
        # write code here
        count = m + n -2
        a = reduce(lambda x,y:x*y,range(count,count-(m-1),-1),1)
        b = reduce(lambda x,y:x*y,range(m-1,0,-1),1)
        res = int(a/b)
        return res
```

31、合并区间

```
# class Interval:
#     def __init__(self, a=0, b=0):
#         self.start = a
#         self.end = b

# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
# @param intervals Interval类一维数组
# @return Interval类一维数组
#
class Solution:
    def merge(self , intervals: List[Interval]) -> List[Interval]:
        if not intervals:
            return []
        intervals = sorted(intervals,key=lambda x:(x.start,x.end))
        res = []
```

```
        res.append(intervals[0])
        for i in intervals[1:]:
            if res[-1].start <= i.start<=res[-1].end and i.end > res[-1].end:
                node = res.pop()
                new_node = Interval(node.start,i.end)
                res.append(new_node)
            elif res[-1].start <= i.start and i.start<= res[-1].end:
                    pass
            else:
                res.append(i)
        return res
```

32、矩阵元素查找

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param mat int整型二维数组
# @param n int整型
# @param m int整型
# @param x int整型
# @return int整型一维数组
#
class Solution:
    def findElement(self , mat: List[List[int]], n: int, m: int, x: int) ->
List[int]:
        # write code here
        row = 0

        for i in mat:
            row += 1
            if i[0]<=x and i[-1]>=x:
                col = 0
                for j in i:
                    if j != x:
                        col += 1
                    if j == x:
                        return [row-1,col]
```

33、链表的奇偶排序

1. 偶链表最后一个记得归None `tmp2.next = None`

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param head ListNode类
# @return ListNode类
#
```

```python
class Solution:
    def oddEvenList(self , head: ListNode) -> ListNode:
        # write code here
        if not head or not head.next:
            return head
        node_head1 = tmp1 = head
        node_head2 = tmp2 = head.next
        cur = node_head2.next
        flag = True
        while cur:
            if flag:
                tmp1.next = cur
                tmp1 = tmp1.next
            if not flag:
                tmp2.next = cur
                tmp2 = tmp2.next
            flag = not flag
            cur = cur.next
        tmp2.next = None
        tmp1.next = node_head2
        return node_head1
```

34、顺时针旋转矩阵

```python
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param mat int整型二维数组
# @param n int整型
# @return int整型二维数组
#1 2 3    7 4 1  (2,0)  (1,0)  (0,0)
#4 5 6    8 5 2  (2,1)  (1,1)  (0,1)
#7 8 9    9 6 3  (2,2)  (1,2)  (0,2)
class Solution:
    def rotateMatrix(self , mat: List[List[int]], n: int) -> List[List[int]]:
        # write code here
        list2 = [[0]*n for i in range(n)]
        for i in range(n):
            for j in range(n):
                list2[i][j] = mat[n-1-j][i]
        return list2
```

35、加起来和为目标值的组合(二)

```python
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param num int整型一维数组
# @param target int整型
# @return int整型二维数组
#
class Solution:
    def combinationSum2(self , num: List[int], target: int) -> List[List[int]]:
```

```
        def function(num,target,res,tmp,start):
            if target == 0:
                #这里一定要这样写(tmp[:])，不然后续tmp的值改变会影响res里添加列表的值
                res.append(tmp[:])
                return
            if target<0 or start>=len(num) :
                return
            for i in range(start,len(num)):
                if i>start and num[i] == num[i-1]:
                    continue
                if num[i]>target:
                    break
                if num[i] <= target:
                    tmp.append(num[i])
                    function(num, target-num[i], res, tmp,i+1)
                    tmp.pop()
        if not num:
            return []
        res = []
        tmp = []
        num.sort()
        function(num, target, res, tmp, 0)
        return res
```

36、验证IP地址

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 验证IP地址
# @param IP string字符串 一个IP地址字符串
# @return string字符串
#
class Solution:
    def solve(self , IP: str) -> str:
        # write code here
        def is_ipv4(str1:str):
            try:
                list1 = str1.split(".")
                if len(list1)!=4:
                    return False
                for i in list1:
                    if i.startswith("0") and i != "0":
                        return False
                    if int(i) >= 0 and int(i)<=255:
                        pass
                    else:
                        return False
                return True
            except:
                return False
        def is_ipv6(str2:str):
            list_zimu = ["A","B","C","D","E","F","a","b","c","d","e","f"]
            try:
                list2 = str2.split(":")
                if len(list2)!=8:
```

```
                    return False
                for i in list2:
                    if len(i)!=4 and i != "0":
                        return False
                    for j in list(i):
                        if j not in list_zimu and not j.isdigit():
                            return False
                return True
            except:
                return False
        if is_ipv4(IP):
            return "IPv4"
        if is_ipv6(IP):
            return "IPv6"
        else:
            return "Neither"
```

37、二进制中1的个数

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
#
# @param n int整型
# @return int整型
#9//2 = 4....1
class Solution:
    def NumberOf1(self , n: int) -> int:
        def f(n):
            res = 0
            while n:
                tmp = n%2
                if tmp == 1:
                    res += 1
                n = n//2
            return res
        if n>=0:
            return f(n)
        if n<0:
            n = abs(n)
            n = n-1
            return 32-f(n)
```

38、最大正方形

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 最大正方形
# @param matrix char字符型二维数组
# @return int整型
#
class Solution:
    def solve(self , matrix: List[List[str]]) -> int:
        # write code here
```

```
        #10100
        #10111
        #11111
        #10111
        if not matrix:
            return 0
        res = 0
        row = len(matrix)
        col = len(matrix[0])
        dp = [[0]*col for i in range(row)]
        for i in range(row):
            for j in range(col):
                if matrix[i][j] == "1":
                    dp[i][j] = 1
                    if j-1>=0 and i-1>=0 and dp[i-1][j-1]>0:
                            for z in range(dp[i-1][j-1]):
                                if matrix[i-z-1][j]!="1" or matrix[i][j-z-
1]!="1":
                                    break
                                dp[i][j]+=1
                            if dp[i][j]>res:
                                res = dp[i][j]
                    else:
                        if dp[i][j]>res:
                            res = dp[i][j]
                else:dp[i][j] = 0
        return res*res
```

39、kmp算法

```
#
# 代码中的类名、方法名、参数名已经指定，请勿修改，直接返回方法规定的值即可
#
# 计算模板串S在文本串T中出现了多少次
# @param S string字符串 模板串
# @param T string字符串 文本串
# @return int整型
#
# ababa
# 00123
# 00012
# aaabcdaaaa
# 0120001233
class Solution:
    def kmp(self , S: str, T: str) -> int:
        def get_next(string:str):
            length = len(string)
            dp = [0]*length
            right = 0
            for i in range(1,length):
                while right>0 and string[i] != string[right]:
                    right = string[right-1]
                if string[i] == string[right]:
                    right += 1
                    dp[i] = right
            return dp
```

```
        length_T = len(T)
        length_S = len(S)
        T_next = get_next(T)
        i = 0
        j = 0
        res = 0
        while i < length_T:
             while j < length_S:
                    if S[j] == T[i]:
                        i += 1
                        j += 1
                        if j == length_S:
                            res += 1
                            j = T_next[j-1]
                            break
                        if i == length_T:
                            break
                    else:
                        j = T_next[j]
                        if j == 0:
                            i += 1
        return res
```

40、将列表中的元素转为字符串

```
a = [1,2,3,4]
a = list(map(str,a))
print(a)
```

```
[1, 2, 3, 4]
['1', '2', '3', '4']
```

41、质数因子

```
import math
while True:
    try:
        a = int(input())
        res = []
        for i in range(2,int(math.sqrt(a)+1)):
            while a%i == 0:
                res.append(i)
                a = a//i
        if a>=2:
            res.append(a)
        res = list(map(str,res))
        print(" ".join(res))
    except:
        break
```

42、斗地主之顺子

```
# 序号1
# 题目：斗地主之顺子
# 在斗地主扑克牌游戏中，扑克牌儿由小到大的顺序为：3,4,5,6,7,8,9,10,J,Q,K,A,2,玩家可以出的扑克牌阵型有单张、对子、顺子、飞机、炸弹等。其中的顺子的出牌规则为:由至少5张由小到大连续递增的扑克牌组成，且不能包含2，例如:{3,4,5,6,7}、{3,4,5,6,7,8,9,10,J,Q,K,A}都是有效的顺子;而{J,Q,K,A,2}、{2,3,4,5,6}、{3,4,5,6}、{3,4,5,6,8}等都不是顺子。
# 给定一个包含13张牌的数组，如果有满足出牌规则的顺子，请输出顺子。
# 如果存在多个顺子，请每行输出一个顺子，且需要按顺子的第一张牌的大小（必须从小到大）依次输出。
# 如果没有满足出牌规则的顺子，请输出NO。
#
# 输入描述:
# 13张任意顺子的扑克牌，每张扑克牌数字用空格隔开，每张扑克牌的数字都是合法的，并且不包括大小王:
# 2 9 J 2 3 4 K A 7 9 A 5 6
# 不需要考虑输入为异常字符的情况。
# 输出描述:
# 组成的顺子，每张扑克牌数字用空格隔开:
# 3 4 5 6 7
#
# 示例1:
# 输入
# 2 9 J 2 3 4 K A 7 9 A 5 6
#
# 输出
# 3 4 5 6 7
def no_1(input1):
    op = ["3","4","5","6","7","8","9","10","J","Q","K","A"]
    def inner(length, poker):
        poker_list = []
        for tmp in op:
            if tmp in poker:
                poker_list.append(tmp)
        op_str = "".join(op)
        res = []
        for start in range(len(poker_list)-length+1):
            for end in range(start+length,len(poker_list)):
                if "".join(poker_list[start:end]) in op_str:
                    res.append(poker_list[start:end])
                else:
                    break
        return res
    res = inner(5,input1)
    return res
if __name__ == '__main__':
    input1 = [
        ["2", "9", "J", "2", "3", "4", "K", "A", "7", "9", "A", "5", "6"],
        ["2", "9", "J", "2", "3", "4", "10", "8", "7", "9", "A", "5", "6"]
    ]
    for tmp in input1:
        res = no_1(tmp)
        print(res)
```

43、猜密码

```
# 序号2
# 题目：猜密码
# 小杨申请了一个保密柜，但是他忘记了密码，只记得密码都是数字，而且所有数字都是不重复的。请你根据他记住的数字范围和密码的最小数字数量，帮他算一下有哪些可能的组合。规则如下：
# 1、输出的组合都是从可选的数字范围中选取的，且不能重复。
# 2，输出的密码数字要按照从小到大的顺序排列，密码组合需要按照字母顺序，从小到大的顺序排序。
# 3、输出的每一个组合的数字的数量要大于等于密码最小数字数量；
# 4、如果可能的组合为空，则返回“None”
#
# 输入描述：
# 1、输入的第一行是可能的密码数字列表，数字间以半角逗号分隔
# 2、输入的第二行是密码最小数字数量
#
# 输出描述：
# 可能的密码组合，每种组合显示成一行，每个组合内部的数字以半角逗号分隔，以从小到大的顺序排列。输出的组合间需要按照字典排序。
# 比如：
# 2,3,4放到2,4的前面
#
# 备注：
# 字典序是指按照单词出现在字典的顺序进行排序的方法，比如：
# a排在b前
# a排在ab前
# ab排在ac前
# ac排在aca前
#
# 示例1：
# 输入
# 2,3,4
# 2
#
# 输出
# 2,3
# 2,3,4
# 2,4
# 3,4

def No_2(input1,input2):
    #空返回为None
    if not input1:
        return None
    #按半角英文切分
    list = input1.split(",")
    list.sort()
    def digital_com(num_list,number, tmp,res,length):
        if len(tmp) >= number:
            res.append(tmp[:])
            if len(tmp) == length:
                return
        for i in range(len(num_list)):
            tmp.append(num_list[i])
            #递归
            digital_com(num_list[i+1:],number,tmp,res,length)
            tmp.pop()
    tmp = []
    res = []
    length = len(input1)
```

```
    digital_com(list,input2,tmp,res,length)
    res.sort()
    return res
if __name__ == '__main__':
    input1 = "1,2,3,4,5,6,7,8,9"
    input2 = 4
    res = No_2(input1,input2)
    print(res)
```

44、IPv4地址转换成整数

```
# 序号3
# 题目：IPv4地址转换成整数
# 存在一种虚拟IPv4地址，由4小节组成，每节的范围为0～128，以# 号间隔，格式如下：
# （1～128）# （0～255）# （0～255）# （0～255）
# 请利用这个特性把虚拟IPv4地址转换为一个32位的整数，IPv4地址以字符串形式给出，要求每个IPv4地址只能对应到唯一的整数上，如果是非法IPv4，返回invalid IP
# 输入描述：
#  输入一行，虚拟IPv4地址格式字符串
# 输出描述：
# 输出以上，按照要求输出整型或者特定字符
# 备注：输入不能确保是合法的IPv4地址，需要对非法IPv4（空串，含有IP地址中不存在的字符，非合法的
# 分十进制，十进制整数不在合法区间内）进行识别，返回特定错误
#
# 示例1：
# 输入
#  100#101#1#5
# 输出
#  1684340997
def Ipv4_to_int(input: str):
    #原ipv4没有用#分割或者为空串返回invalid IP
    if "#" not in input:
        return "invalid IP"
    list = input.split("#")
    if len(list) != 4:
        return "invalid IP"
    bin_com = ""
    for index,val in enumerate(list):
        #分割不是数字返回invalid IP
        if not val.isdigit():
            return "invalid IP"
        #确定第一小节在(1~128)
        if index == 0 and (int(val)<1 or int(val) > 128):
            return "invalid IP"
        #确定后面的三小节在(0~255)
        if index>0 and (int(val)<0 or int(val)>255):
            return "invalid IP"
        tmp = bin(int(val))
        tmp = tmp[2:]
        bin_com = bin_com + ("0"*(8-len(tmp))+tmp)
    return int(bin_com,2)
if __name__ == '__main__':
    input1 = "100#101#1#5"
    res =  Ipv4_to_int(input1)
    print(res)
```

```
    input1 = "128#255#1#1"
    res = Ipv4_to_int(input1)
    print(res)
```

45、英文输入法

```
# 序号：4
# 标题：英文输入法
# 主管期望你来实现英文输入法单词联想功能。需求如下：
# 依据用户输入的单词前缀，从已输入的英文语句中联想出用户想输入的单词，按字典序输出联想到的单词序列，如果联想不到，请输出用户输入的单词前缀。
# 注意：
# 1.英文单词联想时，区分大小写，
# 2.缩略形式如“don't”，判定为两个单词，“Don”和“t”
# 3.输出的单词序列，不能有重复单词，且只能是英文单词，不能有标点符号。
# 输入描述：
#
# 输入为两行。
# 首行输入一段由英文单词word和标点符合组成的语句str；
# 接下来一行为一个英文单词前缀pre。
# 0<word.length()<=20
# 0<str.length()<=10000
# 0<pre<=20
#
# 输出描述：
# 输出符合要求的单词序列或单词前缀，存在多个时，单词之间以单个空格分割
#
# 示例1：
# 输入
# I love you
# He
#
# 输出
# He
import re
import bisect
def No_4(string1:str,word_pre:str):
    #将非单词字符串作为分隔符
    split_list = re.split("\W+", string1)
    split_list.sort()
    print(split_list)
    res_set = set()
    #用二分法查找第一个匹配的位置
    first_index = bisect.bisect_left(split_list,word_pre)
    #如果没匹配上返回前缀
    if first_index == len(split_list):
        return word_pre
    else:
        flag = True
        for i in split_list[first_index:]:
            if i.startswith(word_pre):
                res_set.add(i)
                # 第一次匹配成功后将flag设为True
                flag = True
            #继第一次匹配后出现没匹配上终止循环
            if flag and not i.startswith(word_pre):
```

```
                break
    res_list = list(res_set)
    res_list.sort()
    return " ".join(res_list)
if __name__ == '__main__':
    string1 = "I Don't know,just do it,jast,jc,jd,je,jf,jg"
    word_pre = "j"
    res1 = No_4(string1,word_pre)
    print(res1)
    word_pre2 = "z"
    res2 = No_4(string1,word_pre2)
    print(res2)
```

46、玩牌高手

```
# 序号：5
# 标题：玩牌高手
# 给定一个长度为n的整型数组，表示一个选手在n轮内可选择的牌面分数。选手基于规则选牌，请计算所有轮结束后其可以获得的最高总分数。选择规则如下：
# 1.在每轮里选手可以选择获取该轮牌面，则其总分数加上该轮牌面分数，为其新的总分数。
# 2.选手也可以不选择本轮牌面直接跳转到下一轮，此时将当前总分数还原为3轮前的总分数，若当前轮次小于等于3（即在第1、2、3轮选择跳过轮次），则总分数置为0。
# 3.选手的初始总分数为0，且必须依次参加每一轮。
# 输入描述：
#
# 第一行为一个小写逗号分割的字符串，表示n轮的牌面分数，1<=n<=20。
# 分数值为整数，-100<=分数值<=100。
# 不考虑格式问题。
#
# 输出描述：所有轮结束后选手获得的最高总分数。
#
# 示例1：
# 输入
# 1，-5，-6，4，3，6，-2
#
# 输出
#  11
def No_5(num_list:list):
    length = len(num_list)
    #每轮对应的分数
    dp = [0]*length
    for index,val in enumerate(num_list):
        #前三轮考虑 拿牌得分[dp[index-1]+val] 和 弃牌后[0分] 作比较
        if index <= 2:
            if index > 0 :
                dp[index] = max(0,dp[index-1] + val)
            #第一轮单独考虑，防止index-1越界
            elif index == 0:
                dp[index] = max(val,0)
        #三轮后考虑 拿牌得分[dp[index-1]+val] 和 回溯到3轮前的分数[dp[index-3]] 作比较
        elif index > 2:
            dp[index] = max(dp[index-3],dp[index-1]+val)
    print("原数组为    %s"%num_list)
    print("对应的分数为%s"%dp)
    return dp[-1]
if __name__ == '__main__':
```

```
    list = [1,-5,-6,4,3,6,-2]
    res = No_5(list)
    print("最后得分:",res)
    print("==================================")
    list = [1,2,3,-50,-50,0]
    res = No_5(list)
    print("最后得分:",res)
    print("==================================")
    list = [1,2,3,100,-99,-99,2]
    res = No_5(list)
    print("最后得分:",res)
    print("==================================")
```

47、找单词

```
# 序号6
# 标题：找单词
# 给一个字符串和一个二维码字符数组，如果该字符串存在于该数组中。则按字符串的字符顺序输出字符串每个字符所在单元格的位置下标字符串，如果找不到返回字符串“N”
# 1.需要按照字符串的字符组成顺序搜索，且搜索到的位置必须是相邻单元格，其中“相邻单元格”是指那些水平相邻或垂直相邻的单元格。
# 2.同一个单元格内的字母不允许被重复使用。
# 3.假定在数组中最多只存在一个可能得匹配。
# 输入描述
#
# 1.第一行为一个数字（N）指示二维数组在后续输入所占的行数。
# 2.第2行到第N+1行输入为一个二维大写字符数组，每行字符用半角,分割。
# 3.第N+2行为待查找的字符串，由大写字符组成。
# 4.二维数组的大小为N*N,
# 0＜N＜＝100。
# 5.单词长度k，0＜k＜1000。
#
# 输出描述：
#
# 输出一个位置下标字符串，拼接格式为：第1个字符行下标+“,”+第1个字符列下标+“,”+第2个字符行下标+“,”+第2个字符列下标.....+“,”+第N个字符行下标+“,”+第N个字符列下标
#
# 示例1：
# 输入
# 4
# A,C,C,F
# C,D,E,D
# B,E,S,S
# F,E,C,A
# ACCESS
#
# 输出
# 0,0,0,1,0,2,1,2,2,2,2,3
def no_6(input_matrix,match_str):
    row_len = len(input_matrix)
    col_len = len(input_matrix[0])
    op = [(0,1),(0,-1),(1,0),(-1,0)]

    def back_tracking(row, col, path, index):
        if "".join(path) == match_str:
            return True
```

```
        if input_matrix[row][col] != match_str[index]:
            return False
        for op_row, op_col in op:
            row += op_row
            col += op_col
            if row in range(row_len) and col in range(col_len) and (row,col) not
in res:
                res.append((row,col))
                path.append(input_matrix[row][col])
                if back_tracking(row, col, path , index+1):
                    return True
                row -= op_row
                col -= op_col
                path.pop()
                res.pop()
            else:
                row -= op_row
                col -= op_col
    for r in range(row_len):
        for c in range(col_len):
          if input_matrix[r][c] ==  match_str[0]:
              res = [(r,c)]
              if back_tracking(r, c, [input_matrix[r][c]], 0):
                  return res
if __name__ == '__main__':
    input_matrix = [
        ['C', 'C', 'C', 'F', 'k'],
        ['C', 'D', 'E', 'C', 'K'],
        ['B', 'M', 'D', 'O', 'D'],
        ['F', 'E', 'F', 'A', 'E'],
        ['F', 'F', 'B', 'B', 'E']
    ]
    res = no_6(input_matrix, 'DECODE')
    print(res)
```

48、两数之和绝对值最小

```
# 序号7
# 标题：两数之和绝对值最小
#
# 【两数之和绝对值最小】给定一个从小到大的有序整数序列（存在正整数和负整数）数组nums，请你在该数
字中找出两个数，其和的绝对值（|nums[x]+nums[y]|)为最小值,并返回这个绝对值。
#
# 每种输入只会对应一个答案，但是数组中同一个元素不能使用两遍。
# 输入描述：
# 一个通过空格分割的有序整数序列字符串，最多1000个整数，且整数数值范围内是-65535到～65535。
#
# 输出描述：
# 两数之和绝对值最小值
#
# 示例1：
# 输入：
# -3 -1 5 7 11 15
#
# 输出
# 2
```

```python
def No_7(num_list):
    length = len(num_list)
    res = float('inf')
    if length<2:
        return

    flag = True
    # 数组同时存在正整数和负整数
    if num_list[0]<0 and num_list[-1]>0:
        for start in range(length-1):
            for end in range(start+1,length-1):
                tmp = abs(num_list[start]+num_list[end])
                #如果在连续变小则继续循环，否则跳出循环
                if tmp<abs(num_list[start]+num_list[end+1]):
                    res = min(res,tmp)
                    flag = False
                    break
            #如果到最后都在连续变小，再比较一下 res 和 起始值和数组最后一位值相加的绝对值
            if flag:
                res = min(res,abs(num_list[start]+num_list[-1]))
            flag = True
        return res
    #数组没有负整数
    elif num_list[0]>=0:
        return num_list[0]+num_list[1]
    #数组没有正整数
    elif num_list[-1]<=0:
        return -num_list[-1] -num_list[-2]
if __name__ == '__main__':
    list_all = [[-3,-1,5,7,11,15],#2
                [-100,99,100,101],#0
                [-10,-6,-3,1,4],#1
                [1,3,5,8,20],#4
                [-100,-90,-55,-54,-10,-1],#11
                [-100,-50,1,2],#3
                [-2,-1,0,100,200]#1
                ]
    for test in list_all:
        print(No_7(test))
```

49、找朋友

```python
# 序号8
# 标题：找朋友
# 在学校中，N个小朋友站成一队，第i个小朋友的身高为height[i]，第i个小朋友可以看到的第一个比自己身高更高的小朋友j。那么j是i的好朋友(要求j>i），请重新生成一个列表，对应位置的输出是每个小朋友的好朋友位置，如果没有看到好朋友，请在该位置用0代替。
# 小朋友人数范围是[0,40000]，
#
# 输入描述：
# 第一行输入N，N表示有N个小朋友
# 第二行输入N个小朋友的身高height[i],都是整数
#
# 输出描述：
# 输出N个小朋友的好朋友的位置
#
```

```
# 示例1：
# 输入
#
# 2
# 100 95
#
# 输出
#
# 0 0
def No_8(children_list):
    #递减栈，栈内记录小朋友索引
    stack = []
    length = len(children_list)
    res = [0]*length
    for key,val in enumerate(children_list):
        #保证栈内递减，当前值大于栈顶值时，将当前索引添加到栈顶索引，并弹出栈顶索引
        while stack and val > children_list[stack[-1]]:
            res[stack[-1]] = key
            stack.pop()
        #当前索引入栈
        stack.append(key)
    return res
if __name__ == '__main__':
    list = [
        [100,95,90,85,80],#0 0 0 0 0
        [4,3,2,1,3,5,6],#5 5 4 4 5 6 0
        [1,2,3,2,1]#1 2 0 0 0
    ]
    for item in list:
        print(No_8(item))
```

50、流水线

```
# 序号9
# 标题：流水线
# 一个工厂有m条流水线来并行完成n个独立的作业，该工厂设置了一个调度系统，在安排作业时，总是优先执行处理时间最短的作业。现给定流水线个数m，需要完成的作业个数作业数n，每个作业的处理时间分别为t1,t2...tn。请你编程计算处理完所有作业的耗时多少？
# 当n＞m时，首先处理时间短的m个作业进入流水线，其他的等待，当某个作业完成时，依次从剩余作业中取处理时间最短的进入处理。
#
# 输入描述
# 第一行为2个整数（采用空格分隔），分别表示流水线的个数m，和作业数n；
# 第二行输入n个整数（采用空格分隔），表示每个作业的处理时长t1,t2....tn。
# 0＜m,n＜100，0＜t1,t2...tn＜100。
# 注：保证输入都是合法的。
#
# 输出描述：
#
# 输出处理完所有作业的总时长
#
# 示例1：
# 输入
#
# 3 5
# 8 4 3 2 10
```

```
#
# 输出
#
# 13
def No_9(input1:str,input2:str):
    input1_list = input1.split(" ")
    tasks_list = list(map(int,input2.split(" ")))
    tasks_list.sort()
    pipLine = int(input1_list[0])
    tasks_num = int(input1_list[1])
    #for循环次数
    length = tasks_num//pipLine
    mod = tasks_num%pipLine
    #如果不能整除，循环次数加一
    if mod:
        length+=1
    #总时长
    total_time = 0
    for count in range(length):
        total_time += tasks_list[mod-1]
        mod += pipLine
    return total_time

if __name__ == '__main__':
    input1_list = ["3 5",
                   "10 5",
                   "3 9",
                   "1 1",
                   ]
    input2_list = ["8 4 3 2 10",#13
                   "1 2 3 4 5",#5
                   "9 2 3 4 5 6 7 8 1",#18
                   "10",#10
                   ]
    for input1,input2 in zip(input1_list,input2_list):
        res = No_9(input1,input2)
        print(res)
```

51、磁盘容量排序

```
# 序号10
# 标题：磁盘容量排序
#       磁盘的容量单位常用的有M、G、T这三个等级，它们之间的换算关系为1T=1024G，1G=1024M。现在给定n块磁盘的容量，请对它们按从小到大的顺序进行稳定排序。例如，给定5块盘的容量，1T，20M，3G，10G6T，3M12G9M，排序后的结果为，20M，3G，3M12G9M，1T，10G6T。注意单位可以重复出现，上述3M12G9M表示的容量即为3M+12G+9M，和12M12G相等。
#
# 输入描述：
# 输入第一行包含一个整数n（2<=n<=100），表示磁盘的个数，接下的n行，每行一个字符串（长度大于2,小于30），表示磁盘的容量，由一个或多个格式为mv的子串组成，其中m表示容量大小，v表示容量单位，例如20M，1T，30G，10G6T，3M12G9M。
#     磁盘容量m的范围为1到1024的正整数，容量单位v的范围只包含题目中提到的M,G,T三种，换算关系如题目描述。
#
# 输出描述：
#
```

```
# 输出n行，表示n块磁盘容量排序后的结果。
#
# 示例1:
# 输入
# 3
# 1G
# 2G
# 1024M
#
# 输出
# 1G
# 1024G
# 2G
def No_10(input1:int,input2:list[str]):
    dic = {}
    for disk in input2:
        val = disk
        #将磁盘的单位全用换算成M的大小
        val =
val.replace("M","*1+").replace("G","*1024+").replace("T","*1024*1024+").rstrip("
+")
        #键为原本磁盘容量的表示，值为换算成M大小的容量
        dic.setdefault(disk,eval(val))
    res  = [disk[0] for disk in sorted(dic.items(),key=lambda x:(x[1]))]
    return res

if __name__ == '__main__':
    input1 = [3,5]
    input2 = [["1G","2G","1024M"],#['1G', '1024M', '2G']
              ["1T","20M","3G","10G6T","3M12T9M"]#['20M', '3G', '1T', '10G6T',
'3M12T9M']
              ]
    for x,y in zip(input1,input2):
        res = No_10(x,y)
        print(res)
```

52、【We are a team】

```
# 序号：11
# 标题：【We are a team】
#    总共n个人在机房，每个人有一个标号（1<=标号<=n），他们分成了多个团队，需要你根据收到的m
条消息判定指定的两个人是否在一个团队中，具体的：
# 1、消息构成为:a b c，整数a、b分别代表了两个人的标号，整数c代表指令。
# 2、c==0代表a和b在一个团队内。
# 3、c==1代表需要判定a和b的关系，如果a和b是一个团队，输出一行“we are a team”，如果不是，输出
一行“we are not a team ”。
# 4、c为其他值，或当前行a或b超出1～n的范围，输出“da pian zi”。
#
# 输入描述：
# 1、第一行包含两个整数n、m（1<=n，m<=100000），分别表示有n个人和m条消息。
# 2、随后的m行，每行一条消息，消息格式为：a b c（1<=a，b<=n，0<=c<=1）。
#
# 输出描述：
#
# 1、c==1时，根据a和b是否在一个团队中输出一行字符串，在一个团队中输出“we are a team”，不在一
个团队中输出“we are not a team”。
```

```python
# 2、c为其他值，或当前行a或b的标号小于1或者大于n时，输出字符串“da pian zi”。
# 3、如果第一行n和m的值超出约定的范围时，输出字符串“NULL”
#
# 示例1:
#
# 输入
# 5 6
# 1 2 0
# 1 2 1
# 1 5 0
# 2 3 1
# 2 5 1
# 1 3 2
#
# 输出
#
# we are a team
# we are not a team
# we are a team
# da pian zi
def No_11(n:int,m:int,message_list:list[list[int]]):
    #n和m的值超出约定的范围时，输出字符串“NULL”
    if n < 1 or n > 100000 or m < 1 or m > 100000:
        print("NULL")
        return
    def get_root(i):
        while i != node[i]:
            i = node[i]
        return i
    #从1开始，索引为节点，索引对应的值意义为父节点
    node = list(range(n+1))
    for unit_list in message_list:
        a,b,c = unit_list
        #
        if c not in [0,1] or a < 1 or a > n or b < 1 or b > n :
            print("da pian zi")
            continue
        elif c == 0:
            a_root = get_root(a)
            b_root = get_root(b)
            node[a_root] = b_root
        elif c == 1:
            a_root = get_root(a)
            b_root = get_root(b)
            if a_root == b_root:
                print("we are a team")
            else:
                print("we are not a team")
if __name__ == '__main__':
    n_input = [5,2,2,0]
    m_input = [6,3,2,2]
    message_list_input = [
                           [[1,2,0],
                            [1,2,1],#we are a team
                            [1,5,0],
                            [2,3,1],#we are not a team
                            [2,5,1],#we are a team
                            [1,3,2]#da pian zi
```

```
                        ],
                        #============================
                          [[1,2,0],
                           [2,1,0],
                           [1,2,1]],#we are a team
                        #============================
                          [[3,4,0],#da pian zi
                           [3,4,1]],# da pian zi
                        #============================
                          [[1,2,0],#n==0  ==>  NULL
                           [1,2,1]]
                          ]
    for m,n,message_list in zip(n_input,m_input,message_list_input):
        No_11(m,n,message_list)
        print("==================")
```

53、目录删除

```
# 序号12
# 题目：目录删除
# 某个文件系统由N个目录，每个目录都有一个独一无二的ID，每个目录只有一个父目录，但每个父目录下可以由0个或多个子目录，目录结构呈树状结构，假设，根目录的ID为0，且根目录没有父目录，其他所有目录的ID用唯一的正整数表示，并统一编号。
# 现给定目录ID和其父目录ID对应的父子关系表[子目录ID，父目录ID]，以及一个待删除的目录ID，请计算并返回一个ID序列，表示因为删除制定目录ID后剩下的所有目录，返回的ID序列以递增顺序输出。
#
# 注意：
# 1、被删除的目录旱或文件编号一定在输入的ID序列中
# 2、当一个目录删除时，它所有的子目录都会被删除
#
# 示例：
# 输入：
# [(8, 6), (10, 8), (6, 0), (20, 8), (2, 6)]
# 8
# 输出：
# 2 6
class Tree(object):
    def __init__(self):
        # 这个列表里装着所有目录id
        self.node_list = []
        # key是父目录id，value是子目录id列表
        self.node_dic = {}
        # 这个列表里装着所有会被删除的目录id
        self.node_remove = []
    #   不重复地在node_list里添加目录id
    def add(self, node_id):
        if node_id not in self.node_list:
            self.node_list.append(node_id)
    #   在字典node_dic添加目录节点关系
    def create_dic(self, seq):
        for (child, parent) in seq:
            # 同时收录在node_list里
            self.add(child)
            self.add(parent)
            self.node_dic.setdefault(parent,[]).append(child)
    #   遍历所有会被删除的目录id
```

```
    def rm_dir(self, rm_id):
        queue = [rm_id]
        while queue:
            tmp = queue.pop()
            if self.node_dic.get(tmp):
                #   添加目录的子目录
                queue = self.node_dic.get(tmp) + queue
            if tmp in self.node_remove:
                continue
            self.node_remove.append(tmp)
    #   遍历结果
    def show_dir(self):
        tmp = self.node_list
        tmp.sort()
        return [_ for _ in tmp[1:] if _ not in self.node_remove]
if __name__ == '__main__':
    input_dir = [
        [(8, 6), (10, 8), (6, 0), (20, 8), (2, 6),(11,10)],
        #           0
        #           |
        #           6
        #         /  \
        #       8     2       ===>  [2, 6]
        #     /  \
        #   10   20
        #   |
        #   11
        [(3, 1), (4, 1), (5, 1), (1, 0), (2, 0 )]
    ]
    input_rm_id = [
        8, 4
    ]
    for dir, rm_id in zip(input_dir,input_rm_id):
        tree = Tree()
        tree.create_dic(dir)
        tree.rm_dir(rm_id)
        res = tree.show_dir()
        print(res)
```

54、服务器失效判断

```
# 序号13
# 服务器失效判断
# 某系统中有众多服务，每个服务用字符串（只包含字母和数字，长度<=10)唯一标识，服务间可能有依赖关系，如A依赖B，则当B故障时导致A也故障。
# 依赖具有传递性，如A依赖B，B依赖C，当C故障的时候会导致B故障，也会导致A故障。
# 给出所有依赖关系，以及当前已知的故障服务，要求输出所有正常的服务。
# 依赖关系：服务1-服务2表示“服务1”依赖“服务2”
# 不必考虑输入异常，用例保证：依赖关系列表、故障列表非空，且依赖关系数，故障服务数都不会超过3000，服务标识格式正常。
# 输入描述：
# 半角逗号分隔的依赖关系列表（换行）
# 半角逗号分隔的故障服务列表
# 输出描述：
# 依赖关系列表中提及的所有服务中可以正常工作的服务列表，用半角逗号分隔，按依赖关系列表中出现的次序排序。特别的，没有正常节点输出单独一个半角逗号。
```

```
#
# 示例1:
# 输入:
# a1-a2,a5-a6,a2-a3
# a5,a2
# 输出:
# a6,a3
class Tree(object):
    def __init__(self):
        self.services = []
        self.invalid_services = []
        self.service_map = {}

    def add(self,service):
        if service not in self.services:
            self.services.append(service)
    def create_tree(self,seq):
        for slave, leader in seq:
            self.add(slave)
            self.add(leader)
            self.service_map.setdefault(leader,[]).append(slave)
    def update_invalid_services(self,inv_service):
        queue = inv_service
        while queue:
            tmp = queue.pop()
            if tmp in self.invalid_services:
                continue
            if self.service_map.get(tmp):
                queue = self.service_map.get(tmp) + queue
            self.invalid_services.append(tmp)
    def show_live_services(self):
        res = [_ for _ in self.services if _ not in self.invalid_services]
        if res:
            return ",".join(res)
        return ","
if __name__ == '__main__':
    all_cases = [
        ([('a1', 'a2'), ('a5', 'a6'), ('a2', 'a3')], ['a5', 'a2']),
        ([('a1', 'a2'), ('a5', 'a6'), ('a2', 'a3')], ['c1']),
        ([('a1', 'a2'), ('a5', 'a6'), ('a2', 'a3'), ('a4', 'a6'), ('a6', 'a2'),
('a2', 'a1')], ['c1', 'a2']),
        ([('a1', 'a2'), ('a5', 'a6'), ('a2', 'a3'), ('a4', 'a6'), ('a6', 'a2'),
('a3', 'a1')], ['a2']),
    ]

    for service,invalid_service in all_cases:
        tree = Tree()
        tree.create_tree(service)
        tree.update_invalid_services(invalid_service)
        res = tree.show_live_services()
        print(res)
```

55、判断字符串子序列

```
# 序号14
# 标题：判断字符串子序列。
```

```
# 给定字符串target和source，判断target是否为source的子序列。你可以认为target和source中仅包含英文小写字母，字符串source可能会很长（长度～=500,000）而target是个短字符串（长度<=100）。字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"abc"是aebycd的一个子序列，而"ayb"不是)。
# 请找出最后一个子序列的起始位置。
# 输入描述：
#
# 第一行为target，短字符长度（长度<=100）
# 第二行为source，长度字符串（长度～=500,000)
#
# 输出描述：
# 最后一个子序列的起始位置即最后一个子序列首字母的下标
#
# 备注：
#
# 若在source中找不到target，则输出-1
#
# 示例1：
# 输入
# abc
# abcaybec
#
# 输出
# 3
def no_14(target,source):
    target_rev_list = list(target)[::-1]
    source_rev_list = list(source)
    length_target = len(target)
    length_source = len(source)
    #匹配逆target序列的第cur个索引位置
    cur = 0
    flag = False
    tmp = 0
    for i in range(length_source-1,-1,-1):
        #匹配成功则匹配下一个
        if target_rev_list[cur] == source_rev_list[i]:
            cur += 1
        if cur == length_target:
            flag = True
            tmp = i
            break
    if flag:
        return tmp
    return -1
if __name__ == '__main__':
    input1 = [
        "abc","a","abc","bc"
    ]
    input2 = [
        "abcaybec","a","abbee","abbcec"
    ]
    for target,source in zip(input1,input2):
        res = no_14(target,source)
        print(res)
```

56、组成最大值

```
# 序号15
# 标题：最大值
# 给定一组数（非负），重排顺序后输出一个最大的整数。
# 示例1：
# 输入：[10,9]
# 输出：   910
# 说明：输出结果可能非常大，所以你需要返回一个字符串而不是整数。
#
# 输入描述：
# 数字组合
#
# 输出描述：
# 最大的整数
#
# 示例1：
# 输入
# 10  9
#
# 输出
# 910
from functools import cmp_to_key
def no_15(array_list):
    def compare(x,y):
        return int(y+x) - int(x+y)
    res = sorted(map(str,array_list),key=cmp_to_key(compare))
    return "0" if not array_list else "".join(res)
if __name__ == '__main__':
    all_cases = [
        [10, 9],#910
        [9, 97, 80, 7, 200, 101, 2000],#9978072002000101
        [8, 1000, 10, 98],#988101000
        [1000, 10, 1010],#1010101000
        [300, 30, 200, 20, 2000],#30300202002000
        [8, 1000, 10, 98, 1010, 10010],#9881010101001010000
        [1000, 10, 10011, 1010, 10010],#101010100111001010000
        [1000, 10, 1, 110, 1100, 1011, 1010, 100,
10010],#11101100101110101010010010001000
        [1000, 10, 1, 110, 1100, 1011, 1010, 100, 10010,
120],#120111011001011101010100101001000
        [30000000000, 647836278492112, 473892473829621834940328, 9, 88,
10]#9886478362784921124738924738296218349403283000000000010
    ]
    for case in all_cases:
        print(no_15(case))
```

57、称砝码

```
while True:
    try:
        a = int(input())
        b = input()
        c = input()
        b = list(map(int,b.split()))
        c = list(map(int,c.split()))
        amount = []
        weights = {0,}
```

```
        for i in range(a):
            for j in range(c[i]):
                amount.append(b[i])

        for i in amount:
            for j in list(weights):
                weights.add(i+j)

        print(len(weights))
    except:
        break
```

58、计算字符串的编辑距离

```
while True:
    try:
        str1 = input()
        str2 = input()
        m = len(str1)
        n = len(str2)

        dp = [[1 for i in range(n+1)] for j in range(m+1)]#重点注意二维数据的创建方法，重点注意其横竖坐标，注意注意
        for i in range(n+1):
            dp[0][i] = i
        for j in range(m+1):
            dp[j][0] = j

        for i in range(1,m+1):
            for j in range(1,n+1):
                if str1[i-1] == str2[j-1]:#如果当前两个字母相同，则跳过，步数不增加
                    dp[i][j]=dp[i-1][j-1]
                else:  #如果两个字母不同，则有三种方式可以达成，删除、插入、替换，选择最小的前状态，步数加1
                    dp[i][j] = min(dp[i-1][j-1],dp[i][j-1],dp[i-1][j])+1
        print(dp[m][n])
    except:
        break
```

59、24点游戏算法

```
while True:
    try:
        a = input()
        a_list = list(map(int,a.split()))
        def dfs(num_list,target):
            if len(num_list) == 1:
                return num_list[0] == target
            else:
                for i in range(len(num_list)):
                    list1 = num_list[:i] + num_list[i+1:]
                    n = num_list[i]
                    if dfs(list1,target + n) or dfs(list1,target - n) or dfs(list1, target * n) or dfs(list1, target/n):
                        return True
```

```
                else:
                    return False
        if dfs(a_list,24):
            print("true")
        else:
            print("false")
    except:
        break
```

60、扑克24点运算

```
import itertools
def fun(a,b,c,d):
    card_dict = {
    '2': 2,
    '3': 3,
    '4': 4,
    '5': 5,
    '6': 6,
    '7': 7,
    '8': 8,
    '9': 9,
    '10': 10,
    'J': 11,
    'Q': 12,
    'K': 13,
    'A': 1}
    rules = ['+', '-', '*', '/']
    # 把牌面转为整数
    x, y, z, m = card_dict[a], card_dict[b], card_dict[c], card_dict[d]
    # 一组数字计算中有三个加减乘除符号，计算所有可能性
    for i in rules:
        for j in rules:
            for k in rules:
                # 因为是从左往右计算，把前面的括起来
                result = eval('((' + 'x' + i + 'y' + ')' + j + 'z' + ')' + k +
'm')
                if result == 24:
                    return a + i + b + j + c + k + d
    else:
        return False
while True:
    try:
        card_list = input().split()
        if 'joker' in card_list or 'JOKER' in card_list:
            print('ERROR')
        else:
            # 所有的牌面顺序
            all_list = itertools.permutations(card_list)
            for each in all_list:
                a, b, c, d = each
                result = fun(a, b, c, d)
                if result:
                    print(result)
            else:
                print('NONE')
```

```
    except:
        break
```

61、记负均正II

```
z = []
f = []
while True:
    try:
        a = int(input())
        if a>=0:
            z.append(a)
        else:
            f.append(a)
    except:
        print(len(f))
        if z:
            print(round(sum(z)/len(z),1))
        else:
            print("0.0")
        break
```

62、人民币转换

```
#1111 1111 1111
#壹仟壹佰壹拾壹亿壹仟壹佰壹拾壹万壹仟壹佰壹拾壹元
#  仟 佰 拾 亿 仟 佰 拾 万 仟 佰 拾 元
b = ["仟","佰","拾","亿","仟","佰","拾","万","仟","佰","拾","元"]
while True:
    try:
        a = input()
        print("人民币",end="")
        #151121.15
        #壹伍 壹壹贰壹
        #壹拾伍 壹仟壹佰贰拾壹
        a = a.replace("0","零")
        a = a.replace("1","壹")
        a = a.replace("2","贰")
        a = a.replace("3","叁")
        a = a.replace("4","肆")
        a = a.replace("5","伍")
        a = a.replace("6","陆")
        a = a.replace("7","柒")
        a = a.replace("8","捌")
        a = a.replace("9","玖")
        a = a.split(".")
        res = ""
        for i,j in zip(list(a[0]),b[-len(a[0]):]):
            res = res+i+j
        if res.startswith("壹拾"):
            res = res[1:]
        if res.startswith("零元"):
            res = ""
        if a[1] == "00":
```

```
            print(res+"整")
        else:
            res = res+a[1][0]+"角"+a[1][1]+"分"
            res = res.replace("零角","")
            res = res.replace("零分","")
            print(res)
    except:
        break
```