def lcs(a,b):

**# 动态规划解最长公共子序列(LCS)**

lena=len(a)

lenb=len(b)

c=[[0 for i in range(lenb+1)] for j in range(lena+1)] #初始化计数矩阵

flag=[[0 for i in range(lenb+1)] for j in range(lena+1)] #初始化标记矩阵

for i in range(lena):

for j in range(lenb):

if a[i]==b[j]:

c[i+1][j+1]=c[i][j]+1

flag[i+1][j+1]='ok'

elif c[i+1][j]>c[i][j+1]:

c[i+1][j+1]=c[i+1][j]

flag[i+1][j+1]='left'

else:

c[i+1][j+1]=c[i][j+1]

flag[i+1][j+1]='up'

return c,flag #c是矩阵，里面存每一对对应的数值； flag标记了回溯方向

def printLcs(flag,a,i,j):

if i==0 or j==0:

return

if flag[i][j]=='ok':

printLcs(flag,a,i-1,j-1)

print(a[i-1],end='') #单行持续输出

elif flag[i][j]=='left':

printLcs(flag,a,i,j-1)

else:

printLcs(flag,a,i-1,j)

#inputa, inputb="AAACCGTGAGTTATTCGTTCTAGAA", "CACCCCTAAGGTACCTTTGGTTC" #ACCTAGTACTTTG

#inputa, inputb="houseboat", "computer"#oue

#inputa, inputb="2193588", "21943588" #2193588

a='ABCBDABC'

b='BDCABAC'

c,flag=lcs(a,b)

for i in c:

print(i)

print('')

for j in flag:

print(j)

print('')

printLcs(flag,a,len(a),len(b))

print('')

**# 数独**

class Solution:

def isValue(self, board, x, y):

# 检查已经填入的坐标是否和列中有的元素相等

for i in range(9):

if i != x and board[i][y] == board[x][y]:

return False

# 检查已经填入的坐标是否和行中有的元素相等

for j in range(9):

if j != y and board[x][j] == board[x][y]:

return False

# 检查每个正方形是否符合（粗线框内只有1~9）

m, n = 3\*(x // 3), 3\*(y // 3) # 这里求出的是3x3网格的左上角的坐标

for i in range(3):

for j in range(3):

if(i+m != x or j+n != y) and board[i+m][j+n] == board[x][y]:

return False

return True

def dfs(self, board):

for i in range(9):

for j in range(9):

if board[i][j] == 0:

for k in '123456789': # 从里面选择一个

board[i][j] = int(k)

if self.isValue(board, i, j) and self.dfs(board):

return True

# 回溯

board[i][j] = 0

# 都不行，说明上次的数字不合理

return False

# 全部便利完，返回True

return True

board = [[0,9,2,4,8,1,7,6,3],

[4,1,3,7,6,2,9,8,5],

[8,6,7,3,5,9,4,1,2],

[6,2,4,1,9,5,3,7,8],

[7,5,9,8,4,3,1,2,6],

[1,3,8,6,2,7,5,9,4],

[2,7,1,5,3,8,6,4,9],

[3,8,6,9,1,4,2,5,7],

[0,4,5,2,7,6,8,3,1]]

s = Solution()

s.dfs(board)

print(board)

**# 素数伴侣**

def getPrimes(n):

"""获取2~n之间的素数，用于对素数进行判断"""

primes = [True for \_ in range(n+1)]

for i in range(2, n+1):

if primes[i]:

yield i

for j in range(i+i, n+1, i):

primes[j] = False

def splitNums(nums):

"""用于将数字分为奇数和偶数部分"""

odd = []

even = []

oddLen = 0

evenLen = 0

for num in nums:

if num % 2 == 0:

even.append(num)

evenLen += 1

else:

odd.append(num)

oddLen += 1

return odd, oddLen, even, evenLen

def removeValue(data, n, i, k):

"""从data的第i行开始去掉数字i"""

for j in range(i, n):

if k in data[j]:

data[j].remove(k)

return data

def countValue(data, n, i, d):

"""统计data从i行开始，d中的元素总共出现的次数"""

d = {k: 0 for k in d}

for j in range(i, n):

for value in data[j]:

if value in d:

d[value] += 1

return d

def func(nums):

odd, oddLen, even, evenLen = splitNums(nums)

maxGroup = min(oddLen, evenLen)

if maxGroup == 0: # 如果没有奇数或者偶数，那么直接必定不会出现素数伴侣

return 0

primes = list(getPrimes(max(odd)+max(even)))

# 每一行代表1个偶数，每一列代表一个奇数

# 此处得到每个一行的偶数能和哪几个奇数组成素数伴侣

data = [[j for j in range(oddLen) if even[i] + odd[j] in primes] for i in range(evenLen)]

data = [row for row in data if row] # 去掉不能和任何数字组成伴侣的偶数行

if not data: # 为空表示没有任何偶数能和奇数组成伴侣

return 0

stack = [] # 存储结果用容器

n = len(data)

for i in range(n-1):

# 分别对每一行进行迭代，判断改行是否符合一下的某一种情况（最后1行不需要判断，直接取最后1行的第一个数据即可）

# 1 如果这一行为空，那么直接忽略这一行

# 2 这一行只能和1个奇数 K 组成伴侣，那么就取这个偶数和K组成伴侣

# 3 若这一行可以和多个奇数组成伴侣，那么去查找K，要求：后面行重复使用到K的次数最少

# 4 去掉后面行用到的K

# 5 若出现情况如：[[1,2], [1], [2, 3], [2, 4]]，避免第一行取到1，

# 在每一次取数之前，将数据按照候选奇数的多少依次排列

data = data[:i] + sorted(data[i:], key=lambda x: len(x))

if len(data[i]) == 0:

continue

if len(data[i]) == 1:

stack.append(data[i][0])

data = removeValue(data, n, i+1, data[i][0])

continue

d = countValue(data, n, i+1, data[i])

minValue = data[i][0]

c = d[minValue]

if c == 0:

stack.append(data[i][0])

data = removeValue(data, n, i+1, data[i][0])

continue

for k, v in d.items():

if v < k:

minValue = k

c = v

stack.append(minValue)

if c == 0:

continue

data = removeValue(data, n, i+1, minValue)

if data[-1]: # 最后1行不需要判断，直接取最后1行的第一个数据即可

stack.append(data[-1][0])

return len(stack)

n = 4

nums = [2,5,6,13]

print(func(nums))

**# 24点**

d = {'3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9,

'10':10, 'J':11, 'Q':12, 'K':13, 'A':1, '2':2}

def f(nums, target):

if len(nums) == 1:

if d[nums[0]] == target:

res.append(nums[0])

return True

else:

return False

for i in range(len(nums)):

a = nums[i]

b = nums[:i] + nums[i+1:]

if f(b, target + d[a]):

res.append('-' + a)

return True

elif f(b, target - d[a]):

res.append('+' + a)

return True

elif f(b, target \* d[a]):

res.append('/' + a)

return True

elif target % d[a] == 0 and f(b, target // d[a]):

res.append('\*' + a)

return True

return False

nums = ['K','Q','6','K']

print(f(nums, 24))

**# N皇后问题**

def solution(n):

scale = n

def safe(x):

for i in range(len(x)-1):

if (x[-1] == x[i]) or (abs(x[-1] - x[i]) == (len(x)-1) - i):

return False

return True

def f(k):

if k == 1:

return [[x] for x in range(1, scale + 1)]

else:

last = f(k-1)

return list(filter(safe, [x+[y] for x in last for y in range(1, scale+1)]))

return f(n)

aaa = len(solution(8))

print(aaa)

**# 01背包问题**

w1 = [2,3,4,5]

v1 = [3,4,5,6]

c1 = 8

def pack1a(w, v, C): #每个东西只能选择一次

dp = [[0 for \_ in range(C+1)] for \_ in range(len(w)+1)]

for i in range(1, len(w)+1):

for j in range(1, C+1):

if j < w[i-1]: #如果剩余容量不够新来的物体 直接等于之前的

dp[i][j] = dp[i-1][j]

else:

dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i-1]]+ v[i-1])

return dp[len(w)][C]

def pack1b(w, v, c):

#它是先得到第一行的值，存到dp中，然后再直接用dp相当于就是上一行的值，所以下面必须用逆序

#否则dp[j-w[i-1]]可能会用到你本行的值，从大到小就不会

dp = [0 for \_ in range(c+1)]

for i in range(1, len(w)+1):

for j in reversed(range(1, c+1)):#这里必须用逆序

if w[i-1] <= j:

dp[j] = max(dp[j], dp[j-w[i-1]]+v[i-1])

return dp[c]

out1a = pack1a(w1,v1,c1)

out1b = pack1b(w1,v1,c1)

#print(out2)

**# 完全背包问题**

def pack2a(w, v, C): #每个东西能选择多次 完全背包问题

dp = [[0 for \_ in range(C+1)] for \_ in range(len(w)+1)]

for i in range(1, len(w)+1):

for j in range(1, C+1):

if j < w[i-1]:

dp[i][j] = dp[i-1][j]

else:

dp[i][j] = max(dp[i-1][j], dp[i][j-w[i-1]] + v[i-1])

return dp

def pack2b(w, v, C):

dp = [0 for \_ in range(C+1)]

for i in range(1, len(w)+1):

for j in (range(1, C+1)):

if w[i-1] <= j:

dp[j] = max(dp[j], dp[j-w[i-1]]+v[i-1])

return dp

out2a = pack2a([2,3,4,5], [3,4,5,6], 8)

out2b = pack2b([2,3,4,5], [3,4,5,6], 8)

**# 多重背包问题**

s = [1,1,1,3]

def pack3a(w, v, s, c):

dp = [0 for \_ in range(c+1)]

for i in range(1, len(w)+1):

for j in reversed(range(1, c+1)):

for k in range(s[i-1] + 1):

if k\*w[i-1] <= j:

dp[j] = max(dp[j], dp[j-k\*w[i-1]]+k\*v[i-1])

return dp[c]

def pack3b(w, v, s, c):

for i in range(len(s)):

k = 1

s\_value = s[i]

while k<=s\_value:

w2.append(k\*w[i])

v2.append(k\*v[i])

s\_value -= k

k \*= 2

if s\_value>0:

w2.append(s\_value\*w[i])

v2.append(s\_value\*v[i])

#前面是划分，后面是0，1背包

dp = [0 for \_ in range(c+1)]

for i in range(1, len(w2)+1):

for j in reversed(range(1, c+1)):

if w2[i-1] <= j:

dp[j] = max(dp[j], dp[j-w2[i-1]]+v2[i-1])

return dp[c]

out3a = pack3a(w1, v1, s, 8)

out3b = pack3a(w1, v1, s, 8)

#print(out3a)

print(out3b)

**# 混合背包问题**

# 第一类物品只能用1次（01背包）；

# 第二类物品可以用无限次（完全背包）；

# 第三类物品最多只能用 sisi 次（多重背包）；

# si=−1 表示第 ii 种物品只能用1次；

# si=0 表示第 ii 种物品可以用无限次；

# si>0 表示第 ii 种物品可以使用 si 次；

def pack4(w, v, c, s):

w2 = []

v2 = []

s2 = []

for i in range(len(s)):

if s[i] == 0 or s[i] == -1:

w2.append(w[i])

v2.append(v[i])

s2.append(s[i])

else:

s\_value = s[i]

k = 1

while k <= s\_value:

w2.append(k\*w[i])

v2.append(k\*v[i])

s2.append(s[i])

s\_value -= k

k \*= 2

if s\_value> 0:

w2.append(s\_value\*w[i])

v2.append(s\_value\*v[i])

s2.append(s[i])

#上面把si>0的背包拆分了（变成0，1背包）下面分成0，1背包和无限背包两种

dp = [0 for \_ in range(c+1)]

for i in range(1, len(w2)+1):

if s2[i-1] == 0:

for j in (range(1, c+1)):

if j-w2[i-1]>=0:

dp[j] = max(dp[j], dp[j-w2[i-1]]+v2[i-1])

else:

# print('i',i)

for j in reversed(range(1, c+1)):

if j-w2[i-1]>=0:

# print('k', k)

dp[j] = max(dp[j], dp[j-w2[i-1]]+v2[i-1])

# print('dp['+str(j)+']', dp[j])

# print(dp)

print(dp[c])

**# 最大升序子序列**

def longest\_upper(nums):

dp = [1]\*len(nums)

for i in range(n):

for j in range(i):

if nums[j] < nums[i]:

dp[i] = max(dp[i],dp[j]+1)

return dp

list\_ = [2,5,1,5,4,5]

#print(longest\_upper(list\_))

def lis(arr):

n = len(arr)

m = [0]\*n

for x in range(n-2,-1,-1):

for y in range(n-1,x,-1):

if arr[x] < arr[y] and m[x] <= m[y]:

m[x] += 1

max\_value = max(m)

result = []

for i in range(n):

if m[i] == max\_value:

result.append(arr[i])

max\_value -= 1

return result

arr = [10, 22, 9, 33, 21, 50, 41, 60, 80]

#print(lis(arr))

**# 编辑距离**

def editDistance(str1, str2):

'''

计算字符串str1和str2的编辑距离

'''

edit = [[i+j for j in range(len(str2)+1)] for i in range(len(str1)+1)]

for i in range(1,len(str1)+1):

for j in range(1,len(str2)+1):

if str1[i-1] == str2[j-1]: # 这里的[i-1], [j-1]对应原字符串的i和j

d = 0

else:

d = 1

edit[i][j] = min(edit[i-1][j]+1,edit[i][j-1]+1,edit[i-1][j-1]+d)

return edit[len(str1)][len(str2)]

while True:

try:

print(editDistance(input(), input()))

except:

break

**# 面积**

'''

n,e=list(map(int,input().split()))

w=[]

arr\_s=[]

for i in range(n):

w0=list(map(int,input().split()))

if len(w)==0:

w.append(w0)

else:

s=abs((w0[0]-w[-1][0])\*w[-1][1])

arr\_s.append(s)

w.append([w0[0],w0[1]+w[-1][1]])

if w[-1][0]<e:

s=abs((e-w[-1][0])\*w[-1][1])

arr\_s.append(s)

#print(w)

#print(arr\_s)

#print(sum(arr\_s))

'''

**# 身高体重**

'''

n=int(input())

heights=[ [i+1,h] for i,h in enumerate(map(int,input().split()))]

weights=list(map(int,input().split()))

heights\_sort=sorted(heights,key=lambda x:x[1])

for i in range(1,n):

h\_c=heights\_sort[i][1]

h\_index=heights\_sort[i][0]

h\_w=heights\_sort[i][0]

for j in range(i-1,-1,-1):

if heights\_sort[j][1]==h\_c:

h\_w1=heights\_sort[j][0]

if weights[h\_w1-1]>weights[h\_w-1]:

heights\_sort[j+1],heights\_sort[j]=heights\_sort[j],heights\_sort[j+1]

elif weights[h\_w1-1]==weights[h\_w-1]:

if h\_index<heights\_sort[j][0]:

heights\_sort[j + 1], heights\_sort[j] = heights\_sort[j], heights\_sort[j + 1]

else:

break

else:

break

result=[str(i) for i,h in heights\_sort]

print(" ".join(result))

'''

**# 广播站（集合合并）**

#row\_data = input().split(',')

row\_data = ['110', '110', '001']

#print(row\_data)

length\_row = len(row\_data)

list\_ = []

result = []

mark = []

for i in range(length\_row):

list\_row = []

for j in row\_data[i]:

list\_row.append(int(j))

list\_.append(list\_row)

def findgroup(index,res):

for i in range(length\_row):

if(list\_[index][i] == 1):

if(i not in res):

res.append(i)

findgroup(i,res)

for i in range(length\_row):

res = []

if i not in mark:

res.clear()

findgroup(i,res)

result.append(res)

mark.extend(res)

print(len(result))

**# 反转字符串**

class Solution(object):

def reverseParentheses(self, s):

"""

:type s: str

:rtype: str

"""

if not s or ")" not in s:

return s

stack = []

for i, char in enumerate(s):

if char == "(":

stack.append(i)

elif char == ")":

left = stack.pop()

right = i

return self.reverseParentheses(s[:left] + s[left + 1:right][::-1] + s[right + 1:])

s = Solution()

s.reverseParentheses("a(bcdefghijkl(mno)p)q")

**# 请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径**

ap = [['a','b','c','e'],

['s','f','c','s'],

['a','d','e','e']]

char\_ = 'abfccs'

class Solution:

def exist(self, board, word) -> bool:

def dfs(i, j, k):#i行索引；j列索引；k当前目标字符在word中的索引

if not 0 <= i < len(board) or not 0 <= j < len(board[0]): return False#行/列索引越界

elif board[i][j] != word[k]: return False #board矩阵中 当前遍历元素和 word目标元素不相等

elif k == len(word) - 1: return True#word的第【len-1】个索引对于的元素也找到了，word遍历完成，找到目标字符串

#行列索引均未越界，且board矩阵中当前遍历元素和word目标元素相等时：

board[i][j] = ''#board矩阵当前元素，修改成空字符，代表此元素已访问过，防止之后搜索时重复访问

res = dfs(i + 1, j, k + 1) or dfs(i - 1, j, k + 1) or dfs(i, j + 1, k + 1) or dfs(i, j - 1, k + 1)#对board当前符合要求的元素，进行下、上、右、左 元素的搜寻，word索引加1

board[i][j] = word[k]#还原当前遍历元素 至初始值

return res

for i in range(len(board)):#i为行索引

for j in range(len(board[0])):#j为列索引

if dfs(i, j, 0): return True#该行代码只对【dfs函数返回True】这个情况进行操作，若【dfs函数返回False】，无操作，继续对board矩阵下一个元素进行遍历。

return False#未搜索到目标字符串

sou = Solution()

sou.exist(ap,char\_)

**# 砝码**

def fama(n,weight,nums):

res = set()

for i in range(nums[0]+1):

res.add(i\*weight[0])

for i in range(1,n):

tmp = list(res)

for j in range(1,nums[i]+1):

for wt in tmp:#变成list在这里才能遍历;tmp是在不断增加数据的

res.add(wt+j\*weight[i])

return len(res)

n = 2

weight = [1,2]

nums = [2,1]

print(fama(n,weight, nums))

**#中心寻找对称字符长度**

def longp(s):

res = ''

for i in range(len(s)):

#先判定奇数的，从i开始左右对比

tmp = helper(s,i,i)

if len(tmp) > len(res):res = tmp

#再判定偶数的，从i和i+1开始对比

tmp = helper(s,i,i+1)

if len(tmp) > len(res):res = tmp

print(len(res))

def helper(s,l,r):

while l >= 0 and r < len(s) and s[l] == s[r]:

l -= 1

r += 1

return s[l+1:r]

longp('cabac')

**#火车进站**

def func(n, trains):

res = []

def rec\_trains(cur\_idx, in\_trains, out\_trains):

# 如果原始火车列表的最后一个元素已经进站，此时只能出站，将入站列表中的火车倒序加入出站火车中

if trains[-1] in in\_trains:

res.append(' '.join(out\_trains + in\_trains[::-1]))

return

# 如果进站列表为空，此时只能进站，进站列表加上当前火车，出站列表不变

elif in\_trains == []:

rec\_trains(cur\_idx + 1, in\_trains + [trains[cur\_idx]], out\_trains)

# 否则，就既有可能进站也有可能出站

else:

# 出站，当前火车索引不变，进站火车列表减去最后一个元素，出站列表加上进站列表刚刚出站的火车

rec\_trains(cur\_idx, in\_trains[:-1], out\_trains + [in\_trains[-1]])

# 进站，当前火车索引加1，进站列表加上当前火车，出站列表不变

rec\_trains(cur\_idx + 1, in\_trains + [trains[cur\_idx]], out\_trains)

rec\_trains(0, [], [])

res.sort()

print('\n'.join(res))

if \_\_name\_\_ == '\_\_main\_\_':

func(3,['1','2','3'])

**# 数字3和5分组, 查看某一组数字能不能挑选相加凑出来target**

def search(ll, target):

if target == 0: #目标降为0

return True

if not ll: #空集

return False

return search(ll[1:], target) or search(ll[1:], target - ll[0])

print(search([1,2,3,4,5],16))

1. time\_dict = {}
2. ms\_dict = {}
3. i = 0
4. result = 0
5. while True:
6. try:
7. time\_tmp = input()
8. s = time\_tmp[:-5]
9. ms = int(time\_tmp[-3:])
10. if time\_dict.\_\_contains\_\_(s):
11. print(ms)
12. if time\_dict[s] == ms:
13. result += 1
14. else:
15. time\_dict[s] = ms
16. result += 1
17. i+=1
18. except:
19. break
20. print(result)