**dictionary.get(key, default\_value)**

**dictionary：表示要检索值的字典对象。**

**key：表示要检索值的键。**

**default\_value（可选）：如果键不存在于字典中，则返回该默认值。**

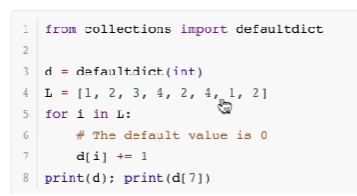
**from collections import defaultdict**

**import heapq**

**q = int(input())**

**l,r = [],[]**

**left,right = defaultdict(int),defaultdict(int)**



**保留小数**

x1 = format(t, ".5f"); print(f"x1=x2={x1}") #先保留，后带格式输出

**埃筛：……………………算质数**

def sieve\_of\_eratosthenes(n):  
 prime = [True] \* (n + 1)  
 prime[0] = prime[1] = False  
 primes = []  
 for p in range(2, n + 1):  
 if prime[p]:  
 primes.append(p)  
 for i in range(p \* p, n + 1, p):  
 prime[i] = False  
 return set(primes)  
n=int(input())  
l=list(map(int,input().split()))  
sieve\_result=sieve\_of\_eratosthenes(1000000) #1e6是浮点数  
# 或者引入from math import sqrt 直接 sqrt(num)  
def is\_tprime(n):  
 sqrt\_n=n\*\*0.5  
 return sqrt\_n \* sqrt\_n==n and sqrt\_n in sieve\_result  
  
for i in l:  
 print('YES' if is\_tprime(i) else 'NO')

# 字符串

**字符串'''http://cs101.openjudge.cn/routine/27314/ 一键换词'''**

text = input()

original\_word, target\_word = input().split()

sentences = text.split(". ")

result = []

for sentence in sentences:

words = sentence.split()

for i in range(len(words)):

word = words[i].strip(",.").lower()

if word == original\_word.lower(): # 等于要替换的字符

if words[i][-1] in {",", "."}: # 后边是标点

tmp = words[i][-1]

words[i] = target\_word + tmp

else:

words[i] = target\_word

modified\_sentence = " ".join(words)

result.append(modified\_sentence.capitalize()) #每一句都更新过了

output = ". ".join(result)

print(output)

**字符串'''延长后在不在里面'''**

t = int(input())  
for \_ in range(t):  
 n, m = map(int, input().split())  
 x = input()  
 words = input()  
 flag=0  
 for i in range(6):  
 if words in x: # 直接用in判断是不是在延长后的串里  
 print(i)  
 flag=1  
 break  
 x += x # 每次循环都延长一遍  
 if flag==0:  
 print(-1)

**Judge if s in t：**

while True:  
 try:  
 s, t = input().split()  
 i = j = 0  
 while i < len(s) and j < len(t):  
 if s[i] == t[j]: # judge if s in t  
 i += 1  
 j += 1  
 if i == len(s):  
 print('Yes')  
 else:  
 print('No')  
 except EOFError:  
 break

**字符串最小周期**

while True: # 求字符串的最小周期

s = input()

if s[0] == '.':

break

length = len(s)

for i in range(1, length + 1): # 遍历可能的周期

flag = 0

if length % i != 0: # 字串长度不是总长度的因数

continue

for j in range(length):

if s[j] != s[j % i]: # s[j]表示当前字符，s[j % i]表示在周期长度为i的情况下对应位置上的字符

flag = 1

break

if not flag: # 没有被打断

print(length // i) # i是周期

Break

**字符串和ascii码转换：字母加密**

l = input()

ans = []

num = "4962873"

i = 0

for idx,a in enumerate(l):

new\_a = ord(a)

if 32 <= new\_a <=122:

new\_a += int(num[i])

else:

new\_a %= int(num[i])

ans.append(chr(new\_a))

i += 1

if i == len(num):

i = 0

print("".join(ans))

print("".join(l))

**字符串：最大新整数，允许删k个数**

n = int(input())  
for \_ in range(n):  
 tmp, tmpk = input().split()  
 arr = list(tmp)  
 k = int(tmpk)  
 ans = []  
 i = 0  
 while k > 0 and i < len(arr) - 1:  
 if arr[i] > arr[i+1]:  
 arr.pop(i)  
 k -= 1  
 i = 0 # 每次重新开始，找降序的，删掉山峰  
 else:  
 i += 1  
 while k > 0: # 经过上个循环已经没有前一个大于后一个的可能了  
 arr.pop(-1)  
 k -= 1  
 print("".join(arr))

字符串的自定义排序：最大最小整数

final\_max = ''  
final\_min = ''  
  
n = int(input())  
l = list(map(str, input().split()))  
  
for i in range(len(l) - 1):  
 for j in range(i + 1, len(l)): # 后边所有数字都遍历到  
 if l[i] + l[j] < l[j] + l[i]:  
 l[i], l[j] = l[j], l[i]  
  
for i in l:  
 final\_max += i  
  
for j in range(len(l) - 1, -1, -1):  
 final\_min += l[j]  
  
print(final\_max, final\_min)

**十进制转为x进制**

print('{0:x}'.format(decimal\_num).upper()) #十进制转化为16进制

print('{0:b}'.format(decimal\_num)) #十进制转化为2进制

print('{0:o}'.format(decimal\_num)) #十进制转化为8进制

print(bin(i),oct(i),hex(i).lower())

# 二进制转十进制

binary\_str = "1010"

decimal\_num = int(binary\_str, 2) # 第一个参数是字符串类型的某进制数，第二个参数是他的进制，最终转化为整数

print(decimal\_num) # 输出 10

**是不是整数：**

if isinstance(num, int):

**生成排列**

from itertools import permutations

my\_list = [1, 2, 3] # 替换成您自己的列表

all\_permutations = list(permutations(my\_list))

import math

def permutation(n, m):

return math.factorial(n) // math.factorial(n-m)

import math

def combination(n, m):

return math.factorial(n) // (math.factorial(m) \* math.factorial(n-m))

**Round**

import math  
print(round(4.4)) # 4,四舍五入  
print(round(4.5)) # 4,舍入到最近的偶数  
print(round(4.7)) # 5,四舍五入

**比较接近**

If math.isclose(n, 0.00, rel\_tol=1e-5) :

**import math**

print(math.ceil(4.4))  
print(math.ceil(4.5))  
print(math.ceil(4.7)) # 全是5

print(math.pow(2,3)) # 8.0

print(math.pow(2,2.5)) # 5.656854249492381

print(9999999>math.inf) # False

print(math.sqrt(4)) # 2.0

print(math.log(100,10)) # 2.0 math.log(x,base) 以base为底，x的对数

# 递归

**二叉树**

def common(a, b):  
 if a == b:  
 return a  
 elif a > b:  
 return common(a // 2, b)  
 else:  
 return common(a, b // 2)  
a, b = map(int, input().split())  
print(common(a, b))

**简单的整数划分：完全背包问题的取法**

while True:

try:

x = int(input())

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

for i in range(1, x + 1):

dp[i][1] = 1 # 划分数是1以下的数，只有一种，就是1111……

dp[1][i] = 1 # 被划分数是1，只有一种，就是被划分数自己.也就是只要两个位置有1，return 1

for i in range(2, x + 1): # i和j有一个是1，dp就是1了，从0开始

for j in range(2, x + 1):

if i > j: # 完全copy递推公式

dp[i][j] = dp[i - j][j] + dp[i][j - 1] # 此时有两种，划分结果有j和无j

elif i < j:

dp[i][j] = dp[i][i]

else:

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

print(dp[x][x])

**复杂的整数划分问题：**

**def div(n,k): # n 分为 k 个正整数，可以重复**  
 dp = [[0] \* (k + 1) for \_ in range(n + 1)]  
 for i in range(n + 1):  
 dp[i][1] = 1 # 划分为1个数只有一种方法，0也  
 for i in range(1, n + 1):  
 for j in range(1, k + 1):  
 if i >= j:  
# 先分出去一个1，剩下的i-1被分为j-1个数  
# dp[i-1][j-1]为包含1的划分的数量  
# dp[i - j][j]为不包含1的划分数量，首先给每个盘子放1苹果，一共减去j个苹果，得到i-j  
# 之后i-j再分到j个盘子  
 dp[i][j] = dp[i - j][j] + dp[i - 1][j - 1]  
 return dp[n][k]  
  
**def div\_diff(n): # 不同的划分，不限个数**  
 dp = [[0] \* (n + 1) for \_ in range(n + 1)] # 把i划分，最大划分数为j  
 for i in range(1, n + 1):  
 for j in range(1, n + 1):  
 if i < j:  
 dp[i][j] = dp[i][i]

#比i大的数没用，可以用自己划分自己  
 elif i == j:   
 dp[i][j] = dp[i][j - 1] + 1

# 最大数不为i（其他的都比j小），最大数为i（只有一种可能）  
 else: # i > j  
 dp[i][j] = dp[i][j - 1] + dp[i - j][j - 1]  
# 最大数不为j时，那么最大为j - 1，dp[i][j - 1]  
# 最大数为j时，先从i中取出j，再对i - j进行划分，不允许划分重复，最大为j-1，dp[i-j][j - 1]  
 return dp[n][n]  
  
**def div\_odd(n): #奇数划分，可重复** dp = [[0] \* (n + 1) for \_ in range(n + 1)] # 把i划分，最大划分数为j  
 dp[0][0] = 1  
 for i in range(1, n + 1):  
 for j in range(1, n + 1):  
 if j % 2 == 0:

# j偶数，最大划分数不能等于j  
 dp[i][j] = dp[i][j - 1]  
 else: # j为奇数  
 if i < j: # 多了没用  
 dp[i][j] = dp[i][i]  
 elif i == j:  
 dp[i][j] = dp[i][j - 1] + 1  
 else:

# 不包括j，最大是j-1，包括j：可以重复使用  
 dp[i][j] = dp[i][j - 1] + dp[i - j][j]  
  
 return dp[n][n]  
  
while True:  
 try:  
 n, k = map(int, input().split())  
 print(div(n,k))  
 print(div\_diff(n))  
 print(div\_odd(n))  
  
 except EOFError:  
 break

**因数分解：递归**

def f(n,m):  
 count = 0  
 if n == 1: # 被分解数是1，有一种分解，就是自己  
 return 1  
 for i in range(m,n+1): # m从2开始，能整除就将被2分解了的n//i再放进去递归一下。最多不过n  
 if n % i == 0: # i是因数  
 count += f(n//i,i) # count累加递归数，更新进入的i  
 return count

n = int(input())  
for k in range(1, n + 1):  
 x = int(input())  
 num = f(x, 2) # 最小因数是2  
 print(num)

**分解为2的次方之和**

n = int(input())  
dp = [0] \* (n+1)  
dp[1] = 1  
for a in range(2,n+1):  
 if a % 2 == 0:  
 dp[a] = (dp[a-1] + dp[a//2]) % (10\*\*9)  
 else:  
 dp[a] = dp[a-1] % (10\*\*9)  
print(str(dp[n]).zfill(9))

**小青蛙跳荷叶，斐波那契**

n = int(input())  
def f(n):  
 if n == 1:  
 return 1  
 elif n == 2:  
 return 2  
 else:  
 a = 1  
 b = 2  
 for i in range(3, n+1):  
 a,b = b, a+b #注意这种写法可以避免RE  
 return b  
print(f(n))

**Pell数列**

def f(n):  
 if n == 1:  
 return 1  
 elif n == 2:  
 return 2  
 a1 = 1  
 a2 = 2  
 for i in range(3, n+1):  
 a1, a2 = a2, (2\*a2+a1)%32767  
 return a2  
n = int(input())  
for \_ in range(n):  
 t = int(input())  
 print(f(t))

**放苹果 n个苹果放k个盘子。类似因数分解和简单的整数划分**

s = int(input())  
for \_ in range(s):  
 way = 0  
 n,k = map(int,input().split())  
 dp = [[0] \* (k+1) for \_ in range(n+1)]

# dp(i,j)有i个苹果，j个盘子，有多少种放法  
 for i in range(n+1):  
 for j in range(k+1):  
 if i == 0 or i == 1 or j == 1:  
 dp[i][j] = 1 # 只有一个盘子，只有1个或0个苹果，0个苹果每个盘子都是空的  
 if j == 0: # 上边三种和j == 0不重合，不可能出现有苹果没盘子的情况。因为只要有苹果都可以放到盘子里  
 dp[i][j] = 0 # 没有盘子  
 for i in range(2, n+1):  
 for j in range(2,k+1):  
 dp[i][j] = dp[i][j-1] + dp[i-j][j]

# 有一个空盘子 + 没有空盘子，即每个盘子先放一个，剩下i-j个放到j  
 print(dp[n][k])

**去掉一个分数将会把他加减1也去掉，boredom**

n = int(input())  
arr=list(map(int,input().split()))  
cnt=[0]\*(max(arr)+1)  
for i in arr:  
 cnt[i]+=i # 桶  
dp=[0]\*(max(arr)+1)  
dp[0]=0  
dp[1]=cnt[1] # 初始条件  
for i in range(2,max(arr)+1):  
 dp[i]=max(dp[i-1],dp[i-2]+cnt[i]) #取i-1，放弃i-1  
print(max(dp))

**核电站：类似于概统题的排列数**

n, m = map(int, input().split())  
DP = [0] \* 60  
DP[0] = 1 #DP[i]是第i个位置的方案数。  
for i in range(1, n + 1):  
 if i < m: #达不到连续放置m个的情况  
 DP[i] = DP[i - 1] \* 2 # 从第1个到第m-1个，方案都可以选择放/不放  
 elif i == m: # 第m个要小心了  
 DP[i] = DP[i - 1] \* 2 - 1  
 else: # i>m  
 DP[i] = DP[i - 1] \* 2 - DP[i - m - 1]  
print(DP[n])

**合法的出栈顺序：结论**

import math

n = int(input())print(math.comb(2\*n,n)//(n+1))

**合法的出栈顺序：**

from functools import lru\_cacheimport sys

sys.setrecursionlimit(10000000)

n = int(input())

@lru\_cache(maxsize=None)

def f(i,j):

#栈内有i个，有j个待处理

if j == 0:

return 1

else:

if i == 0:

return f(i+1,j-1) # 栈空，只能进站

else:

return f(i-1, j) + f(i+1, j-1) #非空，可以出战可以进站

print(f(0,n))

**相似字符串**

def check(a, d):  
 similar = []  
 for v in d.values():  
 if a==v:  
 continue  
 elif len(a) == len(v):  
 count\_diff = sum(1 for j, k in zip(a, v) if j != k)  
 if count\_diff == 1:  
 similar.append(v)  
 elif len(a) == len(v) + 1:  
 for i in range(len(a)):  
 if a[:i] + a[i + 1:] == v:

# 删除单词A的一个字母后得到单词B  
 similar.append(v)  
 break  
 elif len(a) == len(v) - 1:  
 for i in range(len(v)):  
 if v[:i] + v[i+1:] == a:  
 similar.append(v)  
 break  
 return similar

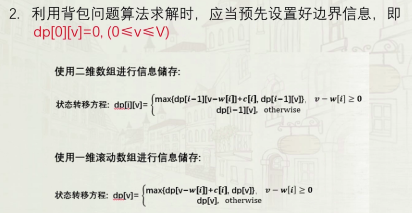
# Dp专题：

**几种思路**

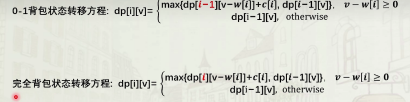
1. **只考虑前i个可选项时，能选的东西**
2. **第i个位置的最大、最小**
3. **一次有几种可能选择，最后选择最大、最小**

n = int(input())  
for \_ in range(n):  
 weight\_limit = int(input())  
 s = int(input())  
 tmp = list(map(int,input().split()))  
 weight = tmp[::2]  
 value = tmp[1::2]  
 unit\_value = [(value[i] / weight[i], weight[i], value[i]) for i in range(s)]  
 unit\_value = sorted(unit\_value, key=lambda x:-x[0]) # 从大到小  
 i = 0  
 bag = 0  
 while i < s and weight\_limit > 0:  
 if unit\_value[i][1] <= weight\_limit:  
 bag += unit\_value[i][2] # 这一类总价值都能带  
 weight\_limit -= unit\_value[i][1] # 总重量减去这一类重量  
 else:  
 bag += weight\_limit \* unit\_value[i][0] # 加上最多能装的价值  
 weight\_limit -= unit\_value[i][1] # 总重量减去这一类重量  
 i += 1  
 print("%.2f" %bag)

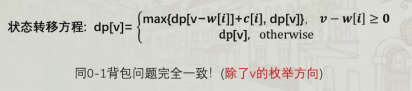
***01背包公式***



***完全背包公式（2维）***



***完全背包公式（1维）***



**数字三角形**

while True:

row = int(input())

if row == 0:

break

arr = [[0]\*row for \_ in range(row)]

for i in range(row):

arr[i][:i+1] = map(int,input().split())

x, y = map(int, input().split())

if row == 1:

print(arr[0][0])

continue

for i in range(row-2,-1,-1):

for j in range(i+1):

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

print(arr[x-1][y-1])

**01背包问题：一维压缩\*\*\*\*\*\*\*\*\*\*\*\*\*采药（最优解）**

T, M = map(int, input().split())  
time = [0]  
value = [0]  
for i in range(M):  
 t, v = map(int, input().split())  
 time.append(t)  
 value.append(v)  
bag = [0] \* (T + 1)

for i in range(1,M+1):  
 for j in range(T, 0, -1): # 倒着枚举  
 if time[i] <= j:  
 bag[j] = max(bag[j - time[i]] + value[i], bag[j])  
print(bag[T])

**01背包问题：一维压缩\*\*\*\*\*\*\*\*\*\*\*\*\*采药（最优解）**

T, M = map(int, input().split())  
bag = [-1] \* (T + 1)  
bag[0] = 0  
  
for i in range(M):  
 t, v = map(int, input().split())  
 for j in range(T, t-1, -1): # 倒着枚举  
 if bag[j-t] != -1:  
 bag[j] = max(bag[j - t] + v, bag[j])  
print(max(bag))

**最长递增子序列：**

a = int(input())  
l = list(map(int, input().split()))  
dp = [1] \* a # dp[i]表示以第i个数字结尾的最长递增子序列长度  
for i in range(1, a):  
 for j in range(i):  
 if l[j] < l[i]: #从第2个数开始每个数和他之前的数都能作为一个子序列！  
 dp[i] = max(dp[i], dp[j] + 1) #在已知的最大子序列的基础上加上1，这个1加的是dp[i]  
max\_length = max(dp) # 最长递增子序列的长度  
print(max\_length)

**Dp 最长递增子序列和：**

import copy  
n = int(input())  
num = list(map(int,input().split()))  
dp = copy.deepcopy(num)  
for i in range(n):  
 for j in range(i):  
 if num[j] < num[i]:  
 dp[i] = max(dp[i],dp[j] + num[i])  
print(max(dp))

**波兰表达式**

s = input().split()

def cal():

label = "+-\*/"

cur = s.pop(0)

if cur in label:

return str(eval(cal() + cur + cal()))

else:

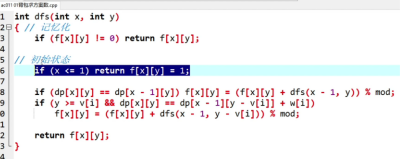
return cur

print("%.6f" % float(cal()))

**01背包问题：二维\*\*\*\*\*\*\*\*\*采药（最优解，注意恰好型要初始化为-无穷）**

T,M=map(int,input().split())  
time=[0]  
value=[0]  
for i in range(M):  
 t,v=map(int,input().split())  
 time.append(t)  
 value.append(v) # bag从1开始遍历  
bag=[[0 for i in range(T+1)] for j in range(M+1)]  
  
for i in range(1,M+1):  
 for j in range(1,T+1): # 正序遍历  
 if time[i]<=j:  
 bag[i][j]=max(bag[i-1][j-time[i]]+value[i],bag[i-1][j])  
 else:  
 bag[i][j]=bag[i-1][j]  
  
print(bag[-1][-1])

**背包问题求最优方案数：**



Print(dfs(n,m))



**神奇的口袋：0-1背包的方案数问题（dfs)，背包最大40，每种取一个**

from functools import lru\_cache

n = int(input())

v = []for \_ in range(n):

v.append(int(input()))

@lru\_cache(maxsize=None)def dfs(index, cur\_v):

res = 0

if index == n:

if cur\_v == 40:

return 1

return 0

for j in range(2):

res += dfs(index + 1, cur\_v + j \* v[index])

return res

print(dfs(0, 0))

**完全背包最优解问题：piggy bank，给定价格和重量，可以无限取**

T = int(input())  
for \_ in range(T):  
 emp, full = map(int,input().split())  
 bag = -emp + full  
 dp = [0] + [float('inf')] \* (bag + 1)  
 num = int(input())  
 for \_ in range(num): # 滚动数组  
 p, w = map(int,input().split())  
 for j in range(w,bag+1): # 正着遍历  
 dp[j] = min(dp[j - w] + p, dp[j])  
 if dp[bag] != float('inf'):  
 print('The minimum amount of money in the piggy-bank is {}.'.format(dp[bag]))  
 else:  
 print('This is impossible.')

**神奇的口袋 0-1背包方案数问题(dp)**

n = int(input())  
item = []  
for \_ in range(n):  
 item.append(int(input()))  
dp = [0] \* 41  
dp[0] = 1  
for weight in item:  
 for vol in range(40, weight - 1, -1):

#dp：容量为vol的最大方案数  
 dp[vol] += dp[vol-weight]  
print(dp[40])

**Coin 多重背包方案数问题，每种可以取几个(dp)**

while True:  
 n, m = map(int, input().split())  
 if n == 0:  
 break  
 data = list(map(int, input().split()))  
 value = data[:n]  
 counts = data[n:]  
 dp = [0] \* (m + 1)  
 dp[0] = 1  
 for i in range(n): # 滚动数组  
 coin, count = value[i], counts[i]  
 for j in range(count): # 遍历该面值数量  
 for v in range(m, coin - 1, -1):  
 dp[v] += dp[v - coin]  
 print(sum(1 for x in dp[1:] if x > 0))

**NBA门票 多重背包最优解问题**

n = int(input())  
if n %50 != 0:  
 print('Fail')  
 exit()  
n //= 50  
nums = list(map(int,input().split()))  
price = [1,2,5,10,20,50,100]  
dp = [0] + [float('inf')] \* n # 有n元钱时最小买多少票  
for i in range(7): # 注意滚动数组，i不作为dp的参数  
 cur\_num = nums[i]  
 cur\_price = price[i]  
 k = 1 # 二进制优化  
 while cur\_num > 0:  
 use\_num = min(cur\_num,k)  
 cur\_num -= use\_num  
 for j in range(n, use\_num \* cur\_price - 1, -1): # 细分背包空间，从大到小滚，因为小的状态会影响大的。  
 dp[j] = min(dp[j], dp[j - use\_num \* cur\_price]+use\_num)  
 k \*= 2  
if dp[n] == float('inf'):  
 print('Fail')  
else:  
 print(dp[n])

**最佳凑单（恰好多于某个数，套最优解问题模板）**

n, v = map(int, input().split()) # 商品数，凑单价  
a = list(map(int, input().split())) # 商品价格  
sum\_a = sum(a)  
dp = [-float("inf")] \* (sum\_a + 1)  
dp[0] = 0  
for i in range(n):  
 for j in range(sum\_a, a[i] - 1, -1):

# 从大到小滚动，细分背包  
 if dp[j - a[i]] >= 0:  
 dp[j] = max(dp[j], dp[j-a[i]] + a[i])  
  
if sum\_a < v:  
 print("0")  
else:  
 for k in range(v,sum\_a+1):  
 if dp[k] > 0:  
 print(dp[k])  
 break

**最佳凑单（恰好多于某个数，套方案数问题模板）**

n, t = map(int, input().split())  
a = list(map(int, input().split()))  
dp = [0] \* (sum(a) + 1)  
dp[0] = 1 # 注意初始化为1，什么都不取也是一种方案  
for i in a:  
 for j in range(sum(a), i - 1, -1): # 从大到小滚动，细分背包  
 dp[j] += dp[j - i]

**最佳凑单（套最优解问题模板）**

n, v = map(int, input().split()) # 商品数，凑单价  
a = list(map(int, input().split())) # 商品价格  
sum\_a = sum(a)  
dp = [[-float("inf")] \* (sum\_a + 1) for \_ in range(n + 1)]  
dp[0][0] = 0  
  
for i in range(1, n + 1): # 商品数  
 for j in range(sum\_a + 1): **# 价格，也是重量\*\*\*\*\***  
 if a[i - 1] <= j:  
 dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - a[i - 1]] + a[i - 1])  
 else:  
 dp[i][j] = dp[i - 1][j]

**导弹拦截：一维dp，初始化，最大不降子序列**

n=int(input())

height=[int(\_) for \_ in input().split()]

dp=[1 for \_ in range(n)] #表示以i位置结尾最长不减子序列的长度。注意，至少能拦截一个导弹，所以，每个初始都是1

for i in range(n):

for j in range(i): # j在i前

if height[j]>=height[i]:

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

print(max(dp))

**最长摆动子序列：**  
n = int(input())  
l = list(map(int, input().split()))  
up = down = 1  
for i in range(n - 1):  
 if l[i] < l[i + 1]: # 一个上升的部分  
 up = down + 1  
 if l[i] > l[i + 1]: # 一个下降的部分  
 down = up + 1  
print(max(down, up))

**下一个排列：结论记住**

def next\_permutation(arr, k):  
 # 交换arr中的两个元素  
 def swap(arr, i, j):  
 arr[i], arr[j] = arr[j], arr[i]  
  
 # 逆序排列，不是数字逆序，是顺序反过来。arr中从索引start到末尾的元素  
 def reverse(arr, start):  
 i, j = start, len(arr)-1  
 while i < j:  
 swap(arr, i, j)  
 i += 1  
 j -= 1  
 index = len(arr) - 2 # 倒数第二个 # 找出给定排列的下一个排列  
 while index >= 0 and arr[index] >= arr[index + 1]: # 找第一个不满足逆序的  
 index -= 1  
 if index >= 0: # 从后往前找第一个大于不满足逆序的  
 j = len(arr) - 1  
 while arr[j] <= arr[index]:  
 j -= 1  
 swap(arr, index, j) # 两个交换顺序  
  
 reverse(arr, index + 1) # 交换顺序后，从index以后都要逆序  
 next\_perm = arr  
 for \_ in range(k-1):  
 next\_perm = next\_permutation(next\_perm, 1)  
 return next\_perm  
  
tot=int(input())  
for \_ in range(tot):  
 n,k = map(int,input().split())  
 arr=list(map(int,input().split()))  
 print(' '.join(map(str, next\_permutation(arr,k)))) #列表逐个输出

next\_perm = next\_permutation(next\_perm, 1)

***回文字串：检查回文***

**def check(s):**

**if s == s[::-1]:**

**return True**

**return False**

**n = int(input())**

**for \_ in range(n):**

**l = list(input())**

**dp = [0] + [float('inf')] \* (len(l)) # 前i个里最少有多少个回文数**

**for i in range(1,len(l)+1): #因为切片，右开**

**for j in range(i):**

**if check(l[j:i]): # j == i 更新dp，每个字母自己都算一个**

**dp[i] = min(dp[i],dp[j]+1)**

**print(dp[len(l)] - 1)**

**更节约时间的回文（切割多少次变成回文）**

**def minP(n,s):  
 dp = [0] \* n  
 for i in range(n-2, -1, -1): # 倒数第二个到0  
 prev = 0  
 for j in range(i + 1, n): # i的后一个开始  
 temp = dp[j]  
 if s[i] == s[j]: # 当前位置与后边的数字相同  
 dp[j] = prev  
 else:  
 dp[j] = min(dp[j]+1, dp[j-1]+1)  
 prev = temp  
 return dp[n - 1]  
n= int(input())  
s = input()  
print(minP(n,s))**

**最大跳跃高度，奇偶时间喝药水（考虑前i个类型）**

**P = int(input())**

**l = []**

**for \_ in range(P):**

**l.append(int(input()))**

**dp = [[0] \* 2 for \_ in range(P+1)]**

**# 上次是1 这次就是0，上次是0，这次就是1**

**# 只有前i个药水时，奶牛在奇数1和偶数0时间的最大跳跃高度**

**for i in range(1,P+1):**

**dp[i][0] = max(dp[i-1][0],dp[i-1][1] - l[i-1])**

**# 前一种情况是没选，跳过了第i瓶，但是奇数时间的更新依赖偶数时间**

**dp[i][1] = max(dp[i-1][1],dp[i-1][0] + l[i-1])**

**print(max(dp[P][0],dp[P][1]))**

**红蓝玫瑰（考虑前i个类型）**

**rose = input()  
n = len(rose)  
dp = [[0] \* 2 for \_ in range(n + 1)]  
if rose[0] == 'R':  
 dp[0][0], dp[0][1] = 0, 1 # 让前1个都变成红的最小时间  
else:  
 dp[0][0], dp[0][1] = 1, 0  
for i in range(1, n): # 前i个都变成红：0 前i个都变成蓝：1  
 if rose[i] == 'R': # i从1开始  
 dp[i][0], dp[i][1] = min(dp[i - 1][0], dp[i - 1][1] + 1), min(dp[i - 1][0] + 1, dp[i - 1][1] + 1)  
 else:  
 dp[i][0], dp[i][1] = min(dp[i - 1][0] + 1, dp[i - 1][1] + 1), min(dp[i - 1][0] + 1, dp[i - 1][1])  
print(dp[n - 1][0])**

**股票买卖（考虑前i个和后i个类型）**

**T = int(input())  
for \_ in range(T):  
 n = int(input())  
 price = list(map(int, input().split()))  
 pre\_max = [0] \* (n + 1) # i天前的最大利润，只考虑到第i天能获得的利润  
 post\_max = [0] \* (n + 1) # i天后的最大利润  
 min\_p = price[0]**

**# 初始化，最小价格先设定为第一天的。pre\_max[0] = 0  
 max\_p = price[-1]**

**# 最大价格先设定为最后一天。pre\_max[-1] = 0  
 post\_max[-1] = -1**

**# 初始化，只考虑最后一天能获得的利润  
  
 for i in range(1, n):  
 min\_p = min(min\_p, price[i])  
 pre\_max[i] = max(pre\_max[i - 1], price[i] - min\_p)  
 # 前i天的最大利润有两种情况：前i-1天的最大利润，或第i天形成最大利润  
 for i in range(n - 2, -1, -1):  
 max\_p = max(max\_p, price[i])  
 post\_max[i] = max(max\_p - price[i], post\_max[i + 1])  
  
 max\_earn = 0  
 for i in range(1, n):  
 max\_earn = max(max\_earn, pre\_max[i] + post\_max[i])  
 print(max\_earn)**

**合唱队形、登山（考虑前i个和后i个）**

**n = int(input())  
l = list(map(int, input().split()))  
left=[1]\*n #在i位置最长上升子序列有多长  
for i in range(n):  
 for j in range(i):  
 if l[j]<l[i]:  
 left[i]=max(left[i],left[j]+1) #从左向右的上升子序列  
  
right=[1]\*n #在i位置从右向左最长上升子序列有多长  
for i in range(n-1,-1,-1):  
 for j in range(n-1,i,-1):  
 if l[i]-l[j]>0:  
 right[i]=max(right[i],right[j]+1) #从右向左的上升子序列  
  
max\_l=[]  
for i in range(n):  
 max\_l.append(left[i]+right[i]-1) #中间那个重复了，要减去一个  
  
print(n-max(max\_l)) # max\_l 队列长度**

**最长公共子序列：（可以不连续）**

**while True:  
 try:  
 n, m = input().split()  
 j = 0  
 dp = [[0] \* (len(n) + 2) for \_ in range(len(m) + 2)]  
 for i in range(1, len(m) + 1):  
 for j in range(1, len(n) + 1):**

**# dp[1][2] dp[1][3]……dp[2][1]……dp[len(m)][len(n)]都得到更新  
 if m[i - 1] == n[j - 1]: # 从0开始索引，同一位置相等，注意要对mn的索引要从0开始。  
 dp[i][j] = dp[i - 1][j - 1] + 1 # m前i-1和n前j-1的公共子序列，加上第i位置相等的这个字符（+1）  
 else:  
 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) #第i个字符不相等，那么就看m前i-1位置和n前j位置有多少相同的子序列，或者m前i位置和n的前j-1位置有多少（二者取最大）  
 print(dp[len(m)][len(n)])  
 except EOFError:  
 break**

**通配符匹配（考虑前i个类型）**

n = int(input())  
for \_ in range(n):  
 s = input()  
 p = input()  
 len\_s, len\_p = len(s), len(p)  
 s = "#" + s # 保护圈  
 p = "#" + p  
 dp = [[False] \* (len\_p+1) for \_ in range(len\_s+1)]  
 dp[0][0] = True # dp[i][j]，s的前i个和p的前j个能否匹配  
 for i in range(1,len\_p+1):  
 if p[i] == "\*": # 标记空字符串和\*能匹配  
 dp[0][i] = dp[0][i-1] #如果第i个是\*，那么两个能不能匹配要看p的前i-1和空字符串能不能匹配  
 for i in range(1, len\_s + 1):  
 for j in range(1, len\_p + 1):  
 if p[j] == "?" or p[j] == s[i]: # p第j 和 s第i能匹配  
 dp[i][j] = dp[i-1][j-1] # p前i和s前j能不能匹配取决于当前位置之前  
 elif p[j] == "\*": # s前i-1和p前j能匹配，或s前i和p前j-1能匹配就行  
 dp[i][j] = dp[i-1][j] or dp[i][j-1]  
 print("yes" if dp[len\_s][len\_p] else "no")

**假期，休息比赛和运动的最小休息天数**

n = int(input())  
l = list(map(int, input().split()))  
dp = [[120] \* 3 for \_ in range(n + 1)]  
dp[0][0] = dp[0][1] = dp[0][2] = 0 # 0 1 2 第i天选休息 比赛 运动，的最小休息时间  
for i in range(1, n + 1):  
 dp[i][0] = min(dp[i - 1]) + 1 # 任何时间都可以休息  
 if l[i - 1] == 1 or l[i - 1] == 3: # 可以比赛  
 dp[i][1] = min(dp[i - 1][2], dp[i - 1][0]) #选比赛，今天就不休息，前一天也不能比赛  
 if l[i - 1] == 2 or l[i - 1] == 3: # 可以运动  
 dp[i][2] = min(dp[i - 1][1], dp[i - 1][0]) #选运动，今天就不休息，前一天也不能比赛  
print(min(dp[n]))

**篮球**

**n = int(input())  
l1 = list(map(int, input().split()))  
l2 = list(map(int, input().split()))  
dp=[[0]\*3 for i in range(n+1)]  
for i in range(1,n+1):  
 dp[i][0]=max(dp[i-1]) # 不拿，不需要加l1或l2  
 dp[i][1]=max(dp[i-1][2],dp[i-1][0])+l1[i-1] # 拿l1，基础是前一个不拿或者拿第二个（取最大）  
 dp[i][2]=max(dp[i-1][1],dp[i-1][0])+l2[i-1] # 拿l2，基础是前一个不拿或拿第一个（取最大）  
print(max(dp[n]))**

**三组数相等**

**n = int(input())  
a = [int(x) for x in input().split()]  
k = sum(a) # 总和  
if n < 3 or k % 3 != 0: #不能的情况  
 print(0)  
else:  
 s = 0  
 ans = 0  
 for i in range(1, n):  
 a[i] += a[i - 1] # 每一个位置都是它及它前边数的和  
 for i in range(n - 1):  
 if a[i] \* 3 == k \* 2: # 找到了后1/3  
 ans += s # 找到后1/3才算找完，才能把前1/3加到结果上  
 if a[i] \* 3 == k: # 找到了前1/3  
 s += 1  
 print(ans)**

**宠物小精灵：二维限制的dp  
N, M, K = map(int, input().split()) # N：精灵球总数，M：初始HP，K：数量  
dp = [[-1] \* (M + 1) for i in range(K + 1)] # dp[i][j] 剩余 j HP， 收 i 个 Pokemon，剩余的最大球数  
dp[0][M] = N  
for i in range(1, K+1): # 所有精灵个数  
 ball, dmg = map(int, input().split())**

**商品数for q in range(i, 0, -1):**

**# 滚动数组从后向前 q: Pokemon个数  
容量 for p in range(M - dmg + 1):**

**# 滚动数组从前向后 p: HP  
 if dp[q - 1][p + dmg] != -1:  
 # 使用 q-1 个球后剩余体力为 p+dmg 的情况是否存在  
 # 也就是使用 q 个球前一个状态是否存在**

**dp[q][p] = max(dp[q][p], dp[q - 1][p + dmg] - ball)  
for i in range(K, -1, -1):  
 for j in range(M, -1, -1):  
 if dp[i][j] != -1:  
 ans = [i, j] # [个数，HP]  
 print(' '.join(map(str, ans)))  
 exit()**

**拔河游戏**

**import math  
def tug\_of\_war(n, weights):  
 total\_sum = sum(weights)  
 n1 = math.ceil(n/2) #//个单数的个数加1来找，不然找出的小的那一半可能会出错  
 target\_sum = math.ceil(total\_sum / 2) #单数的话加一个避免边界出错  
 dp = [[False] \* (target\_sum + 10) for \_ in range(n1+10)]  
 dp[0][0] = True  
  
 for i in range(n): #所有人数  
 for j in range(n1, 0, -1): #从一半人数开始向下  
 for k in range(target\_sum, weights[i]-1, -1): #从一半重量开始向下（从后向前是因为后边的情况只用考虑前边的情况）  
 if dp[j-1][k-weights[i]]: #包括j人数-1人的队有k-weights[i]的重量，这种情况是否可行  
 dp[j][k] = True #可行，则加上第i个人的重也可行  
  
 team1\_sum = 0  
 for i in range(target\_sum, -1, -1): #因为在单数在前面加了1，很可能在这里找不到  
 if dp[n1][i]:  
 team1\_sum = i  
 break  
  
 if team1\_sum == 0: #上面找不到在这里接着找  
 for i in range(target\_sum, -1, -1):  
 if dp[n1-1][i]:  
 team1\_sum = i  
 break  
  
 team2\_sum = total\_sum - team1\_sum  
 return min(team1\_sum, team2\_sum), max(team1\_sum, team2\_sum)  
  
n = int(input())  
weights = []  
for \_ in range(n):  
 weights.append(int(input()))  
  
result = tug\_of\_war(n, weights)  
print(result[0], result[1])**

**开餐馆：**

**T = int(input())  
for \_ in range(T):  
 n, k = map(int, input().split())  
 place = list(map(int, input().split()))  
 profit = list(map(int, input().split()))  
 dp = [0] \* (n + 1) # 考虑前i个，第i个不开的最大利润  
 dp[0] = profit[0] # 第0个开，利润就是开的利润  
  
 for i in range(1, n):  
 dp[i] = max(dp[i - 1], profit[i]) # 不受距离影响的部分，不开或只开i  
 for j in range(i):  
 if place[i] - place[j] > k: # 没有等号  
 dp[i] = max(dp[j] + profit[i], dp[i]) # 比较受距离影响的部分。j和i距离足够大，开j的最大利润+i的利润  
 print(dp[n - 1])**

**最大子矩阵：**

**n = int(input())  
m = []  
while len(m) < n \* n:  
 m.extend(list(map(int, input().split())))  
matrix = [m[i \* n:(i + 1) \* n] for i in range(n)]  
def maxSubMatrix(matrix,n):  
 max\_sum = 0  
 for i in range(n):  
 dp = [0] \* n  
 for j in range(i, n): #  
 for k in range(n):  
 dp[k] += matrix[j][k] # dp1234n先赋为原矩阵第一行的各个数字。  
 # 下一轮循环变成第一行各个数字加上第二行各个数字  
 max\_sum = max(max\_sum, maxSubArray(dp)) #maxSA找最大子数组的和，不断更新最大子矩阵的和  
 return max\_sum  
def maxSubArray(nums):  
 max\_sum = float('-inf')  
 cur\_sum = 0  
 for num in nums: #n um是传入列表中的元素遍历  
 cur\_sum = max(num, cur\_sum + num) # cur\_sum + num相当于算了从第一个到元素到最后一个元素的和  
 # 但这个和不一定是一直累积，发现一个特别大的num就重新开始记录。例如2 -3 显然第二次还是2.  
 # 如果当前扫的数比之前所有扫过的数加和都大，那么就从当前扫的数开始记  
 max\_sum = max(max\_sum, cur\_sum) # 刷新一遍最大和  
 return max\_sum  
print(maxSubMatrix(matrix,n))**

**田忌赛马：**

**while True:  
 n = int(input())  
 if n == 0:  
 break  
 tian = [0]+list(map(int, input().split()))  
 king = [0]+list(map(int, input().split()))  
 tian[1:] = sorted(tian[1:],reverse=True)  
 king[1:] = sorted(king[1:],reverse=True)  
 dp = [[0] \* (n + 1) for \_ in range(n + 1)] # 边界，没有出马就不变，只考虑tian的前i个和王的前j个  
 for i in range(1, n + 1):  
 for j in range(1, n + 1):  
 if tian[i] > king[j]:  
 w = 2  
 dp[i][j] = max(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1] + w)  
 elif tian[i] == king[j]:  
 w = 1  
 dp[i][j] = max(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1] + w)  
 else:  
 w = 0  
 dp[i][j] = max(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1] + w)  
 print((dp[n][n] - n) \* 200)**

**布线问题：**

**N = 100  
n = int(input("请输入端点个数："))  
a = [0] \* N  
dp = [[0] \* N for \_ in range(N)]  
print("请输入各端点对应的下端点：")  
a = list(map(int,input().split()))  
  
for i in range(1, n + 1): # 上端点为1的情况  
 if i < a[1]:  
 dp[1][i] = 0  
 else:  
 dp[1][i] = 1 # 有一根有效线  
  
for i in range(2, n + 1): # 其他情况  
 for j in range(1, n + 1):  
 if j < a[i]: # 小于对应线的坐标  
 dp[i][j] = dp[i-1][j]  
 else: # 大于等于  
 if dp[i-1][j] > dp[i-1][a[i]-1] + 1: # 更新dp最大值  
 dp[i][j] = dp[i-1][j]  
 else:  
 dp[i][j] = dp[i-1][a[i]-1] + 1  
print(dp[n][n])**

**汉诺塔**

**def move\_p(num, init, dest):  
 print("{}:{}->{}".format(num,init,dest))  
def move(num,init,temp,dest):  
 if num == 1:  
 move\_p(1,init,dest)  
 else:  
 move(num-1,init,dest,temp) # 最初，中转，最终  
 move\_p(num,init,dest)  
 move(num-1,temp,init,dest)  
num,a,b,c = input().split()  
move(int(num),a,b,c)**

**汉诺塔4塔**

**def hanoi\_four\_towers(n, source, target, auxiliary1, auxiliary2):**

**if n == 0:**

**return 0**

**if n == 1:**

**return 1**

**min\_moves = float('inf')**

**for k in range(1, n):**

**three\_tower\_moves = 2\*\*(n-k)-1**

**moves = hanoi\_four\_towers(k, source, auxiliary1, auxiliary2, target) +\**

**three\_tower\_moves +\**

**hanoi\_four\_towers(k, auxiliary1, target, source, auxiliary2)**

**min\_moves = min(min\_moves, moves)**

**return min\_moves**

**for n in range(1, 13):**

**print(hanoi\_four\_towers(n, 'A','D','B','C'))**

**汉诺塔4塔**

**d = [0] \* 13  
f = [float('inf')] \* 13  
  
d[1] = 1  
for i in range(2, 13): # 三塔汉诺塔问题的解  
 d[i] = 2\*\*i - 1  
f[1] = 1  
for i in range(2, 13):  
 for j in range(1, i):  
 f[i] = min(f[i], f[i - j] \* 2 + d[j])  
  
for i in range(1, 13):  
 print(f[i])**

# 几个数之和

**Counter**

from collections import Counter  
n=int(input())  
for \_ in range(n):  
 set1=[]  
 set2=[]  
 s=int(input())  
 set1\_num=int(input())  
 set1=list(map(int,input().split()))  
 set2\_num=int(input())  
 set2=list(map(int,input().split()))  
 res=0  
 counter1=Counter(set1)  
 counter2=Counter(set2)  
 for i in counter1:  
 if s-i in counter2:  
 res+=counter1[i]\*counter2[s-i]  
 print(res)

**三数之和，counter版本**

from collections import Counter  
nums = list(map(int, input().split()))  
nums.sort()  
numdict = Counter(nums)  
ans = set()  
for idx, a in enumerate(nums):  
 if a > 0:  
 break  
 for b in nums[idx+1:]:#Counter.items()键是原始数字，值是出现频率  
 if -a-b in numdict and all(numdict[n] >= cnt for n,cnt in Counter([a,b,-a-b]).items()): ans.add(tuple(sorted([a,b,-a-b])))  
print(len(ans))

**三个数之和：**

def threeSum(nums):  
 def threeSum(nums):  
 nums.sort()  
 count = 0  
 n = len(nums)  
 for i in range(n-2):  
 if nums[i] > 0:  
 break  
 if i > 0 and nums[i] == nums[i-1]:  
 continue # 跳过重复元素  
 left = i + 1  
 right = n - 1  
 while left < right:  
 total = nums[i] + nums[left] + nums[right]  
 if total == 0:  
 count += 1  
 left += 1  
 right -= 1  
 while left < right and nums[left] == nums[left-1]:  
 left += 1 # 跳过重复的元素  
 while left < right and nums[right] == nums[right+1]:  
 right -= 1  
 elif total > 0:  
 right -= 1  
 else:  
 left += 1  
 return count  
nums = list(map(int, input().split()))  
result = threeSum(nums)  
print(result)

# 双指针

**'''目标最近的两数之和 双指针'''**  
T = int(input())  
num = list(map(int, input().split()))  
num.sort()  
k = res = float('inf')  
le = 0  
ri = len(num) - 1  
  
while le < ri:  
 ans = num[le] + num[ri]  
 if ans == T:  
 print(ans)  
 exit()  
 elif ans > T: #大于目标就减去  
 ri -= 1  
 else: # 小于目标就加上  
 le += 1  
 if abs(T - ans) < k: # 记录最小间距  
 k = abs(T - ans)  
 res = ans  
 if abs(T - ans) == k:  
 res = min(res, ans)  
print(res)  
  
**'''缩减字符 双指针'''**  
n = input().lower() + "\*" # 保护圈，保证最后一次能出去  
s = -1  
e = 0  
for i in range(1, len(n)):  
 if n[i] != n[i - 1]:  
 e = i - 1 # 结束位置的索引  
 print("({},{})".format(n[i - 1], e - s), end ="")  
 s = e #开始位置  
  
**'''hello 双指针'''**  
s = input()  
hello = "hello"  
i = 0  
j = 0  
while i < len(s) and j < len(hello): #双指针，也能防止j越界  
 if s[i] == hello[j]: #hello是字符串  
 j += 1 #发现一个字母'hello'就往后走一个，如果顺序是错的，错过一个就再也找不到了！  
 i += 1 #对于输入的字符串，每次都要走一个  
if j == len(hello): #如果刚好j的长度等于hello  
 print("YES")  
else:  
 print("NO")  
  
**'''军备竞赛 双指针'''**  
p = int(input())  
cost = list(map(int, input().split()))  
cost.sort()  
l = 0  
r = len(cost) - 1  
gap = 0  
while l <= len(cost) - 1:  
 if cost[l] <= p: # 有钱就买  
 p -= cost[l]  
 l += 1  
 gap += 1  
 else:  
 if l >= r: # 双指针的条件  
 break  
 p += cost[r]  
 r -= 1  
 gap -= 1  
 if gap < 0: # 差异不能小于0  
 gap = 0  
 break  
print(gap)

四元组四个数之和等于0：

import array as arr # 紧凑数组  
n = int(input())  
a = arr.array('i', [0]\*(n+1))  
b = arr.array('i', [0]\*(n+1))  
c = arr.array('i', [0]\*(n+1))  
d = arr.array('i', [0]\*(n+1))  
for i in range(n):  
 a[i],b[i],c[i],d[i] = map(int,input().split())  
dic1 = {}  
for i in range(n):  
 for j in range(n):  
 if not (a[i]+b[j] in dic1):  
 dic1[a[i]+b[j]] = 0  
 dic1[a[i]+b[j]] += 1  
res = 0  
for i in range(n):  
 for j in range(n):  
 if -(c[i]+d[j]) in dic1:  
 res += dic1[-(c[i]+d[j])]  
print(res)

# 二分

**二分查找：浮点数，网线主管**

def check(mid):

cnt = 0

for i in range(len(l)):

cnt += l[i]//mid # 根数 是整数

if cnt >= k: # mid小

return True

else:

return False

def binary\_s(le,ri):

while ri - le > 1:

mid = (le + ri)//2

# 因为不可能有小于1cm的段所以当作整数做

if check(mid):

le = mid

else:

ri = mid

return le

n, k = map(int,input().split())

l = []for \_ in range(n):

q = float(input())

l.append(int(q\*100))

max\_a = sum(l) // k

max\_limit = max(l) + 1

if max\_a < 1:

print("0.00")

else:

print("{:.2f}".format(binary\_s(0,max\_limit)/100))

**不定方程求解**

def check(x):

return x\*\*3 - 5\*x\*\*2 + 10\*x - 80

def bisect\_right(lo = 0, hi = 10): # lo下界，hi上界

while hi - lo > 1e-10:

mid = (lo + hi) / 2

if check(mid) < 0:

lo = mid # 连续型不要搞加1

else:

hi = mid

return lo

print("%.9f" % bisect\_right(0,10))

**二分查找 木材加工**

n, k = map(int, input().split())

arr = []

for i in range(n):

arr.append(int(input()))

if sum(arr) < k:

print("0")

elif sum(arr) == k:

print("1")

else:

def check(x):

count = 0

for i in range(n):

count += arr[i] // x

if count >= k:

return True

else:

return False

def binary\_search(left, right):

while left < right:

mid = (left + right) // 2 # mid必须在循环里

if check(mid): # 猜小了，可行

left = mid + 1 # 标准

else:

right = mid

return left - 1

r = max(arr) + 1

print(binary\_search(0, r))

**二分查找：放弃考试：**

from decimal import Decimal

def check(v,r,c,m): #mid值进入，原始的两个数组（分别代表考试得分和考试总数），放弃的考试数

y=[r[i]-v\*c[i] for i in range(len(r))] #F(v)>0等价于r[i]-mid\*c[i]大于0

y.sort() #从小到大排y[i]

if sum(y[m:])>0: #要考察的是如果存在没放弃的比赛得到的y[i]

return True

else:

return False

def bisect\_right(r, c, m): #lo下界，hi上界

lo = 0

hi = 100

while hi-lo > 1e5: #v从0到1之间枚举，注意这个题目可能除出来很小的double，如果限定>0就会一直死循环

mid = (lo + hi) / 2 # 找中间index（二分）,从mid开始猜,和源码不同在于这个是连续的

if check(mid,r,c,m): # v猜小了（此使F(v)大于0，因为F(v)为0的时候最优，又因为F(v)和v反比，

# 所以希望F(v)变小一直到0，所以下一次v应该变大。因此下一次从中间到最大猜

lo = mid

else: # v猜大了，下一次从最小到中间猜

hi = mid

return lo\*100

while True:

n,m=map(int,input().split())

if n==0 and m==0:

break

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

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

print(Decimal(bisect\_right(r,c,m)).quantize(Decimal("1."),rounding="ROUND\_HALF\_UP")) # 保留小数点后一位，四舍五入

L, N, M = map(int, input().split()) # 终点L，起点到终点之间有N个，能移走M个  
stone = [0] # 起点  
for \_ in range(N):  
 stone.append(int(input()))  
stone.append(L) # 加上终点  
  
**河中跳房子**  
def check(mid):  
 num = 0  
 now = 0  
 for i in range(1, len(stone)):  
 if stone[i] - now < mid:  
 num += 1  
 else:  
 now = stone[i]  
 if num > M:  
 return True # 删除太多了，符合要求的距离比现在设定的短  
 else:  
 return False # 删太少了，符合要求的距离比现在设定的长  
def bisect\_stone(lo=0, hi=L + 1): # L是从起点到终点的距离，假如所有石头都能移走，那么最长的最短跳跃距离是L  
 while lo < hi:  
 mid = (lo + hi) // 2  
 if check(mid):  
 hi = mid  
 else:  
 lo = mid + 1 # 如果不+1就死循环，注意和连续型二分的不同  
 return (lo - 1)  
print(bisect\_stone(lo=0, hi=L + 1))

**月度开销**

m, n = map(int, input().split())  
ex = []  
for i in range(m):  
 ex.append(int(input()))  
def check(mid): # 枚举最大月度开销，判断在这个开销下的周期数会分出多少个。  
 # 如大于限制就说明最大月度开销设计太小  
 sum\_ex, num = 0, 1  
 for i in range(m):  
 if sum\_ex + ex[i] > mid: # 发现超过我设定的最大值了  
 sum\_ex = ex[i] # 分一个新的周期，这个周期目前花费ex[i]  
 num += 1 # 周期数加一  
 else:  
 sum\_ex += ex[i]  
 if num > n: # 目前的月度开销太小了，  
 return True  
 else: # 目前的月度开销大了，或者num等于n了  
 return False  
def binary\_search(le, ri):  
 while le < ri:  
 mid = (le + ri) // 2  
 if check(mid):  
 le = mid + 1  
 else:  
 ri = mid  
 return le  
  
le = max(ex)  
ri = sum(ex) + 1  
print(binary\_search(le, ri))

**Agressive cows：**

n, c = map(int,input().split())  
pos = []  
for \_ in range(n):  
 pos.append(int(input()))  
pos.sort()  
  
def check(mid):  
 now,num = min(pos), 1 # 初始从最小栏开始看，不管在哪里，先可以放一头牛  
 for i in range(1,n):  
 if pos[i] - now >= mid: # 符合条件，mid是最小距离  
 num += 1  
 now = pos[i] # 更新上一个点  
 if num >= c: # 找小了，可以放的牛数量多于需要的，注意等于号。（等于号代表恰好满足的时刻，return left所以取等的时候必须更新le）  
 return True  
 else:  
 return False # 找大了  
  
def binary\_search(le,ri):  
 while le < ri:  
 mid = (le + ri)//2  
 if check(mid):  
 le = mid + 1  
 else:  
 ri = mid  
 return le - 1  
  
max\_len = max(pos) - min(pos) + 1  
print(binary\_search(0,max\_len))

**二分查找，查找最相近的元素：**

import bisect  
n = int(input())  
l = list(map(int, input().split()))  
m = int(input())  
for \_ in range(m):  
 search = int(input())  
 index = bisect.bisect\_left(l, search) # 如果出现多次，就插在左边  
 print(index)  
 if index >= len(l): # 比所有数都大  
 print(l[-1])  
 elif index == 0:  
 print(l[0])  
 else:  
 if l[index] - search >= search - l[index-1]: # 前一个和后一个比，插入之后还需要比较和左右两边哪个最小  
 print(l[index-1])  
 else:  
 print(l[index])

**和为给定数**

import bisect  
def binary\_search(arr, target, left, right):  
 index = bisect.bisect\_left(arr,target,left,right)  
 if index == len(arr): # 插入的数大过所有的了  
 return False  
 else:  
 if arr[index] == target: # 刚好是目标  
 return True  
 else:  
 return False  
  
n = int(input())  
a = list(map(int, input().split()))  
m = int(input())  
a.sort()  
  
for i in range(n - 1): # 已知a[i]，每次搜索的右边界向右移动一个，以找到第二个数  
 if binary\_search(a, m - a[i], i + 1, n - 1):  
 print(a[i], m - a[i])  
 break  
else:  
 print("No")

# DFS BFS

**Dfs BFS专题：拯救公主，迷宫+贪心**

from collections import deque

dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]

s = int(input())

for \_ in range(s):#多组输入

flag = 0

row, col = map(int, input().split())

arr = []

for \_ in range(row):

arr.append(list(input()))

queue = deque()

for i in range(row):

for j in range(col):

if arr[i][j] == "r":

start = (i, j)

arr[i][j] = "#" # 变成墙才能不再走

if arr[i][j] == "a":

end = (i, j)

arr[i][j] = "@" # 变成路才能走到

visited = [[0] \* col for \_ in range(row)]

# visited[start[0]][start[1]] = 1 # 开始的路不要重新走，已经标记了“#”也可以不需要这一步

queue.append((start[0], start[1], 0))

ans = []

while queue:

x, y, step = queue.popleft()

if (x, y) == end:

visited[x][y] = 0 # 标记终点为没走过，否则就走不到

ans.append(step) # 都装进去，取时间最短

flag = 1

if arr[x][y] == "x": # 分开处理路和守卫，拉平打守卫和正常路的距离

arr[x][y] = "@" # 打守卫

queue.append((x, y, step + 1)) # 下一步可以通过守卫，即下一次可以进下面的else

else: # 不是守卫，可能是路或墙壁

for d in range(4):

nx = x + dire[d][0]

ny = y + dire[d][1]

if 0 <= nx <= row - 1 and 0 <= ny <= col - 1: # 越界

if arr[nx][ny] == "#": # 是墙壁则不处理，跳过

continue

if visited[nx][ny] == 0: # 没走过，nx ny可能是路也可能是守卫

visited[nx][ny] = 1 # 标记走过

queue.append((nx, ny, step + 1))

if flag == 0:

print("Impossible")

else:

print(min(ans))

**寻宝：迷宫BFS**

from collections import deque  
dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]  
row, col = map(int, input().split())  
arr = [[10] \* (col + 2)]  
for \_ in range(row):  
 arr.append([10] + [int(x) for x in input().split()] + [10])  
arr.append([10] \* (col + 2)) #输入加保护圈

for i in range(1, row + 1):  
 if end:  
 break  
 for j in range(1, col + 1):  
 if arr[i][j] == 1:  
 arr[i][j] = 0  
 end = [i, j] #找终点，标记终点可行  
 break  
  
visited = [[0] \* (col + 2) for \_ in range(row + 2)]  
visited[1][1] = 1  
queue = deque()  
queue.append((1, 1, 0))  
while queue:  
 x, y, step = queue.popleft()  
 if [x, y] == end:  
 print(step)  
 exit() # 直接退出，和flag一样  
 for d in range(4):  
 nx = x + dire[d][0]  
 ny = y + dire[d][1]  
 if 1 <= nx <= row and 1 <= ny <= col: # 必须判断有没有越界，有保护圈也不例外  
 if arr[nx][ny] == 0 and visited[nx][ny] == 0:  
 visited[nx][ny] = 1 # 标记已走过避免重新走  
 queue.append((nx, ny, step + 1))  
print("NO")

**鸣人和佐助 BFS**

from collections import deque

M, N, T = map(int, input().split())

d = [[0, 1], [1, 0], [0, -1], [-1, 0]]

arr = [['0'] \* (N + 2) for \_ in range(M + 2)]

visited = set()

for i in range(1, M + 1):

arr[i][1:-1] = input()

for i in range(1, M + 1):

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

if arr[i][j] == "@":

start = (i, j, T, 0)

visited.add((i,j,T))

queue = deque([start])

while queue:

x, y, Chakra, step = queue.popleft()

for dire in d:

nx, ny = x + dire[0], y + dire[1]

if arr[nx][ny] == "\*" and (nx, ny, Chakra) not in visited:

queue.append((nx, ny, Chakra, step + 1))

visited.add((nx, ny, Chakra))

# 守卫点可以看作拆点

if arr[nx][ny] == "#" and Chakra > 0 and (nx, ny, Chakra-1) not in visited: # 还有能量，能打守卫

queue.append((nx, ny, Chakra - 1, step + 1))

visited.add((nx, ny, Chakra - 1))

if arr[nx][ny] == "+": # 不预先处理符号

print(step+1)

exit()

print("-1")

**跨步迷宫 BFS**

for d in dire:

nx = x + d[0]

ny = y + d[1]

nx\_2 = x + 2 \* d[0]

ny\_2 = y + 2 \* d[1]

if 1 <= nx <= row and 1 <= ny <= col: # 先判一步越界不越界，因为不允许跨步穿墙

if (nx, ny) not in vis and arr[nx][ny] == 0:

vis.add((nx, ny))

q.append((nx, ny, step + 1))

if 1 <= nx\_2 <= row and 1 <= ny\_2 <= col: # 再判两步越界不越界

if (nx\_2, ny\_2) not in vis and arr[nx\_2][ny\_2] == 0:

vis.add((nx\_2, ny\_2))

q.append((nx\_2, ny\_2, step + 1))

**传送点迷宫问题 BFS**

from collections import deque  
dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]  
row, col = map(int, input().split())  
tranV = []  
arr = [['1'] \* (col + 2)]  
for i in range(1, row + 1):  
 r = ['1'] + [x for x in input().split()] + ['1'] # 注意有空格  
 arr.append(r)  
 if '2' in r: # 只写一个循环，找到两个传送点  
 for idx, val in enumerate(r):  
 if val == '2':  
 tranV.append((i, idx))

arr.append(['1'] \* (col + 2))  
def transMap(a): # 定义一个映射  
 for v in tranV:  
 if v != a:  
 return v  
  
q, vis = deque(), set()  
q.append((1, 1, 0))  
vis.add((1, 1))  
while q:  
 x, y, step = q.popleft()  
 if x == row and y == col:  
 print(step)  
 exit()  
 for d in dire:  
 nx, ny = x + d[0], y + d[1]  
 if 1 <= nx <= row and 1 <= ny <= col:  
 if arr[nx][ny] != '1' and (nx, ny) not in vis:  
 q.append((nx, ny, step+1))  
 vis.add((nx, ny))  
 if arr[nx][ny] == '2': # 是传送点要特判  
 other = transMap((nx, ny))  
 if other not in vis:  
 q.append((other[0],other[1],step+1))  
 vis.add(other)  
print(-1)

**马走日 BFS，有多少途径遍历所有点**

x = int(input())  
dire = [[-2, 1], [2, -1], [2, 1], [-2, -1], [1, 2], [-1, 2], [1, -2], [-1, -2]]  
for \_ in range(x): # 多组数据输入  
 tot, res = 0, 0  
 def dfs(res, x, y):  
 global tot # 多组数据有全局变量lru  
 if res == n \* m: # dfs参数res，走过一个点就加1  
 tot += 1 # 遍历所有点的路径数  
 return  
 s[x][y] = 1 # 和迷宫可行路径数一样  
 for i in range(8):  
 nx = x + dire[i][0]  
 ny = y + dire[i][1]  
 if s[nx][ny] == 0:  
 s[nx][ny] = 1  
 dfs(res + 1, nx, ny)  
 s[nx][ny] = 0 # 下一次还能走  
 s[x][y] = 0  
 n, m, x, y = map(int, input().split())  
 s = [[500] \* (m + 4) for i in range(n + 4)]  
 for i in range(2, n + 2): # 更大的保护圈  
 s[i][2:-2] = [0] \* m  
 x += 2  
 y += 2 # 调整坐标  
 # s[x][y] = 1 # 起点在函数外标记，永远都不能再走  
 dfs(1, x, y)  
 print(tot)  
 tot = 0

from collections import deque  
dire = [(-2, 1), (2, -1), (2, 1), (-2, -1), (1, 2), (-1, 2), (1, -2), (-1, -2)]  
row, col, x, y = map(int, input().split())  
k = int(input())  
barrier = set()  
for \_ in range(k):  
 a, b = map(int, input().split())  
 barrier.add((a, b))  
up1,down1,right1,left1 = [(-2, 1), (-2, -1)], [(2, 1), (2, -1)], [(1, 2), (-1, 2)], [(1, -2), (-1, -2)] # 上下右左  
up,down,right,left = set(up1),set(down1),set(right1),set(left1)  
  
temp = [[-1,0],[1,0],[0,1],[0,-1]]  
q, vis = deque(), set()  
ans = [['-1'] \* (col + 2) for \_ in range(row + 2)]  
vis.add((x, y))  
q.append((x, y, 0))

while q:  
 x, y, step = q.popleft()  
 ans[x][y] = str(step)  
 b = [] # 清空障碍方向列表  
 for td in temp: # 每一步都要重新判  
 tx,ty = x + td[0], y + td[1]  
 if (tx,ty) in barrier:  
 b.append((td[0],td[1])) # 障碍方向加入列表  
  
 valid\_d = set(dire) # 每走一步都要重置valid\_d  
 for b\_dire in b:  
 if b\_dire[0] == -1: # 上  
 valid\_d.difference\_update(up)  
 elif b\_dire[0] == 1:  
 valid\_d -= down # 集合减法  
 elif b\_dire[1] == 1: # 右  
 valid\_d.difference\_update(right)  
 else:  
 valid\_d.difference\_update(left)  
  
 if len(valid\_d) > 0:  
 for d in valid\_d:  
 nx, ny = x + d[0], y + d[1]  
 if 1 <= nx <= row and 1 <= ny <= col:  
 if (nx, ny) not in vis and (nx,ny) not in barrier: # 已经被障碍棋子占据了  
 vis.add((nx, ny))  
 q.append((nx, ny, step + 1))  
for i in range(1,row+1):  
 print(\*ans[i][1:-1])

**螃蟹采蘑菇 DFS**

import sys  
sys.setrecursionlimit(100000000)  
n = int(input())  
dire = [[-1, 0], [1, 0], [0, -1], [0, 1]]  
start = []  
cnt = 0  
  
def dfs(x1, y1, x2, y2, flag):  
 global cnt  
 if (x1, y1) == des or (x2, y2) == des:  
 cnt += 1  
 return  
 for d in range(4):  
 nx1 = x1 + dire[d][0]  
 nx2 = x2 + dire[d][0]  
 ny1 = y1 + dire[d][1]  
 ny2 = y2 + dire[d][1]  
 if flag == 1: # 横着，通过卡条件模拟移动  
 if d <= 1 and arr[nx1][ny1] == 0 and arr[nx2][ny2] == 0: # 上下走，如果不在if内，会爆栈  
 arr[nx1][ny1] = 1  
 arr[nx2][ny2] = 1 # 原地修改为墙壁  
 dfs(nx1, ny1, nx2, ny2, flag)  
 if d == 2 and arr[nx1][ny1] == 0: # 左走  
 arr[nx2][ny2] = 1  
 dfs(nx1, ny1, nx2, ny2, flag)  
 if d == 3 and arr[nx2][ny2] == 0: # 右走  
 arr[nx1][ny1] = 1  
 dfs(nx1, ny1, nx2, ny2, flag)  
  
 else:  
 if d > 1 and arr[nx1][ny1] == 0 and arr[nx2][ny2] == 0: # 左右走  
 arr[nx1][ny1] = 1  
 arr[nx2][ny2] = 1  
 dfs(nx1, ny1, nx2, ny2, flag)  
 if d == 0 and arr[nx1][ny1] == 0: # 上走  
 arr[nx2][ny2] = 1  
 dfs(nx1, ny1, nx2, ny2, flag)  
 if d == 1 and arr[nx2][ny2] == 0: # 下走  
 arr[nx1][ny1] = 1  
 dfs(nx1, ny1, nx2, ny2, flag)  
 return  
  
flag = 0  
arr = [[500] \* (n + 2)]  
for i in range(n):  
 arr.append([500] + [int(\_) for \_ in input().split()] + [500])  
arr.append([500] \* (n + 2))  
for i in range(1, n + 1):  
 for j in range(1, n + 1):  
 if arr[i][j] == 5:  
 start.append((i, j)) # 处理起点和终点  
 arr[i][j] = 1  
 if arr[i][j] == 9:  
 des = (i, j)  
 arr[i][j] = 0  
  
start\_1, start\_2 = start[0], start[1]  
x1, y1, x2, y2 = start\_1[0], start\_1[1], start\_2[0], start\_2[1]  
if x1 == x2: # 横着  
 flag = 1  
  
dfs(x1, y1, x2, y2, flag)  
if cnt != 0:  
 print("yes")  
else:  
 print("no")

**数字接近 抓住那头牛BFS**

from collections import deque  
maxn = int(1e5) # 设定边界  
visited = [-1] \* (maxn\*2+100)  
a = [0]\*4  
n = int(input())  
q = deque()  
q.append((1,0))  
visited[1] = 1  
while q:  
 pos, step = q.popleft()  
 if pos == n:  
 print(step)  
 break  
 a[1] = pos \* 2  
 a[2] = pos + 1  
 for i in range(1,3):  
 if visited[a[i]] == -1 and a[i] <= maxn:  
 q.append((a[i],step + 1))

**更复杂的 抓住那头牛**

from collections import deque  
MAXN = int(1e5) + 10 #防止越界  
N, K = map(int, input().split())  
visited = [False] \* 2 \* MAXN  
q = deque()  
q.append((N, 0)) # 起点，0步开始  
visited[N] = True  
while q:  
 pos, steps = q.popleft()  
 if pos == K: # 这样判断更好些，不和visited混  
 print(steps)  
 a = [0] \* 4  
 a[1] = pos - 1  
 a[2] = pos + 1  
 a[3] = 2 \* pos # 走到的点  
 for i in range(1,4):  
 if 0 <= a[i] <= MAXN and not visited[a[i]]:  
 q.append((a[i], steps + 1)) # 能走，队列加上走过的节点，以及此时所有步数  
 visited[a[i]] = True #不再重复过这个点

**决战双十一**

def plans(n, price, count, all\_plans, plan):  
 if count == n + 1:  
 all\_plans.append(plan[:])  
 # n（购物计划的长度）、price（包含每个店铺价格信息的字典）、  
 # count（当前正在考虑的店铺索引）、all\_plans（存储所有购物计划的列表）  
 # plan（当前正在构建的购物计划）  
 return  
 for i in price[count].keys(): # 每个count对应一个商品，i值是遍历卖该商品的所有店铺。  
 plan.append(i) # plan加入i店铺  
 plans(n, price, count + 1, all\_plans, plan) # 递归,count+1  
 plan.pop() # 清除现在的计划表，以生成新的计划表  
 return

def buy(n, m, price, coupon):  
 all\_plans = []  
 plans(n, price, 1, all\_plans, []) # 调用plans 算所有计划  
 final\_prices = [] # 最终价格  
 for plan in all\_plans: # 遍历all\_plans计划集，plan存了所有方案数，每个列表有“商品数”个位置，每个位置对应“店铺编号”。  
 tot\_sp = [] # 每个店铺的总价（算这个才能计算如何使用用店铺券）  
 prices = [price[i][plan[i - 1]] for i in range(1, n + 1)] # 商品i在店铺plan[i-1]的价格  
 total = sum(prices) # 这个计划的总价  
 total = total - total // 300 \* 50 # 满多少个300，减多少个50  
  
 for i in range(1, m + 1): # 循环店铺  
 price\_sp = [price[j + 1][plan[j]] for j in range(n) if plan[j] == i]  
 tot\_sp.append(sum(price\_sp))  
  
 store = 0  
 for tot in tot\_sp: # 看每个店铺能用什么店铺券  
 store += 1  
 discount = 0  
 for j in coupon[store]:  
 if tot >= j[0]:  
 discount = max(j[1], discount) # 能享受的最大折扣  
 total -= discount  
 final\_prices.append(total) # 一种计划的总价total计算完成，存入列表  
  
 return min(final\_prices)  
  
n, m = map(int, input().split())  
price = {} # {商品编号：{店编号：商品价格}}  
for i in range(n):  
 price\_i = {}  
 a = input().split()  
 for j in a:  
 price\_i[int(j.split(':')[0])] = int(j.split(':')[1])  
 price[i + 1] = price\_i  
  
coupon = {} # {店铺号：[(满多少,减多少),(满多少,减多少)]}  
for i in range(m):  
 coupon\_i = []  
 a = input().split()  
 for j in a:  
 temp = j.split("-")  
 coupon\_i.append(tuple(map(int, temp)))  
 coupon[i + 1] = coupon\_i  
  
print(buy(n, m, price, coupon))

dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]

**晶矿的个数%………………分两个找的dfs，不需要保护圈，最大连通区域**  
def dfs\_r(i, j):  
 rn = 0  
 if arr[i][j] == "r": **#只有等于r的有意义，事先判断等于r，加了保护圈**  
 rn += 1  
 arr[i][j] = "#"  
 for h in range(4):  
 nx = i + dire[h][0]  
 ny = j + dire[h][1]  
 if arr[nx][ny] == "r":  
 rn += dfs\_r(nx, ny)  
 return rn  
  
def dfs\_b(i,j):  
 bn = 0  
 if arr[i][j] == "b":  
 bn += 1  
 arr[i][j] = "#"  
 for h in range(4):  
 nx = i + dire[h][0]  
 ny = j + dire[h][1]  
 if arr[nx][ny] == "b":  
 bn += dfs\_b(nx, ny)  
 return bn  
  
red,black=[],[]  
n = int(input())  
for \_ in range(n):  
 k = int(input())  
 arr = [["#"] \* (k + 2) for \_ in range(k + 2)]  
 for t in range(1, k+1):  
 arr[t][1:-1] = list(input()) **#保护圈形式**  
  
 for i in range(1, k + 1):  
 for j in range(1, k + 1):  
 if arr[i][j] == "r": #这里判断过，所以不需要保护圈  
 red.append(dfs\_r(i, j))  
  
 if arr[i][j] == "b":  
 black.append(dfs\_b(i,j))  
  
 print(len(red),len(black))  
 red,black = [],[]

**迷宫可行路径数：dfs**

row, col = map(int, input().split())  
dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]  
arr = [["#"] \* (col + 2)]  
for \_ in range(row):  
 arr.append(["#"] + [int(x) for x in input().split()] + ["#"])  
arr.append(["#"] \* (col + 2))  
visited = [[0] \* (col + 2) for \_ in range(row + 2)]  
cnt = 0  
  
def dfs(x, y):  
 global cnt  
 if x == row and y == col:  
 cnt += 1  
 return  
 visited[x][y] = 1 # 走过nx之后不会再回到上一个点  
 for d in dire:  
 nx = x + d[0]  
 ny = y + d[1]  
 if 1 <= nx <= row and 1 <= ny <= col:  
 if arr[nx][ny] == 0 and visited[nx][ny] == 0:  
 visited[nx][ny] = 1  
 dfs(nx, ny)  
 visited[nx][ny] = 0  
 visited[x][y] = 0 # 以x y为根的路都走完了才能释放，这样才能保证每条路都探索完  
dfs(1, 1)  
print(cnt)

**指定步数的迷宫问题：dfs**

row, col, step = map(int, input().split())  
dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]  
arr = [["#"] \* (col + 2)]  
for \_ in range(row):  
 arr.append(["#"] + [int(x) for x in input().split()] + ["#"])  
arr.append(["#"] \* (col + 2))  
canReach = False  
arr[row][col] = "e"  
k = step  
  
def dfs(x, y, step):  
 global canReach  
 for i in range(4):  
 nx = x + dire[i][0]  
 ny = y + dire[i][1]  
 if arr[nx][ny] == "e": # 下一步就到终点  
 if step == 1:  
 canReach = True  
 return  
 if arr[nx][ny] == 0:  
 if step > 0: # 还有剩余的步数，还能继续走  
 arr[x][y] = 1  
 dfs(nx, ny, step - 1)  
 arr[x][y] = 0  
  
dfs(1, 1, step)  
if canReach:  
 print("Yes")  
else:  
 print("No")

**矩阵最大权值路径dfs**

import copy #最大权值路径特有的部分

row, col = map(int, input().split())

dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]

arr = [["#"] \* (col + 2)]

for \_ in range(row):

arr.append(["#"] + [int(x) for x in input().split()] + ["#"])

arr.append(["#"] \* (col + 2))

max\_value = -float('inf')

visited = [[0] \* (col + 2) for \_ in range(row + 2)]

way,ans = [],[]

way.append((1, 1))

# 先加上第一个点，逻辑是因为dfs函数只考虑了第一个点后的

def dfs(x, y, nowValue):

global max\_value,ans

if x == row and y == col:

if nowValue > max\_value:

max\_value = nowValue # 不要忘记随时更新最大值

ans = copy.deepcopy(way) # 深拷贝

return

visited[x][y] = 1

for d in dire:

nx = x + d[0]

ny = y + d[1]

if 1 <= nx <= row and 1 <= ny <= col:

if visited[nx][ny] == 0:

visited[nx][ny] = 1

way.append((nx, ny))

nextValue = arr[nx][ny] + nowValue

dfs(nx,ny,nextValue)

way.pop() # 和visited.pop()是一个道理

visited[nx][ny] = 0

visited[x][y] = 0

dfs(1,1,arr[1][1]) #第一个点的值

for a in ans: # ans存的是最短路径的每一步

print(a[0],a[1])

**NBA门票dfs**

from functools import lru\_cache  
@lru\_cache(maxsize=None)  
def dfs(money, rest):

# 枚举花完钱的所有可能的票数组合，每次举一个面值的票数。结束标志是枚举到小面值也用完。  
 l = len(rest)  
 par\_value = ticket[l - 1] #当前最大面值  
 if l == 1:  
 if money % par\_value == 0 and money <= par\_value \* rest[0]:

# 前一个条件，可以凑出来，后一个条件，能花完。  
 # 前一个条件排除票没卖完钱花不下去，后一个条件排除钱没花完票卖完  
 return money // par\_value  
 else:  
 return float('inf')  
 new\_rest = tuple(rest[i] for i in range(l - 1)) #元组删除tuple  
 limit = min(money // par\_value, rest[l - 1]) # 该面值票数可能多于在该面值下花完money所需的票数，防止money减到小于0  
 return min(dfs(money - par\_value \* j, new\_rest) + j for j in range(limit, -1, -1)) #也有可能不买这个面值，range最小到0  
  
tot = int(input())  
ticket = [1,2,5,10,20,50,100]  
num = tuple(map(int, input().split()))  
if tot % 50 != 0: # 除一下 简化  
 print('Fail')  
else:  
 tot = tot // 50  
 res = dfs(tot, num)  
 if res == float('inf'):  
 print('Fail')  
 else:  
 print(res)

**砝码称重 经典dfs**

num=list(map(int,input().split()))  
wei=[1,2,3,5,10,20]  
w=set() # 去掉重复种类  
def dfs(index,cur\_w): # 通过index保证每次都取下一个砝码  
 if index == 6:  
 if cur\_w!=0:  
 w.add(cur\_w)  
 return  
 for i in range(num[index]+1):  
 # 先取0个1g砝码，再去看2g的，再3g一直到取完6种  
 # 回到第一个砝码，变为取1个砝码。  
 # 不能一个都不取，但是可以不取其中某种，所以for循环从0开始  
 dfs(index+1,cur\_w + i \* wei[index])  
dfs(0,0)  
print('Total={}'.format(len(w)))

**全排列：dfs 肯定是要回溯的**

def dfs(index, all\_perm, perm):  
 if index == n :  
 all\_perm.append(perm[:])  
 return  
 for a in num:  
 if a not in visited:  
 visited.add(a)  
 perm.append(a)  
 dfs(index + 1, all\_perm, perm)  
 visited.remove(a)  
 perm.pop()  
 return  
num = input()  
n = len(num)  
all\_perm = []  
visited = set()  
dfs(0, all\_perm, [])  
for a in all\_perm:  
 print("".join(a))

**组合乘积：dfs**

n = int(input())  
s = list(map(int, input().split()))  
num = len(s)  
visited = set()  
def dfs(idx, p):  
 if p == n:  
 return True  
 for i in range(idx,num):  
 if p \* s[i] <= n:  
 if dfs(i + 1, p \* s[i]): #保证下一个选的是他后边  
 return True  
 return False  
if dfs(0,1):  
 print("YES")  
else:  
 print("NO")

**组合乘积 dfs+lru**

**from functools import lru\_cache**

n = int(input())  
s = list(map(int,input().split()))  
s.sort()  
num = len(s)  
visited = set()  
flag = 0  
@lru\_cache(maxsize=None)  
def dfs(idx, p):  
 global flag  
 if p == n:  
 flag = 1  
 return  
 for a in s:  
 if a \* p > n: # 剪枝，但效果不好  
 break  
 if a not in visited:  
 visited.add(a)  
 np = p \* a  
 dfs(idx + 1, np)  
 visited.remove(a)  
  
dfs(0,1)  
print("YES" if flag else "NO")

**最大连通区域面积：dfs 和晶矿一样**

def dfs(j, k):  
 res = 0  
 direction = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (-1, 1), (1, -1), (-1, -1)]  
 if arr[j][k] == "W": 进入就判断，不要等到nx和ny  
 res += 1  
 arr[j][k] = "."  
 for i in range(len(direction)):  
 res += dfs(j+direction[i][0],k+direction[i][1])  
 return res  
  
T = int(input())  
for \_ in range(T):  
 x, y = map(int, input().split())  
 arr=[["."]\*(y+2) for \_ in range(x+2)]  
 for i in range(1,x+1):  
 arr[i][1:-1]=input()  
 ans = 0  
 for j in range(1,x+1):  
 for k in range(1,y+1):  
 if arr[j][k] == "W":  
 ans=max(ans,dfs(j, k))  
 print(ans)

**最大连通区域面积：bfs 和晶矿一样**

from collections import deque  
T = int(input())  
for \_ in range(T):  
 row, col = map(int, input().split())  
 dire = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (-1, 1), (1, -1), (-1, -1)]  
 arr = [["#"] \* (col + 2)]  
 for \_ in range(row):  
 arr.append(["#"] + [x for x in input()] + ["#"])  
 arr.append(["#"] \* (col + 2))  
 queue = deque()  
 ans = 0  
 def bfs(x,y,cnt=0):  
 queue.append((x, y))  
 while queue:  
 x, y = queue.popleft()  
 if arr[x][y] == "W":  
 cnt += 1  
 arr[x][y] = "." #一进去就判断，否则会忽略只有1个的情况，# 把这一片全标记为"." 判过即标记，不走回头路  
 for d in dire:  
 nx = x + d[0]  
 ny = y + d[1]  
 if 1 <= nx <= row and 1 <= ny <= col:  
 if arr[nx][ny] == "W":   
 queue.append((nx, ny))  
 return cnt  
  
 for x in range(1, row + 1):  
 for y in range(1, col + 1):  
 if arr[x][y] == "W":  
 ans = max(bfs(x,y,0),ans)  
 print(ans)  
 ans = 0

**矩阵中的块：bfs 最基础最标准的**

from collections import deque  
row, col = map(int, input().split())  
dire = [[0, 1], [0, -1], [1, 0], [-1, 0]]  
arr = [["#"] \* (col + 2)]  
for \_ in range(row):  
 arr.append(["#"] + [int(x) for x in input().split()] + ["#"])  
arr.append(["#"] \* (col + 2))  
queue = deque()  
cnt = 0  
def bfs(x,y):  
 queue.append((x, y))  
 while queue:  
 x, y = queue.popleft() # 弹出第一个  
 for d in dire:  
 nx = x + d[0]  
 ny = y + d[1]  
 if 1 <= nx <= row and 1 <= ny <= col:  
 if arr[nx][ny] == 1: # 把这一片全标记为0  
 arr[nx][ny] = 0  
 queue.append((nx, ny)) # 下一次走这个  
  
for x in range(1, row + 1):  
 for y in range(1, col + 1):  
 if arr[x][y] == 1:  
 bfs(x,y)  
 cnt += 1  
print(cnt)

**单词序列bfs 看指定单词能不能通过已有的单词变到新的，zip函数**

**from collections import deque  
  
def diff(a, b):  
 count\_diff = sum(1 for j, k in zip(a, b) if j != k) # zip函数  
 if count\_diff == 1:  
 return 1  
 return 0  
  
a,b = input().split()  
words = list(input().split())  
queue = deque([(a,2)])  
visited = [0] \* 31 #辅助空间，如果visit过words[i]赋值为1  
  
while queue:  
 word, step = queue.popleft()  
 if diff(word,b): # 字符和目标只差1个！  
 print(step)  
 exit()  
  
 for i in range(len(words)): # 每个都要扫哦！我不知道这次是words[i]和哪个word比哦！  
 if visited[i]==0 and diff(word, words[i]): # 没有辅助空间就会反复扫那个单词！  
 visited[i] = 1  
 queue.append((words[i],step + 1))  
  
print("0")**

**Bfs小游戏：heapq优化时间复杂度**

import heapq  
dire = [[0, 1], [1, 0], [0, -1], [-1, 0]]  
cnt\_tot = 0  
while True: # 以下所有内容都套在大while True里  
 width, height = map(int, input().split())  
 cnt = 1  
 if width == 0 and height == 0:  
 break # 每组数据结束标志  
 cnt\_tot += 1 # 数据组数+1  
 print('Board #{}:'.format(cnt\_tot)) # 按照要求格式输出  
 arr = [[" "] \* (width + 2)]  
 for \_ in range(height):  
 arr.append([" "] + [x for x in input()] + [" "])  
 arr.append([" "] \* (width + 2)) # 输入加保护圈，注意这行  
 while True: # 输入每对点  
 x1, y1, x2, y2 = map(int, input().split())  
 if x1 == 0 and x2 == 0 and y1 == 0 and y2 == 0:  
 break  
 queue = []  
  
 flag = 0 # 标记能否找到  
 visited = set()  
 arr[y2][x2] = " " # 能走到  
 heapq.heappush(queue, (0, y1, x1, -1)) # 这两行注意转置  
 visited.add((-1, y1, x1)) # 初始方向标记为不可能的值，这样进去之后第一个方向无论是什么都会算一步  
 while queue:bfs核心部分  
 step, x, y, d = heapq.heappop(queue)  
 if x == y2 and y == x2:  
 flag = 1  
 break  
 for idx in range(4): # idx = 0：下 1：右 2：上 3：左  
 nx = x + dire[idx][0]  
 ny = y + dire[idx][1]  
 if 0 <= ny <= width + 1 and 0 <= nx <= height + 1 and (idx, nx, ny) not in visited: # 保证不越界，没重复走过  
 if arr[nx][ny] != "X":  
 visited.add((idx, nx, ny))  
 if idx == d:  
 d\_cnt = 0  
 else:  
 d\_cnt = 1  
 heapq.heappush(queue, (step + d\_cnt, nx, ny, idx))  
  
 if flag:  
 print('Pair {}: {} segments.'.format(cnt, step))  
 else:  
 print('Pair {}: impossible.'.format(cnt))  
 cnt += 1  
 arr[y2][x2] = "X" # 能走到  
 print() # 每组数据输出后再加一个空行

**有界的深度优先搜索：# 特殊的建图方式**

**from functools import lru\_cache  
node, line, d= map(int,input().split())  
dic = {}  
for i in range(line):  
 a, b = map(int,input().split())  
 if a not in dic:  
 dic[a] = [b]  
 else:  
 dic[a].append(b) # 字典赋值  
 if b not in dic:  
 dic[b] = [a]  
 else:  
 dic[b].append(a)  
  
for i in dic:  
 dic[i].sort() # 按照从大到小走  
  
start = int(input())  
# def深度优先搜索  
route = []  
visited = set()  
@lru\_cache(maxsize=None)  
def dfs(cur, depth):  
 if depth > d or cur in visited:  
 return  
 # 设定边界**

**visited.add(cur) # 走过一次不能再走，否则走过的节点会出现多次，对照一下鸣人佐助  
 route.append(cur)  
  
 if cur not in dic: # 走到头  
 return  
  
 for next in dic[cur]: # 旧的字典值变成新的字典键，无脑遍历，无需加减  
 dfs(next, depth + 1)  
  
dfs(start, 0)  
print(\*route)**

**走山路，找走山路消耗体力最小**

import heapq

row, col, p = map(int, input().split())

dire = [[0, -1], [0, 1], [1, 0], [-1, 0]]

arr = [["#"] \* (col + 2)]for \_ in range(row):

arr.append(["#"] + [x for x in input().split()] + ["#"])

arr.append(["#"] \* (col + 2))

visited = [[0] \* (col + 2) for \_ in range(row + 2)]

min\_value = float('inf') #标记最小值初值为无限大

for \_ in range(p):

flag = 0

sr, sc, er, ec = map(int, input().split())

sr += 1 # 处理坐标

sc += 1

er += 1

ec += 1

if arr[sr][sc] == "#" or arr[er][ec] == "#":

print("NO")

continue

pq = []

heapq.heappush(pq, (0, sr, sc))

while pq:

nowValue, x, y = heapq.heappop(pq)

if nowValue >= min\_value: # 剪枝，不可能

continue

visited[x][y] = 1 # 标记已经走过

if x == er and y == ec:

if nowValue < min\_value:

min\_value = nowValue

flag = 1

for d in dire:

nx, ny = x + d[0], y + d[1] # 可以走

if 1 <= nx <= row and 1 <= ny <= col and not visited[nx][ny] and arr[nx][ny] != "#":

nextValue = nowValue + abs(int(arr[nx][ny]) - int(arr[x][y])) # 更新值

heapq.heappush(pq, (nextValue, nx, ny))

if flag == 0:

print("NO")

else:

print(min\_value)

visited = [[0] \* (col + 2) for \_ in range(row + 2)]

min\_value = float('inf')

**踩方格：每次三个方向，多少种移动方法dfs**

def dfs(i, j, n, arr):  
 if n == 0 and arr[i][j] != 1:

# 走完了，一定要注意n==0不算完，还要最后一个点没走过  
 return 1  
 if arr[i][j] == 0:  
 arr[i][j] = 1 # 标记走过  
 res = (dfs(i - 1, j, n - 1, arr) +  
 dfs(i, j - 1, n - 1, arr) +  
 dfs(i, j + 1, n - 1, arr))  
 arr[i][j] = 0 # 一条路走完了，下一条能走上一条走过的路  
 else:  
 return 0  
 return res  
  
n = int(input())  
arr = [[0] \* 51 for \_ in range(51)]  
i = j = 25  
print(dfs(i, j, n, arr))

ans = []

**八皇后dfs**def dfs(A, cur\_row=0):  
 if cur\_row == len(A):  
 ans.append(''.join(str(i+1) for i in A))  
 return  
 for col in range(len(A)): # 一列一列去找  
 if all(A[row] != col and abs(A[row] - col) != cur\_row - row for row in range(cur\_row)): # 在目前行之前的所有行都要扫一遍，如果之前行的某列放了皇后，目前行就不能放  
 # A[row]代表了cur\_row行前放皇后的列  
 A[cur\_row] = col # 第row行应该放在第col列  
 dfs(A, cur\_row + 1)  
  
dfs([None] \* 8)  
n = int(input())  
for \_ in range(n):  
 k = int(input())  
 print(ans[k-1])

**算24，四个数加减乘除能否算出来24**

import sys  
from functools import lru\_cache  
sys.setrecursionlimit(1000000000)  
@lru\_cache(maxsize=None) # 速度有极大提升  
def dfs(num):  
 if len(num) == 1:  
 return abs(num[0] - 24) <= 1e-7  
  
 for i in range(len(num)):  
 for j in range(i+1, len(num)): # 保证i和j不同  
 a = num[i]  
 b = num[j] # 要计算必须取两个值  
 remaining = []  
  
 for k in range(len(num)):  
 if k != i and k != j:  
 remaining.append(num[k]) # 除了i j编号以外，列表剩下的数字存到remaining  
  
 if dfs(tuple(remaining + [a + b])) or dfs(tuple(remaining + [a \* b])): # 存tuple可以用lrucache  
 return True  
 if a > b and dfs(tuple(remaining + [a - b])):  
 return True  
 if b > a and dfs(tuple(remaining + [b - a])):  
 return True  
  
 if b != 0 and dfs(tuple(remaining + [a / b])):  
 return True  
 if a != 0 and dfs(tuple(remaining + [b / a])):  
 # a和b被计算了，所以a和b计算得到的值可以看作一个新的数进入递归  
 return True  
 return False  
  
while True:  
 num = list(map(int, input().split()))  
 if num == [0, 0, 0, 0]:  
 break  
 if dfs(tuple(num)):  
 print("YES")  
 else:  
 print("NO")

**约瑟夫\*\*\*\*\*\*\*\*\*\*\*\*\*模板**

while True:

n, p, m = map(int, input().split())

if {n, p, m} == {0}:

break

monkeys = [i for i in range(1, n+1)]

index = p - 1

# 将起始位置转换为索引，如果从第一个开始，索引为0

while len(monkeys) != 1:

index = (index + m - 1) % len(monkeys)

print(monkeys.pop(index),end=",")

print(monkeys[0])

**深拷贝和浅拷贝：\*\*\*\*\*\*\*\***

import copy  
a=[[1, 2, 3], 'abc', [1, 3], 4]  
b=a # [[1, 2, 3, 4], 'def', [1, 3], 4, 5]  
c=copy.copy(a) # [[1, 2, 3, 4], 'abc', [1, 3], 4]  
d=copy.deepcopy(a) # [[1, 2, 3], 'abc', [1, 3], 4]  
e = a[:] # [[1, 2, 3, 4], 'abc', [1, 3], 4]

**Sort不影响price\_2 = price[:]的副本**

**计算中位数：**def getmid(l,n):  
 l.sort()  
 if n % 2 == 0:  
 mid = (l[n//2] + l[n//2-1])/2  
 else:  
 mid = l[(n-1)//2]  
 return mid

**麦森数**

import math

p = int(input())

length = int(p \* math.log10(2) + 1)

print(length)

num = pow(2, p, 10\*\*500) - 1

strnum = str(num).zfill(500)[-500:] # 不足填满0

for i in range(0,500,50):

print(strnum[i:i+50])

**直觉类贪心，和幸福的寒假生活**

n=int(input())

l=[]

for \_ in range(n): #存进去所有木头坐标

l.append(list(map(int,input().split())))

s=2

if n>1:

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

x,h=l[i] # 坐标减高度比上一个树的坐标大，

if x-h>l[i-1][0]:

s+=1

elif x+h<l[i+1][0]:

s+=1

l[i][0]+=h

print(s)

else:

print("1")

# 矩阵

**螺旋矩阵\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

n = int(input())

s = [["#"] \* (n + 2)] #第一行

matrix = s + [["#"] + [0] \* n + ["#"] for \_ in range(n)] + s

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

matrix[i][1:-1] = list(input().split())

direction = [[0, 1], [1, 0], [0, -1], [-1, 0]] #方向右下左上

r = 1

c = 1 # 注意保护圈

N = 0

dr, dc = direction[0]

ans = ''

for j in range(1, n \* n + 1):

ans += matrix[r][c]

matrix[r][c] = "#" # 走过了，变成墙

if matrix[r + dr][c + dc] == "#": # 遇到墙，换方向

N += 1

dr, dc = direction[N % 4]

r += dr

c += dc

print(ans)

**洋葱，要大幅度修改模板  
for j in range(1, n \* n + 1):  
 ans += matrix[r][c]  
 matrix[r][c] = "#"  
 if matrix[r + dr][c + dc] == "#":  
 N += 1  
 dr, dc = direction[N % 4] #最后一个点N=1就结束了，不会进入max循环  
 if N % 4 == 0:  
 max\_r = max(max\_r,ans)  
 ans = 0  
 r += dr  
 c += dc  
print(max(max\_r,ans))**

**垃圾炸弹：（先处理好每个格子最大能清除的垃圾数）**

d = int(input())  
n = int(input())  
board = []  
board = [[0 for \_ in range(1025)] for \_ in range(1025)]  
for i in range(n):  
 x, y, k = map(int, input().split())  
 for i in range(max(x - d, 0), min(x + d + 1, 1025)):  
 for j in range(max(y - d, 0), min(y + d + 1, 1025)):  
 board[i][j] += k  
  
res = max\_point = 0  
for i in range(0, 1025):  
 for j in range(0, 1025):  
 if board[i][j] > max\_point:  
 max\_point = board[i][j]  
 res = 1   
 elif board[i][j] == max\_point:  
 res += 1  
print(res, max\_point)

**卷积关键是矩阵乘法的顺序：**

**m,n,p,q=map(int,input().split())  
m\_1 = [[0 for \_ in range(n)] for \_ in range(m)]  
m\_2 = [[0 for \_ in range(q)] for \_ in range(p)]  
result = [[0 for \_ in range(n+1-q)] for \_ in range(m+1-p)]  
  
for i in range(m):  
 m\_1[i]=list(map(int,input().split()))  
for i in range(p):  
 m\_2[i]=list(map(int,input().split()))  
  
for i in range(m+1-p):  
 for j in range(n+1-q):  
 for d in range(p):  
 for c in range(q):  
 result[i][j] += m\_2[d][c]\*m\_1[i+d][j+c]  
  
for i in range(m+1-p):  
 print(' '.join(map(str,result[i])))**

**垃圾炸弹：（避免越界）**

A,B,K=map(int,input().split())

matrix=[[1 for \_ in range(B)] for \_ in range(A)]

for i in range(K):

R,S,P,T=map(int,input().split()) #(R,S),边长,有没有砸到

for i in range(A):

for j in range(B): # 遍历，能处理到这个区域内所有点

if T==1:

if (i >= max(0, R - 1 - P // 2) and i <= min(A - 1, R - 1 + P // 2) and j >= max(0, S - 1 - P // 2) and j <= min(B - 1, S - 1 + P // 2)):

if matrix[i][j]==1: # 如果之前判断可能存在

matrix[i][j]=1 # 现在因为是1所以仍然存在

else:

matrix[i][j]=0

else:

if (i >= max(0, R - 1 - P // 2) and i <= min(A - 1, R - 1 + P // 2) and j >= max(0,S - 1 - P // 2) and j <= min(B - 1, S - 1 + P // 2)):

matrix[i][j]=0

print(sum(i.count(1) for i in matrix))

**矩阵乘法：**

while True:  
 try:  
 n,m=map(int,input().split())  
 m1=[[0 for \_ in range(m)] for \_ in range(n)]  
 for i in range(n):  
 m1[i]=list(map(int,input().split()))  
 a,b=map(int,input().split())  
 m2=[[0 for \_ in range(b)] for \_ in range(a)]  
 for i in range(a):  
 m2[i]=list(map(int,input().split()))  
 x,y=map(int,input().split())  
 m3=[[0 for \_ in range(y)] for \_ in range(x)]  
 for i in range(x):  
 m3[i]=list(map(int,input().split()))  
#前两个限制能乘，后两个限制能加  
 if m==a and n==x and b==y:  
 res=[[0 for \_ in range(b)] for \_ in range(n)]  
 for i in range(n):  
 for j in range(b):  
 for k in range(m):  
 res[i][j]+=m1[i][k]\*m2[k][j] #矩阵乘法关键几行  
 res[i][j]+=m3[i][j]  
 print(res[i][j],end=" ")  
 print()  
  
 else:  
 print("Error!")  
 except EOFError:  
 break

**岛屿周长**

n,m=map(int,input().split())

matrix=[[0 for \_ in range(m+2)] for \_ in range(n+2)] # 保护圈

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

matrix[i][1:-1]=list(map(int,input().split())) #切片赋值

N=0

direction=[[0,-1],[0,1],[1,0],[-1,0]]

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

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

if matrix[i][j]: #是岛屿

N+=4

for k in direction:

di,dj=k

if matrix[i+di][j+dj]: #有连接的岛屿

N-=1 #减去一条边

print(N)

**不可形成2\*2方格**

n,m,k=map(int,input().split())  
matrix=[[0 for i in range(m+3)] for j in range(n+3)]  
  
for i in range(k):  
 a,b=map(int,input().split())  
 matrix[a][b]=1  
 if ((matrix[a-1][b-1]==1 & matrix[a][b-1]==1 & matrix[a-1][b]==1) # 左上，上，左  
 or (matrix[a][b-1]==1 & matrix[a+1][b-1]==1 & matrix[a+1][b]==1) # 左下，下，左  
 or (matrix[a][b+1]==1 & matrix[a+1][b+1]==1 & matrix[a+1][b]==1) # 右下，下，右  
 or (matrix[a-1][b]==1 & matrix[a-1][b+1]==1 & matrix[a][b+1]==1)): # 右上，上，右  
 print(i+1)  
 exit()  
print('0')

# 横坐标减→向上，纵坐标减→向左

# 获取矩阵维度  
n = int(input("请输入矩阵维度："))  
# 定义矩阵  
matrix = [[0 for \_ in range(n)] for \_ in range(n)]  
for i in range(n):  
 matrix[i] = list(map(int, input().split()))  
def determinant(matrix):  
 # 检查矩阵维度是否正确  
 if len(matrix) != len(matrix[0]):  
 raise ValueError("输入矩阵必须是一个方阵！")  
 # 如果矩阵是 2x2 的，直接计算行列式  
 if len(matrix) == 2:  
 det = matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]  
 # 如果矩阵不是 2x2 的，应用 Laplace 展开法计算行列式  
 else:  
 det = 0  
 for j in range(len(matrix)): #按照第一行展开！  
 det += (-1) \*\* j \* matrix[0][j] \* determinant(minor(matrix, 0, j))  
 return det  
# 计算矩阵的子（去掉第i行和第j列）矩阵  
def minor(matrix, i, j):  
 minor\_matrix = [row[:j] + row[j + 1:] for row in (matrix[:i] + matrix[i + 1:])]  
 return minor\_matrix

熄灯问题

from itertools import product  
import copy  
  
rmap = {0:1, 1:0} # 0转1，1转0的写法  
martix = [[0] \* 8]  
for \_ in range(5):  
 martix.append([0] + list(map(int, input().split())) + [0])  
martix.append([0] \* 8)  
for cases in product([0, 1], repeat=6):  
 martixcopy = copy.deepcopy(martix) #建立一个副本，必须使用deepcopy  
 ans = [list(cases)]  
 for i in range(1, 6):  
 for j in range(1, 7):  
 if ans[i - 1][j - 1]:  
 martixcopy[i][j] = rmap[martixcopy[i][j]]  
 martixcopy[i][j - 1] = rmap[martixcopy[i][j - 1]]  
 martixcopy[i][j + 1] = rmap[martixcopy[i][j + 1]]  
 martixcopy[i + 1][j] = rmap[martixcopy[i + 1][j]]  
 ans.append(martixcopy[i][1:7])  
 if martixcopy[5][1:7] == [0, 0, 0, 0, 0, 0]:  
 for line in ans[:-1]:  
 print(\*line)

# 日期转换（语法）

**日历转换：**日、月从1开始计数，年从0开始计数。秒数为整数。假设 0:0:0 1.1.2000 等同于特殊日历法的 0:0:0 1.1.0。

关键：从年月日转天数注意月不要多算。从天数转年月日需要注意

a = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31] # 平年  
b = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31] # 闰年  
result = []  
  
def is\_leap\_year(year):  
 if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):  
 return True  
 else:  
 return False  
# 2001年12月18日，12月还没过完  
def convert\_to\_special\_calendar(date):  
 hour, minute, second, day, month, year = map(int, date.split())  
 day += (year - 2000) \* 365 + ((year - 2001) // 4 - (year - 2001) // 100 + (year - 2001) // 400) + 1  
 if is\_leap\_year(year):  
 for i in range(1, month): # 不算本月  
 day += b[i]  
 else:  
 for i in range(1, month):  
 day += a[i]

# 算年数，如果不是1000的倍数皆大欢喜，如果是，注意年从0开始计数  
 year = day // 1000  
 if day % 1000 == 0: # 年从0开始计数，所以如果是1000 2000这样的数字，第1000天是第0年最后一天，year-1=0  
 year -= 1  
 month = 10 # 月份加满  
 elif day % 100 == 0:

# 是100，1100这样的数，是某个月的最后一天，刚好一个月  
 month = (day % 1000) // 100 # month从1开始算

# 不是100的倍数，比如99，天数算不出来  
 else:  
 month = (day % 1000) // 100 + 1  
 day = (day - 1) % 100 + 1  
 # 日从1开始计数的标准算法，原来总天数-1与新的一年天数取模+1  
  
 # 秒数是在一天内完成的  
 second += hour \* 3600 + minute \* 60  
 second = second \* 125 // 108 # 将旧的秒数转化为新的秒数（原来一秒不等于现在一秒）  
 hour = second // 10000  
 minute = (second % 10000) // 100  
 second = second % 100 # 最终秒数  
 result.append("{}:{}:{} {}.{}.{}".format(hour, minute, second, day, month, year))  
 return result  
  
# 读取输入的测试样例数目  
n = int(input())  
# 代替中间符号的细节  
for i in range(n):  
 date = input().replace(":", " ").replace(".", " ")  
 special\_date = convert\_to\_special\_calendar(date)  
 print(result[i])

**阿尔法星人翻译官：**

name = ['negative', 'zero', 'one', 'two', 'three', 'four', 'five', 'six',  
 'seven', 'eight', 'nine', 'ten', 'eleven', 'twelve', 'thirteen',  
 'fourteen', 'fifteen', 'sixteen', 'seventeen', 'eighteen', 'nineteen', 'twenty',  
 'thirty', 'forty', 'fifty', 'sixty', 'seventy', 'eighty', 'ninety',  
 'hundred', 'thousand', 'million']  
hun = ['hundred', 'thousand', 'million']  
num = ['-', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30  
 , 40, 50, 60, 70, 80, 90, 100, 1000, 1000000]  
  
n = list(input().split())  
ans = []  
res = 0  
if n[0] == "negative":  
 ans.append(num[0])  
 n.pop(0)  
i = 0 # 处理正负号  
res = 0  
tmp = 0  
for i in n:  
 if i in ("thousand", "million"): # 结算  
 res += num[name.index(i)] \* tmp  
 tmp = 0  
 continue # 关键在这里啊  
 if i == "hundred": # 乘法  
 tmp \*= num[name.index(i)] # 关键在这里啊  
 else: # 加法  
 tmp += num[name.index(i)]  
ans.append(str(res + tmp))  
print("".join(ans))

**回文日期**

n = input()  
a = [0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31] # 闰年  
ans=[]  
def genpal(day,mon):  
 l\_half = str(mon \* 100 + day)  
 if len(l\_half)<4:  
 l\_half ='0'+l\_half # 长度不够要加上0  
 f\_half = l\_half[::-1]  
 total = f\_half + l\_half # 生成回文日期  
 return total  
  
for mon in range(1, 13):  
 for day in range(1, a[mon] + 1):  
 palindrome = genpal(day,mon)  
 if '10000101' <= palindrome <= n:  
 ans.append(palindrome)  
ans.sort()  
print(" ".join(ans))

# 不需要考虑2月29天生成的回文年是不是闰年

**maya日历**

mtzolkin = ["imix", "ik", "akbal", "kan", "chicchan", "cimi", "manik", "lamat", "muluk", "ok", "chuen", "eb", "ben", "ix", "mem", "cib", "caban","eznab", "canac", "ahau"]  
mhaab = ["pop", "no", "zip", "zotz", "tzec", "xul", "yoxkin", "mol", "chen", "yax", "zac", "ceh", "mac", "kankin", "muan", "pax", "koyab","cumhu", "uayet"]  
def getMonthNum(mon):  
 return mhaab.index(mon) # 输入字符月份，获取月份的序号0-19   
n = int(input()); print(n)  
for \_ in range(n):  
 dh, mh, yh = input().split(" ") # 解读：年从0开始，第260天仍然是第0年。  
 dh = dh.replace(".",'') # mtzolkin月编号是0-19，而且20一循环。  
 dh,yh = int(dh),int(yh) # 所以日除20取余就得到对应月的字符  
 m = getMonthNum(mh) # 日从1开始，而且13一循环，总天数减一方便计算  
 td = yh \* 365 + m \* 20 + dh # 总天数减1  
 print(td % 13 + 1, mtzolkin[td % 20], td // 260)

**装箱问题 简化版taxi**

import math  
n=int(input())  
l=list(map(int,input().split()))  
count\_4=0  
  
space\_of\_1=0  
count\_2=0  
count\_1=0  
  
for i in l:  
 if i==4:  
 count\_4+=1 #taxi数  
 if i==3:  
 count\_4+=1  
 space\_of\_1+=1  
 if i==2:  
 count\_2+=1  
 if i==1:  
 count\_1+=1  
  
count\_4 += math.ceil(count\_2\*2/4)  
space\_of\_1+= 4\*math.ceil(count\_2\*2/4)-count\_2\*2  
  
if space\_of\_1>=count\_1:  
 print(count\_4)  
else:  
 count\_4+=math.ceil((count\_1-space\_of\_1)/4)  
 print(count\_4)

# Greedy专题

**完美的爱**

from collections import defaultdict  
n = int(input())  
ls = [int(i) - 520 for i in input().split()]  
pre\_sum = [ls[0]]  
for i in range(1, n):  
 pre\_sum.append(pre\_sum[i - 1] + ls[i]) # 前缀和  
mp = defaultdict(int)  
ans = 0  
for i in range(n):  
 x = pre\_sum[i]  
 if x == 0:  
 ans = max(ans, i + 1) # 前缀和为0说明前i个平均是520  
 if x in mp:  
 ans = max(ans, i - mp[x]) # 前缀和相等的情况和我们推出来的一致  
 else:  
 mp[x] = i  
print(ans \* 520)

**厌恶的数**

case=int(input())

for i in range(case):

num\_in\_a,hate=map(int,input().split())

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

sum,ans=0,-1

for i in range(num\_in\_a):

sum += num[i] # 前缀和

if sum % hate !=0:

ans = max(ans, num\_in\_a - i - 1, i + 1)

#如果全部数字和都不能整除hate，输出的是n

#如果全部数字和都能整除hate，输出了其中一个标记点

print(ans)

**充实的寒假生活，参加活动类问题，按照结束时间排序，先选第一个，之后如果不重叠就选**

n = int(input())  
act = []  
for \_ in range(n):  
 s,e = map(int,input().split())  
 act.append((s,e))  
act = sorted(act, key = lambda x:x[1])  
right = act[0][1]  
cnt = 1  
for i in range(1,n):  
 if act[i][0] > right:  
 cnt += 1  
 right = act[i][1]  
print(cnt)

**世界杯只因：按照右端点比较**

N = int(input())  
a = list(map(int,input().split()))  
intervals = [(max(0,i-a[i]),min(N-1,i+a[i])) for i in range(N)] # i从0开始啊，计算监控覆盖区间  
intervals.sort()  
ans, right, temp, index = 0,0,-1,0  
while index < N and right < N:  
 while index < N and intervals[index][0] <= right: # 左端点小于第一个没被盖住的  
 temp = max(temp,intervals[index][1]) # temp更新为右端点  
 index += 1  
 right = temp + 1 # 每一轮更新第一个没被扫过的点  
 ans += 1  
print(ans)

**雷达安装**

segment.sort() # 按照雷达覆盖左端点排序

right = segment[0][1]

for i in range(1,n):

if segment[i][0] <= right: # 能被同一个雷达覆盖

right = min(segment[i][1], right)

else: # 需要添加一个雷达

right = segment[i][1]

cnt += 1

**# 排队服务：**queue = list(map(int, input().split()))  
queue.sort()  
wait = count = 0  
for i in range(n):  
 if wait <= queue[i]: # 其他人总共的等待时间  
 count += 1  
 wait += queue[i]

**# 去掉一些差值使总体差最小：**n, m = map(int, input().split())  
t = list()  
final = list()  
l = list(map(int, input().split()))  
l.sort(reverse=True)  
for i in range(n - 1):  
 t.append(l[i] - l[i + 1])  
t.sort(reverse=True)  
sum\_s = sum(t[:m - 1]) # 差最大单独拿出来，最多能拿出m-1组，从0开始标记序号到m-2  
print(l[0] - l[-1] - sum\_s)

**Holiday hotel：**  
 for i in range(n):  
 c, d = map(int, input().split())  
 hotel.append((c, d))  
 hotel.sort() # 按照价格排序  
 ans = 0  
 maxd = 100000000  
 for i in range(n):  
 if hotel[i][1] < maxd: # i越往后距离越大，价格小于前边选中的，那么ans+1  
 ans += 1  
 maxd = hotel[i][1]  
 print(ans)

**分发糖果，左扫一遍右扫一遍**

n=int(input())  
rating=list(map(int,input().split()))  
sugar=[1 for \_ in range(len(rating))]  
def check(rating,sugar,i):  
 if rating[i]>rating[i+1] and sugar[i]<=sugar[i+1]:  
 sugar[i]=sugar[i+1]+1  
 if rating[i]<rating[i+1] and sugar[i]>=sugar[i+1]:  
 sugar[i+1]=sugar[i]+1  
 return sugar[i],sugar[i+1]  
for i in range(n-1):  
 sugar[i],sugar[i+1]=check(rating,sugar,i)  
for i in range(n-2,-1,-1):  
 sugar[i], sugar[i + 1] = check(rating, sugar, i)  
print(sum(sugar))

# 一些补丁之字符串：

for i in range(0, min(len(a), len(b))):  **# 字符串前缀和后缀**  
 postfix = a[-i-1:] #[-1:]只有最后一个  
 prefix = b[:i+1] #[:1]只有第一个  
 if prefix == postfix:  
 t = len(prefix)  
 break  
print(len(a) - t)

# 一些补丁之矩阵：

逆时针旋转90°

n = int(input())  
for \_ in range(n):  
 k = int(input())  
 ans = 0  
 a = [input() for \_ in range(k)]  
 for ii in range(k // 2):  
 for j in range(ii, k - ii - 1):  
 new = [a[ii][j], a[k - 1 - j][ii], a[k - 1 - ii][k - 1 - j], a[j][k - 1 - ii]] # 只需要四个位置相同  
 c = max(new)  
 for t in new:  
 ans += ord(c) - ord(t) # 只能从小往大变  
 print(ans)

# Heapq和stack

import heapq # 剪绳子，开销最小  
n = int(input())  
leng = list(map(int,input().split()))  
heapq.heapify(leng) # 变成堆  
res = 0  
while len(leng) > 1:  
 a = heapq.heappop(leng)  
 b = heapq.heappop(leng)  
 res += a + b # 每次加上最小的两段  
 heapq.heappush(leng,a+b)  
print(res)

**滑动窗口最大值**

from heapq import heappop, heappush  
n, k = map(int, input().split())  
nums = list(map(int, input().split()))  
if not nums: # 特判  
 res = []  
elif k == 1:  
 res = nums  
else:  
 heap = [] # 堆默认从小到大  
 for i in range(k):  
 heappush(heap, (-nums[i], i)) # 第一个窗口的值存进去，取负号，保证堆顶到底从大到小  
 res = [-heap[0][0]]  
 for i in range(k, len(nums)): # 一步一步挪  
 heappush(heap, (-nums[i], i)) # 加入新的元素  
 while heap[0][1] <= i - k: # 堆顶元素不在窗口内  
 heappop(heap) # 删掉堆顶  
 res.append(-heap[0][0]) # 每次都加入新窗口的堆顶值取负，即最大值  
print(' '.join(map(str, res)))

**护林员**

def f(n, m, board): # n 行 m 列  
 height = [0 for \_ in range(m + 1)]  
 res = 0  
 for i in range(n): # 遍历行，以哪一层作为底层  
 sk = [-1]  
 for j in range(m + 1):

# 遍历列，看以这一行为底的高度  
 # 计算j位置的高度，如果遇到1则置为0，否则递增  
 height[j] = 0 if j == m or board[i][j] == '1' else height[j] + 1  
# 在栈非空且当前高度小于栈顶高度的情况下，弹出栈顶元素，计算以弹出元素为高度的矩形面积，并更新最大矩形面积 res， # 确保栈非空  
 while len(sk) > 1 and height[j] < height[sk[-1]]: # 当height归零时才能满足  
 res = max(res, (j - sk[-2] - 1) \* height[sk[-1]]) # 底（行向，列标号相减）乘以高（列向）  
 sk.pop()  
 sk.append(j) # 如果不进while循环，就存上列标号，列标号是递增的  
 return res

n, m = map(int, input().split())

board = [[0] \* m for \_ in range(n)]

for i in range(n):

board[i] = list(input().split())

print(f(n, m, board))

**括号匹配**

arr = input()  
stack, correct = [], []  
for i in range(len(arr)):  
 if arr[i] == "(":  
 stack.append(i)  
 correct += " "  
 elif arr[i] == ")":   
 if not stack: #栈内没有左括号，右括号无法匹配  
 correct += "?"  
 else:  
 correct += " "  
 stack.pop() #删掉存进去的左括号标号  
 else:  
 correct += " "  
while stack:  
 correct[stack[-1]] = "$" #有左括号没有被匹配  
 stack.pop()

**是否是素数**

def is\_prime(n):

if n <= 1:

return False

if n <= 3:

return True

if n % 2 == 0 or n % 3 == 0:

return False

i = 5

while i \* i <= n:

if n % i == 0 or n % (i + 2) == 0:

return False

i += 6

return True

**欧拉筛**

def euler\_sieve(n):

is\_prime = [True] \* (n + 1)

primes = []

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

if is\_prime[i]:

primes.append(i)

for p in primes:

if i \* p > n:

break

is\_prime[i \* p] = False

if i % p == 0:

break

return primes

**约瑟夫问题**

while True:

monkey = []

n,m = map(int,input().split())

if {n,m} == {0,0}:

break

monkey = [i for i in range(1, n + 1)]

index = 0

while len(monkey) != 1:

temp = monkey.pop(0)

index += 1

if index == m:

index = 0

continue

monkey.append(temp)

print(monkey[0])

**格式化输出**

number = 123.456789

formatted\_number = f"{number:.2f}" # 保留两位小数

print(formatted\_number) # 输出: 123.46

：

**斐波那契数列**

def fibonacci\_iterative(n):

if n <= 0:

return 0

elif n == 1:

return 1

a, b = 0, 1

for \_ in range(2, n + 1):

a, b = b, a + b

return b

**T-prime**

def euler\_sieve(n):

is\_prime = [True] \* (n + 1)

is\_prime[0] = is\_prime[1] = False

primes = []

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

if is\_prime[i]:

primes.append(i)

for p in primes:

if i \* p > n:

break

is\_prime[i \* p] = False

if i % p == 0:

break

return is\_prime

s = euler\_sieve(1000000)

input()

for i in map(int,input().split()):

sqrt\_i = i\*\*0.5

if sqrt\_i % 1 == 0:

if s[int(sqrt\_i)]:

print('YES')

else:

print('NO')

else:

print('NO')

**输入**

while True: try:

line = input() # 读取一行输入

print(f"你输入的内容是：{line}")#或者主要代码部分 except EOFError:

break # 捕获到 EOFError 时退出循环

import sys# 使用 sys.stdin.read() 读取所有输入

data = sys.stdin.read() # 读取整个输入流

print(data)

**从列表中删除元素**

法一：my\_list = [1, 2, 3, 4, 2, 5] my\_list.remove(2) # 删除第一个 2

法二：my\_list = [1, 2, 3, 4, 5]

removed\_element = my\_list.pop(2) # 删除索引为 2 的元素 (即 3)