Problems in cs101.openjudge.cn

2020 fall, Complied by Hongfei Yan

How to find the problems?
Visit http://cs101.openjudge.cn/, click '练习'.

What is OpenJudge? What kind of a site/resource is it?
OpenJudge is a project joining people interested in and taking part in programming contests. On one hand, OpenJudge is a social network dedicated to programming and programming contests. On the other hand, it is a platform where contests are held regularly, the participant's skills are reflected by their rating and the former contests can be used to prepare.

Basic Programming Exercises

#. title, algorithm, link

2981: 大整数加法

http://cs101.openjudge.cn/practice/2981

求两个不超过200位的非负整数的和。

输入

有两行，每行是一个不超过200位的非负整数，可能有多余的前导0。

输出

一行，即相加后的结果。结果里不能有多余的前导0，即如果结果是342，那么就不能输出为0342。

样例输入

```
22222222222222222222
33333333333333333333
```

样例输出

```
55555555555555555555
```

来源：程序设计实习2007

```
print(int(input()) + int(input()))
```

18161: 矩阵运算

matrices, http://cs101.openjudge.cn/practice/2981

现有三个矩阵A, B, C，要求矩阵运算A·B+C并输出结果

矩阵运算介绍：
矩阵乘法运算必须要前一个矩阵的列数与后一个矩阵的行数相同，
如m行n列的矩阵A与n行p列的矩阵B相乘，可以得到m行p列的矩阵C，
矩阵C的每个元素都由A的对应行中的元素与B的对应列中的元素一一相乘并求和得到，
即C[i][j] = A[i][0]*B[0][j]* + *A[i][1]*B[1][j] + ... +A[i][n-1]*B[n-1][j]

(C[i][j]表示C矩阵中第i行第j列元素)。

矩阵的加法必须在两个矩阵行数列数均相等的情况下进行，
如m行n列的矩阵A与m行n列的矩阵B相乘，可以得到m行n列的矩阵C，
矩阵C的每个元素都由A与B对应位置的元素相加得到，
即C[i][j] = A[i][j] + B[i][j]

输入

输入分为三部分，分别是A,B,C三个矩阵的内容。
每一部分的第一行为两个整数，代表矩阵的行数row和列数col
接下来row行，每行有col个整数，代表该矩阵这一行的每个元素

输出

如果可以完成矩阵计算，输出计算结果，与输入格式类似，不需要输出行数和列数信息。
如果不能完成矩阵计算，输出"Error!"

样例输入

```
Sample Input1:
3 1
0
1
0
1 2
1 1
3 2
3 1
3 1
3 1

Sample Output1:
3 1
4 2
3 1
```

样例输出

```
Sample Input2:
1 1
0
2 1
1
3
1 1
9

Sample Output2:
Error!
```

提示

sample1 计算过程

$$\begin{vmatrix}0\end{vmatrix} \quad \begin{vmatrix}00\end{vmatrix}$$
$$\begin{vmatrix}1\end{vmatrix} \cdot \begin{vmatrix}11\end{vmatrix} = \begin{vmatrix}11\end{vmatrix}$$
$$\begin{vmatrix}0\end{vmatrix} \quad \begin{vmatrix}00\end{vmatrix}$$

$$\begin{vmatrix}00\end{vmatrix} \quad \begin{vmatrix}31\end{vmatrix} \quad \begin{vmatrix}31\end{vmatrix}$$
$$\begin{vmatrix}11\end{vmatrix} + \begin{vmatrix}31\end{vmatrix} = \begin{vmatrix}42\end{vmatrix}$$
$$\begin{vmatrix}00\end{vmatrix} \quad \begin{vmatrix}31\end{vmatrix} \quad \begin{vmatrix}31\end{vmatrix}$$

思路：矩阵运算，如果没有学过可以百度下矩阵乘法（这是线性代数/高等代数的初步）

```
A,B,C = [],[],[]

a,b = map(int, input().split())
for i in range(a):
    A.append(list(map(int, input().split())))

c,d = map(int, input().split())
for i in range(c):
    B.append(list(map(int, input().split())))

e,f = map(int, input().split())
for i in range(e):
    C.append(list(map(int, input().split())))

if b!=c or a!=e or d!=f:
    print("Error!")
else:
    D = [[0 for j in range(f)] for i in range(e)]
    for i in range(e):
        for j in range(f):
            for k in range(b):
                D[i][j] += A[i][k] * B [k][j]
            D[i][j] += C[i][j]

    for i in range(e):
        print(' '.join([str(j) for j in D[i]]))
```

19944: 这一天星期几v0.2

math, http://cs101.openjudge.cn/practice/19944/, cs10119 Final Exam

在日常生活中，计算某一个具体的日期是星期几，往往需要去翻阅日历。请你帮助更快地计算出每个日期在一星期是第几天。

参考：蔡勒公式（Zeller's congruence），是一种计算任何一日属一星期中哪一日的算法，由德国数学家克里斯提安·蔡勒推算出来，可以计算1582年10月15日之后的情况。

$$w = (y + [\frac{y}{4}] + [\frac{c}{4}] - 2c + [\frac{26(m+1)}{10}] + d - 1) \bmod 7$$

公式都是基于公历的置闰规则来考虑。公式中的符号含义如下：

w：星期（计算所得的数值对应的星期：0-Sunday；1-Monday；2-Tuesday；3-Wednesday；4-Thursday；5-Friday；6-Saturday

c：年份前两位数

y：年份后两位数

m：月（m的取值范围为3至14，即在蔡勒公式中，某年的1、2月要看作上一年的13、14月来计算，比如2003年1月1日要看作2002年的13月1日来计算）

d：日

[ ]：称作高斯符号，代表向下取整，即，取不大于原数的最大整数。

mod：同余（这里代表括号里的答案除以7后的余数）

输入

第一行1个整数n，代表输入日期的个数。
接下来n行分别为n个以8位数字表示的日期，如20190101
保证所有输入都在蔡勒公式的计算范围之内。

输出

共n行，每行为该日期对应的weekday名称（Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday）。

思路：就是套公式，但是注意，year会切分两段，但year又有可能做整体进行减法运算，所以，先判断是否运算，再切分。

2020fall-cs101-赵春源

```
Day = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday','Saturday']
T = int(input())
while T > 0:
    x = input()
    year = int(x[0:4])
    m = int(x[4:6])
    if m < 3:
        m += 12
        year -= 1
    y = int(str(year)[2:4])
    c = int(str(year)[0:2])
    d = int(x[6:8])
    #print(y, c, m, d)
    print(Day[(y + y//4 + c//4 - 2*c + (26*(m+1))//10 + d -1 ) % 7])
    T -= 1
```

```
import math

d =
{0:'Sunday',1:'Monday',2:'Tuesday',3:'Wednesday',4:'Thursday',5:'Friday',6:'Satu
rday'}

def what_day(y,c,m,d):
    w = (y + math.floor(y/4)+math.floor(c/4)-2*c+ \
        math.floor(26*(m+1)/10) + d - 1)%7
    return w

n = int(input())
for _ in range(n):
    s = input()
    year = int(s[:4])
    month = int(s[4:6])
    if month==1:
        year -= 1
        month = 13
    if month==2:
        year -= 1
        month = 14

    day = int(s[-2:])

    year_s = str(year)
    wd = what_day(int(year_s[2:]), int(year_s[:2]), month, day)

    print(d[wd])
```

19949: 提取实体v0.2

string, http://cs101.openjudge.cn/practice/19949/, cs10119 Final Exam

一个句子里面一些特殊的单词被称作实体，实体是存在于现实世界中并且可以与其他物体区分开来的物体，如“John has an apple .”这句话中，“John”和“apple”都是实体。

现在我们有很多个人工标注好实体的英文文档，每篇文档有很多个句子，每个句子中，每个实体的每单词都添加了“###”前缀，并且添加了“###”后缀，表明两个“###”之间的部分是实体或者是实体的一部分。例如：

1）两个“###”之间是实体：

“###John### has an ###apple### ”中有两个实体，是John和apple
2）两个“###”之间是实体的一部分，即是，连续的几个被“###”前后包裹的单词被认为是同一个实体：

“###Shelley### ###Berkley### , a Democratic representative”中有一个实体，是Shelley Berkley

“###Dominic### ###J.### ###Baranello### , an enduring power in Democratic Party”中有一个实体，是Dominic J. Baranello

请你帮助统计每篇文档里有多少个实体，暂时不考虑句子间的实体有重复的情况。

输入

第一行为1个整数N，代表文档里的句子数目。
接下来N行，每行代表一个英文句子，每个句子有多少单词是未知的，词与词之间用空格分隔。

输出

1个整数，代表该篇文档里的实体数目。

```
T = int(input())
ans = 0
while T>0:
    T -= 1
    ans += (input().replace('### ###', '')).count('###') // 2
print(ans)
```

```
n = int(input())

cnt = 0
for _ in range(n):
    s = input()
    s = s.replace(r"### ###"," ")
    #print(s)
    while True:
        begin = s.find("###")
        if begin == -1: break

        end = s.find("###", begin+3)

        cnt += 1
        s = s[end+3:]

print(cnt)
```

12560: 生存游戏

matrix, http://cs101.openjudge.cn/practice/12560/, cs10116 final exam

有如下生存游戏的规则：

给定一个n*m(1<=n,m<=100)的数组，每个元素代表一个细胞，其初始状态为活着(1)或死去(0)。

每个细胞会与其相邻的8个邻居（除数组边缘的细胞）进行交互，并遵守如下规则：

任何一个活着的细胞如果只有小于2个活着的邻居，那它就会由于人口稀少死去。

任何一个活着的细胞如果有2个或者3个活着的邻居，就可以继续活下去。

任何一个活着的细胞如果有超过3个活着的邻居，那它就会由于人口拥挤而死去。

任何一个死去的细胞如果有恰好3个活着的邻居，那它就会由于繁殖而重新变成活着的状态。

请写一个函数用来计算所给定初始状态的细胞经过一次更新后的状态是什么。

注意：所有细胞的状态必须同时更新，不能使用更新后的状态作为其他细胞的邻居状态来进行计算。

输入

第一行为n和m，而后n行，每行m个元素，用空格隔开。

输出

n行，每行m个元素，用空格隔开。

2020fall-cs101，张一丹

1）加保护圈；2）查找，满足条件，替换。

```
def check(board, y, x):
    c = board[y-1][x-1]+board[y-1][x]+ board[y-1][x+1]+ \
        board[y][x-1]+board[y][x+1]+ \
        board[y+1][x-1]+board[y+1][x]+board[y+1][x+1]
    if board[y][x] and (c<2 or c>3):
        return 0
    elif board[y][x]==0 and c==3:
        return 1

    return board[y][x]

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

board=[]
board.append( [0 for x in range(m+2)] )
for y in range(n):
    board.append([0] +[int(x) for x in input().split()] + [0])

board.append( [0 for x in range(m+2)] )

# in place solver
bn = [[0]*m for y in range(n)]
for y in range(n):
    for x in range(m):
        bn[y][x] = check(board, y+1, x+1)

for y in range(n):
    print(' '.join([str(x) for x in bn[y] ]))
```

18182: 打怪兽

implementation/sorting/math, http://cs101.openjudge.cn/practice/18182/

Q神无聊的时候经常打怪兽。现在有一只怪兽血量是b，Q神在一些时刻可以选择一些技能打怪兽，每次释放技能都会让怪兽掉血。
现在给出一些技能t_i,x_i，代表这个技能可以在ti时刻使用，并且使得怪兽的血量下降x_i。这个打怪兽游戏有个限制，每一时刻最多可以使用m个技能（一个技能只能用一次）。如果技能使用得当，那么怪兽会在哪一时刻死掉呢？

输入

第一行是数据组数nCases, nCases≤ 100
对于每组数据，第一行是三个整数n,m,b，n代表技能的个数，m代表每一时刻可以使用最多m个技能，b代表怪兽初始的血量。

$1 \le n \le 1000$，$1 \le m \le 1000$，$1 \le b \le 10^9$

接下来n行，每一行一个技能t_i,x_i，$1 \le t_i \le 10^9$，$1 \le x_i \le 10^9$

输出

对于每组数据，输出怪兽在哪一时刻死掉，血量小于等于0就算挂，如果不能杀死怪兽，输出alive

样例输入

```
2
1 1 10
1 5
2 2 10
1 5
1 5
```

样例输出

```
alive
1
```

提示：技能不保证按照时刻顺序输入

来源：cs101-2015 期末机考

2020fall-cs101，杨永祥，思路比较简单，时间做 key，技能为 value，然后创建字典，在遍历过程中修改 value中保存的技能伤害就可。

2020fall-cs101，成泽恺。思路：创建字典，将技能的 t_i 作为键，x_i 作为值存入，x_i 存成列表形式，然后每一个回合按时间从前向后遍历字典的值，按 m 判断这一时刻最多能打掉多少血，situation 初始值设为 alive，一旦血量小于等于 0，赋值为这个回合，最后输出 situation。

```
cases = int(input())
for i in range(cases):
    situation = "alive"
    n,m, b= map(int, input().split())
    a={}
    for i in range(n):
        x,y = map(int,input().split())
        if x not in a:
            a[x] = [y]
        else:
            a[x].append(y)

    c = sorted(a)
    for i in c:
        if m >= len(a[i]):
            b -= sum(a[i])
        else:
            a[i] = sorted(a[i], reverse=True)
```

```
            b -= sum(a[i][:m])
        if b <= 0:
            situation = i
            break

    print(situation)
```

```
nCases = int(input())
while nCases>0:
        nCases -= 1
        n,m,b = [int(i) for i in input().split()]

        skill = []
        for i in range(n):
                t,x = [int(i) for i in input().split()]
                skill.append( (t, x) )

        sort_skill = sorted(skill, key=lambda a: (a[0], -a[1]))

        if sort_skill[0][1]>=b:
                print(sort_skill[0][0])
                continue

        pre_t = sort_skill[0][0]

        first = True
        cnt = 0
        for e in sort_skill:
                if first:
                        b -= e[1]
                        cnt += 1
                        first = False
                        continue

                if e[0]==pre_t:
                        cnt += 1
                else:
                        cnt = 1
                        pre_t = e[0]

                if cnt>m:
                        continue

                if e[1]>=b:
                        print(e[0])
                        break
                else:
                        b -= e[1]
        else:
                print('alive')
```

18211: 军备竞赛

greedy/two pointers, http://cs101.openjudge.cn/practice/18211

鸣人是木叶村的村长，最近在跟敌国进行军备竞赛，他手边有N份武器设计图，每张设计图有制作成本（大于等于零）且最多使用一次，可以选择花钱制作或是以同样的价钱卖给敌国，同时任意时刻敌国的武器不能比我国更多，鸣人的目标是在不负债的前提下武器种类比敌国越多越好。

输入

第一行为起始整数经费p,并且0≤p。且要求任何时刻p不能小于0.
第二行为n个整数，以空格分隔，并且0≤每个整数。代表每张设计图的制作成本，同时也是卖价，最多用一次(无法又制作又卖).

输出

一个整数，代表武器种类最多比敌国多多少.

思路：类似二分双指针left, right，有钱就买（制作），没钱就卖。感谢2020fall-cs101 汤建浩，指正cnt<0情况。

```
p = int(input())
n = [int(x) for x in input().split()]
n.sort()

cnt = 0
left = 0
right = len(n) - 1

while left<=right:
    if n[left]<=p:
        cnt += 1
        p -= n[left]
        left += 1
    else:
        if right==left:
            break

        p += n[right]
        cnt -= 1
        if cnt<0:
            cnt=0
            break

        right -= 1

print(cnt)
```

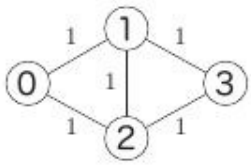
19943: 图的拉普拉斯矩阵

implementation/matrix, http://cs101.openjudge.cn/practice/19943/

在图论中，度数矩阵是一个对角矩阵，其中包含的信息为的每一个顶点的度数，也就是说，每个顶点相邻的边数。邻接矩阵是图的一种常用存储方式。如果一个图一共有编号为0,1,2，...n-1的n个节点，那么邻接矩阵A的大小为n*n，对其中任一元素Aij，如果节点i，j直接有边，那么Aij=1；否则Aij=0。

将度数矩阵与邻接矩阵逐位相减，可以求得图的拉普拉斯矩阵。具体可见下图示意。

$$L := D - A$$



$$D = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad L = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

现给出一个图中的所有边的信息，需要你输出该图的拉普拉斯矩阵。

输入

第一行2个整数，代表该图的顶点数n和边数m。
接下m行，每行为空格分隔的2个整数a和b，代表顶点a和顶点b之间有一条无向边相连，a和b均为大小范围在0到n-1之间的整数。输入保证每条无向边仅出现一次（如1 2和2 1是同一条边，并不会在数据中同时出现）。

输出

共n行，每行为以空格分隔的n个整数，代表该图的拉普拉斯矩阵。

样例输入

```
4 5
2 1
1 3
2 3
0 1
0 2
```

样例输出

```
2 -1 -1 0
-1 3 -1 -1
-1 -1 3 -1
0 -1 -1 2
```

来源：cs101 2019 Final Exam

2020fall-cs101，蓝克轩。思路：因为公式是L=D-A，所以接受输入时，可以一次过在对角线加1(D)，再在相应的元素减1(-A).

2020fall-cs101，郭冠廷。思路：拉普拉斯矩阵不用算两个相减,因为对角线和非对角线肯定不相关联,直接输入到同一个矩阵中即可。

```
n, m = map(int, input().split())
ans = [[0 for i in range(n)] for j in range(n)]
for i in range(m):
    knot1, knot2 = map(int, input().split())
    ans[knot1][knot1] += 1
    ans[knot2][knot2] += 1
    ans[knot1][knot2] -= 1
    ans[knot2][knot1] -= 1
for j in range(n):
    print(' '.join(map(str, ans[j])))
```

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

D = [[0]*n for _ in range(n)]

A = [[0]*n for _ in range(n)]

for _ in range(m):
    n1,n2 = map(int, input().split())
    D[n1][n1] += 1
    D[n2][n2] += 1
    A[n1][n2] = 1
    A[n2][n1] = 1

#print(D)
#print(A)
for r in range(n):
    for c in range(n):
        D[r][c] -= A[r][c]

#print(D)


# print
for row in D:
    print(' '.join(map(str,row)))
```

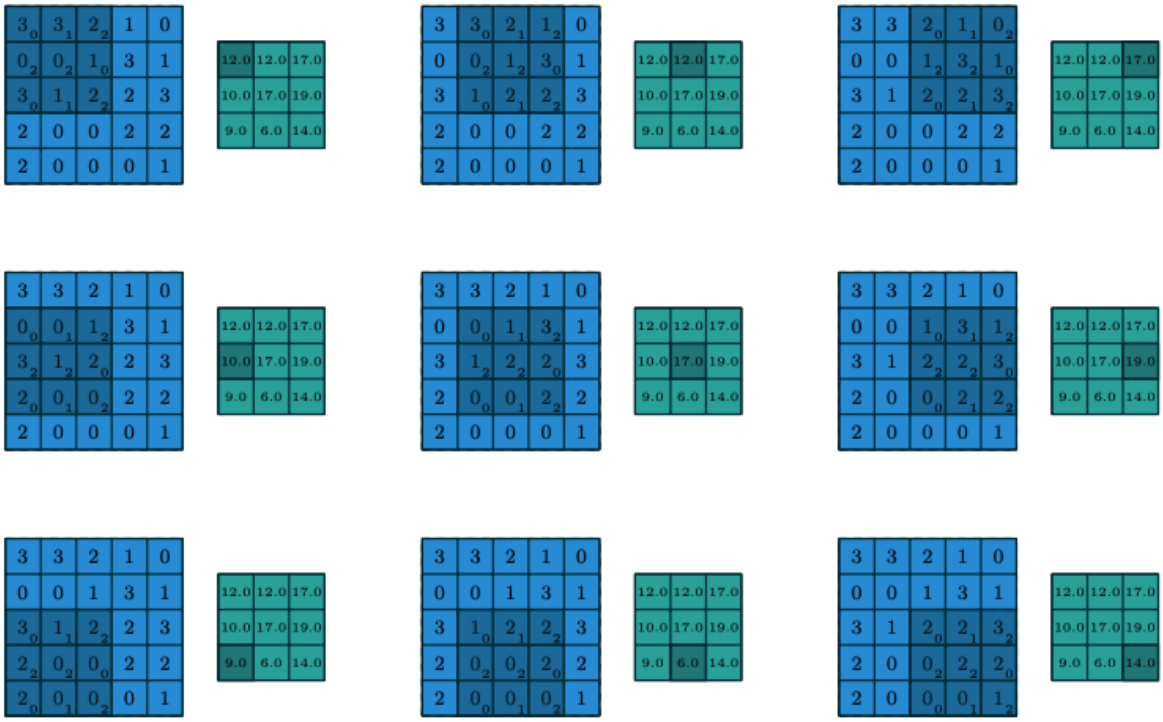
19942: 二维矩阵上的卷积运算v0.2

implementation/matrix, http://cs101.openjudge.cn/practice/19942/

描述

二维矩阵上的卷积，是卷积神经网络中经常需要进行的一种运算。该运算在输入的二维矩阵上滑动不同的卷积核，并在每一个滑动的位置上将卷积核与输入图像对应位置的元素进行相乘并逐个求和的运算。下图很好地说明了运算的结果矩阵的每一个位置是如何计算得来的。本题中在行列方向的滑动均以1为步长进行，且输入输出均为整数。

如对第1行第1列，12 = 3*0+3*1+2*2+0*2+0*2+1*0+3*0+1*1+2*2。



输入

第一行4个整数，分别代表二维矩阵的行数和列数m和n以及卷积核的行数和列数p和q
（1 <= p <= m; 1 <= q <= n）
接下来m行，每行为空格分隔的n个整数，代表二维数组中每行的数据。
接下来p行，每行为空格分隔的q个整数，代表卷积核中每行的数据。

输出

易知，p*q的卷积核在m*n的矩阵上以1为步长滑动，横向一共有m+1-p个位置，纵向一共有n+1-q个位置。
因此输出共m+1-p行，每行为以空格分隔的n+1-q个整数，代表卷积运算后的结果。

样例输入

```
Sample1 Input:
5 5 3 3
3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
0 1 2
2 2 0
0 1 2

Sample1 Output:
12 12 17
10 17 19
9 6 14
```

样例输出

```
Sample2 Input:
5 4 4 4
10 -8 6 9
7 -9 1 0
1 -9 2 -5
-3 6 -1 2
-10 -2 -1 -2
-9 -1 -6 10
3 -2 -5 9
-10 -3 -10 7
3 -2 -7 0

Sample2 Output:
-46
-77
```

来源：cs101-2019 Final Exam

思路：按照输入构造两个矩阵，再用for loop进行卷积运算。

```
m,n,p,q = map(int, input().split())
yuan=[[int(x) for x in input().split()] for _ in range(m)]
juan=[[int(x) for x in input().split()] for _ in range(p)]
answer=[[None]*(n-q+1)  for _ in range(m-p+1)]
def j(x,y):
    s=0
    for i in range(p):
        for j in range(q):
            s += juan[i][j]*yuan[i+x][j+y]
    return s

for a in range(m-p+1):
    for b in  range(n-q+1):
        answer[a][b] = str(j(a,b))

for i in range(m-p+1):
    print(' '.join(answer[i]))
```

2020fall-cs101，顾臻宜。OJ 19942 二维矩阵上的卷积运算.

思路：依题意，个人习惯先将一行的各列相加，再将各行相加。想清楚即可。

```
m,n,p,q = map(int, input().split())
A = [[0 for x in range(n)] for i in range(m)]   #二维矩阵
B = [[0 for x in range(q)] for i in range(p)]   #卷积核
C = [[0 for x in range(n+1-q)] for i in range(m+1-p)]   #二维矩阵上的卷积运算
for i in range(m):
    A[i] = [int(x) for x in input().split()]
for i in range(p):
    B[i] = [int(x) for x in input().split()]

for i in range(m+1-p):
    for j in range(n+1-q):
        for s in range(p):
            for t in range(q):
```

```
                C[i][j] = C[i][j] + A[i+s][j+t] * B[s][t]

for i in range(m + 1 - p):
    print(' '.join(map(str, C[i])))
```

1700: 八皇后问题解输出

dfs, http://cs101.openjudge.cn/practice/1700

在国际象棋棋盘上放置八个皇后，要求每两个皇后之间不能直接吃掉对方。

输入：无输入。

输出：按给定顺序和格式输出所有八皇后问题的解（见Sample Output）。

样例输入

```

```

样例输出

```
No. 1
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
No. 2
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
No. 3
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
...以下省略
```

提示：此题可使用函数递归调用的方法求解。

来源：计算概论05

```
ans = []
```

```
def queen(A, cur=0):                    #考虑放第cur行的皇后
    if cur == len(A):                   #如果已经放了n个皇后，一组新的解产生了
        #ans.append(''.join([str(x+1) for x in A]))
        ans.append(A[:])
        return

    for col in range(len(A)):           #将当前皇后逐一放置在不同的列，每列对应一组解
        for row in range(cur):          #逐一判定，与前面的皇后是否冲突
            if A[row] == col or abs(col - A[row]) == cur - row:
                break
        else:                           #若都不冲突
            A[cur] = col                #放置新皇后，在cur行，col列
            queen(A, cur+1)             #对下一个皇后位置进行递归

queen([None]*8)
for m in range(92):
    print("No.",m+1)
    output = [[0 for i in range(8)] for j in range(8)]
    for x in range(8):
        output[ ans[m][x] ][x] = 1
    for n in range(8):
        print(" ".join(map(str, output[n])))
```

12757:阿尔法星人翻译官

implementation, http://cs101.openjudge.cn/practice/12757

阿尔法星人为了了解地球人，需要将地球上所有的语言转换为他们自己的语言，其中一个小模块是要将地球上英文表达的数字转换为阿尔法星人也理解的阿拉伯数字。 请你为外星人设计这个模块，即给定一个用英文表示的整数，将其转换成用阿拉伯数字表示的整数。这些数的范围从 - 999,999,999到 + 999,999,999。 下列单词是你的程序中将遇到的所有有关数目的英文单词：

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

输入

输入一行，由几个表示数目的英文单词组成(长度不超多200)。注意：负号将由单词negative表示。当数的大小超过千时，并不用完全单词hundred表示。例如1600将被写为"one thousand six hundred", 而不是"sixteen hundred"。

输出

输出一行，表示答案。

样例输入

```
negative seven hundred twenty nine
```

样例输出

```
-729
```

其他参考样例：
six： 6
one million one hundred one：1000101
eight hundred fourteen thousand twenty two：814022

```
tokens = [str(i) for i in input().split()]
dic={"zero":0, "one":1, "two":2, "three":3, "four":4, "five":5, "six":6,
     "seven":7, "eight":8, "nine":9, "ten":10, "eleven":11, "twelve":12,
     "thirteen":13, "fourteen":14, "fifteen":15, "sixteen":16, "seventeen":17,
     "eighteen":18, "nineteen":19, "twenty":20, "thirty":30, "forty":40,
     "fifty":50, "sixty":60, "seventy":70, "eighty":80, "ninety":90,
     "hundred":100, "thousand":1000, "million":1000000}

sign = 1
if tokens[0]=="negative":
    sign = -1
    del tokens[0]

total = 0
tmp = 0
for i in tokens:
    if i in ("thousand", "million"):
        total += tmp*dic[i]
        tmp = 0
        continue
    if i == "hundred":
        tmp *= dic[i]
    else:
        tmp += dic[i]

print( sign * (total + tmp) )
```

18106: 螺旋矩阵

matrix, http://cs101.openjudge.cn/practice/18106

给定一个n(1<=n<=20)，生成一个n*n的二维数组，并用1到n^2对该数组用螺旋顺序进行填充。
如给定n=3时，生成的数组如下：
[
[1, 2, 3],
[8, 9, 4],
[7, 6, 5]
]

输入

一行n

输出

n行，每行n个元素，并用空格将数组元素隔开。

样例输入

```
3
```

样例输出

```
1 2 3
8 9 4
7 6 5
```

来源：cs101-2016 期末机考备选

思路：就是走到四个边，就转90度。

先定义方向，然后一直向前走，如果撞边（如果下一个位置不为0，代表撞边），就转向。

```
n = int(input())
s = [[401]*(n+2)]
mx = s + [[401] + [0]*n + [401] for _ in range(n)] + s

dirL = [[0,1], [1,0], [0,-1], [-1,0]]

row = 1
col = 1
N = 0
drow, dcol = dirL[0]

for j in range(1, n*n+1):
    mx[row][col] = j
    if mx[row+drow][col+dcol]:
        N += 1
        drow, dcol = dirL[N%4]

    row += drow
    col += dcol

for i in range(1, n+1):
    print(' '.join(map(str, mx[i][1:-1])))
```

1810: 校门外的树

implementation, http://cs101.openjudge.cn/practice/1810

某校大门外长度为L的马路上有一排树，每两棵相邻的树之间的间隔都是1米。我们可以把马路看成一个数轴，马路的一端在数轴0的位置，另一端在L的位置；数轴上的每个整数点，即0，1，2，……，L，都种有一棵树。
马路上有一些区域要用来建地铁，这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标都是整数，区域之间可能有重合的部分。现在要把这些区域中的树（包括区域端点处的两棵树）移走。你的任务是计算将这些树都移走后，马路上还有多少棵树。

输入

输入的第一行有两个整数L（1 <= L <= 10000）和 M（1 <= M <= 100），L代表马路的长度，M代表区域的数目，L和M之间用一个空格隔开。接下来的M行每行包含两个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。

输出

输出包括一行，这一行只包含一个整数，表示马路上剩余的树的数目。

样例输入

```
500 3
150 300
100 200
470 471
```

样例输出

```
298
```

来源：noip2005普及组

```
L, m = map(int, input().split())

dp = [1]*(L+1)

for i in range(m):
    s, e = map(int, input().split())
    for j in range(s, e+1):
        dp[j] = 0

print(dp.count(1))
```

1748: 约瑟夫问题

implementation, http://cs101.openjudge.cn/practice/1748

约瑟夫问题：有n只猴子，按顺时针方向围成一圈选大王（编号从1到n），从第1号开始报数，一直数到m，数到m的猴子退出圈外，剩下的猴子再接着从1开始报数。就这样，直到圈内只剩下一只猴子时，这个猴子就是猴王，编程求输入n，m后，输出最后猴王的编号。

输入

每行是用空格分开的两个整数，第一个是 n, 第二个是 m (0 < m,n <=300)。最后一行是：

0 0

输出

对于每行输入数据（最后一行除外），输出数据也是一行，即最后猴王的编号

样例输入

```
6 2
12 4
8 3
0 0
```

样例输出

```
5
1
7
```

思路：因为退出圈的起点会移动m步，所以加个m-1，又因为是个圆环，需要计算mod n。

```
while True:
    n, m = map(int, input().split())
    if n + m == 0:
        break
    mon = []    # monkey
    for i in range(1, n+1):
        mon.append(i)

    t = m - 1
    for j in range(n):
        if len(mon) == 1:
            print(sum(mon))
        else:
            del mon[t]
            n -= 1
            t = (t + m - 1) % n
```

2020fall-cs101，李元锋。

思路：没接触编程前就听说过此题的大名了，这题可以用递归思想。首先先简化问题，考虑当n=5,m=3的情况： (1,2,3,4,5)，第一轮报数后变成 (4,5,1,2)。如果我们可以把 (4,5,1,2)变成(1,2,3,4)，那不就可以重复这个流程直到 1了嘛。这个流程可以用公式(func(n-1,m)+m-1)%n+1给出，直接递归即可。

```
def func(n,m):
    if n == 1:
        return n
    return (func(n-1, m) + m-1) % n + 1

while True:
    n, m = map(int, input().split())
    if n+m == 0:
        break
    print(func(n, m))
```

```
while True:
    n, m = map(int, input().split())
    if n+m == 0:
        break

    dp = [0] + [1]*n
    cur = 1
    while sum(dp)!=1:
        cnt = 0
        while True:
```

```
            if dp[cur] == 1:
                cnt += 1
                if cnt==m:
                    break

                cur += 1
                if cur>n:
                    cur = 1
            else:
                cur += 1
                if cur>n:
                    cur = 1

        dp[cur] = 0
        cur += 1
        if cur>n:
            cur = 1

    else:
        print(dp.index(1))
```

1694: 假币问题

brute force, http://cs101.openjudge.cn/practice/1694

赛利有12枚银币。其中有11枚真币和1枚假币。假币看起来和真币没有区别，但是重量不同。但赛利不知道假币比真币轻还是重。于是他向朋友借了一架天平。朋友希望赛利称三次就能找出假币并且确定假币是轻是重。例如:如果赛利用天平称两枚硬币，发现天平平衡，说明两枚都是真的。如果赛利用一枚真币与另一枚银币比较，发现它比真币轻或重，说明它是假币。经过精心安排每次的称量，赛利保证在称三次后确定假币。

输入

第一行有一个数字n，表示有n组测试用例。
对于每组测试用例：
输入有三行，每行表示一次称量的结果。赛利事先将银币标号为A-L。每次称量的结果用三个以空格隔开的字符串表示：天平左边放置的硬币 天平右边放置的硬币 平衡状态。其中平衡状态用`up'',down'', 或 ``even''表示, 分别为右端高、右端低和平衡。天平左右的硬币数总是相等的。

输出

输出哪一个标号的银币是假币，并说明它比真币轻还是重(heavy or light)。

样例输入

```
1
ABCD EFGH even
ABCI EFJK up
ABIJ EFGH even
```

样例输出

```
K is the counterfeit coin and it is light.
```

来源：East Central North America 1998，计算概论05

解题思路：在题目描述中，已经明确保证三次称重后能够确定假币，即输入的三组称量数据有唯一的答案。硬币有三种状态：较重的假币、较轻的假币、真币。由于只有1枚假币，且总共只有12枚银币，因此可以对所有的情况进行枚举。假币可能是任意一枚，有12种情况：且假币可能比真币重，也可能比真币轻，有两种情况。在这全部的24种情况当中，只有一种情况能够符合三组称量数据，这就是所要寻找的答案。

假设用0、-1和1表示真币、较轻币和较重币的重量。分别计算天平左侧与右侧的重量，若下面三个条件中有一个不满足，则说明假设不成立，当前的情况与称量数据矛盾。

（1）如果天平左侧较重，且称量结果为up.

（2）如果天平右侧较重，且称量结果为down.

（3）如果天平左右两侧重量相同，且称量结果为even.

解题思路：可以把银币分类，真币的重量为 0,轻假币为-1，重假币为 1，然后把秤变为条件，依次对所有银币假设为假币，总共有 24种情况，遍历一遍就可以了。如果通过三个秤(条件)，就输出当前的银币。

```
status = [0]*12
left = ['' for _ in range(3)]
right = ['' for _ in range(3)]
result = ['' for _ in range(3)]

def Balanced():
    for _ in range(3):
        leftW = rightW = 0
        for k in range(len(left[i])):
            leftW += status[ord(left[i][k]) - ord('A')]
            rightW += status[ord(right[i][k]) - ord('A')]

        if leftW>rightW and result[i][0]!='u':
            return False
        if leftW == rightW and result[i][0]!='e':
            return False
        if leftW<rightW and result[i][0]!='d':
            return False

    return True

for _ in range(int(input())):
    for i in range(3):
        left[i], right[i], result[i] = input().split()

    for i in range(12):
        status[i] = 0

    i = 0
    for i in range(12):
        status[i] = 1
        if Balanced():
            break

        status[i] = -1
        if Balanced():
            break

        status[i] = 0
```

```
    print('{} is the counterfeit coin and it is {}.'.format( chr(i+ord('A')), \
                    "heavy" if status[i]>0 else "light"))
```

2020fall-cs101，李逸晨

解题思路：2020fall-cs101，方磊。参考了李逸晨大佬的代码，觉得大佬的思路真的非常清晰而且解法应该是题解中最好的。

对一个轻的假币来说（重的同理），在称量中，只要它在天平左边天平右边就会下降，在天平右边天平右边就会上升，不在天平上天平两边就会平衡。只有轻的假币才会满足这种特性。

如果三个式子都成立，那么这就是假币。

```
# https://stackabuse.com/any-and-all-in-python-with-examples/
# The method any(iterable) behaves like a series of or operators between
#   each element of the iterable we passed.
# The all(iterable) method evaluates like a series of and operators between
#   each of the elements in the iterable we passed.

for _ in range(int(input())):
    L = [[],[],[]]
    for i in range(3):
        L[i] = input().split()
    for f in 'ABCDEFGHIJKL':
        if all((f in i[0] and i[2]=='up') or (f in i[1] and i[2]=='down')
               or ( f not in i[0] + i[1] and i[2]=='even') for i in L):
            print("{} is the counterfeit coin and it is {}.".format(f,'heavy'))
            break
        if all((f in i[0] and i[2]=='down') or (f in i[1] and i[2]=='up')
               or (f not in i[0]+i[1] and i[2]=='even') for i in L):
            print("{} is the counterfeit coin and it is {}.".format(f,'light'))
            break
```

7813: 圣诞老人的礼物-Santa Clau’s Gifts

greedy, http://cs101.openjudge.cn/practice/7813

圣诞节来临了，在城市A中圣诞老人准备分发糖果，现在有多箱不同的糖果，每箱糖果有自己的价值和重量，每箱糖果都可以拆分成任意散装组合带走。圣诞老人的驯鹿最多只能承受一定重量的糖果，请问圣诞老人最多能带走多大价值的糖果。

输入

第一行由两个部分组成，分别为糖果箱数正整数n(1 <= n <= 100)，驯鹿能承受的最大重量正整数w（0 < w < 10000），两个数用空格隔开。其余n行每行对应一箱糖果，由两部分组成，分别为一箱糖果的价值正整数v和重量正整数w，中间用空格隔开。

输出

输出圣诞老人能带走的糖果的最大总价值，保留1位小数。输出为一行，以换行符结束。

样例输入

```
4 15
100 4
412 8
266 7
591 2
```

样例输出

```
1193.0
```

解题思路：计算平均值降序取，注意每箱糖果都可以拆分成任意散装组合带走。

```
N, TW = map(int, input().split())   # TW = total weight
bags = []
for _ in range(N):
    v, w = map(int, input().split())
    x = v/w
    bags.append([x, v, w])

bags.sort(reverse = True)

ans = 0
for i in bags:
    if i[2] <= TW:
        ans += i[1]
        TW -= i[2]
    else:
        ans += TW * i[0]
        break

print("{:.1f}".format(ans))
```

```
n, tw = map(int, input().split())

d = dict()
for _ in range(n):
    v, w = map(int, input().split())
    t = v/w
    if t not in d:
        d[t] = [v, w]
    else:
        d[t] = [d[t][0]+v, d[t][1]+w]

a = sorted(d, reverse=True)

ans = 0
for i in a:
    if d[i][1] <= tw:
        ans += d[i][0]
        tw -= d[i][1]
    else:
```

```
        ans = ans + tw * i
        break

print("{:.1f}".format(ans))
```

12065: 方程求解

binary search, http://cs101.openjudge.cn/practice/12065

求下面方程的根：$f(x) = x^3 - 5x^2 + 10x - 80 = 0$。

输入

-

输出

精确到小数点后9位。

解题思路：对$f(x)$求导，得到$f'(x) = 3x^2 - 10x + 10$。由一元二次方程求根公式可知方程$f'(x)$无解，因此$f'(x) > 0$恒成立，一元三次方程$f(x)$关于x是单调递增的，且$f(0) = -80 < 0$而$f(10) = 520 > 0$，则根必在$[0, 10]$之间，由于$f(x)$在$[0, 10]$内是单调递增的，所以可以用二分查找的办法在区间中寻找根。

由此可以看出二分查找的一个应用：难以求解或者无法直接求解的方程求根问题，首先求其范围，再进一步缩小范围，逼近要求的解。二分查找的好处是每次二分都将查找的范围缩小一半。

2020fall-cs101，苏荣薰。

解题思路：找到一个整数区间[a,b]，使 f(a)f(b)<0，便可由零点存在定理知[a,b]内存在 f(x)=0的一个解。然后不断二分区间套用上述方法，直到所得到的解足够要求的精度。做的时候想不到判断精度够不够的方法，上网找了恍然大悟可以用区间长度来判断，如果比要求精度小就说明已经达到所求精度。

2020fall-cs101，胡新越

```
left = 0
right = 10
eps = 1e-12

func = lambda x:x**3 - 5*(x**2) + 10*x - 80
while right - left > eps:
    mid = (left + right)/2
    if func(mid) > 0:
        right = mid
    else:
        left = mid
print(format(right, ".9f"))
```

终止，逼近0，就是小于一个很小的数。

```
left = 0
right = 10
e = 1e-12

f = lambda x:x**3 - 5*(x**2) + 10*x - 80
```

```
while True:
    x = (left + right)/2
    y = f(x)

    if abs(y) < e:
        print('{:.9f}'.format(x))
        break

    if y < 0:
        left = x
    elif y>0:
        right = x
```

2020fall-cs101，施惟明。迭代法

```
def f(x):
    return x*x*x-5*x*x+10*x-80
def ff(x):
    return 3*x*x-10*x+10
x =0
x0 = -10000
while abs(x-x0) > 1e-10:
    x0 = x
    x = x - f(x)/ff(x)

print('%.9f' % x)
```

2020fall-cs101，李博海

其实我这个不是二分搜索，是“迭代法”，这种解方程方法来源于以前的计算器实用技巧（甚至可以解多元任意方程组），时间复杂度未知，且迭代能否收敛看天。

```
prex, x = 0, 5
while abs(x - prex) > 1e-11:
    prex = x
    x = (5*prex*prex - 10*prex + 80) ** (1/3)

print('{:.9f}'.format(x))
```

2020fall-cs101，阚立言。

用了大概一个小时多，写了一个使用牛顿法解方程的函数，可以解任意多项式方程，改一下代码用exec()，放进去math库或许还能解超越方程。输入表达式，初值，步长即可求解。受限于方法，一次只能求一个解。但是这个人工求导精度很差，不过通常够用了。

讨论：二分法对于初值是有要求的，如果零点不在区间内会error，但是二分法的好处精度可控，说精确到九位就是九位，而且精度可以做到最好。

牛顿法好在无论多么离谱的初值都能给算出解来，而且速度一般比二分法快，但是因为手动涉及到小量的除法，精度就不好说了（设置收敛半径过小可能出现死循环）。

不过综合各种因素看，还是牛顿法更加普适些。

```
# Newton's method
def solve():
```

```
    global a
    #c = input()
    c = "x**3-5*x**2+10*x-80"
    #a = float(input())
    a = 4.0

    def f(x):
        return eval(c)

    def df(x,dx):
        return (f(x+dx)-f(x))/dx

    while abs(f(a)) >1e-10:
        if df(a,1e-10) == 0:
            a -= 1e-10
        else:
            a = a-f(a)/df(a,1e-10)
    return a

ans = solve()
print("{:.9f}".format(ans))
```

19963: 买学区房 v0.3

sort/math, http://cs101.openjudge.cn/practice/19963

小明同学的家长为了让小明同学接受更好的教育，最近在考虑买某重点中学附近的房子，已知他们买房子要考虑两个因素：房子离学校的距离，以及房子的价格。现在他们有一系列备选的房子，已知这些房子距离学校的x方向距离和y方向距离，以及每栋房子的价格。小明的家长认为，只有同时满足了以下两个条件的房子H买了才是不亏的：

1.小明的家长比较精打细算，所以房子的性价比大于所有备选房子性价比的中位数（定义见下图）
2.小明的家长想攒钱，所以房子的价格小于所有备选房子价格的中位数

注：
性价比 = 房子和学校之间的交通距离 / 房子的价格
（由于该城市的街道布局接近方格状，且从学校到房子无法穿墙而过，房子H去学校的交通距离定义为，房子H距离学校的x方向距离和y方向距离的和）

现在需要你来帮小明的家长判断，一系列备选房子里，值得买的房子有多少栋。

Even-sized population [edit]

Consider an ordered population of 10 data values {3, 6, 7, 8, 8, 10, 13, 15, 16, 20}. What are the 4-quantiles (the "quartiles") of this dataset?

Quartile	Calculation	Result
Zeroth quartile	Although not universally accepted, one can also speak of the zeroth quartile. This is the minimum value of the set, so the zeroth quartile in this example would be 3.	3
First quartile	The rank of the first quartile is 10×(1/4) = 2.5, which rounds up to 3, meaning that 3 is the rank in the population (from least to greatest values) at which approximately 1/4 of the values are less than the value of the first quartile. The third value in the population is 7.	7
Second quartile	The rank of the second quartile (same as the median) is 10×(2/4) = 5, which is an integer, while the number of values (10) is an even number, so the average of both the fifth and sixth values is taken—that is (8+10)/2 = 9, though any value from 8 through to 10 could be taken to be the median.	9
Third quartile	The rank of the third quartile is 10×(3/4) = 7.5, which rounds up to 8. The eighth value in the population is 15.	15
Fourth quartile	Although not universally accepted, one can also speak of the fourth quartile. This is the maximum value of the set, so the fourth quartile in this example would be 20. Under the Nearest Rank definition of quantile, the rank of the fourth quartile is the rank of the biggest number, so the rank of the fourth quartile would be 10.	20

So the first, second and third 4-quantiles (the "quartiles") of the dataset {3, 6, 7, 8, 8, 10, 13, 15, 16, 20} are {7, 9, 15}. If also required, the zeroth quartile is 3 and the fourth quartile is 20.

Odd-sized population [edit]

Consider an ordered population of 11 data values {3, 6, 7, 8, 8, 9, 10, 13, 15, 16, 20}. What are the 4-quantiles (the "quartiles") of this dataset?

Quartile	Calculation	Result
Zeroth quartile	Although not universally accepted, one can also speak of the zeroth quartile. This is the minimum value of the set, so the zeroth quartile in this example would be 3.	3
First quartile	The first quartile is determined by 11×(1/4) = 2.75, which rounds up to 3, meaning that 3 is the rank in the population (from least to greatest values) at which approximately 1/4 of the values are less than the value of the first quartile. The third value in the population is 7.	7
Second quartile	The second quartile value (same as the median) is determined by 11×(2/4) = 5.5, which rounds up to 6. Therefore, 6 is the rank in the population (from least to greatest values) at which approximately 2/4 of the values are less than the value of the second quartile (or median). The sixth value in the population is 9.	9

输入

第1行为1个整数，n，代表备选房子的数目
第2行为n个房子离学校的x，y距离对，如“(x,y)”，距离均为整数
第3行为n个整数，代表每个房子的价格

输出

一个整数，代表n个房子中值得买的房子数目。

样例输入

```
Sample1 Input:
5
(100,200) (50,50) (100,300) (150,50) (50,50)
100 300 200 400 500

Sample1 Output:
2

说明：共有5个房子备选，离学校的交通距离分别是100+200=300，50+50=100，100+300=400，150+50=200，50+50=100。这些房子的性价比依次为300/100=3， 100/300=1/3， 400/200=2，200/400=0.5， 100/500=0.2，中位数是0.5，满足第1个条件的只有第1个和第3个房子。这些房子的价格中位数是300，因此满足第2个条件的只有第1个和第3个房子。所以同时满足两个条件的有第1个和第3个房子，输出2.
```

样例输出

```
Sample2 Input:
3
(10,90) (20,180) (30,270)
100 200 300

Sample2 Output:
0
```

说明：共有三个房子备选，离学校的交通距离分别是10+90=100，20+180=200，30+270=300。这些房子性价比依次为100/100=1，200/200=1，300/300=1，中位数是1，没有房子满足第1个条件，因此不用考虑第二个条件，输出一定是0.

提示

求中位数,要先进行数据的排序（从小到大）,然后计算中位数的序号,分数据为奇数与偶数两种来求.排序时,相同的数字不能省略.
如果总数个数是奇数的话,按从小到大的顺序,取中间的那个数.
如果总数个数是偶数的话,按从小到大的顺序,取中间那两个数的平均数.

来源：cs101 2019 Final Exam

```
n = int(input())

pairs = [i[1:-1] for i in input().split()]
distances = [ sum(map(int,i.split(','))) for i in pairs]

prices = [int(x) for x in input().split()]

# ratio = distance/price
r = []
for i in range(n):
    r.append(distances[i]/prices[i])


H = zip(r,prices)
H = sorted(H, key=lambda x: (-x[0],x[1]))

#print(H)

prices.sort()
r.sort()

import math
if n%2 == 0:
    rank = int(n/2)
    price_sq = (prices[rank-1] + prices[rank])/2
    r_sq = (r[rank-1] + r[rank])/2
else:
    rank = math.ceil(n/2)
    price_sq = prices[rank-1]
    r_sq = r[rank-1]

cnt = 0
for h in H:
    if h[0]>r_sq and h[1]<price_sq:
```

```
        cnt += 1

print(cnt)
```

19948: 因材施教

greedy, http://cs101.openjudge.cn/practice/19948

有一所魔法高校招入一批学生，为了贯彻因材施教的理念，学校打算根据他们的魔法等级进行分班教育。在确定班级数目的情况下，班级内学生的差异要尽可能的小，也就是各个班级内学生的魔法等级要尽可能的接近。
例如：现在有(n = 7)位学生，他们的魔法等级分别为(r = [2, 7, 9, 9, 16, 28, 45])，我们要将他们分配到(m = 3)个班级，如果按照([2, 7], [9, 9], [16, 28, 45])的方式分班，则他们的总体差异为(d = (7 - 2) + (9 - 9) + (45 - 16) = 34)。

输入

第一行为两个整数:学生人数n和班级数目m，1 <= m <= n <= 10^5。
第二行为n个整数：每位学生的魔法等级ri，1 <= ri <= 10^9。

输出

一个整数：学生的最小总体差异d。

样例输入

```
Sample1 Input
7 3
2 7 9 9 16 28 45

Sample1 Output
14

解释：最小总体差异的分班方式为([2, 7, 9, 9, 16], [28], [45])
```

样例输出

```
Sample2 Input
15 9
90 73 116 47 400 212 401 244 13 372 248 56 194 482 177

Sample2 Output
65

解释：最小总体差异的分班方式为([13], [47, 56, 73, 90], [116], [177, 194], [212], [244, 248], [372], [400, 401], [482])
```

来源：cs101 2019 Final Exam

思路：本质就是去掉m-1个最大的差值。先升序排序魔法等级，再按照差值逆序排。

```
n,m = map(int,input().split())
r = [int(x) for x in input().split()]
r.sort()
rd = []
```

```
for i in range(len(r)-1):
    rd.append(r[i+1] - r[i])

rd.sort(reverse=True)
d = sum(rd)

for i in rd:
    d -= i
    m -= 1
    if m==1:
        break

print(d)
```

思路：无论怎么分班，最后的总体差异一定是 n-m个两两差异的和，将列表排序后两两作差，再将两两差异排序，取前 n-m个数之和即为答案取前 n-m个数之和即为答案。

n个数俩俩做差得n-1个差，其中最大的m-1个差单独分班，剩余的分到一个班（n-1-(m-1)）=n-m.

```
n,m = map(int,input().split())
r = [int(x) for x in input().split()]
r.sort()
rd = []
for i in range(len(r)-1):
    rd.append(r[i+1] - r[i])
rd.sort()
print(sum(rd[:n-m]))
```

OPTIONAL PROBLEMS

12559: 最大最小整数 v0.2

string/sorting, http://cs101.openjudge.cn/practice/12559

假设有n个正整数，将它们连成一片，将会组成一个新的大整数。现需要求出，能组成的最大最小整数。

比如，有4个正整数，23，9，182，79，连成的最大整数是97923182，最小的整数是18223799。

输入

第一行包含一个整数n，1<=n<=1000。
第二行包含n个正整数，相邻正整数间以空格隔开。

输出

输出为一行，为这n个正整数能组成的最大的多位整数和最小的多位整数，中间用空格隔开。

样例输入

```
Sample1 in:
4
23 9 182 79

Sample1 out:
97923182 18223799
```

样例输出

```
Sample2 in:
2
11 113

Sample2 out:
11311 11113
```

思路：求minimum时，对相邻两strA[k]与A[k+1]，比较A[k]+A[k+1]与A[k+1]+A[k]的大小，若A[k+1]+A[k]大，颠倒A[k]与A[k+1]；最多交换len(A)-1次。求maximum时，颠倒求minimum时的有序序列即可。使用冒泡排序，循环(n-1)次。

把这些数当成字符串处理，然后采用类似冒泡排序的做法排出大小。

```
n = int(input())
nums = input().split()
for i in range(n - 1):
    for j in range(i+1, n):
        #print(i,j)
        if nums[i] + nums[j] < nums[j] + nums[i]:
            nums[i], nums[j] = nums[j], nums[i]

ans = "".join(nums)
nums.reverse()
print(ans + " " + "".join(nums))
```

2020fall-cs101，黄旭

思路：这道题的关键应该是找到排序的方式，前一个数和后一个数比较，如果位数不足，就要重新从第一位开始比，所以说我就先取这个数列的最大位数，然后把每个数都扩充到相同位数进行比较，就可以了。

```
from math import ceil
input()
lt = input().split()

max_len = len(max(lt, key = lambda x:len(x)))
lt.sort(key = lambda x: tuple([int(i) for i in x]) * ceil(max_len/len(x)))
lt1 = lt[::-1]
print(''.join(lt1),''.join(lt))
```

21608: 你和你比较熟悉的同学

bfs/dfs, http://cs101.openjudge.cn/practice/21608

2020年秋季学期，Henry老师讲授北京大学《计算概论B》课程，其中12班127人，13班37人。临近期末，12月18日做了一个简单的问卷调查。以这种方式做为一次点名，同时也希望了解学生们的一些情况。

问卷内容是：写出自己和脑海中的班里（包括12班、和13班）的另外4名同学的名字。每位同学的**名字用一个不重复的正整数表示。**如果认识的同学少于4个，也可以少填写。**每位同学最多提交1次问卷调查。**

根据收集到的结果，形成了一张班级社会网络图 G = (V, E)，如下所示。V是顶点的集合，E是边的集合。在这个有向图G中，每位同学是一个顶点，边是认识关系。**注意认识关系是单向的**，即甲认识乙，不代表乙认识甲。

Henry老师想知道在有向图G中，找最大有向子树，所包含的顶点的个数。

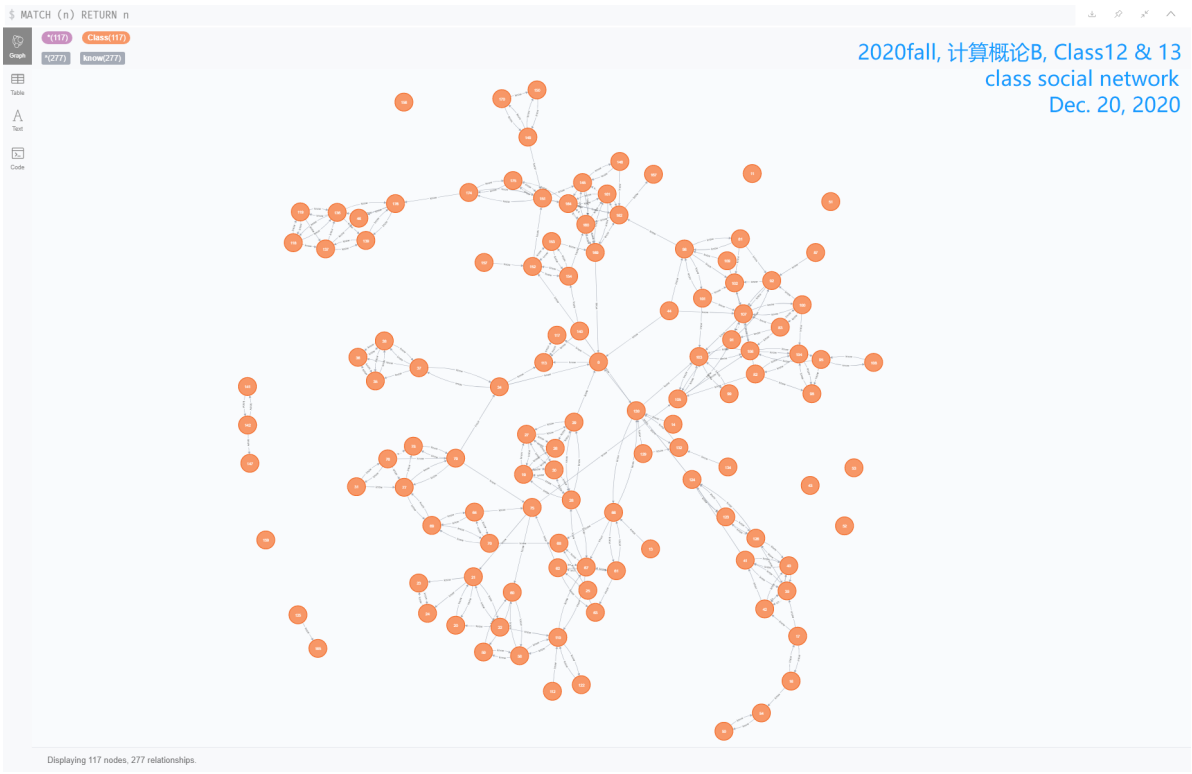
【说明：

1）对于一个点，计算从这个点（包括自己）开始沿着有向边能到达的点的数量。

2）用数学的定义就是求图中是根数的子图的顶点最大值。

3）不是强连通图，单向能到就可以

】



输入格式

输入第一行为一个数字 n（$1 < n \leq 200$），表示调查问卷反馈人数。

第$2 \sim n+1$ 行是同学之间的认识关系。
每一行中，分号之前是提交问卷同学名字，之后是认识的同学名字。
-1表示没有熟悉的同学。
分号及整数，由空格分隔。

输出格式

输出一个整数。表示最大半连通子图中所包含的顶点的个数。

样例输入1

```
sample1 in:
5
53 : -1
118 : 119 136 137
92 : 107 93 102 91
102 : -1
130 : 66 132 135 103
```

样例输出1

```
5
```

样例输入2

```
13
53 : -1
118 : 119 136 137
92 : 107 93 102 91
102 : -1
130 : 66 132 135 103
42 : 39 40 41
39 : 42 40 41 127
43 : 94
134 : 135 132 131 128
136 : 119 118 176 139
17 : 16 39 42
96 : 81 102 162 101
141 : 142
```

样例输出2

```
7
```

解题思路：bfs, dfs都可以过。dfs可以练习递归写法，非递归写法。

```python
# OJ 21608
# https://www.codespeedy.com/breadth-first-search-algorithm-in-python/
def bfs(graph, initial):
    visited = []
    queue = [initial]

    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)

            if node not in graph:
                continue

            for nei in graph[node]:       # neighbour
                queue.append(nei)
```

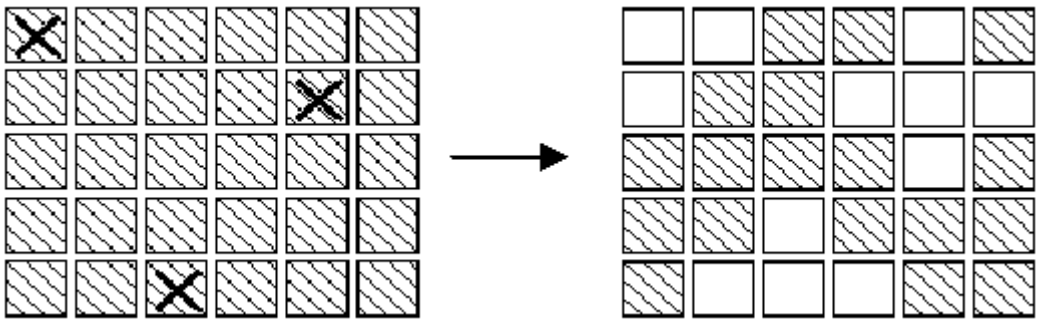
```
    return visited

# https://stackoverflow.com/questions/43430309/depth-first-search-dfs-code-in-
python
# https://www.codespeedy.com/depth-first-search-algorithm-in-python/
def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        if node not in graph:
                return visited

        for nei in graph[node]:         # neighbour
            dfs(graph, nei, visited)
    return visited

def dfs_Non_recursion(graph, initial):
    visited = [initial]
    stack = [initial]
    while stack:
        node = stack[-1]
        if node not in visited:
            visited.append(node)

        if node not in graph:
            stack.pop()
            continue

        remove_from_stack = True
        for nei in graph[node]:         # neighbour
            if nei not in visited:
                stack.append(nei)
                remove_from_stack = False
                break
        if remove_from_stack:
            stack.pop()
    return visited

## create graph
graph = {}
ids = set()

for _ in range(int(input())):
    ln = input().split(':')
    u = int(ln[0].rstrip())
    ids.add(u)
    if u not in graph:
        neighbours = [int(_) for _ in ln[1].split()]
        graph[u] = neighbours

## bfs travel
maxp = 0
for i in ids:
    bfs_path = bfs(graph, i)
    if -1 in bfs_path:
        bfs_path.remove(-1)
    maxp = max(maxp, len(bfs_path))
    #print(len(bfs_path), bfs_path)
#print(maxp)
```

```
## dfs_Non_recursion travel
maxp = 0
for i in ids:
    dfs_path = dfs_Non_recursion(graph, i)
    if -1 in dfs_path:
        dfs_path.remove(-1)
    maxp = max(maxp, len(dfs_path))
    #print(len(bfs_path), bfs_path)
print(maxp)

## dfs travel
maxp = 0
for i in ids:
    dfs_path = dfs(graph, i, [])
    if -1 in dfs_path:
        dfs_path.remove(-1)
    maxp = max(maxp, len(dfs_path))
    #print(len(dfs_path), dfs_path)
#print(maxp)
```

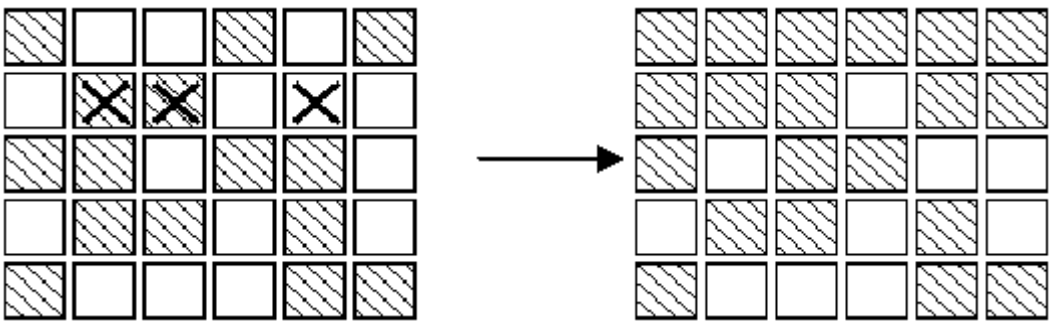
1813: 熄灯问题

brute force, http://cs101.openjudge.cn/practice/1813

有一个由按钮组成的矩阵，其中每行有6个按钮，共5行。每个按钮的位置上有一盏灯。当按下一个按钮后，该按钮以及周围位置(上边、下边、左边、右边)的灯都会改变一次。即，如果灯原来是点亮的，就会被熄灭；如果灯原来是熄灭的，则会被点亮。在矩阵角上的按钮改变3盏灯的状态；在矩阵边上的按钮改变4盏灯的状态；其他的按钮改变5盏灯的状态。



在上图中，左边矩阵中用X标记的按钮表示被按下，右边的矩阵表示灯状态的改变。对矩阵中的每盏灯设置一个初始状态。请你按按钮，直至每一盏等都熄灭。与一盏灯毗邻的多个按钮被按下时，一个操作会抵消另一次操作的结果。在下图中，第2行第3、5列的按钮都被按下，因此第2行、第4列的灯的状态就不改变。



请你写一个程序，确定需要按下哪些按钮，恰好使得所有的灯都熄灭。根据上面的规则，我们知道

1）第2次按下同一个按钮时，将抵消第1次按下时所产生的结果。因此，每个按钮最多只需要按下一次；

2）各个按钮被按下的顺序对最终的结果没有影响；

3）对第1行中每盏点亮的灯，按下第2行对应的按钮，就可以熄灭第1行的全部灯。如此重复下去，可以熄灭第1、2、3、4行的全部灯。同样，按下第1、2、3、4、5列的按钮，可以熄灭前5列的灯。

输入

5行组成，每一行包括6个数字（0或1）。相邻两个数字之间用单个空格隔开。0表示灯的初始状态是熄灭的，1表示灯的初始状态是点亮的。

输出

5行组成，每一行包括6个数字（0或1）。相邻两个数字之间用单个空格隔开。其中的1表示需要把对应的按钮按下，0则表示不需要按对应的按钮。

样例输入

```
0 1 1 0 1 0
1 0 0 1 1 1
0 0 1 0 0 1
1 0 0 1 0 1
0 1 1 1 0 0
```

样例输出

```
1 0 1 0 0 1
1 1 0 1 0 1
0 0 1 0 1 1
1 0 0 1 0 0
0 1 0 0 0 0
```

2020fall-cs101，曲诤博。题本来是很难的，还好题干中给了提示，最后的难点有两个，一个是如何生成 0，1 排列组合的所有 64 个 list，这里我用了迭代的方法，还有一个就是如何实现题干的方法，这里用了一下深拷贝，之后就没啥问题了。

2020fall-cs101，顾臻宜。解题思路：A 记开关实时亮暗，B记开关操作与否。一旦确定第1 行的操作，剩
余第2、3、4、5 行的操作就确定；枚举第1 行（共2^6=64 种可能）。
小知识：二维数组需要深度拷贝：import copy；A=copy.deepcopy(X)。

```
X = [[0,0,0,0,0,0,0,0]]
Y = [[0,0,0,0,0,0,0,0]]
for _ in range(5):
    X.append([0] + [int(x) for x in input().split()] + [0])
    Y.append([0 for x in range(8)])
X.append([0,0,0,0,0,0,0,0])
Y.append([0,0,0,0,0,0,0,0])

import copy
for a in range(2):
    Y[1][1] = a
    for b in range(2):
        Y[1][2] = b
        for c in range(2):
```

```
            Y[1][3] = c
            for d in range(2):
                Y[1][4] = d
                for e in range(2):
                    Y[1][5] = e
                    for f in range(2):
                        Y[1][6] = f

                        A = copy.deepcopy(X)
                        B = copy.deepcopy(Y)
                        for i in range(1, 7):
                            if B[1][i] == 1:
                                A[1][i] = abs(A[1][i] - 1)
                                A[1][i-1] = abs(A[1][i-1] - 1)
                                A[1][i+1] = abs(A[1][i+1] - 1)
                                A[2][i] = abs(A[2][i] - 1)
                        for i in range(2, 6):
                            for j in range(1, 7):
                                if A[i-1][j] == 1:
                                    B[i][j] = 1
                                    A[i][j] = abs(A[i][j] - 1)
                                    A[i-1][j] = abs(A[i-1][j] - 1)
                                    A[i+1][j] = abs(A[i+1][j] - 1)
                                    A[i][j-1] = abs(A[i][j-1] - 1)
                                    A[i][j+1] = abs(A[i][j+1] - 1)
                        if A[5][1]==0 and A[5][2]==0 and A[5][3]==0 and A[5]
[4]==0 and A[5][5]==0 and A[5][6]==0:
                            for i in range(1, 6):
                                print(" ".join(repr(y) for y in [B[i][1],B[i]
[2],B[i][3],B[i][4],B[i][5],B[i][6] ]))
```

2020fall-cs101，李元锋。思路：最难的一题，暴力一定超时。从第一行开始，注意到如果要消除第一行的灯，必然要按它下面的灯，这样就可以把枚举情况降至 2^6。先用 iterations模块生成所有 6个(0,1)组合的 list，然后根据 list判断下一步做什么，以此类推。最后生成答案即可。

2020fall-cs101，李宗远

```
from copy import deepcopy
from itertools import product
rmap = {0:1, 1:0}
matrix_backup = [[0] * 8] + [[0, *map(int, input().split()), 0] for i in
range(5)] \
    + [[0] * 8]

for test in product(range(2), repeat=6):
    matrix = deepcopy(matrix_backup)
    triggers = [list(test)]
    for i in range(1, 6):
        for j in range(1, 7):
            if triggers[i - 1][j - 1]:
                matrix[i][j] = rmap[matrix[i][j]]
                matrix[i - 1][j] = rmap[matrix[i - 1][j]]
                matrix[i + 1][j] = rmap[matrix[i + 1][j]]
                matrix[i][j - 1] = rmap[matrix[i][j - 1]]
                matrix[i][j + 1] = rmap[matrix[i][j + 1]]
```

```
            triggers.append(matrix[i][1:7])
    if matrix[5][1:7] == [0, 0, 0, 0, 0, 0]:
        for trigger in triggers[:-1]:
            print(' '.join(map(str, trigger)))
```

1289: Tian Ji -- The Horse Racing

greedy, http://cs101.openjudge.cn/practice/1289

Here is a famous story in Chinese history.

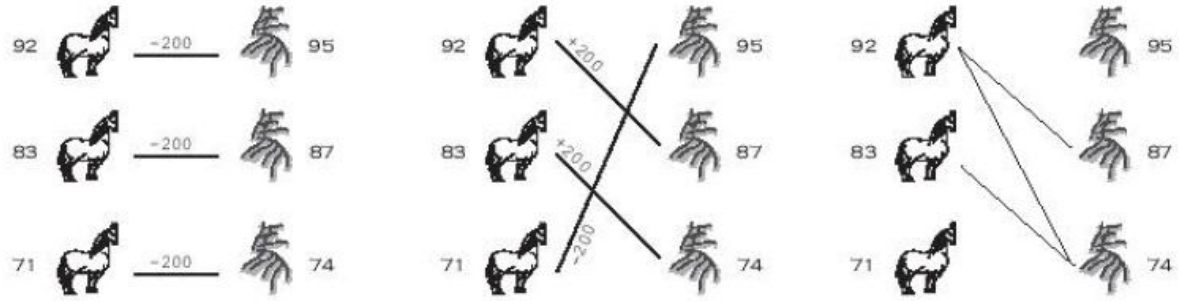
That was about 2300 years ago. General Tian Ji was a high official in the country Qi. He likes to play horse racing with the king and others.

Both of Tian and the king have three horses in different classes, namely, regular, plus, and super. The rule is to have three rounds in a match; each of the horses must be used in one round. The winner of a single round takes two hundred silver dollars from the loser.

Being the most powerful man in the country, the king has so nice horses that in each class his horse is better than Tian's. As a result, each time the king takes six hundred silver dollars from Tian.

Tian Ji was not happy about that, until he met Sun Bin, one of the most famous generals in Chinese history. Using a little trick due to Sun, Tian Ji brought home two hundred silver dollars and such a grace in the next match.

It was a rather simple trick. Using his regular class horse race against the super class from the king, they will certainly lose that round. But then his plus beat the king's regular, and his super beat the king's plus. What a simple trick. And how do you think of Tian Ji, the high ranked official in China?



Were Tian Ji lives in nowadays, he will certainly laugh at himself. Even more, were he sitting in the ACM contest right now, he may discover that the horse racing problem can be simply viewed as finding the maximum matching in a bipartite graph. Draw Tian's horses on one side, and the king's horses on the other. Whenever one of Tian's horses can beat one from the king, we draw an edge between them, meaning we wish to establish this pair. Then, the problem of winning as many rounds as possible is just to find the maximum matching in this graph. If there are ties, the problem becomes more complicated, he needs to assign weights 0, 1, or -1 to all the possible edges, and find a maximum weighted perfect matching...

However, the horse racing problem is a very special case of bipartite matching. The graph is decided by the speed of the horses -- a vertex of higher speed always beat a vertex of lower speed. In this case, the weighted bipartite matching algorithm is a too advanced tool to deal with the problem.

In this problem, you are asked to write a program to solve this special case of matching problem.

输入

The input consists of up to 50 test cases. Each case starts with a positive integer n (n<=1000) on the first line, which is the number of horses on each side. The next n integers on the second line are the speeds of Tian's horses. Then the next n integers on the third line are the speeds of the king's horses. The input ends with a line that has a single `0' after the last test case.

输出

For each input case, output a line containing a single number, which is the maximum money Tian Ji will get, in silver dollars.

样例输入

```
3
92 83 71
95 87 74
2
20 20
20 20
2
20 19
22 18
0
```

样例输出

```
200
0
0
```

来源：Shanghai 2004

2020fall-cs101，顾臻宜。解题思路：以max 赢max，以min 赢min，以min 输max；不断del更新。一开始看群里讨论以为分行输入是指一行田忌一行国王交替进行，就是不AC；出门吹了会冷风之后回来又试了一次田忌所有的数据分行输完、再国王，AC了。

```
for _ in range(50):
    n = int(input())
    if n==0:
        break
    A = [[], []]
    for _ in range(n):
        for x in input().split():
            A[0].append(int(x))
        if len(A[0]) == n:
            break
    for _ in range(n):
        for y in input().split():
            A[1].append(int(y))
        if len(A[1]) == n:
            break
```

```
    A[0].sort(reverse=True)
    A[1].sort(reverse=True)

    answer = 0

    for _ in range(n):
        if A[0][0] > A[1][0]:
            answer += 1
            del A[0][0]
            del A[1][0]
        else:
            if A[0][-1] > A[1][-1]:
                answer += 1
                del A[0][-1]
                del A[1][-1]
            else:
                if A[0][-1] < A[1][0]:
                    answer -= 1
                del A[0][-1]
                del A[1][0]

    print(200*answer)
```

2442: 4 Values whose Sum is 0

binary search, http://cs101.openjudge.cn/practice/2442

The SUM problem can be formulated as follows: given four lists A, B, C, D of integer values, compute how many quadruplet (a, b, c, d) ∈ A x B x C x D are such that a + b + c + d = 0 . In the following, we assume that all lists have the same size n .

输入

The first line of the input file contains the size of the lists n (this value can be as large as 4000). We then have n lines containing four integer values (with absolute value as large as 228) that belong respectively to A, B, C and D .

输出

For each input file, your program has to write the number quadruplets whose sum is zero.

样例输入

```
6
-45 22 42 -16
-41 -27 56 30
-36 53 -37 77
-36 30 -75 -46
26 -38 -10 62
-32 -54 -6 45
```

样例输出

```
5
```

提示

Sample Explanation: Indeed, the sum of the five following quadruplets is zero: (-45, -27, 42, 30), (26, 30, -10, -46), (-32, 22, 56, -46),(-32, 30, -75, 77), (-32, -54, 56, 30).

思路：利用dict实现，查找是O(1)，保证不超时。用两个dict记录a和b的和的组合数，还有c和d的和的组合数，结果空间爆了。只记录a和b之和的组合数，再遍历c和d之和的时候边检验是否有a和b之和的相反数，若有就加对应的组合数。

```
n = int(input())
a = [0]*(n+1)
b = [0]*(n+1)
c = [0]*(n+1)
d = [0]*(n+1)

for i in range(n):
    a[i],b[i],c[i],d[i] = map(int, input().split())

dict1 = {}
for i in range(n):
    for j in range(n):
        if not a[i]+b[j] in dict1:
            dict1[a[i] + b[j]] = 0
        dict1[a[i] + b[j]] += 1

ans = 0
for i in range(n):
    for j in range(n):
        if -(c[i]+d[j]) in dict1:
            ans += dict1[-(c[i]+d[j])]

print(ans)
```

```
# https://docs.python.org/3/library/array.html
import array as arr

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())


dict1 = {}
for i in range(n):
    for j in range(n):
        if not a[i]+b[j] in dict1:
            dict1[a[i] + b[j]] = 0
        dict1[a[i] + b[j]] += 1
```

```
ans = 0
for i in range(n):
    for j in range(n):
        if -(c[i]+d[j]) in dict1:
            ans += dict1[-(c[i]+d[j])]

print(ans)
```

二分查找解题思路：在这本书的89页。《算法基础与在线实践》郭炜等编著，2017年。

可以先考虑更简单的情况：给定两组整数a、b，求出和为0的二元组的个数。思路是：给定了两数之和为0，对a中每一个数a[i]，判断-a[i]是否在b中。这样问题就转化为了一个查找问题，可以使用二分查找加快查找的速度。首先对输入元素进行排序，从小到大枚举每一个元素a[i]，使用二分查找判断-a[i]是否在数组中，然后计算出现的次数。

回到本题，面对四组整数之和的问题，可以将问题转化为两组整数的问题。首先枚举出a、b两组所有可能的和sum1，以及c、d两组所有可能的和sum2，将这两组和看成新的数组，然后利用上述思路，使用二分查找计算满足条件的二元组个数了。

为了求出满足条件的二元组的个数，需要采用变形的二分查找，在有重复元素的数组中返回小于或等于目标的最大元素，若返回元素等于目标元素，则沿着数组计数该元素出现的次数这里采用左右都是闭区间的区间规则。若 targe ≤ mid，则 right = mid；若 targe > mid，则 left = mid + 1，两者都保证 left ≤ target ≤ right，并且在遇到重复的 target 时，right 会一直减少到第一次出现 target 的位置。循环终止条件为 left == right。

同学反馈，用Python，或者超时，或者爆内存。所以直接上C++。

C++教程，https://www.runoob.com/cplusplus/cpp-tutorial.html

```
#include <iostream>
#include <algorithm>
using namespace std;

const int MAXN = 4001;
int a[MAXN], b[MAXN], c[MAXN], d[MAXN];
int sum1[MAXN * MAXN], sum2[MAXN * MAXN];
int t = 0;

int BinarySearch(int target)
{
    int num = 0;
    int left = 0, right = t - 1;
    while(left < right){
        int mid = left + (right - left)/2;
        if (target <= sum2[mid])
            right = mid;
        else
            left = mid + 1;
    }
    while(sum2[left]==target && left<t){    // left<t 防止越界
        num++;
        left++;
    }
    return num;
```

```
}
int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i] >> b[i] >> c[i] >> d[i];

    t = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            sum1[t] = a[i] + b[j];
            sum2[t] = c[i] + d[j];
            t++;
        }
    sort(sum1, sum1 + t);
    sort(sum2, sum2 + t);

    int ans = 0;
    for (int i=0; i < t; i++)
        ans += BinarySearch( -sum1[i] );

    cout << ans << '\n';
    return 0;
}
```

提交人 结果 内存 时间 代码长度 语言 提交时间
HFYan(GMyhf) Time Limit Exceeded 53728kB 133010ms 1918 B Python3 2020/12/18 11:15AM

```
# https://stackoverflow.com/questions/2272819/sort-a-part-of-a-list-in-place
# 快速排序实现及其pivot的选取
# https://blog.csdn.net/qq_31903733/article/details/82945605
import random

def quicksort(arr, start , stop):
    if(start < stop):
        pivotindex = partitionrand(arr, start, stop)
        quicksort(arr , start , pivotindex - 1)
        quicksort(arr, pivotindex + 1, stop)

def partitionrand(arr , start, stop):
    randpivot = random.randrange(start, stop)
    arr[start], arr[randpivot] = arr[randpivot], arr[start]
    return partition(arr, start, stop)

def partition(arr,start,stop):
    pivot = start # pivot
    i = start + 1 # a variable to memorize where the
                  # partition in the array starts from.
    for j in range(start + 1, stop + 1):
        if arr[j] <= arr[pivot]:
            arr[i] , arr[j] = arr[j] , arr[i]
            i = i + 1
    arr[pivot] , arr[i - 1] = arr[i - 1] , arr[pivot]
```

```
    pivot = i - 1
    return (pivot)
# end quicksort

def BinarySearch(target):
    num = 0;
    left = 0
    global t
    right = t - 1
    while left < right:
        mid = (left + right)//2;
        if target <= sumr2[mid]:
            right = mid
        else:
            left = mid + 1

    while(sumr2[left]==target and left<t):    # left<t 防止越界
        num += 1
        left += 1

    return num


n = int(input())
a = [0]*(n+1)
b = [0]*(n+1)
c = [0]*(n+1)
d = [0]*(n+1)

# https://docs.python.org/3/library/array.html
import array as arr
sumr1 = arr.array('i', [])
sumr2 = arr.array('i', [])

for i in range(n):
    a[i],b[i],c[i],d[i] = map(int, input().split())

t = 0;
for i in range(n):
    for j in range(n):
        sumr1.append( a[i] + b[j] )
        sumr2.append( c[i] + d[j] )
        t += 1

quicksort(sumr1, 0, t-1)
quicksort(sumr2, 0, t-1)

ans = 0
for i in range(t):
    ans += BinarySearch( -sumr1[i] )

print(ans)
```

18108:池塘数目

matrix/dfs, http://cs101.openjudge.cn/practice/1980

一场暴雨之后，农田里形成了大大小小的池塘。农田用N×M个格子表示，格子有两种状态，有水('W')和无水('.')。相邻两个有水的格子属于一个池塘，一个格子被视作和它周围八个格子都相邻。

现在需要数出在农田里有多少个池塘。

输入

第一行是一个整数，表示一共有 T 组数据。
每组第一行包含两个整数N和M。
接下来的N行，每行有M个字符('W'或者'.')，表示格子的当前状态。字符之间没有空格。

输出

每组数据对应一行，输出农田里的池塘数目。

样例输入

```
2
2 2
w.
.w
10 12
w........ww.
.www.....www
....ww...ww.
.........ww.
.........w..
..w......w..
.w.w.....ww.
w.w.w.....w.
.w.w......w.
..w.......w.
```

样例输出

```
1
3
```

来源：cs101-2016 期末机考备选

思路：设一个函数，将到访过的W标记，然后如果周围没有被标记的W就是新的池塘。用for loop每个地方过一遍就懂有几个池塘了。

2020fall-cs101，汪元正。解题思路：此题便是要算图中连通分支个数。

```
dir = [[-1, 0], [1, 0], [0, -1], [0, 1], [-1, -1], [-1, 1], [1, -1], [1, 1]]
def dfs(x, y):
    if x < 0 or x >= N or y < 0 or y >= M or board[x][y] != 'W':
        return
    board[x][y] = '.'
    for i in range(len(dir)):
        dfs(x + dir[i][0], y + dir[i][1])

T = int(input())
```

```
while T!=0:
    T -= 1
    N, M = map(int, input().split())
    board = []
    ans = 0
    for i in range(N):
        board.append(list(input()))
    for i in range(N):
        for j in range(M):
            if board[i][j] == 'W':
                ans += 1
                dfs(i, j)
    print(ans)
```

18160:最大连通域面积

matrix/dfs, http://cs101.openjudge.cn/practice/18160

一个棋盘上有棋子的地方用（'W'）表示，没有的地方用点来表示，现在要找出其中的最大连通区域，一个格子被视作和它周围八个格子都相邻。

现在需要 找出最大的连通区域的面积是多少，一个格子代表面积为1。

输入

输入的第一行是一个整数，表示一共有 T 组数据。
每组第一行包含两个整数N和M。
接下来的N行，每行有M个字符('W'或者'.')，表示格子的当前状态。字符之间没有空格。

输出

每组数据对应一行，输出最大的连通域的面积，不包含任何空格。

样例输入

```
2
2 2
W.
.W
10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.
```

样例输出

```
2
16
```

来源：cs101-2017 期末机考备选

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

area = 0
def dfs(x,y):
    global area
    if matrix[x][y] == '.':return
    matrix[x][y] = '.'
    area += 1
    for i in range(len(dire)):
        dfs(x+dire[i][0],y+dire[i][1])

T = int(input())

for _ in range(T):
    n,m = map(int,input().split())

    sur = 0
    matrix = [['.' for _ in range(m+2)] for _ in range(n+2)]
    for i in range(1,n+1):
        s = input()
        for j in range(1,m+1):
            matrix[i][j] = s[j-1]

    for i in range(1,n+1):
        for j in range(1,m+1):
            if matrix[i][j] == 'W':
                area = 0
                dfs(i,j)
                sur = max(sur,area)
    print(sur)
```

4101:晶矿的个数

matrix/dfs, http://cs101.openjudge.cn/practice/4101

在某个区域发现了一些晶矿，已经探明这些晶矿总共有分为两类，为红晶矿和黑晶矿。现在要统计该区域内红晶矿和黑晶矿的个数。假设可以用二维地图m[][]来描述该区域，若m[i][j]为#表示该地点是非晶矿地点，若m[i][j]为r表示该地点是红晶矿地点，若m[i][j]为b表示该地点是黑晶矿地点。一个晶矿是由相同类型的并且上下左右相通的晶矿点组成。现在给你该区域的地图，求红晶矿和黑晶矿的个数。

输入

第一行为k，表示有k组测试输入。
每组第一行为n，表示该区域由n*n个地点组成，3 <= n<= 30
接下来n行，每行n个字符，表示该地点的类型。

输出

对每组测试数据输出一行，每行两个数字分别是红晶矿和黑晶矿的个数，一个空格隔开。

样例输入

```
2
6
r##bb#
###b##
#r##b#
#r##b#
#r####
######
4
####
#rrb
#rr#
##bb
```

样例输出

```
2 2
1 2
```

```python
# http://cs101.openjudge.cn/practice/4101/

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

def dfs(x,y,c):
    m[x][y] = '#'
    for i in range(len(dire)):
        tx = x+dire[i][0]
        ty = y+dire[i][1]
        if m[tx][ty]==c:
            dfs(tx,ty,c)

k= int(input())
for _ in range(k):
    n = int(input())
    m = [[0 for _ in range(n+2)] for _ in range(n+2)]

    for i in range(1,n+1):
        s = input()
        for j in range(1,n+1):
            m[i][j] = s[j-1]

    r = 0 ; b=0
    for i in range(1,n+1):
        for j in range(1,n+1):
            if m[i][j] =='r':
                dfs(i,j,'r')
                r +=1
            if m[i][j] =='b':
                dfs(i,j,'b')
                b+=1
    print(r, b)
```

1980: 陪审团的人选

dp, http://cs101.openjudge.cn/practice/1980

在遥远的国家佛罗布尼亚，嫌犯是否有罪，须由陪审团决定。陪审团是由法官从公众中挑选的。先随机挑选n个人作为陪审团的候选人，然后再从这n个人中选m人组成陪审团。选m人的办法是：

控方和辩方会根据对候选人的喜欢程度，给所有候选人打分，分值从0到20。为了公平起见，法官选出陪审团的原则是：选出的m个人，必须满足辩方总分和控方总分的差的绝对值最小。如果有多种选择方案的辩方总分和控方总分的之差的绝对值相同，那么选辩控双方总分之和最大的方案即可。

输入

输入包含多组数据。每组数据的第一行是两个整数n和m，n是候选人数目，m是陪审团人数。注意，1≤ n≤ 200, 1≤ m≤ 20 而且 m≤ n。接下来的n行，每行表示一个候选人的信息，它包含2个整数，先后是控方和辩方对该候选人的打分。候选人按出现的先后从1开始编号。两组有效数据之间以空行分隔。最后一组数据n=m=0

输出

对每组数据，先输出一行，表示答案所属的组号,如 'Jury #1', 'Jury #2', 等。接下来的一行要象例子那样输出陪审团的控方总分和辩方总分。再下来一行要以升序输出陪审团里每个成员的编号，两个成员编号之间用空格分隔。每组输出数据须以一个空行结束。

样例输入

```
4 2
1 2
2 3
4 1
6 2
0 0
```

样例输出

```
Jury #1
Best jury has value 6 for prosecution and value 4 for defence:
 2 3
```

来源：Southwestern European Regional Contest 1996, POJ 1015, 程序设计实习2007

【思路】

设d[j][k]表示该选第j个人且辩方与控方之差为k时最大的辩控和。

设p为辩方分数, d为控方分数，v(i)=p[i]-d[i], S(i)=p[i]+d[i]

有转移式： d[j][k] = max{ d[j-1][k-v(i)] + S(i) }

转移式表示第j个人选i，且i必须要满足在d[j-1][k-v(i)]的最优选择中没有出现过。

用path[j][k]记录差值为k时所选的第j个人，一方面检查i是否出现过，一方面方便构造解。

差值会为负值，因此将差值全部偏移N个单位。

```
# 陪审团的人选, http://cs101.openjudge.cn/practice/1980/
maxn = 400 + 10      # -20 X m <= maxn <= 20 X m, where 1<=m<=20
```

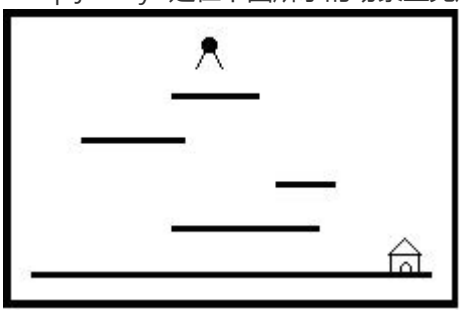
```
cnt = 0
while True:
    try:
        n, m = map(int, input().split())
    except ValueError:      #跳过空行
        continue

    if n + m == 0:
        break

    p = [0]     # 辩方分数
    d = [0]     # 控方分数
    for _ in range(n):
        tp, td = map(int, input().split())
        p.append(tp)
        d.append(td)

    cnt += 1

    if n == m:
        prosecution = sum(p)
        defence = sum(d)
        print('Jury #{}'.format(cnt))
        print("Best jury has value {} for prosecution and value {} for
defence:".format(prosecution, defence))
        print(' '.join(map(str, range(1,n+1))))
        continue

    #f = []     转移方程：f[j][k]=f[j-1][x]+d[i]+p[i]; 且k=x+p[i]-d[i], f[j-1][x]是
满足条件的最大值
    #path = []  记录j个人评审差为k,这种情况下的前一个人j-1是谁

    result = [None] * maxn
    f = [[-1]*5*maxn for _ in range(m+1)]
    path = [[0]*5*maxn for _ in range(m+1)]


    minP_D = 20 * m   # 避免下标为负,所以推广到零以上.题目中的辩控差为0，对应于程序中的辩控
差为20*m
    f[0][minP_D] = 0  # 初始化条件. 选0个人使得辩控差为nMinP_D的方案，其辩控和就是0

    for j in range(m):  # 每次循环选出第j个人，共要选出m人
        for k in range(minP_D*2 + 1):   # 可能的辩控差为[0，nMinP_D*2]
            if f[j][k] >= 0:        # 方案 f(j,k)可行
                for i in range(1, n+1):    # 在方案f(j,k)的基础上，挑下一个人，逐个试
                    if (f[j][k] + p[i] + d[i]) > (f[j+1][k+p[i]-d[i]]):
                        t1 = j
                        t2 = k
                        # 第i个人是否已经在方案f(j,k)中被选上了
                        while (t1 > 0) and (path[t1][t2] != i):
                            t2 = t2 - p[path[t1][t2]] + d[path[t1][t2]]
                            t1 -= 1

                        # 说明第i个人在方案f(j,k)中没有被选上，那么就选它
                        # 形成方案f(j+1,k+p[i]-d[i])
                        if t1 == 0:
                            # 记录新方案的辩控和
```

```
                            f[j+1][k+p[i]-d[i]] = f[j][k] + p[i] + d[i]
                            # 选了第i 个人，就要将其记录在path里
                            path[j+1][k+p[i]-d[i]] = i
    #i = minP_D
    j = k = 0
    # determine minimum possible |D(J)-P(J)|
    # 找选了m个人，且实际辩控差绝对值最小的方案。最终方案的程序中辩控差为k
    while (f[m][minP_D+j] < 0) and (f[m][minP_D-j] < 0):
        j += 1

    if f[m][minP_D+j] > f[m][minP_D-j]:                    # 计算k
        k = minP_D + j
    else:
        k = minP_D - j

    # sum(pi) + sum(di) = f(j,k)              (1)
    # sum(pi) - sum(di) = k - minP_D          (2)
    # sum(pi) = ((1) + (2)) // 2
    # sum(di) = ((1) - (2)) // 2
    prosecution = (k - minP_D + f[m][k]) // 2
    defence = (minP_D - k + f[m][k]) // 2

    print('Jury #{}'.format(cnt))
    print("Best jury has value {} for prosecution and value {} for
defence:".format(prosecution, defence))

    # 最终方案f(m, k)的最后一个人选记录在Path[m][k]中，
    # 那么从Path[m][k]出发，顺藤摸瓜就能找出方案f(m, k)的所有人选
    for i in range(1, m+1):
        result[i] = path[m - i + 1][k]
        a = result[i]
        b = p[a] - d[a]
        k = k - b

    ans = []
    for i in range(1, m+1):
        if result[i] != None:
            ans.append(result[i])

    ans = sorted(ans)   # 按人选编号从小到大排序，以便按要求输出
    print(' '.join(map(str, ans)))
```

1979: Help Jimmy

dp, http://cs101.openjudge.cn/practice/1979

"Help Jimmy" 是在下图所示的场景上完成的游戏：



场景中包括多个长度和高度各不相同的平台。地面是最低的平台，高度为零，长度无限。

Jimmy老鼠在时刻0从高于所有平台的某处开始下落，它的下落速度始终为1米/秒。当Jimmy落到某个平台上时，游戏者选择让它向左还是向右跑，它跑动的速度也是1米/秒。当Jimmy跑到平台的边缘时，开始继续下落。Jimmy每次下落的高度不能超过MAX米，不然就会摔死，游戏也会结束。

设计一个程序，计算Jimmy到底地面时可能的最早时间。

输入

第一行是测试数据的组数t（0 ≤ t ≤ 20）。每组测试数据的第一行是四个整数N，X，Y，MAX，用空格分隔。N是平台的数目（不包括地面），X和Y是Jimmy开始下落的位置的横竖坐标，MAX是一次下落的最大高度。接下来的N行每行描述一个平台，包括三个整数，X1[i]，X2[i]和H[i]。H[i]表示平台的高度，X1[i]和X2[i]表示平台左右端点的横坐标。1 ≤ N ≤ 1000，-20000 ≤ X, X1[i], X2[i] ≤ 20000，0 < H[i] < Y ≤ 20000（i = 1..N）。所有坐标的单位都是米。

Jimmy的大小和平台的厚度均忽略不计。如果Jimmy恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。测试数据保证问题一定有解。

输出

对输入的每组测试数据，输出一个整数，Jimmy到底地面时可能的最早时间。

样例输入

```
1
3 8 17 20
0 10 8
0 10 13
4 14 3
```

样例输出

```
23
```

来源：POJ Monthly--2004.05.15, CEOI 2000, POJ 1661, 程序设计实习2007

```
# OJ 1979, http://cs101.openjudge.cn/practice/1979/
# Top-Down. ref: https://www.cnblogs.com/zswbky/p/6792910.html
def dfs(i:int, a:int, x:int):
    if dp[i][a]!=-1:
        return dp[i][a]

    left = right = 10**8    # greater than 20000*2*1000

    flag = True
```

```
    for j in range(i+1, N+1):
        if p[i][2] - p[j][2] > MAX:
            flag = False
            break

        if p[j][0] <= x <= p[j][1]:
            left = dfs(j,0,p[j][0]) + p[i][2]-p[j][2] + x-p[j][0]
            right = dfs(j,1,p[j][1]) + p[i][2]-p[j][2] + p[j][1]-x
            flag = False
            break

    dp[i][a] = min(left, right)
    if flag and p[i][2]<=MAX:
        dp[i][a] = p[i][2]

    return dp[i][a]

for _ in range(int(input())):
    dp = [[-1,-1] for _ in range(1000+10)]
    p = []      #platform

    N,X,Y,MAX = map(int, input().split())
    p.append( [X,X,Y] )  # xi1, xi2, height
    for _ in range(N):
        p.append([int(x) for x in input().split()])
    #p.append([-2000,2000,0])

    p.sort(key = lambda x:-x[2])

    print(dfs(0,1,X))
```

1759: 最长上升子序列

dp, http://cs101.openjudge.cn/practice/1759

一个数的序列b_i，当$b_1 < b_2 < ... < b_S$的时候，我们称这个序列是上升的。对于给定的一个序列($a_1, a_2, ..., a_N$)，我们可以得到一些上升的子序列($a_{i1}, a_{i2}, ..., a_{iK}$)，这里$1 \le i_1 < i_2 < ... < i_K \le N$。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是4，比如子序列(1, 3, 5, 8).

你的任务，就是对于给定的序列，求出最长上升子序列的长度。

输入

输入的第一行是序列的长度N ($1 \le N \le 1000$)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000。

输出

最长上升子序列的长度。

样例输入

```
7
1 7 3 5 9 4 8
```

样例输出

```
4
```

来源：翻译自 Northeastern Europe 2002, Far-Eastern Subregion 的比赛试题

2020fall-cs101，王君宇。思路：和最大上升子序列和类似，都是需要双重循环的 dp，外循环确定每个元素 dp初始值为 1，内循环遍历小数进行转移方程。

```
input()
b = [int(x) for x in input().split()]

n = len(b)
dp = [1]*n

for i in range(n):
    for j in range(i):
        if b[j]<b[i]:
            dp[i] = max(dp[j]+1, dp[i])

print(max(dp))
```

2020fall-cs101，李博海

```
import bisect
n = int(input())
l = [int(x) for x in input().split()]
dp = [1e9]*n
for i in l:
    dp[bisect.bisect_left(dp, i)] = i
print(bisect.bisect_left(dp, 1e8))
```

2020fall-cs101，章斯岚。这里用i+1是为了防止a[0:i]可能是空列表导致runtime error。

与 OJ3532最大上升子序列和，一样，i改为1即可。

```
dp = [0]*(10000+2)
input()
for i in map(int, input().split()):
    dp[i+1] = max(dp[0:i+1]) + 1
print(max(dp))
```

1762: 数字三角形

dp/dfs, http://cs101.openjudge.cn/practice/1762

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5

      (图1)
```

图1给出了一个数字三角形。从三角形的顶部到底部有很多条不同的路径。对于每条路径，把路径上面的数加起来可以得到一个和，你的任务就是找到最大的和。

注意：路径上的每一步只能从一个数走到下一层上和它最近的左边的那个数或者右边的那个数。

输入

输入的是一行是一个整数N (1 < N ≤ 100)，给出三角形的行数。下面的N行给出数字三角形。数字三角形上的数的范围都在0和100之间。

输出

输出最大的和。

样例输入

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

样例输出

```
30
```

来源：翻译自 IOI 1994 的试题

要求dp实现一次，dp+dfs实现一次

2020fall-cs101，蓝克轩。

dp方法：用2d list记录三角形，然后从倒数第二排开始更改，将每个数改成下面两个可以加上的数的最大和，就能往上推出最大路径。

```
n = int(input())
tri = []   # triangle

for i in range(n):
    tri.append(list(map(int, input().split()))+[0 for j in range(n-i-1)])

for i in range(n-2,-1,-1):
    for j in range(i+1):
        tri[i][j] += max(tri[i+1][j], tri[i+1][j+1])

print(tri[0][0])
```

2020fall-cs101，李嘉钰。

解题思路：从三角形的最底层开始往上走，相邻的两个数字只有值较大者可以向上走，并且和上层的数字结合，值较小者被淘汰。最后，在三角形顶端的数字就是所有路径中的最大和。

```
n = int(input())
tri = []   # triangle

for i in range(n):
    tri.append([int(x) for x in input().split()])

for i in range(n-1, 0, -1):
    for x in range(len(tri[i-1])):
        tri[i-1][x] = max(tri[i-1][x] + tri[i][x], tri[i-1][x] + tri[i][x+1])

print(tri[0][0])
```

2020fall-cs101，孙湛劼

显然贪心是行不通的，所以考虑 dp。记$S_{i,j}$ 为以第 (i,j) 位置的元素结尾的路径的和的最大值，那么对一个 (i,j) 位置的数，他可以接在 $(i-1,j-1)$ 和 $(i-1,j)$ 的后面，那么就得到了 dp方程
$S_{i,j} = max\{S_{i-1,j}, Si-1,j\} + a_{i,j}$

```
N = int (input())
S = []
for i in range (N):
    S.append([int (x) for x in input().split()])

ans = [[S[0][0]]]
for i in range (1,N):
    SS = []
    for j in range (i+1):
        if j==0:
            SS.append(ans[i-1][0]+S[i][0])
        if 1<=j<=i-1:
            SS.append(max(ans[i-1][j-1],ans[i-1][j])+S[i][j])
        if j==i:
            SS.append(ans[i-1][i-1]+S[i][j])
    ans.append(SS.copy())

print (max(ans[N-1]))
```

dp+dfs方法

```
n = int(input())

node = []
node.append( [-1 for _ in range(n+2)] )
for _ in range(n):
    tmp = [int(_) for _ in input().split()]
    node.append([-1] +tmp + [-1]*(n - len(tmp)) + [-1])

node.append( [-1 for _ in range(n+2)] )
```

```
opt = [[0]*(n+2) for _ in range(n+2)]

def dp(i,j):
    if opt[i][j]>0:
        return opt[i][j]

    if node[i+1][j]==-1 or node[i+1][j+1]==-1 :
        return node[i][j]

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

ans = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        ans = max( ans, dp(i,j) )

print(ans)
```

2020fall-cs101，章斯岚。这种题目尽量不用二维数组可以使代码简化。直接开一个一维数组就可以解决存储问题（每一层存储后都可以丢弃）

```
a = []
dp = [0]*102
for i in range(int(input())):
    a.append(list(map(int,input().split())))
a = a[::-1]
for i in a:
    for j in range(len(i)):
        dp[j] = max(dp[j+1], dp[j]) +i[j]
print(dp[0])
```

90: 滑雪

dp/dfs, http://cs101.openjudge.cn/practice/90/

Michael喜欢滑雪百这并不奇怪， 因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael想知道载一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子

```
 1  2  3  4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。在上面的例子中，一条可滑行的滑坡为24-17-16-1。当然25-24-23-...-3-2-1更长。事实上，这是最长的一条。

输入

输入的第一行表示区域的行数R和列数C(1 ≤ R,C ≤ 100)。下面是R行，每行有C个整数，代表高度h，0≤h≤10000。

输出

输出最长区域的长度。

样例输入

```
5 5
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

样例输出

```
25
```

来源：USACO

思路：用一个2d list作为记录经过路径长(dp)，起始都为0。然后dfs，如果如果dp[i][j]大于0，就是已经检验过最长经过路径，所以return dp[i][j]；否则代表是第一次检验，检查上下左右的格子，如果高度小就可以走，返回max{dp[i][j], 四周的格的dfs + 1}。

2020fall-cs101，刘安澜。我对老师的代码进行了一个改进，因为本题说高度最大值不超过10000，所以保护圈元素取10001即可满足不超过边界，这样函数更加简洁。

2020fall-cs101，李元锋。正常思路是从第一个元素开始，判断滑雪最长距离，过一遍。但这里有很多重复计算的子问题，所以引入 dp函数，如果坐标周围有比自己低的坡道，判断滑哪条周围的坡道最长就滑哪条。然后直接遍历一遍二维数组即可，这里要添加保护圈以防万一滑出边界。

```
r, c = map(int, input().split())
node = []       # height of each element
node.append( [100001 for _ in range(c+2)] )
for _ in range(r):
    node.append([100001] +[int(_) for _ in input().split()] + [100001])

node.append( [100001 for _ in range(c+2)] )

dp = [[0]*(c+2) for _ in range(r+2)]
dx = [-1, 0, 1, 0]
dy = [ 0, 1, 0,-1]

def dfs(i,j):
    if dp[i][j]>0:
        return dp[i][j]

    for k in range(4):
        if node[i+dx[k]][j+dy[k]] < node[i][j]:
            dp[i][j] = max( dp[i][j], dfs(i+dx[k], j+dy[k])+1 )

    return dp[i][j]

ans = 0
for i in range(1, r+1):
```

```
    for j in range(1, c+1):
        ans = max( ans, dfs(i,j) )

print(ans+1)
```

2020fall-cs101，王康安。虽然开了个叫 dp的数组，但我感觉解法其实更像贪心。刚巧昨天在 oj上写了一道 Huffman树的题（18164：剪绳子），这里我的策略就很相似。把所有区域的高度和坐标存到一个栈里，排序，每次都把栈内高度最高的区域弹出去并处理。当前这个区域是到不了那些之前已经弹出的，比当前这个区域更高的区域的，也就是说，之前那些区域都不会影响这一步的策略，具有无后效性和最优子结构性质。状态转移方程非常直白就不多说了。上代码，毕竟没有按 dfs去写所以标了注释：

```
r,c = [int(x) for x in input().split()]
rgn = [[20000]*(c+2)]
for i in range(r):
    rgn.append([20000]+[int(x) for x in input().split()]+[20000])
rgn.append([20000]*(c+2))       #读入上保护圈方便处理边界情况

stack = []
for i in range(1,r+1):
    for j in range(1,c+1):
        stack.append([rgn[i][j], i, j])
stack.sort()                    #所有节点按从小到大排序

loc = [[1,0],[-1,0],[0,1],[0,-1]]   #方向数组
dp = [[1]*(c+2) for x in range(r+2)] #dp数组初始化全为1 上保护圈只是为了和rgn数组下标保持一致

while stack != []:
    temp = stack.pop()  #用temp数组保存出栈值该值也是当前栈内最大值
    for i in range(4):
        if rgn[temp[1]+loc[i][0]][temp[2]+loc[i][1]]<rgn[temp[1]][temp[2]]:
            #由于是栈内最大值此条件一定成立这里仅是为了处理保护圈情况
            dp[temp[1]+loc[i][0]][temp[2]+loc[i][1]] = \
            max(dp[temp[1]+loc[i][0]][temp[2]+loc[i][1]],dp[temp[1]][temp[2]]+1)
            #状态转移方程
out = 0
for i in range(1,r+1):
    out = max(out,max(dp[i]))
print(out)
```

1756: 八皇后

dfs/bfs, http://cs101.openjudge.cn/practice/1756

描述：会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将8个皇后放在棋盘上（有8 * 8个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。

对于某个满足要求的8皇后的摆放方法，定义一个皇后串a与之对应，即$a = b_1 b_2 \dots b_8$,其中b_i为相应摆法中第i行皇后所处的列数。已经知道8皇后问题一共有92组解（即92个不同的皇后串）。

给出一个数b，要求输出第b个串。串的比较是这样的：皇后串x置于皇后串y之前，当且仅当将x视为整数时比y小。

八皇后是一个古老的经典问题：**如何在一张国际象棋的棋盘上，摆放8个皇后，使其任意两个皇后互相不受攻击。**该问题由一位德国**国际象棋排局家 Max Bezzel** 于 1848年提出。严格来说，那个年代，还没有“德国”这个国家，彼时称作“普鲁士”。1850年，**Franz Nauck** 给出了第一个解，并将其扩展成了“ **n皇后** ”问题，即**在一张 n** x **n 的棋盘上，如何摆放 n 个皇后，使其两两互不攻击。**历史上，八皇后问题曾惊动过“数学王子”高斯(Gauss)，而且正是 Franz Nauck 写信找高斯请教的。

输入

第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数b(1 ≤ b ≤ 92)

输出

输出有n行，每行输出对应一个输入。输出应是一个正整数，是对应于b的皇后串。

样例输入

```
2
1
92
```

样例输出

```
15863724
84136275
```

八皇后思路：回溯法。从第一行第一列开始放置皇后，然后在每一行的不同列都放置，如果与前面不冲突就继续，有冲突则回到上一行继续下一个可能性。

详细步骤（参考 https://www.jianshu.com/p/deb028537af2）：
1）判断新的皇后是否与已经存在的皇后打架，加入是第一个则不用判断直接加入。

2）第二行新生成的皇后，然后与第一行判断是否打架，是的话，向右移动一个格子，否则添加上去。

3）第三行从左至右从第一个格子开始，判断是否与上面所有的皇后打架，是的话，向右移动一个格子，否则添加上去。

。。。

8）到第八行，新生成一个皇后，判断是否与上面所有的皇后打架，没有则添加。有则向右移动一个格子。当移动至一个不打架的格子，则一个解法已经生成。向后则寻找第二个方案，将第8行的皇后删除（for循环的下一列），新生成的皇后在刚才最后一个皇后的右边，为什么在右边呢？因为左边刚才已经判断过都失败了，所以新生成的在右边，然后在判断是否与上边的皇后打架。

9）当向右移动最后一个格子而且与上边的皇后打架，则删除掉此皇后（for循环的下一列），然后把上一行的皇后向右移动一个格子（for循环的下一列），第8行从左向右从0开始生成一个新的皇后。然后步骤8）.

10）直到第一行的皇后走到第一行的最后一个，第二行也找到最后一个格子的皇后，而且失败，则是所有解法都寻找完成。

```
ans = []
def queen_dfs(A, cur=0):                    #考虑放第cur行的皇后
    if cur == len(A):                       #如果已经放了n个皇后，一组新的解产生了
```

```
        ans.append(''.join([str(x+1) for x in A]))
        return

    for col in range(len(A)):       #将当前皇后逐一放置在不同的列，每列对应一组解
        for row in range(cur):      #逐一判定，与前面的皇后是否冲突
            if A[row] == col or abs(col - A[row]) == cur - row:
                break
        else:                       #若都不冲突
            A[cur] = col            #放置新皇后，在cur行，col列
            queen_dfs(A, cur+1)     #对下一个皇后位置进行递归

queen_dfs([None]*8)
for _ in range(int(input())):
    print(ans[int(input()) - 1])
```

2020fall-cs101，李博海。思路：bfs

```
def queen_bfs(A, cur=0):
    stack = [[None]*8, 0]
    while stack:
        cur = stack.pop()
        A = stack.pop()
        if cur == len(A):
            #print(A)
            ans.append(''.join([str(x+1) for x in A]))
            continue

        for col in range(len(A)):
            for row in range(cur):
                if A[row]==col or abs(col-A[row])==cur-row:
                    break
            else:
                A[cur] = col
                stack.append(A[:])
                stack.append(cur+1)

ans = []
queen_bfs([None]*8)
ans.sort()
for _ in range(int(input())):
    print(ans[int(input()) - 1])
```

c not in [a, a-2, a+2....] 不能在对角线上

```
#OJ eight queens

A = []
for a in range(1,9):
    for b in range(1,9):
        if b not in [a, a-1, a+1]:
            for c in range(1,9):
                if c not in [a,a-2,a+2, b,b-1,b+1]:
                    for d in range(1,9):
                        if d not in [a,a-3,a+3,b,b-2,b+2,c,c-1,c+1]:
                            for e in range(1,9):
```

```
                                if e not in[a,a-4,a+4,b,b-3,b+3,c,c-2,c+2,d,d-
1,d+1]:
                                    for f in range(1,9):
                                        if f not in [a,a-5,a+5,b,b-4,b+4,c,c-
3,c+3,d,d-2,d+2,e,e-1,e+1]:
                                            for g in range(1,9):
                                                if g not in [a,a-6,a+6,b,b-
5,b+5,c,c-4,c+4,d,d-3,d+3,e,e-2,e+2,f,f-1,f+1]:
                                                    for h in range(1,9):
                                                        if h not in [a,a-
7,a+7,b,b-6,b+6,c,c-5,c+5,d,d-4,d+4,e,e-3,e+3,f,f-2,f+2,g,g-1,g+1]:
                                                            A.append(''.join(
map(str, [a,b,c,d,e,f,g,h] )))
for _ in range(int(input())):
    print(A[int(input()) - 1])
```

2020fall-cs101，蓝克轩。

思路：本的想法是对角线一个是差不变，一个是和不变，可是做起来太复杂了，看了解答才发现可以直接利用斜率的绝对值=1，即绝对值(列差)=行差。另外，还发现就是由于棋盘是对称的，可以只生成一半的解，再用99999999减去前半部的解求得后半部的解。

```
ans=[[] for j in range(46)]
num=0
def queen(a,step):
    global num
    if num>45:
        return 0
    if step==8:
        ans[num]=a[:]
        num+=1
        return 0
    for col in range(8):
        a[step]=col
        safe=True
        for before in range(step):
            if a[before]==col or abs(col-a[before])==step-before:
                safe=False
                break
        if safe:
            queen(a,step+1)

queen([None for i in range(8)],0)
for _ in range(int(input())):
    a=int(input())-1
    print("".join((str(x+1) for x in ans[a]) if a<46 else (str(8-x) for x in
ans[91-a])))
```

3532: 最大上升子序列和

dp, http://cs101.openjudge.cn/practice/3532/

一个数的序列bi，当$b_1 < b_2 < ... < b_S$的时候，我们称这个序列是上升的。对于给定的一个序列($a_1, a_2, ...,a_N$)，我们可以得到一些上升的子序列($a_{i1}, a_{i2}, ..., a_{iK}$)，这里1 <= i1 < i2 < ... < iK <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中序列和最大为18，为子序列(1, 3, 5, 9)的和.

你的任务，就是对于给定的序列，求出最大上升子序列和。注意，最长的上升子序列的和不一定是最大的，比如序列(100, 1, 2, 3)的最大上升子序列和为100，而最长上升子序列为(1, 2, 3)

输入

输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000（可能重复）。

输出

最大上升子序列和

样例输入

```
7
1 7 3 5 9 4 8
```

样例输出

```
18
```

思路：从第一个数开始逐次递推，考虑第 i个数的情况时，再从第一个数开始逐个检验，如果第 i个数大于前 i个数中的第 j个数，那么将前 j个数的最大上升子序列和再加上第 i个数，即构成前 i个数上升子序列和的一种情况，再取这些情况中的最大值，即得到前 i个数的最大上升子序列和。最后依次递推，即可得到整个序列的最大上升子序列和。

主要思路就是记录把每个数作为序列最后一位时的序列和，取 max。即，以每一项为末项的最大上升子序列和。

2020fall-cs101，邹思清。感觉跟我之前做的dp不太一样。之前的dp大多是计算n位之前满足的答案，而这道题使用的递推公式也不仅仅是相邻几项，而且还使用了max，做的时候没有想到，又学到新方法了。

```
input()
b = [int(x) for x in input().split()]

n = len(b)
dp = [0]*n

for i in range(n):
    dp[i] = b[i]
    for j in range(i):
        if b[j]<b[i]:
            dp[i] = max(dp[j]+b[i], dp[i])

print(max(dp))
```

2020fall-cs101，李东航

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

print(max(dp))
```

2020fall-cs101，章斯岚。这里用i+1是为了防止a[0:i]可能是空列表导致runtime error

```
dp = [0]*(10000+2)
input()
for i in map(int, input().split()):
    dp[i+1] = max(dp[0:i+1]) + i
print(max(dp))
```

18164: 剪绳子

greedy/huffman, http://cs101.openjudge.cn/practice/18164/

小张要将一根长度为L的绳子剪成N段。准备剪的绳子的长度为L1,L2,L3...,LN，未剪的绳子长度恰好为剪后所有绳子长度的和。

每次剪断绳子时，需要的开销是此段绳子的长度。

比如，长度为10的绳子要剪成长度为2,3,5的三段绳子。长度为10的绳子切成5和5的两段绳子时，开销为10。再将5切成长度为2和3的绳子，开销为5。因此总开销为15。

请按照目标要求将绳子剪完最小的开销时多少。

已知，1<=N <= 20000，0<=Li<= 50000

输入

第一行：N，将绳子剪成的段数。
第二行：准备剪成的各段绳子的长度。

输出

最小开销

样例输入

```
3
2 3 5
```

样例输出

```
15
```

来源：cs101-2017 期末机考备选

```
# OJ18164
import sys
try: fin = open('test.in','r').readline
except: fin = sys.stdin.readline

n = int(fin())
import heapq
a = list(map(int, fin().split()))
heapq.heapify(a)
ans = 0
for i in range(n-1):
    x = heapq.heappop(a)
    y = heapq.heappop(a)
    z = x + y
    heapq.heappush(a, z)
    ans += z
print(ans)
```

1017: 装箱问题

greedy, http://cs101.openjudge.cn/practice/1017/

一个工厂制造的产品形状都是长方体，它们的高度都是h，长和宽都相等，一共有六个型号，他们的长宽分别为1*1, 2*2, 3*3, 4*4, 5*5, 6*6。这些产品通常使用一个 6*6*h 的长方体包裹包装然后邮寄给客户。因为邮费很贵，所以工厂要想方设法的减小每个订单运送时的包裹数量。他们很需要有一个好的程序帮他们解决这个问题从而节省费用。现在这个程序由你来设计。

输入：输入文件包括几行，每一行代表一个订单。每个订单里的一行包括六个整数，中间用空格隔开，分别为*1*1至66*这六种产品的数量。输入文件将以6个0组成的一行结尾。

输出：除了输入的最后一行6个0以外，输入文件里每一行对应着输出文件的一行，每一行输出一个整数代表对应的订单所需的最小包裹数。

解题思路：4*4, 5*5, 6*6这三种的处理方式较简单，就是每一个箱子至多只能有其中1个，根据他们的数量添加箱子，再用2*2和1*1填补。1*1, 2*2, 3*3这些就需要额外分情况讨论，若有剩余的3*3,每4个3*3可以填满一个箱子，剩下的3*3用2*2和1*1填补装箱。剩余的2*2，每9个可以填满一个箱子，剩下的与1*1一起装箱。最后每36个1*1可以填满一个箱子，剩下的为一箱子。

样例输入

```
0 0 4 0 0 1
7 5 1 0 0 0
0 0 0 0 0 0
```

样例输出

```
2
1
```

来源：Central Europe 1996

直接用总数把bcdef占的位置都减掉就可以了，思路就清晰起来了。

```
import math
rest = [0,5,3,1]

while True:
    a,b,c,d,e,f = map(int,input().split())
    if a + b + c + d + e + f == 0:
        break
    boxes = d + e + f           #装4*4, 5*5, 6*6
    boxes += math.ceil(c/4)     #填3*3
    spaceforb = 5*d + rest[c%4] #能和4*4 3*3 一起放的2*2
    if b > spaceforb:
        boxes += math.ceil((b - spaceforb)/9)
    spacefora = boxes*36 - (36*f + 25*e + 16*d + 9*c + 4*b)      #和其他箱子一起的填
的1*1

    if a > spacefora:
        boxes += math.ceil((a - spacefora)/36)
    print(boxes)
```

16528: 充实的寒假生活

greedy/dp, http://cs101.openjudge.cn/practice/16528/, cs10117 Final Exam

寒假马上就要到了，龙傲天同学获得了从第0天开始到第60天结束为期61天超长寒假，他想要尽可能丰富自己的寒假生活。
现提供若干个活动的起止时间，请计算龙同学这个寒假至多可以参加多少个活动？注意所参加的活动不能有任何时间上的重叠，在第x天结束的活动和在第x天开始的活动不可同时选择。

输入

第一行为整数n，代表接下来输入的活动个数(n < 10000)
紧接着的n行，每一行都有两个整数，第一个整数代表活动的开始时间，第二个整数代表全结束时间

输出

输出至多参加的活动个数

2020fall-cs101-苏荣薰，Greedy解题思路：只要按照结束时间排序，然后参加第一个活动，接下来的活动能参加就参加。因为这样的话不会使结果更坏，能参加尽量多的活动。假设最优解并不包括第一个结束的活动，那么最优解中第一个活动必然可以替换成第一个结束的活动，选择参加第一个结束的活动不会使结果更坏。

解题思路：随时间减少，可参加的活动数量也减少，因此应当以结束时间排序（如果选最先开始的，最短时间的活动都会出现问题）。排序后第一个活动一定能参加，使用贪心算法，选择开始时间晚于上一次结束时间的事件，次数加 1。

样例输入

```
5
0 0
1 1
2 2
3 3
4 4
```

样例输出

```
5
```

来源：cs10117 Final Exam

要求dp实现一次，greedy实现一次。这个题目，初看用不了dp，数据预处理一下就可以了，因为dp不排斥数据预处理。

Greedy

```
n=int(input())
m=[[int(x) for x in input().split()] for i in range(n)]
for i in range(n):
    m[i][0],m[i][1]=m[i][1],m[i][0]
m.sort()
y=1
a=m[0][0]
for i in range(n-1):
    if m[i+1][1]>a:
        y+=1
        a=m[i+1][0]
print(y)
```

Greedy

```
n = int(input())
s = [[int(x) for x in input().split()] for _ in range(n)]
s.sort(key = lambda x:x[1])

m = s[0][1]
a = 1
for i in range(1,n):
    if s[i][0] > m:
        a += 1
        m = s[i][1]

print(a)
```

dp解法，“最大上升子序列和”思路

```
n = int(input())
act = [None]*61
for _ in range(n):
    s,e = map(int, input().split())
    if act[s]==None:
```

```
        act[s] = e
    elif act[s] > e:
        act[s] = e

dp = [1]*61
for i in range(61):
    if act[i]!=None:
        for j in range(i):
            if act[j]!=None and act[j]<i:
                dp[i] = max(dp[i], dp[j]+1)
print(max(dp))
```

dp，就是把不可能的情况剔除了，从头扫描一遍。字典中保存的是，如果结束时间一样，保留开始时间晚的。另外，单独处理开始时间是0的活动。

体会：DP类题目，除了递推公式，开始前的的数据预处理还是挺重要的。例如：OJ16528 充实的寒假生活，字典中保存的是，如果结束时间一样，保留开始时间晚的。

2020fall-cs101，蔡清远。本题dp应该有两个思路：一是以时间作为状态转移变量，dp[t]表示前t天内所能参加的最大活动数，时间复杂度O(max(T))，而这样做需要巧妙的数据预处理，即构建一个活动结束时间与最晚开始时间的映射表（此处包含一步贪心，开始时间晚使任务覆盖时间更短，不会使结果更坏）。二是以考虑的活动数作为状态转移变量，在对活动结束时间排序后，dp[i]代表考虑前i个活动时，所能参加的最大活动数，时间复杂度O(n^2)，即使考虑到可以用二分搜索简化，时间复杂度也为O(nlogn)。因此，就本题数据范围而言，无疑应当使用第一种方法。

```
n = int(input())
event = [-1] * 61
for x in range(n):
    start, end = map(int, input().split())
    event[end] = max(event[end], start)

dp = [0]*61
for t in range(61):
    if event[t] == -1:
        dp[t] = max(dp[t], dp[t-1])
    else:
        dp[t] = max(dp[t], dp[t-1], dp[event[t] -1] +1) #不取这次活动，取这次活动。类似背包

print(dp[60])
```

```
n = int(input())
activity = {}
for _ in range(n):
    s,e = map(int, input().split())
    if e not in activity:
        activity[e] = s
    elif activity[e] < s:
        activity[e] = s

ans = end = 0
for i in range(0,61):
```

```
    if (ans==0) and (i in activity) and (activity[i]==0):
        ans = 1
        end = i
        continue

    if (i in activity) and (activity[i]>end):
        ans += + 1
        end = i

print(ans)
```

2020fall-cs101，赵春源代码，安磊解说思路。

借鉴了赵春源同学的代码。res[i]代表截止到前 i天参加的活动数。ans[i]代表第 i个活动参加时可以参加的活动总数。假设某个活动开始的时间为 k，则它的前 k天举办的最大活动数再加上 1即为考虑了该活动之后的活动总数。如果这个活动是从第零天开始的，那么只能参加这一个活动，ans即等于 1。如此递推即可。需要注意的是在参加了第 i个活动之后，必须更新第 i个活动截止日期之前的活动总数。

```
n = int(input())
lis = []
for i in range(n):
    a,b = map(int,input().split())
    lis.append((a,b))
lis.sort(key = lambda x:x[0])
res = [0] * 61
ans = [0] * n
for i in range(n):
    if lis[i][0] > 0:
        ans[i] = max(res[:lis[i][0]]) + 1
    else:
        ans[i] = 1

    res[lis[i][1]] = max(ans[i],res[lis[i][1]])

print(max(ans))
```

16531: 上机考试

matrix, http://cs101.openjudge.cn/practice/16531/ , cs10117 Final Exam

一个班的同学在M行*N列的机房考试，给出学生的座位分布和每个人的做题情况，统计做题情况与周围（上下左右）任一同学相同的学生人数。
另外，由于考试的优秀率不能超过40%，请统计这个班的优秀人数（可能为0，相同分数的同学必须要么全是优秀，要么全不是优秀）。

输入

第一行为两个整数，M和N
接下来M行每行N个整数，代表对应学生的编号，编号各不相同，范围由0到M*N-1
接下来M*N行，顺序给出编号由0到n-1的同学的做题情况，1代表做对，0代表做错。

输出

两个整数，以空格分隔；分别代表“与周围任一同学做题情况相同的学生人数”和“班级优秀的人数”

样例输入

```
样例一输入
2 5
0 1 2 3 4
5 6 7 8 9
1 1 1
1 0 1
0 0 0
0 0 1
0 0 0
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1

样例一输出
6 0

样例一解释：
编号为0 5 6 7 8 9的同学做题情况与周围同学相同，因此第一个整数是6。
全对的同学人数已经超过了40%，因此优秀的人数为0
```

样例输出

```
样例二输入：
1 3
1 0 2
0 1 0 0
0 0 0 0
0 0 0 0

样例二输出：
0 1

样例二解释：
并不存在与相邻同学做题情况相同的同学，并且做对一题的同学比例小于40%，因此有一人优秀
```

提示：M*N行做题情况可能为空。因为如果所有学生做题过程中，一直没有提交，则空。

来源：cs10117 Final Exam

代码参考：2018fall-cs101, 李子安；2020fall-cs101, 周晋飞

```
m,n = map(int,input().split())
s = [[-1]*(n+2)]
num = s + [[-1]+ [int(x) for x in input().split()]+[-1] for _ in range(m)] + s
grade = [[int(x) for x in input().split()] for i in range(m*n)]

ans = 0
for i in range(1, m+1):
    for j in range(1, n+1):
        a = num[i][j]
        for b in [num[i-1][j], num[i+1][j], num[i][j+1], num[i][j-1]]:
            if b!=-1 and grade[b]==grade[a]:
```

```
                ans += 1
                break

gsum = [sum(i) for i in grade]
gsum.sort(reverse = True)
c = [i > gsum[int(m * n *0.4)] for i in gsum]
print(ans, c.count(True))
```

```
# 2018fall-cs101, 李子安
import math
m,n = [int(x) for x in input().split()]
s = [[-1]*(n+2)]
l = s+[[-1]+ [int(x) for x in input().split()]+[-1] for i in range(m)]+s

c = 0
t = [[int(x) for x in input().split()] for i in range(m*n)]

for i in range(1,m+1):
    for j in range(1,n+1):
        a = t[l[i][j]]
        b = [l[i-1][j],l[i+1][j],l[i][j+1],l[i][j-1]]
        for k in b:
            if k!=-1 and t[k]==a:
                c += 1
                break

y = []
for i in range(m*n):
    x = sum(t[i])
    y.append(x)
y = sorted(y,reverse=True)

bound = math.floor(0.4*m*n)
u = 0
if m*n<=2:
    u = 0
elif m*n>2 and y[bound-1]!=y[bound]:
    u = bound
elif m*n>2 and y[bound-1]==y[bound]:
    for i in y:
        if i>y[bound-1]:
            u += 1

print(c,u)
```

Notes

1: 一维数组可以浅拷贝，二维数组复制需要深拷贝

需要清楚，浅拷贝shallow copy会在什么情况下出现问题，二维数组复制需要深拷贝deepcopy。举例：

```
```

```
row =3; col = 4

# 1D array
A1 = [0] * col
A1_copy = A1[:]
A1_copy[1] = 1

# two methods to output
print("#1D array")
print(A1)
print(A1_copy)
print(' '.join(map(str, A1_copy)), sep='\n')
print()

# 2D array
matrix = [[-1] * col for _ in range(row)]

# two methods to output
print("#2D array")
print("#method1 output")
for i in range(row):
    print(matrix[i])
print()

# ' '.join(row) for row in matrix) returns a string for every row,
#   e.g. A B when row is ["A", "B"].
#
# *(' '.join(row) for row in matrix), sep='\n') unpacks the generator
#   returning the sequence 'A B', 'C D', so that print('A B', 'C D', sep='\n')
#   is called for the example matrix given.
print("#method2 output")
print( *(' '.join(map(str, row)) for row in matrix), sep='\n')
print()

print("#change one element")
matrix[0][1] = 2
for i in range(row):
    print(matrix[i])
print()

import copy
mx_copy = copy.deepcopy(matrix)     # matrix copied
print( *(' '.join(map(str, row)) for row in mx_copy), sep='\n')
print()
```

程序运行，输出结果

```
#1D array
[0, 0, 0, 0]
[0, 1, 0, 0]
0 1 0 0

#2D array
#method1 output
[-1, -1, -1, -1]
```

```
[-1, -1, -1, -1]
[-1, -1, -1, -1]

#method2 output
-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1

#change one element
[-1, 2, -1, -1]
[-1, -1, -1, -1]
[-1, -1, -1, -1]

-1 2 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1
```

2: 变量名起的有意义

可以按照题面描述中用到的符号，起变量名，或者起有意义的变量名。

避免程序写长后，混淆使用变量名。尤其是二维数组的行列变量，注意不要混淆

3: 数据结构dict, set时间复杂度是O(1)

dict, set的时间复杂度低，能用的时候优先使用。

4: 通常使用split()，而不是split(' ')

对于codeforces，或者openjudge，接收输入，通常不需要自己指定分隔符。

5: 函数默认参数

可以参考 OJ 1756 八皇后的递归函数，第一次调用，使用了默认参数。def queen(A, cur=0):

6: 精确到小数点后9位

print('{:.9f}'.format(x))

7. enumerate函数

The basic syntax is is *enumerate(sequence, start=0)*

The output object includes a counter like so: (0, thing[0]), (1, thing[1]), (2, thing[2]), !

8. 浮点数判断相等

通常是 return abs(f1 - f2) <= allowed_error 。例如：OJ12065方程求解，可以allowed_error = 1e-12

References

[1]. Student assignments, fall 2020.

[2]. https://csrgxtu.github.io/2015/03/20/Writing-Mathematic-Fomulars-in-Markdown/

[3]. 数据类型操作时间复杂度，https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt

[4]. 《算法基础与在线实践》郭炜等编著，2017年。