描述

给定一个由不同的小写字母组成的字符串，输出这个字符串的所有全排列。 我们假设对于小写字母有'a' < 'b' < ... < 'y' < 'z'，而且给定的字符串中的字母已经按照从小到大的顺序排列。

输入

输出这个字符串的所有排列方式，每行一个排列。要求字母序比较小的排列在前面。字母序如下定义：

已知S = s1s2...sk , T = t1t2...tk，则S < T 等价于，存在p (1 <= p <= k)，使得 s1 = t1, s2 = t2, ..., sp - 1 = tp - 1, sp < tp成立。

样例输入

```
abc
```

样例输出

```
abc
acb
bac
bca
cab
cba
```

```python
s=input()

def arrange(s):
    if len(s)==1:
        return [s]
    result=[]
    for i in range(len(s)):
        for sub_arrange in arrange(s[:i]+s[i+1:]):
            result.append(s[i]+sub_arrange)
    return result

for a in arrange(s):
    print(a)
```

一个数的序列bi，当b1 < b2 < ... < bS的时候，我们称这个序列是上升的。对于给定的一个序列(a1, a2, ..., aN)，我们可以得到一些上升的子序列(ai1, ai2, ..., aiK)，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 <= 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。

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
```

```python
n = int(input())
nums = list(map(int, input().split()))
tails = []
for num in nums:
    # 二分查找插入位置
    left, right = 0, len(tails)
    while left < right:
        mid = (left + right) // 2
        if tails[mid] < num:
            left = mid + 1
        else:
            right = mid
    if left == len(tails):
        tails.append(num)
    else:
        tails[left] = num  # 替换更小的末尾元素
print(len(tails))
```

描述

```
        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
```

```python
n = int(input())
triangle = []
for _ in range(n):
    row = list(map(int, input().split()))
    triangle.append(row)

dp = [0] * n
dp[0] = triangle[0][0]

for i in range(1, n):
    for j in range(i, -1, -1):
        current = triangle[i][j]
        if j == 0:
            dp[j] += current
        elif j == i:
            dp[j] = dp[j-1] + current
        else:
            dp[j] = max(dp[j-1], dp[j]) + current

print(max(dp))
```

描述

已知矩阵的大小定义为矩阵中所有元素的和。给定一个矩阵，你的任务是找到最大的非空(大小至少是1 * 1)子矩阵。

比如，如下4 * 4的矩阵

```
0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2
```

的最大子矩阵是

```
9 2
-4 1
-1 8
```

这个子矩阵的大小是15。

输入

输入是一个N * N的矩阵。输入的第一行给出N (0 < N <= 100)。再后面的若干行中，依次（首先从左到右给出第一行的N个整数，再从左到右给出第二行的N个整数……）给出矩阵中的N^2个整数，整数之间由空白字符分隔（空格或者空行）。已知矩阵中整数的范围都在[-127, 127]。

输出

输出最大子矩阵的大小。

样例输入

```
4
0 -2 -7 0 9 2 -6 2
-4 1 -4  1 -1
8  0 -2
```

样例输出

```
15
```

```python
import sys

def find_sum(line,n):
    dp=[0]*(n+1)
    for i in range(1,n+1):
        dp[i]=max(dp[i-1]+line[i-1],line[i-1])
    return max(dp)

def calculate_column(height,matrix,n):#height*k(k
    global max_sum
    for j in range(n - height):  # j表示取得是哪几
        column = []
        for i in range(0,n):#如tie可考虑用列前缀和
            column.append(sum(line[i] for line in
        summ=find_sum(column,n)
        if summ>max_sum:
            max_sum=summ

def main():
    global max_sum
    max_sum=-10000000
    n=int(input())
    inp=sys.stdin.read().strip()
    matrix=[[0]*n for _ in range(n)]
    i=j=0
    for strs in inp.split():
        matrix[i][j]=int(strs)
        j+=1
        if j==n:
            j-=n
            i+=1
    for h in range(n):
        calculate_column(h, matrix, n)
    print(max_sum)

if __name__=='__main__':
    main()
```

描述

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是称为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗?

输入

输入的第一行有两个整数T（1 <= T <= 1000）和M（1 <= M <= 100），T代表总共能够用来采药的时间，M代表山洞里的草药的数目。接下来的M行每行包括两个在1到100之间（包括1和100）的的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出

输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

样例输入

70 3 71 100 69 1 1 2

样例输出

3

```python
def herb(T, M, herbs):
    dp =[[0]*(T + 1) for i in range(M + 1)]
    for i in range(1, M + 1):
        time, value = herbs[i - 1]
        for j in range(T + 1):
            # 不采
            dp[i][j] = dp[i - 1][j]
            # 采
            if j >= time:
                dp[i][j] = max(dp[i][j], dp[i - 1][j - time] + value)
    return dp[M][T]

T, M = map(int, input().split())  # 总时间，草药数量
herbs=[]
for _ in range(M):
    time,value=map(int,input().split())
    herbs.append((time,value))

print(herb(T, M, herbs))
```

描述

木材厂有一些原木，现在想把这些木头切割成一些长度相同的小段木头，需要得到的小段的数目是给定了。当然，我们希望得到的小段越长越好，你的任务是计算能够得到的小段木头的最大长度。

木头长度的单位是厘米。原木的长度都是正整数，我们要求切割得到的小段木头的长度也要求是正整数。

输入

第一行是两个正整数N和K(1 ≤ N ≤ 10000, 1 ≤ K ≤ 10000)，N是原木的数目，K是需要得到的小段的数目。 接下来的N行，每行有一个1到10000之间的正整数，表示一根原木的长度

输出

输出能够切割得到的小段的最大长度。如果连1厘米长的小段都切不出来，输出"0"。

样例输入

```
3 7
232
124
456
```

样例输出

```
114
```

```python
def is_valid(length, woods, k):
    # 判断是否能用给定的长度切出至少 k 段
    return sum(wood // length for wood in woods) >= k

def max_piece_length(woods, k):
    left, right = 1, max(woods)
    best = 0

    while left <= right:
        mid = (left + right) // 2
        if is_valid(mid, woods, k):
            best = mid  # mid 可行，尝试更大
            left = mid + 1
        else:
            right = mid - 1

    return best

# 读取输入
N, K = map(int, input().split())
woods = [int(input()) for _ in range(N)]

# 特殊情况处理：无论如何都切不出K段
if sum(wood for wood in woods) < K:
    print(0)
else:
    print(max_piece_length(woods, K))
```

输入

你的任务是写一个程序读取一些测试数据。每组测试数据表示一个计算机的文件结构。每组测试数据以'*'结尾，而所有合理的输入数据以'#'结尾。一组测试数据包括一些文件和目录的名字（虽然在输入中我们没有给出，但是我们总假设ROOT目录是最外层的目录）。在输入中,以']'表示一个目录的内容的结束。目录名字的第一个字母是'd'，文件名字的第一个字母是'f'。文件名可能有扩展名也可能没有（比如fmyfile.dat和fmyfile）。文件和目录的名字中都不包括空格,长度都不超过30。一个目录下的子目录个数和文件个数之和不超过30。

输出

在显示一个目录中内容的时候，先显示其中的子目录（如果有的话），然后再显示文件（如果有的话）。文件要求按照名字的字母表的顺序显示（目录不用按照名字的字母表顺序显示，只需要按照目录出现的先后显示）。对每一组测试数据，我们要先输出"DATA SET x:"，这里x是测试数据的编号（从1开始）。在两组测试数据之间要输出一个空行来隔开。

你需要注意的是，我们使用一个'|'和5个空格来表示出缩排的层次。

样例输入

```
file1
file2
dir3
dir2
file1
file2
]
]
file4
dir1
]
file3
*
file1
file2
]
]
file4
dir1
]
file3
*
file2
file1
*
#
```

样例输出

```
DATA SET 1:
ROOT
|     dir3
|     |     dir2
|     |     file1
|     |     file2
|     dir1
file1
file2
file3
file4

DATA SET 2:
ROOT
file1
file2
```

```python
class Directory:
    def __init__(self, name):
        self.name = name
        self.subdirs = []
        self.files = []

def process_test_case(lines, case_number):
    root = Directory('ROOT')
    stack = [root]
    current_dir = root

    for line in lines:
        line = line.strip()
        if not line:
            continue
        if line == ']':
            if len(stack) > 1:
                stack.pop()
                current_dir = stack[-1]
        elif line.startswith('d'):
            new_dir = Directory(line)
            current_dir.subdirs.append(new_dir)
            stack.append(new_dir)
            current_dir = new_dir
        elif line.startswith('f'):
            current_dir.files.append(line)

    print(f"DATA SET {case_number}:")
    print("ROOT")
    print_directory(root, 0)

def print_directory(directory, level):
    for subdir in directory.subdirs:
        indent = '|     ' * (level + 1)
        print(f"{indent}{subdir.name}")
        print_directory(subdir, level + 1)
    for filename in sorted(directory.files):
        indent = '|     ' * level
        print(f"{indent}{filename}")

import sys

def main():
    input_lines = []
    for line in sys.stdin:
        line = line.strip()
        if line == '#':
            break
        input_lines.append(line)

    test_cases = []
    current_case = []
    for line in input_lines:
        if line == '*':
            test_cases.append(current_case)
            current_case = []
        else:
            current_case.append(line)

    for i, case in enumerate(test_cases, 1):
        process_test_case(case, i)
        if i < len(test_cases):
            print()

if __name__ == "__main__":
    main()
```

描述

给出4个小于10个正整数，你可以使用加减乘除4种运算以及括号把这4个数连接起来得到一个表达式。现在的问题是，是否存在一种方式使得得到的表达式的结果等于24。

这里加减乘除以及括号的运算结果和运算的优先级跟我们平常的定义一致（这里的除法定义是实数除法）。

比如，对于5，5，5，1，我们知道5 * (5 – 1 / 5) = 24，因此可以得到24。又比如，对于1，1，4，2，我们怎么都不能得到24。

输入

输入数据包括多行，每行给出一组测试数据，包括4个小于10个正整数。最后一组测试数据中包括4个0，表示输入的结束，这组数据不用处理。

输出

对于每一组测试数据，输出一行，如果可以得到24，输出“YES”；否则，输出“NO”。

样例输入

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

样例输出

```
YES
NO
```

```python
def count24(numl:list):
    if len(numl) == 1: # 如果列表长度为1，那么比较列表中唯一一个元素是否等于24
        if abs(numl[0] - 24) <= 0.0001:
            return True
        else:
            return False # 我们需要避免浮点误差对计算结果造成影响
    for i in range(len(numl) - 1): # 如果列表长度超过1，那么在列表中选择两个数字进行运算
        for j in range(i + 1, len(numl)):
            n1 = numl[i]
            n2 = numl[j]
            can = [n1 + n2, n1 - n2, n2 - n1, n1 * n2]
            if abs(n2 - 0) > 0.0001:
                can.append(n1 / n2)
            if abs(n1 - 0) > 0.0001:
                can.append(n2 / n1) # can列表中存储了这两个选定的数字可以进行的二元运算方式
            for v in can:
                newl = [v] # 构造一个新的列表，先放进计算结果
                for _ in range(len(numl)):
                    if _ != i and _ != j: # 然后放进刚刚没有被计算的数字
                        newl.append(numl[_])
                if count24(newl): # 递归来考虑这个长度少了一的列表能否算出24，如果能，那么这个也能
                    return True
    return False # 如果这个列表的所有方法都试过了没有一个能算出24，那返回不能算出24

while True:
    numl = list(map(int, input().split()))
    if numl == [0, 0, 0, 0]:
        exit(0) # 输入数据并且判断是否结束
    if count24(numl):
        print('YES')
    else:
        print('NO')
```

… …

如上图所示，由正整数1，2，3……组成了一颗二叉树。我们已知这个二叉树的最后一个结点是n。现在的问题是，结点m所在的子树中一共包括多少个结点。

比如，n = 12，m = 3那么上图中的结点13，14，15以及后面的结点都是不存在的，结点m所在子树中包括的结点有3，6，7，12，因此结点m的所在子树中共有4个结点。

输入：

输入数据包括多行，每行给出一组测试数据，包括两个整数m，n (1 <= m <= n <= 1000000000)。最后一组测试数据中包括两个0，表示输入的结束，这组数据不用处理

输出：

对于每一组测试数据，输出一行，该行包含一个整数，给出结点m所在子树中包括的结点的数目。

样例输入

```
3 12
0 0
```

样例输出

```
4
```

```python
def count_nodes(m, n):
    count = 0
    level = 0
    while m <= n:
        nodes_in_level = min(2**level, n - m + 1)
        count += nodes_in_level
        m *= 2
        level += 1
    return count

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

形如$a^3= b^3 + c^3 + d^3$的等式被称为完美立方等式。例如$12^3= 6^3 + 8^3 + 10^3$ 。编写一个程序，对任给的正整数N (N≤100)，寻找所有的四元组(a, b, c, d)，使得$a^3 = b^3 + c^3 + d^3$，其中a,b,c,d 大于 1, 小于等于N，且b<=c<=d。

输入

一个正整数N (N≤100)。

输出

每行输出一个完美立方。输出格式为： Cube = a, Triple = (b,c,d) 其中a,b,c,d所在位置分别用实际求出四元组值代入。

请按照a的值，从小到大依次输出。当两个完美立方等式中a的值相同，则b值小的优先输出、仍相同则c值小的优先输出、再相同则d值小的先输出。

样例输入

```
24
```

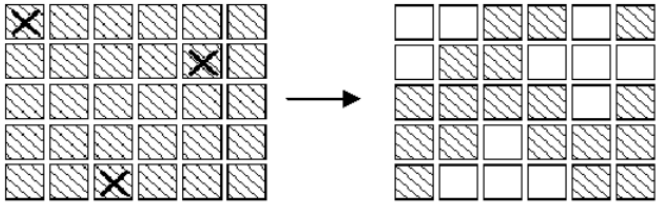
样例输出

```
Cube = 6, Triple = (3,4,5)
Cube = 12, Triple = (6,8,10)
Cube = 18, Triple = (2,12,16)
Cube = 18, Triple = (9,12,15)
Cube = 19, Triple = (3,10,18)
Cube = 20, Triple = (7,14,17)
Cube = 24, Triple = (12,16,20)
```

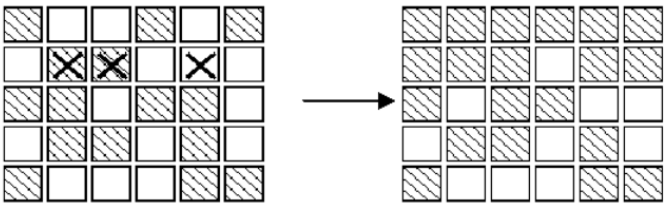
来源

```python
n=int(input())
dict={i:i**3 for i in range(2,n+1)}
for a in range(6,n+1):
    for b in range(2,a-2):
        for c in range(b+1,a-1):
            for d in range(c+1,a):
                if dict[a]==dict[b]+dict[c]+dict[d]:
                    print(f'Cube = {a}, Triple = ({b},{c},{d})')
```

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
```

```python
import copy

# 输入
start = [input().split() for _ in range(5)]


# 翻转函数
def flip(grid, r, c):
    for dr, dc in [(0, 0), (0, 1), (0, -1), (1, 0), (-1, 0)]:
        nr, nc = r + dr, c + dc
        if 0 <= nr < 5 and 0 <= nc < 6:
            grid[nr][nc] = '0' if grid[nr][nc] == '1' else '1'

# 解决函数
def solve(start):
    for mask in range(2**6):  # 枚举第一行的按法
        state = copy.deepcopy(start)
        press = [['0'] * 6 for _ in range(5)]

        # 按第一行
        for j in range(6):
            if (mask >> j) & 1: #第一行j列按了
                press[0][j] = '1'
                flip(state, 0, j)

        # 从第二行开始逐行决定按法
        for i in range(1, 5):
            for j in range(6):
                if state[i - 1][j] == '1':  # 如果上一行灯亮，必须按
                    press[i][j] = '1'
                    flip(state, i, j)

        # 检查最后一行是否全灭
        if all(state[4][j] == '0' for j in range(6)):
            return press

    return "impossible"

# 输出
ans = solve(start)
if ans == "impossible":
    print(ans)
else:
    for row in ans:
        print(" ".join(row))
```

```
   1   2   3   4   5   6   7
 #############################
1#   |   #   |   #   |   |   #
 #####---#####---#---#####---#
2#   #   |   #   #   #   #   #
 #---#####---#####---#####---#
3#   |   |   #   #   #   #   #
 #---#########---#####---#---#
4#   #   |   |   |   |   #   #
 #############################
        (图 1)

 #  = Wall
 |  = No wall
 -  = No wall
```

图1是一个城堡的地形图。请你编写一个程序，计算城堡一共有多少房间，最大的房间有多大。城堡被分割成m×n(m≤50，n≤50)个方块，每个方块可以有0~4面墙。

输入

程序从标准输入设备读入数据。第1、2行每行1个整数，分别是南北向、东西向的方块数。在接下来的输入行里，每个方块用一个数字(0≤p≤50)描述。用一个数字表示方块周围的墙，1表示西墙，2表示北墙，4表示东墙，8表示南墙。每个方块用代表其周围墙的数字之和表示。城堡的内墙被计算两次，方块(1,1)的南墙同时也是方块(2,1)的北墙。输入的数据保证城堡至少有两个房间。

输出

输出2行，每行一个数，表示城堡的房间数、城堡中最大房间所包括的方块数。结果显示在标准输出设备上。

样例输入

4 7 11 6 11 6 3 10 6 7 9 6 13 5 15 5 1 10 12 7 13 7 5 13 11 10 8 10 12 13

样例输出

5 9

来源

1164

```python
def solve():
    rows = int(input())
    cols = int(input())
    grid = []
    for _ in range(rows):
        row = list(map(int, input().split()))
        grid.append(row)

    visited = [[False for _ in range(cols)] for _ in range(rows)]
    room_count = 0
    max_room_size = 0

    directions = [(-1, 0, 2), (1, 0, 8), (0, -1, 1), (0, 1, 4)]

    for i in range(rows):
        for j in range(cols):
            if not visited[i][j]:
                stack = [(i, j)]
                visited[i][j] = True
                current_room_size = 0
                while stack:
                    x, y = stack.pop()
                    current_room_size += 1
                    for dx, dy, wall in directions:
                        nx = x + dx
                        ny = y + dy
                        if 0 <= nx < rows and 0 <= ny < cols:
                            if not visited[nx][ny] and (grid[x][y] & wall) == 0:
                                visited[nx][ny] = True
                                stack.append((nx, ny))
                room_count += 1
                if current_room_size > max_room_size:
                    max_room_size = current_room_size


    print(room_count)
    print(max_room_size)

solve()
```

P1130:红与黑

总时间限制: 1000ms 内存限制: 65536kB

描述

有一间长方形的房子，地上铺了红色、黑色两种颜色的正方形瓷砖。你站在其中一块黑色的瓷砖上，只能向相邻的黑色瓷砖移动。请写一个程序，计算你总共能够到达多少块黑色的瓷砖。

输入

包括多个数据集合。每个数据集合的第一行是两个整数W和H，分别表示x方向和y方向瓷砖的数量。W和H都不超过20。在接下来的H行中，每行包括W个字符。每个字符表示一块瓷砖的颜色，规则如下 1）'.'：黑色的瓷砖； 2）'#'：红色的瓷砖； 3）'@'：黑色的瓷砖，并且你站在这块瓷砖上。该字符在每个数据集合中唯一出现一次。 当在一行中读入的是两个零时，表示输入结束。

输出

对每个数据集合，分别输出一行，显示你从初始位置出发能到达的瓷砖数(记数时包括初始位置的瓷砖)。

样例输入

6 9 ....#. .....# .... #@...# .#..#. 0 0

样例输出

45

```
def dfs(x,y,visited,map,count):
    visited[x][y]=True
    count+=1
    row=len(map)#行数
    col=len(map[0])#列数
    directions=[(0,1),(1,0),(-1,0),(0,-1)]
    for dx,dy in directions:
        nx=x+dx
        ny=y+dy
        if 0<=nx<=row-1 and 0<=ny<=col-1 and not visited[nx][ny] and map[nx][ny]==0:
            count=dfs(nx,ny,visited,map,count)
    return count
while True:
    n,m=map(int,input().split())
    if n==0 and m==0:
        break
    maps=[[0 for _ in range(n)] for _ in range(m)]
    for i in range(m):
        a=input()
        a=list(str(a))
        for j in range(n):
            if a[j]=='#':
                maps[i][j]=1
            elif  a[j]=='.':

                maps[i][j]=0
            else:
                maps[i][j]=0
                current_x=i
                current_y=j
    visited=[[0 for _ in range(n)] for _ in range(m)]
    print(dfs(current_x,current_y,visited,maps,0))
```

P0820:冰阔落 I 总时间限制: 10000ms 内存限制: 65536kB 描述 老王喜欢喝冰阔落。

初始时刻，桌面上有n杯阔落，编号为1到n。老王总想把其中一杯阔落倒到另一杯中，这样他一次性就能喝很多很多阔落，假设杯子的容量是足够大的。

有m 次操作，每次操作包含两个整数x与y。

若原始编号为x 的阔落与原始编号为y的阔落已经在同一杯，请输出"Yes"；否则，我们将原始编号为y 所在杯子的所有阔落，倒往原始编号为x 所在的杯子，并输出"No"。

最后，老王想知道哪些杯子有冰阔落。

输入 有多组测试数据，少于 5 组。 每组测试数据，第一行两个整数 n, m (n, m<=50000)。接下来 m 行，每行两个整数 x, y (1<=x, y<=n)。 输出 每组测试数据，前 m 行输出 "Yes" 或者 "No"。 第 m+1 行输出一个整数，表示有阔落的杯子数量。 第 m+2 行有若干个整数，从小到大输出这些杯子的编号。 样例输入 3 2 1 2 2 1 4 2 1 2 4 3 样例输出 No Yes 2 1 3 No No 2 1 4 来源 Gong linyuan, Xu Yewen

```python
while True:
    try:
        n, m = map(int, input().split())
        cups = [1] * n
        sets = {i + 1: [i + 1] for i in range(n)}
        positions = list(range(1, n + 1))

        for _ in range(m):
            x, y = map(int, input().split())
            x_set = positions[x - 1]
            y_set = positions[y - 1]
            if x_set == y_set:
                print("Yes")
            else:
                print("No")
                sets[x_set].extend(sets[y_set])
                for item in sets[y_set]:
                    positions[item - 1] = x_set
                sets[y_set] = []
                cups[x_set - 1] = 1
                cups[y_set - 1] = 0

        active_cups_count = sum(cups)
        print(active_cups_count)
        active_cups = [i + 1 for i in range(n) if cups[i] == 1]
        print(" ".join(map(str, active_cups)))
    except EOFError:
        break
```

描述

假设你经营着一家公司，公司在北京和南京各有一个办公地点。公司只有你一个人，所以你只能每月选择在 一个城市办公。在第i个月，如果你在北京办公，你能获得Pi的营业额，如果你在南京办公，你能获得Ni的营 业额。但是，如果你某个月在一个城市办公，下个月在另一个城市办公，你需要支付M的交通费。那么，该怎 样规划你的行程（可在任何一个城市开始），才能使得总收入（总营业额减去总交通费）最大?

输入

输入的第一行有两个整数T（1 <= T <= 100）和M（1 <= M <= 100），T代表总共的月数，M代表交通 费。接下来的T行每行包括两个在1到100之间（包括1和100）的的整数，分别表示某个月在北京和在南京办 公获得的营业额。

输出

输出只包括一行，这一行只包含一个整数，表示可以获得的最大总收入。

样例输入

```
4 3
10 9
2 8
9 5
8 2
```

样例输出

```
31
```

```python
t, m = map(int, input().split())
incomelist = []
for _ in range(t):
    incomelist.append(list(map(int, input().split()))) # 读入两地营业额数据
dp = [[0] * (t) for _ in range(2)] # 准备动态规划数据表格
dp[0][0] = incomelist[0][0] # 第一个月可以选择在任意一地开始。
dp[1][0] = incomelist[0][1]
for col in range(1, t): # 问题泛化：前n个月在两地分别最多可以获得多少净收益?
    dp[0][col] = max(dp[0][col - 1], dp[1][col - 1] - m) + incomelist[col][0]
    dp[1][col] = max(dp[1][col - 1], dp[0][col - 1] - m) + incomelist[col][1]
# 关系：本地第n个月最大营业额 = 最大值（本地第n-1个月最大营业额，异地第n-1个月最大营业额 - 交通费） + 本地第n个月的营业额。
print(max(dp[0][-1], dp[1][-1]))
```

描述

N (1 ≤ N ≤ 100) cows, conveniently numbered 1..N, are participating in a programming contest. As we all know, some cows code better than others. Each cow has a certain constant skill rating that is unique among the competitors.

The contest is conducted in several head-to-head rounds, each between two cows. If cow A has a greater skill level than cow B (1 ≤ A ≤ N; 1 ≤ B ≤ N; A ≠ B), then cow A will always beat cow B.

Farmer John is trying to rank the cows by skill level. Given a list the results of M (1 ≤ M ≤ 4,500) two- cow rounds, determine the number of cows whose ranks can be precisely determined from the results. It is guaranteed that the results of the rounds will not be contradictory.

输入

- Line 1: Two space-separated integers: N and M
- Lines 2..M+1: Each line contains two space-separated integers that describe the competitors and results (the first integer, A, is the winner) of a single round of competition: A and B

输出

- Line 1: A single integer representing the number of cows whose ranks can be determined

输入

- Line 1: Two space-separated integers: N and M
- Lines 2..M+1: Each line contains two space-separated integers that describe the competitors and results (the first integer, A, is the winner) of a single round of competition: A and B

输出

- Line 1: A single integer representing the number of cows whose ranks can be determined

样例输入

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

样例输出

```
2
```

来源

USACO 2008 January Silver

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

# 初始化可达性矩阵，牛的编号从1到n
reachable = [[False] * (n + 1) for _ in range(n + 1)]

for _ in range(m):
    a, b = map(int, input().split())
    reachable[a][b] = True

# Floyd-Warshall算法计算传递闭包
for k in range(1, n + 1):
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if reachable[i][k] and reachable[k][j]:
                if not reachable[i][j]:
                    reachable[i][j] = True

count = 0
for i in range(1, n + 1):
    stronger = 0
    weaker = 0
    for j in range(1, n + 1):
        if i == j:
            continue
        if reachable[j][i]:
            stronger += 1
        if reachable[i][j]:
            weaker += 1
    if stronger + weaker == n - 1:
        count += 1

print(count)
```

描述

人生来就有三个生理周期，分别为体力、感情和智力周期，它们的周期长度为23天、28天和33天。每一个周期中有一天是高峰。在高峰这天，人会在相应的方面表现出色。例如，智力周期的高峰，人会思维敏捷，精力容易高度集中。因为三个周期的周长不同，所以通常三个周期的高峰不会落在同一天。对于每个人，我们想知道何时三个高峰落在同一天。对于每个周期，我们会给出从当前年份的第一天开始，到出现高峰的天数（不一定是第一次高峰出现的时间）。你的任务是给定一个从当年第一天开始数的天数，输出从给定时间开始（不包括给定时间）下一次三个高峰落在同一天的时间（距给定时间的天数）。例如：给定时间为10，下次出现三个高峰同天的时间是12，则输出2（注意这里不是3）。

输入

一行，包含四个整数：p, e, i和d，相邻两个整数之间用单个空格隔开。 p, e, i分别表示体力、情感和智力高峰出现的时间（时间从当年的第一天开始计算）。d 是给定的时间，可能小于p, e, 或 i。 所有给定时间是非负的并且小于等于365, 所求的时间小于等于21252。

输出

一个整数，即从给定时间起，下一次三个高峰同天的时间（距离给定时间的天数）。

样例输入

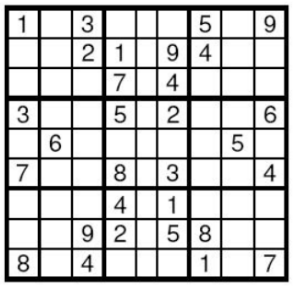
```
4 5 6 7
```

样例输出

```
16994
```

```python
p,e,i,d=map(int,input().split())
sum=0
while True:
    if (sum-p)%23==0 and (sum-e)%28==0 and (sum-i)%33==0:
        print(sum-d if sum>d else sum+21252-d)
        break
    sum+=1
```

Sudoku is a very simple task. A square table with 9 rows and 9 columns is divided to 9 smaller squares 3x3 as shown on the Figure. In some of the cells are written decimal digits from 1 to 9. The other cells are empty. The goal is to fill the empty cells with decimal digits from 1 to 9, one digit per cell, in such way that in each row, in each column and in each marked 3x3 subsquare, all the digits from 1 to 9 to appear. Write a program to solve a given Sudoku-task.



输入

The input data will start with the number of the test cases. For each test case, 9 lines follow, corresponding to the rows of the table. On each line a string of exactly 9 decimal digits is given, corresponding to the cells in this line. If a cell is empty it is represented by 0.

输出

For each test case your program should print the solution in the same format as the input data. The empty cells have to be filled according to the rules. If solutions is not unique, then the program may print any one of them.

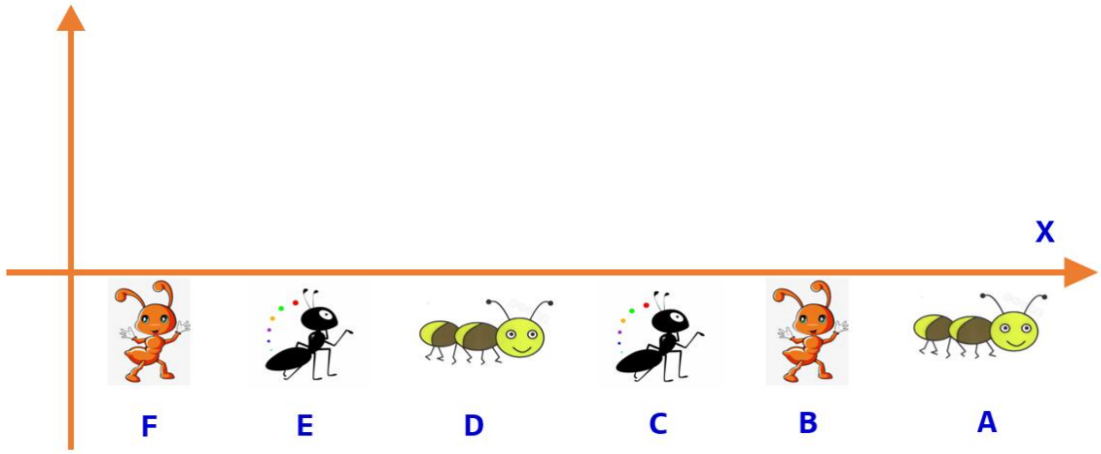
样例输入

```
1
103000509
002109400
000704000
300502006
060000050
700803004
000401000
009205800
804000107
```

样例输出

```
143628579
572139468
986754231
391542786
468917352
725863914
237481695
619275843
854396127
```

为了促进蚂蚁家族身体健康，提高蚁族健身意识，蚂蚁王国举行了越野跑。假设越野跑共有N个蚂蚁参加，在一条笔直的道路上进行。N个蚂蚁在起点处站成一列，相邻两个蚂蚁之间保持一定的间距。比赛开始后，N个蚂蚁同时沿着道路向相同的方向跑去。换句话说，这N个蚂蚁可以看作x轴上的N个点，在比赛开始后，它们同时向X轴正方向移动。假设越野跑的距离足够远，这N个蚂蚁的速度有的不相同有的相同且保持匀速运动，那么会有多少对参赛者之间发生"赶超"的事件呢？此题结果比较大。



输入

第一行1个整数N。
第一行1个整数N。
第2... N +1行：N 个非负整数，按从前到后的顺序给出每个蚂蚁的跑步速度。对于50%的数据，2<=N<=1000。对于100%的数据，2<=N<=100000。

输出

一个整数，表示有多少对参赛者之间发生赶超事件。

样例输入

```
5
1
5
10
7
6

5
1
5
5
7
6
```

样例输出

```
7

8
```

```python
def mergesort(a):
    ans=i=j=s=0
    if len(a)<=1:
        return 0
    left,right=a[:len(a)//2],a[len(a)//2:]
    ans+=mergesort(left)
    ans+=mergesort(right)
    a.clear()
    while i<len(left) and j<len(right):
        if left[i]>=right[j]:
            a.append(right[j])
            s+=i
            j+=1
        else:
            a.append(left[i])
            i+=1
    if i<len(left):
        a+=left[i:]
    while j<len(right):
        a.append(right[j])
        s+=i
        j+=1
    return ans+s
while True:
    try:
        s=input()
        if s=='':
            continue
        n=int(s)
        a=[int(input()) for _ in range(n)]
        print(mergesort(a))
    except EOFError:
        break
```

描述

小明有一些矩形的材料，他要从这些矩形材料中切割出一些正方形。

当他面对一块矩形材料时，他总是平行于较短的切割一刀，切出一块最大的正方形，剩下一块矩形，然后再切割剩下的矩形材料，直到全部切为正方形为止。

例如，对于一块两边长度分别为 5 和 3 的材料（记为 5×3），小明会依次切出 3×3、2×2、1×1、1×1 共 4 个正方形。

现在小明有一块矩形的材料，两边长分别是 N 和 M。请问小明最终会 切出多少个正方形?

输入

一行两个整数N和M。
1 <= N, M <= 3000

输出

一个整数代表答案

样例输入

5 3

样例输出

4

```
n,m=map(int,input().split())
count_square=0
while m>0 and n>0:
    m,n=max(m,n)-min(m,n),min(m,n)
    count_square+=1
print(count_square)
```

- 描述

  输入两个串s1,s2，找出s2在s1中所有出现的位置两个子串的出现不能重叠。例如'aa'在 aaaa 里出现的位置只有0,2

- 输入

  第一行是整数n 接下来有n行，每行两个不带空格的字符串s1,s2

- 输出

  对每行，从小到大输出s2在s1中所有的出现位置。位置从0开始算 如果s2没出现过，输出 "no" 行末多输出空格没关系

- 样例输入

```
4
ababcdefgabdefab ab
aaaaaaaaa a
aaaaaaaaa aaa
112123323 a
```

- 样例输出

```
0 2 9 14
0 1 2 3 4 5 6 7 8
0 3 6
no
```

```python
n=int(input())
input1=[]
result=[]
for i in range(n):
    s1,s2=map(str,input().split())
    current=[]
    if len(s1)>=len(s2):
        j=0
        while j in range(len(s1)-len(s2)+1):
            if s2==s1[j:j+len(s2)]:
                current.append(j)
                j+=len(s2)
            else:
                j+=1
        if current!=[]:
            result.append(current)
        else:
            result.append(['no'])
    else:
        result.append(['no'])
for i in result:
    for j in range(len(i)):
        print(i[j],end=" "if j<len(i)-1 else "\n")
```

给定一个只有2,3的正整数

能够将其中你最多一个数从2变成3

请输出改变后最大的值

输入

一个正整数 n 1 < n < 10^4

输出

一个正整数

样例输入

2333

样例输出

3333

提示

试着使用尽可能少的代码行数完成

```
print(input().replace('2', '3', 1))
```

描述

我们称一个字符的数组S为一个序列。对于另外一个字符数组Z,如果满足以下条件，则称Z是S的一个子序列： （1）Z中的每个元素都是S中的元素 （2）Z中元素的顺序与在S中的顺序一致。例如：当S = (E,R,C,D,F,A,K)时， （E，C，F）和（E，R）等等都是它的子序列。而（R，E）则不是。

现在我们给定两个序列，求它们最长的公共子序列的长度。

输入

一共两行，分别输入两个序列。

输出

一行，输出最长公共子序列的长度。

样例输入

```
ABCBDAB
BDABA
```

样例输出

```
4
```

```python
def long_com(s1, s2):
    m, n = len(s1), len(s2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]


s1 = input().strip()
s2 = input().strip()

print(long_com(s1, s2))
```

描述

给一个由1跟0组成的方形地图，1代表土地，0代表水域

相邻(上下左右4个方位当作相邻)的1组成孤岛

现在你可以将0转成1，搭建出一个链接2个孤岛的桥

请问最少要将几个0转成1，才能建成链接孤岛的桥。

题目中恰好有2个孤岛(顾答案不会是0)

输入

一个正整数n，代表几行输入 n行0跟1字串

输出

一个正整数k，代表最短距离

样例输入

3 110 000 001

样例输出

2

提示

```python
import sys
from collections import deque

def main():
    n = int(sys.stdin.readline())
    grid = []
    for _ in range(n):
        line = sys.stdin.readline().strip()
        grid.append(list(line))

    # Find all islands using BFS
    islands = []
    visited = [[False for _ in range(n)] for _ in range(n)]
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for i in range(n):
        for j in range(n):
            if grid[i][j] == '1' and not visited[i][j]:
                queue = deque()
                queue.append((i, j))
                visited[i][j] = True
                island = []
                while queue:
                    x, y = queue.popleft()
                    island.append((x, y))
                    for dx, dy in directions:
                        nx, ny = x + dx, y + dy
                        if 0 <= nx < n and 0 <= ny < n and grid[nx][ny] == '1' and not visited[nx][ny]:
                            visited[nx][ny] = True
                            queue.append((nx, ny))
                islands.append(island)

    # There should be exactly two islands
    island1 = islands[0]
    island2 = islands[1]

    min_distance = float('inf')
    for (x1, y1) in island1:
        for (x2, y2) in island2:
            distance = abs(x1 - x2) + abs(y1 - y2) - 1
            if distance < min_distance:
                min_distance = distance

    print(min_distance)

if __name__ == "__main__":
    main()
```

描述

泰波拿契数列 Tn 定义是

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

给定n请算出Tn

n的范围:1<=n<=30

输入

一个正整数n

输出

一个正整数k

样例输入

4

样例输出

4

提示

T3=0 + 1 + 1 = 2 T4=1 + 1 + 2 = 4

```
i0, i1, i2 = 0, 1, 1
for _ in range(0, int(input())-2):
    i2, i1, i0 = i0+i1+i2, i2, i1
print(i2)
```

描述

将正整数n 表示成一系列正整数之和，n=n1+n2+…+nk, 其中n1>=n2>=…>=nk>=1，k>=1 。 正整数n 的这种表示称为正整数n 的划分。正整数n 的不同的划分个数称为正整数n 的划分数

输入

一个整数N(0 < N <= 30)。

输出

输出N的划分数。

样例输入

5

样例输出

7

提示

5, 4+1, 3+2, 3+1+1, 2+2+1, 2+1+1+1, 1+1+1+1+1

```
n = int(input().strip())
board = [[0]*(n+1) for _ in range(n+1)]
board[1][1] = 1
for i in range(2,n+1):
    board[i][1] = 1
for m in range(2,n+1):
    for k in range(2,n+1):
        if m-k>=0:
            board[m][k] = board[m-k][k] + board[m-1][k-1]
        else:
            board[m][k] = 0

ans = 0
for i in range(1,n+1):
    ans += board[n][i]

print(ans)
```

250205:放平果（盘子相同）

总时间限制: 65536ms 单个测试点时间限制: 65535ms 内存限制: 65535kB

描述

把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用K表示）5，1，1和1，5，1 是同一种分法。

输入

苹果个数m 和盘子个数n(0<=M，1<=N<=10)

输出

不同的放法数目

样例输入

```
7 3
```

样例输出

```
8
```

```python
m,n=map(int,input().split())
ans=0
def dfs(apple,plant,mx):
    global ans
    if plant==1:
        if apple>=mx:
            ans+=1
        return
    for i in range(mx, apple+1):
        dfs(apple-i,plant-1,i)

dfs(m,n,0)
print (ans)
```

输入

On the first and only line are the numbers A, B, and C. These are all integers in the range from 1 to 100 and C≤max(A,B).

输出

The first line of the output must contain the length of the sequence of operations K. The following K lines must each describe one operation. there are several sequences of minimal length, output any one of them. If the desired result can't be achieved, the first and only line of the fil must contain the word 'impossible'.

样例输入

```
3 5 4
```

样例输出

```
6
FILL(2)
POUR(2,1)
DROP(1)
POUR(2,1)
FILL(2)
POUR(2,1)
```

```python
from collections import deque

def bfs(A, B, C):
    queue = deque([((0, 0), [])])
    visited = {(0, 0)}

    while queue:
        (a, b), path = queue.popleft()
        if a == C or b == C:
            print(len(path))
            for PATH in path:
                print(PATH)
            return

        #FILL(1)
        if (A, b) not in visited:
            visited.add((A, b))
            queue.append(((A, b), path + ["FILL(1)"]))
        #FILL(2)
        if (a, B) not in visited:
            visited.add((a, B))
            queue.append(((a, B), path + ["FILL(2)"]))
        #DROP(1)
        if (0, b) not in visited:
            visited.add((0, b))
            queue.append(((0, b), path + ["DROP(1)"]))
        #DROP(2)
        if (a, 0) not in visited:
            visited.add((a, 0))
            queue.append(((a, 0), path + ["DROP(2)"]))
        #POUR(1, 2)
        new_a = a - min(a, B - b)
        new_b = b + min(a, B - b)
        if (new_a, new_b) not in visited:
            visited.add((new_a, new_b))
            queue.append(((new_a, new_b), path + ["POUR(1,2)"]))
        #POUR(2, 1)
        new_a = a + min(b, A - a)
        new_b = b - min(b, A - a)
        if (new_a, new_b) not in visited:
            visited.add((new_a, new_b))
            queue.append(((new_a, new_b), path + ["POUR(2,1)"]))

    print("impossible")
    return

A, B, C = map(int, input().split())
bfs(A, B, C)
```

古代有一个梵塔，塔内有三个座A、B、C，A座上有n个盘子，盘子大小不等，大的在下，小的在上。三个座都可以用来放盘子。有一个和尚想把这n个盘子从A座移到C座，但每次只能允许移动一个盘子，并且在移动过程中，3个座上的盘子始终保持大盘在下，小盘在上。输入盘子数目n，要求输出移动的步骤。

输入

盘子数目n (n < 8)

输出

移动方案

样例输入

3

样例输出

A->C

A->B

C->B

A->C

B->A

B->C

A->C

```python
def hanoi(n, start, end, auxiliary):
    if n == 1:
        print(f"{start}->{end}")
    else:
        hanoi(n-1, start, auxiliary, end)
        print(f"{start}->{end}")
        hanoi(n-1, auxiliary, end, start)

n = int(input())
hanoi(n, 'A', 'C', 'B')
```

描述

国际象棋的棋盘是由8×8共64个方格构成，棋子放在方格里面。如果两个皇后棋子在同一行、同一列，或者在某个正方形的对角线上，那么这两个皇后就会互相攻击。请在棋盘上摆放8个皇后，使得它们都不会互相攻击。这是经典的8皇后问题。

现在要解决N皇后问题：将N个皇后摆放在一个N行N列的国际象棋棋盘上，要求任何两个皇后不能互相攻击。输入皇后数N(1<=N<=9),输出所有的摆法。无解输出"NO ANSWER"。行列号都从0开始算。

输入

一个整数N，表示要把N个皇后摆放在一个N行N列的国际象棋棋盘上

输出

所有的摆放放案。每个方案一行，依次是第0行皇后位置、第1行皇后位置.....第N-1行皇后位置。 多种方案输出顺序如下：优先输出第0行皇后列号小的方案。如果两个方案第0行皇后列号一致，那么优先输出第1行皇后列号小的方案.....以此类推

样例输入

4

样例输出

1 3 0 2 2 0 3 1

```python
def isok(i, num): # 判断是否为符合要求的位置
    for col in range(i):
        if res[col] == num or abs(res[col] - num) == abs(i - col):
            return False
    return True

def putq(col):
    global res, ans
    if col == N: # 如果列数为N，说明已经摆好一个完整的棋盘，输出到ans表中暂存。
        ans.append([res[_] for _ in range(0,N)])
        return
    for row in range(0, N): # 枚举一列中的每一个位置
        if isok(col, row): # 如果这个位置对于它之前的棋子来说是和要求的摆放
            res[col] = row # 在这个位置放上一个棋子
            putq(col + 1) # 递归到下一列
            res[col] = -1 # 拿走这个棋子
    return

N = int(input())
res = [-1] * (N)
ans = []
putq(0)
if ans: # 如果ans表不为空，输出。否则，输出"NO ANSWER"。
    for _ in ans:
        print(*_)
else:
    print("NO ANSWER")
```

描述

小明有很多猪，他喜欢玩叠猪游戏，就是将猪一头头叠起来。猪叠上去后，还可以把顶上的猪拿下来。小明知道每头猪的重量，而且他还随时想知道叠在那里的猪最轻的是多少斤。

输入

有三种输入 1)push n n是整数(0<=0 <=20000)，表示叠上一头重量是n斤的新猪 2)pop 表示将猪堆顶的猪赶走。如果猪堆没猪，就啥也不干 3)min 表示问现在猪堆里最轻的猪多重。如果猪堆没猪，就啥也不干

输入总数不超过100000条

输出

对每个min输入，输出答案。如果猪堆没猪，就啥也不干

样例输入

pop min push 5 push 2 push 3 min push 4 min

样例输出

2 2

```
import sys

data = sys.stdin.read().splitlines()
pigs = []
m_pigs = []

def push(x,pigs,m_pigs):
    pigs.append(x)
    if not m_pigs:
        m_pigs.append(x)
    elif x<=m_pigs[-1]:
        m_pigs.append(x)

for line in data:
    if line =="min":
        if m_pigs:
            print(m_pigs
    elif line =="pop":
        if pigs:
            x=pigs.pop()
            if x == m_pi
                m_pigs.p
    else:
        a,b = line.split
        push(int(b),pigs
```

求左边的字符一定比右边的字符先入栈，出栈顺序无要求。再给定若干字符串，对每个字符串，判断其是否是可能的x中的字符的出栈序列。

输入

第一行是原始字符串x 后面有若干行(不超过50行)，每行一个字符串，所有字符串长度不超过100

输出

对除第一行以外的每个字符串，判断其是否是可能的出栈序列。如果是，输出"YES"，否则，输出"NO"

样例输入

```
abc
abc
bca
cab
```

样例输出

```
YES
YES
NO
```

求左边的字符一定比右边的字符先入栈，出栈顺序无要求。再给定若干字符串，对每个字符串，判断其是否是可能的x中的字符的出栈序列。

输入

第一行是原始字符串x 后面有若干行(不超过50行)，每行一个字符串，所有字符串长度不超过100

输出

对除第一行以外的每个字符串，判断其是否是可能的出栈序列。如果是，输出"YES"，否则，输出"NO"

样例输入

```
abc
abc
bca
cab
```

样例输出

```
YES
YES
NO
```

```python
def judge(n):
    if len(n) != len(str):
        return "NO"
    data=[]
    i=0
    j=0
    while i < len(n) or j < len(str):
        if len(data)>0 and data[-1] == n[i]:
            del data[-1]
            i+=1
        else:
            if j >= len(str):
                break
            data.append(str[j])
            j+=1
    if i == len(n):
        return "YES"
    else:
        return "NO"
str=input()
while True:
    try:
        s=input()
        print(judge(s))
    except EOFError:
        break
```