P1000:上机

总时间限制: 1000ms 内存限制: 65536kB

描述

又到周末了，同学们陆陆续续开开心心的来到机房上机。jbr也不例外，但是他到的有点晚，发现有些机位上已经有同学正在做题，有些机位还空着。细心的jbr发现，一位同学来到机房，坐在机位i上，如果他的左右两边都空着，他将获得能力值a[i]；如果当他坐下时，左边或者右边已经有一个人在上机了，他将获得能力值b[i]；如果当他坐下时，他的左边右边都有人在上机，他将获得能力值c[i]。

同时他发现，已经在上机的同学不会受到刚要坐下的同学的影响，即他们的能力值只会在坐下时产生，以后不会发生变化;第一个机位左边没有机位，最后一个机位右边没有机位，无论何时坐在这两个机位上将无法获得c值。

这时jbr发现有一排机器还空着，一共有N个机位，编号1到N。这时有N位同学们陆陆续续来到机房，一个一个按照顺序坐在这排机位上。聪明的jbr想知道怎么安排座位的顺序，可以使这N位同学获得能力值的和最大呢?

输入

第一行一个整数N(1<= N <= 10000) 第二行N个数，表示a[i] 第三行N个数，表示b[i] 第四行N个数，表示c[i] (1<= a[i],b[i],c[i] <=10000)

输出

一个整数，表示获得最大的能力值和

样例输入

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

样例输出

```
14
```

```python
n = int(input())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
c = list(map(int, input().split()))
c[0], c[-1] = 0, 0
buffer = {}


def divide_dp(start: int, end: int, left: bool, right: bool) -> int:
    '''二分动归，通过中间座位最终结果情况计算对应情况下的最大值，left与right变量记录了这一数开始坐人的时候，左右两侧的状态，超出座位范围记为False'''
    global n, a, b, c
    # 将某个座位有一个人坐下时两侧有人的数量记做状态标号
    buffer.setdefault((start, end, left, right), -1)
    if buffer.setdefault((start, end, left, right), -1) != -1:
        return buffer[(start, end, left, right)]
    if start == end - 1:
        # 第一递归出口
        if not left:
            if not right:
                buffer[(start, end, left, right)] = max(
                    a[start] + b[end], b[start] + a[end])
                return buffer[(start, end, left, right)]
            else:
                buffer[(start, end, left, right)] = max(
                    a[start] + c[end], b[start] + b[end])
                return buffer[(start, end, left, right)]
        else:
            if not right:
                buffer[(start, end, left, right)] = max(
                    b[start] + b[end], c[start] + a[end])
                return buffer[(start, end, left, right)]
            else:
                buffer[(start, end, left, right)] = max(
                    b[start] + c[end], c[start] + b[end])
                return buffer[(start, end, left, right)]

    if start == end:
        # 第二递归出口
        if not left:
            if not right:
                buffer[(start, end, left, right)] = a[start]
                return buffer[(start, end, left, right)]
            else:
                buffer[(start, end, left, right)] = b[start]
                return buffer[(start, end, left, right)]
        else:
            if not right:
                buffer[(start, end, left, right)] = b[start]
                return buffer[(start, end, left, right)]
            else:
                buffer[(start, end, left, right)] = c[start]
                return buffer[(start, end, left, right)]
    # 继续二分
    mid = (start + end) // 2
    # 选择mid的情况，即0,1,2，对应有四种情况
    ans = divide_dp(start, mid - 1, left, True) + \
        divide_dp(mid + 1, end, True, right) + a[mid]
    # mid选择0
    ans = max(ans, divide_dp(start, mid - 1, left, True) +
              divide_dp(mid + 1, end, False, right) + b[mid])
    ans = max(ans, divide_dp(start, mid - 1, left, False) +
              divide_dp(mid + 1, end, True, right) + b[mid])
    # mid选择1的两种情况
    ans = max(ans, divide_dp(start, mid - 1, left, False) +
              divide_dp(mid + 1, end, False, right) + c[mid])
    # mid选择2
    buffer[(start, end, left, right)] = ans
    return ans


print(divide_dp(0, n-1, False, False))
```

P1190:变换的迷宫

总时间限制: 1000ms 内存限制: 65536kB

描述

你现在身处一个R*C 的迷宫中，你的位置用"S" 表示，迷宫的出口用"E" 表示。

迷宫中有一些石头，用"#" 表示，还有一些可以随意走动的区域，用"." 表示。

初始时间为0 时，你站在地图中标记为"S" 的位置上。你每移动一步（向上下左右方向移动）会花费一个单位时间。你必须一直保持移动，不能停留在原地不走。

当前时间是K 的倍数时，迷宫中的石头就会消失，此时你可以走到这些位置上。在其余的时间里，你不能走到石头所在的位置。

求你从初始位置走到迷宫出口最少需要花费多少个单位时间。

如果无法走到出口，则输出"Oop!"。

输入

第一行是一个正整数 T，表示有 T 组数据。 每组数据的第一行包含三个用空格分开的正整数，分别为 R、C、K。 接下来的 R 行中，每行包含了 C 个字符，分别可能是 "S"、"E"、"#" 或 "."。 其中，0 < T <= 20，0 < R, C <= 100，2 <= K <= 10。

输出

对于每组数据，如果能够走到迷宫的出口，则输出一个正整数，表示最少需要花费的单位时间，否则输出 "Oop!"。

样例输入

```
1
6 6 2
...S..
...#..
.#....
...#..
...#..
..#E#.
```

样例输出

```
7
```

```python
def route(maze, k):
    m, n = len(maze), len(maze[0])
    visited = [ [[0] * k for _ in range(n)] for _ in range(m)]
    time = 0
    l1, l2 = [(i, j) for i in range(m) for j in range(n) if maze[i][j] == 'S'], []
    for i, j in l1:
        visited[i][j][0] = 1

    def nbr(i, j):
        for ii, jj in ((-1,0),(1,0),(0,-1),(0,1)):
            if i+ii in range(m) and j+jj in range(n):
                yield (i+ii, j+jj)

    while len(l1) > 0:
        time += 1
        for i, j in l1:
            for x, y in nbr(i, j):
                if visited[x][y][time % k] == 1:
                    continue
                if maze[x][y] in '.S' :
                    visited[x][y][time % k] = 1
                    l2.append((x, y))
                elif maze[x][y] == 'E' :
                    return time
                elif maze[x][y] == '#' and time % k == 0:
                    visited[x][y][0] = 1
                    l2.append((x, y))
        l1, l2 = l2, []
    else:
        return None

if __name__ == "__main__":
    for _ in range(int(input())):
        m, n, k = map(int, input().split())
        maze = [ list(input())[:n] for _ in range(m)]
        res = route(maze, k)
        print(res if res else "Oop!")
```

P1210:Saving Tang Monk

总时间限制: 1000ms 内存限制: 65536kB

描述

《Journey to the West》(also 《Monkey》) is one of the Four Great Classical Novels of Chinese literature. It was written by Wu Cheng'en during the Ming Dynasty. In this novel, Monkey King Sun Wukong, pig Zhu Bajie and Sha Wujing, escorted Tang Monk to India to get sacred Buddhism texts.

During the journey, Tang Monk was often captured by demons. Most of demons wanted to eat Tang Monk to achieve immortality, but some female demons just wanted to marry him because he was handsome. So, fighting demons and saving Monk Tang is the major job for Sun Wukong to do.

Once, Tang Monk was captured by the demon White Bones. White Bones lived in a palace and she cuffed Tang Monk in a room. Sun Wukong managed to get into the palace. But to rescue Tang Monk, Sun Wukong might need to get some keys and kill some snakes in his way.

The palace can be described as a matrix of characters. Each character stands for a room. In the matrix, 'K' represents the original position of Sun Wukong, 'T' represents the location of Tang Monk and 'S' stands for a room with a snake in it. Please note that there are only one 'K' and one 'T', and at most five snakes in the palace. And, '.' means a clear room as well '#' means a deadly room which Sun Wukong couldn't get in.

There may be some keys of different kinds scattered in the rooms, but there is at most one key in one room. There are at most 9 kinds of keys. A room with a key in it is represented by a digit(from '1' to '9'). For example, '1' means a room with a first kind key, '2' means a room with a second kind key, '3' means a room with a third kind key... etc. To save Tang Monk, Sun Wukong must get ALL kinds of keys(in other words, at least one key for each kind).

For each step, Sun Wukong could move to the adjacent rooms(except deadly rooms) in 4 directions(north,west,south and east), and each step took him one minute. If he entered a room in which a living snake stayed, he must kill the snake. Killing a snake also took one minute. If Sun Wukong entered a room where there is a key of kind N, Sun would get that key if and only if he had already got keys of kind 1,kind 2 ... and kind N-1. In other words, Sun Wukong must get a key of kind N before he could get a key of kind N+1 (N>=1). If Sun Wukong got all keys he needed and entered the room in which Tang Monk was cuffed, the rescue mission is completed. If Sun Wukong didn't get enough keys, he still could pass through Tang Monk's room. Since Sun Wukong was a impatient monkey, he wanted to save Tang Monk as quickly as possible. Please figure out the minimum time Sun Wukong needed to rescue Tang Monk.

输入

There are several test cases.

For each case, the first line includes two integers N and M(0 < N <= 100, 0 <= M <= 9), meaning that the palace is a N * N matrix and Sun Wukong needed M kinds of keys(kind 1, kind 2, ... kind M).

Then the N*N matrix follows.

The input ends with N = 0 and M = 0.

输出

For each test case, print the minimum time (in minute) Sun Wokong needed to save Tang Monk. If it's impossible for Sun Wokong to complete the mission, print "impossible".

样例输入

```
3 1
K.S
##1
1#T
3 1
K#T
.S#
1#.
3 2
K#T
.S.
21.
0 0
```

样例输出

```
5
impossible
8
```

```python

input = sys.stdin.read
DIRS = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 4 个方向：上、下、左、右

class Node:
    def __init__(self, x, y, time, key, snake):
        self.x = x  # 当前位置
        self.y = y  # 当前位置
        self.time = time  # 当前耗费时间
        self.key = key  # 当前拿到的钥匙数量
        self.snake = snake  # 当前杀死的蛇的状态（用 bitmask 记录）

    def __lt__(self, other):
        return self.time < other.time  # 优先队列需要按照时间排序（Dijkstra）

def bfs(maze, N, M):
    # 记录初始位置和蛇的位置
    x0, y0, snake_count = 0, 0, 0
    snake_map = [[-1] * N for _ in range(N)]  # 记录蛇的编号
    for i in range(N):
        for j in range(N):
            if maze[i][j] == 'K':  # 孙悟空的起点
                x0, y0 = i, j
            elif maze[i][j] == 'S':  # 记录蛇的位置
                snake_map[i][j] = snake_count
                snake_count += 1

    # Dijkstra 需要的优先队列
    queue = []
    heapq.heappush(queue, Node(x0, y0, 0, 0, 0))

    # 记录访问状态：memo[x][y][key_count] 表示在 (x, y) 拥有 key_count 把钥匙的最短时间
    INF = float('inf')
    memo = [[[INF] * (M + 1) for _ in range(N)] for _ in range(N)]
    memo[x0][y0][0] = 0  # 初始状态

    while queue:
        node = heapq.heappop(queue)

        # 遍历四个方向
        for dx, dy in DIRS:
            nx, ny = node.x + dx, node.y + dy
            if 0 <= nx < N and 0 <= ny < N:
                cell = maze[nx][ny]
                new_time = node.time + 1
                new_key = node.key
                new_snake_mask = node.snake

                if cell == '#':  # 死亡房间，不能进入
                    continue
                elif cell == 'S':  # 遇到蛇
                    snake_id = snake_map[nx][ny]
                    if (node.snake & (1 << snake_id)):  # 已杀死，正常通行
                        pass
                    else:  # 需要额外 1 分钟杀死蛇
                        new_time += 1
                        new_snake_mask |= (1 << snake_id)
                elif cell.isdigit():  # 遇到钥匙
                    key_id = int(cell)
                    if key_id == node.key + 1:  # 必须按顺序拾取
                        new_key += 1
                elif cell == 'T' and new_key == M:  # 营救，并且钥匙足够
                    return new_time

                # 如果新的状态更优，则更新
                if new_time < memo[nx][ny][new_key]:
                    memo[nx][ny][new_key] = new_time
                    heapq.heappush(queue, Node(nx, ny, new_time, new_key, new_snake_mask))

    return "impossible"  # 无法到达

# 读取输入
def main():
    input_data = input().strip().split("\n")
    index = 0
    results = []

    while index < len(input_data):
        # 读取 N, M
        N, M = map(int, input_data[index].split())
        if N == 0 and M == 0:
            break

        # 读取营救地图
        maze = [list(input_data[i]) for i in range(index + 1, index + 1 + N)]
        results.append(str(bfs(maze, N, M)))

        # 更新索引
        index += N + 1

    # 输出结果
    print("\n".join(results))

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

P0960:Zipper

总时间限制: 1000ms 内存限制: 65536kB

描述

Given three strings, you are to determine whether the third string can be formed by combining the characters in the first two strings. The first two strings can be mixed arbitrarily, but each must stay in its original order.

For example, consider forming "tcraete" from "cat" and "tree":

String A: cat String B: tree String C: tcraete

As you can see, we can form the third string by alternating characters from the two strings. As a second example, consider forming "catrtee" from "cat" and "tree":

String A: cat String B: tree String C: catrtee

Finally, notice that it is impossible to form "cttaree" from "cat" and "tree".

输入

The first line of input contains a single positive integer from 1 through 1000. It represents the number of data sets to follow. The processing for each data set is identical. The data sets appear on the following lines, one data set per line.

For each data set, the line of input consists of three strings, separated by a single space. All strings are composed of upper and lower case letters only. The length of the third string is always the sum of the lengths of the first two strings. The first two strings will have lengths between 1 and 200 characters, inclusive.

输出

For each data set, print:

Data set n: yes

if the third string can be formed from the first two, or

Data set n: no

if it cannot. Of course n should be replaced by the data set number. See the sample output below for an example.

样例输入

3 cat tree tcraete cat tree catrtee cat tree cttaree

样例输出

Data set 1: yes Data set 2: yes Data set 3: no

```
from collections import Counter
n=int(input())
for case in range(1,n+1):
    s1,s2,s3=input().split()
    len1,len2=len(s1),len(s2)
    len3=len(s3)
    if len3!=len1+len2 or Counter(s1+s2)!=Counter(s3):
        print(f"Data set {case}: no")
        continue
    dp=[False]*(len2+1)
    dp[0]=True
    for j in range(1,len2+1):
        dp[j]=dp[j-1] and(s2[j-1]==s3[j-1])
    for i in range(1,len1+1):
        dp[0]=dp[0] and(s1[i-1]==s3[i-1])
        for j in range(1,len2+1):
            pos=i+j-1
            from_s1=(s1[i-1]==s3[pos])and dp[j]
            from_s2=(s2[j-1]==s3[pos])and dp[j-1]
            dp[j]=from_s1 or from_s2
    print(f"Data set {case}: {'yes' if dp[len2] else 'no'}")
```

P1390:畜栏保留问题

总时间限制: 1000ms 内存限制: 65536kB

描述

农场有N头牛，每头牛会在一个特定的时间区间[A, B]（包括A和B）在畜栏里挤奶，且一个畜栏里同时只能有一头牛在挤奶。现在农场主希望知道最少几个畜栏能满足上述要求，并要求给出每头牛被安排的方案。对于多种可行方案，主要输出一种即可。

输入

输入的第一行包含一个整数N(1 ≤ N ≤ 50, 000)，表示有N牛头；接下来N行每行包含两个数，分别表示这头牛的挤奶时间[Ai, Bi](1 ≤ A≤ B ≤ 1, 000, 000)。

输出

输出的第一行包含一个整数，表示最少需要的畜栏数；接下来N行，第i+1行描述了第i头牛所被分配的畜栏编号（从1开始）。

样例输入

```
5
1 10
2 4
3 6
5 8
4 7
```

样例输出

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

```python
import heapq
import sys

def assign_cows() :
    n=int(sys.stdin.readline())
    cows=[]
    for i in range(n) :
        a,b=map(int,sys.stdin.readline().split())
        cows.append((a,b,i))
    cows.sort()

    stalls=[]
    assignment=[0]*n
    stall_id=0
    for a,b,idx in cows :
        if stalls and stalls[0][0]<a :
            end_time,sid=heapq.heappop(stalls)
            assignment[idx]=sid
            heapq.heappush(stalls,(b,sid))
        else:
            stall_id+=1
            assignment[idx]=stall_id
            heapq.heappush(stalls,(b,stall_id))

    print(stall_id)
    for a in assignment :
        print(a)

assign_cows()
```

描述

在一门课程中，一共有n场考试。假如你在i场考试中可以答对bi道题中的ai道，那么你的累计平均分定义为：100·Σai/Σbi。已知你这i场考试的答题情况，并且允许你放弃其中的k场考试，请你确定你最高能够得到多少的累计平均分。

假设该课程一共有3门考试，你的答题情况为5/5，0/1和2/6。如果你每门都参加，你的累计平均分为100·(5+0+2)/(5+1+6)= 50分。如果你放弃第3场考试，你的累计平均分则提高到了100·(5+0)/(5+1)= 83.33 ≈ 83分。

输入

有多组测试数据，每组测试数据包括3行。 每组测试数据第一行有两个数n和k，接下来一行有n个数ai，最后一行n个数bi。 (1 ≤ k < n ≤ 1000) (1 ≤ ai ≤ bi ≤ 1, 000, 000, 000)。 输入的最后一行为0 0，不作处理。

输出

输出最高的累计平均分。 (四舍五入到整数)

样例输入

```
3 1
5 0 2
5 1 6
4 2
1 2 7 9
5 6 7 9
0 0
```

样例输出

```
83
100
```

```python
def check(v, a, b, n, k):
    temp = []
    for i in range(n):
        temp.append(a[i] - v * b[i])
    temp.sort()
    sum_val = sum(temp[k:])
    return sum_val >= 0


def solve():
    while True:
        n,k=map(int,input().split())
        if n==0 and k==0:
            break
        a=list(map(int,input().split()))
        b=list(map(int,input().split()))

        L,R = 0.0, 1.0
        while R-L>1e-5:
            mid=(L+R)/2.0
            if check(mid,a,b,n,k):
                L=mid
            else:
                R=mid
        print(round(100*L))
solve()
```

描述

给定一棵二叉树的前序遍历和中序遍历的结果，求其后序遍历。

输入

输入可能有多组，以EOF结束。 每组输入包含两个字符串，分别为树的前序遍历和中序遍历。每个字符串中只包含大写字母且互不重复。

输出

对于每组输入，用一行来输出它后序遍历结果。

样例输入

```
DBACEGF ABCDEFG
BCAD CBAD
```

样例输出

```
ACBFGED
CDAB
```

```python
def premidtopost(pre:str, mid:str):
    if len(pre) <= 1:
        return pre
    root = pre[0] # 前序表达式的第一个是root
    pos = mid.index(root) # 找到root在中序表达式中的位置。
    lmid = mid[0:pos] # 中序表达式中，在root左边的是左子树的中序表达式。
    lpre = pre[1:pos + 1] # 前序表达式中，root后面紧跟着是左子树的前序表达式，注意到前序表达式的长度应该和中序表达式一样。
    rmid = mid[pos + 1:] # 中序表达式中root右边的是右子树的中序表达式。
    rpre = pre[pos + 1:] # 前序表达式中，左子树的前序表达式之后紧跟着右子树的前序表达式。
    return premidtopost(lpre, lmid) + premidtopost(rpre, rmid) + root # 按照“左，右，根”来返回后序表达式。

while True:
    try:
        a, b = input().split()
    except:
        exit(0) # 读入长度未知的数据。

    print(premidtopost(a, b))
```

描述

Stockbrokers are known to overreact to rumours. You have been contracted to develop a method of spreading disinformation amongst the stockbrokers to give your employer the tactical edge in the stock market. For maximum effect, you have to spread the rumours in the fastest possible way.

Unfortunately for you, stockbrokers only trust information coming from their "Trusted sources" This means you have to take into account the structure of their contacts when starting a rumour. It takes a certain amount of time for a specific stockbroker to pass the rumour on to each of his colleagues. Your task will be to write a program that tells you which stockbroker to choose as your starting point for the rumour, as well as the time it will take for the rumour to spread throughout the stockbroker community. This duration is measured as the time needed for the last person to receive the information.

输入

Your program will input data for different sets of stockbrokers. Each set starts with a line with the number of stockbrokers. Following this is a line for each stockbroker which contains the number of people who they have contact with, who these people are, and the time taken for them to pass the message to each person. The format of each stockbroker line is as follows: The line starts with the number of contacts (n), followed by n pairs of integers, one pair for each contact. Each pair lists first a number referring to the contact (e.g. a '1' means person number one in the set), followed by the time in minutes taken to pass a message to that person. There are no special punctuation symbols or spacing rules.

Each person is numbered 1 through to the number of stockbrokers. The time taken to pass the message on will be between 1 and 10 minutes (inclusive), and the number of contacts will range between 0 and one less than the number of stockbrokers. The number of stockbrokers will range from 1 to 100. The input is terminated by a set of stockbrokers containing 0 (zero) people.

输出

For each set of data, your program must output a single line containing the person who results in the fastest message transmission, and how long before the last person will receive any given message after you give it to this person, measured in integer minutes. It is possible that your program will receive a network of connections that excludes some persons, i.e. some people may be unreachable. If your program detects such a broken network, simply output the message "disjoint". Note that the time taken to pass the message from person A to person B is not necessarily the same as the time taken to pass it from B to A, if such transmission is possible at all.

样例输入

3 2 2 4 3 5 2 1 2 3 6 2 1 2 2 2 5 3 4 4 2 8 5 3 1 5 8 4 1 6 4 10 2 7 5 2 0 2 2 5 1 5 0

样例输出

3 2 3 10

```
    while True :
        n=int(input())
        if n==0 :
            break

        INF=float('inf')
        dist=[[INF]*(n+1) for _ in range(n+1)]
        for i in range(1,n+1) :
            dist[i][i]=0

        for u in range(1,n+1) :
            ipt=list(map(int,input().split()))
            num=ipt[0]

            for i in range(1,num+1) :
                v,w=ipt[2*i-1],ipt[2*i]
                dist[u][v]=w

        for k in range(1,n+1) :
            for i in range(1,n+1) :
                for j in range(1,n+1) :
                    if dist[i][k]+dist[k][j]<dist[i][j] :
                        dist[i][j]=dist[i][k]+dist[k][j]

        min_time=INF
        person=0

        for i in range(1,n+1) :
            person_max_time=max(dist[i][1:])

            if person_max_time<min_time :
                min_time=person_max_time
                person=i

        if min_time==INF :
            print("disjoint")
        else :
            print(person,min_time)
```

描述

The Department of National Defence (DND) wishes to connect several northern outposts by a wireless network. Two different communication technologies are to be used in establishing the network: every outpost will have a radio transceiver and some outposts will in addition have a satellite channel. Any two outposts with a satellite channel can communicate via the satellite, regardless of their location. Otherwise, two outposts can communicate by radio only if the distance between them does not exceed D, which depends of the power of the transceivers. Higher power yields higher D but costs more. Due to purchasing and maintenance considerations, the transceivers at the outposts must be identical; that is, the value of D is the same for every pair of outposts.

Your job is to determine the minimum D required for the transceivers. There must be at least one communication path (direct or indirect) between every pair of outposts.

输入

The first line of input contains N, the number of test cases. The first line of each test case contains 1 <= S <= 100, the number of satellite channels, and S < P <= 500, the number of outposts. P lines follow, giving the (x,y) coordinates of each outpost in km (coordinates are integers between 0 and 10,000).

输出

For each case, output should consist of a single line giving the minimum D required to connect the network. Output should be specified to 2 decimal points.

样例输入

```
1
2 4
0 100
0 300
0 600
150 750
```

样例输出

```
212.13
```

```python
def find_root(node:int):
    global father
    if father[node] == -1:
        return node
    else:
        return find_root(father[node])

def union(a:int, b:int):
    global father, level
    root_a = find_root(a)
    root_b = find_root(b)
    if root_a == root_b:
        # 已经联通
        return False
    else:
        if level[root_a] > level[root_b]:
            father[root_b] = root_a
        elif level[root_a] == level[root_b]:
            father[root_b] = root_a
            level[root_a] += 1
        else:
            father[root_a] = root_b
        return True

for sample in range(n):
    s, p = map(int, input().split())
    site = []
    # 按编号储存的站点
    father = [-1] * p
    level = [1] * p
    for i in range(p):
        x, y = map(int, input().split())
        site.append((x, y))
    distance = []
    for i in range(p):
        for j in range(i + 1, p):
            distance.append(((site[i][0] - site[j][0]) ** 2 + (site[i][1] - site[j][1]) ** 2, i, j))
    distance.sort(key = lambda x: x[0])
    real_dis = []
    # 储存派上用场的无限电收发
    for i in distance:
        d, a, b = i
        # 提高可读性
        if union(a, b):
            # 如果a与b仍未相连
            real_dis.append(d)
            # 每次都能多连上一个
        if len(real_dis) == p-1:
            break
    for i in range(s-1):
        real_dis.pop(-1)
    ans.append(real_dis[-1])
for i in ans:
    print(f'{sqrt(i):.2f}')
```

描述

The system of Martians' blood relations is confusing enough. Actually, Martians bud when they want and where they want. They gather together in different groups, so that a Martian can have one parent as well as ten. Nobody will be surprised by a hundred of children. Martians have got used to this and their style of life seems to them natural. And in the Planetary Council the confusing genealogical system leads to some embarrassment. There meet the worthiest of Martians, and therefore in order to offend nobody in all of the discussions it is used first to give the floor to the old Martians, than to the younger ones and only than to the most young childless assessors. However, the maintenance of this order really is not a trivial task. Not always Martian knows all of his parents (and there's nothing to tell about his grandparents!). But if by a mistake first speak a grandson and only than his young appearing great-grandfather, this is a real scandal. Your task is to write a program, which would define once and for all, an order that would guarantee that every member of the Council takes the floor earlier than each of his descendants.

输入

The first line of the standard input contains an only number N, 1 <= N <= 100 — a number of members of the Martian Planetary Council. According to the centuries-old tradition members of the Council are enumerated with the natural numbers from 1 up to N. Further, there are exactly N lines, moreover, the I-th line contains a list of I-th member's children. The list of children is a sequence of serial numbers of children in a arbitrary order separated by spaces. The list of children may be empty. The list (even if it is empty) ends with 0.

输出

The standard output should contain in its only line a sequence of speakers' numbers, separated by spaces. If several sequences satisfy the conditions of the problem, you are to write to the standard output any of them. At least one such sequence always exists.

样例输入

5 0 4 5 1 0 1 0 5 3 0 3 0

样例输出

2 4 5 3 1

```python
from collections import defaultdict, deque

def topological_sort(n, children):
    # 构建图和计算入度
    graph = defaultdict(list)
    in_degree = [0] * (n + 1)

    for i in range(1, n + 1):
        for child in children[i]:
            graph[i].append(child)
            in_degree[child] += 1

    # 初始化队列，将所有入度为0的节点加入队列
    queue = deque()
    for i in range(1, n + 1):
        if in_degree[i] == 0:
            queue.append(i)

    # 进行拓扑排序
    result = []
    while queue:
        node = queue.popleft()
        result.append(node)
        for child in graph[node]:
            in_degree[child] -= 1
            if in_degree[child] == 0:
                queue.append(child)

    return result

# 输入处理
n = int(input())
children = defaultdict(list)
for i in range(1, n + 1):
    children[i] = list(map(int, input().split()))[:-1]

# 执行拓扑排序
order = topological_sort(n, children)

# 输出结果
print(" ".join(map(str, order)))
```

描述

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ want to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

输入

- Line 1: Two space-separated integers: N and C
- Lines 2..N+1: Line i+1 contains an integer stall location, xi

输出

- Line 1: One integer: the largest minimum distance

样例输入

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

样例输出

```
3
```

提示

OUTPUT DETAILS:

```python
def isvalid(num): # 检查是否是可能的距离值
    global barn
    cnt = 1
    prev = barn[0] # 在x1处放一只牛
    for i in range(len(barn)):
        if barn[i] - prev >= num:
            cnt += 1
            prev = barn[i] # 在放了一只牛之后的第一个距离大于num的stall放另一只牛
        else:
            pass
    if cnt >= c: # 如果放的牛数量超过c，说明是可能的距离值。
        return True
    else:
        return False


n, c = map(int, input().split())
barn = []
for i in range(n):
    barn.append(int(input()))
barn.sort() # 对stall的坐标由小到大排序。
left = 1 # 二分查找，确定牛的间隔的最小值的最大值。
right = barn[-1] // c + 1
while left <= right:
    mid = (left + right) // 2
    if isvalid(mid):
        left = mid + 1
    else:
        right = mid - 1
print(right)
```

描述

有n堆果子（n<=10000），多多决定把所有的果子合成一堆。 每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子数量之和。可以看出，所有的果子经过n-1次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。 设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

输入

两行，第一行是一个整数n(1< = n< =10000)，表示果子的种类数。 第二行包含n个整数，用空格分隔，第i个整数ai(1< = ai< =20000)是第i堆果子的数目。

输出

一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于2^31。

样例输入

```
3
1 2 9
```

样例输出

```
15
```

提示

哈夫曼编码

```python
import heapq

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

heapq.heapify(fruits)
total_energy = 0

while len(fruits) > 1:
    a = heapq.heappop(fruits)
    b = heapq.heappop(fruits)
    energy = a + b
    total_energy += energy
    heapq.heappush(fruits, energy)

print(total_energy)
```

描述

You have just moved from a quiet Waterloo neighbourhood to a big, noisy city. Instead of getting to ride your bike to school every day, you now get to walk and take the subway. Because you don't want to be late for class, you want to know how long it will take you to get to school. You walk at a speed of 10 km/h. The subway travels at 40 km/h. Assume that you are lucky, and whenever you arrive at a subway station, a train is there that you can board immediately. You may get on and off the subway any number of times, and you may switch between different subway lines if you wish. All subway lines go in both directions.

输入

Input consists of the x,y coordinates of your home and your school, followed by specifications of several subway lines. Each subway line consists of the non-negative integer x,y coordinates of each stop on the line, in order. You may assume the subway runs in a straight line between adjacent stops, and the coordinates represent an integral number of metres. Each line has at least two stops. The end of each subway line is followed by the dummy coordinate pair -1,-1. In total there are at most 200 subway stops in the city.

输出

Output is the number of minutes it will take you to get to school, rounded to the nearest minute, taking the fastest route.

样例输入

```
0 0 10000 1000
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1
```

样例输出

```
21
```

```python
    def readCoord():
        l = list(map(int, input().split()))
        return [(l[i], l[i+1]) for i in range(0, len(l), 2)]

    def dist(a, b):
        return math.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)

    class Point:
        def __init__(self, coord, minute=float("inf")):
            self.coord = coord
            self.minute = float(minute)
            self.connTo = {}

    def buildGraph(home, school, subway):
        graph = {home:Point(home), school:Point(school)}
        def addEdge(a, b, v):
            if a is b:
                return
            if a not in graph:
                graph[a] = Point(a)
            if b not in graph:
                graph[b] = Point(b)
            """
            单位换算: m / (km/h) = m / (1000*m/60min) = 0.06min
            """
            minute = dist(a, b) / v * 0.06
            if graph[a] in graph[b].connTo:
                minute = min(minute, graph[b].connTo[graph[a]])
            graph[a].connTo[graph[b]] = graph[b].connTo[graph[a]] = minute

        v0, v1 = 10, 40
        addEdge(home, school, v0)
        for line in subway:
            for i in range(len(line)-1):
                addEdge(line[i], line[i+1], v1)
            for stop in line:
                for vert in graph:
                    if stop is not vert:
                        addEdge(stop, vert, v0)
        return graph

    import heapq
    def dijkstra(graph, home, school):
        graph[home].minute = float(0)
        pq = [(graph[home].minute, home)]
        while pq:
            minute, curr = heapq.heappop(pq)
            if curr == school:
                return
            if minute > graph[curr].minute:
                continue
            for nbr in graph[curr].connTo:
                new_minute = minute + graph[curr].connTo[nbr]
                if new_minute < nbr.minute:
                    nbr.minute = new_minute
                    heapq.heappush(pq, (new_minute, nbr.coord))

    if __name__ == "__main__":
        home, school = readCoord()
        subway = []
        while True:
            try:
                subway.append(readCoord()[:-1])
            except EOFError:
                break
        graph = buildGraph(home, school, subway)
        dijkstra(graph, home, school)
        print(round(graph[school].minute))
```

描述

大学生电影节在北大举办! 这天，在北大各地放了多部电影，给定每部电影的放映时间区间，区间重叠的电影不可能同时看（端点可以重合），问李雷最多可以看多少部电影。

输入

多组数据。每组数据开头是n(n<=100)，表示共n场电影。 接下来n行，每行两个整数(0到1000之间)，表示一场电影的放映区间 n=0则数据结束

输出

对每组数据输出最多能看几部电影

样例输入

```
8
3 4
0 7
3 8
15 19
15 20
10 15
8 18
6 12
0
```

样例输出

```
3
```

来源

Guo Wei

```python
import sys

def main():
    while True:
        line = sys.stdin.readline()
        if not line:
            break
        n = int(line.strip())
        if n == 0:
            break
        intervals = []
        for _ in range(n):
            a, b = map(int, sys.stdin.readline().strip().split())
            intervals.append((a, b))
        # 按结束时间排序
        sorted_intervals = sorted(intervals, key=lambda x: x[1])
        count = 0
        last_end = -1
        for start, end in sorted_intervals:
            if start >= last_end:
                count += 1
                last_end = end
        print(count)

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

25T31:充实的寒假生活

总时间限制: 1000ms 内存限制: 65536kB

描述

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

样例输入

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

样例输出

```
5
```

```python
n = int(input())
activities = []
for _ in range(n):
    s, e = map(int, input().split())
    activities.append((e, s))  # 存储为（结束时间，开始时间）便于排序

# 按结束时间升序排序
activities.sort()

count = 0
last_end = -1  # 初始化上一个活动的结束时间，确保第一个活动可以选

for e, s in activities:
    if s > last_end:
        count += 1
        last_end = e

print(count)
```

描述

一张普通的国际象棋棋盘，它被分成 8 乘 8 (8 行 8 列) 的 64 个方格。设有形状一样的多米诺牌，每张牌恰好覆盖棋盘上相邻的两个方格，即一张多米诺牌是一张 1 行 2 列或者 2 行 1 列的牌。那么，是否能够把 32 张多米诺牌摆放到棋盘上，使得任何两张多米诺牌均不重叠，每张多米诺牌覆盖两个方格，并且棋盘上所有的方格都被覆盖住？我们把这样一种排列称为棋盘被多米诺牌完美覆盖。这是一个简单的排列问题，同学们能够很快构造出许多不同的完美覆盖。但是，计算不同的完美覆盖的总数就不是一件容易的事情了。不过，同学们 发挥自己的聪明才智，还是有可能做到的。 现在我们通过计算机编程对 3 乘 n 棋盘的不同的完美覆盖的总数进行计算。

任务
对 3 乘 n 棋盘的不同的完美覆盖的总数进行计算。

输入

一次输入可能包含多行，每一行分别给出不同的 n 值 (即 3 乘 n 棋盘的列数)。当输入 -1 的时候结束。

n 的值最大不超过 30.

输出

针对每一行的 n 值，输出 3 乘 n 棋盘的不同的完美覆盖的总数。

样例输入

```
2
8
12
-1
```

样例输出

```
3
153
2131
```

```python
import sys
sys.setrecursionlimit(10000)
def f(n):
    if n == 0:
        return 1
    elif n == 2:
        return 3
    else:
        return 4*(f(n-2)) - f(n-4)

while True:
    k = int(input().strip())
    if k == -1:
        exit(0)
    if k % 2 == 1:
        print(0)
    else:
        print(f(k))
```

描述

波兰表达式是一种把运算符前置的算术表达式，例如普通的表达式2 + 3的波兰表示法为+ 2 3。波兰表达式的优点是运算符之间不必有优先级关系，也不必用括号改变运算次序，例如(2 + 3) * 4的波兰表示法为* + 2 3 4。本题求解波兰表达式的值，其中运算符包括+ - * /四个。

输入

输入为一行，其中运算符和运算数之间都用空格分隔，运算数是浮点数。

输出

输出为一行，表达式的值。 可直接用printf("%f\n", v)输出表达式的值v。

样例输入

```
* + 11.0 12.0 + 24.0 35.0
```

样例输出

```
1357.000000
```

提示

可使用atof(str)把字符串转换为一个double类型的浮点数。atof定义在math.h中。 此题可使用函数递归调用的方法求解。

来源

计算概论 05

```python
sign = {'+', '-', '*', '/'}

def cal(a, b, char):
    a, b = float(a), float(b)
    if char == '+':
        return a + b
    elif char == '-':
        return a - b
    elif char == '*':
        return a * b
    else:
        return a / b

def polish(s):
    stack = []
    for token in reversed(s):  # 逆序遍历波兰表达式
        if token in sign:
            a = stack.pop()
            b = stack.pop()
            stack.append(cal(a, b, token))
        else:
            stack.append(token)  # 数字直接入栈
    print(f'{stack[0]:.6f}')  # 保持 6 位小数输出

# 读取输入并拆分
s = input().split()
polish(s)
```

P0070:八皇后问题

总时间限制: 10000ms
内存限制: 65536kB
描述
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
No. 4
1 0 0 0 0 0 0 0
```

```python
broad = [(0, None), (1, None), (2, None), (3, None), (4, None), (5, None), (6, None), (7, None)] # 初始化棋盘
cnt = 1 # 记录现在在产生第几种解法
# oj似乎在处理global的时候会有问题，如果不把global的变量放在函数前面，就会编译错误：
# E0602: Undefined variable 'cnt' (undefined-variable)
def placeQueen(num):
    """
    放皇后并且检查这个子儿是否合法。
    :param num: 已经放了的皇后个数，或者说，由于我们知道每一列只能有一个皇后，这也可以理解为在第几列进行讨论。
    :return: None
    """
    global cnt
    global broad
    if num == 8: # 如果已经放了8个皇后
        output()
        cnt += 1 # 记录的解法数加一
    for i in range(8): # 遍历每一行
        still_ok = True
        for j in range(num): # 检查现在操作的棋子之前放下的每一个棋子与现在操作棋子的位置关系，从而判断放置是否合规。
            if broad[j][1] == i or num - j == abs(broad[j][1] - i): # 在同一行或者在对角线上（不可能在同一列，参见num参数说明）
                still_ok = False
        if still_ok: # 通过检查的话
            broad[num] = (num, i) # 在刚刚讨论过了的位置放下一个棋子。
            placeQueen(num + 1) # 讨论下一列
            broad[num] = (num, None) # 回溯情况，相应列的棋子清空。

def output():
    """
    按照要求输出棋盘
    :return:None
    """
    print(f'No. {cnt}') # 输出解法的序号
    square_broad = [['0'] * 8 for i in range(8)] # 按照要求输出棋盘
    for i in broad:
        square_broad[i[1]][i[0]] = '1' # 有棋子的地方是1
    for i in square_broad:
        print(*i)

placeQueen(0)
```

描述

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

输入

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

```python
from functools import lru_cache
@lru_cache(maxsize=None)
def f(n,m) :
    if n==1 :
        return 1
    else :
        return (f(n-1,m)+m-1)%n+1

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

P0240:全排列

基本信息

总时间限制: 1000ms 内存限制: 65536kB

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
```