Problem A. Asteroids

Input file: `asteroids.in`
Output file: `asteroids.out`

Association of Collision Management (ACM) is planning to perform the controlled collision of two asteroids. The asteroids will be slowly brought together and collided at negligible speed. ACM expects asteroids to get attached to each other and form a stable object.

Each asteroid has the form of a convex polyhedron. To increase the chances of success of the experiment ACM wants to bring asteroids together in such manner that their centers of mass are as close as possible. To achieve this, ACM operators can rotate the asteroids and move them independently before bringing them together.

Help ACM to find out what minimal distance between centers of mass can be achieved.

For the purpose of calculating center of mass both asteroids are considered to have constant density.

Input

Input file contains two descriptions of convex polyhedra.

The first line of each description contains integer number n — the number of vertices of the polyhedron ($4 \le n \le 60$). The following n lines contain three integer numbers x_i, y_i, z_i each — the coordinates of the polyhedron vertices ($-10^4 \le x_i, y_i, z_i \le 10^4$). It is guaranteed that the given points are vertices of a convex polyhedron, in particular no point belongs to the convex hull of other points. Each polyhedron is non-degenerate.

The two given polyhedra have no common points.

Output

Output one floating point number — the minimal distance between centers of mass of the asteroids that can be achieved. Your answer must be accurate up to 10^{-5}.

Sample input and output

asteroids.in	asteroids.out
8 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1 5 0 0 5 1 0 6 -1 0 6 0 1 6 0 -1 6	0.75

Problem B. Business Center

Input file: `business.in`
Output file: `business.out`

International Cyber Police Corporation (ICPC) had built a new mega-tall business center to host its headquarters and to lease some space for extra profit. It has so many floors, that it is impractical to have a separate button in each of its m elevator cars for each individual floor. Instead, each elevator car has just two buttons. One button in i-th elevator car makes it move up u_i floors, the other makes it move down d_i floors. The business center is so high, that we can ignore its height for this problem (you will never reach the top floor), but you cannot go below the ground floor. All floors are numbered by integer numbers starting from zero, zero being the ground floor.

You start on the ground floor of the business center. You have to choose one elevator car out of m to ride on. You cannot switch elevators cars after that. What is the lowest floor above the ground floor you can get to after you press elevator car buttons exactly n times?

Input

The first line of the input file contains two integer numbers n and m ($1 \leq n \leq 1\,000\,000$, $1 \leq m \leq 2\,000$) — the number of button presses and the number of elevator cars to choose from. The following m lines describe elevator cars. Each line contains two integer numbers u_i and d_i ($1 \leq u_i$, $d_i \leq 1\,000$).

Output

Write to the output file a single positive integer number — the number of the lowest floor above ground floor that can be reached by one of m elevators after pressing its buttons exactly n times.

Sample input and output

business.in	business.out
10 3 15 12 15 4 7 12	13

Problem C. Central Element

Input file: `standard input`
Output file: `standard output`

This is an interactive problem.

There is a permutation P of numbers 1 through n, not known to you, $P = \langle P_1, P_2, ..., P_n \rangle$. You can ask the following type of questions: Given three distinct positions i, j and k, which of P_i, P_j and P_k is central? Element is central if it is neither minimal nor maximal.

For example, if the permutation is $\langle 2, 1, 4, 3 \rangle$, and you ask about positions 1, 2, and 3, you receive 2, because 2 is the central element of the set $\{P_1, P_2, P_3\} = \{2, 1, 4\}$. Note that you don't get the information at which position among 1, 2, and 3 it is located.

Your task is to find the permutation P. Actually, for each permutation P there is a set $S(P)$ of permutations that cannot be distinguished from P using the allowed questions. You must find any permutation from this set.

Interaction protocol

First, your program must read from the standard input one line with the integer n, the size of the permutation.

The program must write to the standard output one line with three positions that you ask a question about and wait for a line in the standard input with a response, then write next question and read next response, and so on until you know the permutation P up to $S(P)$.

Once you know the answer, output one line with the word "`OK`" and the permutation P.

Input

The first line of the standard input contains n, the size of the permutation ($3 \le n \le 200$).

Each of the next lines of the standard input contains response to your question — the number that is central among the numbers at the asked positions.

Output

When you're asking questions, each line of the standard output should contain three different integers from the range of 1 to n, space-separated. You can ask at most 2 000 questions.

When you're stating the answer, the line of the standard output should contain the word "`OK`", and the numbers $P_1, P_2, \ldots, P_n$, all space-separated. After printing this line your program must exit.

You must flush standard output after printing each line.

Sample input and output

standard input	standard output
4	1 2 3
2	2 3 4
3	1 2 4
2	1 3 4
3	OK 2 1 4 3

Problem D. Database

Input file: `database.in`
Output file: `database.out`

Peter studies the theory of relational databases. Table in the relational database consists of values that are arranged in rows and columns.

There are different *normal forms* that database may adhere to. Normal forms are designed to minimize the redundancy of data in the database. For example, a database table for a library might have a row for each book and columns for book name, book author, and author's email.

If the same author wrote several books, then this representation is clearly redundant. To formally define this kind of redundancy Peter has introduced his own normal form. A table is in Peter's Normal Form (PNF) if and only if there is no pair of rows and a pair of columns such that the values in the corresponding columns are the same for both rows.

How to compete in ACM ICPC	Peter	peter@neerc.ifmo.ru
How to win ACM ICPC	Michael	michael@neerc.ifmo.ru
Notes from ACM ICPC champion	Michael	michael@neerc.ifmo.ru

The above table is clearly not in PNF, since values for 2rd and 3rd columns repeat in 2nd and 3rd rows. However, if we introduce unique author identifier and split this table into two tables — one containing book name and author id, and the other containing book id, author name, and author email, then both resulting tables will be in PNF.

How to compete in ACM ICPC	1
How to win ACM ICPC	2
Notes from ACM ICPC champion	2

1	Peter	peter@neerc.ifmo.ru
2	Michael	michael@neerc.ifmo.ru

Given a table your task is to figure out whether it is in PNF or not.

Input

The first line of the input file contains two integer numbers n and m ($1 \le n \le 10\,000$, $1 \le m \le 10$), the number of rows and columns in the table. The following n lines contain table rows. Each row has m column values separated by commas. Column values consist of ASCII characters from space (ASCII code 32) to tilde (ASCII code 126) with the exception of comma (ASCII code 44). Values are not empty and have no leading and trailing spaces. Each row has at most 80 characters (including separating commas).

Output

If the table is in PNF write to the output file a single word "`YES`" (without quotes). If the table is not in PNF, then write three lines. On the first line write a single word "`NO`" (without quotes). On the second line write two integer row numbers r_1 and r_2 ($1 \le r_1, r_2 \le n$, $r_1 \ne r_2$), on the third line write two integer column numbers c_1 and c_2 ($1 \le c_1, c_2 \le m$, $c_1 \ne c_2$), so that values in columns c_1 and c_2 are the same in rows r_1 and r_2.

Sample input and output

database.in	database.out
3 3 How to compete in ACM ICPC,Peter,peter@neerc.ifmo.ru How to win ACM ICPC,Michael,michael@neerc.ifmo.ru Notes from ACM ICPC champion,Michael,michael@neerc.ifmo.ru	NO 2 3 2 3
2 3 1,Peter,peter@neerc.ifmo.ru 2,Michael,michael@neerc.ifmo.ru	YES

Problem E. Exclusive Access 2

Input file: `exclusive.in`
Output file: `exclusive.out`

Having studied mutual exclusion protocols in the previous year's competition you are now facing a more challenging problem. You have a big enterprise system with a number concurrently running processes. The system has several resources — databases, message queues, etc. Each concurrent process works with two resources at a time. For example, one process might copy a job from a particular database into the message queue, the other process might take a job from the message queue, perform the job, and then put the result into some other message queue, etc.

All resources are protected from concurrent access by mutual exclusion protocols also known as *locks*. For example, to access a particular database process acquires the lock for this database, then performs its work, then releases the lock. No two processes can hold the same lock at the same time (that is the property of mutual exclusion). Thus, the process that tries to acquire a lock *waits* if that lock is taken by some other process.

The main loop of the process that works with resources P and Q looks like this:

```
loop forever
  DoSomeNonCriticalWork()
  P.lock()
  Q.lock()
  WorkWithResourcesPandQ()
  Q.unlock()
  P.unlock()
end loop
```

The order in which locks for resources P and Q are taken is important. Consider a case where process c had acquired lock P with `P.lock()` and is waiting for lock Q in `Q.lock()`. It means that lock Q is taken by some other process d. If the process d is working (not waiting), then we say that there is a *wait chain* of length 1. If d had acquired lock Q and is waiting for another lock R, which is acquired by a working process e, then we say that there is a *wait chain* of length 2, etc. If any process in this wait chain waits for lock P that is already taken by process c, then we say that the wait chain has infinite length and the system *deadlocks*.

For this problem, we are interested only in alternating wait chains where processes hold their first locks and wait for the second ones. Formally:

> *Alternating wait chain* of length n ($n \geq 0$) is an alternating sequence of resources R_i ($0 \leq i \leq n+1$) and distinct processes c_i ($0 \leq i \leq n$): R_0 c_0 R_1 c_1 ... R_n c_n R_{n+1}, where process c_i acquires locks for resources R_i and R_{i+1} in this order. Alternating wait chain is a deadlock when $R_0 = R_{n+1}$.

You are given a set of resources each process works with. Your task is to decide the order in which each process has to acquire its resource locks, so that the system never deadlocks and the maximum length of any possible alternating wait chain is minimized.

Input

The first line of the input file contains a single integer n ($1 \leq n \leq 100$) — the number of processes.

The following n lines describe resources that each process needs. Each resource is designated with an uppercase English letter from `L` to `Z`, so there are at most 15 resources. Each line describing process contains two different resources separated by a space.

Output

On first line of the output file write a singe integer number m — the minimally possible length of the maximal alternating wait chain.

Then write n lines — one line per process. On each line write two resources in the order they should be taken by the corresponding process to ensure this minimal length of the maximal alternating wait chain. Separate resources on a line by a space. If there are multiple satisfying orderings, then write any of them. The order of the processes in the output should correspond to their order in the input.

Sample input and output

exclusive.in	exclusive.out
2 P Q R S	0 P Q R S
6 P Q Q R R S S T T U U P	0 P Q R Q R S T S T U P U
4 P Q P Q P Q P Q	0 P Q P Q P Q P Q
3 P Q Q R R P	1 P Q Q R P R
6 P Q Q S S R R P P S R Q	2 P Q Q S R S P R P S R Q

Problem F. Funny Language

Input file: `funny.in`
Output file: `funny.out`

There is a well know game with words. Given a word you have to write as many other words as possible using the letters from the given word. If the letter repeats multiple times in the original word, you can use it up to as many times in the new words. The order of letters in the original word does not matter. For example, given the word CONTEST you can write NOTE, NET, ON, TEST, SET, etc.

Now you are in charge of writing a new dictionary. Your task is to sneak n new words into it. You know in advance m words W_i $(1 \le i \le m)$ that you will have to play a game with and you need to figure out which new n words to add to the dictionary to maximize the total number of words you can write out of these m words.

More formally, find such a set of nonempty words S where $|S| = n$, $W_i \notin S$ for any i, and $\sum_{1 \le i \le m} |S_i|$ is maximal, where $S_i \subset S$ is the set of words that can be written using letters from W_i.

Input

The first line of the input file contains two integer numbers n $(1 \le n \le 100)$ — the number of new words you can add to the dictionary and m $(1 \le m \le 1\,000)$ — the number of words you will play the game with. The following m lines contain original words. Each word consists of at most 100 uppercase letters from A to Z.

Output

Write to the output file n lines with a new word on a line.

Sample input and output

funny.in	funny.out
3 5 A ACM ICPC CONTEST NEERC	C CN E

Problem G. Garbling Game

Input file:	`garbling.in`
Output file:	`garbling.out`

Pavel had invented a new game with a matrix of integer numbers. He takes $r \times c$ matrix with r rows and c columns that is filled with numbers from 1 to rc left to right and top to bottom (1 is written in the upper-left corner, rc is written in the lower-right corner). Then he starts to rearrange the numbers is the matrix by following the rules that are explained below and writes down a sequence of numbers on a separate piece of paper. He calls it *garbling* of the matrix.

The rules of rearrangement are defined by *garbling map* that is $(r-1) \times (c-1)$ matrix of letters L, R, and N. Initial 4×5 matrix and the sample 3×4 garbling map for it are shown below.

(1)	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

L	R	L	R
N	L	L	R
L	N	N	L

Pavel garbles the matrix in a series of turns. On his first turn Pavel takes the number in the first row and the first column (it is put in parenthesis on the above picture for clarity) and writes it down.

Having written down the number he performs one *garbling turn*:

Pavel looks at the letter in the garbling map that corresponds to the position of the number he had just written down (one the first turn it is the letter in the upper-left corner). Depending on the letter in the garbling map the 2×2 block of the matrix whose upper-left corner contains the number he had just written (highlighted in the above picture) is rearranged in the following way:

- R — the block is rotated *clockwise*.
- L — the block is rotated *counterclockwise*.
- N — Pavel does not change the matrix on this turn.

On the second turn Pavel takes the number in the first row and second column, writes it down, and performs the garbling turn, and so on. In $c-1$ turns he finishes the first row and moves to the second row and so on he proceeds left to right and top to bottom. In $(r-1)(c-1)$ turns he had written down $(r-1)(c-1)$ numbers and garbled the whole matrix, so he starts again in the upper-left corner continuing garbling the matrix from left to right and top to bottom.

The matrices below show the effect of the first four turns with the sample garbling map.

2	(7)	3	4	5
1	6	8	9	10
11	12	13	14	15
16	17	18	19	20

2	6	(7)	4	5
1	8	3	9	10
11	12	13	14	15
16	17	18	19	20

2	6	4	(9)	5
1	8	7	3	10
11	12	13	14	15
16	17	18	19	20

2	6	4	3	9
(1)	8	7	10	5
11	12	13	14	15
16	17	18	19	20

The following sequence of numbers is written down in the first 4 turns: `1 7 7 9`. On 5th turn the number from the second row and the first column is written, but the matrix remains unchanged, since the second row and the first column of the garbling map contains N. In six turns Pavel gets `1 7 7 9 1 8`.

Given the garbling map and the number of moves Pavel makes in this game, find out how many times each number gets written down by Pavel. You need to provide the answer modulo 10^5.

Input

The first line of the input file contains three integer numbers — r, c, and n, where r, c ($2 \le r,\ c \le 300$) are the dimensions of the initial matrix, n ($0 \le n < 10^{100}$) is the number of turns Pavel makes.

The following $r-1$ lines contain garbling map with $c-1$ characters R, L, or N on a line.

Output

Write to the output file rc lines with one integer number per line. On i-th line write the number of times number i gets written down by Pavel modulo 10^5 while he makes his n turns.

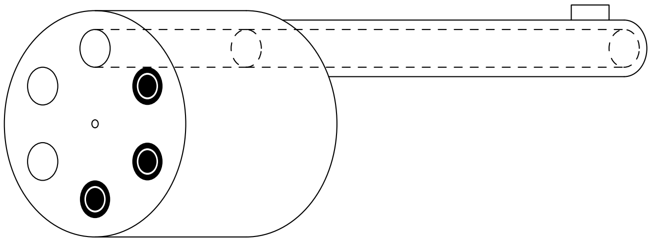
Sample input and output

garbling.in	garbling.out
4 5 6 LRLR NLLR LNNL	2 0 0 0 0 0 2 1 1 0 0 0 0 0 0 0 0 0 0 0
4 5 666666 LRLR NLLR LNNL	37038 37038 0 0 30864 37036 11112 30864 30864 30864 30864 30864 11110 30865 18519 30864 30864 0 18518 18518

Problem H. Headshot

Input file: `headshot.in`
Output file: `headshot.out`

You have a revolver gun with a cylinder that has n chambers. Chambers are located in a circle on a cylinder. Each chamber can be empty or can contain a round. One chamber is aligned with the gun's barrel. When trigger of the gun is pulled, the gun's cylinder rotates, aligning the next chamber with the barrel, hammer strikes the round, making a shot by firing a bullet through the barrel. If the chamber is empty when the hammer strikes it, then there is no shot but just a "click".



You have found a use for this gun. You are playing Russian Roulette with your friend. Your friend loads rounds into some chambers, randomly rotates the cylinder, aligning a random chamber with a gun's barrel, puts the gun to his head and pulls the trigger. You hear "click" and nothing else — the chamber was empty and the gun did not shoot.

Now it is your turn to put the gun to your head and pull the trigger. You have a choice. You can either pull the trigger right away or you can randomly rotate the gun's cylinder and then pull the trigger. What should you choose to maximize the chances of your survival?

Input

The input file contains a single line with a string of n digits "0" and "1" ($2 \leq n \leq 100$). This line of digits represents the pattern of rounds that were loaded into the gun's chambers. "0" represent an empty chamber, "1" represent a loaded one. In this representation, when cylinder rotates before a shot, the next chamber to the right gets aligned with the barrel for a shot. Since the chambers are actually located on a circle, the first chamber in this string follows the last one. There is at least one "0" in this string.

Output

Write to the output file one of the following words (without quotes):

- "`SHOOT`" — if pulling the trigger right away makes you less likely to be actually shot in the head with the bullet (more likely that the chamber will be empty).
- "`ROTATE`" — if randomly rotating the cylinder before pulling the trigger makes you less likely to be actually shot in the head with the bullet (more likely that the chamber will be empty).
- "`EQUAL`" — if both of the above choices are equal in terms of probability of being shot.

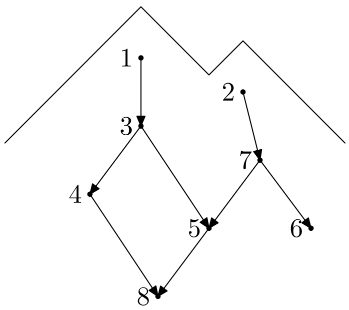
Sample input and output

headshot.in	headshot.out
0011	EQUAL
0111	ROTATE
000111	SHOOT

Problem I. Inspection

Input file: `inspection.in`
Output file: `inspection.out`

You are in charge of a team that inspects a new ski resort. A ski resort is situated on several mountains and consists of a number of slopes. Slopes are connected with each other, forking and joining. A map of the ski resort is represented as an acyclic directed graph. Nodes of the graph represent different points in ski resort and edges of the graph represent slopes between the points, with the direction of edges going downwards.



Your team has to inspect each slope of the ski resort. Ski lifts on this resort are not open yet, but you have a helicopter. In one flight the helicopter can drop one person into any point of the resort. From the drop off point the person can ski down the slopes, inspecting each slope as they ski. It is fine to inspect the same slope multiple times, but you have to minimize the usage of the helicopter. So, you have to figure out how to inspect all the slopes with the fewest number of helicopter flights.

Input

The first line of the input file contains a single integer number n ($2 \le n \le 100$) — the number of points in the ski resort. The following n lines of the input file describe each point of the ski resort numbered from 1 to n. Each line starts with a single integer number m_i ($0 \le m_i < n$ for i from 1 to n) and is followed by m_i integer numbers a_{ij} separated by spaces. All a_{ij} are distinct for each i and each a_{ij} ($1 \le a_{ij} \le n$, $a_{ij} \ne i$) represents a slope going downwards from point i to point a_{ij}. Each point in the resort has at least one slope connected to it.

Output

On the first line of the output file write a single integer number k — the minimal number of helicopter flights that are needed to inspect all slopes. Then write k lines that describe inspection routes for each helicopter flight. Each route shall start with single integer number from 1 to n — the number of the drop off point for the helicopter flight, followed by the numbers of points that will be visited during inspection in the corresponding order as the slopes are inspected going downwards. Numbers on a line shall be separated by spaces. You can write routes in any order.

Sample input and output

inspection.in	inspection.out
8 1 3 1 7 2 4 5 1 8 1 8 0 2 6 5 0	4 1 3 4 8 1 3 5 8 2 7 6 7 5

Problem J. Java Certification

Input file: `javacert.in`
Output file: `javacert.out`

You have just completed Java Certification exam that contained n questions. You have a score card that explains your performance. The example of the scorecard is given below.

You have correctly answered 78 questions out of 87.

Basic Concepts	100%
Declarations	100%
Expressions	83%
Classes and Interfaces	92%
Multithreading	75%
Collections	93%

From this scorecard you can infer that the questions are broken down into m categories (in the above example $m = 6$). Each category contains n_i questions ($1 \le n_i \le n$), so that $\sum_{1 \le i \le m} n_i = n$. You know that you have correctly answered k questions out of n (in the above example $k = 78$ and $n = 87$), so you can easily find the number of incorrect answers $w = n - k$ (in the above example $w = 9$).

You do remember several questions that you were unsure about and you can guess what category they belong to. To figure out if your answers on those questions were right or wrong, you really want to know how many incorrect answers you have given in each category.

Let w_i ($0 \le w_i \le n_i$) be the number of *incorrect* answers in i-th category, $\sum_{1 \le i \le m} w_i = w$. From the scorecard you know the *percentage of correct answers* in each category. That is, for each i from 1 to m you know the value of $100(n_i - w_i)/n_i$ rounded to the nearest integer. The value with a fractional part of 0.5 is rounded to the nearest even integer.

It may not be possible to uniquely find the valid values for w_i. However, you guess that the questions are broken down into categories in a mostly uniform manner. You have to find the valid values of w_i and n_i, so that to minimize the difference between the maximum value of n_i and the minimum value of n_i. If there are still multiple possible values for w_i and n_i, then find any of them.

Input

The first line of the input file contains three integer numbers — k, n, and m, where k ($0 \le k \le n$) is the number of correctly answered questions, n ($1 \le n \le 100$) is the total number of questions, m ($1 \le m \le 10$) is the number of question categories. The following m lines of the input file contain one integer number from 0 to 100 (inclusive) on a line — percentages of the number of the correct answers in each category. The input file always corresponds to some valid set of w_i and n_i.

Output

Write to the output file m lines with two integer numbers w_i and n_i on a line, separated by a space — the number of incorrect answers and the total number of questions in each category, satisfying constraints as given in the problem statement.

Sample input and output

<table>
<tr><th>javacert.in</th><th>javacert.out</th></tr>
<tr><td><pre>78 87 6
100
100
83
92
75
93</pre></td><td><pre>0 13
0 13
3 18
1 13
4 16
1 14</pre></td></tr>
</table>

Problem K. K-equivalence

Input file:	`kequiv.in`
Output file:	`kequiv.out`

Consider a set K of positive integers.

Let p and q be two non-zero decimal digits. Call them K-equivalent if the following condition applies:

> For every $n \in K$, if you replace one digit p with q or one digit q with p in the decimal notation of n then the resulting number will be an element of K.

For example, when K is the set of integers divisible by 3, the digits 1, 4, and 7 are K-equivalent. Indeed, replacing a 1 with a 4 in the decimal notation of a number never changes its divisibility by 3.

It can be seen that K-equivalence is an equivalence relation (it is reflexive, symmetric and transitive).

You are given a finite set K in form of a union of disjoint finite intervals of positive integers.

Your task is to find the equivalence classes of digits 1 to 9.

Input

The first line contains n, the number of intervals composing the set K ($1 \le n \le 10\,000$).

Each of the next n lines contains two positive integers a_i and b_i that describe the interval $[a_i, b_i]$ (i. e. the set of positive integers between a_i and b_i, inclusive), where $1 \le a_i \le b_i \le 10^{18}$. Also, for $i \in [2..n]$: $a_i \ge b_{i-1} + 2$.

Output

Represent each equivalence class as a concatenation of its elements, in ascending order.

Output all the equivalence classes of digits 1 to 9, one at a line, sorted lexicographically.

Sample input and output

kequiv.in	kequiv.out
1 1 566	1234 5 6 789
1 30 75	12 345 6 7 89