Problem A. Ackerman's Function

Input file:	`ackerman.in`
Output file:	`ackerman.out`
Time limit:	1 second
Memory limit:	64 megabytes

Ackerman's function is well known to all specialists in the theory of computation. It is the function in two positive integer arguments defined as follows:

$$A(n,m) = \begin{cases} 2m, & \text{if } n = 1 \\ 2, & \text{if } n > 1,\ m = 1 \\ A(n-1, A(n, m-1)), & \text{if } n > 1,\ m > 1 \end{cases}$$

It is not primitive recursive, more of that, $A(i,i)$ grows faster than any primitive recursive function.

In this problem your task is to calculate

$$A(n,m) \bmod t$$

for given t and several n and m. Here "$x \bmod y$" means the remainder of integer division of x by y — such r that $0 \le r < y$ and there exists integer q, such that $x = qy + r$.

Input

The first line of the input file contains t ($1 \le t \le 100$).

Several lines follow. Each of them contains one testcase — n and m ($1 \le m, n \le 100$).

The last line of the input file contains two zeroes, it should not be processed.

Output

For each testcase output its number followed by $A(n,m) \bmod t$. Use format shown in the sample output.

Example

ackerman.in	ackerman.out

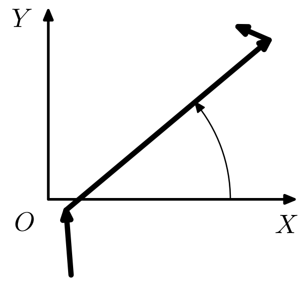
3
3 3
2 7
1 18
0 0
	Case 1: 1
	Case 2: 2
	Case 3: 0

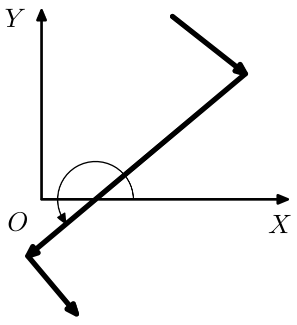
Problem B. The Minimal Angle

Input file:	`angle.in`
Output file:	`angle.out`
Time limit:	1 second
Memory limit:	64 megabytes

Peter has a sheet of paper with a polyline drawn on it. He wants to rotate this sheet to minimize the total angle of the line.

For each segment of the polyline Peter considers a directed counter-clockwise angle in range from 0 to 2π formed by OX axis and this segment. Peter calls the total angle of the polyline the sum of angles for all segments, taken modulo 2π.





For simplicity, the sheet must be rotated only relatively to the point $(0, 0)$.

Input

The input consists of one or more test cases. Each case starts with a line containing N ($1 \le N \le 100$). N lines follow, each contains integer coordinates of a point — x and y. They do not exceed 10000 by an absolute value. The polyline may have self-intersections or self-touchings. Polyline contains $N - 1$ segments, i-th one is directed from point i to point $i + 1$, no segment has zero length.

The input is terminated by a case with $N = 0$. The total sum of all Ns in the input does not exceed 30000.

Output

For each case output the minimal possible total angle on the first line. The angle must be in range from 0 to 2π. The second line must contain a counter-clockwise rotation angle for the polyline needed to obtain this minimal value. Next N lines must contain coordinates of the points after the rotation. Separate output for subsequent cases with an empty line.

Note that range from 0 to 2π for angles means that 0 is included, but 2π is not included.

Example

`angle.in`	`angle.out`

`2`
`0 0`
`1 1`
`0`
	`0.00000000000000000000`
	`5.49778714378213817000`
	`0.00000000000000000000 0.00000000000000000000`
	`1.41421356237309505000 -0.00000000000000000000`

Problem C. Yellow Code

Input file:	`code.in`
Output file:	`code.out`
Time limit:	1 second
Memory limit:	64 megabytes

Inspired by Gray code, professor John Yellow has invented his own code. Unlike in Gray code, in Yellow code the adjacent words have many different bits.

More precisely, for $s = 2^n$ we call the permutation $a_1, a_2, \ldots, a_s$ of all n-bit binary words the *Yellow code* if for all $1 \le k < s$ words a_k and a_{k+1} have at least $\lfloor n/2 \rfloor$ different bits and a_1 and a_s also have at least $\lfloor n/2 \rfloor$ different bits.

Given n you have to find the n-bit Yellow code or detect that there is none.

Input

Input file contains the number n ($2 \le n \le 12$).

Output

Output 2^n n-bit binary vectors in the order they appear in some n-bit Yellow code, one on a line. If there is no n-bit Yellow code, output "`none`" on the first line of the output file.

Example

code.in	code.out
4	0000 1111 0001 1110 0010 1101 0011 1100 0101 1011 0100 1010 0110 1000 0111 1001

Problem D. Yet Another Digit

Input file:	`digit.in`
Output file:	`digit.out`
Time limit:	1 second
Memory limit:	64 megabytes

When making different calculations we usually use decimal notation. On the other side, computers usually store and process information in binary form. Both decimal and binary notations are special cases of so called *base notations*. Base notation is the way to represent integer number as the sum of powers of some integer number B called *the base* of the notation, taken with some coefficients called *digits*. Thus the weight of the i-th digit is proportional to B^i (digits being numbered from zero).

Most base notations that we use have a nice property — for any particular number the value of each digit is defined unambiguously. For example, consider the number 5. In binary notation the rightmost digit is 1, the next one is 0, the second digit is 1 and all the others are 0.

However, this property just follows from the fact that the number of different values for digits is equal to the base of the notation. If we allow more different values for each digit, the uniqueness of representation would be lost. Consider *redundant* binary notation, where three digits are allowed: 0, 1 and 2. In this notation some particular number may have several representations. For example, 5 can be represented as both 101 and 21.

Find out the number of different representations of the given number R in redundant binary notation.

Input

Input file contains one integer number R ($0 \le R \le 10^{100}$).

Output

Output one integer number — the number of ways R can can be represented in redundant binary notation.

Example

`digit.in`	`digit.out`
5	2
7	1

Problem E. Graduated Lexicographical Ordering

Input file:	`grlex.in`
Output file:	`grlex.out`
Time limit:	1 second
Memory limit:	64 megabytes

Consider integer numbers from 1 to n. Let us call the sum of digits of an integer number its *weight*. Denote the weight of the number x as $w(x)$.

Now let us order the numbers using so called *graduated lexicographical ordering*, or shorter *grlex ordering*. Consider two integer numbers a and b. If $w(a) < w(b)$ then a goes before b in grlex ordering. If $w(a) = w(b)$ then a goes before b in grlex ordering if and only if the decimal representation of a is lexicographically smaller than the decimal representation of b.

Let us consider some examples.

$120 <_{grlex} 4$ since $w(120) = 1 + 2 + 0 = 3 < 4 = w(4)$.

$555 <_{grlex} 78$ since $w(555) = 15 = w(78)$ and "`555`" is lexicographicaly smaller than "`78`".

$20 <_{grlex} 200$ since $w(20) = 2 = w(200)$ and "`20`" is lexicographicaly smaller than "`200`".

Given n and some integer number k, find the position of the number k in grlex ordering of integer numbers from 1 to n, and the k-th number in this ordering.

Input

Input file contains n and k ($1 \le k \le n \le 10^{18}$).

Output

On the first line print the position of the number k in grlex ordering of integer numbers from 1 to n. On the second line print the integer number that occupies the k-th position in this ordering.

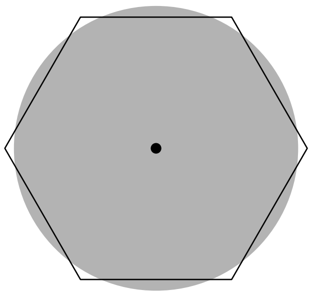
Example

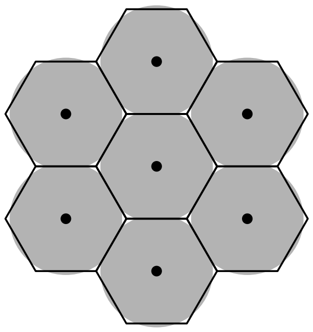
grlex.in	grlex.out
20 10	2 14

Problem F. GSM

Input file:	`gsm.in`
Output file:	`gsm.out`
Time limit:	1 second
Memory limit:	64 megabytes

The company “Gigafon” has recently decided to build cellular network in Flatland. It is well known that the best way to plan the network is to put base stations in the centers of the cells of regular hexagonal grid as shown on the picture.





“Gigafon” has selected the side of the hexagon and the range of each base station. Now the company owners want to know which part of the Flatland will be covered. To estimate this value they ask you to find what part of each hexagon is covered by the network.

You know that business people are very anxious about their money, so they want the answer to be very precise.

Input

Input file contains two integer numbers H and R — the side of the hexagon and the range of each base station ($1 \le H, R \le 50$).

Output

Output the ratio of the area of hexagon covered by the network to the area of the hexagon itself. Your answer must be accurate up to one hundred digits after the decimal point.

Example

`gsm.in`
`50 47`
`gsm.out`
`0.98451302979698180310018085619028469741866686583424668241728873534676912 670242017381769233473449336186969315064`

Sample output contains 110 digits after the decimal point. It is satisfactory to output exactly 100 digits. Your answer will be accepted if it differs from the correct answer by no more than 10^{-100}.

Your answer must be contained within a single line.

Problem G. Warehouse Keeper

Input file:	`keeper.in`
Output file:	`keeper.out`
Time limit:	1 second
Memory limit:	64 megabytes

The company where Jerry works owns a number of warehouses that can be used to store various goods. For each warehouse the types of goods that can be stored in this warehouse are known. To avoid problems with taxes, each warehouse must store only one type of goods, and each type of goods must be stored in at most one warehouse.

Jerry is planning to receive a new lot of goods in a couple of days and he must store the goods in the warehouses. However there are some goods in some warehouses already and Jerry wants to move as few of them as possible.

Help him to find the maximal number of types of goods that he can store in the warehouses and the minimal number of goods he must move in order to do that.

Input

The first line of the input file contains integer numbers m and n ($2 \le m, n \le 200$) — the number of warehouses and the number of types of goods respectively.

The following m lines describe warehouses. Each line contains k_i — the number of various types of goods that can be stored in this warehouse (remember, only one type of goods can be stored in a warehouse at a time), followed by k_i integer numbers — the types of goods that can be stored.

The last line contains m integer numbers — for each warehouse either 0 is provided if there is no goods in this warehouse, or the type of goods that is currently stored in this warehouse if there is one. It is guaranteed that the initial configuration is correct, that is, each warehouse stores the goods it can store, and no type of goods is stored in more than one warehouse.

Output

On the first line of the output file print p — the maximal number of types of goods that can be stored in the warehouses, and q — the minimal number of goods that need to be moved in order to do that. After that output m integer numbers — for each warehouse output the type of goods that must be stored in this warehouse, or 0 if none must be.

Remember that you may only move goods that are already stored in some houses to other ones, you are not allowed to dispose them.

Example

keeper.in	keeper.out

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

Problem H. Don't Go Left

Input file:	`left.in`
Output file:	`left.out`
Time limit:	1 second
Memory limit:	64 megabytes

Turing Machine (TM) is a well known abstraction, used in theoretical computer science. TM has a tape alphabet Σ and a finite set U of states. TM has an access to the tape, infinite in both directions. Tape is accessed with the head that can move along it, read and write characters to it. The tape is initially filled with blanks ($B \in \Sigma$) and contains the input word $\omega \in (\Sigma \setminus \{B\})^*$.

The *transition function* of TM is the function $\phi : U \times \Sigma \to U \times \Sigma \times \{\leftarrow, \rightarrow, \downarrow\}$. TM acts in the following way. Let its states be numbered from 1 to u where $u = |U|$. Let 1 be the initial and u be the terminal state. Initially TM is in the initial state and its head points to the first character of the input word ω.

Let TM be in a state q and its head point to the character c. If q is the terminal state, it stops and is said to *accept* the word ω. Let q be non-terminal state. If $\phi(q, c) = \langle r, d, \alpha \rangle$, TM changes its state to r, writes d on the input tape instead of c and moves its head in the direction α. That is, if α is $\leftarrow$, it moves its head to the previous character of the tape, if its is $\rightarrow$, it moves its head to the next character and if it is $\downarrow$, it does not move its head.

TM is called *consecutive* if whatever the input word ω is, it never moves its head to the left, that is, it never moves it to the previous character on the tape.

In this problem you have to determine whether the given TM is consecutive.

Input

The first line of the input file contains u — the number of states of the given TM and $s = |\Sigma|$ — the number of characters in its tape alphabet ($2 \le u \le 100$, $2 \le s \le 100$). Denote the characters from Σ by $c_1, c_2, \ldots, c_s$. We will consider that $B = c_s$, that is — blank is the last character of the alphabet.

Next $(u - 1) \cdot s$ lines describe ϕ. The line of the input file with the number $1 + (i - 1) \cdot s + j$ describes $\phi(i, c_j)$. Each line contains three integer numbers: k ($1 \le k \le u$) — the new state of TM, l ($1 \le l \le s$) — the character c_l is written on the tape and α ($-1 \le \alpha \le 1$) — the direction of the move (-1 stands for $\leftarrow$, 0 for $\downarrow$, and 1 for $\rightarrow$). The last state is accepting, so the transitions from it are of no matter, therefore they are not described.

Remember, that the tape is initially filled with blanks and blank never occurs inside the input word. However, the input word may be empty, this is the only case when the head of TM initially points to a blank.

Output

Print "`YES`" if the TM given is consecutive and "`NO`" if it is not.

Example

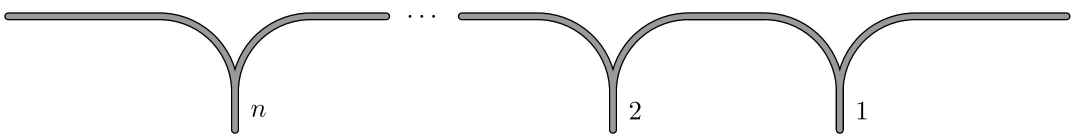
left.in	left.out

4 3
1 1 1
2 2 0
3 3 1
2 1 -1
2 2 0
2 3 0
3 1 1
4 3 1
2 3 0
	YES

Problem I. Railroad Sort

Input file:	`railsort.in`
Output file:	`railsort.out`
Time limit:	1 second
Memory limit:	64 megabytes

Consider the railroad station that has n dead-ends designed in a way shown on the picture. Dead-ends are numbered from right to left, starting from 1.



Let 2^n railroad cars get from the right. Each car is marked with some integer number ranging from 1 to 2^n, different cars are marked with different numbers.

You can move the cars through the dead-ends using the following two operations. If the car x is the first car on the path to the right of the dead-end i, you may move this car to this dead-end. If the car y is the topmost car in the dead-end j you can move it to the path on the left of the dead-end. Note, that cars cannot be moved to the dead-end from the path to its left and cannot be moved to the path on the right of the dead-end they are in.

Your task is to rearrange the cars so that the numbers on the cars listed from left to right were in the ascending order and all the cars are to the left of all the dead-ends.

One can prove that the required rearranging is always possible.

Input

The first line of the input file contains n — the number of dead-ends ($1 \le n \le 13$). The second line contains 2^n integer numbers — the numbers on the cars, listed from left to right.

Output

Output the sequence of operations. Each operation is identified with the number of the car moved in this operation. The type of the operation and the dead-end used are clearly determined uniquely.

Example

`railsort.in`	`railsort.out`

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

The sequence of the operations in the example is the following: first we move all cars through dead-end 1 without changing their order, after that we put cars 3, 2 and 1 to the dead-end 2 and take them out of it, changing their order to the reverse. Finally we move the car 4 through the dead-end 2.