Problem A. Bermutation

Input file:	`bermutation.in`
Output file:	*`standard output`*
Time limit:	1 second
Memory limit:	256 mebibytes

Your are given a permutation p of integers from 1 to n. You are allowed to modify this permutation in the following way: choose a segment of $2b$ consecutive elements and swap the halves of this segment. Formally, if you choose the segment $a_i, a_{i+1} \ldots, a_{i+2b-1}$, you will get $a_{i+b}, a_{i+b+1}, \ldots, a_{i+2b-1}, a_i, a_{i+1}, \ldots, a_{i+b-1}$ after swapping its halves.

Consider the set S of all permutations which can be obtained from the given permutation p by applying this modification zero or more times. The segment of $2b$ consecutive elements for each modification can be chosen independently of the segments chosen for other modifications. List all these permutations in lexicographical order and enumerate them starting from 1. Your task is to find the number of p itself in this ordered list. Print the answer modulo 120 586 241.

Input

The first line of input contains one positive integer T, the number of test cases. The test cases follow.

Each test case is given on two lines. The first line contains with two integers n and b ($2 \le n \le 10^5$, $1 \le b$ and $2b \le n$). The second line of each test case contains n integers: the permutation p. Each integer from 1 to n appears on this line exactly once.

The sum of all n in the input does not exceed 10^5.

Output

Print the 1-based number of permutation p in the lexicographically ordered list of all permutations which can be obtained by the described modifications, modulo 120 586 241.

Example

bermutation.in	standard output

2
5 1
2 3 1 4 5
5 2
2 3 1 4 5
	31
	3

Problem B. Grid

Input file:	`grid.in`
Output file:	*standard output*
Time limit:	1.5 seconds
Memory limit:	256 mebibytes

There is a city square in one country which looks exactly like a rectangular $N \times M$ grid. The government decided to hold a meeting of its supporters, so right now, the city square is full of people. More precisely, there is exactly one person in each cell of the grid.

Each person looks in one of the four cardinal directions: notrh, east, south, or west. But when two neighboring people look directly at each other, they get embarrassed because they know that they came to the meeting only to get paid.

The government is well aware of the situation, and so it doesn't want any pair of people to get embarrassed, because if there are enough people who understand that they came for a wrong reason, they can start a riot. Still, the meeting participants are greedy, so they will gladly turn 90 degrees in any desired direction any time they get paid 1 turning coin. Each person can be paid and turned several times.

The government wants to turn people in such way that in the resulting position, no two persons get embarrassed by looking directly at each other. Your task is to achieve that with the minimum amount of money spent.

Input

The first line of input contains two integers N and M, the dimensions of the grid ($1 \le N, M \le 50$). Each of the next N lines describes one row of the grid. Each of these lines contains M characters denoting the direction in which the persons are looking. Each of the characters is one of "`^`" (for notrh), "`>`" (for east), "`v`" (for south), or "`<`" (for west).

Output

Print the minimum amount of turning coins the government has to spend to avoid people getting embarrassed.

Examples

grid.in	standard output

`3 3`
`>v<`
`>^<`
`^^^`
	`2`

`1 10`
`>>><<<>><<`
	`2`

Problem C. Heap

Input file:	`heap.in`
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

A *d-ary heap* of size n is an array $a_1, a_2, \ldots, a_n$ where for each pair of indices i and j such that $1 \le i \le n$, $1 \le j \le n$ and $(i-1) \cdot d + 2 \le j \le i \cdot d + 1$, the strict inequality $a_j > a_i$ holds. For example, for a ternary ($d = 3$) heap of size 8, the required inequalities are: $a_2 > a_1$, $a_3 > a_1$, $a_4 > a_1$, $a_5 > a_2$, $a_6 > a_2$, $a_7 > a_2$, and $a_8 > a_3$.

Consider all d-ary heaps of size n which also happen to be permutations of numbers from 1 to n. Let us write them all down in lexicographical order comparing permutations as sequences of numbers. After that, we enumerate the heaps in the resulting ordered list starting from 1.

You are given a d-ary heap of size n which is also a permutation. Your task is to find the number of this heap in the ordered list constructed above. As the answer can be very large, compute it modulo $(10^9 + 7)$.

Input

The first line of input contains a positive integer T: the number of test cases. The test cases follow.

Each test case is given on two lines. The first of these lines contains two integers n and d ($1 \le n \le 3000$, $1 \le d \le 3000$). The second line contains n integers describing the permutation. Each integer from 1 to n occurs on that line exactly once. The given permutation is also a d-ary heap.

The sum of all values of n in the input does not exceed 3000.

Output

For each test case, print the 1-based number of the given sequence in the lexicographically ordered list of d-ary heaps which are also permutations, modulo $(10^9 + 7)$.

Example

heap.in	standard output

2
5 2
1 3 2 4 5
5 3
1 4 3 2 5
	7
	12

Problem D. Lines

Input file:	`lines.in`
Output file:	*`standard output`*
Time limit:	1 second
Memory limit:	256 mebibytes

You are given n lines on a plane. Your task is to select the maximum possible number of lines so that among the selected ones, no two lines are the same, no two lines are parallel and no two lines have an intersection at a point with $x = 0$.

Input

The first line of input contains one positive integer T, the number of test cases. The test cases follow.

Each test case starts with a line containing an integer n, the number of lines ($1 \le n \le 3000$). Each of the next n lines of input contain three integers A, B and C describing a line as a set of points (x, y) for which the equation $Ax + By + C = 0$ holds ($-10^9 \le A, B, C \le 10^9$, $A^2 + B^2 > 0$).

The sum of n in the input does not exceed 3000.

Output

For each test case, first, on a separate line, print the number k: the maximum possible number of lines that can be selected. On the next line, print k integers: the numbers of the chosen lines in any order. The lines are numbered starting from 1 in the order they are given in the input.

If there are several optimal answers, print any one of them.

Example

lines.in	standard output

2
2
1 1 0
1 1 1
2
-1 1 1
-2 1 1
	1
	1
	1
	1

Problem E. Yet Another Problem About Permutations

Input file:	`permutation.in`
Output file:	*`standard output`*
Time limit:	3 seconds
Memory limit:	256 mebibytes

This is a problem about permutations. If you are not familiar with some of the terms used below, please see the note following the example.

A permutation p is said to be *simple* if the length of each of its cycles does not exceed two. For example, permutation $2, 1, 4, 3$ is *simple*, but permutation $3, 1, 2$ is not.

You are given a permutation p. Your task is to represent it as a product of minimal number of simple permutations.

Input

The first line of input contains one integer T, the number of test cases ($1 \le T \le 10^5$). The test cases follow.

Each of next T lines describes a single test case. Each test case description consists of an integer n, the length of the permutation p ($1 \le n \le 10^5$), followed by n distinct integers $p_1, p_2, \ldots, p_n$, the permutation p itself ($1 \le p_i \le n$, each number from 1 to n appears in the permutation exactly once).

The total length of all permutations in the input is not greater than 10^6.

Output

For each test case, start by printing a line containing an integer k, the minimal number of simple permutations in the product. The next k lines must describe simple permutations $q^{(1)}$, $q^{(2)}$, …, $q^{(k)}$, one per line. On i-th of these lines, print n distinct integers from 1 to n describing permutation $q^{(i)}$. The product $q^{(1)} \circ q^{(2)} \circ \ldots \circ q^{(k)}$ must be equal to p.

If there are several optimal answers, print any one of them.

Example

permutation.in	standard output

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

Note

A *permutation* of length n is a sequence of n integers where each integer from 1 to n appears exactly once.

A *cycle* in a permutation p is a sequence $i_1, i_2, \ldots, i_t$ of distinct integers from 1 to n such that $p_{i_1} = i_2$, $p_{i_2} = i_3$, …, $p_{i_{t-1}} = i_t$ and $p_{i_t} = i_1$. The number $t \ge 1$ is called the *length* of the cycle.

The *product* $a \circ b$ of two permutations a and b is a permutation c such that for each i, $c_i = a_{b_i}$. For example, if $a = 3\,2\,1$ and $b = 1\,3\,2$, their product is $a \circ b = 3\,1\,2$.

The *product* of three or more permutations can be evaluated in any order, for example, $a \circ b \circ c = (a \circ b) \circ c = a \circ (b \circ c)$.

Problem F. Strange Sequence

Input file:	`sequence.in`
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	256 mebibytes

Consider the following well-known sequence s consisting of strings of digits. Let s_0 = “2”. Each next term is obtained by describing the previous term: split the previous term into consecutive groups of equal digits, and for each such group, write the size of the group followed by the digit the group consists of. Thus, the first few terms are constructed as follows:

String	Description
s_0 = “2”	one 2
s_1 = “12”	one 1, one 2
s_2 = “1112”	three 1s, one 2
s_3 = “3112”	one 3, two 1s, one 2
s_4 = “132112”	one 1, one 3, one 2, two 1s, one 2
s_5 = “1113122112”	...

Your task is to find the length of the n-th term of this sequence modulo 7 340 033.

Input

The first line of input contains one integer n ($0 \le n \le 10^{18}$).

Output

Print the length of s_n modulo 7 340 033.

Examples

`sequence.in`	`standard output`
0	1
2	4

Problem G. Shortest Accepted Word

Input file:	`shortest-accepted.in`
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

In this problem, we consider strings consisting of lowercase Latin letters "`a`", "`b`" and "`c`".

For this problem, let us define a *regular expression* recursively as follows:

1. A single character "`$`" is a regular expression accepting the empty string.
2. Single characters "`a`", "`b`" and "`c`" are regular expressions accepting strings "`a`", "`b`" and "`c`", respectively.
3. If P is a regular expression, "`(`P`)`" is also a regular expression accepting all strings accepted by P.
4. If P is a regular expression which is a **direct** result of applying any of the rules 1–4, its *iteration*, denoted as "P`*`", is also a regular expression accepting strings of the form $s = u_1 u_2 \ldots u_k$ (a concatenation of zero or more strings) where k is any non-negative integer and each string u_i is accepted by P.
5. If P and Q are regular expressions which are **direct** results of applying any of the rules 1–5, their *concatenation*, denoted simply as "PQ", is also a regular expression accepting strings of the form $s = uv$ (a concatenation of u and v, each of which may be empty) where the prefix u is accepted by P and the suffix v is accepted by Q.
6. If P and Q are regular expressions which are **direct** results of applying any of the rules 1–6, their *union*, denoted as "P`|`Q", is also a regular expression accepting both strings accepted by P and strings accepted by Q.

The restrictions in rules 4–6 are imposed to prevent ambiguities and prioritize operations: when reading a regular expression, evaluate iteration, then concatenation, then union. Parentheses play the usual role of prioritizing operations enclosed in them. For example, the regular expression "`a(bac|ac*)`" is read as "accept `a` followed by either (`b` followed by `a` followed by `c`) or (`a` followed by zero or more copies of `c`)".

Given a regular expression r, find the shortest string s which is accepted by this regular expression r. If there are several such strings, find the lexicographicaly smallest one.

Input

The first line of input contains one integer T, the number of test cases ($1 \leq T \leq 300$).

Each of the next T lines describes a single test case. Each test case description consists of a regular expression r which is a string constructed by the above rules. Its length is from 1 to 300 characters.

The sum of all lengths of regular expressions is not greater than 300.

Output

For each test case print a single line containing the shortest string accepted by the given regular expression r. If there is more than one such string, print the **lexicographically smallest** one.

If the answer is an empty string, print a single character "`$`" instead.

Example

shortest-accepted.in	standard output
3 a ab|ac*(ca|cb) ((ab|ac)a)*	a ab $

Problem H. Work

Input file:	`work.in`
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

There are N men and M different working assignments for them. You are given a matrix A in which $A_{i,j} = 1$ if i-th worker is qualified to complete j-th job, and $A_{i,j} = 0$ otherwise. A worker can be assigned to a job only if he is qualified to complete that job.

Your goal is to assign workers to jobs in such a way that the distribution of the amounts of jobs done by workers is as close as possible to uniform (in Euclidean metric). This means that the N-dimensional vector in which i-th element is the amount of jobs completed by i-th worker must be as close as possible to the N-dimensional vector in which each element is equal to the real number M/N.

An additional requirement is that each job which can be completed at all must be assigned to exactly one worker.

Input

The first line of input contains two integers N and M ($1 \le N, M \le 300$). It is followed by N lines each containing M characters. Each of these characters is either '`0`' or '`1`'. These lines represent the matrix A.

Output

Print N lines. On i-th line, first, print k_i, the amount of jobs assigned to i-th worker. After that, print the numbers of those jobs. If there are several different ways to assign jobs and get an optimal distribution, print any one of them.

Examples

work.in	standard output

`3 3`
`111`
`111`
`111`
	`1 1`
	`1 2`
	`1 3`

`2 4`
`0100`
`1100`
	`1 2`
	`1 1`

Note

The Euclidean distance between two vectors $(u_1, u_2, \ldots, u_N)$ and $(v_1, v_2, \ldots, v_N)$ is the real number

$$\sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2 + \ldots + (v_N - u_N)^2}.$$