Constructor

SEASON 10 – SECOND ROUND

Annie and Bobby play with their new constructor, which consists of colored cubes. The colors of the cubes are M in total and they are labeled with small Latin letters. It is known that there are C_i cubes colored in L_i. The children take turns to add cubes into a sequence. In the beginning, they choose a color and put all the cubes of this color into the sequence. After that they choose another color and add some of the cubes at the front of the sequence and the remaining ones at the end. They do this until they run out of cubes. It is possible to put all the cubes of a certain color only at the front or only at the end.

Write a program which determines whether by doing this Annie and Bobby can construct N given sequences of cubes, represented as strings.

Input

The first line of the input file `constructor.in` contains the numbers N and M. The following M lines contain a symbol L_i and a number C_i, separated by a space. Each of the last N lines contains a string which describes a sequence that you have to check.

Output

On N lines of the output file `constructor.out` print "Yes" или "No" depending on whether the corresponding sequence of cubes could be obtained or not.

Constraints

$1 \leq N \leq 100$
$1 \leq M \leq 26$
The total length of the strings will not exceed 100.
The sum of all C_i will not exceed 100.

Example

Input	Output
5 5 a 2 b 3 c 1 x 2 y 4 aabbbcxxyyyy ycbaabbxxyyy ababbxxyyycy xxbacabbyyy yaabbbcxyyyx	Yes Yes No No No

Partsort

SEASON 10 – SECOND ROUND

Given is a numeric sequence, which is a permutation of the numbers from 1 to $\boldsymbol{N}$. We say that the sequence is sorted if for each $i = 1, 2, \ldots, N - 1$ the inequality $A_i < A_{i+1}$ is satisfied, where A_i denotes the i-th number in the sequence. In the not-so-distant past the computers had way less operational memory and namely because of this to sort a sequence was not a trivial task.

We know that we can load in the memory at most $\boldsymbol{K}$ of the elements of the sequence at a time. That is why, we first sort the numbers with indices between 1 and K, then those between 2 and $K + 1$ and so on until we sort the elements from index $N - K + 1$ to index N. Unfortunately, this is not always sufficient to sort the sequence. For instance, if $N = 5$, $K = 3$ and $A = \{4, 5, 3, 1, 2\}$, we ger the following changes: $\{\mathbf{3}, \mathbf{4}, \mathbf{5}, 1, 2\} \rightarrow \{3, \mathbf{1}, \mathbf{4}, \mathbf{5}, 2\} \rightarrow \{3, 1, \mathbf{2}, \mathbf{4}, \mathbf{5}\}$. So, after one such step, we will change the original sequence to $\{3, 1, 2, 4, 5\}$ and it will take us one more step to sort the sequence completely.

Write a program, which finds the number of steps that the algorithm will do, in order to sort the given sequence. The program has to process $\boldsymbol{T}$ test cases during a single execution.

Input

The first line of the input file `partsort.in` contains a single number T. Each of the following T lines describes one test case in the format – $N, K, A_1, A_2, \ldots, A_N$.

Output

On N lines of the output file `partsort.out` print one number equal to the required number of steps needed to sort the sequence from the corresponding test case.

Constraints

$2 \leq K \leq N \leq 10\,000$

The sum of $N \div K$ over all test cases will not exceed 100.

Example

Input	Output
3 5 3 4 5 3 1 2 3 2 1 2 3 7 7 7 6 5 4 3 2 1	2 0 1

Bit inversion

SEASON 10 – SECOND ROUND

You are given **N** binary numbers, each with **M** bits (the numbers can have leading zeros). You have a special device, which can perform the following operation: for a given index of a number and a position of a bit, the device inverts all bits of the number with the given index (each 0 becomes a 1 and each 1 becomes a 0) and also it inverts the bit on the given position in all other numbers. Your goal is to use the special device in such a way that all N numbers become equal to zero.

Input (bitinversion.in)

On the first line of the input are given the numbers **N** and **M**. On each of the next N lines, a binary number with M bits is given.

Output (bitinversion.out)

If it's impossible to make all numbers equal to zero using the described operation, output the number **-1** on a single line. Otherwise, on the first line of the output print the number **K** ($0 \le K \le 1\,000\,000$) - the number of operations you are going to perform. On each of the following K lines print two numbers $\mathbf{x_i}$ and $\mathbf{y_i}$ ($0 \le x_i < N$, $0 \le y_i < M$, the most significant bit of each number has position M-1 and the least significant has position 0) - the index of a number and the position of a bit you'll give the device for the i-th operation. It can be proved that if it's possible to make all numbers equal to zero, you can do it with no more than 1 000 000 operations. If there are several ways to do it, output any of them.

Constraints

$1 \le N*M \le 1\,000\,000$

Example

Input	Output
2 3 010 100	3 0 1 0 0 1 1

Explanation

The numbers change the following way:
(010, 100) → (101, 110) → (010, 111) → (000, 000)