Bill Restoration

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

This is a run-twice task. Your solution will be executed twice.

The members of the jury of the Olympiad made a major purchase in Innopolis. For convenience, one person paid for the entire bill, and the others had to pay him their share later. The bill, of course, was immediately lost, but a photo of a rather poor quality that was taken for a report, survived.

Not only are the individual items on the bill not visible, but also k consecutive digits in the total amount are smudged. Nevertheless, the jury members, albeit with great difficulty, managed to reconstruct the information about the purchase. Your task is to emulate this scenario.

On the **first run**, you are given an integer A, consisting of exactly n digits — the total cost of the purchase, and a number k — the number of consecutive digits that need to be "blurred out". It is guaranteed that $n \geq 10^k + 10$. You must select exactly k consecutive digits of the number and replace them with the symbol '`#`'.

On the **second run**, your program receives a string that was obtained during the first launch by replacing k consecutive digits with '`#`'. Your goal is to fully restore the original number A, i.e., output the k digits that are "hidden" behind the hash symbols.

Input

This problem uses multi-test, each test contains multiple sets of input data.

For the first run, the first input line contains the number 1 and two integers t and k — the number of sets of input data and the number of digits to be hidden in each of them ($1 \leq k \leq 6$). In the i-th of the following t lines, a string A for the i-th set of input data is given, consisting of digits (characters from '`0`' to '`9`') and representing the bill amount ($10^k + 10 \leq |A| \leq 10^6 + 20$).

It is guaranteed that the sum $|A|$ for all sets of input data in one test does not exceed $2 \cdot 10^6 + 10^5$.

For the second run, the first line contains the number 2 and an integer t — the number of sets of input data. In the following t lines, strings A' for all sets of input data are given. Each A' consists of characters from '`0`' to '`9`' or '`#`' and was obtained during the first run.

Between the two executions, the sets of input data may be shuffled.

Output

For the first run, for each set of input data output on a separate line an integer l from 1 to $n - k + 1$ — the starting position of the segment of k digits that will be replaced with '`#`'.

For the second run, for each set of input data output on a separate line a string of length exactly k consisting of digits — the sequence that was hidden behind the segment of '`#`'. **In the second run the answers should be printed in the same order as the strings appear in the second run's input data**.

Interaction Protocol

To avoid receiving incorrect verdicts such as `Idleness Limit Exceeded` or `Security Violation`, terminate the output of each line with a newline character ('`\n`') and flush the output stream buffer (`cout.flush()` in C++, `System.out.flush()` in Java, and `sys.stdout.flush()` in Python).

Scoring

Points for each subtask are awarded only if all tests for that subtask and the necessary subtasks are successfully passed.

Subtask	Points	Constraints	Necessary subtasks	Feedback policy
0	–	examples		full
1	10	$k = 1$, $t = 1$, '1' $\leq A_i \leq$ '2' for all i and A is generated uniformly at random		full
2	13	$k = 1$, '0' $\leq A_i \leq$ '2' for all i	1	first error
3	17	$k = 1$, '0' $\leq A_i \leq$ '7' for all i	1, 2	first error
4	17	$k \leq 2$, '0' $\leq A_i \leq$ '7' for all i	1 – 3	first error
5	21	'0' $\leq A_i \leq$ '7' for all i	1 – 4	first error
6	22	no additional constraints	0 – 5	first error

Examples

standard input	standard output	Notes
1 2 1 12345678901234567890 11111111111111111111	20 11	First run
2 2 123456789012345678 9# 1111111111#111111111	0 1	Second run

standard input	standard output	Notes
1 1 2 10000000002000000003000000000 40000000005000000005999999999 49999999993999999992999999999 19999999995555555555	51	First run
2 1 10000000002000000003000000000 40000000005000000000##99999999 49999999993999999992999999999 19999999995555555555	59	Second run