

Problem A. Mr. Buridan and coffee shops

Input file: `stdin`
Output file: `stdout`
Time limit: 3 seconds
Memory limit: 256 megabytes

Buridan's donkey is a famous philosophical mind experiment. Imagine putting a hungry donkey in front of two identical heaps of hay. The donkey would like to eat, but as the situation's perfectly symmetric, it can't decide whether to pick the left or right heap. And so, it stays in place until it dies of hunger.

Little David is still trying to find a place to live in Manhattan. Besides other stuff, he likes visiting coffee shops. But he just realized that he's in the same danger as Buridan's donkey: if some 2 coffee shops were equally distant from his flat, it's possible that he couldn't decide which one to visit and that wouldn't end well...

For simplicity, we'll picture Manhattan as a square grid, where we can only move in the 4 basic directions. There are coffee shops at some grid points and David also wants to live at some grid point (possibly containing a coffee shop).

The input contains a map of Manhattan. For every intersection (grid point), find out if little David can live there — e.g. if all coffee shops are at different distances from there.

Input

The first line of the input contains an integer N — dimension of the Manhattan square grid. The grid is formed by N vertical and N horizontal streets, so we have N^2 intersections (grid points). The rest of the input contains N lines of N integers each: 0 if there's no coffee shop at that point and 1 if there is one.

Output

Print N lines of N characters each: A if little David can live at that point and N otherwise.

Examples

stdin	stdout
5 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	A A A A A A A A A N N N N N A A A A A A A A A A A
3 1 0 1 0 0 0 1 0 1	N N N N N N N N N
2 0 1 0 0	A A A A

Note

In all tests, $1 \leq N \leq 10^3$.

Problem B. Tug of war

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

In Hillside Village, a traditional tug of war tournament takes place. Since there's no horizontal field in the whole village, the game is played on a slope of a hill. The hill's steepness is represented by a coefficient $q > 1$. There are N players starting in the tournament, numbered 1 to N from youngest to oldest. Player number i has strength s_i .

At the start of tournament, all players are active. The tournament consists of several matches. All active players play in a match. At the start of a match, they split into 2 teams: the *upper* team (on the top of the hill) consists of the K oldest players; the *lower* team consists of the rest. A team's strength is the sum of strengths of all its players. The upper team has an obvious disadvantage; the win iff their strength is strictly larger than q times the lower team's strength. The tournament ends as soon as the upper team wins.

After each match, one player stops being active. The natives noticed that it's always either the youngest active player (who has to go home) or the oldest active player (who visits the buffet). The value of K remains unchanged, so if the oldest player leaves, then the oldest player of the lower team automatically switches teams. (Notice that if such a player doesn't exist, then the tournament would end anyway.)

The input contains the number of players N , their strengths $s_{1..N}$, the coefficient of hill's steepness q and the size of the upper team K . Write a program which computes the maximum number of matches that could be played. In other words, it's supposed to find such an order of leaving players, for which the upper team wins for the first time as late as possible.

Input

The first line of the input contains integers N, K ($1 \leq K \leq N$) and a real number q .

The next line contains N positive integers: s_1 to s_N .

Output

Print a single integer - the maximum possible number of played matches.

Examples

stdin	stdout
5 2 1.01 40 70 1000 30 20	4
12 7 2.0 2 4 9 5 3 3 4 8 8 6 1 6	4

Note

In all tests, $N \leq 5 \cdot 10^5$, $s_i \leq 10^9$ for all $1 \leq i \leq N$, $q \leq 1000$. Furthermore, q is given up to 3 decimal places.

Problem C. Up and down

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

Baška had a sequence of N integers from the range $[0, M]$. For every 2 consecutive elements of the sequence, she wrote down $<$, $=$ or $>$ depending on the earlier one of them being lesser, equal or greater than the later one. For example, as for a sequence $(3, 1, 7, 7, 4)$ we have $3 > 1 < 7 = 7 > 4$, Baška would've written down $><=>$.

Olívia doesn't know Baška's original sequence. She only knows N, M and the sequence of characters that Baška wrote down. Now, she'd like to reconstruct Baška's sequence — as far as possible.

Find one such sequence **with minimal sum** or decide that it doesn't exist. If there are multiple correct sequences with minimal sum, find any one of them.

Input

The input consists of 2 lines. The first contains integers N and M ; the second contains the sequence that Baška wrote down - a string of $N - 1$ characters $<$, $=$ or $>$.

Output

Print one line containing your reconstruction of Baška's original sequence — N space-separated integers. If no such sequence exists, print N times -1 .

Examples

stdin	stdout
5 1000000000 ><=>	1 0 1 1 0
4 2 »>	-1 -1 -1 -1

Note

There are 10 sets of tests, in which the following restrictions hold:

Set(s) no.	Length of sequence	Size of elements	Other
1,6	$2 \leq N \leq 10$	$M = 10^9$	There's exactly one $>$ and no $=$ in the sequence.
2,7	$2 \leq N \leq 20$	$M = 10^9$	
3,8	$2 \leq N \leq 1000$	$M = 998$	
4,9	$2 \leq N \leq 75000$	$0 \leq M \leq 10^9$	There's no $=$ in the sequence.
5,10	$2 \leq N \leq 250000$	$0 \leq M \leq 10^9$	

Problem D. Balanced strings

Input file: `stdin`
Output file: `stdout`
Time limit: 1.5 seconds
Memory limit: 256 megabytes

We call a string *balanced* if all its characters (those which occur in it at least once) occur in it an equal number of times. For example, *aaaaa*, *badcx*, *bbaaab* are balanced strings. However, *abacb* isn't a balanced string.

Given a string, find its longest (continuous) balanced substring.

The substring we're looking for doesn't have to contain all characters which occur in the original string. For example, the correct answer for *cbababac* is the substring *bababa*.

Input

The input consists of a single string of lowercase letters of English alphabet. The string has length N and the characters are indexed from 0 to $N - 1$.

Output

Print a single line containing two space-separated integers: indices of the character by which the substring we're looking for starts and the one by which it ends. If there are multiple optimal solutions, find the one that starts the earliest.

Examples

stdin	stdout
baab	0 3
baaab	1 3
cbababac	1 6
cbabadbabae	1 4

Note

There are 10 sets of tests, in which the following restrictions hold:

Set(s) no.	Length of input	Set of letters used
1	$1 \leq N \leq 20$	$[ab]$
2	$1 \leq N \leq 1000$	$[ab]$
3-4	$1 \leq N \leq 1000$	$[abcdefgh]$
5	$1 \leq N \leq 10^5$	$[ab]$
6	$1 \leq N \leq 10^5$	$[abc]$ (the optimal solution doesn't use all three)
7	$1 \leq N \leq 10^5$	$[abc]$
8-10	$1 \leq N \leq 5 \cdot 10^4$	$[abcde]$

Problem E. ACGT

Input file: `stdin`
Output file: `stdout`
Time limit: 4 seconds
Memory limit: 512 megabytes

Usáamec is trying to construct a genome from DNA fragments. A DNA fragment is a sequence of letters $[ACGT]$. Usáamec has found N fragments already, let's denote them as $F_{1..N}$. There are also M rules describing which fragments can follow directly after which ones.

Constructing a DNA

DNA is a sequence of fragments F_{a_i} (for some sequence $\{a_i\}_{i=1}^k$; $1 \leq a_i \leq N$) such that for every $i > 1$, the a_i -th fragment can follow after the a_{i-1} -st one. We construct the DNA by concatenating those fragments and obtain a string of letters $[ACGT]$ as a result.

Distance of strings

We can modify a string in steps, where each step is one of these 3 types of changes: adding a character into the string, removing or changing one character in the string. We define the *distance* of strings p and q as the smallest number of changes we need to make in p to turn it into q (clearly, any string has distance 0 from itself). For example, $ATTA$ and AGA have distance 2 — we need to remove one T and change the other into G .

You're given some DNA fragments and rules that describe which fragment can follow after which one. You're also given a target DNA string r , a *very small* non-negative integer d and indices of 2 fragments, u and v ($u \neq v$). You should write a program that finds *one possible* sequence of fragments that starts with fragment u , ends with fragment v and the DNA that's constructed by it has distance from r of at most d . Specifically, it's not required to minimize this distance.

You may assume that no fragment can follow directly after itself and that if fragments y and z can follow after x , then the first characters of F_y and F_z are different.

Input

The first line of the input contains the number of fragments N and number of rules M . The following N lines contain one fragment F_i each. The following M lines contain 2 integers a, b (1 rule) each — it means that fragment F_b can follow directly after fragment F_a . The next-to-last line contains integers d, u, v and the last one contains string r .

Output

In case there's no sequence of fragments that satisfies the conditions, print one line with one integer -1 . Otherwise, print 2 lines - the size of your sequence on the first line and this sequence, as space-separated indices of fragments, on the second one.

Examples

stdin	stdout
3 3 CG AT TCC 1 2 2 1 1 3 0 1 3 CGATCGTCC	1 2 1 3
4 4 CCC A T GGG 1 2 2 3 3 2 2 4 1 1 4 CCCATTGGG	1 2 3 2 4
4 4 CCC A T GGG 1 2 2 3 3 2 2 4 1 1 4 CCCAAAAGGG	-1

Note

The input files are based on real data, which means there won't be many similar fragments.

There are 10 tests; in all of them, $d \leq 5$, $n \leq 1000$, $m \leq 4000$ and both the total length L_T of all fragments and R (the length of r) won't exceed 10^6 . In addition, the following restrictions hold:

Set(s) no.	Restriction
1-8	$L_T \leq 5 \cdot 10^4, R \leq 5 \cdot 10^4$
2-3, 7-8	$L_T \leq 1000, R \leq 1000$
1	$d = 0$
6	$d = 1$
2-5	$n = 2, m = 1$
9-10	$m \leq 1000$