Problem A. Complexity

Input file:	`complexity.in`
Output file:	`complexity.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Modern algorithms often have complexities that are hard to remember and understand, like $O(VE \log \frac{V^2}{E})$. Andrew likes writing programs much more than studying theoretical algorithms, so he has invented a new way to remember those mindboggling complexity formulas: he will not remember them at all! Instead, when he needs to know the complexity of a particular algorithm, he will write the corresponding program, run it, measure the running time, and figure out the complexity formula that fits it.

In this problem we will consider the simplified case where the input to the program is just one positive integer N. Given this positive integer N and the corresponding running time X (which is also a positive integer) of the program, Andrew needs to represent X as a sum of products of N, $\log N$, $\log \log N$ and so on, for example: $X = N \log \log N + N \cdot N$.

Since both N and X are integers, we define $\log a$ to be integer as well: it is the *floor* of the binary logarithm of N, meaning the largest integer k such that $2^k \le a$. So for example $\log \log N$ means the floor of the binary logarithm of the floor of the binary logarithm of N. The logarithm of zero or any negative number is undefined.

Andrew doesn't allow the formula to contain integer factors (he remembers that there weren't any in his complexity theory classes for some reason) and parentheses (that would be too complicated). The only type of formula he's interested in is the sum of one or more terms, where each term is a product of one or more factors, and each factor is one of N, $\log N$, $\log \log N$, $\log \log \log N$ and so on.

Since there can be several such formulas that evaluate to X, Andrew wants the one that has the fewest mentions of N in it (since every factor has exactly one mention, this means the fewest total number of factors). If there are several formulas with the fewest Ns, he doesn't care which one to choose.

Input

The only line of the input file contains two integers N and X, $1 \le N, X \le 100\,000$.

Output

Output one integer — the fewest number of Ns in a formula that evaluates to X.

Examples

complexity.in	complexity.out
100 3604	5

Note

$N \log N \log N + \log \log N + \log \log N = 100 \cdot 6 \cdot 6 + 2 + 2 = 3604$.

Problem B. Divisibility Tree

Input file:	`divisibility-tree.in`
Output file:	`divisibility-tree.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

A rooted tree is called a *divisibility tree* when every node of the tree is assigned a positive integer, such that the number assigned to a parent of each node is *strictly* less than the number assigned to the node, and divides it evenly.

Given a rooted tree with numbers assigned to its leaves (nodes with no children), you need to assign numbers to all other nodes in such a way that the tree becomes a divisibility tree.

Input

The first line of the input file contains an integer n, denoting the number of nodes in the tree, $1 \le n \le 1000$.

The next n lines describe the nodes of the tree. The i-th node is described with 2 integers p_i and a_i. The value of p_i is the 1-based index of the parent of this node (or -1 if this node is the root). Additionally, p_i is always less than i. The value of a_i is the number assigned to this node, or -1 if no number is assigned yet. Numbers will be assigned to all leaves, and to no other nodes.

The root of the tree is the first node. All assigned numbers don't exceed 10^9.

Output

Output n positive integers separated by spaces — the numbers assigned to the nodes. If there are several possible solutions, output any one. If there is no solution, output n times -1.

Examples

divisibility-tree.in	divisibility-tree.out
5 -1 -1 1 2 1 -1 3 6 3 8	1 2 2 6 8
3 -1 -1 1 -1 2 2	-1 -1 -1

Problem C. Progressing Fraction

Input file:	`fraction.in`
Output file:	`fraction.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Young Andrew was always wondering: how many powers of 2 start with 7 in decimal notation? There seem to be relatively few at the beginning (the smallest one is $2^{46} = 70368744177664$), but such powers start to appear more often afterwards. Imagine how excited he was to learn that about $5.7991946977\ldots\%$ of all powers of two start with 7!

He is now after a more general question. Given a number n, and a geometric progression $a_i = b \cdot q^i$, $i \geq 0$, what is the fraction of the elements of that progression with decimal notation that has the decimal notation of n as prefix? More formally, if c_i out of the first i elements of the progression start with n in decimal notation, you need to find the limit $\lim\limits_{i \to \infty} \frac{c_i}{i}$. It is guaranteed that the limit always exists.

Input

The only line of the input file contains three integers n, b and q. $1 \leq n, b, q \leq 1000$.

Output

Output one floating-point number — the sought fraction. Your answer will be considered correct if it is within 10^{-9} of the right answer.

Examples

fraction.in	fraction.out
7 1 2	0.057991946977686705

Problem D. 4-Character Percentage

Input file:	`percentage.in`
Output file:	`percentage.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Andrew has invented a novel way of remembering his passwords. First, he chooses a string s consisting of at least 4 lowercase English letters. Then, he looks at all 4-character sequences that can be obtained by taking any 4 characters of s and writing them in the same order as they appear in s (more formally, he looks at all quadruples of numbers $1 \le i < j < k < l \le length(s)$, and the corresponding strings $s_i s_j s_k s_l$). Then, he counts the number of times each 4-character string appears in this list, and orders all 4-character strings that appear at least once by frequency. Then, he uses several most frequent ones as passwords.

You need to help him to find these frequent 4-character strings. More specifically, given s, you need to find all 4-character strings that form at least 1% of all strings from the above list, and find the frequency for each of those strings. Formally, for each 4-character string t that appears n_t times out of n total items in the list, we find the *truncated percentage* p_t which is the largest integer such that $\frac{n_t}{n} \ge \frac{p_t}{100}$. For each 4-character string t where $p_t \ge 1$ you need to output this string, together with the truncated percentage. Additionally, if there is at least one string t with $p_t = 0$ in the above list, you should output "`Others less than 1%`" in the end.

Input

The only line of the input file contains the string s consisting of lowercase English letters. The length of s is between 4 and 10000, inclusive.

Output

Output all 4-character strings that take at least 1% of the list in the decreasing order of their truncated percentage. When several strings have the same truncated percentage, order them lexicographically. For each string, you should print the string, then a space, then the truncated percentage with the "`%`" sign after it, in one line. Output "`Others less than 1%`" in the end if there are strings with the truncated percentage of 0 appearing at least once in the list. See sample output for further clarification.

Examples

percentage.in	percentage.out
tests	ests 20% tess 20% test 20% tets 20% tsts 20%
aabbccdd	abcd 22% aabc 5% aabd 5% aacd 5% abbc 5% abbd 5% abcc 5% abdd 5% accd 5% acdd 5% bbcd 5% bccd 5% bcdd 5% aabb 1% aacc 1% aadd 1% bbcc 1% bbdd 1% ccdd 1%
testaaaaaaaaaaaaaaaaaaaaaaaaaaaa	aaaa 59% taaa 17% eaaa 8% saaa 8% Others less than 1%

Problem E. Random Strings

Input file:	`random-strings.in`
Output file:	`random-strings.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

A *test generator* is a ubiquitous part of an algorithm contest problem. Here, we're studying a test generator that generates a single string consisting of lowercase English letters.

The test generator in question has two *strategies*:

- the first strategy is to generate a completely random string: each letter is chosen uniformly and independently at random.
- the second strategy is slightly more complicated: first letter is chosen uniformly at random. Then, whenever a letter is chosen, its probability to be chosen at the next step and later decreases by the factor of 2. More formally, each of the 26 letters is assigned a *weight*, with the initial value of all weights equal to 1. At each step, we pick a random letter independently, but according to the current weights (letter x has probability $\frac{weight_x}{\sum_i weight_i}$ of being chosen). The weight of the chosen letter is divided by 2 afterwards, and the process is repeated for the next letter.

Given many relatively long strings built by this generator, can you determine the strategy used for each string?

Input

The first line of the input file contains one integer n, $1 \le n \le 100$, the number of strings to analyze.

The next n lines contain one string of lowercase English letters each. The length of each string is *exactly* 1000. These strings will be generated according to one of the above strategies (different strategies may be used for different strings).

Output

Output n lines. The i-th line should contain "`FIRST`" if the first strategy (all letters are chosen independently and uniformly) was used for this string, and "`SECOND`" otherwise.

Examples

random-strings.in	random-strings.out
4 kshklmfsnhggrefxnkcviqdgnawgtq jpdutxmiselqokxjemccnphgdbszlf drmigzsvclhpiyogtefkqwurhabsgy ylfhbkgwerdbkxqriltuzkvpsxgddt	FIRST SECOND SECOND FIRST

Note

Note that the strings in the example input are too short — such testcases won't be used when judging. Every input string will contain exactly 1000 letters.

Problem F. Rotor Traversal

Input file:	`rotor.in`
Output file:	`rotor.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

There are many ways to traverse an undirected graph. One of them is *rotor traversal*: for each vertex, we choose a *rotor sequence*, which is a permutation of all vertices adjacent to this vertex. Then, we start our traversal at some vertex. When we reach a vertex for this first time, we continue to the first vertex of its rotor sequence. When we reach a vertex for the second time, we continue to the second vertex of its rotor sequence, and so on. When we reach the end of a rotor sequence, we start from the beginning. The overall traversal stops whenever we reach the last unvisited vertex.

This problem is concerned with rotor traversals of trees, so let's take a look at the tree from the below example. Vertex 1 is connected to vertices 5 and 4, its rotor sequence is [5, 4]. Vertex 2 is connected only to vertex 4, its rotor sequence is [4]. Vertex 3 is connected only to vertex 4, its rotor sequence is [4]. Vertex 4 is connected to vertices 1, 2 and 3, its rotor sequence is [1, 3, 2]. Vertex 5 is connected only to vertex 1, its rotor sequence is [1]. The rotor traversal starting from vertex 1 is: 1, 5, 1, 4, 1, 5, 1, 4, 3, 4, 2.

Given a tree (a connected graph without cycles), you need to choose rotor sequences for it. What is the minimal and maximal possible number of steps in a rotor traversal starting from vertex 1?

Input

The first line of the input file contains n, $1 \le n \le 100$. The next $n - 1$ lines describe edges, each with two 1-based indices of the nodes that the edge connects, separated with a space.

Output

Output two descriptions of rotor sequences, one for the minimal number of steps and one for the maximal number of steps, separated with a blank line.

Each description should start with the sought number of steps on a line by itself, followed by n lines describing the rotor sequences. The i-th of those lines should describe the rotor sequence for the i-th vertex, listing the numbers of vertices that are connected to it in the required sequence, separated by spaces.

Examples

rotor.in	rotor.out
5 4 1 5 1 2 4 3 4	7 5 4 4 4 3 2 1 1 11 5 4 4 4 1 3 2 1

Problem G. Possible Shifts

Input file:	`shifts.in`
Output file:	`shifts.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Andrew is studying classical substring matching. More formally, the problem he is studying is: given two strings s and t, find all shifts i such that the substring of t starting with character number i and of the same length as s is equal to s.

Andrew is trying to invent a new, fast algorithm for solving that problem. The key idea of the algorithm will be: we might need to compare just a few characters to reduce the number of possible shifts significantly. You need to help him quantify "significantly".

You are given several facts of the form $s_i = t_j$, meaning that i-th (1-based) character of s is equal to j-th (1-based) character of t, and $s_i \neq t_j$, meaning that the i-th (1-based) character of s is *not* equal to the j-th (1-based) character of t. How many shifts i are possible shifts? We define a *possible shift* as such shift that a match of s in this position of t is still possible given the facts. You may assume that the number of possible characters is very big.

Note that when the input data contains a contradiction, no shifts are possible, so the answer is 0.

Input

The first line of the input file contains three integers n, l_s, l_t, $0 \le n \le 100$, $1 \le l_s \le l_t \le 10^9$. n is the number of the facts given to you, l_s is the length of s and l_t is the length of t. The next n lines contain the facts. Each line contains an index i, $1 \le i \le l_s$ followed by space, followed by character "`=`" or "`!`" , followed by a space, followed by an index j, $1 \le j \le l_t$. Such line means that the i-th (1-based) character of s is equal (in case of "`=`") or not equal (in case of "`!`") to the j-th (1-based) character of t.

Output

Output the number of the possible shifts given the facts.

Examples

shifts.in	shifts.out
6 3 10 1 ! 1 1 = 10 2 = 10 3 = 10 1 ! 5 1 ! 8	1

Problem H. Small Graph

Input file:	small-graph.in
Output file:	small-graph.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Consider a permutation $p_1, p_2, \ldots, p_n$ of integers between 1 and n. A directed graph G with at least n vertices is called an *inversion graph* of that permutation when for any i, j such that $1 \le i, j \le n$, $i \ne j$ the j-th vertex of G is reachable via some path from the i-th vertex of G *if and only if* $i < j$ and $p_i > p_j$ (such pair is called an *inversion* of the permutation).

Naturally, there exist many inversion graphs for each permutation. You need to find a relatively small one: the graph must contain at most $30n$ vertices and at most $30n$ edges.

Input

The first line of the input file contains one integer n, $1 \le n \le 1000$, denoting the size of the permutation. The second line of the input file contains the permutation itself.

Output

In the first line of the output file, print two integers v and a — the number of vertices and arcs of the graph, respectively. The next a lines should contain the arcs. Each arc should be described by two vertex numbers (between 1 and v) — the source and destination of the arc.

The number v should be at least n and at most $30n$, a should be at least 0 and at most $30n$.

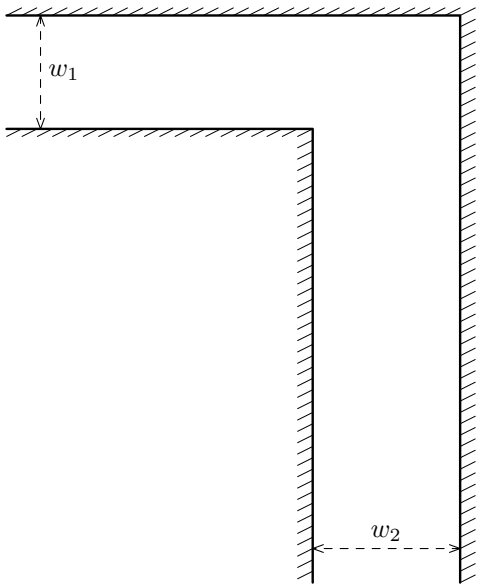
Examples

small-graph.in	small-graph.out
4 3 4 1 2	5 4 1 5 2 5 5 3 5 4

Problem I. High Speed

Input file:	`speed.in`
Output file:	`speed.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

One of the most common elements of driving is the *90-degree turn*. We will define such turn as a combination of two roads that intersect at the right angle, as in the following picture:



A car can drive along any path that is a smooth sequence of straight line segments and circular arcs. We define *smooth* as: whenever two different parts of a path meet, they must have the same direction vector in the meeting point.

We need to find a path from far far away on the left road to far far away on the bottom road. We define the *turning radius* of a path as the smallest radius of all arcs on the path.

What is the largest possible turning radius of a path through the given corner?

Input

The only line of the input file contains two integers w_1 and w_2, denoting the width of the left road and the width of the bottom road, respectively. $1 \le w_1, w_2 \le 100$.

Output

Output one floating-point value: the largest possible turning radius that still allows to drive through the given corner. Your answer will be considered correct when it's within 10^{-9} relative error of the right answer.

Examples

speed.in	speed.out
10 10	34.14213562373095

Note

Here's one of the possible optimal paths for the example: