Problem A. Arnold

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

Arnold's principle (sometimes also called *Stigler's law of eponimy*) claims that "No scientific discovery is named after its original discoverer". Of course this law applies to itself: it is first mentioned by neither Arnold nor Stigler. In order to respect it, this problem is not invented by any of them and does not mention these scientists anywhere except for this paragraph.

You are given an unrooted tree of n vertices. Each edge has associated length. Some vertices are *special* and have desired height. Your task is to find all such vertices u that, if the tree is rooted at u, then for every special vertex, its desired height matches its actual height.

Input

On the first line of input, there are two integers n and k: the number of vertices of the tree and the number of special vertices ($0 \leqslant k \leqslant n \leqslant 500\,000$, $2 \leqslant n$).

Each of the next $n - 1$ lines contains three integers u_i, v_i, w_i ($1 \leqslant u_i, v_i \leqslant n$, $1 \leqslant w_i \leqslant 1000$). For each i, there is an edge between vertices u_i and v_i of length w_i.

Each of the next k lines contains two integers s_j, h_j ($1 \leqslant s_j \leqslant n$, $0 \leqslant h_j \leqslant 10^9$). For each j, vertex s_j is a special vertex with desired height h_j. All s_j are distinct.

Output

On the first line, print one integer m: the number of appropriate roots. On the second line, print m integers: the numbers of these roots in any order. Vertices are numbered from 1 to n as in the input.

Example

standard input	standard output
7 2 1 2 1 2 3 1 2 4 1 1 5 1 5 6 1 5 7 1 2 1 5 3	2 3 4

Problem B. Bloom

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	256 mebibytes

Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. Elements can be added to the set but not removed.

A Bloom filter for a universe S is an array of m bits initially set to zero and l different hash functions $S \to [0, m)$: $h_1, \ldots, h_l$.

To add an element $s \in S$ to the set, feed it to each hash function and set bits $h_1(s), \ldots, h_l(s)$ to one. To test whether an element is in the set, look at bits $h_1(s), \ldots, h_l(s)$. If any of them is zero, this element is "definitely not in the set", otherwise it is "possibly in the set". So, Bloom filter allows false positive matches but not false negatives.

Your teammate implemented his own Bloom filter for integers from 0 to 10^9 inclusive. He selected a prime number m and defined one hash function (otherwise it would be too complicated for him): $h(s) = (As + B) \bmod m$ where $0 \leqslant A, B < m$. Then he selected n numbers $a_1, \ldots, a_n$ and added them to the filter. After this operation, k bits were set: $b_1, \ldots, b_k$.

You want to reproduce his experiment. Unfortunately, you don't know the values of A and B. So you decided to find all pairs (A, B) such that adding all numbers $a_1, \ldots, a_n$ to the Bloom filter using your parameters will set exactly those bits.

Input

The first line of input contains three integers n, k and m ($1 \leqslant n, k \leqslant 10^6$, $2 \leqslant m \leqslant 10^6$, m is prime). The second line of input contains n integers $a_1, \ldots, a_n$ ($0 \leqslant a_i \leqslant 10^9$) which are the numbers added to the filter. The third line contains k integers $b_1, \ldots, b_k$ ($0 \leqslant b_j < m$) which are the numbers of bits that must be set to one after adding all elements to the filter. All other bits must be zeroes.

Output

On the first line of output, print one integer p, the number of pairs. On each of the next p lines, print two integers A and B which represent a pair of parameters such that adding all numbers $a_1, \ldots, a_n$ to the Bloom filter using these parameters will set exactly the bits $b_1, \ldots, b_k$.

It is guaranteed that the number of pairs p will be at most 10^6.

Examples

standard input	standard output
4 4 11 1 6 8 9 0 2 8 9	1 3 6
4 4 11 1 6 8 9 1 2 3 4	0
6 2 11 11 12 22 23 33 34 0 1	2 1 0 10 1

Problem C. Collatz

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Collatz function f is defined on positive integers as follows:

$$\begin{cases} f(x) = \frac{x}{2}, \text{ if } x \bmod 2 = 0, \\ f(x) = 3 \cdot x + 1, \text{ otherwise.} \end{cases}$$

Let us define a *Collatz sequence* of a number x as $x, f(x), f(f(x)), \ldots$. For example, for 10 it looks like 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ... The *Collatz hypothesis* states that for any positive integer, its Collatz sequence will eventually reach the "4, 2, 1" loop.

This problem has not yet been solved, so we won't ask you to prove or disprove the hypothesis. Just find the position of the first 1 in the Collatz sequence of a random number. You can find the exact meaning of the word "random" for this problem in the Notes section below.

Input

The first and only line of input contains an integer n in binary notation ($1 \leqslant n \leqslant 2^{200\,000}$). There are no leading zeroes.

Output

Print one integer: the position of the first occurrence of 1 in the Collatz sequence of n. Sequence elements are numbered starting from zero. Although the input is random, it is guaranteed that the answer for each test is finite.

Example

standard input	standard output
1010	6

Note

There are three categories of tests:

1. samples;
2. "small" tests: $n \leqslant 2^{1000}$, n is random, seed is selected manually by the jury;
3. "large" tests: no additional constraints, n is random, seed is selected randomly without studying the structure of the resulting test.

Here, "n is random" means "length of n is selected manually, the first digit is 1, all other digits are uniformly and independently set to zero or one using some pseudo-random number generator".

Problem D. Dirichlet

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	512 mebibytes

A well-known *Dirichlet's principle* (or sometimes *pigeonhole principle*) states that if $n+1$ pigeons are put into n holes, there will be a hole with at least two pigeons in it.

However, only a few know about another, similar, principle: if $n-1$ pigeons are put into n holes, there exists an empty hole! Even more, if no two pigeons take the same hole, there will be only one empty hole at the end. Sometimes you can observe this principle in nature when birds settle into hollows in a tree.

There is an unweighted tree with n vertices, and in every vertex, there is a hole. There are $n-1$ birds, and they select holes one after another. The birds do not like to live close to each other, so each bird selects a vertex such that the closest already selected vertex is as far from it as possible (in terms of distance on an unweighted tree). In case there are several such vertices, the bird selects the vertex with the least possible number. The first bird selects vertex 1.

When all birds settle, there will be only one empty hole left. Find which one it will be.

Input

The first line of input contains a single integer n, the number of vertices in the tree ($2 \leqslant n \leqslant 10^5$). Each of the next $n-1$ lines contains two integers u_i and v_i: the edges of the tree ($1 \leqslant u_i, v_i \leqslant n$).

Output

Print one integer: the number of a vertex with an empty hole. Vertices are numbered from 1 to n.

Example

standard input	standard output
3 1 2 2 3	2

Problem E. Einstein

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

This is an interactive problem.

Doubtlessly, Albert Einstein has done a lot for modern physics. However, there are some funny things attributed to him. For example, Einstein's five-houses riddle: a puzzle where you are given a bunch of information of the kind "the Brit lives in the red house" or "the person who smokes Pall Mall rears birds", and you should determine who lives in which house. There is a rumor that once Einstein's friends could not find out where is his own house because he told them his address with a puzzle! His friends made guesses: "I think you live in the house number x", and Einstein replied: "You are right!" or "You are wrong, and exactly k of your friends' guesses were closer to my house than yours". Luckily, now you don't have to solve any puzzles and may visit Einstein's museum in Bern, Kramgasse 49.

But if you want to practice in guessing, we can help. Jury has selected an **odd** number from 1 to n, and you have to guess it. You can post **even** numbers as queries, and for each query, you will receive the number of your previous queries which were closer to the selected number than the last query. Your task is to finally guess the selected number by posting an **odd** number as a guess.

You are allowed to make no more than 7 posts, including the final guess.

Interaction Protocol

You will play several games (test cases) in one test, from 1 to 100. When a new test case starts, you are given a line with one integer n ($1 \leqslant n \leqslant 100$, n is even). If $n = 0$, your program must quit immediately.

To make a query, print a single line "`?` x" with an even integer x ($0 \leqslant x \leqslant n$). A response to each query will be a single integer on a separate line: the number of your previous queries which were closer to the selected number then the last query.

To make a guess, print a single line "`!` y" with an odd integer y ($0 \leqslant y \leqslant n$). If you guessed correctly, the evaluation will proceed to the next test case. Otherwise, it will terminate with the outcome "`Wrong Answer`".

The total number of lines printed to solve a single test case can not be more than 7. If you print more lines, the evaluation will terminate with the outcome "`Wrong Answer`".

Do not forget to flush the output after each printed line. Otherwise, your solution will likely be terminated with the outcome "`Idleness Limit Exceeded`".

In each test case of each test, the number that you must guess is fixed and will be the same each time your solution is evaluated on that test case.

Example

standard input	standard output
2 6 0 1 1 0	 ! 1 ? 2 ? 6 ? 0 ! 3

Problem F. Fourier

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Given $n = 2^k$ complex numbers $a_0, \ldots, a_{n-1}$, their *discrete Fourier transform* $x_0, \ldots, x_{n-1}$ is defined as follows:

$$x_k = \sum_{j=0}^{n-1} a_j \cdot e^{-i \cdot 2\pi \cdot jk/n}.$$

You are given n integers $a_0, \ldots, a_{n-1}$. Produce n real numbers $b_0, \ldots, b_{n-1}$ such that all values of their Fourier transform are real (that is, their imaginary part is zero) and the total difference $\sum_{i=0}^{n-1} |a_i - b_i|$ is as small as possible.

Input

The first line of input contains one integer n ($1 \leqslant n \leqslant 2^{20}$, n is a power of two). The second line of input contains n space-separated integers $a_0, \ldots, a_{n-1}$ ($|a_i| \leqslant 10^9$).

Output

Print one real number: the minimal achievable total difference. Your answer will be accepted if the absolute or relative difference from the correct answer will not exceed 10^{-9}.

Example

standard input	standard output
4 1 2 3 4	2.000000000

Problem G. Galilei

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Galileo Galilei was a famous astronomer. He was the first to use the telescope to look at the sky. He discovered many celestial bodies, and it appeared that tracking all of them in mind was quite hard... There is a rumor that he invented a tracking system for his inventions and discoveries called *Galilei's inventions tracker*, or simply *git*. It allowed to save many versions of various information. For example, if you would like to look at some parameters of the star you observed a few months ago, you could just query this system. Your task is to reproduce the behavior of a simplified version of this tracker. A brief description follows.

A *commit* is a representation of the entire system state. Commits are named with sequential integers starting from 0. Initially, there is only one commit number 0. Commits are ordered as a rooted tree with root at 0: every commit except the root has exactly one parent. Commit a is called an *ancestor* of b if it is possible to come from b to a going up to parent several times. One commit is always selected as the *working copy*. Several operations are supported:

- `co` x, "checkout": select commit x as the working copy.
- `ci`, "commit": create a new commit whose parent is the current working copy and select it as the working copy. The new commit gets the lowest possible id.
- `re` x, "rebase": let your working copy be y. Find a commit t which is the lowest common ancestor of x and y. Copy the path from t to y (not including t) on top of x. New commits are numbered as follows: at first, the closest one to x gets the lowest possible number, then the second commit gets the next lowest possible number, and so on. The last copied commit becomes your working copy.

  If x is an ancestor of y, nothing happens and the path is not copied. If y is an ancestor of x, then the path is not copied and x becomes the working copy.

Your task is to implement all these operations.

Input

The first line of input contains one integer n ($0 \leqslant n \leqslant 10^5$). Each of the next n lines contains one of the following commands:

- `co x`: checkout at commit x
- `ci`: commit
- `re x`: rebase over commit x

It is guaranteed that the commit number is always valid. Additionally, it is guaranteed than after all operations, there will be no more than 10^{18} commits.

Output

After every rebase operation, print the number of the working copy commit after this operation.

Example

standard input	standard output
7 ci ci co 0 ci ci re 1 re 4	6 9

Problem H. Hadamard

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Hadamard matrix is a square matrix made of ± 1 such that every two rows of it are orthogonal. Two vectors $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ are called orthogonal if and only if $a_1 b_1 + \cdots + a_n b_n = 0$. A *canonical Hadamard matrix* of order k is a square matrix of size 2^k defined as follows:

$$\begin{cases} A_0 = (1) \\ A_k = \begin{pmatrix} A_{k-1} & A_{k-1} \\ A_{k-1} & -A_{k-1} \end{pmatrix} \end{cases}$$

In your thesis, you study Hadamard matrices with some special properties. You need a beautiful picture of the matrix for the title. You think the matrix is beautiful if its main diagonal is a sequence $x_1, \ldots, x_n$ (more formally, $a_{i,i} = x_i$ for all $1 \leqslant i \leqslant n$, where $x_i = \pm 1$. Of course, matrix should relate to Hadamard matrix someway, so you decided to take a canonical Hadamard matrix and rearrange its rows in a way that it becomes beautiful.

Input

The first line of input contains one integer k ($0 \leqslant k \leqslant 20$), the order of the matrix. The second line contains a string of length 2^k made only of characters "+" and "-": signs of values which should appear on the main diagonal.

Output

If the solution exists, print 2^k distinct integers from 1 to 2^k. The i-th of these integers must be the number of row of the canonical matrix which should be on position i.

If there is no solution, print "`Impossible`" (without quotes).

Example

standard input	standard output
2 +-++	3 4 2 1

Problem I. Iwatani

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You are playing Pac-Man and are deadly stuck on a level: every time you start this level, you are soon killed by ghosts. In this particular version of the game, ghosts appear only after T seconds after the level starts, and Pac-Man can move in any direction with a fixed speed of 1 unit per second. There are n cherries on the board, and in order to win, you have to collect at least half of them (not rounded!). Since the only problem for you is ghosts, you want to collect all required cherries in no more than T seconds.

However, due to your acquaintance with Mr. Toru Iwatani, the inventor of the game, you personally know from the author that it is possible by design to collect all cherries and even return to the starting point in no more than T seconds! However, you have to collect only half of them, and do not have to return to the starting point.

You start from the point $(0, 0)$.

Input

On the first line of input, there is an integer n: the number of cherries on the board ($1 \leqslant n \leqslant 5000$). Each of the next n lines contain two integers x_i, y_i: the coordinates of i-th cherry. The last line of the input contains a real number T ($0 < T \leqslant 10^{18}$, T contains at most 10 digits after the decimal point). All coordinates do not exceed 10^5 by absolute value. It is guaranteed that it is possible to collect all cherries and return to the origin in at most $T - 10^{-3}$ seconds.

Output

Print $\lceil \frac{n}{2} \rceil$ integers: the numbers of cherries which you will visit in the order you will visit them. The cherries are numbered from 1 to n as in the input.

Example

standard input	standard output
3 1 1 0 2 -1 1 5.66	3 2

Problem J. Jordan

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	256 mebibytes

Jordan measure is a method of measuring sets in $\mathbb{R}^n$ space. In this problem, we will consider only two-dimensional space (Cartesian plane). The measure of a set S is a nonnegative real number and is denoted as $\mu(S)$. In most cases, the measure of a figure is its area.

If S is a rectangle with sides parallel to coordinate axes and side lengths a and b, then $\mu(S) = a \cdot b$ and S is called a *simple rectangle*. If S is a disjoint union of a finite number of simple rectangles ($S = R_1 \sqcup \ldots \sqcup R_n$), then $\mu(S) = \mu(R_1) + \ldots + \mu(R_n)$ and S is called a *simple set*. Although Jordan measure can also measure non-simple sets, it is not needed in this problem.

You have a polygon with n vertices with sides parallel to coordinate axes and want to calculate its measure by definition. You know that your figure is a simple set, so you have to split it into some non-intersecting simple rectangles. You are very lazy and do not want to calculate the measure many times, so you have to split the figure into the **minimum possible** number of rectangles.

Input

The first line of input contains one integer n: the number of vertices in the polygon ($4 \leqslant n \leqslant 500$, n is even). Each of the next n lines contains two space-separated integers: the coordinates of polygon vertices in counter-clockwise order. Coordinates do not exceed 10^4 by absolute value. The polygon is guaranteed to be a simple set, to have non-zero area and to have no self-touches and no self-intersections.

Output

The first line of output must contain k, the minimum possible number of rectangles. Each of the next k lines must contain four integers: the coordinates of two opposite vertices of the rectangle. No two rectangles in the output can intersect. The union of all rectangles must exactly form the input polygon.

Example

standard input	standard output
6 0 0 2 0 2 1 1 1 1 2 0 2	2 0 0 2 1 0 1 1 2

Problem K. Kuratowski

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3.5 seconds
Memory limit:	256 mebibytes

Kuratowski-Pontryagin theorem is a forbidden graphs characterization of planar graphs. It states that a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 (the complete graph on five vertices) or of $K_{3,3}$ (complete bipartite graph on six vertices, three of which connect to each of the other three).

Little Max wants to check whether his favorite graph is planar. He is only a pupil and does not know the word "subdivision", so he at least wants to find a subgraph of his graph which is **exactly** K_5 or $K_{3,3}$. Help him! Find either K_5 or $K_{3,3}$, or report their nonexistence.

A subgraph of a graph is a subset of its vertices and its edges, where both endpoints of every selected edge are among the subset of selected vertices. As opposed to *generated subgraph*, any subset of edges can be taken, not necessarily all edges between selected vertices. For example, $K_{3,3}$ is a subgraph of K_6.

Input

The first line of input contains two space-separated integers n and m: the number of vertices and edges of the graph ($1 \leqslant n \leqslant 400$, $0 \leqslant m \leqslant \frac{n(n-1)}{2}$). Each of the next m lines contains two space-separated integers from 1 to n which denote an edge of the graph. There are no multiple edges and loops.

Output

If neither K_5 nor $K_{3,3}$ is a subgraph of the given graph, print the single word "`NO`".

If you found a K_5 subgraph, print "`K5`" on the first line and five integers on the second one: the numbers of vertices which constitute the subgraph.

If you found a $K_{3,3}$ subgraph, print "`K33`" on the first line and two more lines with three integers each: the numbers of vertices in two parts of the subgraph.

If there are several subgraphs of one or both types, print any one of them.

Example

standard input	standard output
6 9 1 4 1 5 1 6 2 4 2 5 2 6 3 4 3 5 3 6	K33 1 2 3 4 5 6