Problem A. Devil's Hell deLivery

Input file:	dhl.in
Output file:	dhl.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Devil's Hell deLivery company delivers literally everything: boxes, packages, packets, datagrams et cetera.

The delivery process from city A to city B works as follows. There are N trucks and K items. The capacity of truck number i equals c_i. The weight of item number j equals w_j. The trucks make one or more delivery steps.

During one step, some items are loaded into some trucks. Items can not be split. Each truck's capacity must not be exceeded by the total weight of the items assigned to it. All trucks depart simultaneously.

At the end of each step, all trucks come back to the place where they started. If there are any items left, another step is performed.

Your task is to distribute items among steps and trucks so that the number of steps is minimized.

Input

The input contains up to a hundred test cases.

Each test case starts with a single line containing two integers N and K ($1 \le N \le 5$, $1 \le K \le 9$): the number of trucks and the number of items, respectively. The following line consists of N integers $c_1, \ldots, c_N$ ($1 \le c_i \le 10^8$): the capacities of the trucks. The following line consists of K integers $w_1, \ldots, w_K$ ($1 \le w_i \le 10^8$): the weights of the items.

Output

For each test case, if the delivery is impossible, write a single line with a single integer -1.

Otherwise, start with a line containing a single integer S: the number of steps required. This number must be minimized. After that, write S lines describing the steps. Each step description must start with an integer I_i, the number of items delivered during the step. This number must be followed by I_i pairs a_j b_j. Each pair a_j b_j means that the item a_j is assigned to the truck b_j.

If there is more than one possible optimal answer, write any one of them.

Example

dhl.in	dhl.out
2 4 10 20 5 5 5 5 2 1 10 10 20	1 4 1 1 2 1 3 2 4 2 -1

Problem B. Domino on Torus

Input file:	`domino-on-torus.in`
Output file:	`domino-on-torus.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Imagine we take a stretchable square paper and cut a rectangle of size $A \times B$ along the grid lines. All squares are numbered from 1 to $A \cdot B$. From this rectangle, we cut another rectangle of size $C \times D$, again along the grid lines, so that sides of length C are parallel to sides of length A. Then, we glue the sides of length B together, and finally, glue the sides of length A together. What we got is a torus with a rectangular hole of $C \times D$ squares cut from it. (A torus is the surface of a bagel.)

The squares of the torus are considered different if they have different numbers.

We shall now tile the external surface of this torus with a hole using stretchable dominoes. Each domino consists of two squares of two different colors: one white and one black. The squares share a common side.

Our tiling has to satisfy the following condition: if two squares of the torus which are adjacent by side belong to different dominoes, they have to be the same color: both white or both black.

Two tilings are considered different if at least one of the following two conditions is satisfied:

1. at least one square of the torus is white in one tiling and black in another;
2. at least one square of the torus is covered by dominoes in such a way that the other square covered by the same domino is Y_1 in one tiling and Y_2 in another, and $Y_1 \neq Y_2$.

Input

The first line of input contains four integers A, B, C and D ($4 \leq A, B \leq 10^9$, $2 \leq C < A$, $2 \leq D < B$, all numbers are even).

Output

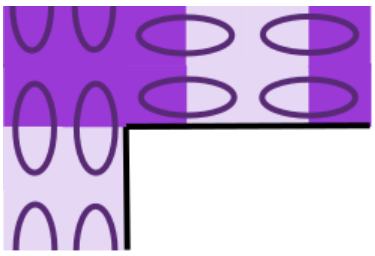
Output the number of tilings.

Example

domino-on-torus.in	domino-on-torus.out
4 6 2 4	4

Explanation

The following picture shows one of the possible tilings in the given example.



Problem C. Small Numbers Search

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

This is an interactive problem.

Jury has a permutation of numbers from 1 to n. Your task is find positions where numbers from 1 to k are placed. To do this, you can use jury's program which can compare numbers in any two positions in the permutation.

Input

The first line of input contains two integers n and k: the order of permutation and the number of positions to find. In all tests except the example, $n = 10\,000$ and $k \le 10$.

Then follow the answers for your requests, one per line. If the first of the two numbers to compare is less than the second one, the line will contain a single character "<", otherwise, it will contain a single character ">".

Output

If you want to compare numbers on positions i and j, you must print one line "`?` i j". Here, i and j must be different integers between 1 and n. You can request a comparison at most 10 700 times.

If you found all positions for all numbers from 1 to k, print "`!` pos_1 pos_2 ... pos_k", and then terminate your program.

To prevent output buffering, after printing each line, consider issuing the command which flushes the buffer. For example, this command may be `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

Also, don't forget to put a newline at the end of every line of your output.

Example

standard input	standard output
3 3 *(waiting for output)* < *(waiting for output)* > *(waiting for output)* <	*(reading input)* ? 1 2 *(reading input)* ? 3 1 *(reading input)* ? 2 3 *(reading input)* ! 1 2 3 *(terminating)*

Explanation

In the example, the jury's permutation is 1 2 3.

Problem D. Maze in a Forest

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

In this interactive problem, you have to get from the entry of an unknown maze to its exit spending not too much time.

The famous warrior Malcolm, The Hero of Arcania, walked at night through a dark forest. He walked for a long time and started to think he got lost, but suddenly ran into a stone wall on his path. The markings on the stone plate which were visible under fallen leaves clearly indicated that he discovered a maze of dwarven origin.

The hero knows little about the forest and the trees growing there. On the bright side, he spent much time in the company of dwarves, so now he can tell a lot about a maze by just looking at its markings.

> The markings on this maze form the characters "SK3". Character "S" means that the maze is a square, that is, the floor of the maze consists of $n \times n$ square cells of size 1×1 meters each. There is a thin solid wall around these cells with two openings. The first one is the entrance near which the hero stands. It is located in the southern wall of the southwestern corner cell of the maze. The second opening is the exit located in the northern wall of the northeastern corner cell of the maze.
>
> Each two cells of the maze sharing a side can be either connected by a passage or separated by a section of a thin inner wall, one meter long. In the maze, from each cell, one can move into any cell sharing a side with it if there is no wall between these cells. Additionally, one can freely move through the exit. The entrance is open only until the hero steps inside the maze. After that, it is immediately closed and opens again only when the hero arrives safely at the exit.
>
> Characters "K3" tell to an experienced maze walker that the maze has the following structure. First, from all possible sections of thin inner walls (there are $n \times (n-1)$ north-to-south sections in total, and just as many east-to-west sections), one third is selected at random (a non-integer number is rounded down) and considered in a random fixed order. Then, the sections are placed into the maze in that order. If some section would divide the square containing the maze into disconnected parts if it was placed, such section is discarded (not placed). Thus it is guaranteed that the resulting maze will be connected.

Malcolm became happy: he knows so much about the maze that he could certainly arrive at the exit. His goal is to start at the cell of the forest adjacent to the maze entrance and come to the cell of the forest adjacent to the maze exit. Besides, he should hurry: the person who travels from the entrance to the exit fast enough will see the treasure hidden in the forest near the exit. Of course, the above statement is true only if no one got to the treasure earlier.

On second thought, there is something to worry about. Firstly, Malcolm does not know the size of the maze: the length of the square side n can be any integer amount of meters from 10 to 200. Secondly, the maze is as dark as the forest around it, so Malcolm has to move almost blindly. At the start of each step, the hero chooses one of four main directions: north, east, south or west. Then he tries to move by one meter in that direction. If the hero runs into an obstacle, he remains where he was. Otherwise, he moves to the neighboring cell in the selected direction. The time required for each step does not depend on the result of the step, so what matters is the total number of steps.

Help the hero to move in such a way that he arrives at the required cell fast enough. The solution will be considered correct if the hero makes at most $5 \cdot n + 300$ steps and arrives at the cell outside of the maze exit.

Interaction Protocol

In this problem, the jury program reacts on the commands that a solution issues to the standard output. Each command determines Malcolm's next step. A command is written on a separate line and consists of one capital English letter: "`N`" for stepping to the North, "`E`" for stepping to the East, "`S`" for stepping to the South и "`W`" for stepping to the West.

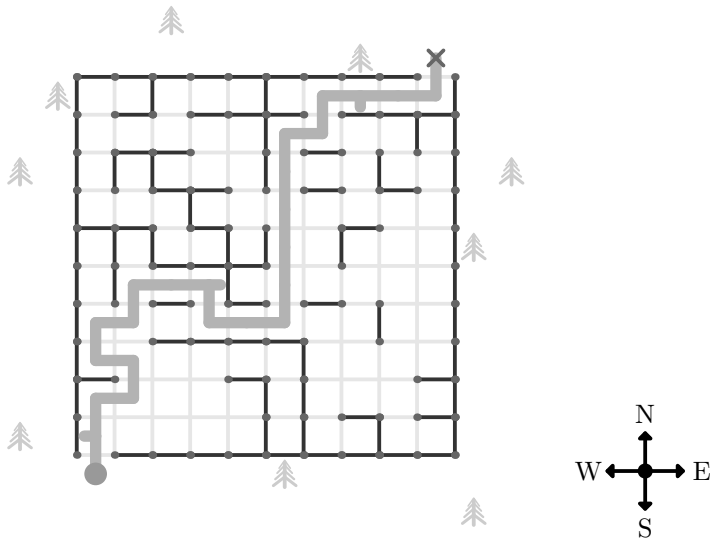
For each issued command, the jury program immediately responds with a line passed to the solution's standard input which contains one word written in lowercase English letters: "`no`" if an obstacle denied the passage, "`ok`" if the step succeeded, but the hero did not arrive at the target cell, and "`end`" if the step succeeded and the hero found himself at the target cell. After recieving "`end`" as a response, the solution must terminate without any further output.

To prevent output buffering, flush the output buffer after each command you issue: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

In each test, the maze is fixed in advance after being randomly generated.

If a solution made at least 100 000 steps, but did not yet exceed the time limit and did not arrive at the target cell, testing is terminated with "`Wrong Answer`" outcome.

Example



Explanation

The picture above shows the maze which is generated in the first test. Malcolm starts at the cell labeled with a circle and has to arrive at the cell labeled with a cross. This maze can be passed, for example, using the following sequence of commands (below the commands are the results of their execution):

N	W	N	E	N	W	N	E	N	E	E	E	S	E	E	N	N	N	N	N	E	N	E	S	E	E	N
ok	no	ok	ok	ok	ok	ok	ok	ok	ok	ok	no	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	no	ok	ok	end

On the picture, this solution is shown by a thick gray line.

Problem E. Grammar

Input file:	`grammar.in`
Output file:	`grammar.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

A *formal grammar* is a way of describing formal languages as $\Gamma = \langle \Sigma, N, S \in N, P \subset N^+ \times (\Sigma \cup N)^* \rangle$ where Σ is called an *alphabet* and its elements are called *terminals*, N is a set of *nonterminals*, S is the starting nonterminal, and P is a set of *production rules* of the form $\alpha \to \beta$.

Here, N^+ contains all strings of one or more elements of N (non-empty strings of nonterminals), and $(\Sigma \cup N)^*$ consists of all strings of zero, one or more elements of $(\Sigma \cup N)$ (strings of terminals and nonterminals, including the empty string).

A grammar is called *context-free* if the left side of each production rule consists of exactly one nonterminal, more formally, $P \subset N \times (\Sigma \cup N)^*$.

For example, let us consider a grammar from the second example test case with alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, set of nonterminals $N = \{S, A\}$ and two production rules:

1. $S \to \mathtt{b}A$
2. $A \to \mathtt{aa}$

One can easily see that it is a context-free grammar.

To create the language generated by a grammar, one needs to start from a string consisting of only start nonterminal S, and then apply production rules one or more times. Applying a production rule is the procedure of finding the left side of that rule somewhere in the current string and replacing it by the string from the right side of that rule. The *language* generated by Γ is the set of all strings consisting **only** of terminals that can be produced by applying production rules one or more times.

For example, there is a string `baa` in the language generated by the grammar described above. To produce it, one could apply productions $S \to \mathtt{b}A \to \mathtt{baa}$. There are no other strings in the language generated by this grammar.

Some grammars may even generate infinite languages, others may generate empty ones.

You are given a context-free grammar with an alphabet consisting of two terminals "`a`" and "`b`". Your task is to check whether the language generated by this grammar contains a string consisting of strictly more characters "`a`" than characters "`b`".

Nonterminals in this task are enumerated from 1 to n. The starting nonterminal always has number 1.

Input

The input consists of one or more test cases.

The first line of each test case contains two integers n and m: the number of nonterminals and the number of production rules ($1 \le n \le 100$, $1 \le m \le 50\,000$).

Each of the next m lines describes one production rule in the following manner. At first, A_i and k_i are given: the number of left side nonterminal ($1 \le A_i \le n$) and the number of characters on the right side of the production rule ($0 \le k_i \le 100$). Then k_i objects follow, each of them is either a nonterminal $B_{i,j}$ ($1 \le B_{i,j} \le n$) or a terminal "`a`" or "`b`". Consecutive characters are separated by single spaces.

The total sum of all n over all test cases does not exceed 1000. The total sum of all m over all test cases does not exceed 50 000. The size of the input does not exceed 5 megabytes.

The input is terminated by a string of two zeroes.

Output

For each test case, output a separate line. It must contain "`YES`" if the language generated by the given

grammar contains a string consisting of strictly more characters "a" than characters "b", otherwise the line must contain "NO".

Example

grammar.in	grammar.out
2 2 1 2 a 2 2 1 b 2 2 1 2 b 2 2 2 a a 2 2 1 2 b 2 2 3 a a 1 0 0	NO YES NO

Problem F. Yet Another Point Searching Problem

Input file:	`minwdist.in`
Output file:	`minwdist.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are given n points on the plane: $A_1, A_2, \cdots, A_n$. Point i has weight w_i. Find such point B that the maximum weighted distance $\max\limits_{i=1}^{n} w_i \cdot |A_i B|$ is minimal possible.

Input

The input consists of one or more test cases.

On the first line of each test case, there is an integer n: the number of points ($1 \le n \le 500\,000$). Each of the next n lines contains three integers: x_i, y_i and w_i. Each of these numbers does not exceed 10^7 by absolute value. All weights are strictly positive.

The test cases follow one another without any gaps. The input is terminated by a line containing a single integer 0. This line must not be considered a test case. The sum of all n in the input does not exceed 500 000. There are no more than 1000 test cases in the input.

Output

For each test case, print two real numbers: the coordinates of point B. Your answer will be considered correct if the absolute or relative error of the maximum weighted distance will be less than 10^{-9}.

Example

minwdist.in	minwdist.out
2 2 2 1 0 0 1 3 0 0 1 6 0 2 0 6 3 0	1.0 1.0 2.4 3.6

Problem G. Nanobugs

Input file:	`nanobugs.in`
Output file:	`nanobugs.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

In this problem, for a collection of bugs each of which is either a spy or an agent, you have to show the true number of spy bugs while not disclosing any spy or agent.

Nanobugs are intelligent robots who obey the orders of their masters. They are usually busy spying, eavesdropping or monitoring other nanobugs. All nanobugs look exactly the same.

Bartosz and Vivek are engineers of two corporations: Eastern Cartel and Southern Trading Company. They have to examine the meeting room in the office of Eastern Cartel where representatives of their corporations will discuss a new contract. Together, they managed to uncover and catch n nanobugs. Each of the bugs is either an Eastern Cartel's security agent or a Southern Trading Company's spy.

Anders is a security expert from SpyTek. His job for today is to help Bartosz and Vivek determine how many spies and agents are among the nanobugs they caught. For each bug, Anders knows whether it is a spy or an agent, but no other person has that knowledge.

Anders sees that there are exactly a spies among the bugs. Bartosz and Vivek, on the other hand, have only learned by standard procedures that there are either a or b spies, and the numbers a and b differ by exactly one. They have no other information about the bugs.

Nanobugs are expensive intelligent machines which especially value their incognito. Sure, being an expert, Anders knows who is the master of each of the bugs. But if for some nanobug, its affiliation (which company the nanobug works for) became known to the engineers or other nanobugs, further functioning of that nanobug (and its very existence!) would be threatened.

As a security expert, Anders has an instrument which will help him in his work: a balance checker. It is a small box with two chambers and an indicator light. Using a balance checker is simple: Anders places some nanobugs in each of the chambers and presses a button. After that, for each corporation, the instrument checks that the number of nanobugs affiliated with it is the same in both chambers. If this is true for both corporations, the indicator light is green, otherwise, it is red.

Anders plans to perform a series of checks with the balance checker. For that, he will start by placing all nanobugs in a row in the order he likes. During each check, Anders will place some nanobugs in the first camera, some in the second camera, press the button, show the result to the engineers, and then return all nanobugs to their places in the row, restoring their original order. With such procedure, it can be said that, during the whole series of checks, the nanobugs are numbered by integers from 1 to n according to their places in the row.

Help Anders prove to the engineers that there are exactly a spies among the n nanobugs they caught, and do it in a way that does not disclose any nanobug's affiliation, or determine that this is not possible.

Input

The first line of input contains three integers n, a and b ($20 \le n \le 50$, $1 \le a, b \le 4$, $|a - b| = 1$). Here, n is the total number of nanobugs. Bartosz and Vivek know that there are either a or b spies among them, and Anders has to prove that the exact number of spies is a.

Output

On the first line, print one integer: the number of checks m ($0 \le m \le 1000$) or -1 if it is impossible to satisfy all conditions. In case of positive answer, print m lines, one for each check.

The description of each check contains exactly the information that Anders shows to the engineers, and consists of three parts. The first part determines which nanobugs are placed in the first chamber: it is denoted by the number of these bugs followed by their numbers in any order. Then follows a character which shows the result of the check: "=" (ASCII code 61) for positive result (green indicator light) or "^"

(ASCII code 94) for negative result (red indicator light). After that, the bugs which are placed in the second chamber are listed in the same format as for the first chamber. All the above is printed in one line, and consecutive numbers or characters are separated by one or more spaces.

No bug can be mentioned twice during any one check.

Note

Note that it is not disclosed anywhere which of the bugs were actually spies and which were agents. This information is available only to Anders and, possibly, to your solution. It must not be disclosed to any other party.

Example

nanobugs.in	nanobugs.out
22 2 1	4 6 1 2 3 4 5 6 = 6 7 8 9 10 11 12 5 13 14 15 16 21 = 5 17 18 19 22 20 3 13 14 17 ^ 3 6 8 9 1 1 ^ 2 2 3

Explanation

In the example, the engineers know that among the 22 nanobugs they caught, there are either one or two spies. Actually, there are exactly two spies, and Anders' job is to prove that. In the presented answer, he decided to perform a series of four checks.

The first check shows that among the first six bugs, there are as much spies as there are among the next six bugs. Here and further on, the same statement is true for agents: if the number of nanobugs in both chambers is the same, the number of spies is the same if and only if the number of agents is the same.

After the second check, we may conclude that among the bugs numbered 13, 14, 15, 16 and 21, there are as much spies as there are among the bugs numbered 17, 18, 19, 22 and 20.

The third check shows that the number of spies among the bugs 13, 14 and 17 is necessarily different from the number of spies among the bugs 6, 8 and 9.

Finally, the fourth check does not provide any information since a check with different number of bugs in the chambers always gives a negative result.

Now, which bugs may have been spies?

Suppose, for example, that Anders initially placed the two spies in positions 2 and 8. Then the first check gave a positive result, as there were five agents and one spy in each of the two chambers. The second check gave a positive result since there were no spies in any of the chambers. The third check gave a negative result: there are no spies among the bugs 13, 14 and 17, but there is one spy among the bugs 6, 8 and 9. The fourth check does not depend on the positions of spies. To conclude, all checks gave the required results. So, the *possibility* that there are exactly two spies is shown.

Suppose now in place of the engineers that there was in fact one spy, and its number was 1. But then, the first check would have given a negative result. Similarly, it can be shown that if there is a single spy which has number 2, 3, ..., 22, at least one of the checks would have given a different result. So, Anders proved the *impossibility* of a situation in which there was exactly one spy among the nanobugs. Recall now that Bartosz and Vivek know there are either one or two spies. It turns out that Anders proved that the number of spies is exactly two.

What remains is to check that no agent and no spy is disclosed. For that, we can present a set of possible arrangements of spies such that each of these arrangements gives the required results during the checks, and furthermore, each nanobug is an agent in at least one of these arrangements, and each nanobug is a spy in at least one of these arrangements. Such a set is, for example, the set of arrangements where the spies have numbers $(1, 8)$, $(2, 9)$, $(3, 8)$, $(4, 9)$, $(5, 8)$, $(6, 7)$, $(6, 10)$, $(6, 11)$, $(6, 12)$, $(13, 17)$, $(13, 18)$, $(13, 19)$, $(13, 20)$, $(13, 22)$, $(14, 18)$, $(15, 17)$, $(16, 17)$ and $(17, 21)$.

Problem H. Non-Shortest Path

Input file:	nsp.in
Output file:	nsp.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

In this problem, you have to travel from one corner of a checkered 4×4 field to the opposite corner using any simple path which is not the shortest one.

There is a checkered field on a plane consisting of 4×4 cells. Each cell of the field is either empty or fully occupied by a wall.

Robot Aurelius is passing a test to measure the intelligence level by solving various problems on that field. He has already completed the parts which involved searching for an arbitrary path and the shortest path between a pair of cells.

The next task requires Aurelius to walk from the upper left corner of the field to the lower right corner by moving through empty cells. A single step consists in moving from the current cell to any other cell which shares a side with the current one. The path must be simple, that is, for each cell, the robot can not visit that cell more than once. Additionally, the length of the path must be strictly greater than the length of the shortest path between the corners on the given field. The length of the path is the number of steps on that path.

Write a program that will help robot Aurelius accomplish the task.

Input

The input contains one or more test cases. Each test case is given on four lines. Each of these lines describes one row of the field and contains exactly four characters defining the cells of that row. An empty cell is denoted by character "`.`" (dot, ASCII code 46), and a wall is denoted by character "`#`" (hash, ASCII code 35). Consecutive test cases are separated by a line consisting of four characters "`-`" (dash, ASCII code 45).

It is guaranteed that the upper left cell and the lower right cell are empty. Additionally, it is guaranteed that all test cases in a single input are distinct.

Output

For each test case, print a line defining the required path. The line must consist of characters corresponding to the commands for moving to an adjacent cell in the order of their execution: "`D`" to move down, "`U`" to move up, "`L`" to move left, and "`R`" to move right. If there are several possible paths, print any one of them. If there is no path with the required properties, print the number "`-1`" instead of the path.

Example

nsp.in	nsp.out
.#.. .... ..#. .#.. ---- ..## ...# #... .#.. ---- ...# ..#. .#.. #...	DDRURURDDD -1 -1

Explanation

The example contains three test cases.

In the first test case, the length of the shortest path is six steps: the only shortest path is "DRRRDD". Other than that, there are three simple paths which are not the shortest: these are the paths "DDRURRDD" and "DRRURDDD" of length 8, and the path "DDRURURDDD" of length 10. Any of them can be printed.

In the second test case, there are eight possible simple paths, but they all have the same length, and thus all are the shortest paths.

In the third test case, there is no path from the upper left corner of the field to the lower right one consisting of empty cells.

Problem I. Composition of Polynomials

Input file:	`polycomp.in`
Output file:	`polycomp.out`
Time limit:	2 seconds (*3.5 seconds for Java*)
Memory limit:	256 mebibytes

You are given polynomials $f(x)$, $g(x)$, $h(x)$ over field $\mathbb{Z}/2\mathbb{Z}$.

Find the polynomial $f(g(x)) \bmod h(x)$.

Input

The first three lines of input contain polynomials f, g and h, one per line. Each polynomial p is descibed as n p_0 p_1 p_2 ... p_n ($1 \le n \le 4000$, $p_i \in \{0, 1\}$ for all i, and $p_n = 1$). The polynomial $p(x)$ is then equal to $p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n$.

Output

Print the resulting polynomial in the same format.

If the answer is the null polynomial, print it as "`0 0`".

Examples

polycomp.in	polycomp.out
5 0 1 0 1 0 1 2 1 1 1 4 0 1 1 0 1	1 1 1
2 1 1 1 3 0 0 1 1 4 1 0 1 0 1	3 1 0 0 1

Note

Let us recall some definitions.

The field $\mathbb{Z}/2\mathbb{Z}$ is a set of two elements 0 and 1 where results of addition, subtraction, multiplication and division are remainders modulo 2 of the corresponding results for ordinary integers.

A polynomial $f(x)$ over this field is an expression of the form $f_n \cdot x^n + f_{n-1} \cdot x^{n-1} + \ldots + f_1 x + f_0$, where coefficients f_n, ..., f_0 are integers from $\mathbb{Z}/2\mathbb{Z}$, and the variable x can hold values from $\mathbb{Z}/2\mathbb{Z}$ too. The maximum integer n such that $f_n \neq 0$ is called the degree of the polynomial $p(x)$.

Polynomials $a(x) = \sum\limits_k a_k x^k$ and $b(x) = \sum\limits_k b_k x^k$ are equal if a_k и b_k are equal for all k.

Addition and subtraction of polynomials are performed component-wise: $a(x) \pm b(x) = \sum\limits_k (a_k \pm b_k) \cdot x^k$.

The product of polynomials $a(x)$ and $b(x)$ is $c(x) = \sum\limits_k c_k x^k$ where $c_s = \sum\limits_{t=0}^{s} (a_t \cdot b_{s-t})$.

Polynomials can be divided by each other. For a non-null polynomial $b(x)$, we say that $a(x)/b(x) = q(x)$ and $a(x) \bmod b(x) = r(x)$ if $q(x) \cdot b(x) + r(x) = a(x)$ and the degree of $r(x)$ is strictly less than the degree of $b(x)$. It can be shown that $q(x)$ and $r(x)$ are uniquely defined.

Composition $a(b(x))$ is the polynomial $\sum\limits_k a_k (b(x))^k$ where the power of a polynomial is defined via multiplication: $(b(x))^0 = 1$, $(b(x))^1 = b(x)$, $(b(x))^p = b(x) \cdot (b(x))^{p-1}$ for $p > 1$. To find the coefficients, expand the expression and sum the coefficients for the same powers of x.

Problem J. Potential

Input file:	potential.in
Output file:	potential.out
Time limit:	3 seconds
Memory limit:	256 mebibytes

You are given a weighted directed graph. Let each vertex i have potential Φ_i. Let w_{uv} be the weight of the edge (u, v). Then, define the new weight as $w'_{uv} = w_{uv} + \Phi_u - \Phi_v$.

Find such integer potentials Φ_i that the weights w' for all edges will be equal.

Input

The first line of input contains an integer t, the number of test cases.

Each test case starts with a line containing two integers n and m: the number of vertices and edges in the graph ($1 \le n \le 300\,000$, $0 \le m \le 300\,000$). Each of the next m lines contains three integers x_i, y_i and w_i: start vertex, end vertex and weight of an edge ($1 \le x_i, y_i \le n$, $-10^9 \le w_i \le 10^9$). It is guaranteed that there are no self-loops and no multiple edges in the graph.

That the sum of all n and all m is guaranteed to not exceed 600 000.

Output

For each test case, on the first line, print "YES" if an integer solution exists, or "NO" otherwise.

If the answer is positive, on next line, print n integers: the potentials of the vertices. The potentials must not exceed 10^{18} by absolute value. It is guaranteed that, if a solution exists, there also exists a solution satisfying the above requirement.

If there is more than one solution, output any one of them.

Example

potential.in	potential.out
2 5 4 1 2 -1 2 3 2 3 4 1 4 5 179 5 5 1 2 1 2 3 1 3 4 1 4 5 0 5 1 2	YES 0 -1 1 2 181 YES 0 0 0 0 -1

Problem K. What the Flex?

Input file:	wtflex.in
Output file:	wtflex.out
Time limit:	0.5 seconds
Memory limit:	256 mebibytes

In this problem, you have to find the next integer in a particular ordering with has the same set of prime divisors.

Archaeologist Kris studies the ancient platform Flex which was recently found near Adobe creek. On this platform, the numbers of different versions of teleporters which have been created by The Elders were once inscribed. Each version number is an integer between 1 and N. On the platform, there are several columns of numbers, one for each teleporter. Each column once listed all created versions of a teleporter in the order they were built. Unfortunately, some of the numbers were erased.

Kris just finished studying version a of teleporter b, and he is now ready to examine the next version of this teleporter. But he does not even know its number! On the other hand, any program which is a correct solution of this problem must know it. Below is what Kris already knows about the version numbers of the teleporters:

1. Each number from 1 to N is the number of some version of some teleporter.
2. Different versions of **one** teleporter have different numbers.
3. Each teleporter has its own set of prime numbers. Later, we consider one fixed teleporter. Let its prime number set be $X = \{p_1, p_2, \ldots, p_m\}$ (we list them so that $p_1 < p_2 < \cdots < p_m$).
4. It is known that any version number of this teleporter is divisible by each prime from X and not divisible by primes not in X. It means that such number can be written as $p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$ where $k_i \geq 1$. So, each version of this teleporter can be mapped to a tuple of m positive integers $(k_1, k_2, \ldots, k_m)$.
5. Version $\alpha = (k_1, k_2, \ldots, k_m)$ was built before version $\beta = (l_1, l_2, \ldots, l_m)$ if and only if there is such i $(0 \leq i < m)$ that $k_1 = l_1$, $k_2 = l_2$, $\ldots$, $k_i = l_i$, but $k_{i+1} < l_{i+1}$. It can be said that the versions are ordered lexicographically by their tuples.

Input

The only line of input contains two integers — the version number a of a teleporter and the number N $(1 \leq a \leq N \leq 10^{18})$.

Output

If there exists a version of the same teleporter which was built just after version a, print its number. Otherwise, print "-1".

Examples

wtflex.in	wtflex.out
6 13	12
12 13	-1