Problem A. Auxiliary Question of the Universe

Input file:	`auxiliary.in`
Output file:	`auxiliary.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

As you probably know, scientists already discovered the Ultimate question of life, the Universe, and everything, and it is "What do you get if you multiply six by nine?". Not satisfied by this, the scientists contracted a small Magrateyan company to construct a mini-computer to find out some more specific question (they named it *auxiliary*), which can theoretically shed more light on life, the Universe or something else.

This computer was built, but unluckily (although not unexpectedly) the result of computation was corrupted and partially lost. Finally the computer constructors managed to receive a string, which is a part of the correct question. After thorough analysis the constructors started to believe that the original result can be reconstructed from the string by adding some letters to it without the string letters being reordered or removed. Also they believe that the correct result is an arithmetic expression (as with the Ultimate question), but since the question is auxiliary, it contains no multiplication, only addition. More precisely, it should correspond to the following grammar:

$$\begin{array}{rcl} \langle\text{expression}\rangle & ::= & \langle\text{term}\rangle \mid \langle\text{term}\rangle\ `\texttt{+}'\ \langle\text{expression}\rangle \\ \langle\text{term}\rangle & ::= & \langle\text{number}\rangle \mid `\texttt{(}'\ \langle\text{expression}\rangle\ `\texttt{)}' \\ \langle\text{number}\rangle & ::= & `\texttt{0}' \ldots `\texttt{9}'\ \ [\langle\text{number}\rangle] \end{array}$$

The constructors do not want to risk again, and they need your help to give just something to their clients. They ask you to reconstruct the question based on the corrupted computer answer which they managed to retrieve.

Input

The input file contains exactly one line — the corrupted auxiliary question. It is a non-empty string which is at most 1000 symbols long. This string contains only symbols '`+`', '`(`', '`)`', and '`0`', . . . , '`9`'.

Output

Output the reconstructed auxiliary question. It's guaranteed that there exists a correct question of less than 5000 symbols and your solution must also be shorter than that. If there is more than one solution, output any one.

Example

auxiliary.in	auxiliary.out
`1+0+1)`	`(1+0+1)`
`2009`	`2009`
`)(()(`	`(0)+((0)+(0))`

Problem B. Bureaucracy

Input file:	`bureau.in`
Output file:	`bureau.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Long ago, in a kingdom far, far away the king decided to keep a record of all laws of his kingdom. From that moment whenever a new law was passed, a corresponding record was added to the law archive.

Many centuries later lawyers discovered that there were only two types of laws in the kingdom:

- *direct law*, that states a new norm;
- *canceling law*, that cancels one of the previous laws.

The law is considered *active* if and only if there is no active law that cancels it.

You are to write program that finds out which laws are still active.

Input

The first line of the input file contains an integer number n ($1 \le n \le 100\,000$) — the number of passed laws.

The following n lines describe one law each. Each description has one of the following formats:

- "`declare`", meaning that a direct law was passed.
- "`cancel` i", where i is the number of law being cancelled by this one.

The laws are numbered from one.

Output

The first line of the output file must contain the number of active laws. Following lines must contain numbers of these laws listed in increasing order.

Example

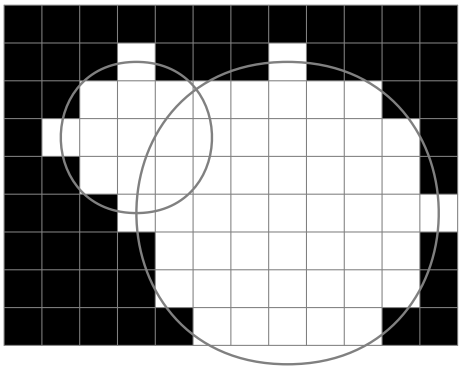
bureau.in	bureau.out
5 declare cancel 1 declare cancel 2 cancel 3	3 1 4 5

Problem C. Circles on a Screen

Input file:	`circles.in`
Output file:	`circles.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Yesterday Andrew wrote a program that draws n white circles on a black screen. The screen is monochrome and it has a resolution $w \times h$ pixels. Pixels are numbered from upper left corner $(0, 0)$ to bottom right one $(w-1, h-1)$.

A circle with the center at pixel (x_c, y_c) and the radius r consists of the pixels with coordinates (x, y) such that $\sqrt{(x_c - x)^2 + (y_c - y)^2} \le r$. If the circle does not fit on the screen, it is truncated. If some pixel belongs to two or more circles, it is white.



The resulting picture was very nice, so Andrew decided to copy it to his wall. He has white wallpaper and he can only draw some parts of wall into black. Now he wants to know the amount of paint he needs. He copies the picture exactly pixel-to-pixel, so you should write a program that calculates the number of black pixels left on a screen after drawing n circles.

Input

In the first line of input file there are three integers: w, h, and n ($1 \le w, h \le 20\,000$; $1 \le n \le 100$). Each of the following n lines contains descriptions of the circle. In $i+1$-th line there are three integers: x_i, y_i, r_i ($0 \le x_i < w$; $0 \le y_i < h$; $0 \le r_i \le 40\,000$). They denote a circle with the center at pixel (x_i, y_i) and radius r_i.

Output

You should output exactly one number — the number of black pixels left on the screen.

Example

circles.in	circles.out
5 3 2 1 1 1 3 1 1	6
12 9 2 3 3 2 7 5 4	51

Note: The picture corresponds to the second example.

Problem D. Dragon's Question

Input file:	`dragon.in`
Output file:	`dragon.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

In a land far-away there lives a noble man, and he has three sons. The elder of them is very clever, his especial strength is calculation: he can easily count a determinant of fifth degree in his mind without paper and pencil. The middle brother is also very talented, he is particularly strong in theoretic questions. But the younger brother has absolutely no talent in mathematics.

One day they went for a walk. Suddenly a wind started to blow and something closed the sun from them: it was a hungry dragon, returning to his lair from unsuccessful hunt.

"Hey, boys. I will give you a problem, and if you do not solve it, nothing will save you!" — said the dragon.

The elder brothers smiled ironically. Of course, they were so clever that no dragon could ask them a question they were not able to answer.

"Give me a positive integer number which is divisible by d and has exactly n digits in it, assuming that d is equal to forty-five and n is equal to three!" — was the dragon's question.

"One hundred and thirty-five." — answered the elder brother.

"Good, go where you want. But I will return and ask you a similar question in a year." — said the upset hungry dragon and flew away.

A year passed, and the elder brother got married and left his parents' home. Two younger brothers went for a walk discussing this event, and met the dragon again.

"Hey boys, give me a positive integer number which is divisible by twenty three and has exactly one digit in it" — asked the dragon.

"No solution" — answered the middle brother.

"You are still too clever, go where you want. But I will return and ask you a similar question." — said the dragon and flew away.

Another year passed and the middle brother got married and left his parents' home. The younger brother now does not go outside, because he does not have enough knowledge to answer the dragon's questions. Please, help him and write a program — the boy is very afraid.

Input

The input file contains the only line with numbers n and d ($1 \le n \le 1000$; $1 \le d \le 1\,000\,000$).

Output

The first and only line of the output file must contain the answer to be given to the dragon — either a n-digit number (without leading zeroes) divisible by d or a string "`No solution`".

Example

dragon.in	dragon.out
`20 1`	`10000000000000000000`
`1 23`	`No solution`
`1 4`	`4`

Problem E. Enigmatic Device

Input file:	`enigmatic.in`
Output file:	`enigmatic.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Yes, it happened! The first contact! Aliens will visit the Earth in 2010! And they promised to bring an enigmatic device which cannot be constructed using existing Earth technologies. Most of the scientists of the world think so! All newspapers already published their leading articles about it.

This device will accept an integer sequence $\{a_i\}$ as its initial input. After that, it can perform the following two operations:

1. Take an interval $[l;r]$ and perform $a_i \leftarrow a_i^2 \bmod 2010$ for all a_i such that $l \le i \le r$.
2. Take an interval $[l;r]$ and output the sum of all a_i such that $l \le i \le r$. Note that the sum is *not* taken modulo 2010.

The amazing thing about this device is that it is able to perform 50 000 operations of this kind with a sequence of 50 000 numbers within 3 seconds. Nobody could do it before!

But Roman does not believe in aliens and thinks that it is only a great hoax made by somebody just to win another million bucks on the stock exchange. His goal is to prove this. So he hired you to write a program to simulate this device.

Given an integer sequence a_i and a sequence of operations, write a program which simulates the behaviour of the strange alien device.

Input

The first line of the input contains the length of the sequence n ($1 \le n \le 50\,000$). The second line contains n numbers a_i forming the initial sequence ($0 \le a_i \le 2009$). The third line contains the number of operations m ($1 \le m \le 50\,000$). The rest of file contains m lines, each describing one operation. The j-th operation is described by its kind k_j (‘`1`’ for squaring, ‘`2`’ for calculating the sum), followed by two integers l_j and r_j ($1 \le l_j \le r_j \le n$).

Output

For each operation of the second kind, write their output on the separate line, in order they appear in the input.

Example

enigmatic.in	enigmatic.out

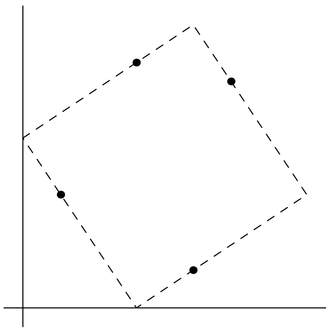
`3`
`17 239 999`
`4`
`2 1 3`
`1 2 3`
`2 2 3`
`2 1 2`
	`1255`
	`1882`
	`858`

Problem F. Four Points

Input file:	`four.in`
Output file:	`four.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Mike is a magician. One of his inventions is a labyrinth that gives supernatural abilities to every person who walks through it. The labyrinth has an extremely complicated internal structure, however, for an external observer it is just a square on the ground.

Mike has found some suitable place for labyrinth on the seashore. He drew its border on the sand and marked four points with small stones so that each side of the square contained exactly one stone and no stone was placed in the corner.



As no picture drawn on the sand stays forever, after a while Mike found only the stones on their places. Now he wonders where the marked square could have been.

Your task is to restore some possible place of the labyrinth and return four corners of the square as a result. You may assume that the seashore is a plane and the stones are points on it.

Input

The first four lines of the input file contain two integer numbers x_i and y_i each — coordinates of the i-th point ($-1\,000 \le x_i, y_i \le 1\,000$). No two points coincide, no three points are collinear.

Output

Output four lines containing two real numbers each — coordinates of the vertices of the square. Vertices should be listed in either clockwise or counterclockwise order. Coordinates must be precise up to 6 digits after the decimal point.

If there are multiple solutions, output any of them. If there is no solution, write four pairs of zeroes instead of the coordinates.

Example

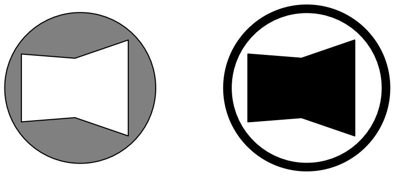
four.in	four.out
6 13 11 12 9 2 2 6	6 0 15 6 9 15 0 9
0 0 5 5 5 0 3 2	0 0 0 0 0 0 0 0

Problem G. Grand Theft Auto Wheel

Input file:	`gtaw.in`
Output file:	`gtaw.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Tommy is a wheel thief. His job was formerly as easy as pie: you lift a car, turn off wheel bolts, take the wheel and run away. But now everybody uses “anti-theft” bolts.

Anti-theft bolt is designed in such a way that it cannot be turned off with a usual wrench. Its head is a cylinder with a hole. To turn the anti-theft bolt off you need a right wrench. The wrench has a ring with a lug that exactly matches the shape of the bolt head.



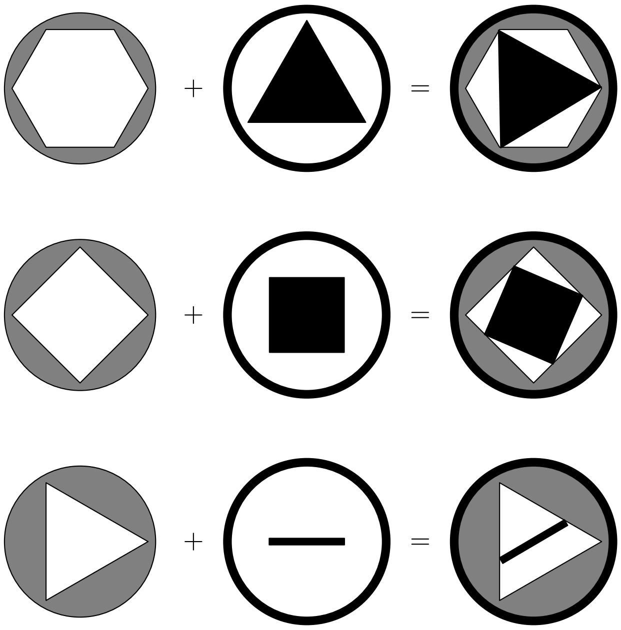
Bolt head and corresponding wrench.

Of course Tommy cannot get wrenches for all possible anti-theft bolts. But sometimes it is possible to turn off the bolt with the wrench that does not match it exactly.

More formally, the wrench can turn off the bolt if and only if two following conditions are satisfied:

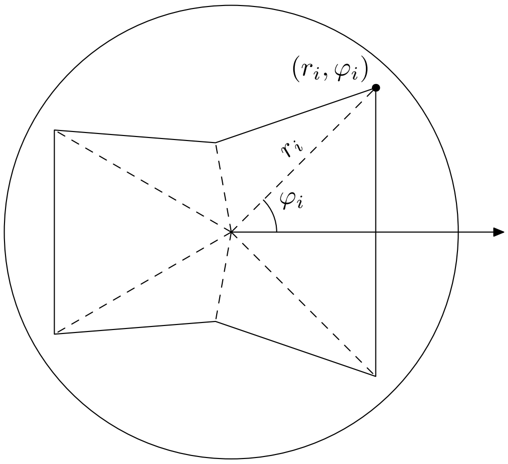
- the ring of the wrench can be joined with the cylinder of the bolt head in such a way that the lug of the wrench is inside the hole of the bolt head;
- the wrench cannot make a full turn when the bolt is fixed.

For example:



Situations where the bolt can be turned off with improper wrench.

Due to technical reasons, the shape of both — hole of the bolt head and lug of the wrench, are always a star-shaped polygons with theirs centers in the center of the bolt or wrench. So if it is described in polar coordinate system as a sequence of pairs (r_i, φ_i) then $\varphi_{i+1} < \varphi_i$ and $\varphi_{i+1} - \varphi_i < 180°$.



Help Tommy do find out if it is possible to turn off the bolt with the wrenches he has.

Input

The first line of input file contains two integer numbers n and r — the number of wrenches and the radii of the bolt head and the wrenches' rings ($1 \le n \le 10$, $1 \le R \le 1000$).

The following lines describe the bolt head. Description consists of an integer number m — number of vertices ($3 \le m \le 100$) and m pairs of integer numbers (r_i, φ_i) ($1 \le r_i < R$; $0^\circ \le \varphi_i < 360^\circ$; $\varphi_i < \varphi_{i+1}$; $\varphi_{i+1} - \varphi_i < 180^\circ$; $\varphi_m - \varphi_1 > 180^\circ$).

The rest lines describe the wrenches in the same format.

Output

The first line of the output file must contain the number of wrenches that can be used to turn off the bolt. The following lines must contain wrench numbers in increasing order.

Example

gtaw.in	gtaw.out
3 10 4 9 0 9 90 9 180 9 270 4 8 45 8 135 8 225 8 315 4 6 45 6 135 6 225 6 315 3 7 0 7 90 6 225	2 1 3

Problem H. Homo or Hetero?

Input file:	`homo.in`
Output file:	`homo.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Consider a list of numbers with two operations:

- `insert number` — adds the specified number to the end of the list.
- `delete number` — removes the first occurrence of the specified number from the list. If the list does not contain the number specified, no changes are performed.

For example: the result of the insertion of a number 4 to the list $[1, 2, 1]$ is the list $[1, 2, 1, 4]$. If we delete the number 1 from this list, we get the list $[2, 1, 4]$, but if we delete the number 3 from the list $[1, 2, 1, 4]$, the list stays unchanged.

The list is *homogeneous* if it contains at least two equal numbers and the list is *heterogeneous* if it contains at least two different numbers. For example: the list $[2, 2]$ is homogeneous, the list $[2, 1, 4]$ is heterogeneous, the list $[1, 2, 1, 4]$ is both, and the empty list is neither homogeneous nor heterogeneous.

Write a program that handles a number of the operations `insert` and `delete` on the empty list and determines list's homogeneity and heterogeneity after each operation.

Input

The first line of the input file contains an integer number n — the number of operations to handle ($1 \le n \le 100\,000$).

Following n lines contain one operation description each. The operation description consists of a word "`insert`" or "`delete`", followed by an integer number k — the operation argument ($-10^9 \le k \le 10^9$).

Output

For each operation output a line, containing a single word, describing the state of the list after the operation:

- "`both`" — if the list is both homogeneous and heterogeneous.
- "`homo`" — if the list is homogeneous, but not heterogeneous.
- "`hetero`" — if the list is heterogeneous, but not homogeneous.
- "`neither`" — if the list is neither homogeneous nor heterogeneous.

Example

homo.in	homo.out
11 insert 1 insert 2 insert 1 insert 4 delete 1 delete 3 delete 2 delete 1 insert 4 delete 4 delete 4	neither hetero both both hetero hetero hetero neither homo neither neither

Problem I. Image Recognition

Input file:	`image.in`
Output file:	`image.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Irene works for *Novel Efforts in Effective Recognition of Characters* (*NEERC*). Her new project concerns image recognition using robots.

Since the approach is quite innovative, Irene starts with a very simple model first. She fixed d images which are called digits 0 to $d-1$. Each image is a $w \times h$ rectangle filled with white and black unit squares (call them pixels). All images are distinct (that is, each two images differ in at least one pixel).

The robot is placed in the upper left pixel of one of the images. It starts executing a program written in a specific programming language described below. The task of the robot is to recognize which of the d images it was placed onto.

The programming language for the robot consists of the following commands:

‘`U`’, ‘`D`’, ‘`L`’, ‘`R`’ — movement commands. The robot moves one pixel up, down, left, or right respectively. If a movement command moves robot outside the image, the task is failed.

‘`(`’ $\langle subprogram_w \rangle$ ‘`:`’ $\langle subprogram_b \rangle$ ‘`)`’ — conditional operator. The robot checks the color of the pixel underneath itself. If it is white then $\langle subprogram_w \rangle$ is executed, otherwise $\langle subprogram_b \rangle$ is executed.

‘`0`’, ‘`1`’, ..., ‘`9`’ — recognized image commands. The robot must execute one of these commands when it knows which image it was placed onto. After such command, the program terminates.

Each movement command takes one time unit to execute. The execution of conditional operator and image recognized commands is instantaneous.

Irene is interested in the program that always works correctly. That is, if a robot is placed onto the image corresponding to the digit i, then the execution of the program must end with the command ‘`i`’.

Given the set of images, design a correct program for the robot, such that its execution time in the worst case is minimal.

Input

The first line contains three integers d, h, and w ($1 \le d \le 10$; $1 \le h, w \le 10$) — the number of considered images, the height and the width of each image.

The rest if the input file contains d descriptions of images. Each description consists of h lines of length w. All characters are either ‘`B`’ or ‘`W`’, representing a black or a white pixel respectively.

Image descriptions are given in the order from 0 to $d-1$. Descriptions are separated by an empty line.

Output

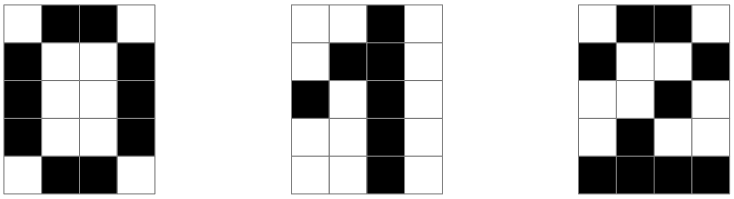
Return a correct program for the robot with minimal possible worst-case execution time. If there are multiple possible programs, output any of them.

All whitespace is ignored when parsing a program.

Example

image.in	image.out

`3 5 4`
`WBBW`
`BWWB`
`BWWB`
`BWWB`
`WBBW`
 	
`WWBW`
`WBBW`
`BWBW`
`WWBW`
`WWBW`
 	
`WBBW`
`BWWB`
`WWBW`
`WBWW`
`BBBB`
	`D(1:D(2:0))`



The robot has to distinguish between these three images in the example.

Problem J. Jealous Numbers

Input file:	`jealous.in`
Output file:	`jealous.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

There is a trouble in Numberland, prime number p is jealous of another prime number q. She thinks that there are more integer numbers between a and b, inclusively, that are divisible by greater power of q than that of p. Help p to get rid of her feelings.

Let $\alpha(n, x)$ be maximal k such that n is divisible by x^k. Let us say that a number n is p-dominating over q if $\alpha(n, p) > \alpha(n, q)$. Find out for how many numbers between a and b, inclusive are p-dominating over q.

Input

The first line of the input file contains a, b, p and q ($1 \le a \le b \le 10^{18}$; $2 \le p, q \le 10^9$; $p \ne q$; p and q are prime).

Output

Output one number — how many numbers n between a and b, inclusive, are p-dominating over q.

Example

jealous.in	jealous.out
`1 20 3 2`	`4`

In the given example 3, 9, 15 and 18 are 3-dominating over 2.

Problem K. Kripke Model

Input file:	`kripke.in`
Output file:	`kripke.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Testing and quality assurance are very time-consuming stages of software development process. Different techniques are used to reduce cost and time consumed by these stages. One of such techniques is software verification. *Model checking* is an approach to the software verification based on *Kripke models*.

A *Kripke model* is a 5-tuple (P, S, S_0, R, L), where P is a finite set of atomic propositions, S is a finite set of model's states, $S_0 \subset S$ is a set of initial states, $R \subset S \times S$ is a transition relation, and $L \subset S \times P$ is a truth relation. In this problem we will not take initial states into account and relation R will be a reflexive relation, so $R(s, s)$ will be true for all states $s \in S$.

A *path* π beginning in state s in the Kripke model is an infinite sequence of states $s_0 s_1 \ldots$ such that $s_0 = s$, and for each $i \geq 0$ the $(s_i, s_{i+1}) \in R$.

Temporal logic and its subset *Computational tree logic* (CTL) are used to describe propositions qualified in terms of time. Kripke models are often used to check properties, described in CTL.

There are two types of formulae in CTL: *state formulae* and *path formulae*. The values of state and path formulae are evaluated for states and paths correspondingly.

If $p \in P$ then p is a state formula that holds in state s iff $(s, p) \in L$.

If f is a path formula, then $\mathbf{A}f$ and $\mathbf{E}f$ are state formulae, where $\mathbf{A}$ and $\mathbf{E}$ are *path quantifiers*:

- $\mathbf{A}f$ holds in a state s, iff f holds for each path beginning in the state s;
- $\mathbf{E}f$ holds in state s, iff there exists a path π, beginning in the state s, such that f holds for π.

If f and g are state formulae, then $\mathbf{G}f$ and $f\mathbf{U}g$ are path formulae, where $\mathbf{G}$ and $\mathbf{U}$ are *temporal operators*:

- $\mathbf{G}f$ (Globally) holds for a path $\pi = s_0 s_1 \ldots$ iff for each $i \geq 0$ the formula f holds in the state s_i;
- $f\mathbf{U}g$ (Until) holds for a path $\pi = s_0 s_1 \ldots$ if there exists $i \geq 0$ such that f holds for each state in the range $s_0, s_1, \ldots, s_{i-1}$, and g holds in state s_i;

To verify a property described by a state formula f means to find all states, f holds for. Verification of an arbitrary property is a pretty complex problem. Your problem is much easier — you are to write a program that verifies a property described by a temporal logic formula $\mathbf{E}(x\mathbf{U}(\mathbf{AG}y))$, where x and y are some atomic propositions.

Input

The first line of the input file contains three positive integer numbers n, m and k — number of states, transitions and atomic propositions ($1 \leq n \leq 10\,000$; $0 \leq m \leq 100\,000$; $1 \leq k \leq 26$).

The following n lines describe one state each. The state i ($1 \leq i \leq n$) is described by c_i — a number of atomic propositions which are true for this state and a space-separated list of these atomic propositions ($0 \leq c_i \leq k$). Atomic propositions are denoted by first k small English letters.

Next m lines describe transitions. Each of them contains two integer numbers s and t ($1 \leq s, t \leq n$; $s \neq t$) — the transition from state s to state t. The verified Kripke model contains implicit loop transitions (s, s) for each state s (they are not listed in the input file). No transition is listed in the input file twice.

The last line of the input file contains the formula of the property to be verified. This formula always has the form "`E(xU(AGy))`", where '`x`' and '`y`' are some atomic propositions.

Output

The first line of the output file must contain the number of states for which the verified property holds. The following lines must contain the numbers of these states listed in increasing order.

Example

kripke.in	kripke.out
7 8 2 1 a 1 a 2 a b 1 b 1 b 1 a 1 a 1 2 2 3 3 4 4 5 5 3 2 6 6 7 7 6 E(aU(AGb))	5 1 2 3 4 5