Problem A. Antiplagiarism

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Prof. Acmeroff teaches programming in the Institute of Business Management. He teaches a relatively new, but popular language called Db. Unfortunately, program plagiarism is a serious headache for the professor, because some students cheat when doing home exersises. So the professor has to check students' works for plagiatrism each time. Thus a computer program, which is able to perform some sort of program comparison might be very useful for him.

The Db language is briefly described as follows:

- The `#` character never appears in programs. The character `\` never appears in programs outside of strings.
- All characters with codes less or equal then 32 are whitespace characters. A sequence of whitespace characters is called whitespace and equivalent to one space character (code 32).
- Comments starting from `/*` run till the first sequence of `*/` characters.
- A comment is equivalent to one space character.
- String literals start from `"` and run till the next `"`, except `\"`.
- An identifier is the longest possible sequence of characters starting from latin letter or underscore and containing latin letters, underscore and digits. For example, sequence `(0x800+)` contains one identifier `x800`. Letter case is significant.
- Some indentifiers are reserved as keywords. The exact list of keywords depends on the language dialect.
- There are no lexical scoping rules.

A program is processed as follows:

1. Each comment is replaced with a single space character.
2. Each whitespace (including space characters from the previous step) is replaced with a signle space character.
3. Each string is replaced with `""`.
4. Space characters are removed, except those separating identifiers and keywords from each other.

Finally, if there exists an one-to-one correspondence between the identifiers of the first program and the indentifiers of the second program, so the replacement of the identifiers in the second program with the corresponding identifiers of the first program yelds two equal texts, such programs are called *identical*.

Write a program, which checks whether two given programs written in Db are identical.

Input

The input for your program consists of the list of keywords and two programs written in Db. The list of keywords and the programs are separated from each other with a single `#` character. The list of keywords section only contains keywords and whitespace. The size of input does not exceed 4MiB. The total number of different identifiers and keywords does not exceed 65535.

Output

Print the word `YES`, if the input programs are identical, and the word `NO` otherwise.

Example

standard input	standard output

`int      void`
`   while if return`
`#`
`/* sizeof(int) == 4 */`
`int main(void) {`
`  int max = 0x80000000, n;`
`  while(scanf("%d", &n) == 1)`
`    if (n > max) max = n;`
`  printf("%d", max);`
`  return 0;`
`}`
`#`
`/* sizeof(int) == 4 */`
`int n(void) {`
`  int max = 0x80000000, main;`
`  while(printf("%d", &main) == 1)`
`    if(main > max)`
` max = main;`
`  scanf("Answer: \"%d\"", max);`
`  return 0;`
`}`
	`YES`

Problem B. Block Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

The rotation game uses a #-shaped board, which can hold 24 pieces of square blocks (see Fig.1). The blocks are marked with symbols 1, 2 and 3, with exactly 8 pieces of each kind.

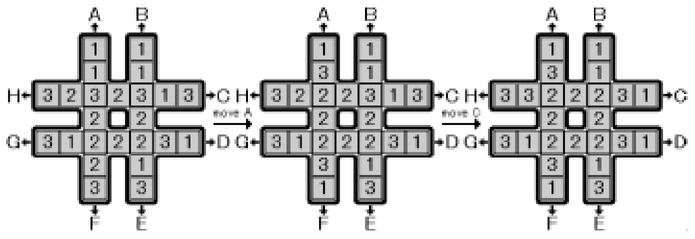

Fig 1.

Initially, the blocks are placed on the board randomly. Your task is to move the blocks so that the eight blocks placed in the center square have the same symbol marked. There is only one type of valid move, which is to rotate one of the four lines, each consisting of seven blocks. That is, six blocks in the line are moved towards the head by one block and the head block is moved to the end of the line. The eight possible moves are marked with capital letters '`A`' to '`H`'. Figure 1 illustrates two consecutive moves, move A and move C from some initial configuration.

Input

The input consists of several test cases. Each test case has only one line that contains 24 numbers, which are the symbols of the blocks in the initial configuration. The rows of blocks are listed from top to bottom. For each row the blocks are listed from left to right. The numbers are separated by spaces. For example, the first test case in the sample input corresponds to the initial configuration on picture above. There are no blank lines between cases. There is a line containing a single 0 after the last test case that ends the input.

Output

For each test case, you must output two lines. The first line contains all the moves needed to reach the final configuration. Each move is a letter, ranging from '`A` to '`H`', and there should not be any spaces between the letters in the line. If no moves are needed, output "`No moves needed`" instead. In the second line, you must output the symbol of the blocks in the center square after these moves. If there are several possible solutions, you must output the one that uses the least number of moves. If there is still more than one possible solution, you must output the solution that is smallest in dictionary order for the letters of the moves. There is no need to output blank lines between cases.

Example

standard input

1 1 1 1 3 2 3 2 3 1 3 2 2 3 1 2 2 2 3 1 2 1 3 3
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
0

standard output
AC 2 DDHH 2

Problem C. Conference

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

A conference on the road safety is held in Bytesburg. A long straight one-lane road goes from the airport to the city. The participants of the conference come to the airport one after another, get to a car waiting for them, and drive to the city. All participants are in a hurry, so they do not follow any speed limits — every car drives along the road with a constant acceleration (the acceleration can be different for different cars). If the faster car approaches the slower one, they collide and crash.

The local police doesn't really like crashes. Therefore, they put several control stations along the road. When the car gets to the station, it can be stopped (the decision to stop the car is made by the officer at the station). The car stops immediately, and then continues to move with its acceleration without waiting, but with zero initial speed. If two cars get to the control point at the same moment they crash anyway.

Now the police wonders whether it can stop the cars at the control stations in such a way that no crashes take place. If it is possible they want to do it in such a way that the last car gets to the city as soon as possible. Help them.

Input

The first line of the input contains two integer numbers: M, N, and a real number L — the number of cars, the number of stations the police put, and the distance from the airport to the city, in meters ($1 \leq M \leq 1000$, $0 \leq N \leq 9$, $0 < L \leq 100\,000$).

The following M lines contain two real numbers each: T_i and A_i — the time when the i-th car leaves the airpot (in seconds) and its acceleration (in m/s^2) ($0 \leq T_i \leq 10\,000$, $0 < A_i \leq 100$, $T_{i+1} > T_i$).

Each of the following N lines contains a real number L_j — the distance from the airport to the corresponding control station.

All real numbers have . (dot) as decimal separator.

Output

If the police cannot prevent all crashes, output one line containing -1.

In the other case the first line of the output must contain the minimal time from the departure of the first car from the airport till the arrival of the last car to the city. The following M lines must describe where the cars must be stopped to achieve this. Each line must contain the number of stops the corresponding car must have, and the numbers of control stations it must stop at.

Example

standard input	standard output

2
2
1000
0 10
4 20
80
900
	16.420090
	0
	1 1

2
2
1000
0 10
1 20
100
900
	-1

Problem D. Decorative Floor

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Frederik decided to decorate his new room with a cool pattern on the floor, composed with colorful parquet bricks. After several days of work, he finally felt satisfied with the pattern he created with the odd shaped bricks. Soon, he found a problem. As the border of the pattern is not a perfect rectangle, the floor is not filled with bricks completely. However, since the shapes of the bricks are quite odd, it is not an easy work to fill the floor with these bricks completely. (Although a brick with a unit size is provided, this kind of brick is quite expensive usually.

Picture below shows an example of a set of bricks) As Frederik was very proud of his brick pattern, he refuses to modify even one brick in his pattern in order to satisfy the need of filling the floor completely. Instead, he asked you to find solutions for him to fill the rest part of the floor completely with the given bricks so that the sum needed to buy these bricks are minimized. Of course, the bricks can not overlap each other to fill the floor. Please note that the bricks can be rotated, but cannot be flipped over. In addition, you may assume that all kinds of bricks can be contained within a 3 × 3 square box.

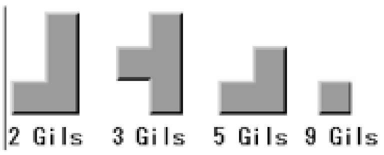

Fig 1.

As the pattern of bricks covers most areas of the floor, the uncovered part of the floor is actually located at the bottom border of the rectangular shaped floor. Therefore, an uncovered part like the one in figure below can be described with a series of integers, which represents the number of square blocks missed in each column, starting from the left.

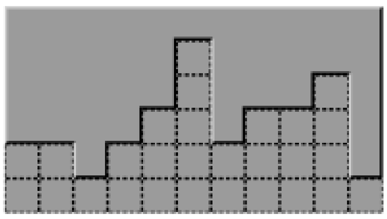

Fig 2.

This shape can be described with 11 integers: 22123523341. In addition, you may assume that these integers are no larger than 5. shows a solution to fill the floor with the brick set in Fig.1 that costs minimal sum.

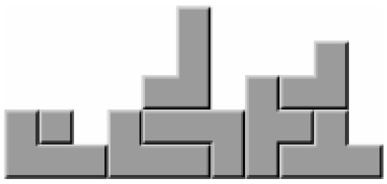

Fig 3.

Input

The input consists of several test cases. The first line of each test case contains an integer n ($1 \leq n \leq 1000$), which is the width of the floor. The second line contains n integers, separated by single spaces. These integers describe the shape of the uncovered floor as mentioned above. The third line contains an integer m ($1 \leq m \leq 100$), which is the number of bricks available. After this is the detailed description of the m bricks.

Each brick description consists of 4 lines. The first line is a positive integer, which is the price of that brick. After this line, there are three lines, each consisting of three characters, which describe the shape of the brick. A dot character represents a space in the shape and a sharp character '#' represents a square block in the shape. You can be sure that the blocks are always connected to form the brick. There is a line containing a zero after the last test case, which signifies the end of the input and should not be processed.

Output

For each test case, print the minimal sum needed to fill the floor with the given bricks. If the floor cannot be filled with the given bricks, output -1 instead.

Example

standard input	standard output
`11`	`28`
`1 4 3 3 2 5 3 2 1 2 2`	`1`
`4`	`-1`
`2`	
`#..`	
`#..`	
`##.`	
`3`	
`.#.`	
`.##`	
`.#.`	
`5`	
`...`	
`#..`	
`##.`	
`9`	
`...`	
`.#.`	
`...`	
`1`	
`1`	
`1`	
`1`	
`..#`	
`...`	
`...`	
`1`	
`1`	
`1`	
`1`	
`.##`	
`...`	
`...`	
`0`	

Problem E. Equalities And Matches

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

In this task we will work with equalities on Roman numbers build from matches.

Roman numbers are written as follows. There are four digits: `I` (letter i), `V`, `X`, and `L`. `I` denotes one, `V` denotes five, `X` denotes ten, and `L` denotes fifty. A natural number is represented by a sequence of digits as follows. No digit can occur more than three times in a row. A number larger than ten are written by giving its tens first, followed by the remaining units. If a larger digit comes before a smaller one they are summed up (summation principle); and if a smaller one comes before a larger then the smaller digit is subtracted from the larger (subtraction principle). The sequence of digits does not contain digits A and B preceding some third digit C, such as $A < C$ and $B < C$.

We will not use large numbers in his trick, so the character `L` cannot occur more than once.

For example, the number `4` is written as Roman `IV`, but not `IIII`. The number `30` is written as Roman `XXX`, but not `XXL`. The number `45` is written as Roman `XLV`, but not `VL`. Finally, the number `100` is too large for Bavid, as it cannot be represented according to the rules specified above. Obviously, any number has no more than one Roman representation.

An *equality* is defined by the following grammar:

```
<equality>  ::= <statement> = <statement>
<statement> ::= <number>
<statement> ::= <number> + <statement>
<statement> ::= <number> - <statement>
```

Where `<number>` is a valid Roman number according to the rules above.

An equality is called *valid* if the number in its left-hand side equals to the number in its right-hand side.

Seven characters are used in equalities: `I`, `V`, `X`, `L`, `+`, `-`, `=`. These characters are constructed from matches as follows: for the characters `I` and `-` one match is needed, for the characters `V`, `X`, `L`, `+`, and `=` two matches are needed.

For a match to be moved it must be first taken from its character and then laid to a new position.

A match is either laid over an existing character (including the one from which the match was taken), or laid between two consequent characters, or before the first character, or after the last one thus forming a new character.

If a match is taken from a one-match character, this character vanishes and its left and right neighbors become adjacent.

If a match is taken from a two-match character, the remaining matches must still represent some valid character. Here is the list of all possible extractions:

- `L` $\rightarrow$ `I` (the horizontal match is taken)
- `+` $\rightarrow$ `I` (the horizontal match is taken)
- `+` $\rightarrow$ `-` (the vertical match is taken)
- `=` $\rightarrow$ `-` (any horizontal match is taken)

There are two ways to lay a match between two consequent characters (and also before the first and after the last characters): one may either put `I` or `-`.

Insertion of a match into a character is permitted if the character into which the match is being inserted remains valid. Here are all possible insertions of this kind:

- `I` → `L` (the horizontal match is laid)
- `I` → `+` (the horizontal match is laid)
- `-` → `=` (the horizontal match is laid)
- `-` → `+` (the vertical match is laid)

Also, it is possible to lay a match over the character from which it has been taken:

- `I` ↔ `-`
- `L` ↔ `+`
- `=` ↔ `+`
- `V` ↔ `X`

Write a program that finds all valid equalities that can be obtained from the input string by moving exactly one match. It is guaranteed that at least one valid equality exists.

Input

The first and the only line of input contains a string consisting of the characters `I`, `V`, `X`, `L`, `+`, `-`, and `=`. The length of the string does not exceed 100. It is guaranteed that the input string is not a valid equality as defined above.

Output

Output all different valid equalities that can be obtained from the input string by moving exactly one match. It is guaranteed that at least one valid equality exists.

Example

standard input	standard output
`I=I-II`	`I=II-I`

`III=-II`	`III=III`
	`I+I=II`

`V-XX=I-XI`	`V-XV=I-XI`
	`X-XX=I-XI`

`II-I=I=I`	`II-I=II-I`
	`II-II=I-I`
	`III-I-I=I`

Problem F. Fantasy Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 Mebibytes

Fiona invented a splendid turn-based strategy game with fantasy theme. There are elves, orcs, undead, dwarves, and even drakes roaring around! As for this problem, you must know that every unit has some number of hitpoints, which is reduced during combats, and you must also understand how the combat proceeds. We will be interested only in the simplest case – a common attack at a defenseless unit.

Such an attack is described by three integers:

- d – the damage done by a successful blow;
- b – the number of blows;
- p – the probability (in percentage) that a blow is successful.

In the game, d and b are properties of the attacker, whereas p is usually given by the terrain occupied by the defender, but for our purposes, we assume that it is a property of the attacker, too (as with magical attacks).

As an example, consider an attack with $d = 6$, $b = 2$, and $p = 60$. There are three possible outcomes:

- By 16 %, the attacker misses both times and does no damage at all.
- By 48 %, the attacker hits with one of the blows and misses with the other one, doing the damage of 6 hitpoints.
- By 36 %, the attacker hits both times, doing the total damage of 12 hitpoints.

When the number of hitpoints of the defending unit becomes non-positive, the unit dies.

Fred is currently playing an add-on where a special unit, the Elvish Princess, has a very special magical attack: Instead of d and b, she is described by an integer m, and when she attacks, the player may choose d and b to be arbitrary positive integers as long as $d \cdot b \leq m$.

Every now and then, Fred needs his Elvish Princess to kill an ugly undead skeleton or a stinking orcish warrior, so he would like to know which choice of d and b gives the highest probability of killing the enemy.

For every unit Fred needs to kill, find the best choice of d and b.

Input

The first line of the input contains an integer m ($1 \leq m \leq 10^6$) and an integer p ($1 \leq p \leq 99$) – the parameters of the attacker. The second line contains an integer n ($1 \leq n \leq 10^5$) – the number of units to be killed. The third line contains n integers $h_1, h_2, \ldots, h_n$ ($1 \leq h_i \leq 10^6$), where h_i is the number of hitpoints of the i-th unit to be killed.

Output

For each unit to be killed, output two integers: the d and b which give the maximum probability of killing the unit. Your answer will be accepted if the probability given by your output differs from the optimum by no more than 10^{-6}.

Example

standard input	standard output

10 60
3
5 6 7
	5 2
	2 5
	7 1

Note

The corresponding probabilities are $84\,\%$, $68.256\,\%$, and $60\,\%$.

Problem G. Galactic Wormholes

Input file:	`standard input`
Output file:	`standard output`
Time limit:	15 seconds
Memory limit:	512 mebibytes

The intergalactic company is going to build a network of wormholesthat will connect the planets of the galaxy far far away. They have already prepared a construction plan, i.e., a sequential order of building the wormholes. Each wormhole will be a bidirectional corridor that will connect two (not necessarily distinct) planets.

A wormhole is *safe* if it is not the only wormhole that connects (directly or indirectly) some pair of planets. In other words, a wormhole H is *unsafe* if there are two planets A and B such that when traveling from A to B we have to use H. (Alternately, note that safe wormholes are the ones that lie on some cycle.)

You will be given the order in which wormholes will be constructed. After each construction there may be some wormholes which have just become safe. (Including, possibly but not necessarily, the wormhole that was just built.) Count those wormholes.

Note that if a wormhole is already safe, it will remain safe for the rest of the construction.

Input

On the first line of input are two integers n and m: n is the number of planets in the plan, m is the number of wormholes. The planets are numbered $1 \ldots n$.

Each of next m lines consists of two space separated integers – the ids of planets next wormhole will join. There can be a wormhole connecting a planet with itself. There can be a multiple wormholes between two planets.

$n \leq 10^6$ and $m \leq 2 \cdot 10^6$.

Output

For each wormhole in the input, output a single line with a single integer: the number of wormholes that just became safe.

Example

standard input	standard output

5 8
2 1
3 3
4 5
2 3
4 5
3 4
4 1
2 5
	0
	1
	0
	0
	2
	0
	4
	1

Problem H. Hack the Lock

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

A new code lock was recently invented, named a Bit Lock.

The lock holds a sequence of bits (displayed as zeroes and ones). The lock has four buttons conveniently located on its sides. The buttons are named `L0`, `R0`, `L1`, and `R1`.

When a user presses `L0` button the leftmost bit of the sequence is erased, hence shifting the sequence one position to the left. Next, a 0-bit is appended to the right of the sequence.

Button `L1` acts similarly except for it adds 1-bit to the right after erasing the leftmost bit.

Buttons `R0` and `R1` act in a similar way but shift the sequence in the opposite direction. Namely, button `R0` erases the rightmost bit shifting the sequence to the right and then appends 0-bit to the left. Button `R1` appends 1-bit instead of 0-bit.

Given the initial and the final configurations, your task is to transform the former to the latter with the fewest possible operations.

Input

The first line of the input contains the initial sequence currently displayed on the screen. The second line contains the desired final sequence. Both sequences are nonempty, have the same length not exceeding 10^5, and consist of zeros and ones (without spaces).

Output

Output the minimum number of button presses.

Examples

standard input	standard output
0111 0110	2
0110 1111	3

Problem I. Islands

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

Given a map of islands and bridges that connect these islands, a Hamilton path, as we all know, is a path along the bridges such that it visits each island exactly once. On our map, there is also a positive integer value associated with each island. We call a Hamilton path the best triangular Hamilton path if it maximizes the value described below.

Suppose there are n islands. The value of a Hamilton path $C_1, C_2, \ldots, C_n$ is calculated as the sum of three parts. Let V_i be the value for the island C_i . As the first part, we sum over all the V_i values for each island in the path. For the second part, for each edge $C_i - C_{i+}1$ in the path, we add the product $V_i \cdot V_{i+1}$. And for the third part, whenever three consecutive islands $C_i - C_{i+1} - C_{i+2}$ in the path forms a triangle in the map, i.e. there is a bridge between C_i and C_{i+2} , we add the product $V_i \cdot V_{i+1} \cdot V_{i+2}$.

Most likely but not necessarily, the best triangular Hamilton path you are going to find contains many triangles. It is quite possible that there might be more than one best triangular Hamilton paths; your second task is to find the number of such paths.

Input

The input starts with a number q on the first line, which is the number of test cases ($1 \le q \le 20$). Each test case starts with a line with two integers n and m, which are the number of islands and the number of bridges in the map, respectively. The next line contains n positive integers, the i-th number being the V_i value of island i. Each value is no more than 100. The following m lines are in the form x y, which indicates there is a (two way) bridge between island x and island y. Islands are numbered from 1 to n. You may assume there will be no more than 13 islands. Each pair of islands is listed at most once.

Output

For each test case, output a line with two numbers, separated by a space. The first number is the maximum value of a best triangular Hamilton path; the second number should be the number of different best triangular Hamilton paths. If the test case does not contain a Hamilton path, the output must be 0 0.

Example

standard input	standard output

2
3 3
2 2 2
1 2
2 3
3 1
4 6
1 2 3 4
1 2
1 3
1 4
2 3
2 4
3 4
	22 3
	69 1

Note

A path may be written down in the reversed order. We still think it is the same path.

Problem J. Joy Of The Speed

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

In a small town far, far away speed limit fine was raised once again. Drivers had been so distracted with such a strict policy that have finally decided to move out of town to a better place. They chose a landing strip in an abandoned airport to organize a drag race competition between their legendary cars. The goal was simple: get your car from the start to the finish as fast as you can. All cars start with zero velocity and no speed limit is set on the landing strip.

Our hero has a special car he made on his own. His car doesn't have a limit on a crankshaft rotation speed as all other cars do. Moreover, his car was so finely tuned and so precisely weighted that he actually had an exact formula to his car acceleration rate: $a = a_0/(p - q \cdot w + w^3)$. Here a_0, p, and q are the secret parameters of his engine; $w = c_k \cdot v/v_0$, where each of c_k is the gear reduction rate for the respective transmission k $(1 \le k \le 5)$, v is a current velocity, and v_0 is the fourth secret parameter. Needless to say that his car was capable to start with any transmission and then switch transmissions instantly and without any restrictions.

Given that this car took so many man-hours to build and tune our hero didn't have time to actually take part in the competition when it was held, but he now possesses the final scoreboard from a friend of his. He is eager to see where his own results could fit on this list.

Assuming that the transmission is switched optimally, calculate the minimum amount of time T to drive from the start to the finish on the hero's car. Also calculate the velocity of the car at the finish line during that race.

Input

The input contains 10 positive real numbers delimited by spaces: L (the length of an airline strip in meters), a_0 (the first secret engine parameter, in meters/sq. second), p and q (the second and the third secret engine parameters), v_0 (the forth secret engine parameter, in meters/second), c_1, c_2, c_3, c_4 and c_5 (the secret gear reduction rate parameters).

You may assume that all numbers are less than 10^9 and the following conditions are met: $4q^3 < 27p^2$, $c_1 > c_2 > c_3 > c_4 > c_5 > 0$.

All input numbers are not greater than 1000 and not less than 0. All numbers are given with up to 6 digits after decimal point.

Output

Please output the values of T and V with 3 decimal digits after the point.

Example

standard input	standard output
`1000 10 2 1 100 5 4 3 2 1`	`18.286 106.679`
`1000 1 1 1 10 16 8 4 2 1`	`59.441 22.386`
`0.1 1 1 1 1 5 4 3 2 1`	`0.369 0.561`

Problem K. Kingdom of Onliners

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

In Kingdom of Onliners N popular bloggers are publishing the messages.

Messages of other bloggers are reposted in the following way: given list of **directed** pairs of blogers (u, v) such as bloger v reposts anything bloger u posted (but reverse may not be true).

You are responsible for the PR-campaign of some brand new smartphone in Kingdom of Onliners, and came with brilliant plan to find the blogger, whose messages will be reposted by all bloggers, and persuade her to post something good about your product. Lets call such a blogger "influencer".

So, given all the information about directions of reposting, before choosing an influencer you are decided to make full list of them.

Input

The first line of the input contains two integers N a M: the number of bloggers and the number of reposting directions.

Then, M more lines follow. Each contains two integers A_i and B_i and denotes that bloger B_i is reposting messages of A_i.

It holds $1 \le N, M \le 10^6$.
For all i it holds $1 \le A_i, B_i \le N$ and $A_i \ne B_i$.

No two directed pair repeats (but there may be pair of bloggers, reposting each other).

Output

Output two lines. The first line should contain the total number of influencers. The second line should contain a space-separated list of these infiuencers **in ascending order**.

Example

standard input	standard output

5 6
2 3
2 5
3 1
3 4
4 2
5 4
	4
	2 3 4 5

3 2
2 1
3 1
	0