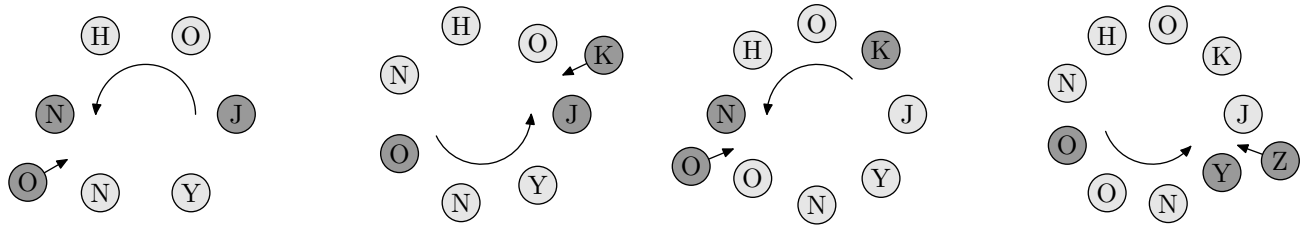
Problem A. Alphabet

Input file:	`alphabet.in`
Output file:	`alphabet.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Johnny is a little boy, and now he is learning alphabet. His father made him a birthday present — a large collection of tokens, each marked with a letter from '`A`' to '`Z`', and Johnny invented an interesting game to help himself learning.

First, Johnny takes several tokens and puts them in a circle. After that he selects one of the tokens to start the game at, and chooses some number k. Each turn Johnny counts k tokens along the circle to find the next winning token and inserts another token to the circle immediately after it. This new token must hold the letter which follows alphabetically the letter written on the winning token, i.e. '`B`' is inserted if the winning token is '`A`', '`C`' is inserted after '`B`', etc. If the winning token is '`Z`', the token '`A`' is inserted. There are so many tokens available that Johnny can always find one with the required letter.

After inserting the token Johnny proceeds to another game turn, this time starting from the newly inserted token. The number k is only selected once and is not changed between turns. The first four turns of a sample game with initial tokens '`J`', '`O`', '`H`', '`N`', '`N`', '`Y`' ('`J`' is the starting one) and $k = 3$ are shown on a figure below:



Georgie is Johnny's elder brother. Since he is already a schoolboy, he plays the same game mentally and astonishes Johnny with his ability to predict the letter inserted at turn i. But as Johnny gets smarter and makes more and more turns, it becomes harder for Georgie to compete, so he decides to write a computer program to find the results quickly.

Input

The first line of the input file contains three integers: n — the number of tokens initially placed in a circle ($1 \le n \le 10\,000$), k — the number of tokens counted each turn ($1 \le k \le 10\,000$), and m — the number of turns ($1 \le m \le 10^9$).

The second line contains a string of n uppercase letters ('`A`' to '`Z`') — the tokens initially placed in a circle, starting from the initial token.

Output

Output a single uppercase English letter which is added to the circle at the m-th turn.

Example

alphabet.in	alphabet.out
6 3 4 JOHNNY	Z

Problem B. Bridges

Input file:	`bridges.in`
Output file:	`bridges.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Once upon a time there was a country in a delta of a far-away river. The country had n islands and there was a town on each island. The towns were connected by roads. There was exactly one route from each town to each other one (possibly passing through some intermediate towns). Unfortunately, each road had to cross the river with a ford, since bridges were not known, so the travel was quite uncomfortable and could only be made by a horse.

When *Bridge Building* technology was discovered the king decided to build bridges instead of some fords to make roads easier to travel. Bridges would allow fords to be crossed even by carriages. The king liked the idea with bridges and ordered to build as many bridges as possible. Unfortunately, the country was quite poor, so only k bridges could be built.

The king asked you — his major advisor — to develop a bridge building plan. You have to choose k fords in such a way that the sum of travel times between all pairs of towns becomes as small as possible. You must assume that the ordinary roads would be traveled by horses, and roads enhanced with bridges would be traveled by carriages.

Input

The first line of the input file contains four integer numbers: n, k, s_h and s_c — the number of towns in the country, the number of bridges to build ($1 \le k < n \le 10\,000$), the speed of the horse and the speed of the carriage in meters per second ($1 \le s_h, s_c \le 100\,000$).

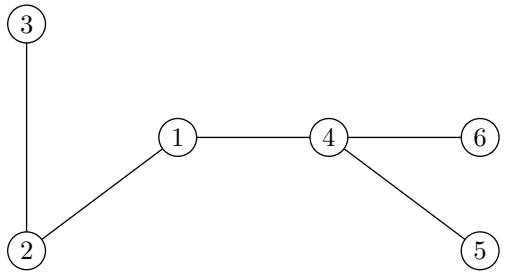
Each of the following $n - 1$ lines contains three integer numbers: b_i, e_i — the towns connected by the road, and l_i — the road length in meters ($1 \le l_i \le 10^6$). Towns are numbered from 1 to n, roads are numbered from 1 to $n - 1$.

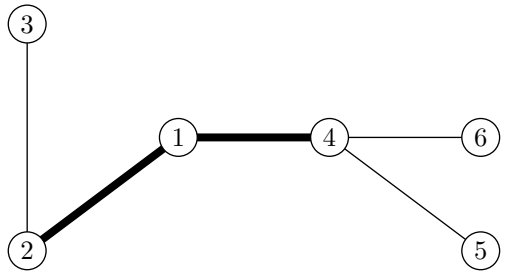
Output

Output k numbers — the numbers of roads where the bridges should be built. If there are several possible optimal bridge building plans, output any of them.

Example

bridges.in	bridges.out
6 2 1 2 1 2 5 3 2 6 1 4 4 4 6 4 4 5 5	1 3





Problem C. Confectionery

Input file:	`confectionery.in`
Output file:	`confectionery.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

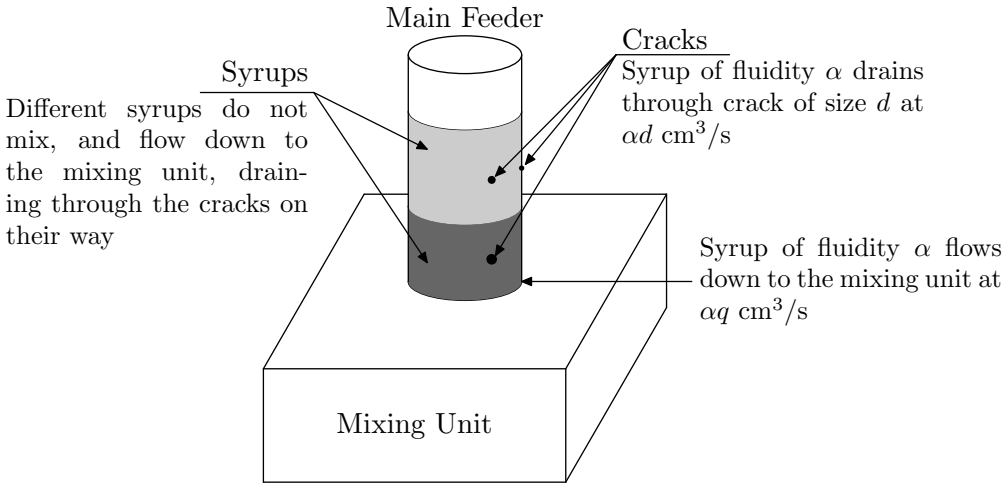
ACM Inc., the manufacturer of Automatic Confectionery Mixers, often receives complaints for its ACM-07 household devices. The weak place of ACM-07 is the main feeder — the vertical plastic tube which contains syrup before it is injected into the mixing unit. The main feeder gets cracked due to mixer vibrations, and the syrup drains through the cracks.

The last complaint was an unusual one. Margaret called ACM Inc. customer support service to express her thanks. She told that her cakes and pastry became incredibly tasty, and she was never able to have the same success without the mixer. The company manager was greatly surprised with this story and decided to send Alex — an ACM-07 constructor — to investigate the case.

Margaret is a daughter of a mathematician, so she always cooks according to the same formalized algorithm and uses the same ingredients. There are n various fruit syrups, which are poured into the feeder one after another, from 1 to n. Syrups have different densities and normally don't mix. When all syrups are in place, Margaret opens the valve and lets the liquid flow to the mixing unit.

Alex found that the main feeder of the mixer was broken in several places, and some amount of syrup drains through the cracks, therefore not getting to the mixing unit. He told Margaret about this, and now she believes that the taste is so special because the proportions of syrups has changed due to drains. Fortunately, the syrups don't mix while running through the feeder, and Alex volunteered to help Margaret to calculate the syrup proportions after passing the feeder tube.

Each syrup is characterized by its fluidity. If the syrup with fluidity α is located at the crack of size d in the feeder, each second αd cm^3 of syrup drains out of the crack. Similarly, if the syrup of fluidity α is at the bottom of the feeder, each second αq cm^3 of syrup flows through the hole at the bottom of the feeder into the mixing unit. Given the positions of the cracks on the feeder, their sizes and the size of the hole at the bottom of the feeder, as well as the initial amount of each syrup, find the amount of each syrup that eventually gets to the mixing unit.



Input

The first line of the input file contains n — the number of syrups ($1 \le n \le 10$), m — the number of cracks ($0 \le m \le 10$), and s — the area of the perpendicular section of the feeder in cm^2 ($1.0 \le s \le 100.0$).

The following n lines describe syrups. Each line contains two real numbers: v_i — the initial volume of the syrup in cm^3 and α_i — the fluidity of the syrup. The syrups are described in order from the one at

the bottom of the feeder to the one at the top ($1.0 \le v_i, \alpha_i \le 100.0$).

The following m lines describe cracks. Each crack is described with two real numbers: x_i and d_i — the distance from the bottom of the feeder to the crack, and its size, respectively ($1.0 \le x_i \le 1000.0$, $0.1 \le d_i \le 10.0$). You can consider the size of cracks be small relative to the size of the feeder, so you can treat them as points.

The last line of the input file contains a real number q ($0.1 \le q \le 100.0$).

Output

Print n real numbers — for each syrup print the volume of the syrup that eventually gets to the mixing unit. Print at least four digits after the decimal point.

Example

confectionery.in	confectionery.out
2 3 2.5 5.0 4.0 6.25 2.0 1.0 0.4 3.0 0.3 3.0 0.2 1.5	4.4736842105 4.8444491796

Problem D. Deciphering

Input file:	`decipher.in`
Output file:	`decipher.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Mr. Ford Trunkings, a well-known archaeologist, has recently discovered the ruins of a strange ancient settlement in the heart of Africa. After a few weeks of investigation, he and his colleagues have understood that they have found something great. The tribe *Velulu*, who lived there millenia ago, seemed to have very high level of development. They even had an alphabet! But most of them probably became victims of a glacial period about 20 000 years ago. So all their cultural achievements were completely lost. Only a few lucky remains of the tribe survived the glacial period, and after thousands of years they restarted their attempt to create a great civilization. They are now known as Zulu. But this is a story for another time...

Our current task will be to decipher *Velulu* texts. But the main difficulty is that there are no spaces in their texts at all. So all words of the text merge into one huge sequence of letters which is very hard to understand.

Fortunately, the archaeologists have already built a draft of *Velulu* language dictionary. Of course they know about recent achievements in computer science that allow one to parse a sequence of letters to a text consisting of words from a given dictionary. They have tried this technique, but after a few attempts they have discovered that there is a huge number of such sequences for almost every text of reasonable size. They don't know whether it is a problem with the method or some peculiarity of *Velulu* language. So they have invented another method which relies not only on dictionary, but also on order of parts of speech in a sentence.

Now they have not only the proposal for dictionary, but also the proposal describing how sentences can be constructed in *Velulu* language. Your task is to find out how many ways a given text can be parsed, according to this information, and provide an example of parsing the text.

Input

You will be given the dictionary, the sentence construction rules and the text. For each word you will know which part of speech it can stand for.

The first line of the input file contains three numbers: n, m and k, where $1 \le n \le 5\,000$ is the number of words in *Velulu* language, $1 \le m \le 10$ is the number of possible sentence construction rules, and $1 \le k \le 10$ is the number of different parts of speech.

Each of the following n lines contains one word and a list of possible parts of speech it can stand for. There are not too many letters in *Velulu* language, so archaeologists have decided to encode them with small English letters. In each line, the word (non-empty, shorter than 20 letters) is given, followed by a space, then number k_i ($1 \le k_i \le 10$) of parts of speech possible for this word, then k_i numbers a_{ij} each denoting a particular part of speech ($1 \le a_{ij} \le k$). All a_{ij} for any word are given in strictly increasing order. Words in the input file are given in arbitrary order (the dictionary is not perfect, and the exact order of letters is not yet known). Each word occurs exactly once.

The following m lines list the sentence construction rules. Each rule is described by a number of words l_i ($1 \le l_i \le 10$) in this specific type of sentence, followed by l_i identifiers of parts of speech b_{ij} ($1 \le b_{ij} \le k$). No rule appears twice.

The last line of the input file is for the text to be deciphered. The text is non-empty and consists of less than 1 000 letters.

Output

The first line of the output file must contain the number of possible ways to parse the text. The ways

are considered different even if the word separation is the same, but the rules used to compose differ for at least one sentence. If the number is more than 10^{18}, output a line "`TOO MANY`" instead.

If a correct parsing of the text exists, output an example of the parsing to the second line. Write the original text with spaces and full stops inserted at corresponding positions to get an acceptable sequence of sentences. Full stops are inserted immediately after the last word of each sentence, and must be followed by a space (see output example for further clarification). The whole text must be completely split to sentences.

If there is more than one acceptable way of parsing, output any one.

Example

decipher.in	decipher.out
5 2 2 ba 1 2 za 2 1 2 a 2 1 2 caba 1 1 ab 1 1 2 1 2 3 2 2 1 abazabacaba	2 ab a. za ba caba.
5 2 4 arrow 1 1 time 1 1 flies 2 1 2 like 2 2 4 an 1 3 5 1 2 4 3 1 5 1 1 2 3 1 timeflieslikeanarrow	2 time flies like an arrow.

Problem E. Equation

Input file:	`equation.in`
Output file:	`equation.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

A few years ago Nick learned set theory, and liked it so much that sets became the subject of his bachelor's thesis. Nick's thesis describes a new kind of mathematical equations — *set equations*. Set equations are composed of *set variables*, which represent various subsets of a universal set Ω. The following four operations are defined for set variables:

- Intersection $\cap$: $A \cap B := \{x : x \in A \wedge x \in B\}$.
- Union $\cup$: $A \cup B := \{x : x \in A \vee x \in B\}$.
- Difference $-$: $A - B := \{x : x \in A \wedge x \notin B\}$.
- Symmetrical difference $\triangle$: $A \triangle B := (A - B) \cup (B - A)$.

The operations are listed in the order of their precedence, i.e. intersection has the highest precedence, union has lower precedence than intersection, difference has lower precedence than union, and symmetrical difference has the lowest precedence. Parentheses can be used to change operations precedence as usual.

The thesis includes a chapter dedicated to automated solving of some specific classes of set equations. Nick has already elaborated a notation for storing equations in text files. Here is some quotation from the thesis:

> The following notation is used for describing set equations:
>
> $$\begin{aligned}
> \langle \text{space or tab} \rangle &\longrightarrow \textit{space} \mid \textit{tabulation symbol} \\
> \langle \text{ws} \rangle &\longrightarrow \langle \text{space or tab} \rangle * \\
> \langle \text{char} \rangle &\longrightarrow \text{`A'} \mid \text{`B'} \mid \ldots \mid \text{`Z'} \mid \text{`a'} \mid \text{`b'} \mid \ldots \mid \text{`z'} \\
> \langle \text{var} \rangle &\longrightarrow \langle \text{char} \rangle \langle \text{char} \rangle * \\
> \langle \text{expr} \rangle &\longrightarrow \langle \text{var} \rangle \\
> &\mid \langle \text{expr} \rangle \langle \text{ws} \rangle \langle \text{operator} \rangle \langle \text{ws} \rangle \langle \text{var} \rangle \\
> &\mid \text{`(' } \langle \text{ws} \rangle \langle \text{expr} \rangle \langle \text{ws} \rangle \text{ `)'} \\
> \langle \text{operator} \rangle &\longrightarrow \text{`+'} \mid \text{`*'} \mid \text{`-'} \mid \text{`\^{}'} \\
> \langle \text{equation} \rangle &\longrightarrow \langle \text{expr} \rangle \langle \text{ws} \rangle \text{`='} \langle \text{ws} \rangle \langle \text{expr} \rangle
> \end{aligned}$$
>
> Operators $\cup$, $\cap$, $-$ and $\triangle$ are encoded with tokens '`+`', '`*`', '`-`' and '`^`' respectively. The '`=`' token denotes the operator of set equality.
>
> It is possible to assign values to some variable using the following notation:
>
> $$\begin{aligned}
> \langle \text{digit} \rangle &\longrightarrow \text{`0'} \mid \text{`1'} \mid \ldots \mid \text{`9'} \\
> \langle \text{element} \rangle &\longrightarrow \langle \text{digit} \rangle * \\
> \langle \text{variable value} \rangle &\longrightarrow \langle \text{var} \rangle \langle \text{ws} \rangle \text{`='} \langle \text{ws} \rangle \langle \text{values} \rangle \\
> \langle \text{values} \rangle &\longrightarrow \langle \text{element} \rangle \\
> &\mid \langle \text{element} \rangle \langle \text{space or tab} \rangle \langle \text{ws} \rangle \langle \text{values} \rangle
> \end{aligned}$$

Unfortunately Nick is not very skilled in programming so he asks you to help and write a program module for automated solving of one specific simple class of equations — the equations where each variable, except Ω, occurs exactly one time.

Input

Input file is written in Nick's notation.

The first line of the input file contains an equation to solve. Total length of the equation does not exceed 1 000 characters. There is exactly one occurece of each variable, except "`Omega`", in the equation.

The following lines contain definitions of variable values. Variable name lengths do not exceed ten characters each. There is always a variable called "`Omega`" that represents the universal set. Both the number of elements in the universal set and the value of each element does not exceed 500.

The total length of all variable definitions does not exceed 100 000 characters.

Output

First line of the output file must contain "`Solution`" if there exists at least one solution of the given equation and "`No Solution`" otherwise.

If the solution exists the following lines must contain such values of all undefined variables so that equation is satisfied. The variables may be listed in arbitrary order.

Example

equation.in	equation.out
Omega - OneOrThree = Two Omega = 1 2 3 Two = 2	Solution OneOrThree = 1 3
Omega * result = Empty Omega = 1 2 3 4 5 Empty =	Solution result=
(one+two) ^(TWO+three) = result one =1 TWO =1 3 4 5 two =2 three=3 Omega=1 2 3 4 5 result = 2 3 4 5	Solution
Omega ^Omega = Omega Omega=123 234 345 456	No Solution

Problem F. Feng Shui

Input file:	`feng.in`
Output file:	`feng.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Feng shui is the ancient Chinese practice of placement and arrangement of space to achieve harmony with the environment. George has recently got interested in it, and now wants to apply it to his home and bring harmony to it.

There is a practice which says that bare floor is bad for living area since spiritual energy drains through it, so George purchased two similar round-shaped carpets (feng shui says that straight lines and sharp corners must be avoided). Unfortunately, he is unable to cover the floor entirely since the room has shape of a convex polygon. But he still wants to minimize the uncovered area by selecting the best placing for his carpets, and asks you to help.

You need to place two carpets in the room so that the total area covered by both carpets is maximal possible. The carpets may overlap, but they may not be cut or folded (including cutting or folding along the floor border) — feng shui tells to avoid straight lines.

Input

The first line of the input file contains two integer numbers n and r — the number of corners in George's room ($3 \le n \le 100$) and the radius of the carpets ($1 \le r \le 1000$, both carpets have the same radius). The following n lines contain two integers x_i and y_i each — coordinates of the i-th corner ($-1000 \le x_i, y_i \le 1000$). Coordinates of all corners are different, and adjacent walls of the room are not collinear. The corners are listed in clockwise order.

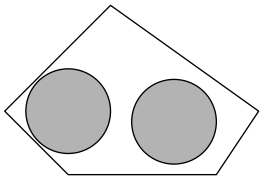
Output

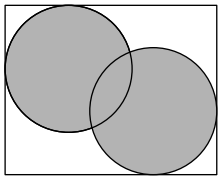
Write four numbers x_1, y_1, x_2, y_2 to the output file, where (x_1, y_1) and (x_2, y_2) denote the spots where carpet centers should be placed. Coordinates must be precise up to 4 digits after the decimal point.

If there are multiple optimal placements available, return any of them. The input data guarantees that at least one solution exists.

Example

feng.in	feng.out
5 2 -2 0 -5 3 0 8 7 3 5 0	-2 3 3 2.5
4 3 0 0 0 8 10 8 10 0	3 5 7 3



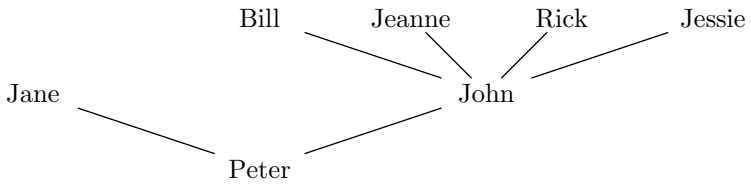


Problem G. Genealogy

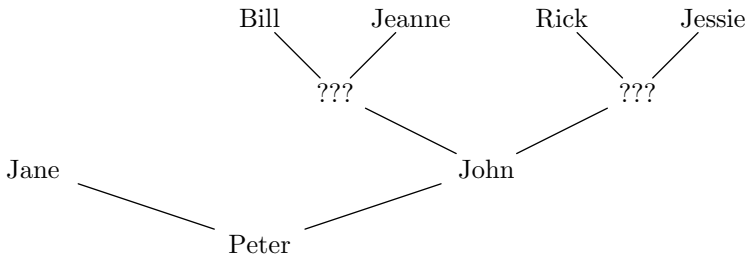
Input file:	`genealogy.in`
Output file:	`genealogy.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Alien Peter wants to trace his family pedigrees. Working hard for several weeks, he has created a beta-version of his family tree. Unfortunately, some of his ancestors have too much parents in this tree (aliens have d parents). So Peter thinks that some of parent-child relations actually are ancestor-descendant relations. Now Peter wants to know, what minimal number of ancestors need to be added to the tree to make it look well-formed (family tree looks well-formed if each alien has no more than d parents, each alien must appear at the tree only once).

For example, if $d = 2$, and beta-version of the family tree looks like this:



then Peter should add at least two ancestors to make it look well-formed:



Input

Let Peter's ancestors, appeared in the beta-version of his family tree, have identifiers from 1 to n (let Peter's identifier be 0).

The first line of input file contains numbers n and d ($2 \le n \le 100\,000$, $2 \le d \le n$). The following line contains n numbers, the i-th number is an identifier of the child of the i-th alien.

Output

Write the minimal number of Peter's ancestors, that should be added to this tree to make it look well-formed.

Example

genealogy.in	genealogy.out
6 2 5 5 0 5 0 5	2

Problem H. Halloween Holidays

Input file:	`halloween.in`
Output file:	`halloween.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Planet Cucurbita is inhabited with intelligent pumpkins. These pumpkins are not only extremely clever, they also are fond of tourism. One of their main routes is the Earth during Halloween.

As you know, pumpkins cannot move by themselves (intelligent pumpkins are not an exception), so they make somebody else to transport them. In the case of Halloween this is done by humans. First, they make people to grow special biological docking stations, then prepare the stations (people cut special holes, fire candles etc – you know the procedure), and after these preparations pumpkins come and have fun. People usually do not see anything and think that this is just a holiday and that this holiday is for humans, but remember – if somebody is frightened at Halloween, he was frightened not by his not-very-friendly friends, but by alien pumpkins.

To use the biological docking station, a pumpkin must have a special transmitter. It's main elements are two rings made of gold and for some unexplainable reasons these rings should be cut from one round plate. The sizes of these rings (inner and outer radii) are pumpkin-specific, so each alien should order a special set for himself.

Mr. Calabaza, an adolescent pumpkin, wants to make his first trip to the Earth. He found a discount plate, which was not redeemed by a previous customer, and it is necessary to check, whether this plate allows Mr. Calabaza to cut the rings he needs from it, or he should order a new larger plate.

Input

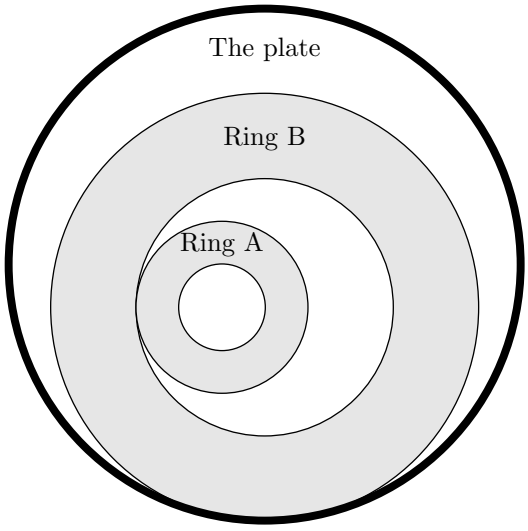
The input file contains five integer numbers A, a, B, b, P ($0 < A, a, B, b, P \le 1\,000\,000$, $a < A$ and $b < B$), separated with spaces. Here, A and B are the outer radii of the rings, a and b are the inner radii of the corresponding rings, and P is the plate radius.

Output

Output a word "`Yes`" if the plate suits Mr. Calabaza, or a word "`No`" if he needs to order another one.

Example

halloween.in	halloween.out
2 1 5 3 6	Yes

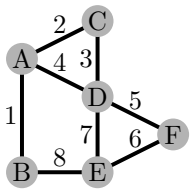


Problem I. Ideal Frame

Input file:	`ideal.in`
Output file:	`ideal.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

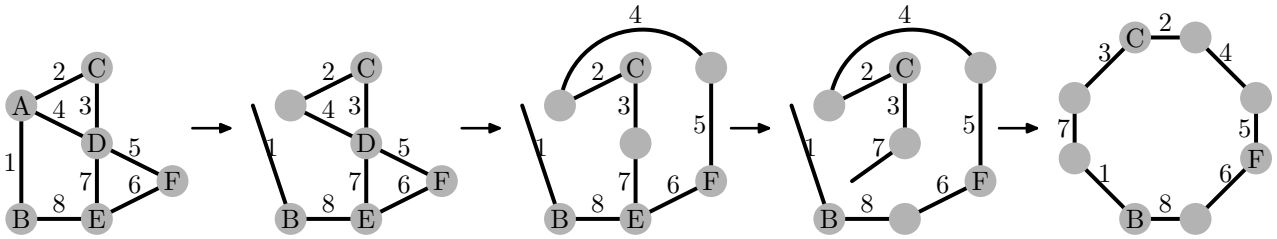
Peter is fond of collecting various funny things. He can spend hours restoring and polishing some useless item he had found in a garbage heap.

Recently he has found a really nice thing. It is a small metal construction that consists of several iron rods soldered together with little tin beads. An example of such construction is shown on the picture below.



Peter has immediately understood that these nice rods are exactly what he needs for the frame around the photo of his girlfriend. The frame must have a form of a closed chain of rods. To do the transformation Peter would unsolder some beads, releasing the rods ends, and after that solder some free rod ends together again. When unsoldering a bead, Peter can separate the ends of the rods connected at that bead, in arbitrary way — that is, he can choose an arbitrary partition of rods connected to the bead into non-empty groups, so that once the bead is unsoldered only the rods belonging to the same group remain connected to each other.

For example, to turn the construction on the picture above to a frame, he could unsolder the bead A, separating the end of rod 1, unsolder the bead D, separating pairs or rods 4–5 and 3–7 (but keeping the rods in pairs connected), and unsolder the bead E, separating the end of the rod 7. After that he needs to solder the free ends of rods 1 and 7 to get the frame 1–7–3–2–4–5–6–8.



Now help Peter to find the way to create the frame from the construction he has got by making as few operations as possible. There are two types of operations: unsoldering a bead, and soldering two free ends of rods together. Unsoldering is counted as one operation no matter how many rods were connected in a bead and how they were rearranged.

Input

The first line of the input file contains two integer numbers n and m — the number of beads and the number of rods, respectively ($0 \le n \le 1000$, $2 \le m \le 50\,000$). Let the beads be numbered from 1 to n. The following m lines contain two integer numbers each — the numbers of the beads connected by the corresponding rod. If the end of the rod is not connected to any bead, 0 is used.

The construction may be not connected (i.e. it can consist of several pieces). There can be beads that have only one rod soldered into it. These beads need not be unsoldered before the corresponding end of the rod is soldered to another one.

There can also be beads that have no rods soldered into it. Since Peter is only interested in rods, these beads need not be soldered/unsoldered.

Output

Output one integer number — the number of operations Peter needs to create the frame from the construction described in the input file.

Example

ideal.in	ideal.out
6 8 1 2 1 3 3 4 1 4 4 6 5 6 4 5 1 5	4
0 2 0 0 0 0	2
3 3 0 1 0 0 2 2	4

Problem J. Japanese Puzzle

Input file:	japan.in
Output file:	japan.out
Time limit:	2 seconds
Memory limit:	64 megabytes

A brand-new Japanese puzzle is coming from the East to strike the world-popular Sudoku game and become an international hit. The rules of this puzzle are kept in secret yet, but the goal is already advertised: given a square grid $n \times n$, where each square contains a block with one of k types of pictures, the player has to rearrange it to get the maximal possible number of equal first rows (two rows are considered equal if both of them are filled with the same pictures in the same order). An unnamed insider of the game production company told the press that the game is about moving blocks of pictures according to some rules, while the overall set of pictures isn't changed (no pictures removed, no new pictures added). She also mentioned that the puzzle is so exciting because there are thousands of ways to swap two arbitrary pictures on a grid leaving the rest of the grid intact.

Andy works at the puzzles review magazine, and of course he got interest in this Japanese news. He realized that the information known so far is enough to find the number of equal first rows in a puzzle winning position. Now Andy wants to write a computer program for calculating this number for any given starting configuration.

For example, if you are given a puzzle which looks this way:

≡	≡	+
≡	*	*
Δ	*	Δ

one of the optimal rearrangements could look like

≡	*	Δ
≡	*	Δ
+	≡	*

Input

The first line of the input file contains two integers n ($1 \le n \le 40\,000$) and k ($1 \le k \le 50\,000$). Each of the next k lines contains the number of blocks with the corresponding type of picture l_i ($l_i > 0$, sum of all l_i is exactly n^2).

Output

Output the maximal possible number of equal first rows at the first line of the output file. The following n lines must contain contents of the row which gives the maximum. Each line shows a single number of picture, in order they must appear. If there are many optimal solutions, any is acceptable.

Example

japan.in	japan.out
3 4 3 3 2 1	2 1 2 3

Problem K. Kennings

Input file:	`kenning.in`
Output file:	`kenning.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Kenning is a form of poetic metaphor, popular in ancient scaldic poetry, when a word is replaced by two or more words. For example, "giver of the gold" is a kenning for "warrior". This substitution is formal, there is no semantic difference between "poor giver of the gold" and "poor warrior". Kennings may be nested, so since "serpent's lair" refers to "gold", "giver of the serpent's lair" also refers to warrior.

Imagine that you need some long text by 9:00 AM yesterday. Don't panic, instead create the text plan and a list of kennings, and then expand the plan using the following algorithm. If the plan is long enough then stop — the text is ready. Otherwise simultaneously replace all words in the plan that have kennings with the corresponding kenning bodies and repeat the algorithm again.

Input

The first line of the input file contains three integer numbers: the width of the resulting text w ($1 \le w \le 255$), the minimal number of non-whitespace symbols in the resulting text l ($1 \le l \le 3\,000$), and the kennings list length n ($1 \le n \le 380$). The kenning list follows, a kenning per line. Each line contains the kenning referent followed by the kenning body. The text plan ends the file.

Each kenning body contains at least two words. Kennings may be recursive, like in replacing "GNU" with "GNU is Not UNIX". The kenning referents are case and grammatic form sensitive, so words "warrior", "Warrior" and "warriors" are different and may be refered by different kennings. The input file is not longer than 3 000 bytes, and contains only English letters, underscores, spaces, line feeds and digits (in the first line only). There are no two kennings with equal referents. All words have w symbols at most. Adjacent words are separated by exactly one space or line feed. No line has leading or trailing spaces.

Output

If the algorithm doesn't terminate, output just words "`No result`" in the only output line.

Otherwise output the algorithm result, a text with at most w characters (including spaces) per line. All line feeds from the original plan must be preserved, line feed must be inserted before a word if the word does not fit into the previous line. Adjacent words in a line must be separated by exactly one space. The lines must not have leading or trailing spaces. Correct output file will not be longer than 10 000 bytes.

Example

kenning.in	kenning.out
21 103 7 king hosts leader vessel windless bay of horns horns bulls spears spears war needles Sudden Fate catched death_of doomed to death Death It was the end Sudden death_of Fjolner in the house of Frodi Death of the king in the vessel	Fate catched doomed to death Fjolner in the house of Frodi It was the end of the hosts leader in the windless bay of bulls spears