Problem A. Assembler

Input file:	`asm.in`
Output file:	`asm.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

Your task is to calculate result of executing program written on simple assembler-like language. Typical program on this language operates with four 32-bit registers `AX`, `BX`, `CX` and `DX`. Each line of program consists of operator and optionally some operands. All possible operators and corresponding operands are listed in the following table:

Operator	**Constrains**	**Description**
`MOV` a b	a is register b is either register or constant	Moves value of b into a
`ADD` a b	a is register b is either register or constant	Adds value of b to a and puts result into a
`SUB` a b	a is register b is either register or constant	Subtracts value of b from a and puts result into a
`MUL` a b	a is register b is constant	Multiplies value of a by b and puts result into a
`LOOP` a	a is positive constant	Repeats all lines between this one and corresponding `ENDL` operator a times
`ENDL`	*none*	Used to finish list of commands which have to be repeated by corresponding LOOP operator.

All constants in the program fit into 32-bit signed integer. All operations are performed on 32-bit signed integers as well, so after executing the following program:

```
MOV AX 2147483647

ADD AX AX
```

`AX` will contain value -2.

There may be any number of nested loops.

Your aim is given the program and initial values of registers to return values of registers after executing the program.

Input

Input file contains program to be executed. Each line consists of operator, which is always uppercase, and corresponding operands. If operand is register, it will be written as two uppercase letters. There are no empty lines, as well as leading or trailing spaces. Operator and operands are separated by exactly one space. First four lines of program are always `MOV` operations, which define initial values of all four registers by some constants.

There will be no more than 10000 lines of code.

Output

Output four space separated integers: values of `AX`, `BX`, `CX` and `DX` after executing the program.

Example

asm.in	asm.out
MOV AX 15 MOV BX 20 MOV CX 25 MOV DX 30 LOOP 3 ADD AX BX ENDL MUL DX 3	75 20 25 90

Problem B. Blue-White Tree

Input file:	`bluewhite.in`
Output file:	`bluewhite.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

Your friend and you are playing the following game. You have a directed tree with some non-negative number of blueberries in each node. Also each node can be either blue or white. Initially all the nodes are white. Your friend and you in turns choose one of the nodes of the tree and perform with it one of the following operations:

- *Eat operation.* This operation may be applied only to nodes with positive number of blueberries. Player eats any positive number of berries and paints node to blue.
- *Collect operation.* This operation may be applied only to white nodes, which have at least one child, and all descendants (not only immediate children) of which are white. Player collects all the blueberries from all descendants of current node and place them into current node. Then he paints current node and all its descendants to blue.
- *Divide operation.* If current node has M blueberries and $M > 1$, players chooses positive number K ($K < M$, $M \ mod \ K > 0$), adds K blue child nodes to current node, each with floor(M/K) blueberries, than eats remaining $M-$floor$(M/K) \times K$ blueberries. If it is impossible to choose value of K satisfying above conditions, player is not able to perform divide operation on this node.

The one, who can't make any turn, loses the game. Your aim is to find who will win the game, if both your friend and you play optimally and you move first.

Input

First line contains one integer number N followed by $N-1$ integer numbers P_i, where P_i is parent of i-th node ($0 \le i < N$).

Second line contains N integer numbers B_i, where B_i is number of blueberries in i-th node.

$1 \le N \le 10^4$

$0 \le P_i < i$

$0 \le B_i \le 1024$

Output

For each test case on the first line of output file write `W` if you win the game and `L` if your friend does.

Example

bluewhite.in	bluewhite.out
5 0 1 1 1 8 0 1 2 4	L
5 0 1 1 1 7 0 1 2 4	W
3 0 0 0 0 0	W

Note: in the third example first player may perform collect operation on the 0*-th node, even though there are no blueberries in the whole tree.*

Problem C. Cruisers

Input file:	`cruisers.in`
Output file:	`cruisers.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

You're developing new game, and now your task is to create module, which is responsible for moving space cruisers. Each cruiser has several characteristics: *maximum speed*, *acceleration* and *turning ability*. Initially cruiser is located at coordinates *sx*, *sy*, with direction vector *dx*, *dy* and has speed *start speed*. Finally cruiser is going to reach coordinates *ex*, *ey*. It follows following actions to reach its destination:

- If it is directed exactly onto destination, it moves straight.
- Otherwise, draw two circumferences with radius, equal to *turning ability*, where point (*sx*, *sy*) is a point of tangent, and line, which goes through the points (*sx*, *sy*) and (*sx*+*dx*, *sy*+*dy*) is tangent to the both circumferences. You may assume, that point (*ex*, *ey*) doesn't lie on either of these circumferences.
- Cruiser moves along one of the circumferences, until it is directed onto destination, then it moves straight. If destination point isn't laying inside any of circumferences, cruiser choses circumference, for which distance cruiser have to follow along it is shorter (it is guaranted that these distances are different). Otherwise the one, which doesn't contain destination point inside, is chosen.

If cruiser has speed 0, it takes *acceleration* points of time to reach *maximum speed*, as well as if it has *maximum speed*, it takes *acceleration* points of time to stop. Cruiser accelerates and brakes with constant acceleration. It can't accelerate or brake slower. During flying, cruiser is going to accelerate until it reaches *maximum speed*, then move some points of time (probably zero, not necessarily integer) with *maximum speed*, and then brake until it stops at the destination. There are two exceptions:

- If after reaching *maximum speed*, it is impossible to stop at the destination point (i.e. even starting to brake immediately after reaching *maximum speed* cruiser is going to have positive speed when it reaches destination), cruiser first accelerate until it reaches some *special speed* and than brakes. *Special speed* is chosen in such way, that if cruiser starts braking immediately after reaching this speed, it is going to stop exactly at the destination point.
- If even starting braking at the first moment of time cruiser can't stop in the destination point, after reaching destination point it moves straight until it stops.

You aim, having all characteristics of cruiser, find out where it will be located at the moment of time *t*.

Input

First line of input contains four real numbers: ms, s, a and r — *maximum speed*, *start speed*, *acceleration* and *turning ability* respectively.

The second one contains six real numbers: sx, sy, ex, ey, dx, dy.

Third line of input contains one real number t.

$1 \le ms \le 1000.0$

$0 \le s \le ms$

$1 \le a \le 1000.0$

$-1000.0 \le sx, sy, ex, ey \le 1000.0$

$-100.0 \le dx, dy \le 100.0$

$0 \le t \le 1e12$

Output

On the only line there must be two real numbers x and y — coordinates of cruiser at the moment of time t. Your answer must be accurate up to four digits after decimal point.

Example

cruisers.in	cruisers.out

`10.0 0.0 10.0 5.0`
`0.0 0.0 100.0 0.0 1.0 0.0`
`10.0`
	`50.0 0.0`

`1.0 1.0 1.0 2.0`
`0.0 0.0 2.0 4.0 1.0 0.0`
`3.14159265358979323846264338327950`
	`2.0 2.0`

`1.0 1.0 1.0 2.0`
`0.0 0.0 1.0 2.0 1.0 0.0`
`3.14159265358979323846264338327950`
	`2.0 -2.0`

Problem D. Develop a Water-Pipe

Input file:	`develop.in`
Output file:	`develop.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

Recently you spent a lot of time playing the following game: there is a water-pipe, which is located on 2-dimensional plane and consisted of square tiles.

Each tile is either `I`-tile, which connects left and right edges of tile or top and bottom ones, `L`-tile, which connects left and top, or right and top, or left and bottom, or right and bottom edges of tile, `T`-tile, which connects any three edges, or, finally, `A`-tile, which connects all four edges.

Also there's a *sink* and *source* tiles. *Sink* tile connects its right edge with some external water-tank, while *source* node connects its left edge with the bath. There may be more that one *sink* and *source* nodes.

Water-pipe is *well-formed*, if all the tiles are placed in such a way, that if some tile connects, for example, its left edge with anything else, then tile, located on the left of that tile, necessarily connects its right edge with some other edges (or water-tank, or bath).

Well-formed water-pipe doesn't require *sink* to be connected to *source*.

Before the game started, all the tiles are rotated some times clockwise for 90°, and then your aim is, rotating them, restore *well-formed* water-pipe.

After some easy levels rules of the game got a little bit more difficult. Especially, the water-pipe can't be made well-formed until you change some `T`-tiles to `A`-tiles.

After some time you have shown this game to your friend, and he began to solve levels much faster then you did.

You don't want to be beaten by your friend and now you need to write a program, which will solve this game in order to solve all remaining levels before you friend has done it.

Input

First line of input contains two integer numbers — N and M — dimensions of field. Then $3N + 1$ lines follow, each contains $4M + 1$ characters. Each tile is defined by 4 rows of 5 characters according to the following table (only dot, underscore, space and vertical slash symbols are used):

`.....  .....  .....  .....`
`. |_.  ._| .  .  _.  ._  .`
`.   .  .   .  . | .  . | .`
`.....  .....  .....  .....`
	`L`-tiles

`.....  .....`
`.___.  . | .`
`.   .  . | .`
`.....  .....`
	`I`-tiles

`.....  .....  .....  .....`
`. |_.  ._|_.  ._| .  ._ _.`
`. | .  .   .  . | .  . | .`
`.....  .....  .....  .....`
	`T`-tiles

`.....`
`._|_.`
`. | .`
`.....`
	`A`-tiles

`.....  .....`
`. )\.  ./( .`
`. )/.  .\( .`
`.....  .....`
Note: here dot, slash, backslash, space and parentheses symbols are used
	Sink and *Source* respectively

`.....`
`.   .`
`.   .`
`.....`
	Empty tile

$1 \le N \le 25$

$1 \le M \le 25$

For more details see sample input.

Output

Output *well-formed* water-pipe, obtained from given one by rotating any number of tiles and by replacing some (possibly zero) `T`-tiles to `A`-tiles in the same format, as in input file. If it is impossible to obtain *well-formed* water-pipe, output file must contain single word `IMPOSSIBLE` without quotes

Example

develop.in

```
3 6
.........................
. )\.___. | ._| .   .   .
. )/.   . | .   .   .   .
.........................
.   .   .   .___.   .   .
.   .   .   .   .   .   .
.........................
.   .   .   ._| .___./( .
.   .   .   .   .   .\( .
.........................
```

develop.out

```
.........................
. )\.___.___._   .   .   .
. )/.   .   . | .   .   .
.........................
.   .   .   . | .   .   .
.   .   .   . | .   .   .
.........................
.   .   .   . |_.___./( .
.   .   .   .   .   .\( .
.........................
```

develop.in

```
3 6
.........................
. )\._ _. | ._| . |_./( .
. )/. | . | . | . | .\( .
.........................
. |_._|_._| .___. | .   .
.   .   .   .   . | .   .
.........................
. |_._|_._|_._|_._ _./( .
.   .   .   .   . | .\( .
.........................
```

develop.out

```
.........................
. )\._ _.___._ _._ _./( .
. )/. | .   . | . | .\( .
.........................
.  _._|_._  . | . | .   .
. | . | . | . | . | .   .
.........................
. |_._|_._|_._|_._|_./( .
.   .   .   .   .   .\( .
.........................
```

develop.in

```
3 6
.........................
. )\.___. | ._| . |_./( .
. )/.   . | . | . | .\( .
.........................
. )\._|_.   .___. | .   .
. )/.   .   .   . | .   .
.........................
.   .   .   ._| ._ _./( .
.   .   .   .   . | .\( .
.........................
```

develop.out

```
IMPOSSIBLE
```

Problem E. Expected Number of Crystals

Input file:	`expected.in`
Output file:	`expected.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

In one very popular on-line game crystals are very important resource. One of the ways to collect crystals is to use magic meadows. The process of collecting crystals on the meadow is following: there's a field, consisted of $N \times N$ cells. M of that cells contain crystal, while others do not. You may reveal any K cells, and collect all revealed crystals.

There may be two different modes of the game — *blind* and *basic*. In *blind* mode first you choose cells, and then all of them get revealed and you know how many crystals you have collected. In *basic* mode after revealing each cell you immediately know whether it contains crystal or not.

Now you wonder what is expected number of crystals you are going to collect, knowing N, M, K and mode.

Input

First line on input contains four integer numbers: N, M, K and V, where V equals to one if you play in *blind* mode, and V equals to two, if you play in *basic* mode.

$1 \le N \le 10^{30}$

$1 \le M \le 10^{30}$

$1 \le K \le 10^{30}$

$M \le N^2$

$K \le N^2$

Output

If the expected number of crystals is integer, output it on one line by itself, otherwise write answer as an irreducible fraction. See second example for clarification.

Examples

expected.in	expected.out
`3 3 3 1`	`1`
`6 10 10 1`	`25/9`

Problem F. Forum Search

Input file:	`forum.in`
Output file:	`forum.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

You're asked to develop a search engine for a forum. The forum consists of threads, each thread consists of posts, and each post is a space-separated set of lowercase words.

Each search query consists of the set of keywords. There are three kinds of search queries:

1. To find all threads, which contain all the keywords inside one post, independently of order. For example, if the query is «`A B C`», and the post is «`D B C A`», the thread, which contains this post, must be returned by the engine.
2. To find all threads, which contain all the keywords inside one post as a subsequence. For example, if the query is «`A B C`», and the post is «`D B C A`», the thread, which contains this post, must not be returned by the engine, while if the post is «`A D B C`», it must be returned.
3. To find all threads, which contain all the keywords inside one post as a substring. For example, if the query is «`A B C`», and the post is «`D B C A`» or «`A D B C`», the thread, which contains this post, must not be returned by the engine, while if the post is «`D A B C`», it must be returned.

You are given the forum content and the set of queries, for each query find all the threads, which satisfy the query.

Input

First line of input file contains one integer number N — the number of threads.

Each thread is described in the following way: first line of the description contains integer number M — the number of posts in the thread, then M lines follow, each contains space separated list of lowercase English words — content of the corresponding post.

After description of threads there is a line with integer number K— number of queries. Each query is described on a separate line as a the type of the query (1, 2 or 3) followed by an arbitrary number of lowercase keywords.

Input file size does not exceed 400KB. The total number of keywords in all queries do not exceed 12000. Each keyword is at least four letters long. All keywords within one particular query are distinct.

You may assume, that all the threads (except the sample test) are random discussions given from one of the real forums about some fantasy epic saga without flood posts.

Even though queries may contain often used words like «that» and «have», the amount of such words among all keywords is low enough to assume that queries almost do not contain such words.

Output

For each query produce one line of output. The line must start with number of threads, which is going to be returned by the engine, followed by description of found threads. For each found thread write its index (1-based), number of posts in the thread, which satisfy the query, and then 1-based indexes of that posts inside the thread. Indexes of threads and posts must be sorted in ascending order.

Each query result must be written on separate line of output file.

Example

forum.in	forum.out

2
2
d b c a
a d b c
1
d a b c
3
1 a b c
2 a b c
3 a b c
	2 1 2 1 2 2 1 1
	2 1 1 2 2 1 1
	1 2 1 1

Problem G. Get the Duck to the Sink

Input file:	`getduck.in`
Output file:	`getduck.out`
Time limit:	4 seconds
Memory limit:	256 Mebibytes

You are playing the following game. There's a field of size NxM tiles and some (possible zero) walls between tiles. One of the tiles is a *sink*, and one of the other tiles is occupied by a *duck*. Your aim is to bring the *duck* to the *sink*. The only kind of movements you can do with the *duck* is *sliding*, which means that you can push the *duck* in any direction, and it will move in that direction until it has touched any wall or the border of the field. You can't push the *duck* again until it has stopped. To solve level the *duck* must stop on the *sink*, just *sliding* through the *sink* is not enough.

After playing this game for a long time you solved all the levels and now you want to do more. You want to generate some new levels. You have drawn a field with walls and *sink*, and now you need to place the *duck* into some place. You have chosen some tiles, into which you want to place it. But soon you realized, that there are some of them, beginning from which it is impossible to solve the level. Now your aim is, having the size of the field, positions of the walls, position of the *sink* and all chosen positions of the *duck*, to find all the places among chosen ones, starting from which it is possible to solve the level.

Input

First line contains two integer numbers N and M — dimensions of the field.

Then $2N + 1$ lines follow, each contains $2M + 1$ characters, where $2k$-th character of $2i$-th line is either space, if tile is empty, `S` if tile contains *sink* and `D` if tile is supposed to contain *duck*.

$(2k + 1)$th character of $2i$-th line is either space if $k > 0$ and $k < M$ and there's no wall between cells (k, i) and $(k + 1, i)$; or `|` otherwise.

$2k - th$ character of $(2i + 1)$th line is either space if $i > 0$ and $i < N$ and there's no wall between cells (k, i) and $(k, i + 1)$; or `-` otherwise.

$(2k + 1)$th character of $(2i + 1)$th line is always `+`.

$1 \le N \le 1000$

$1 \le M \le 1000$

Output

Output file must contain field from input file without D-letters in places, starting from which it is impossible to reach sink.

Example

getduck.in

```
5 5
+-+-+-+-+-+
|         |
+ +-+ + + +
|  S   D D|
+ + + + + +
|    D  | |
+ + + + + +
| |     | |
+ + + +-+ +
|         |
+-+-+-+-+-+
```

getduck.out

```
+-+-+-+-+-+
|         |
+ +-+ + + +
|  S   D  |
+ + + + + +
|    D  | |
+ + + + + +
| |     | |
+ + + +-+ +
|         |
+-+-+-+-+-+
```

Problem H. How Many Chipmunks are There?

Input file:	`howmany.in`
Output file:	`howmany.out`
Time limit:	3 seconds
Memory limit:	256 Mebibytes

In the secret laboratory KoDaPet (which name is abbreviation of last names of her members), which is also known as Lopatin and Porschegi, new kind of war chipmunks is invented. They're much cleverer then any other chipmunks, even that Chip and Dale. But work is not finished yet, and scientists from this laboratory still are in process of getting new even more powerful and even smarter chipmunks. Each chipmunk has its own strength and knowledge. Though, each chipmunk in KoDaPet is individual, so even if two chipmunks has exactly same strength and knowledge, they are completely different.

You probably want to ask, how does KoDaPet achieve new chipmunks? Very easily. All that you need to know to understand it is what two chipmunks do, when they meet each other.

If two chipmunks meet and difference in their strengths is 7 or more, then more strength one is going to eat weaker one. Otherwise, chipmunks are going to play different strange game, and new chipmunk appears. This new chipmunk has strength and knowledge equal to the sum of strengths and sum of knowledges of its parents respectively. But, if his parents will meet again, and play their strange game again, new chipmunk will be exactly the same as previous one created by these parents, and, because he violates «each chipmunk is individual» rule, he is going to be killed by scientists immediately.

Actually, scientists don't want chipmunks to conquer the world, so if for particular chipmunk his knowledge exceeds 4000 or his strength exceeds 20, he is going to be killed immediately as well.

Initially scientists have some chipmunks with some knowledges from 1 to 4000 inclusively and strength 1. They want to produce some chipmunks with knowledge k and strength s. Unfortunately, according to old and truthful prophecy, if for particular knowledge and strength maximum possible number of chipmunks, that can be achieved, equals to m modulo 40961 (which is prime), the world would be in a terrible danger if you try to produce them, so you're asked to check whether this condition satisfied or not.

Input

First line of input contains three integer numbers m, k and s.

Second line contains from 1 to 4000 integer numbers a_i — number of chipmunks with knowledge i and strength 1 scientists already have. If line contains less than 4000 numbers, scientists initially do not have chipmunks with higher knowledge.

$1 \le k \le 4000$

$1 \le s \le 20$

$0 \le m < 40961$

$0 \le ai < 40961$

Output

If the maximum number of chipmunks that can be achieved equals to m modulo 40961, write only line with word `YES`. Otherwise write `NO`.

Example

howmany.in	howmany.out
30 4 4 3	YES

Problem I. Inhabitants (or Yet Another Mincost Maxflow Problem)

Input file:	`inhabitants.in`
Output file:	`inhabitants.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

The Dark Lord has got the control of the Country. Every person is strictly forbidden to leave the city he is currently in, even though there's a lot of one-way roads, which connects different cities of the country.

Actually, Dark Lord doesn't watch what people of the country do all the time. Most of the time he weaves plots against the guy, whose name is Rand, with thirteen Chosen Ones. All that Dark Lord really does in order to control the country is he checks whether the number of citizens in each city remains the same between two check-ups. The time between two consecutive check-ups is constant and equals to one month. If during the check-up Dark Lord realizes, that the number of citizens in one or more cities have been changed since the last check-up, he will immediately destroy the whole Country.

Unbelievable coincidence, but it takes exactly one month to move from any city to any other city, if there's a road between them with corresponding direction. So if the person leaves the city exactly after the check-up, he is going to reach other city right before the following check-up. The only problem that this action must not change the number of citizens neither in the city he left nor in the destination city, so some other person must have left the destination city while one more person must have come to city the first person left.

Being under Dark Lord's control, people are getting unhappy. And the level of average happiness in the country is getting worse and worse. At the same time, roads have kind of magic effect: they change happiness of each person, traveled using them. More precisely, each road has it's own *magic value*, and when someone goes by this road, his happiness reduces by *magic value* of the road.

Many people outside the Country want to enter it, while a lot of people inside the country want to leave it. You are probably going to ask, how can anybody want to enter the Country which is under the Dark Lord's control. I also wonder why. Nevertheless, they want.

There are N cities in the Country, numbered from 1 to N. There's only one road, by which people can enter the Country. This road leads from outside the Country to the city 1. Also, the only road, by which people can leave the country, leads from city N to outside.

These two roads are very wide, and any number of people can enter or leave the country within one month, while each road inside the country has *capacity value* — the maximum number of people who can pass this road within one month.

You're one of the followers of the Creator, and now your aim is to satisfy as many enter or leave requests as possible, if it is known that the number of these requests is definitely more than the Country can satisfy without being destroyed. Also, you want to keep the average happiness of the Country as high as possible. Unfortunately, is it not known how many people are there in each city (but you may assume that for each particular city this number is strictly greater then the sum of *capacity values* of all the roads in the country), so you can't calculate neither average happiness before people, who are going to move to other city, left their cities nor after they reached their destinations. But of course you can calculate the average change of happiness of people who used their opportunity to move to another city. So, your final aim is for one particular month:

1. Find out what is the maximum number of people who can enter the Country and the number of people who can leave it (as you see, these number are equal);
2. After maximizing the number of people who enter the country, minimize the average value, which will be subtracted from happiness of people, who are going to move from one city to another (not

including ones who entered or left the country).

Input

First line on input contains two integer number N and M — number of cities and roads respectively. Then M lines follow, each contains four integer numbers: u_i, v_i, c_i, m_i, where i-th road leads from u_i to v_i, has capacity value c_i and magic value m_i.

$2 \le N \le 20$

$1 \le M \le 20$

$1 \le u_i \le N$

$1 \le v_i \le N$

$1 \le c_i \le 6$

$-10 \le m_i \le 10$

There's at least one directed path from 1-st city to N-th one.

Two cities may be connected by any number of roads, as well as road may connect the city with itself.

Output

The only line of output file must contain one real number — minimum average number, which was subtracted from happiness of people who traveled during one month. Your answer must be accurate up to four digits after decimal point.

Examples

inhabitants.in	inhabitants.out
2 1 1 2 1 4	4.0

3 4
1 2 2 1
2 3 1 2
2 3 1 3
2 3 1 4
	1.75

7 6
1 2 2 5
2 3 2 5
3 7 2 5
4 5 2 3
5 6 2 3
6 4 2 3
	4.0

7 6
1 2 2 5
2 3 2 5
3 7 2 5
4 5 2 6
5 6 2 6
6 4 2 6
	5.0

Problem J. JavaScript Interpreter

Input file:	`js.in`
Output file:	`js.out`
Time limit:	2 seconds
Memory limit:	256 Mebibytes

In this problem you have to implement simplified interpreter of JavaScript language, which operates on simplified document object model of an HTML page.

Page is going to be shown in the text browser with resolution of 20 rows and 40 columns.

Initially, before executing JavaScript code, page is absolutely empty, and document model contains the only element — *document*.

To create new element method *createElement* of *document* object is used. It takes one parameter, which for this problem is always equal to `"span"` in double quotes, and returns the pointer to the created element.

Each element has following attributes:

attribute name	**constrains**	**description**	**default value**
left	integer $-100 \le left \le 100$	Offset from parent element's left border	0
top	integer $-100 \le top \le 100$	Offset from parent element's top border	0
width	integer $0 \le width \le 100$	Width of the element	10
height	integer $0 \le height \le 100$	Height of the element	10
border	boolean *true* or *false*	Whether element has border or not	*false*
zIndex	integer $0 \le zIndex \le 100$	Elements with higher zIndex are drawn over the ones with lower zIndex	0
innerText	string From 0 to 100 characters	Text inside the element	*Empty string*
align	string «center» or «left» or «right»	Text align inside the element	«left»

Program may contain any number of variables, where variable name is a set of English letters, digits and underscore characters. First character of variable name is always letter. Variable names are case sensitive.

Each variable has to be defined before it is first time used. Initially the only defined variable is *document*. There are four types of variables in this problem: element, integer, string and boolean.

To define new variable the following constructions are used:

`var` *variable_name* = *value*;

or

`var` *variable_name* = *other_variable_name*;

Where *value* is either integer number, string in double quotes, true or false or calling of *createElement* method. It uniquely defines type of the variable. Note, that after command

`var moo = "5";`

`moo` have type string, not integer.

To change value of some variable following constructions are used:

variable_name = *value*;

or

variable_name = *other_variable_name*;

All elements are stored as references. It means that executing following code:

```
var moo = document.createElement( "span" );
var q = moo;
q.width=100;
```

Will change width of both `q` and `moo` (because `q` and `moo` point to the same element).

Though, the following code

```
var moo = document.createElement( "span" );
var q = moo;
q = document.createElement( "span" );
q.width=100;
```

Will not change width of `moo`, because `q` points now to other element.

All strings, booleans and integers are stored by value, not by reference.

To change any attribute except innerText of some element the following construction is used:

element.`style`.*attribute_name* = *value*;

or

element.`style`.*attribute_name* = *variable_name*;

To change *innerText* the same construction, but without *style* member, is used.

Members of *style* and *innerText* are also variables, so the following code is OK:

```
moo.style.left = q.style.top;
```

To add child element to an existing one following construction is used:

element1.`appendChild(` *element2* `);`

Where *element2* is either variable, which contains element, or calling of *createElement* method, and *element1* is either already created span element or *document*. *element1* doesn't point to any descendant of *element2* or to *element2* itself.

You may assume, that code consists only of described constructions, as well as all operations of changing attribute use values or variables of appropriate type (for example there will never be construction which tries to assign string value to *left* attribute).

document has type *element* (slightly extended by adding method *createElement*), but the only properties of it which may be changed are *innerText* and *align*.

Commands are separated by semicolon character. Any number of whitespace characters may be used anywhere but inside variable names, methods, variable values or attribute names. Whitespaces are characters with ASCII codes 9, 10, 13 and 32 (‘`\t`’, ‘`\n`’, ‘`\r`’ and ‘ ’ respectively).

Your aim is to execute program and render the page after performing all operations.

The following rules must be followed during rendering page:

1. If two elements (which have the same parent) overlap, the one with higher zIndex is rendered on the overlapped area. If their zIndex-es are equal, the one which was added to the parent element later is rendered on the overlapped area.
2. If element doesn't have border, it changes to space every character of the parent with coordinates

x, y for which both *element's left* $\leq x <$ *element's left* + *element's width* and *element's top* $\leq y <$ *element's top* + *element's height*.

3. If elements does have border, it additionally surrounded by `'+'`, `'-'` and `'|'` characters (see sample output for details). Border is drawn outside the area filled by the span.

4. Text is written inside the span. If text length is higher that *width* of span, it has to be written on several lines. To separate text follow the following algorithm: if after writing new word current line will not exceed *width* of the span, write it. Otherwise start new line. If word's length is higher then *width* of the span, write *width* characters on one line and repeat process for remaining part of the word starting from the next line. If text *align* is «center» and one of the lines can't be centered exactly (i.e. its length doesn't equal to *width* of the span modulo 2), one trailing space must be added to this line. If number of lines of text is more than *height* of span, all extra lines must be omitted. Note that in this problem *innerText* is always single space separated set of English words and numbers.

5. Everything, that doesn't fit into the parent span (or screen, if parent of the span is document), including borders, must be omitted.

Input

Input file contains code to be executed

Code won't contain more than 1000 semicolons.

Output

Write 22 lines, each containing 42 characters — the final state of the screen, surrounded with a border. See sample output for details.

Example

js.in

```
var el = document.createElement( "span" );
var el2 = document.createElement( "span" );
var el3 = document.createElement( "span" );
var q = true; el.style.border = q; el2.style.border = q;
el.style.width = 20; el.style.height = 15; el.style.top = 2;
el2.style.left = 5; el2.style.top = el2.style.left;
el2.style.width = 5;
el2.innerText = "abra abra abra abra abra ab ra kadabra z obama";
el3.style.left = 21; el3.style.top = 3; el3.style.height = 15;
el3.innerText = "Moo cow likes fresh grass very much";
el3.style.align = "center";
el3.style.border = true;
el.appendChild( el2 ); el.appendChild( document.createElement( "span" ) );
document.appendChild( el );
document.appendChild( el3 );
document.innerText = "This problem is not supposed to be solved";
document.style.align = "right";
el.style.zIndex = 1;
```

js.out

```
+----------------------------------------+
|      This problem is not supposed to be|
|--------------------+             solved|
|                    |----------+        |
|                    | Moo cow  |        |
|                    |  likes   |        |
|                    |  fresh   |        |
|          +         |grass very|        |
|          |         |   much   |        |
|          |         |          |        |
|          |         |          |        |
|          |         |          |        |
|          |         |          |        |
|    |ab ra|         |          |        |
|    |kadab|         |          |        |
|    |ra z |         |          |        |
|    |obama|         |          |        |
|    |     |         |          |        |
|--------------------+          |        |
|                    +----------+        |
|                                        |
+----------------------------------------+
```

Problem K. Keep Them Separately! (Division 2 Only)

Input file:	`keepsep.in`
Output file:	`keepsep.out`
Time limit:	3 seconds
Memory limit:	256 Mebibytes

As you probably know, in Japanese schools pupils often visit different clubs attached to the school. In one of such schools theater club wants to perform a set of shows. There are k seats in the concert hall of theater club, and club plans to invite exactly k pupils for every show. Every pupil, invited to the show, has its own ticket with number of his seat. The leader of the club wants to distribute tickets in such way that for every two shows there is at least one seat that was occupied by pupils from different clubs on these two shows.

Moreover, some clubs are rival, for example members of karate club are never going to take seat adjacent to the seat of one of the members of pink butterfly lovers club and visa versa. Now theater club wonders how many shows can they perform before either they have to distribute seats in exactly same way as on one of previous shows or they have to seat two pupils from rivals clubs to the adjacent seats.

Two seats are considered adjacent if the difference in their numbers is equal to one.

Input

First line of input contains three integer numbers: n, m and k — number of clubs, number of pair of clubs which are rival, and number of seats in the concert hall. m lines follow, each contain 2 integer numbers a_i, b_i, where ai and bi are numbers of clubs which are rival. All the pairs are distinct.

$1 \le k \le 100$

$1 \le n \le 100$

$1 \le ai < bi \le n$

Output

The only line of output file must contain one integer number — number of shows theater club can perform.

Examples

keepsep.in	keepsep.out
2 1 3 1 2	2

3 2 3
1 2
1 3
	9

Note: In the first example possible seatings are 1-1-1 and 2-2-2. In the second example possible seatings are 1-1-1, 2-2-2, 2-2-3, 2-3-2, 2-3-3, 3-2-2, 3-2-3, 3-3-2 and 3-3-3.

Problem L. Numbers on the Field (Division 1 Only)

Input file:	`numbers.in`
Output file:	`numbers.out`
Time limit:	2 seconds
Memory limit:	384 Mebibytes

Once there was very interesting article on the IT Happens web-site. One fellow was telling that he asked his friend, who was a programmer, to solve the following problem:

There's a 5×5 field. You have to place numbers from 1 to 4 into its cells, following the following rules: 1 may be placed into any cell. 2 may be placed only into cells, which are adjacent with at least one cell which contains 1. 3 may be placed only into cells, which are adjacent to at least one cell with 1 and at least one cell with 2. Finally, 4 may be placed only if it is adjacent to at least one cell with 1, to at least one cell with 2 and to at least one cell with 3. The aim is to place numbers in such a way, that their sum is maximal possible.

Two cells are considered adjacent if they share an edge.

Actually, the friend of author's of the article solved this problem. And now you are supposed to do the same. The only additional moment: the field in this problem isn't necessarily empty, it may contain some numbers inside already. And your aim is to place numbers into free cells in such way, that both added numbers and numbers that were initially in the field satisfy problem conditions, and among all such placements choose one, which maximizes the sum of numbers. If there's more than one such placement, any of them may be chosen.

Input

Input file contains five lines, each line contains five space separated numbers from 0 to 4, where 0 means that the cell is empty.

Output

If it is impossible to place numbers in order to satisfy the requirements, output file must contain the only line `-1`.

Otherwise on the first line of input write the maximal possible sum. Then write five rows with five space separated numbers on each — the resulting placement.

Examples

numbers.in	numbers.out

1 0 0 1 1
0 0 1 1 1
0 1 1 1 0
1 1 1 0 0
1 1 0 0 1
	43
	1 2 3 1 1
	4 3 1 1 1
	2 1 1 1 3
	1 1 1 4 2
	1 1 2 3 1

1 2 0 2 1
2 3 0 3 2
3 4 0 4 3
1 2 0 2 1
2 3 0 3 2
	55
	1 2 3 2 1
	2 3 1 3 2
	3 4 1 4 3
	1 2 3 2 1
	2 3 1 3 2