Problem A. Generalized Assignment

Input file:	`assignment.in`
Output file:	`assignment.out`
Time limit:	1 second
Memory limit:	64 megabytes

The famous assignment problem is formulated as follows: given $n \times n$ matrix, choose one element in each row in such a way that there is one element chosen in each column, and the sum of the chosen elements is minimal possible.

Let us generalize this problem. Given $n \times n$ matrix, choose k elements in each row in such a way, that there are k elements chosen in each column, and the sum of the chosen elements is minimal possible.

Input

The first line of the input file contains n and k ($1 \le k \le n \le 50$). The next n lines contain n integer numbers each and represent the given matrix. The elements of the matrix do not exceed 10^3 by their absolute values.

Output

Output one number — the minimal possible sum of the elements, chosen in a way described in a problem statement.

Example

assignment.in	assignment.out

4 2
1 2 3 4
2 3 4 5
1 3 5 7
5 7 2 4
	22

Problem B. Bandits

Input file:	`bandits.in`
Output file:	`bandits.out`
Time limit:	1 seconds
Memory limit:	64 megabytes

After robbing a caravan on a road, m bandits have obtained a loot of n similar diamonds. They decided to divide the loot. To do so they have ordered themselves by their birthdate, and make proposals in turn.

When the proposal is made, the bandits vote either for, or against it. If the strict majority of bandits votes for the proposal, it is accepted, in the other case the author of the proposal is killed.

The proposal states for each of the still alive bandits, what number of diamonds should he get. All bandits are smart, greedy, bloodthirsty and careful. That means, that the bandit votes against the proposal if and only if he is certain that he will survive and will get the same number of diamonds or more in future assuming that all the other bandits also vote optimally.

Find out what is the maximal number of diamonds the youngest bandit (the one who makes the first proposal) can get. If he is killed regardless of his proposal, output “-1”.

Input

Input file contains m and n ($1 \le m \le 2000$, $1 \le n \le 2000$).

Output

Print the maximal number of diamonds the first bandit can get, or “-1” if he is killed regardless of his proposal.

Example

`bandits.in`	`bandits.out`
`5 1000`	`997`
`2 1000`	`-1`

Problem C. Matrix Game

Input file:	`matrix.in`
Output file:	`matrix.out`
Time limit:	1 second
Memory limit:	64 megabytes

Fred and Steven play the following game. They take a matrix $a_{i,j}$ of size $n \times n$. After that each of them creates a vector of n non-negative real values, that sum to 1. So, Fred selects n numbers: $x_1, x_2, \ldots, x_n$ such that $x_i \ge 0$ and $\sum x_i = 1$, and Steven selects n numbers $y_1, y_2, \ldots, y_n$ such that $y_i \ge 0$ and $\sum y_i = 1$.

After they create their vectors, they calculate the *value* of the game:

$$v = x^T A y = \sum_i \sum_j x_i a_{i,j} y_j.$$

This is the value that Fred wins. If $v > 0$, Steven pays to Fred v dollars, if $v < 0$, Fred pays to Steven $-v$ dollars.

Help Fred to find such vector x that the value of the game in the worst case is as large as possible.

Input

The first line of the input file contains n ($2 \le n \le 8$). The following n lines contain n integer numbers each — the given matrix. The elements of the matrix do not exceed 100 by their absolute values.

Output

On the first line of the output file print v — the maximal possible value of the game. After that print n real numbers $x_1, x_2, \ldots, x_n$ — the vector Fred should create. The answer must be accurate up to 10^{-4}.

Example

matrix.in	matrix.out

2
1 0
0 1
	0.5
	0.5 0.5

Problem D. Paper Mosaic

Input file:	`mosaic.in`
Output file:	`mosaic.out`
Time limit:	3 seconds
Memory limit:	64 megabytes

Little Peter has recently found several hardpaper circles of equal radius in the old box on the attic. He decided to make a mosaic out of them. So he cut several polygons out of the circles.

Before cutting the polygons, he had drawn them on the circles. The vertices of each polygon were on the border of the circle, so each polygon was inscribed into the circle it was cut out of. Polygons did not intersect, but could touch each other. Peter could cut several polygons out of one circle.

Peter has shown his mosaic to his friend John. Little John wonders — how many circles did Peter use to create the mosaic. But Peter cannot remember. Given polygons from the mosaic, help the boys to find out what minimal number of circles Peter could use to create his mosaic.

Input

The first line of the input file contains n — the number of polygons in Peter's mosaic ($1 \le n \le 20$). The following n lines describe polygons. Each line starts with m_i — the number of vertices in the polygon ($3 \le m_i \le 6$). After that m_i real numbers follow — the lengths of the sides of the polygon in counterclockwise order. All sides have length from 10^{-2} to 10^{2}.

It can be proven that the information about the lengths of the sides is enough to restore the inscribed polygon. It is guaranteed that all polygons have the same radius of the circle they are inscribed to.

Output

Output one number — the minimal number of circles that Peter could use to cut out all the polygons of his mosaic.

Example

mosaic.in	mosaic.out

2
3 3 4 5
3 5 4 3
	1

Problem E. Shortest Path

Input file:	`path.in`
Output file:	`path.out`
Time limit:	1 second
Memory limit:	64 megabytes

You are given a weighed directed graph and a vertex s in it. For each vertex u find the length of the shortest path from s to u.

Input

The first line of the input file contains n, m and s — the number of vertices and edges in the graph, and the number of the starting vertex, respectively ($2 \le n \le 2\,000$, $1 \le m \le 5\,000$).

The following m lines describe edges. Each edge is specified with its start vertex, its end vertex, and its weight. The weight of any edge is integer and does not exceed 10^{15} by its absolute value. There can be several edges between a pair of vertices. There can be an edge between a vertex and itself.

Output

Output n lines — for each vertex u output the length of the shortest path from s to u, '`*`' if there is no path from s to u, or '`-`' if there is no shortest path from s to u.

Example

path.in	path.out

```
6 7 1
1 2 10
2 3 5
1 3 100
3 5 7
5 4 10
4 3 -18
6 1 -1
```
	```
	0
	10
	-
	-
	-
	*
	```

Problem F. Cutting Puzzle

Input file:	`puzzle.in`
Output file:	`puzzle.out`
Time limit:	1 second
Memory limit:	64 megabytes

A well known puzzle question is: how many cuts are needed to cut a $3 \times 3 \times 3$ cube into unit cubes? After each cut you are allowed to rearrange the parts in a way you like, so you may cut several parts in one action.

The answer is: six, because all six faces of the interior cube must be cut out.

The puzzle can be easily generalized: how many cuts are needed to cut an $a \times b \times c$ parallelepiped into unit cubes? This is exactly the question you have to answer in this problem. Of course, after each cut you are allowed to rearrange the parts in a way you like.

Input

Input file contains three integer numbers: a, b, and c ($1 \le a, b, c \le 10^{1000}$), separated by spaces.

Output

Output one integer number — the required number of cuts.

Example

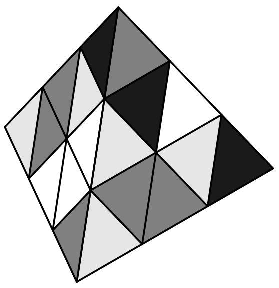
puzzle.in	puzzle.out
`3 3 3`	`6`

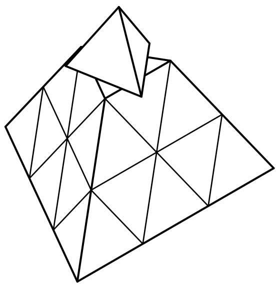
Problem G. Pyraminx

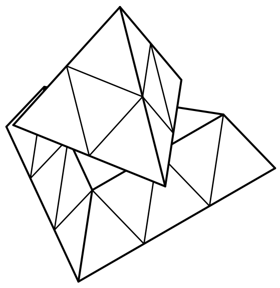
Input file:	`pyraminx.in`
Output file:	`pyraminx.out`
Time limit:	15 seconds
Memory limit:	128 megabytes

Pyraminx is a puzzle similar to the famous Rubik's cube. The puzzle has a form of a regular tetrahedron that is cut to three parts of equal height in each direction parallel to one of its faces. These cuts divide each of its faces into 9 triangles. Therefore the surface of the Pyraminx is divided into 36 triangles.

Each triangle is colored into one of the four different colors. There are 9 triangles of each color. To solve the puzzle one must rearrange the triangles in such a way that each face is colored in its own color. To do so, it is allowed to make moves represented by various rotations of the parts of the tetrahedron.


Pyraminx


Small rotation


Big rotation

There are two types of rotations: small and big. In a small rotation you may turn a small tetrahedron near one of the vertices around its central axis 120 degrees clockwise or counterclockwise. The big rotation is similar to the small one, but you rotate the part of the tetrahedron consisting of two layers.

Given the initial combination of triangles on the tetrahedron, find the sequence of moves that rearranges them in order to sovle the puzzle.

Input

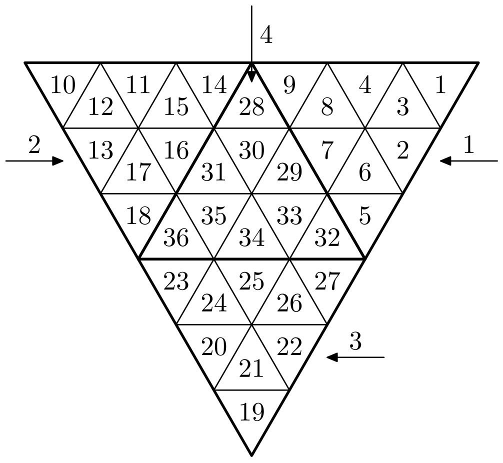
Input file contains 36 integer numbers ranging from 1 to 4. There are 9 instances of each number.

Let the faces of Pyraminx be numbered from 1 to 4, and the puzzle itself be placed on a table with the fourth face looking down. Let the faces with numbers 1, 2 and 3 be listed in counterclockwise order when looking from above.

The first 9 numbers describe the triangles on the first face, by rows from top to bottom, from left to right in each row. The next 9 numbers describe the triangles on the second face in the same way. The next 9 numbers describe the triangles on the third face in the same way. Left and right side are identified when looking directly into the face from the outside.

The last 9 numbers describe the triangles on the fourth face. Let us look at the face in such a way, that the edge connecting it to the third face is at the bottom. Then the triangles are described by rows from top to bottom, from left to right in each row.

The flattened version of the Pyraminx with numbers on the triangles representing the order they are given in the input file is provided on the following picture. The picture shows flattened Pyraminx as viewed from the inner side, outer side down.



Output

If it is impossible to solve the puzzle, print “`Impossible`” on the first line of the output file. In the other case print the sequence of moves that solves the puzzle.

Let the vertices of the tetrahedron be numbered from 1 to 4 in such a way that no vertex is incident to the face with the same number. Each move is identified with its size (small or big), vertex that is in the tetrahedron part which is rotated, and the direction of the rotation (clockwise or counterclockwise). The direction of the rotation is identified with looking at the considered vertex in the direction of the axis of the rotation.

To print the move, print its size (`S` or `B`) followed by its vertex (1, 2, 3 or 4) followed by its direction (`W` for clockwise, `C` for counterclockwise).

You need not find the shortest sequence of moves, but the number of moves in your sequence must not exceed 100 000.

Example

pyraminx.in	pyraminx.out

`1 2 2 2 2 2 2 2 2`
`3 1 1 1 1 1 1 1 1`
`2 3 3 3 3 3 3 3 3`
`4 4 4 4 4 4 4 4 4`
	`S4C`

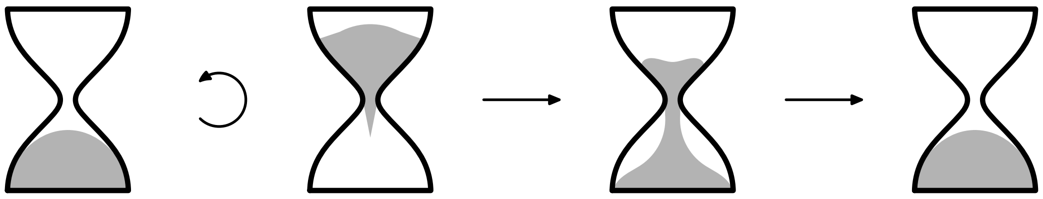
`4 2 2 2 2 2 2 2 2`
`3 1 1 1 1 1 1 1 1`
`2 3 3 3 3 3 3 3 3`
`1 4 4 4 4 4 4 4 4`
	`Impossible`

Problem H. Sand-Glass

Input file:	`sand.in`
Output file:	`sand.out`
Time limit:	1 second
Memory limit:	64 megabytes

Andie has recently found the old sand-glass in his grandfather's box. The sand-glass consists of two equal glass cones, connected to each other with a small hole.

Initially one of the cones contains some sand, and the other one is empty. After flipping the glass, the sand starts to fall down through the hole, and goes completely to the other cone in m minutes.



Now Andie wants to measure time with his sand-glass. But he is not satisfied with m-minute periods, he wants to be able to measure each minute.

Of course, the shape of the the cones is not perfect, and the sand falling through the hole forms a complicated surface. But it does not really worry Andie. He need not measure time very precisely.

So he assumes, that:

- both cones have a form of a perfect cone with height h and radius r;
- the sand initially fills the subcone of the upper cone, with height s;
- each moment the sand fills some subcone of the upper cone;
- the sand falls down through the hole with the constant speed, completely going through in m minutes.

Help him to find the positions at which he should put marks on the cone, so that the sand reached the new mark each minute. The positions should be measured along the generatrix of the cone, from the point where the two cones are connected. The first mark must be placed at the level of the sand in the beginning.

Input

Input file contains four integer numbers: h, r, s and m ($1 \le s < h \le 1000$, $1 \le r \le 1000$, $1 \le m \le 1000$).

Output

Output m real numbers — at which positions should the marks be placed. Your answer must be accurate up to 10^{-6}.

Example

`sand.in`	`sand.out`

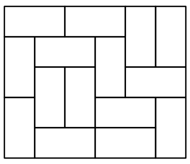
`2 2 1 2`	`1.41421356237309505`
	`1.12246204830937298`

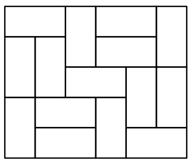
Problem I. Solid Tilings

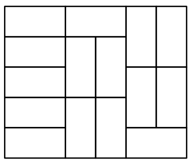
Input file:	`solid.in`
Output file:	`solid.out`
Time limit:	1 second
Memory limit:	64 megabytes

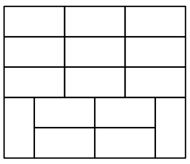
The *Broken Tiles* company's new offer promises its rich clients to pave their rectangular yards with nice 2×1 and 1×2 pavement tiles.

The tiling is called *solid* if it is not possible to split the tiled rectangle by a straight line, not crossing the interior of any tile. For example, on the picture below the tilings (a) and (b) are solid, while the tilings (c) and (d) are not.


(a)


(b)


(c)


(d)

Now the managers of the company wonder, how many different solid tilings exist for an $m \times n$ rectangle. Help them to find that out.

Input

The input file contains m and n ($1 \le m \le 8$, $1 \le n \le 16$).

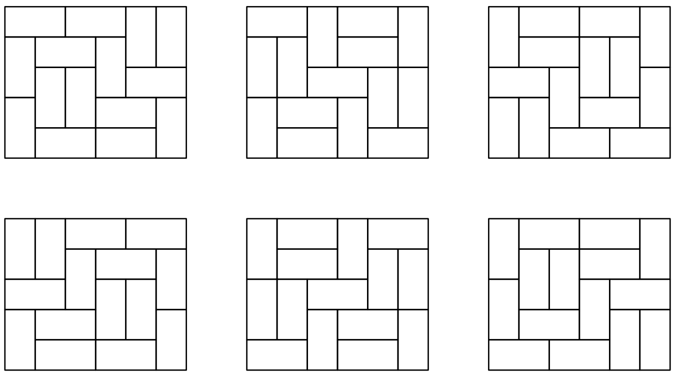
Output

Output one integer number — the number of solid tilings of $m \times n$ rectangle with 2×1 and 1×2 pavement tiles.

Example

`solid.in`	`solid.out`
`2 2`	`0`
`5 6`	`6`

All solid tilings for the 5×6 rectangle are provided on the picture below:



Problem J. Lucky Tickets

Input file:	`tickets.in`
Output file:	`tickets.out`
Time limit:	1 second
Memory limit:	64 megabytes

One airline had a contest among companies using it for business flights. The winner of the contest would get the number of tickets for the flights of the airline. The particular flights could be chosen by the winner, but there is one condition: for each city the company can choose at most one flight from this city, and at most one flight to this city.

Macrohard company has won the contest, and is now planning to use the tickets as a bonus award for its employees. The company has several offices in different cities. Each office was asked to choose the best employee, and nominate it for the award. This employee must choose the city he wishes to make a trip to, and the list of such requests is passed to the head office of the company.

The managers of the company now want to choose the tickets to take as prizes to distribute them among employees. The have a list of pairs x_i, y_i, each of which specifies the request for the employee from the city x_i to go to the city y_i. To satisfy such request, the company must choose a ticket from city x_i to the city y_i. However, there can be no direct flight from x_i to y_i. In such case it is allowed to satisfy a request using two flights to it — from x_i to some city z_i, and from z_i to y_i. Assigning more than two flights to the request is not allowed, because in this case the awarded person would have to spend too much time in flight transfers.

Help employees to find out whether it is possible to satisfy all the requests, and if it is possible, how to do it.

Input

The first line of the input file contains n — the number of requests, and m — the number of available flights ($1 \le n \le 500$, $1 \le m \le 20\,000$).

The following n lines describe requests — each request is a pair of the city names x_i, y_i, separated by comma. All source cities are different, for each request the destination city is different from the source city.

The following m lines describe flights — each line contains the names of the cities it connects, separated by comma. Each flight can be used only once, in either direction. There is at most one flight between a pair of cities.

Each city name consists of at most 40 letters of the English alphabet, numbers and spaces. You must ignore leading and trailing spaces in the city name.

Output

On the first line of the output file print "`YES`" if it is possible to satisfy all the requests, or "`NO`" in the other case.

If all requests can be satisfied, the following n lines must describe the way to do it. The i-th of these lines must contain the number of the flights used to satisfy the i-th request, and the numbers of the used flights.

Flights are numbered starting from one, in the order they are given in the input file.

Example

tickets.in	tickets.out
2 5	YES
St Petersburg, Paris	1 3
Petrozavodsk, London	2 2 5
St Petersburg, Moscow	
Petrozavodsk, Moscow	
Paris, St Petersburg	
Paris, Moscow	
London, Moscow	
2 4	NO
St Petersburg, Paris	
Petrozavodsk, London	
St Petersburg, Moscow	
Petrozavodsk, Moscow	
Paris, Moscow	
London, Moscow	
2 1	NO
Tokyo, Kioto	
Kioto, Tokyo	
Tokyo, Kioto	