Problem A. Appearance Analysis

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

You have taken a photo of the Faculty of Electrical Engineering and Computing "C" building here in Zagreb with its equal sized windows neatly arranged in rows and columns. Some of the windows are painted in curious designs and you are trying to analyze the photo and determine the number of distinct designs.

We represent the photo as a rectangular grid of characters with r rows and c columns. Every window is a rectangular area and all windows have the same dimensions. Each cell in a window is either clear (denoted by the "`.` character) or painted (denoted by the "`+` character). Two windows are considered to be of the *same design* if one can be rotated a multiple of 90 degrees and placed on top of the other so that they perfectly match. When comparing designs, we are not allowed to flip a window inside out.

Windows are regularly aligned in rows and columns with exactly one row of brick cells (denoted by the "`#` character) framing each window. More precisely, there is a single row of "`#`" characters between two consecutive window rows as well as before the first window row and after the last window row. Similarly, there is a single column of "`#`" characters between two consecutive window columns as well as before the first window column and after the last window column. The exact number of window rows and window columns is arbitrary. The window dimensions are also arbitrary. However, a window consists of at least one cell and, again, all windows in the photo have the same dimensions.

Find the number of different window designs in the photo.

Input

The first line contains two integers r and c ($3 \le r, c \le 111$) — number of rows and the number of columns of the photo. Each of the following r lines contains a string consisting of c characters — one row of the photo.

Output

Output a single integer — the number of different window designs in the photo.

Examples

standard input	standard output
11 16	4
################	
#....#++++#+...#	
#....#++.+#+...#	
#....#.++.#++.+#	
#....#....#++++#	
################	
#....#.+..#++++#	
#..++#.+..#++.+#	
#+...#....#.++.#	
#+...#..++#....#	
################	
9 21	4
#####################	
#...+#++++#+...#..+.#	
#..+.#.++.#.+..#..+.#	
#.+..#....#..+.#..+.#	
#####################	
#.+..#....#..+.#.+..#	
#..+.#.++.#.+..#.+..#	
#...+#++++#+...#.+..#	
#####################	

Problem B. Bipartite Blanket

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

In a *bipartite graph*, nodes are partitioned into two disjoint sets A and B such that every edge connects a node from A with a node from B. A *matching* M is a set of edges where no two edges share a common node. We say that a matching M *blankets* a set of nodes V if every node in V is an endpoint of at least one edge in M.

We are given a bipartite graph where each node is assigned a *weight* — a positive integer. Weight of a set of nodes is simply the sum of the weights of the individual nodes.

Given an integer threshold t, find the number of different sets of nodes V such that the weight of V is at least t and V is blanketed by at least one matching M.

Input

The first line contains two integers n and m ($1 \le n, m \le 20$) — the number of nodes in A and B respectively. Let us denote the nodes of A with $a_1, a_2, \ldots, a_n$ and the nodes of B with $b_1, b_2, \ldots, b_m$.

The following n lines contain m characters each that describe the edges of the graph. The j-th character in the i-th line is “`1` if there is an edge between a_i and b_j and “`0` otherwise.

The following line contains n integers $v_1, v_2, \ldots, v_n$ ($1 \le v_k \le 10\,000\,000$) — the weights of the nodes $a_1, a_2, \ldots a_n$. The following line contains m integers $w_1, w_2, \ldots, w_m$ ($1 \le w_k \le 10\,000\,000$) — the weights of the nodes $b_1, b_2, \ldots b_m$.

The following line contains an integer t ($1 \le t \le 400\,000\,000$) — the weight threshold.

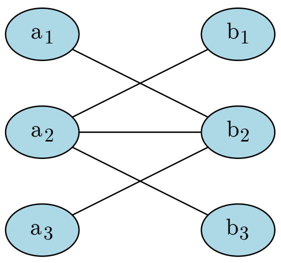
Output

Output the number of sets of nodes whose weight is at least t and are blanketed by at least one matching M.

Example

standard input	standard output
`3 3`	`3`
`010`	
`111`	
`010`	
`1 2 3`	
`8 5 13`	
`21`	
`3 2`	`13`
`01`	
`11`	
`10`	
`1 2 3`	
`4 5`	
`8`	

Note



In the first example above, subset $\{a_1, a_2, b_2, b_3\}$ is blanketed by matching $\{(a_1, b_2), (a_2, b_3)\}$ and has weight 21. Subsets $\{a_3, b_2, b_3\}$ and $\{a_2, a_3, b_2, b_3\}$ are both blanketed by matching $\{(a_2, b_3), (a_3, b_2)\}$, and have weights 21 and 23 respectively. All the other subsets either weigh less than 21 or are not blanketed by any matching. For example, subset $\{a_2, a_3, b_1, b_3\}$ has weight 26, but is not blanketed by any matching, so it's not included in the count.

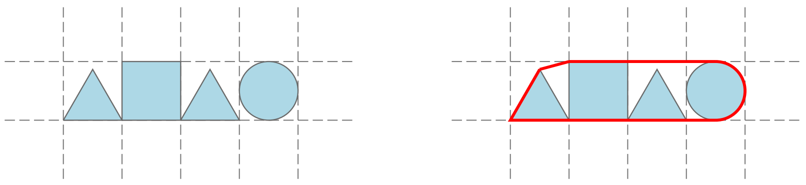
Problem C. Convex Contour

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

A number of geometric shapes are neatly arranged in a rectangular grid. The shapes occupy consecutive cells in a single row with each cell containing exactly one shape. Each shape is either:

- a square perfectly aligned with the grid square,
- a circle inscribed in the grid square,
- or an equilateral triangle with a side corresponding to the bottom side of the grid square.

At the pictures below you can see the shapes from the first example input and their convex contour.



The shapes from the first example input and their convex contour.

Informally, the *convex contour* of an arrangement is the shortest line that encloses all the shapes. Formally, we can define it as the circumference of the convex hull of the union of all shapes.

Given an arrangement of shapes, find the length of its contour.

Input

The first line contains an integer n $(1 \le n \le 20)$ — the number of shapes. The following line contains a string consisting of n characters that describes the shapes in the arrangement left to right. Each character is an uppercase letter “S, “C or “T denoting a square, circle or a triangle respectively.

Output

Output a single floating point number — the length of the contour. The solution will be accepted if the absolute or the relative difference from the judges solution is less than 10^{-6}.

Example

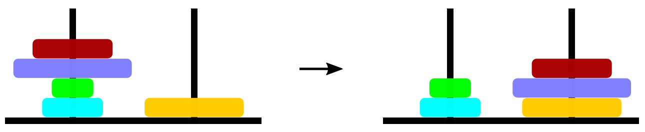
standard input	standard output
4 TSTC	9.088434417
3 SCT	7.50914177324

Problem D. Dancing Disks

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3.5 seconds
Memory limit:	512 mebibytes

Luka has mastered the Towers of Hanoi puzzle and has invented a somewhat similar game with disks and rods. The puzzle consists of n wooden disks of different sizes and 36 rods. The disks are numbered with integers 1 through n in the order of increasing size. The rods are organized in 6 rows numbered 1 through 6 top to bottom and 6 columns numbered 1 through 6 left to right.

An example step:



The puzzle starts with all n disks stacked in arbitrary order on the rod in the upper-left corner. In each step, a player can pick up a stack of one or more disks from the top of a single rod R and transfer them (without reordering) to the top of the rod that is either immediately below R or immediately to the right of R. The goal of the game is to have all the disks stacked on the rod in the lower-right corner neatly ordered by increasing size top to bottom.

Given the initial order of the disks on the rod in the upper-left corner, find any valid sequence of steps that solves the puzzle. You may assume that a solution always exists.

Input

The first line contains an integer n $(1 \le n \le 40\,000)$ — the number of disks. The following line contains a sequence of n different integers $d_1, d_2, \ldots, d_n$ $(1 \le d_k \le n)$ — the initial order of disks, bottom to top, on the rod in the upper-left corner.

Output

Output m lines where m is the number of steps in your solution. The k-th line should contain four tokens r_k, c_k, p_k, n_k, describing the k-th step in your solution. The tokens should be as follows:

- r_k and c_k are integers between 1 and 6 denoting the row number and the column number of the rod we are picking up the disks from,
- p_k is an uppercase letter “`D` or “`R` denoting that we are transferring the disks to the rod directly below or directly to the right respectively,
- n_k is a positive integer denoting the number of disks we are transferring in this step.

All steps have to be valid according to the rules above and solve the puzzle correctly.

Example

standard input	standard output
6 1 6 5 4 3 2	1 1 D 6 2 1 D 6 3 1 D 6 4 1 D 6 5 1 D 6 6 1 R 6 6 2 R 6 6 3 R 6 6 4 R 6 6 5 R 5 6 5 R 1

Note

In the example above, the first 9 steps simply move all the disks, without reordering, to the rod in row 6, column 5 — immediately to the left of the target rod in the lower-right corner. In the following step, the stack of five disks from the top of the rod — (6, 5, 4, 3, 2) bottom to top — are moved to the target rod. Finally, disk 1 is moved to the target rod obtaining the target bottom-to-top order (6, 5, 4, 3, 2, 1).

Problem E. Easy Equation

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

Given an integer k greater than 1, it is possible to prove that there are infinitely many triples of positive integers (a, b, c) that satisfy the following equation:

$$a^2 + b^2 + c^2 = k(ab + bc + ca) + 1.$$

Given positive integers n and k, find n arbitrary triples (a_1, b_1, c_1), (a_2, b_2, c_2), $\ldots$, (a_n, b_n, c_n) that all satisfy the equation. Furthermore, all $3n$ integers $a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n$ should be different positive integers with at most 100 decimal digits each.

Input

The first line contains two integers k and n ($2 \le k \le 1\,000, 1 \le n \le 1\,000$) — the constant k in the equation and the target number of triples.

Output

Output n lines. The i-th line should contain three space separated integers a_i, b_i and c_i with at most 100 digits each — the i-th of the solutions you found.

Examples

standard input	standard output

2 8	1 2 6
	3 10 24
	12 35 88
	15 28 84
	4 5 18
	14 33 90
	40 104 273
	21 60 152

3 3	1 3 12
	8 21 87
	44 165 615

Problem F. Free Figurines

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

A *matryoshka doll* is a set of wooden figurines of increasing sizes that can be nested one inside the other. They are nested by placing a figurine inside a larger figurine which is, in turn, placed inside a yet larger figurine, etc. We may never place more than one figurine *directly* inside another figurine, no matter the sizes of the figurines involved.

We are currently playing with a set of n figurines denoted by integers 1 through n in the order of increasing size. If a figurine a is placed directly inside a figurine b we say that b is a *parent* of a, and if a figurine has no parent we call it *free*. A *configuration* of the whole set can be described by specifying the current parent of each figurine.

We are allowed to perform the following steps while playing:

- Place a free figurine inside a larger free figurine that is currently empty.
- Open a non-empty free figurine and take out the figurine placed directly inside.

Find the minimal number of steps to obtain the given target configuration from the given initial configuration.

Input

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

The following line contains a sequence of n integers $p_1, p_2, \ldots, p_n$ ($0 \le p_k \le n$) describing the initial configuration. The k-th integer p_k is 0 if the figurine k is free, or the parent of figurine k otherwise.

The following line contains a sequence of n integers $q_1, q_2, \ldots, q_n$ ($0 \le q_k \le n$) describing the target configuration in the same format.

You may assume that both configurations are valid: a figurine is always smaller than its parent and no two figurines have the same parent.

Output

Output a single integer — the minimal number of steps.

Examples

standard input	standard output

`7`
`3 5 4 0 7 0 0`
`3 5 0 6 7 0 0`
	`2`

`6`
`2 5 4 0 0 0`
`2 6 4 5 0 0`
	`3`

Problem G. Geohash Grid

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4.5 seconds
Memory limit:	512 mebibytes

Geohash is a procedure of coding map coordinates to scalar values for the purpose of efficient storage and querying of geographical data in databases. In this problem, a *map* is a $2^n \times 2^n$ rectangular grid embedded in a standard coordinate system with the x coordinate growing rightwards and the y coordinate growing upwards. A *map cell* is a unit square aligned with the coordinate axes whose lower-left corner is a point with integer coordinates (x, y) such that $0 \le x, y < 2^n$.

There are a total of 2^{2n} cells in a $2^n \times 2^n$ map. Given a map cell c, its geohash $h(c)$ is a $2n$-bit non-negative integer constructed bit by bit starting from the most significant bit by setting the *viewport* to the entire map and repeating the following two steps n times:

1. We divide the viewport into two equal areas — the left half and the right half. If the cell c is in the left half, the next bit is 0, otherwise the next bit is 1. The new viewport is the area containing the cell c.

2. We divide the viewport into two equal areas — the bottom half and the top half. If the cell c is in the bottom half the next bit is 0, otherwise the next bit is 1. The new viewport is the area containing the cell c.

A *geohash interval* $[a - b]$ is a set of cells whose geohash values are between a and b, both inclusive. Often, it is useful to approximate a map region with a set of geohash intervals. Given a set of cells C, and an integer t, an *optimal t-approximation* of C is a minimal-area region that contains C and can be described as an union of at most t geohash intervals. Formally, it is a set S of at most t geohash intervals such that:

- Each cell of C is contained in at least one interval in S.

- The total number of cells in the union of all intervals in S is minimal possible.

You are given a map region C described as a set of cells in the interior of a polygon whose sides are aligned with the grid. You are also given q target integers $t_1, t_2, \ldots, t_q$. For each t_k find the area of an optimal t_k-approximation of C.

Input

The first line contains an integer n ($1 \le n \le 30$) — the binary logarithm of the map dimensions.

The following line contains an even integer m ($4 \le m \le 200$) — the number of vertices of the polygon. The k-th of the following m lines contains two integers x_k and y_k ($0 \le x_k, y_k \le 2^n$) — the coordinates of one vertex of the polygon. The vertices are given in the counterclockwise order. Each polygon side is either vertical or horizontal. You may assume that the polygon does not intersect or touch itself, or contains consecutive parallel sides.

The following line contains an integer q ($1 \le q \le 100\,000$) — the number of queries. The k-the of the following q lines contains a single integer t_k ($1 \le t_k \le 10^9$) — the k-th query.

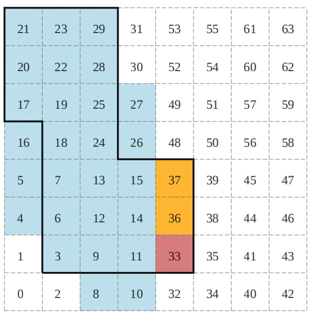
Output

The k-th line should contain the size of the optimal t_k-approximation of the given region.

Example

standard input	standard output
3	32
8	30
1 1	26
5 1	24
5 4	
3 4	
3 8	
0 8	
0 5	
1 5	
4	
2	
3	
5	
7	

Note



In the example above, the intervals $[3 - 29]$, $[33 - 33]$ and $[36 - 37]$ form an optimal 3-approximation of the given region. The total area of the union of the three intervals is 30.

Problem H. Hangar Hurdles

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

You are evaluating constructions plans for a giant new hangar that will house an airplane assembly line. The hangar floor can be represented as a rectangular grid consisting of n rows and n columns where every cell is either empty or blocked. The rows are numbered with integers 1 through n top to bottom, while the columns are numbered with integers 1 through n left to right.

It is important that large crates containing airplane parts can be freely moved between various locations inside the hangar. We can model a crate as a square aligned with grid cells and centered in one of the cells. Therefore, for an odd integer k, a *crate of size* k consists of cells in k consecutive rows and k consecutive columns. The crate can be moved up, down, left or right as long as it fits completely inside the grid and never contains a blocked cell.

You are given q pairs of cells A_k and B_k. For each pair, find the size of the largest crate that can be centered in A_k and then moved across the hangar floor until its centered in B_k.

Input

The first line contains an integer n ($2 \le n \le 1\,000$) — the size of the hangar floor. Each of the following n lines contains a string of exactly n characters describing one row of the floor. The character "`#` denotes a blocked cell while the character "`.` denotes an empty cell.

The following line contains an integer q ($1 \le q \le 300\,000$) — the number of queries. The k-th of the following q lines contains four integers $r_{A_k}, c_{A_k}, r_{B_k}, c_{B_k}$ ($1 \le r_{A_k}, c_{A_k}, r_{B_k}, c_{B_k} \le n$) — the row number and column number of cells A_k and B_k respectively. Cell A_k will be different than the cell B_k. Also, both cells will always be empty.

Output

Output q lines. The k-th line should contain a single integer s_k — the size of the largest crate that can be moved from A_k to B_k. If no crate can be moved from A_k to B_k then s_k should be 0.

Example

standard input	standard output
7 .....#. ...#.#. ....#.. ....### ....#.. #...... .... 5 2 5 5 2 2 5 3 6 2 2 6 3 2 2 6 6 1 1 7 7	1 0 3 1 1

Problem I. Invisible Integers

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

Invisible integers is a simple game where the player tries to guess a hidden sequence of integers 1 through 9 given a number of *hints*. Each hint is a sequence of *distinct* integers generated as follows:

- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.
- Starting from the chosen position we traverse all the integers of the hidden sequence in the chosen direction. We append the traversed integers to the end of the hint while skipping over the integers already in the hint.

Find the length of the shortest hidden sequence consistent with the given hints.

Input

The first line contains an integer n ($1 \le n \le 10$) — the number of hints. Each of the following n lines contains one hint. Each hint is a sequence of at least 1 and at most 9 distinct space-separated integers between 1 and 9 inclusive terminated with the integer 0.

Output

If there is no solution output -1. Otherwise, output a single integer — the length of the shortest hidden sequence consistent with the given hints.

Example

stanard input	standard output

5
1 2 0
3 4 0
1 4 3 0
3 1 4 2 0
1 2 4 3 0
	7

3
1 2 0
2 3 0
3 4 0
	-1

Note

In the first example, (1, 2, 1, 4, 1, 3, 4) is one sequence of minimal length consistent with the given hints:

- Hint (1, 2) can be obtained by starting from the 3rd element and heading left.
- Hint (3, 4) can be obtained by starting from the 6th element and heading right.
- Hint (1, 4, 3) can be obtained by starting from the 3rd element and heading right.
- Hint (3, 1, 4, 2) can be obtained by starting from the 6th element and heading left.
- Hint (1, 2, 4, 3) can be obtained by starting from the 1st element and heading right.

Problem J. Jazz Journey

Input file:	`standard input`
Output file:	`standard output`
Time limit:	6 seconds
Memory limit:	512 mebibytes

Ivan is planning a large European tour with his jazz band. There are a total of n cities in Europe, numbered with integers 1 through n. Ivan is planning d concerts in cities $a_1, a_2, \ldots, a_d$ in that exact order, never having two consecutive concerts in the same city ($a_i \neq a_{i+1}$), possibly visiting some of the cities many times and, finally, ending the tour in the same city where it begun ($a_1 = a_d$).

Ivan always takes a direct flight between cities a_i and a_{i+1}. However, he is trying to be smart with his ticket purchases in order to save money. As you may know, airlines price tickets based on supply and demand and, for example, it may be possible that one-way tickets are more expensive than round trip tickets between same cities.

Generally, there are two kinds of tickets available for purchase:

- *One-way* ticket from the *origin* city a to *destination* city b can be used to fly from a to b once (but not in the opposite direction).
- *Round trip* ticket from the *origin* city a to *destination* city b can be used to fly once from a to b, and once from b to a. The return segment (from b to a) does not need to be used. However, the segments have to be flown in order — it is not allowed for Ivan to use the return segment of a ticket to fly from b to a unless he has used the first segment of that ticket to fly from a to b before.

You are given a list of available airfares, find the least amount of money Ivan needs to spend on tickets to be able to complete his journey. Ivan can purchase an arbitrary number of tickets for each airfare. Once again, Ivan needs to take a direct flight from a_i to a_{i+1} for every $i = 1, 2, \ldots, d-1$. You may assume that is possible to complete the journey using the airfares.

Input

The first line contains two integers n and d ($2 \leq n, d \leq 300\,000$) — the number of cities in Europe and the number of concerts. The following line contains integers $a_1, a_2, \ldots, a_d$ ($1 \leq a_i \leq n$, $a_i \neq a_{i+1}$, $a_1 = a_d$) — the planned tour schedule.

The following line contains an integer m ($3 \leq m \leq 300\,000$) — the number of airfares. The k-th of the following m lines contains four tokens s_k, d_k, t_k, p_k, describing the k-th airfare as follows:

- s_k and d_k ($1 \leq s_k, d_k \leq n$, $s_k \neq d_k$) are the origin and the destination city respectively,
- t_k is an uppercase letter "`O` or "`R` denoting a one-way or round trip ticket respectively,
- p_k ($1 \leq p_k \leq 10^9$) is the ticket price, an integer.

Output

Output the least amount of money necessary to purchase tickets that allow Ivan to complete the planned tour.

Examples

standard input	standard output
2 5 1 2 1 2 1 4 1 2 R 6 1 2 O 3 2 1 O 3 1 2 R 5	10
4 10 1 2 3 1 2 1 3 2 4 1 9 2 4 O 10 1 3 R 1 3 1 R 10 2 3 R 20 1 2 R 10 1 2 O 20 2 3 O 5 3 2 O 5 4 1 O 10	60

Problem K. Key Knocking

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

Goran is recovering from his knee surgery and is experimenting with a smart card used for storing cryptographic keys. In this problem, a *key* is a bitstring of size $3n$ where n is a positive integer. Particular bits of the key are indexed with integers 1 through $3n$ left to right. *Weight* of a key is the number of pairs of different neighbouring bits increased by one. For example, the weight of the key “`000` is 1, while the weight of the key “`011010100` is 7.

He has discovered that he can tamper with the key by sending small jolts of electricity through the circuits of the smart card. In particular, he can reliably perform the following operation: pick two arbitrary neighbouring bits of the key and flip both of them. For example, one operation can change the key “`000` to “`110`.

Given a key of size $3n$ find any sequence of at most n operations that transforms the given key to a key of weight at least $2n$. You may assume that a solution always exists.

Input

The first line contains a string consisting of digits “`0` and “`1` — the initial key. The length of the key is $3n$ where n is a positive integer such that $1 \le n \le 100\,000$.

Output

The first line should contain an integer m where $0 \le m \le n$ — the number of operations in your solution. The following line should contain m integers $a_1, a_2, \ldots, a_m$ describing your solution. The number a_k is the index of the left one of two bits being flipped in the k-th step.

If the initial key already has weight at least $2n$ you may output only the integer 0.

Examples

standard input	standard output

`000000000`	`3`
	`2 5 6`

`111001000111`	`2`
	`3 9`

`010101`	`0`

Problem L. Lost Logic

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

Gustav is reading about *2-satisfiability*, a well known problem of assigning truth values to boolean variables in order to satisfy a list of *constraints* — simple logical formulas involving two variables each.

We are using n variables $x_1, x_2, \ldots, x_n$ that can take on values 0 (false) and 1 (true). A constraint is a formula of the form $a \to b$ where both a and b are possibly negated variables. As usual, $\to$ denotes logical implication: $a \to b$ is 0 only when a is 1 and b is 0. The negation of variable a is denoted by $!a$.

Given an assignment of values to variables, we say that the constraint is *satisfied* if it evaluates to 1. Gustav has constructed a list of constraints and correctly concluded that there are *exactly three* different assignments of values to variables that satisfy all the constraints. Gustav has wrote down all three assignments but has, unfortunately, lost the list of constraints.

Given three assignments of n values to variables, find any list consisting of at most 500 constrains such that the three given assignments are the *only* assignments that satisfy all the constraints.

Input

The first line contains an integer n ($2 \le n \le 50$) — the number of variables. Three lines follow, each describing one assignment. The k-th line contains n space-separated integers $v_1^k, v_2^k, \ldots, v_n^k$, where each v_i^k is either 0 or 1 and denotes the value of the variable x_i in the k-th assignment. All three assignments will be different.

Output

If there is no solution output a single line containing the integer −1.

Otherwise, the first line should contain an integer m where $1 \le m \le 500$ — the number of constraints in your solution. The k-th of the following m lines should contain the k-th constraint. Each constraint should be a string constructed according to the following rules:

- A *variable* is a string of the form "xi where i is an integer between 1 and n inclusive written without leading zeros.
- A *literal* is a string consisting of a variable possibly preceded by the "`!` character.
- A *constraint* is a string of the form "a `->` b where both a and b are literals. The implication sign consists of the "minus" character and the "greater-than" character and there is a single space character both before and after the implication sign.

Examples

standard input	standard output

3
0 0 0
0 1 0
1 0 0
	3
	x1 -> !x2
	x3 -> x1
	x3 -> x2

4
0 0 1 0
1 0 0 0
1 0 1 1
	-1