Problem A. Astronomer's Nightmare

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

Prediction of asteroid trajectories and collisions is a routine task for employees of observatories. Bob has spent a lot of hours observing two asteroids. Now, he knows exactly how they look like and what their velocities are. But after a night without sleep, he isn't able to determine if they are going to collide or if they are a result of some former collision.

Given a description of convex asteroids and their velocities, determine if they have collided or they will collide. You should neglect gravitational field as they are far from all planets. You should also assume that there is no collision in reference time.

Input

The input is divided into two blocks. Each block represents one asteroid. Asteroid is a convex hull of given points. A block starts with a line containing one number $4 \le n \le 5 \cdot 10^4$, the number of points. Each of the lines between 2 and $n+1$ consists of three integers $-10^9 \le x, y, z \le 10^9$. You can assume that the given points are not coplanar (i.e, there exists such four points that do not lie on the same plane). Finally, each block ends with a line consisting of three integers v_x, v_y, v_z, $-2 \cdot 10^6 \le v_x, v_y, v_z \le 2 \cdot 10^6$ describing the velocity of the asteroid.

Output

Print One line containing "`YES`" if asteroids have collided or will collide (i.e, they have a common point at some time). Output "`NO`" in the opposite case.

Example

standard input	standard output

8
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
-1 0 0
8
5 0 0
5 0 1
5 1 0
5 1 1
6 0 0
6 0 1
6 1 0
6 1 1
1 0 0
	YES

8
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
0 1 0
8
5 5 5
5 5 6
5 6 5
5 6 6
6 5 5
6 5 6
6 6 5
6 6 6
0 2 0
	NO

Problem B. Build Them All!

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

One day John was very bored, and thus opened his favorite algorithms book: “Introduction to Algorithms”. The section dedicated to B-Trees saved his day. He took out his notebook and jotted down these notes:

A B-Tree of degree $T \geq 2$ is a rooted tree that satisfies the following properties:

- Every node X has a number $n[X]$ (between $T-1$ and $2T-1$) of comparable keys that are stored in increasing order, that is to say: $Key_1 < Key_2 < \ldots < Key_{n[X]}$.
- Each non-leaf node X has exactly $n[X]+1$ children that also are B-Trees of degree T.
- The keys Key_i separate the ranges of keys stored in each subtree: if K_i is any key stored in the subtree with root C_i, then: $K_1 < Key_1 < K_2 < Key_2 < \ldots < Key_{n[X]} < K_{n[X]+1}$
- All leaves have the same depth, which is the tree's height h.

At this point he started building B-Trees. To build one, he would first populate a random list of integers. Similarly, he would go on and generate the tree's degree T. Finally, he built the B-Tree using each integer from the list eaxactly once, and making sure each node had exactly $T-1$ keys.

If you are bored right now, don't worry, it won't go on for long. Imitate John and build B-Trees in the same fashion. Of course, only if it's possible.

Input

The first line contains an integer indicating the number of tests cases, which is at most 50. Each test case consists of two lines.

The first line contains two integers T $(2 \leq T \leq 7)$ and L $(1 \leq L \leq 2 \cdot 10^4)$ indicating, respectively, the degree of the B-Tree and the size of the list. The second line contains exactly L integers, each between 1 to $2 \cdot 10^4$, separated by single spaces, the elements of the list. It is guaranteed that all elements of the list are unique.

Output

For each test case, output one line containing “`Case #i:`”, where i is the case number (starting from 1) . Then, if it is impossible to build a B-Tree according to the stated restrictions, output a second line containing “`INVALID`” (without the quotes).

Otherwise, print h lines (h is the height of the B-Tree). The i-th level nodes must be printed in ascending order of keys on the i-th of these lines. The keys of each node will be separated by single spaces. You must append a semicolon (‘`;`’) to the right of the rightmost key of every node.

Examples

standard input	standard output

```
3
2 7
4 6 8 9 34 15 67
3 8
3 6 1 2 4 5 7 8
2 4
7 6 16 5
```
	```
	Case #1:
	9;
	6; 34;
	4; 8; 15; 67;
	Case #2:
	3 6;
	1 2; 4 5; 7 8;
	Case #3:
	INVALID
	```

Problem C. Count The Rebuildings

Input file:	`standard input`
Output file:	`standard output`
Time limit:	0.5 seconds
Memory limit:	512 mebibytes

There is a unique way to build a cartesian tree on a set of points (with pairwise distinct x-coordinates and pairwise distinct y-coordinates). A boy throws n points into a 1×1 square randomly and inserts them into the cartesian tree (which is initially empty). We will say that a *rebuilding* took place if the newly-inserted vertex is not a leaf in the new tree. You have to count the expected number of rebuildings.

Since the probability of some two points having same x's or same y's is zero, we can assume that all of them are distinct.

Input

The only line of input contains one integer n ($1 \leq n \leq 10^9$).

Output

Print the answer to the problem with absolute or relative error at most 10^{-9}.

Examples

standard input	standard output
`2`	`0.5`
`5`	`2.23888888888`

Note

Cartesian tree is a data structure which unites a binary search tree and a binary heap.

Strictly speaking, a cartesian tree is a binary tree such that each of its nodes stores a pair (x, y), with x being the *key*, and y being the *priority*. The cartesian tree must be a binary search tree by x's and a binary heap by y's. If we assume that all x's and all y's are distinct, we conclude that for an element (x_0, y_0) all of its left descendants have $x < x_0$, all of its right descendants have $x > x_0$, and all its descendants must have $y < y_0$. (Definition is translated from ITMO Wikipages).

Problem D. Diophantine

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

Given prime p and two integers a and m. Check if the equation $n^n + n^m \mod p = a$ have at least one positive integer solution.

Input

The first line contains the number of test cases d ($1 \le d \le 50$). Each of the following d lines contains space-separated integers p, a, and m ($2 \le p < 10^9$, $0 \le a < p$, $1 \le m \le 20$; $m < p$). The number p is prime.

Output

Output one line for each test case. If there exists a positive integer $n < 10^{18}$ such that $n^n + n^m \mod p = a$, print "`YES`", a space, and any of the numbers n satisfying these constraints. Otherwise print "`NO`".

Example

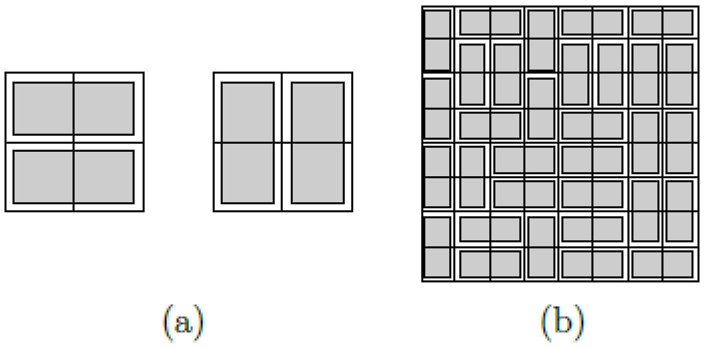
standard input	standard output

2
11 3 1
11 8 2
	YES 567
	YES 2

Problem E. Experimental Coverings

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

A question we may ask ourselves if we have a lot of free time (or if we are mathematicians, which is practically the same thing) is the following: given a board of $N \times N$ square cells, in how many ways can we cover it with dominoes so that the pieces do not overlap and there are no cells left uncovered? Dominoes are rectangles composed of two adjacent square cells, and can be either horizontal (1×2) or vertical (2×1). For example, there are only two ways to cover a 2×2 board, but there are 12,988,816 ways to cover an 8×8 board.



At picture above shown two ways to cover a 2×2 board with horizontal dominoes and vertical dominoes and a possible covering of an 8×8 board.

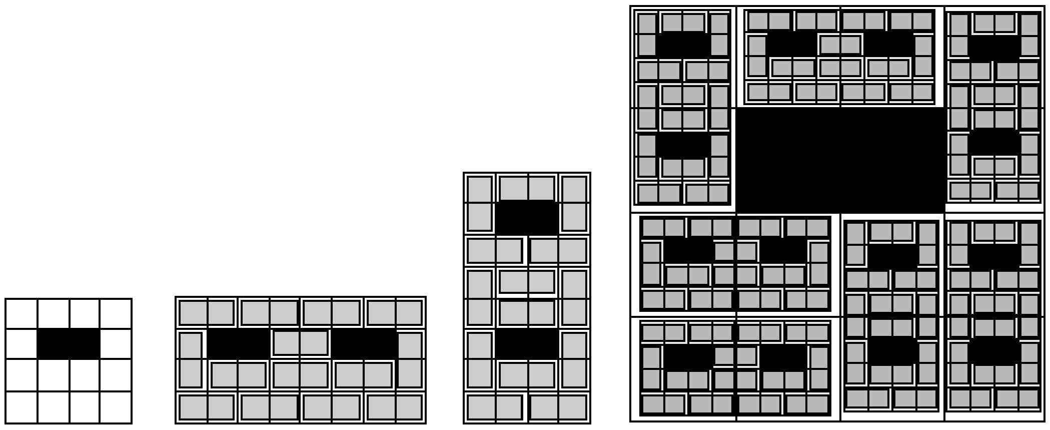
For those with even more free time, there is a variant of this question in which we consider a board with some of its cells occupied, so that we should count the number of coverings in which the domino pieces cover only the unoccupied cells without overlapping or leaving holes. For example, there are two ways to cover a 3×3 board in which the central cell is occupied, and 75272 ways to do so in a 7×7 board.

In this problem we will turn our attention to a variant of this variant of the original question, specifically thought for those of you with too much free time. We will again consider an $N \times N$ board in which some cells can be occupied, but we now want to cover it using fractal dominoes of order K.

Given an $N \times N$ board, a *fractal domino* of order $K = 0$ is a regular domino piece, i.e. a rectangle of 1×2 or 2×1 cells. A fractal domino piece of order $K > 0$ consists of a regular domino in which each cell is a copy of the given $N \times N$ board, and the piece as a whole is covered by fractal dominoes of order $K - 1$.

Note that in general for $K > 1$ there can be more than one fractal domino piece of order $K - 1$ of each type (horizontal or vertical). In this case, we assume that when using a domino piece it is chosen with uniform probability among all possible fractal domino pieces of the same type and order. In a similar fashion, if there is more than one possible covering for a given board or fractal domino, we will assume all of them are equally probable.

When covering a board with fractal dominoes of order K one will use a certain amount of pieces of order K, a larger amount of pieces of order $K - 1$, an even larger amount of pieces of order $K - 2$, and so on. This will go on up to the point where one uses a possibly huge amount of pieces of order zero, i.e. regular domino pieces with nothing inside. For example, in Figure 2d the board is covered with seven fractal dominoes of order one and 98 fractal dominoes of order zero.



(a) A 4×4 board with two occupied cells;

(b) one of the 89 horizontal dominoes of order one for this board;

(c) one of the 52 vertical dominoes of order one for the same board;

(d) a covering of the board in (a) with the fractal dominoes of order one in (b) and (c).

The task in this problem is to calculate the expected proportion of fractal dominoes of order zero (i.e. regular pieces) that are vertical, if we assume that all coverings of a given board with fractal dominoes of a given order K are equally probable.

Input

The first line of the input contains two integers N and K, representing the size of the board and the order of the fractal dominoes that we want to use to cover it, respectively ($2 \le N \le 8$ and $0 \le K \le 10^9$). The following N lines contain N numbers each, being the j-th number in the i-th line a 1 if the cell in row i and column j of the board is occupied, and 0 otherwise. The given board is not completely occupied, and it is always possible to cover it using dominoes.

Output

Print a number representing the expected proportion of fractal dominoes of order zero that are vertical, when we consider that all coverings of the given board with fractal dominoes of order K are equally probable with absolute error 10^{-5} or better.

Examples

standard input	standard output

4 2
0 0 0 0
0 1 1 0
0 0 0 0
0 0 0 0
	0.5075

Problem F. Fantastic Mountains

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

The area of the Fantastic Mountains can be imagined as a rectangle in a plane with cartesian coordinate system. This rectangle has two opposite vertices in points $(0, 0)$ and (w, h), where w and h are positive integers. There are n major peaks in the mountains, each of which is located in one grid point of the rectangle (a grid point is a point having integer coordinates). More and more tourists discover the beauty of Great Bytean Mountains and everyone would like to build a house there. The rules of Fantastic Mountain Park are very strict though: in each grid point of the rectangle at most one house can be built and moreover there cannot be any house on the peak of any mountain. So there are $(w + 1) \cdot (h + 1) - n$ possible locations of houses.

Some of the locations are considered better than other. We say that a grid point (x, y) has a northern neighbour if there is a mountain peak in some point $(x, y + d)$, where d is a positive integer. Similarly, southern, eastern and western neighbours could be defined. In this way, every grid point that is not a mountain peak may have between 0 and 4 neighbours — the more neighbours, the better is the location (because of a better view of the mountains). The director of Fantastic Mountain Park would like to know the maximal profit that can be achieved by selling locations to tourists. Help him and count how many grid points in the mountains (excluding mountain peaks) have 0, 1, 2, 3 and 4 neighbours.

Write a program which:

- reads the description of Fantastic Mountains from the standard input,
- counts the numbers of grid points, having 0, 1, 2, 3 and 4 neighbours,
- writes the result to the standard output.

Input

The first line of input contains three integers w, h and n ($1 \le w, h \le 10^9$, $1 \le n \le 5 \cdot 10^5$), separated by single spaces. The following n lines contain the descriptions of locations of mountain peaks in the park. Each of them contains two integers x and y ($0 \le x \le w$, $0 \le y \le h$), separated by a single space. All peaks are in distinct locations.

Output

The first and only line of output should contain 5 integers, separated by single spaces and denoting the numbers of grid points (excluding peaks), having exactly 0, 1, 2, 3 and 4 neighbours.

Example

standard input	standard output

4 3 6
0 3
2 3
2 1
0 1
3 2
1 2
	1 7 2 3 1

Problem G. Guard's Practice

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

A group of M Swiss guards is performing the shooting practice corresponding for the month. There are N targets in the field and the guards are placed in different positions on a straight line.

The guards have a software that lets them know the trajectory of each shot fired, defining it with two points in the 2D plane, but the software does not give them all the information they need. For now they are only interested in you to develop a new functionality; they desire to know if the projectile goes through at least one target or passes between any two of them.

You should know that a guard never shoots backwards from the line of fire, because this may create a slightly unpleasant situation for the chiefs that are behind. The maximum number of targets that are aligned is always lower than N. Furthermore, there are no two targets placed in the same location and there are no targets behind the line of fire.

Are you ready to solve this little problem? I hope so!

Input

The first line of input contains two integers $5 \le N \le 10^5$, $1 \le M \le 10^5$. In each of the following N lines, two integers X_i, Y_i representing the position of the i-th target on the field. Then M lines follow, each with four integers X_{a_j}, Y_{a_j}, X_{b_j}, Y_{b_j}, specifying the position of j-th guard with the coordinates (X_{a_j}, Y_{a_j}) and the coordinates (X_{b_j}, Y_{b_j}) for defining the trajectory, e.g. the projectile is fired from (X_{a_j}, Y_{a_j}) and is directed in the direction of (X_{b_j}, Y_{b_j}), continuing its trajectory once it passes through this point until goes through the shooting range completely without changing its direction. In each line of the input the numbers are separated by a single space. $(-10^7 \le X_i, Y_i, X_{a_j}, Y_{a_j}, X_{b_j}, Y_{b_j} \le 10^7)$.

You may assume that there are no two guards at the same point, that all guards are placed on the one straight line, all the targets are on the one side from that line and all trajectories are leading to same side of that line where the targets placed.

Output

The output consists of M lines. In the j-th line will be given the information needed by the j-th guard. If the projectile fired by the j-th guard breaks down at least one objective or passes between any two, you must outputting "YES"; otherwise, you must to show "NO", both without quotes.

Example

standard input	standard output

6 3
0 7
-2 6
0 5
2 5
-2 4
-1 3
-2 1 2 5
1 1 -1 4
4 1 7 3
	YES
	YES
	NO

Problem H. Hardware For Cookies

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

One of the most difficult stages to automate in an industrial cookie production line is the one corresponding to packing. The goal is to design an apparatus capable of counting a precise number G of cookies to be put in a package, and the challenge resides in the fact that the cookies may have markedly different shapes, e.g. of various animals.

In the cookie factory where you work the design team has been unable to overcome these difficulties. The best design they have achieved corresponds to a counting machine which, when activated, can select C_i cookies with probability P_i for $i = 1, 2, \ldots, K$. Having already spent the entire budget allocated to this task, you will have to find a way to make something useful out of this.

Fortunately, you have come up with the following idea to save the project. You shall build an automated packager using N counting machines such as the one described above, along with a special regulating software that will perform the following process iteratively. Before starting the first iteration the N counting machines are activated, each one selecting certain number of cookies. In each iteration, the software will choose a non-empty subset S of the N machines such that the sum of the selected cookies is as close as possible to the desired value G (in the sense that the absolute value of the difference between this number and G is minimum). If there is more than one subset satisfying this condition, the software will choose as S any of them with uniform probability. The cookies selected by the counting machines in S will then be removed and packed together. Finally, each of the machines in S will be activated again in order for it to select certain number of cookies, while the counting machines not in S remain unaltered (that is, with the number of cookies they selected in the previous iteration). This process will be repeated until the desired number M of packages has been obtained.

For example, let's assume there are $N = 3$ counting machines, which we will call m_1, m_2 and m_3, having each of them the possibility to select $C_1 = 1$ or $C_2 = 2$ cookies with probability $P_1 = P_2 = 0.5$. If we want to produce $M = 2$ packages with $G = 5$ cookies per package, a possible turn of events is the following. Before starting, the three machines select $C_2 = 2$ cookies each (this will happen with probability $0.5 \cdot 0.5 \cdot 0.5 = 0.125$). In the first iteration, the software can then choose among the subsets $\{m_1, m_2, m_3\}$, $\{m_1, m_2\}$, $\{m_1, m_3\}$ and $\{m_2, m_3\}$, each with probability 0.25 because they all have a difference of one cookie with the desired number $G = 5$. Supposing that the subset $S = \{m_2, m_3\}$ is chosen, then the four cookies selected by machines m_2 and m_3 will be packed together. These machines will then be activated again, resulting for example in m_2 selecting $C_1 = 1$ cookie and m_3 selecting $C_2 = 2$ cookies (which will occur with probability 0.25). At this point the first iteration ends, the counting machines m_1, m_2 and m_3 having selected 2, 1 and 2 cookies respectively. In the second iteration, the software must necessarily choose the subset $S = \{m_1, m_2, m_3\}$ as it contains exactly five cookies, which will therefore be packed together. Finally, the three counting machines will be activated once more and the process will end, having produced the $M = 2$ desired packages. Note that the net probability for the process here described is $1/128$, and that the average number of cookies per package is in this case 4.5 (because two packages were produced, one of them having four and the other five cookies).

Your boss is not completely convinced that this system will work, so he requires from you some concrete proof of concept. To convince him, it will suffice to calculate the expected number of cookies per package after producing M packages consecutively, if you assume that the N counting machines always select cookies according to the given probabilities.

Input

The first line of input contains four integers N, K, G and M. The value N represents the number of counting machines to use ($1 \le N \le 4$), K represents the number of possible quantities of cookies each counting machine can select ($1 \le K \le 6$), G represents the desired number of cookies per package

$(1 \leq G \leq 100)$, and M represents the total number of packages to produce $(1 \leq M \leq 10^7)$. Each of the following K lines contains an integer C_i and a rational P_i, indicating that the counting machines will select C_i cookies with probability P_i $(1 \leq C_i \leq 100$ and $0 < P_i \leq 1$ for $i = 1, 2, \ldots, K$, being P_i given with exactly two digits after the decimal marker). Note that $C_i \neq C_j$ for $i \neq j$ and that sum of all P_i is equal to 1.

Output

Print one the expected number of cookies per package after having produced M packages as described in the problem statement with absolute error 10^{-6} or better.

Example

standard input	standard output

`3 2 5 1`
`1 0.50`
`2 0.50`
	`4.312500`

`3 2 5 2`
`1 0.50`
`2 0.50`
	`4.327148`

Problem I. Interesting Matching

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

For given two sets S and T of numbers, let an *interesting matching* of S and T be a set of pairs (s, t), $s \in S$, $t \in T$ such that each number in S is paired with at least one number in T and each number in T is paired with at least one number in S. For example, the following is an interesting matching of two sets $S = \{2, 8, 9, 10, 11\}$, $T = \{0, 3, 4, 6, 7, 11\}$:

$$M_1 = \{(2, 0), (2, 3), (2, 4), (8, 6), (9, 7), (10, 11), (11, 11)\}$$

Whereas the following is not a matching of S and T,

$$M_2 = \{(2, 0), (8, 3), (9, 4), (10, 6), (11, 7)\}$$

since the number 11 in set T is not paired with any number in S.

For a pair (a, b) in an interesting matching, the cost of the pair is defined to be the absolute value of the difference between a and b. The cost of a matching is the sum of the cost of all pairs in the interesting matching. For example, the cost of the matching M_1 is 10.

Given two sets of numbers, compute the minimum cost interesting matching of the two sets. For example, the matching M_1 is a minimum cost interesting matching of the given two sets S and T.

Input

Input consists of three lines. The first line contains two integers. The first integer, n_1, is the number of integers in the first set, and the second integer, n_2, is the number of integers in the second set, where $1 \le n_1, n_2 \le 5 \cdot 10^5$. The second line of each test case contains n_1 integers for the first set and the third line contains n_2 integers for the second set. The integers in each set are arranged in increasing order. All integers in the sets are between 0 and 10^9, inclusively.

Output

Print one integer — the cost of the minimum cost interesting matching of the two sets of integers.

Example

standard input	standard output

5 6
2 8 9 10 11
0 3 4 6 7 11
	10

Problem J. Junctions, Roads and Racing

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

Bytesburg plans to organize the street race of Formula-1 like Grand Prix of Monaco in Monte-Carlo.

The road network of the city consists of N junctions and M bidirectional roads. The junctions are numbered by sequential integers between 1 and N. Any road is connecting two distinct junctions. Any road is either plain road or the highway. It is known that any two cities are connected directly or indirectly by highways.

Track for the race must conform to next conditions:

- start and finish of the track are at the same junction;
- for any road cars will use this road at most once (but its ok to visit same junction several times);
- all highways must be used as part of the track.

Check if it is possible to choose the track, and, if yes, print maximum number of roads it may consist of.

Input

First line of the input contains two integers N $(1 \le N \le 29)$ and M $(1 \le M \le N(N-1)/2)$. Each of next M lines describes one road and contains three integers A, B and T $(1 \le A, B \le N)$. A and B are junctions, connected by this road, and T is the road type: if $T = 0$, then it is plain road, otherwise it is highway. You may assume that two cities are connected with at most one road and that no road connects the city to itself.

Output

If it is impossible to choose the track, print -1. Otherwise print maximum number of roads in the track.

Example

standard input	standard output

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

4 6
1 2 1
2 3 1
3 4 1
4 1 1
1 3 0
2 4 0
	4

Problem K. Keep The Angle Maximal

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

There are n points $A_1, \ldots, A_n$ in the plane such that no three of them lie on a common line. For each pair (i, j) $(1 \le i < j \le n)$ you have to find such index k (different from i and j) that the angle $A_i A_k A_j$ is the largest possible.

Input

The first line contains one integer n $(3 \le n \le 1000)$.

i-th of the next n lines contains two integer x_i and y_i — coordinates of point A_i $(-10^9 \le x_i, y_i \le 10^9)$.

It is guaranteed that all points are distinct and no three points lie on a common line.

It is also guaranteed that all tests except for the sample cases were constructed in the following manner: jury chooses preliminary positions of the points, then each coordinate of each point is increased by a random number between 0 and 1000. If the resulting test is invalid, the generation starts over.

Output

Print $n - 1$ line.

i-th string should contain i integers. j-th number of i-th string should be the answer for the pair $(j, i+1)$.

If there are several valid answers, print any of them.

Examples

standard input	standard output

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

Problem L. Look At The Sign

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

There an $n \times n$ matrix A. n rectangular submatrices of A are specified. We consider all ways to choose one element from each rectangle. For each option, we place 1 in each of the chosen elements, and 0 in all other elements, and compute the determinant of the resulting matrix. Let S be the sum of all determinants obtained this way. You have to determine the *sign* of S.

The determinant of the $n \times n$ matrix B is defined as $\sum\limits_{j=1}^{n} (-1)^{i+j} B_{ij} M_{ij}$, where B_{ij} is the element from i-th row and j-th colimn and M_{ij} is the determinant of the submatrix obtained by removing the i-th row and the j-th column of B; the determinant of 1×1 matrix is equal to its only element.

Input

First line of the input consists of one integer N ($1 \le n \le 10^6$). Each of next n lines contains four integers x_1, x_2, y_1 and y_2 ($1 \le x_1, x_2, y_1, y_2 \le n$) — top left and bottom right corners of the respective submatrix. Note that the submatrices may overlap.

Output

Is the sum is positive, print 1, if it is negative, print -1, if it is equal to zero — print 0.

Example

standard input	standard output

3
1 1 1 1
2 2 3 3
3 3 2 2
	-1

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