Problem 2. Chess

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

Given is a toroidal three-dimensional chessboard $n \times n \times n$ ($2 \leqslant n \leqslant 4$).

The cells have coordinates (x, y, z) ($0 \leqslant x, y, z < n$). Cell (x, y, z) is adjacent to cells $((x \pm 1) \bmod n,\ y,\ z)$, $(x,\ (y \pm 1) \bmod n,\ z)$ and $(x,\ y,\ (z \pm 1) \bmod n)$. Two distinct cells (x_1, y_1, z_1) and (x_2, y_2, z_2) are said to be *corner-neighbours* iff the maximum of "mod-distances" along x, y and z axes is 1. Here, the "mod-distance" between two integers a and b is the value $\min(|a - b|,\ n - |a - b|)$.

There are k ($0 \leqslant k \leqslant 4$) aggressive black kings living on that chessboard, and they want to take the non-aggressive white king. Black and white move in turn. During a single move, player should move a single king of his colour to some corner-neighbouring cell. Exactly one king moves during a single move. Kings can take each other. This is done by moving to the cell where the other king is placed. One is only allowed to take king of the opposite colour. The king which is taken does not exist: it can't move or take other kings, and it doesn't occupy any cell. No two pieces can occupy the same cell.

You are given the initial coordinates of the pieces. The white king is the first to move. The question is: can the black kings manage to take him? If the answer is positive, you also have to find out the minimal number of moves to achieve that goal. The white king will resist this, so he'll try to get taken as late as possible.

Input

Input consists of one or more test cases.

Each test case starts with two integers n ($2 \leqslant n \leqslant 4$, the size of the chessboard) and k ($0 \leqslant k \leqslant 4$, the number of aggressive black kings). After that, $(k + 1) \cdot 3$ integers 0 through $n - 1$ follow: the coordinates of $k + 1$ kings (first 1 white king, then k black ones). Initially, all pieces occupy distinct cells.

Input will be terminated with a test case with $n = k = 0$ which should not be processed. There are no more than 10^4 tests.

Each integer in the input is followed by a nonempty sequence of spaces and/or newline characters.

Output

For each test case, write "`YES`" if the black kings manage to take the white king, and "`NO`" otherwise. In the first case, you should also write the minimal number of moves required if both players move optimally. You only have to count the black player's moves.

Adhere to the sample output format below as close as possible.

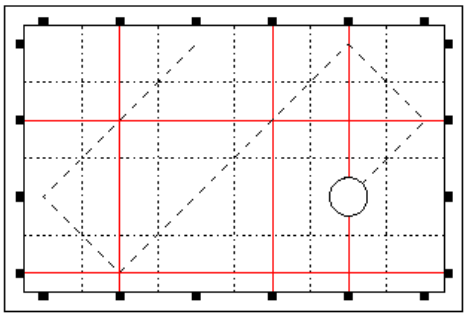
Example

checkmate.in	checkmate.out
2 4 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 4 4 2 2 2 0 0 0 0 0 1 0 1 0 0 1 1 0 0	Case #1: YES 1 Case #2: YES 4

Problem 3. Laser Billiards

Input file:	`billiards.in`
Output file:	`billiards.out`
Time limit:	8 seconds
Memory limit:	256 Mebibytes

Recently, a new billiards variation was invented in Byteland, named "laser billiards". The table for this game is n decimeters long and m decimeters wide. For the purposes of explanation, we will refer to length as horizontal direction and width as vertical direction, as if the table was drawn on a two-dimensional plane with sides parallel to coordinate axes and corners at points $(0, 0)$, $(n, 0)$, $(0, m)$ and (n, m). Along the edges of the table there is a barrier of width $1/4$ decimeters. There are also $n+m$ laser-receiver pairs mounted in the barriers in such a way that n vertical and m horizontal laser beams cover the table. The i-th ($1 \leqslant i \leqslant n$) vertical beam crosses the j-th ($1 \leqslant j \leqslant m$) horizontal beam at point $(i - 1/2, j - 1/2)$. Before the game, some laser beams are turned on, others are turned off. During the game, the state of laser beams does not change.



A ball for laser billiards has diameter $1/2$ decimeters. For each moment of time when a ball crosses one or more laser beams, the player is awarded one point. The next point can be awarded for the intersection of the same laser beam only if the laser beam and the ball weren't overlapping for some time after the previous intersection.

At the start of the game, the ball is positioned at point $(x - 1/2, y - 1/2)$. After that, it starts moving uniformly with velocity (x_v, y_v) (in decimeters per second) for t seconds. The collisions of the ball with the barrier are elastic. The effect of friction is negligible.

Your task is to calculate the total number of points awarded to the player. The starting and ending moments of time are also counted, e. g. if the ball intersects one or more laser beams in the initial position, the player is awarded one point at time 0.

Input

The first line of input contains two integers n and m, the dimensions of the table ($3 \leqslant n, m \leqslant 10^5$).

The second line contains n characters each of which is either '`0`' or '`1`'. This line describes the state of vertical laser beams: if i-th character is '`0`', i-th vertical laser is turned off; if it is '`1`', the laser is turned on.

The third line contains m characters each of which is either '`0`' or '`1`'. This line describes the state of horizontal laser beams in a similar fashion: if j-th character is '`0`', j-th horizontal laser is turned off; if it is '`1`', the laser is turned on.

The fourth line contains an integer k, the number of test cases ($1 \leqslant k \leqslant 10^4$). Each of the next k lines contains five integers x, y, x_v, y_v and t which are the coordinates of the initial position of the ball, the coordinates of velocity and the total time ($1 < x < m$, $1 < y < n$, $x_v, y_v \in \{-1, 1\}$, $1 \leqslant t \leqslant 10^9$).

Output

Output k lines. Line i should contain one integer: the total number of points awarded to the player in i-th test case.

Example

billiards.in	billiards.out
4 6 1010 010110 1 5 2 1 1 8	6
3 3 011 100 2 2 2 -1 -1 68 2 2 1 1 76	69 77

Problem 5. Sequences

Input file:	`seq.in`
Output file:	`seq.out`
Time limit:	4 seconds
Memory limit:	64 Mebibytes

You are given a sequence of integers A: $a_1, a_2, \ldots, a_n$.

Let us call a sequence of indices $c_1, c_2, \ldots, c_p$ where $1 \leqslant c_i \leqslant n$ *lowering* if for subsequence $a_{c_1}, a_{c_2}, \ldots, a_{c_p}$ the inequality $a_{c_1} > a_{c_2} > \ldots > a_{c_p}$ holds.

A «lowering» sequence of indices $c_1, c_2, \ldots, c_p$ is lexicographically smaller than another «lowering» sequence $d_1, d_2, \ldots, d_p$ if there exists $k \in [1, p]$ with the following properties: $c_i = d_i$ for all $i \in [1, k-1]$ and $c_k < d_k$. Using this rule one can sort lexicographically (in increasing order) all «lowering» sequences of fixed length p built upon given sequence A.

Your task is to find k-th «lowering» sequence in the sorted list. Numeration in list is 1-based.

Input

The first line of the input file contains three integer numbers n, p and q — the length of the sequence A, the length of the «lowering» sequences considered and the number of queries correspondingly ($1 \leqslant n, q \leqslant 10^5$, $1 \leqslant p \leqslant 10$). The second line contains n integers a_i ($-10^9 \leqslant a_i \leqslant 10^9$) — the elements of sequence A.

The next q lines contain queries: j-th line contains integer k_j ($1 \leqslant k_j \leqslant 10^{18}$) — the number of the «lowering» sequence you need to find.

Output

Output a lines. i-th line should contain the answer for the i-th query — k_i-th «lowering» sequence in the order of increasing indices (p integers from 1 to n) or -1 in case it doesn't exist.

Example

seq.in	seq.out
5 3 3 -1 6 5 2 1 1 5 3	2 3 4 -1 2 4 5

Problem 7. Tables

Input file:	`tables.in`
Output file:	`tables.out`
Time limit:	10 seconds
Memory limit:	256 Mebibytes

The Individual Programming Championship of Byteland consists of k rounds. Each of the n participants takes part in each round. After a round, participants are ranked 1 through n based on their relative performance in that round (ties are considered impossible). Thus the results table of each round can be viewed as a permutation of numbers $1, 2, \ldots, n$: first goes the winner's number, then the number of second place finisher, and so on.

Let the *distance* between results tables $p_1, p_2, \ldots, p_n$ and $q_1, q_2, \ldots, q_n$ be the sum

$$D(p,q) = \sum_{i=1}^{n} \min(|p'_i - q'_i|, 8)$$

where p' and q' are inverse permutations of p and q, respectively: p'_1 is the place of first participant in results table p, p'_2 is the place of second participant in p, and so on; the same goes for q'.

The Championship sponsors proposed to make the final results table $r_1, r_2, \ldots, r_n$ in such a way that the sum of k distances between r and the results of each round is the least possible. Your task is to calculate that sum.

Input

The first line of input contains two integers n and k, the number of participants and the number of rounds ($2 \leqslant n \leqslant 5000$, $2 \leqslant k \leqslant 3$). Each of the next k lines contains the results of that particular round and consists of n integers which form a permutation of numbers 1, 2, ..., n. The first element of the permutation is the number of the first place, the second one is the number of the second place, and so on.

Output

Output one integer: the minimal possible sum of distances from the final results table to the results tables of each of the k rounds.

Example

tables.in	tables.out
5 2 5 2 4 3 1 2 4 1 3 5	8

Problem 11. Dirty Dishes

Input file:	`dishes.in`
Output file:	`dishes.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

They say there are three things that one cannot have enough of looking at: they are the fire burning, the water flowing and others working. Jack and Jill have been married for several years but Jack never gets tired of watching Jill cleaning the kitchen in her swift and neat way!

As Jill is cleaning the kitchen, she piles up the dirty dishes by the sink. Each newly found dirty dish is put on the top of the pile and when Jill wants to wash the dishes, she takes a dish from the top of the pile.

Jack has been watching his wife attentively and each time she added a new dirty c-colored dish to the pile, Jack made a note in his notebook that a c-colored dish had been added to the pile. Similarly, when Jill took a c-colored dish from the top of the pile, Jack noted that a c-colored dish had been taken from the pile. From time to time Jack would leave to get more popcorn and then he would probably miss some of Jill's actions. In this case he wrote an asterisk "`*`" in his notebook.

Next day, as Jack was scanning through the notes, he got interested: what is the least number of dirty dishes that could be on the kitchen before the cleaning? Note that Jill doesn't arrange the dishes in more than one pile. Jill only puts the dishes to the top of the pile and only takes them from the top of the pile. Before the cleaning and after it the pile is empty.

Input

The only input line contains the notes from Jack's notebook. Using Latin alphabet, he wrote a lowercase letter if Jill added a dish of the corresponding color to the pile. Also, Jack wrote an uppercase letter if Jill took a dish of the corresponding color from the pile. For example, the string "`afFaAA`" represents the following actions:

- Jill adds an a-colored dish to the pile,
- Jill adds an f-colored dish to the pile,
- Jill takes an f-colored dish from the pile,
- Jill adds an a-colored dish to the pile,
- Jill takes an a-colored dish from the pile,
- Jill takes an a-colored dish from the pile.

An "`*`" represents the periods of time when Jack went to get more popcorn and could have missed one or more Jill's actions. It is guaranteed that Jack went to get more popcorn no more than five times. So the number of characters "`*`" doesn't exceed 5.

The given string consists of lowercase and uppercase Latin letters and asterisks "`*`". It is not empty and contains no more than 2500 characters. The input contains at least one letter.

Output

On the first line of the input file print the single number — the least possible number of dirty dishes on the kitchen before the cleaning. Print `-1` if Jack's notes surely have a mistake and there's no solution.

Examples

dishes.in	dishes.out
ab*bB	3
afFaAA	3
**bbB*Da*	4
a**b	-1

Note

An example of a sequence of actions for the first sample is abBAbB.

In the second sample Jack never left the room.

In the third sample he left the room four times, two of them one after another. An example of Jill's action succession is like that: "bbBdDaAB".

The fourth sample illustrates no solution.

Problem 13. Dining Room

Input file:	`dinner.in`
Output file:	`dinner.out`
Time limit:	3 seconds
Memory limit:	256 mebibytes

One famous company has employees of two types: extroverts and introverts. These two kinds of people differ in many ways, and in this problem we consider the difference in their behavior during the dinner.

The dining room in this company can be represented as a rectangular grid $N \times M$. In other words, each point with integer coordinates x, y ($0 \leqslant x \leqslant N, 0 \leqslant y \leqslant M$) contains a table. Each table is for one person.

We say that the (Euclidean) distance between points with coordinates (x_1, y_1) and (x_2, y_2) is equal to $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

When an introvert comes to the dining room, he selects such a free table that the minimal distance from it to the nearest occupied table is as large as possible. If there are several tables with the largest minimal distance, he selects the one of them with the smallest x coordinate. If there are still several tables, he selects the one of them with the minimal y coordinate.

When an extrovert comes to the dining room, he selects such a free table that the maximal distance from it to the farthest occupied table is as small as possible. If there are several tables with the smallest maximal distance, he selects the one of them with the smallest x coordinate. If there are still several tables, he selects the one of them with the minimal y coordinate.

You are given the size of the dining room and the description of Q events. Each event is either an arrival or a departure of one person. For each arrival you have to find the table that he occupies.

Input

The first line on the input contains integers N, M, Q, separated by spaces ($1 \leqslant N, M \leqslant 5000$, $0 \leqslant Q \leqslant 100$). Next Q lines describe events. If the i'th event is an arrival of an extrovert, the i'th line contains single string "`ext`"; if the event is an arrival of an introvert, the line contains single string "`int`"; if the event is a departure of some person, the line contains the 1-based index (among all events) of the event that describes his arrival.

It is guaranteed that the input data is correct, i.e. for each arriving human there is at least one empty table in the dining room at the moment of his arrival, and for each event of the departure the index described in the line corresponds to an event of arrival and the person arrived in this event is still inside the dining room.

Output

For each arrival print a line containing two space-separated integers — the coordinates of the table being occupied.

Examples

dinner.in	dinner.out
5 5 5 int 1 ext 3 int	0 0 0 0 0 0
4 4 8 ext ext int int int 2 int int	0 0 0 1 4 4 4 0 0 4 2 2 0 2

Problem 17. King's Palace Garden

Input file:	garden.in
Output file:	garden.out
Time limit:	3 seconds
Memory limit:	256 mebibytes

King of Lemuria Julian XCIX has built a new palace for himself in the middle of Nowhere. Nowhere is a large plain savanna with several baobab trees. Now Julian wants to design the Palace Garden around his palace. Lemuria customs dictate that the Garden should have a shape of a convex polygon with corners in centers of baobab trees and without any baobabs inside the Garden. But the Palace should be strictly inside it.

Your task is to help Julian to select the Garden of the maximal area. If there is more than one garden of the same maximal area possible, find any of them.

Input

In the first line of input there is one integer N — number of baobabs, $0 \leqslant N \leqslant 500$. Second line contains two integers — coordinates of the palace. Each one of the following N lines also contains two integers — coordinates of baobab tree. All coordinates (of the Palace and of baobabs) satisfy the condition $-10^9 \leqslant x, y \leqslant 10^9$. All $N + 1$ points are pairwise distinct and no three of them lie on the same line.

Output

The first line of the output should contain one number with exactly one digit after the decimal point — area of the garden.

The second line should contain one integer — number of corners of the garden.

The third line should contain the numbers of baobabs which stand in the corners of the garden in counterclockwise order. Baobabs in the input data are numbered from 1.

If there is no garden satisfying the conditions, the output should contain one number -1.

Examples

garden.in	garden.out
4 5 5 1 1 1 9 7 5 10 6	24.0 3 1 3 2
3 0 0 1 1 0 1 1 0	-1

Problem 19. Translation

Input file:	`brainfuck.in`
Output file:	`brainfuck.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Peter Herpinson is sure that the ultimate programming language of the future is the one with an unspeakable name. He has designed a processor which can directly perform instructions in that language. He now wants to sell his brainchild and launch commercial production of the processor. But since most programs are written in more conventional languages, Peter needs compilers to translate programs into that language. Peter has decided to go with just a demo translator from another language, a very simple one he's created earlier. He called that language VSL — a Very Simple Language. Currently Peter is busy making a presentation for his investors. He has delegated the writing of the translator to you.

The VSL language which Peter created is indeed very simple. All variables are 4 byte signed integers. Initial values for all variables are 0. Since variables have the same type, their declaration isn't necessary. The language has 4 types of operators: read, write, assign and compare. All but the compare operator are written in a separate line. The grammar of the operators is shown below.

$$
\begin{aligned}
\textit{operator} &::= \textit{read\_op} \mid \textit{write\_op} \mid \textit{assign\_op} \\
\textit{read\_op} &::= \texttt{read(}\textit{var\_name}\texttt{)} \\
\textit{write\_op} &::= \texttt{write(}\textit{expression}\texttt{)} \\
\textit{assign\_op} &::= \textit{var\_name}\texttt{=}\textit{expression} \\
\textit{expression} &::= \textit{number} \mid \textit{var\_name} \mid \textit{sum} \mid \texttt{(}\textit{expression}\texttt{)} \\
\textit{sum} &::= \textit{expression} + \textit{expression}
\end{aligned}
$$

Here *number* is an integer nonnegative constant not exceeding 9, *var_name* is a variable name, a word containing only lower case Latin symbols and digits, the digit not being allowed at the beginning of the variable name. Each variable has a unique name. The name cannot contain more than 10 symbols.

The compare operator has the following format:

$$
\begin{array}{c}
\texttt{if(}\textit{expression cmp\_op expression }\texttt{)then} \\
\textit{operator1} \\
\textit{operator2} \\
\ldots\ldots\ldots\ldots\ldots \\
\textit{operatork} \\
\texttt{[else} \\
\textit{operator1e} \\
\textit{operator2e} \\
\ldots\ldots\ldots\ldots\ldots \\
\textit{operatorme}\texttt{]} \\
\texttt{end}
\end{array}
$$

where *cmp_op* is one of the symbols '`<`', '`>`', '`=`', '`#`' meaning respectively "less", "more", "equals" and "is not equal to". The "`if` ... `then`" construct takes a separate line followed by a number (possibly none) of lines with operators that are to be performed if the condition is satisfied. This is optionally followed by "`else`" in a separate line, which begins a block of operators performed in the case when the condition is not satisfied. The operator ends with the word "`end`" in a separate line. The words "`read`", "`write`", "`if`", "`then`", "`else`", and "`end`" are keywords and cannot be variable names.

You must write a program which translates a program written in VSL to a program written in the unspeakable language. The program in unspeakable consists of a series of commands performed by the processor which Peter created. Apart from the processor the system has an unlimited amount of memory consisting of signed integer 4-byte cells, as well as a pointer to the current cell. Initially all memory cells are set to zero value. The unspeakable language has the following commands each of which is performed in 1 CPU cycle:

Command	Description
`>`	Moves the pointer 1 cell to the right
`<`	Moves the pointer 1 cell to the left
`+`	Increases the value of the cell where the pointer is set to by 1
`-`	Decreases the value of the cell where the pointer is set to by 1
`[`	If the value of the cell the pointer is set to is zero then move to the paired command `]` (taking into account the nesting), else continue performing commands from the next one on
`]`	If the value of the cell the pointer is set to is not zero then move to the paired command `[` (taking into account the nesting), else continue performing commands from the next one on
`.`	Output the value of the cell which the pointer is set to as a byte. The cell must have a value from 0 to 255 included.
`,`	Read the symbol from the input stream to the cell which the pointer is set to. The cell accepts a value from 0 to 255 inclusively according to the read symbol.

The opening and closing square bracket operators ('`[`' and '`]`') are paired. This means that the operators '`[`' and '`]`' must form a correct bracket sequence. Formally correct bracket sequences (CBS) can be defined as

CBS ::= " " | [CBS] | CBS CBS

Here " " means an empty line, while CBS CBS means a concatenation of two possibly different correct bracket sequences.

Please note that since the memory is unlimited the '`>`' and '`<`' operators can always be performed. Since the language variables are signed integers the decrease option applied to the number 0 yields -1. A program in the language ends when the execution of the program goes beyond the last command of the program.

To simplify input-output let all input and output numbers be greater or equal to 0 and less or equal to 9. The input is a string of symbols — digits, and the `read()` operator reads the following unread symbol and writes it to the variable that it has been given. At the same time the '`,`' operator of the unspeakable language reads the ASCII code of the following digit into the current cell. The ASCII codes of digits from 0 through 9 equal 48 to 57 respectively. Let's think that the `write()` operator receives an expression with a value between 0 and 9, and that the `write()` operator outputs a single symbol, namely, the digit equal to the expression value. The program in the unspeakable language must put out a string of symbols with ASCII codes from 48 to 57. This means that when the "point" operator is performed the cell must contain a number between 48 and 57 corresponding to the digit being put out at the moment.

Input

The input file contains a program in VSL for translation. It is guaranteed that the data given to the VSL program on the input be such that the value of any expression calculated in the program is between 0 and 9 while the initial program does not read more than 4 digits from the input stream. The number of lines in a program is not more than 100. Each expression contains no more than 10 operations of addition. Spaces can be put at random places between lexemes. The line length in the input file is not more than 256 symbols.

Output

The output file must contain a single line — a program in the unspeakable language. The resulting program is considered correct if with any input data with account of limitations explained above the program outputs the same string of digits as the initial program. The program in the unspeakable language should not try to read more digits than the initial program given the same input data. The program must end in no more than 10^6 CPU cycles. The number of commands in the output program also should not exceed 10^6. It is guaranteed that such a program exists.

Example

brainfuck.in
read (a) if (a >0)then read(b) end read(c) write(a +(b))
brainfuck.out
,>++++++++[<------>-]<[<,<++++++++[>------<-]]>[>+<-]>>+++++++++[<++++++>-]<.

Problem 23. Flights

Input file:	`flights.in`
Output file:	`flights.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

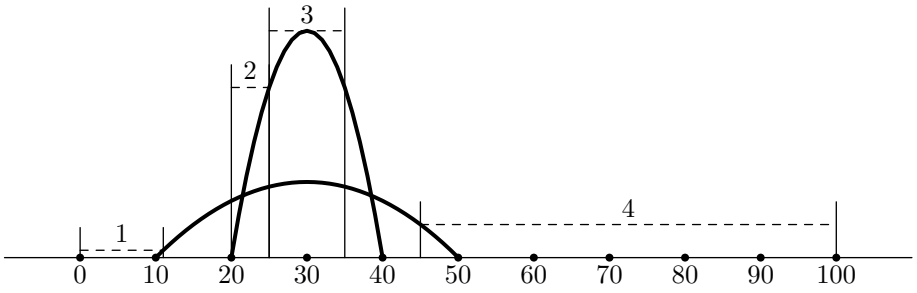
Army is busy: military exercises had started yesterday. All types of forces are doing, hopefully, a good job. For example, artillery is launching missiles, while aviation is delivering supplies to infantry.

The military ground space is a straight line. Aviation bases and infantry regiments are located somewhere on the line, and artillery is launching ballistic missiles everywhere. All missile launches are planned (don't forget, it's just an exercise), each at a certain time along a certain trajectory. Aviation flights are also planned in certain time and space intervals. Everything will be fine, but there are those missiles, which can be deadly even during exercises!

You should help aviation generals to plan the minimal safe altitude for each flight. Given the information about flight's time and space intervals, the minimal safe altitude for a flight is the minimal altitude such that all missile trajectories in the corresponding time and space interval are at or below this altitude. If there are no missiles in the flight's time and space interval, then the minimal safe altitude is defined to be zero.

Ballistic missiles are launched from the ground, which is defined to have a zero altitude, and fly along a vertically symmetrical parabola. Missile speed is ignored for this problem, missiles are assumed to follow their trajectory instantaneously.

For example, the picture below shows trajectories of two ballistic missiles in solid lines, and the minimal safe altitudes for four different flights in dashed lines. Vertical lines delimit space intervals of each flight in this sample. Time intervals of the flights in this sample include the launch of both missiles.



Input

The first line of input contains a single integer n — the number of missile launches planned ($1 \leqslant n \leqslant 50\,000$).

The following n lines describe one missile launch each. Each line contains three integers: the missile launch point p and coordinates of the highest point of missile trajectory x and y ($0 \leqslant p < x \leqslant 50\,000, 0 < y \leqslant 50$); p and x are coordinates along the military ground line, y gives the altitude of the highest point of missile trajectory. Missiles are launched one by one every minute in the order they are described in the input.

Next line contains the single integer m — the number of flights planned ($1 \leqslant m \leqslant 20\,000$).

The following m lines describe one flight each. Each line contains four integers: t_1 and t_2 ($1 \leqslant t_1 \leqslant t_2 \leqslant n$) — the time interval for the flight, and x_1 and x_2 ($0 \leqslant x_1 \leqslant x_2 \leqslant 50\,000$) — the space interval for the flight along the military ground line. Time and space intervals are inclusive of their endpoints. Time moment 1 corresponds to the first missile launch, and time moment n corresponds to the last one.

Output

For each flight write a separate line with the minimal safe altitude, with absolute error not exceeding 10^{-4}.

Example

flights.in	flights.out
2 10 30 10 20 30 30 4 1 2 0 11 1 2 20 25 1 2 25 35 1 2 45 100	0.975 22.5 30.0 4.375
2 0 10 10 30 40 10 6 1 2 0 32 1 1 19 35 2 2 0 32 1 2 15 35 1 2 21 27 1 2 2 100	10.0 1.9 3.6 7.5 0.0 10.0

Problem 29. Huzita Axiom 6

Input file:	`huzita.in`
Output file:	`huzita.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

The first formal axiom list for origami was published by Humiaki Huzita and Benedetto Scimemi and has come to be known as the Huzita axioms. These axioms describe the ways in which a fold line can be generated by the alignment of points and lines. A version of the six axioms follows.

1. For points p_1 and p_2, there is a unique fold that passes through both of them.
2. For points p_1 and p_2, there is a unique fold that places p_1 onto p_2.
3. For lines l_1 and l_2, there is a fold that places l_1 onto l_2.
4. For a point p_1 and a line l_1, there is a unique fold perpendicular to l_1 that passes through point p_1.
5. For points p_1 and p_2 and a line l_1, there is a fold that places p_1 onto l_1 and passes through p_2.
6. For points p_1 and p_2 and lines l_1 and l_2, there is a fold that places p_1 onto l_1 and p_2 onto l_2.

Roman is a good coder, but he is new to origami construction, so he decided to write a program to calculate the necessary folds for him. He already finished coding the cases for the first five axioms, but now he is stuck on the harder case, the axiom number 6. So he decided to hire a team of good coders — your team — to implement this case for his program.

Input

The input consists of one or more test cases. The total number of test cases t is specified in the first line of the input file. It does not exceed 20 000.

Each test case consists of exactly four lines, describing l_1, p_1, l_2 and p_2, in that order. Each line is described by four integers — the coordinates of two different points lying on it: x_1, y_1, x_2, y_2. Each point is described just by two integers — its x and y coordinates. All coordinates do not exceed 10 by their absolute values. It is guaranteed that neither p_1 lies on l_1 nor p_2 lies on l_2. Lines l_1 and l_2 are different, but points p_1 and p_2 may be the same.

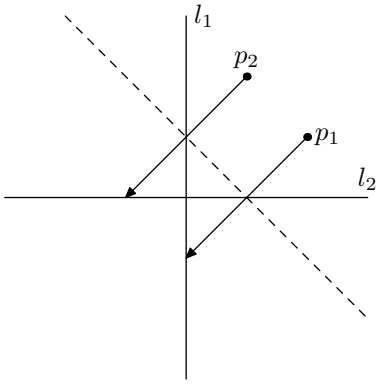
Output

For each test case write a separate line with the description of the straight line one should use for folding. Use the same format as in the input — specify the coordinates of two points on it. Either x or y coordinates of those two points must differ by at least 10^{-4}. Coordinates must not exceed 10^9 by their absolute value. The judging program will check that both the distance between p_1 after folding and l_1; and the distance between p_2 after folding and l_2 do not exceed 10^{-4}. If there are multiple solutions, any one will do. If there are no solutions, output the line of four zeros, separated by spaces.

Example

The picture to the right illustrates the first sample. The fold line is dashed.

huzita.in	huzita.out
2 0 0 0 1 2 1 0 0 1 0 1 2 0 0 0 1 5 0 1 0 1 1 6 0	0.0 1.0 2.0 -1.0 0 0 0 0



Problem 31. Report

Input file:	`standard input or input.txt`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	64 mebibytes

The regional office found out about a weird system of numeration for official documents used in one of the local subordinate offices. One set of digits is used for odd positions in numbers, and another set is used for even positions (positions are considered to be enumerated from right to left starting with 0 position). There are two rules to obey:

1. Numbers that can be generated with such system cannot be skipped.
2. Numbers are generated in ascending order.

For example, if someone uses digits 0,5,6 for even positions and 0 and 7 for odd positions, then the numbers will be generated in the following order: 0, 5, 6, 70, 75, 76, 500, 505, 506, 570, 575, 576, 600, ... It soon turned out that some other offices had also decided to adopt this numeration system.

The central office has decided to move on and adopt this system too. The office employees assumed that this numeration system will be used not only for documents, but also for the page numbers of the reports. The boss asks you to write a program, that, for given sets of digits and amounts of times each digit appeared while numerating pages (it is required that ALL pages are numerated) will calculate the total amount of pages for the given report. If it is impossible to calculate this amount, the program should output "`NO`" (quotes are for clarity only and should not appear in the output).

Input

The first line contains two integer numbers L and K ($2 \leqslant L, K \leqslant 10$). L and K are the amounts of digits used for even and odd sets correspondingly. The second line contains the digits used for even positions, and the third line contains the digits used for odd positions. Two sets can share some digits. The last line contains ten numbers separated by spaces. The first number gives the amount of zeros, the second — the amount of ones and etc. The last one gives the amount of nines. These numbers do not exceed 10^{10}.

Output

The output should contain the single line with the answer.

Example

standard input or input.txt	standard output
3 2 0 6 5 7 0 7 0 0 0 0 10 4 6 0 0	12
3 2 0 6 5 7 0 9 0 1 0 0 10 4 6 0 0	NO