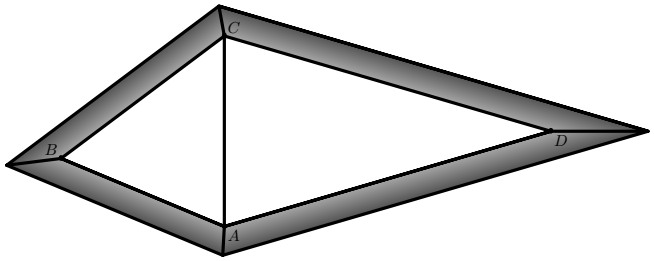
Problem A. The Baguette Master

Input file:	`baguette.in`
Output file:	`baguette.out`
Time limit:	1 second
Memory limit:	256 megabytes

A special type of board called *baguette slat* is used to make picture frames. The edges of the slat have different finish and thus *inside* and *outside* edges are distinguished. Only the inside edge can touch the picture in a frame.

Most pictures are rectangular, but recently the workshop where you work as the carpenter's apprentice got an order to prepare frames for an exhibition of works of an avant-garde group. You were assigned to prepare the slats for this order. Having looked at the picures, you were taken aback — they were not rectangular! More close examination confirmed that each picture is a convex quadrangle (or, as a special case, a triangle).

In any case, the order must be fulfilled... Unfortunately, your first attempts, based only on the lengths of the sides of the pictures, failed. You did not account for the fact that a quadrangle is not fully defined by its side lengths alone. Now your friends have told you that knowing the length of one of the diagonals in addition to the sides is sufficient.



Can you detemine the minimal length of slat needed for a frame for one of those pictures? Note that a frame consist of four (or three in case of triangular picture) trapezoidal pieces. Both the inner and the outer perimeter of the frame form a quadrangle (or a triangle) and the picture must fit inside the frame with no gaps or overlaps.

Input

The first line of input contains the width of the slat. The second line contains five numbers giving the shape and size of the picture. If A, B, C, D are the corners of the picture in the order they appear on the perimeter of the picture, the five values are the lengths of the sides AB, BC, CD, DA and the diagonal AC. All the numbers are positive integer, none of them exceeds 10^4.

Output

Output the required length of slat, accurate to within 10^{-3}.

Example

baguette.in	baguette.out
2 13 15 25 25 14	102.126
4 20 10 15 18 27	100.453
4 20 7 15 18 27	116.569

Note

The picture above illustrates the first sample. In the third sample the picture shape in fact is a triangle.

Problem B. Pouring Water

Input file:	barrels.in
Output file:	barrels.out
Time limit:	1 second
Memory limit:	256 megabytes

There are N fairly large barrels. Initially the first barrel contains 1 litre of water, the second 2 litres etc. Thus, the last barrel contains N litres of water.

The following *pouring operation* is allowed: if the barrel p contains A litres and the barrel q contains B litres of water and $A \le B$, then A litres may be poured from the barrel q into the barrel p, leaving $2 \cdot A$ litres in the barrel p and $B - A$ litres in the barrel q. We'll denote such an operation $s(p, q)$.

Using only the pouring operation described above, you need to collect the maximal possible amount of water in one barrel. You may execute no more than 10^7 pouring operations, and when the goal is reached, the extra operations are prohibited!

Input

The only line of input contains the integer N ($3 \le N \le 100$).

Output

The first line of output must contain K, the number of pouring operations. Each of the following K lines must describe one pouring operation $s(p, q)$ as two space-separated integers p and q. The operations must be given in the order they are performed. If there are several solutions, output any one of them.

Example

barrels.in	barrels.out
5	7 1 3 4 5 5 4 1 2 3 5 1 3 4 1

Problem C. Old Chess Sets

Input file:	`chess.in`
Output file:	`chess.out`
Time limit:	1 second
Memory limit:	256 megabytes

In preparation for remodeling of your flat, you're clearing out the junk that has accumulated over the years and find, among other forgotten things, two chess sets you had received as presents. Unfortunately, neither of the sets is complete... In the end you don't have the heart to throw them out and you decide to see if you can assemble one full set from the two partial ones.

Recall that a complete chess set consists of 32 pieces, 16 white and 16 black. There's a king, a queen, two bishops, two knights, two rooks and eight pawns in each color. In the following, the pieces will be named by color and rank, separated by a single space, for example "`white king`", "`black pawn`" etc.

Which pieces from the second set have to be added to the first to get one full set?

Input

The first line of input contains two integers k_1 and k_2 ($1 \le k_1, k_2 < 32$), the numbers of figures left in each set. The follwoing k_1 lines list the contents of the first set in arbitrary order and the following k_2 lines after that the contents of the second set.

Output

Output (in arbitrary order) $32 - k_1$ lines listing the figures that have to be moved from the second set to the first. If a complete set can't be collected, output "`impossible`" as the first and only line.

Examples

chess.in	chess.out
30 10 black knight white rook white rook black pawn black pawn white pawn black king white queen white bishop black pawn black bishop black rook white knight black pawn white pawn white pawn white pawn white pawn white pawn white pawn white pawn white pawn black pawn white knight white king black pawn black pawn black knight black rook black bishop white bishop black knight white rook white rook black pawn black pawn white pawn black king black queen white bishop black pawn	black queen black pawn
3 1 black knight white rook white rook black pawn	impossible

Problem D. Checkmate with Bishop and Knight

Input file:	`chess2.in`
Output file:	`chess2.out`
Time limit:	1 second
Memory limit:	256 megabytes

Having assembled a chess set from the two old ones, you became nostalgic... Forgetting about the clean-up, you started to recall the school chess club and one of the more complex topics you studied there: how to checkmate the opponent's king with a bishop and a knight. Unfortunately, you could not recall much of it, and decided to settle for a simpler task: to verify whether a checkmate could be achieved in the next move.

Write a program to answer this question!

We also remind you of the chess rules relevant to solving this particular problem:

- A square *is under attack* of a piece on the board if the piece could reach that square in the next move.
- No piece can move into a square occupied by another piece of the same color; even so, the occupied square is still considered to be under attack by the first piece.
- A piece can move into a square occupied by a piece of the opponent; in that case the piece of the opponent is *captured* (removed from the board).
- A king can move to an adjacent square vertically, horizontally or diagonally, except it can't move into a square that is under attack by any piece of the opponent.
- A bishop can move into any square that is on the same diagonal as the bishop's current position, except it can't move over squares occupied by other pieces.
- A knight moves in a weird zigzag way: first to an adjacent square vertically or horizontally (even if that square is occupied) and then to an adjacent square diagonally in a direction away from the initial square.
- A player is checkmated when his king becomes under attack and he can't remove the threat (by either moving the king to another square, blocking the attack with another piece or capturing the attacking piece) in the next move.

Input

The input consists of four lines containing the positions of the white king, white bishop, white knight and black king in the algebraic notation. In that notation, rows of the board are numbered "`1`"... "`8`" from bottom to top and the columns are marked "`a`"... "`h`" from left to right. Thus, the lower left square of the board is "`a1`" and the upper right one "`h8`". It may be assumed that the black king is not under attack in the given position.

Output

If checkmating in the next move is possible, output, without any spaces, the piece to move ("`K`" for king, "`B`" for bishop, "`N`" for knight) and the square it moves to. If there are several possible moves, output any one of them.

If checkmate in one move is not possible, output the word "`impossible`".

Examples

chess2.in	chess2.out
a3 c2 a5 a1	Nb3
a3 c4 a5 a1	impossible

Problem E. Dance Party

Input file:	dance.in
Output file:	dance.out
Time limit:	1 second
Memory limit:	256 megabyte

Dance parties, whether formal balls or less formal events, usually have dress codes for the guests. The code may be more or less strict. In case of a formal ball, the style of music and dances, such as Classicism, Empire, Rococo, Romantism, Biedermeier, Modernism, also determines the dress code. In less formal parties, any simple unusual idea can be good fun.

For one party, it was decided to assign a color for each guest. The colors would be drawn in advance and everyone would have to come dressed in the assigned color. Each color would be assigned to two persons and guests with matching colors would form pairs for the first dance. Everything looked fine until it was found that the color lists for ladies and gentelmen were not separate; instead, everyone drew their color from one big list. Thus, it was possible, that a pair will be formed either by a gentelman and a lady, or two ladies, or even two gentelmen. The organizer of the event responded to the concerns with "the probability of a male-male pair is extremely low", but Pat still suspects that the probability of getting at least one such pair is too high. To compute the actual probability, he asked the organizer to provide the number of colors and the numbers of ladies and gentelmen expected to participate in the dance. Having gotten these, he now wonders how to actually compute the probability.

Input

The only line of input contains three integers C, M and F ($1 \le C, M, F \le 10^9$; $M + F \le 2C$), the numbers of colors, gentelmen, and ladies, respectively.

Output

The first and only line of output must contain the probability explained above, with either the absolute or the relative error not exceeding 10^{-9}.

Example

dance.in	dance.out
20 10 12	0.776809067131648

Problem F. Determinant

Input file:	det.in
Output file:	det.out
Time limit:	4 seconds
Memory limit:	64 megabytes

You're given a non-degenerate $n \times n$ matrix $A = (a_{i,j})$ over the ring of integers and an integer δ. Check whether it is possible to change exactly one elment of the matrix A so that the determinant of the resulting matrix A' would equal δ.

Input

The first line of input contains n, the order of the matrix ($1 \le n \le 30$). The following n lines contain $a_{i,j}$, the elements of the matrix ($|a_{i,j}| \le 100$). The last line contains the integer δ ($|\delta| \le 10^{100}$).

Output

If the given matrix A can be converted into a matrix A' whose determinant is δ by changing one element, the first and only line of output must contain three integers, the row and column index of the element and its new value. If no change is required, output the indices and value of any element. If the desired A' can't be obtained, output -1 as the answer.

Examples

det.in	det.out
3 3 8 4 9 2 5 6 7 1 42	1 1 10
2 3 5 7 11 17	-1

Problem G. Midsummer Fires

Input file:	`fires.in`
Output file:	`fires.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

A rural flatland in Northeastern Europe is modeled as the Cartesian plane with the x axis directed to the East and y axis to the North. Households of negligible size are located at M points with integer coordinates.

N lowly hills rise above the plane. Each of them has the form of a regular quadrilateral pyramid of 1/3 unit height with a unit square with vertices at integer coordinates as the base.

Traditionally an exuberant celebration is held in the Midsummer night and bonfires are lit on hill tops. Beliefs have it that the fires would bring wealth to all the places they shine upon.

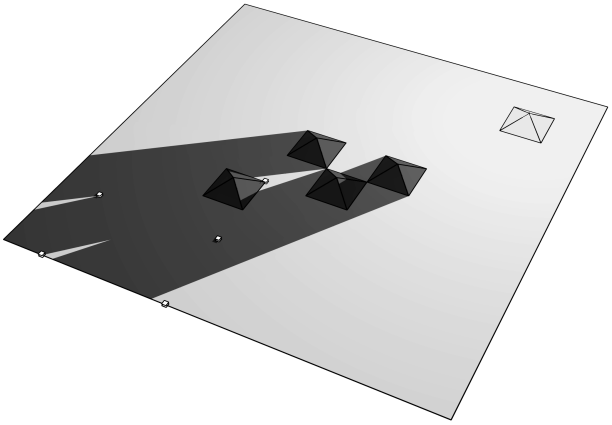
Consider a single fire on the first hill and regard it as a point light source. Find out which households receive its blessing. Households count as illuminated also if they sit on the border between lit and shadowed space.

Input

The first line of input contains integers M and N ($1 \le M, N \le 10^5$).

Each of following N lines contains X_i and Y_i ($-10^8 \le X_i, Y_i \le 10^8$), the coordinates of the North-east corner of the i-th hill. The points are distinct.

Each of next M lines contains X_j and Y_j ($-10^8 \le X_j, Y_j \le 10^8$), the coordinates of the households. The points are distinct.



Output

For each household in the order in which they appear in the input, write the line ":)" if the household is illuminated by the bonfire on the first hill or ":(" otherwise.

Example

fires.in	fires.out
5 5 9 9 5 6 7 6 6 5 4 4 1 2 1 0 4 4 4 2 4 0	:) :) :) :(:)

Problem H. Kids' Play

Input file:	`kids.in`
Output file:	`kids.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

It is not easy to keep the children engaged in the day care. Someone's missing Mommy, someone's finger hurts, someone can't decide what they want to play next... Now Natalya Alexandrovna invented a new game. She prepared a bunch of pairs of cards with pictures. There are N desks in her classroom and Natalya Alexandrovna picks N pairs from her supplies (the pictures are numbered from 1 to N) and lays two cards on each desk. Of the two cards, one lies face up and the other face down. Even more, from each pair, exactly one card is face up.

In the beginning of the game, the K ($K \le N$) kids in her care on that day sit down at the desks, at most one kid at each desk. In the first minute of the game, each kid has to peek at the card laid on the desk face down and move to the desk where the same picture is face up. After that the kids continue to move according to the same rules every following minute.

The best part of the game is that kids can continue playing until the parents come to collect them. And even if some kids are collected before others, the remaining ones can still carry on.

There's only one problem: Natalya Alexandrovna is unable to tell the parents the location of their child at any given moment. She agrees to give you the initial positions of the kids and information about all cards (even the ones laid face down) to help her respond fast to the parents looking for their offspring.

Input

The first line of input contains the integers N and K ($1 \le K \le N \le 10^5$). The following N lines contain two integers each: the numbers of the pictures on the cards laid face up and face down on the corresponding desk. The following K lines also contain two integers each: the number of the desk from 1 to N that was the initial desk number of the corresponding kid and the number of minutes from the start of the game to the moment when the parents come looking for the kid (integer between 1 and 10^9).

Output

Output K lines, one for each kid. The j-th line must contain a single integer, the number of the desk at which the j-th kid is sitting when his/her parents arrive (kids are numbered in order they appear in the input file).

Example

kids.in	kids.out
6 4 1 2 2 3 3 1 5 4 4 5 6 6 1 10 3 3 4 101 6 1000	2 3 5 6

Problem I. Odd Factor

Input file:	`oddfactor.in`
Output file:	`oddfactor.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Odd factor of an undirected graph is a spanning subgraph (that is, a subgraph containing all vertices of the original graph) where the local degree of each vertex is odd.

Given an undirected graph, either construct any of its odd factors or verify that none exists.

Input

The first line of input contains the number of vertices n ($1 \le n \le 10^5$) and the number of edges m ($0 \le m \le 10^6$) of the graph. The follwing m lines describe the edges, with each edge given on a separate line as two integers a_i and b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$), the numbers of the vertices the edge connects.

Output

If an odd factor can be constructed, the first line of output must contain the number of edges in the factor and the following lines must list the edges. If there's no odd factor, output -1 as the answer.

Examples

oddfactor.in	oddfactor.out
4 4 1 2 2 3 3 4 4 1	2 1 2 3 4
4 3 1 2 1 3 1 4	3 1 2 1 3 1 4
3 3 1 2 1 3 2 3	-1

Problem J. One-Armed Bandit

Input file:	`onearmedbandit.in`
Output file:	`onearmedbandit.out`
Time limit:	1 second
Memory limit:	256 megabytes

One-armed bandit is a gambling machine with a chance to win a prize many times bigger than the bet. The game consists of rotating the reels with the goal to obtain a winning combination. The better the combination, the bigger the payout.

Dennis has learned that the machine operates in base B and winning rounds are the ones with a fixed sum of the digits of the sequence number of the round. He also knows the sequence number of the last winning round and wants to win T more times. Help him compute the sequence number of the round when he reaches his target and can finish. Dennis is prepared to play for a very long time, as there's no clock in the casino anyway.

Input

The first line of input contains two integers: B ($2 \le B \le 36$), the base of the number system used by the machine, and T ($1 \le T \le 10^{18}$), the target number of winning rounds Dennis wants to play. The second line contains the sequence number of the last winning round; the number has no leading zeroes and is at most 10^4 digits long. The characters `0...9A...Z` are used as digits.

Output

The first and only line of output must contain the sequence number of the last round that Dennis plays. If that number would be longer than 10^4 digits in base B, output -1 as the answer.

Example

onearmedbandit.in	onearmedbandit.out
10 2 11	101
11 676 22431699A7A625	22431699A88626