Problem B. Double Clique

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

You are given an undirected graph G with n nodes and m edges. The set of vertices is V and the set of edges is E.

Let the *Complement* of G be G'. The *Complement* of a graph is a graph with all of the same nodes, but if there's no edge between nodes a and b in G, then there is an edge between a and b in G', and if there is an edge between a and b in G, then there is no edge between a and b in G'.

A *Clique* is a subset of nodes that have an edge between every pair. A subset of nodes S is called a *Double Clique* if S forms a clique in G, and $V - S$ forms a clique in G'. Note that an empty set of nodes is considered a clique.

Given a graph, count the number of double cliques in the graph modulo $10^9 + 7$.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n and m ($1 \le n, m \le 2 \times 10^5$), where n is the number of nodes and m is the number of edges in the graph. The nodes are numbered $1..n$. Each of the next m lines will contain two integers a and b ($1 \le a < b \le n$), representing an edge between nodes a and b. The edges are guaranteed to be unique.

Output

Output a single integer, which is the number of Double Cliques in the graph modulo $10^9 + 7$.

Examples

standard input	standard output
3 3 1 3 1 2 2 3	4

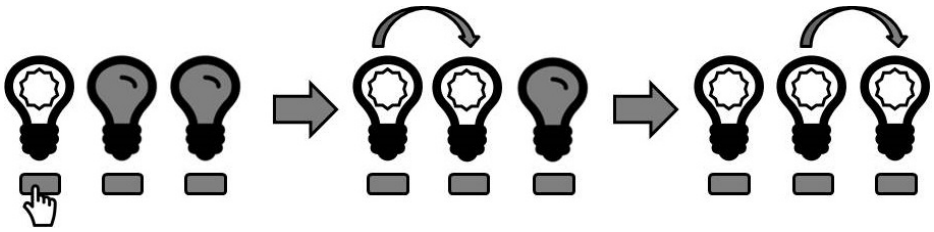
Problem C. Flashing Fluorescents

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

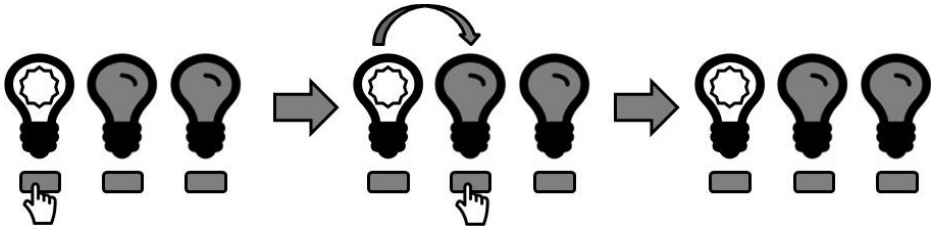
You have n lights, each with its own button, in a line. Pressing a light's button will toggle that light's state; if the light is on, it will turn off, and if the light is off, it will turn on. The lights change at 1 second timesteps. You can press a button at any time, but it will not take effect until the next timestep. Before each timestep, you may choose to push at most one button (you may also choose to not press any button).

Pushing a button will affect not just the light in question, but all lights down the line. More specifically, if you choose to press the i-th button right before the k-th timestep, then the $(i+m)$-th light will toggle on the $(k+m)$-th timestep (with $i+m \le n$). For example, if you press button 5 just before time 19, then light 5 will toggle at time 19, light 6 will toggle at time 20, light 7 will toggle at time 21, and so on. If you push a button that will take effect at the same time as its light would have toggled due to an earlier button press, then the two cancel each other out, including subsequent toggles.

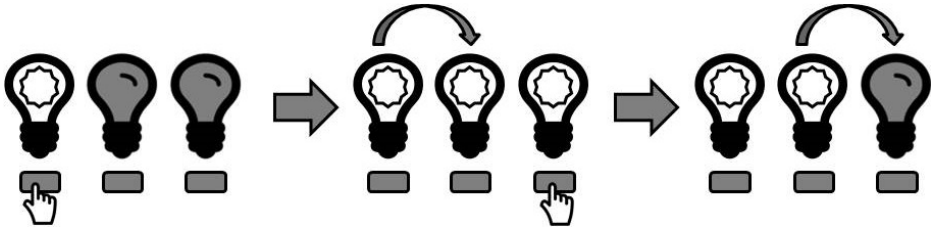
Suppose there are three lights, all of which are off at the start. If you press the first button before the first timestep, this will happen in three timesteps:



Now, suppose you press the first button before the first timestep, and then the second button between the first and second timesteps. The button press will cancel out the propagation, and this will happen (note that the propagation will go no further):



Now, suppose you press the first button before the first timestep, and then the third button between the first and second timesteps. All three lights will be on at the second timestep (but not the third):



You wish to turn on all the lights. What is the earliest time you could possibly see all of the lights turned on? Note that if the lights are all on at time t but not at time $t+1$ due to this propagation, t is still the correct answer.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single string S ($1 \leq |S| \leq 16$). The string S will contain only the characters 1 and 0, where 1 represents that that light is initially on, and 0 represents that that light is initially off. The first character is light 1, the next is light 2, and so on.

Output

Output a single integer, which is the earliest time at which all of the lights are on.

Examples

standard input	standard output
1101	1
1	0
000	2

Problem D. Missing Gnomes

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

A family of n gnomes likes to line up for a group picture. Each gnome can be uniquely identified by a number $1..n$ written on their hat.

Suppose there are 5 gnomes. The gnomes could line up like so: $1, 3, 4, 2, 5$.

Now, an evil magician will remove some of the gnomes from the lineup and wipe your memory of the order of the gnomes. The result is a subsequence, perhaps like so: $1, 4, 2$.

He then tells you that if you ordered all permutations of $1..n$ in lexicographical order, the original sequence of gnomes is the first such permutation which contains the remaining subsequence. Your task is to find the original sequence of gnomes.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n and then m ($1 \le m \le n \le 10^5$), where n is the number of gnomes originally, and m is the number of gnomes remaining after the evil magician pulls his trick. Each of the next m lines will contain a single integer g ($1 \le g \le n$). These are the remaining gnomes, in order. The values of g are guaranteed to be unique.

Output

Output n lines, each containing a single integer, representing the first permutation of gnomes that could contain the remaining gnomes in order.

Examples

standard input	standard output
5 3 1 4 2	1 3 4 2 5
7 4 6 4 2 1	3 5 6 4 2 1 7

Problem E. Prefix Free Code

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Consider n initial strings of lower case letters, where no initial string is a prefix of any other initial string. Now, consider choosing k of the strings (no string more than once), and concatenating them together. You can make this many such composite strings:

$$n \times (n-1) \times (n-2) \times \ldots \times (n-k+1)$$

Consider sorting all of the composite strings you can get via this process in alphabetical order. You are given a test composite string, which is guaranteed to belong on this list. Find the position of this test composite string in the alphabetized list of all composite strings, modulo $10^9 + 7$. The first composite string in the list is at position 1.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers, first n and then k ($1 \le k \le n$), where n is the number of initial strings, and k is the number of initial strings you choose to form composite strings. The upper bounds of n and k are limited by the constraints on the strings, in the following paragraphs.

Each of the next n lines will contain a string, which will consist of one or more lower case letters $a..z$. These are the n initial strings. It is guaranteed that none of the initial strings will be a prefix of any other of the initial strings.

Finally, the last line will contain another string, consisting of only lower case letters $a..z$. This is the test composite string, the position of which in the sorted list you must find. This test composite string is guaranteed to be a concatenation of k unique initial strings.

The sum of the lengths of all input strings, including the test string, will not exceed 10^6 letters.

Output

Output a single integer, which is the position in the list of sorted composite strings where the test composite string occurs. Output this number modulo $10^9 + 7$.

Examples

standard input	standard output
5 3 a b c d e cad	26
8 8 font lewin darko deon vanb johnb chuckr tgr deonjohnbdarkotgrvanbchuckrfontlewin	12451

Problem H. Recovery

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

Consider an $n \times m$ matrix of ones and zeros. For example, this 4×4:

$$\begin{matrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}$$

We can compute even parity for each row, and each column. In this case, the row parities are $[0, 1, 1, 0]$ and the column parities are $[1, 0, 0, 1]$ (the parity is 1 if there is an odd number of 1s in the row or column, 0 if the number of 1s is even). Note that the top row is row 1, the bottom row is row n, the leftmost column is column 1, and the rightmost column is column m.

Suppose we lost the original matrix, and only have the row and column parities. Can we recover the original matrix? Unfortunately, we cannot uniquely recover the original matrix, but with some constraints, we can uniquely recover a matrix that fits the bill. Firstly, the recovered matrix must contain as many 1's as possible. Secondly, of all possible recovered matrices with the most 1's, use the one which has the smallest binary value when you start with row 1, concatenate row 2 to the end of row 1, then append row 3, row 4, and so on.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of exactly two lines. The first line will contain a string R ($1 \le |R| \le 50$), consisting only of the characters 0 and 1. These are the row parities, in order. The second line will contain a string C ($1 \le |C| \le 50$), consisting only of the characters 0 and 1. These are the column parities, in order.

Output

If it is possible to recover the original matrix with the given constraints, then output the matrix as $|R|$ lines of exactly $|C|$ characters, consisting only of 0's and 1's. If it is not possible to recover the original matrix, output -1.

Examples

standard input	standard output
0110 1001	1111 0111 1110 1111
0 1	-1
11 0110	1011 1101

Problem I. Red Black Tree

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a rooted tree with n nodes. The nodes are numbered $1..n$. The root is node 1, and m of the nodes are colored red, the rest are black.

You would like to choose a subset of nodes such that there is no node in your subset which is an ancestor of any other node in your subset. For example, if A is the parent of B and B is the parent of C, then you could have at most one of A, B or C in your subset. In addition, you would like exactly k of your chosen nodes to be red.

If exactly m of the nodes are red, then for all $k = 0..m$, figure out how many ways you can choose subsets with k red nodes, and no node is an ancestor of any other node.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n ($1 \le n \le 2 \times 10^5$) and m ($0 \le m \le min(10^3,\ n)$), where n is the number of nodes in the tree, and m is the number of nodes which are red. The nodes are numbered $1..n$.

Each of the next $n-1$ lines will contain a single integer p ($1 \le p \le n$), which is the number of the parent of this node. The nodes are listed in order, starting with node 2, then node 3, and so on. Node 1 is skipped, since it is the root. It is guaranteed that the nodes form a single tree, with a single root at node 1 and no cycles.

Each of the next m lines will contain single integer r ($1 \le r \le n$). These are the numbers of the red nodes. No value of r will be repeated.

Output

Output $m + 1$ lines, corresponding to the number of subsets satisfying the given criteria with a number of red nodes equal to $k = 0..m$, in that order. Output this number modulo $10^9 + 7$.

Examples

standard input	standard output
4 1 1 1 1 3	5 4
4 4 1 1 1 1 2 3 4	1 4 3 1 0
14 4 1 2 1 2 3 4 5 5 13 8 10 4 4 8 3 12 13	100 169 90 16 0

Problem K. Zoning Houses

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Given a registry of all houses in your state or province, you would like to know the minimum size of an axis-aligned square zone such that every house in a range of addresses lies in the zone or on its border. The zoning is a bit lenient and you can ignore any one house from the range to make the zone smaller.

The addresses are given as integers from $1..n$. Zoning requests are given as a consecutive range of houses. A valid zone is the smallest axis-aligned square that contains all of the points in the range, ignoring at most one.

Given the (x, y) locations of houses in your state or province, and a list of zoning requests, you must figure out for each request: What is the length of a side of the smallest axis-aligned square zone that contains all of the houses in the zoning request, possibly ignoring one house?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers n and q ($1 \le n, q \le 10^5$), where n is the number of houses, and q is the number of zoning requests.

The next n lines will each contain two integers, x and y ($-10^9 \le x, y \le 10^9$), which are the (x,y) coordinates of a house in your state or province. The address of this house corresponds with the order in the input. The first house has address 1, the second house has address 2, and so on. No two houses will be at the same location.

The next q lines will contain two integers a and b ($1 \le a < b \le n$), which represents a zoning request for houses with addresses in the range $[a..b]$ inclusive.

Output

Output q lines. On each line print the answer to one of the zoning requests, in order: the side length of the smallest axis-aligned square that contains all of the points of houses with those addresses, if at most one house can be ignored.

Examples

standard input	standard output
3 2 1 0 0 1 1000 1 1 3 2 3	1 0
4 2 0 0 1000 1000 300 300 1 1 1 3 2 4	300 299

Problem L. Arithmetic Sequences

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

An Arithmetic Sequence of integers is one in which the next number in the sequence is obtained by adding a constant to the current number. For example, this is an arithmetic sequence (the constant is 7):

3, 10, 17, 24, 31, ...

Given a part of an arithmetic sequence with some numbers missing, fill in the missing numbers.

Input

Input will consist of a single line with exactly ten integers. Eight of them will be 0, the other two will be positive. The two positive integers may be anywhere among the ten integers, and will be no larger than 1,000. The 0 values represent missing values from the sequence.

Output

If it is possible to complete the sequence with integers, then output ten integers on a single line, with a single space between them, by replacing the 0 values with the correct numbers. If it is not possible to complete the sequence with integers, simply output a single -1. Although the two non-zero inputs are positive, the rest of the sequence might not be. Likewise, while the two non-zero inputs are not greater than 1,000, the rest of the sequence might not be.

Examples

standard input	standard output
5 0 15 0 0 0 0 0 0 0	5 10 15 20 25 30 35 40 45 50
5 0 0 15 0 0 0 0 0 0	-1
0 0 0 15 0 3 0 0 0 0	33 27 21 15 9 3 -3 -9 -15 -21
0 0 19 0 0 0 0 0 19 0	19 19 19 19 19 19 19 19 19 19

Problem M. Congruent Numbers

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

A congruent number is an integer that is the area of some right triangle where the length of each side of the triangle is a rational number. For this problem, we'll only consider the legs of the right triangle, and not the hypotenuse.

A rational number is a fraction, p/q, where p, the numerator, and q, the denominator, are integers. Note that if $q = 1$, then $p/1$ is an integer, so any integer is a rational number. Given two rational numbers which are the non-hypotenuse legs of a right triangle, determine if the area of that triangle is a congruent number. For the purposes of this problem, it is not necessary for the length of the hypotenuse to be a rational number.

Input

Input will consist of a single line with four integers p_1, q_1, p_2 and q_2 ($1 \le p_1, q_1, p_2, q_2 \le 10^5$) where p_1/q_1 and p_2/q_2 are the rational numbers which are the sides of a right triangle.

Output

Output a single integer, which is 1 if the area of the triangle is an integer, 0 if not. Note that the area has to be an integer, not just a rational number.

Examples

standard input	standard output
3 1 4 1	1
15 1 28 3	1
1 2 3 4	0
1 1 10 1	1

Problem N. Offside

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Of the seventeen enumerated laws of Soccer, Offside as described by Law 11 is perhaps the most contentious. In short, offside consists of two conditions: the offside position, and a state of play that turns the offside position into an offense. In an attempt to consistently enforce offside offenses (and generally ratchet down the poor sportsmanship of parents who consider themselves smarter than the referees, but too selfish to serve as referees themselves), your team has been contacted to implement the analysis algorithm for Offside Enforcement Technology, soon to be known as OET. This phase of OET will focus solely on the Offside Position, which may be stated as:

- Any part of a player's head, body, or feet (but not hands or arms) is in the opponents' side of the field (excluding the halfway line), AND
- Any part of a player's head, body, or feet (but not hands or arms) is closer to the opponents' goal line than BOTH the ball AND all but 1 opponent

For the purposes of OET, consider each player to be a single point, and determine if the offense is in an offside position. The field will be $100m \times 80m$, with the point $(0, 0)$ being the center of the halfway line, and the opponent's goal line running from $(50, -40)$ to $(50, 40)$.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will have exactly 23 lines of input. Each line will contain two integers, x and y ($-50 \le x \le 50$, $-40 \le y \le 40$), which indicate the position of the ball or a player on the field in meters. The first line will indicate the position of the ball, then the next 11 will be the offense and the last 11 will be the defense. No two players will be in the same position.

Output

Output a single integer, 1 of the play is offside, and 0 if it is not.

Examples

standard input	standard output
9 4 -45 -40 -11 18 44 -16 36 -32 -29 -30 7 -33 47 27 5 -37 -11 18 -15 -6 -7 -1 1 22 -39 11 -9 -9 10 38 -43 -8 -16 -29 43 -27 2 -27 -4 -30 49 -15 -48 10	1
9 4 -45 -40 -11 18 44 -16 36 -32 -29 -30 7 -33 47 27 5 -37 -11 18 -15 -6 -7 -1 1 22 -39 11 -9 -9 10 38 -43 -8 -16 -29 48 -27 2 -27 -4 -30 49 -15 -48 10	0

Problem O. Call Centre

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You just became the manager of a call centre. There are many variables to take into account to run an efficient call centre. To make these decisions, you need good data. Your call centre is huge. Typically, many new calls are made every second. Your system logs each incoming call. To get started with "data analytics", you want to answer simple questions like "between time s and time t, how many new calls were made to our call centre?"

Input

The first line of input consists of two integers n and q. Here, $1 \le n \le 86400$ is the number of seconds your call centre is open for each day and $1 \le q \le 5 \cdot 10^4$ is the number of queries you want to make. The next line contains n integers, each between 0 and 100. Consecutive integers will be separated by a single space. The i'th integer on this line indicates how many new calls were received by your call centre yesterday in the i'th second of operation. Finally, q lines follow. Each consists of two integers $1 \le s \le t \le n$ separated by a single space.

Output

Output consists of q lines, one for each query. For each query with integers s and t, you should output the total number of calls your call centre received yesterday between the s-th and t-th second of operation (including the calls received exactly at second s and at second t).

Examples

standard input	standard output
13 7 5 3 13 0 0 1 4 3 12 1 0 0 17 1 3 1 4 1 13 3 3 4 4 2 7 8 11	21 21 59 13 0 21 16