Problem A. Archeologists

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 megabytes

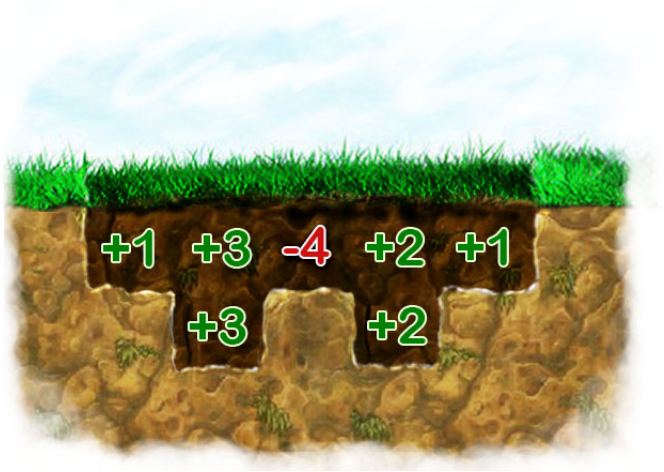
Your treasure hunter team has just discovered a giant archeological site, full of precious metals and valuable antiquities. The site is composed of n digging spots on a line.

The initial plans suggest that each of the n digging spots has a net profit associated with it. The i-th spot's associated profit is p_i. More specifically, this means that your team would gain p_i dollars for each meter dug in the i-th spot. Note that p_i may also be negative, which means that the running cost of the excavating machinery surpasses the actual gain from digging in the i-th spot.

Naturally, you would want to dig as much as possible in the most profitable spots. However, in order not to cause landslides, you are not allowed to have slopes that are too steep. More precisely, for any two adjacent spots, the difference between the digging depth at these spots cannot differ by more than 1 meter. In particular, spots 1 and n can be dug only at most 1 meter deep.

What is the largest net profit that you can obtain, under these conditions?

For instance, a valid digging plan that turns out to be optimal in the case of the first example input is illustrated below. The net profit of such plan is 8.



Input

The first line of the input will contain a positive integer n ($1 \le n \le 250\,000$).

The second line of the input will contain n integers p_i ($-10^6 \le p_i \le 10^6$), separated by spaces.

Output

Output exactly one integer, the largest profit that you can obtain.

Examples

standard input	standard output
5 1 3 -4 2 1	8
4 1 1 -2 3	5
5 -1 -3 0 -5 -4	0

Problem B. Reverse Game

Input file:	standard input
Output file:	standard output
Time limit:	1 second
Memory limit:	256 megabytes

Alice and Bob are playing a turn-based game. The rules of the game are as follows:

1. At the beginning of the game some binary string s is chosen.
2. On his turn player has to choose some substring t of s, equal to one of `10`, `110`, `100`, `1010`. Then the player has to reverse t. For example, if $s =$ `010101`, the player can select substring $t =$ `1010` and reverse it, obtaining $s =$ `001011`
3. The player who can't make a move (who can't choose an appropriate substring t) loses.
4. The players cannot skip a turn.

Which player has the winning strategy, if Alice moves first?

A string a is a substring of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Input

The only line of the input contains a binary string s ($1 \le |s| \le 10^6$) — the string with which Alice and Bob play.

Output

If Alice wins, output `Alice`. Otherwise, output `Bob`.

Examples

standard input	standard output
010	Alice
1111	Bob
1010	Bob
1010001011001	Alice

Note

In the first sample, Alice can choose substring `10` of `010` and reverse it, obtaining string `001`. Bob can't make any move with this string, and loses.

In the second sample, Alice can't make a single move and loses.

Problem C. 3-colorings

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

This is an output-only problem. Note that you still have to send code which prints the output, not a text file.

A valid 3-coloring of a graph is an assignment of colors (numbers) from the set $\{1, 2, 3\}$ to each of the n vertices such that for any edge (a, b) of the graph, vertices a and b have a different color. There are at most 3^n such colorings for a graph with n vertices.

You work in a company, aiming to become a specialist in creating graphs with a given number of 3-colorings. One day, you get to know that in the evening you will receive an order to produce a graph with exactly $6k$ 3-colorings. You don't know the exact value of k, only that $1 \le k \le 500$.

You don't want to wait for the specific value of k to start creating the graph. Therefore, you build a graph with at most 19 vertices beforehand. Then, after learning that particular k, you are allowed to add at most 17 edges to the graph, to obtain the required graph with exactly $6k$ 3-colorings.

Can you do it?

Input

There is no input for this problem.

Output

First, output n and m ($1 \le n \le 19$, $1 \le m \le \frac{n(n-1)}{2}$) — the number of vertices and edges of the initial graph (the one built beforehand). Then, output m lines of form (u, v) — the edges of the graph.

Next, for every k from 1 to 500 do the following:

Output e — the number of edges you will add for this particular k ($1 \le e \le 17$). Then, output e lines of the form (u, v) — the edges you will add to your graph.

There can't be self-loops, and for every k, all $m + e$ edges you use have to be pairwise distinct. The number of 3-colorings of the graph for a particular k has to be exactly $6k$.

Example

standard input	standard output
-	3 2 1 2 2 3 1 1 3 0

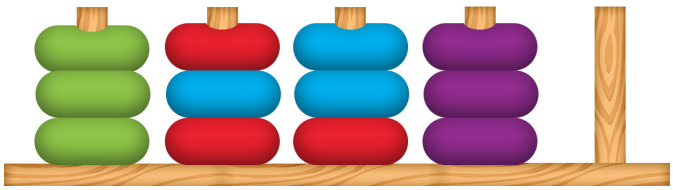
Note

The sample output is given as an example. It contains the output for $k = 1, 2$.

Problem D. Disk Sort

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given $n + 1$ rods and $3n$ disks. Initially, each of the first n rods contains exactly 3 disks. Each of the disks has one of n colors (identified by numbers from 1 to n). Moreover, there are exactly 3 disks of each of the n colors. The $n + 1$-th rod is empty.



At each step we can select two rods a and b ($a \neq b$) such that a has at least 1 disk and b has at most 2 disks, and move the topmost disk from rod a to the top of rod b. Note that no rod is allowed to contain more than 3 disks at any time.

Your goal is to sort the disks. More specifically, you have to do a number of operations (potentially 0), so that, at the end, each of the first n rods contains exactly 3 disks **of the same color**, and the $n+1$-th rod is empty.

Find out a solution to sort the disks in at most $6n$ operations. It can be proven that, under this condition, a solution always exists. If there are multiple solutions, any one is accepted.

Input

The first line of the input contains a positive integer n ($1 \le n \le 1\,000$). The next 3 lines of the input contain n positive integers $c_{i,j}$ each ($1 \le i \le 3$, $1 \le j \le n$, $1 \le c_{i,j} \le n$), the color each of the disks initially placed on the rods. The first of the 3 lines indicates the upper row, the second line indicates the middle row, and the third line indicates the lower row.

Output

The first line of the output must contain a non-negative integer k ($0 \le k \le 6n$), the number of operations. Each of the following k lines should contain two **distinct** numbers a_i, b_i ($1 \le a_i, b_i \le n + 1$, for all $1 \le i \le k$), representing the i-th operation (as described in the statement).

Examples

standard input	standard output
4 2 3 1 4 2 1 1 4 2 3 3 4	8 3 5 3 5 2 3 2 5 2 3 5 2 5 2 5 2
2 1 2 1 2 1 2	0

Problem E. Divisible by 3

Input file:	standard input
Output file:	standard output
Time limit:	2 seconds
Memory limit:	256 megabytes

For an array $[b_1, b_2, \ldots, b_m]$ of integers, let's define its **weight** as the sum of pairwise products of its elements, namely as the sum of $b_i b_j$ over $1 \le i < j \le m$.

You are given an array of n integers $[a_1, a_2, \ldots, a_n]$, and are asked to find the number of pairs of integers (l, r) with $1 \le l \le r \le n$, for which the weight of the subarray $[a_l, a_{l+1}, \ldots, a_r]$ is divisible by 3.

Input

The first line of the input contains a single integer n ($1 \le n \le 5 \cdot 10^5$) — the length of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — the elements of the array.

Output

Output a single integer — the number of pairs of integers (l, r) with $1 \le l \le r \le n$, for which the weight of the corresponding subarray is divisible by 3.

Examples

standard input	standard output
3 5 23 2021	4
5 0 0 1 3 3	15
10 0 1 2 3 4 5 6 7 8 9	20

Note

In the first sample, the weights of exactly 4 subarrays are divisible by 3:

- $\text{weight}([5]) = \text{weight}([23]) = \text{weight}([2021]) = 0$
- $\text{weight}([5, 23, 2021]) = 56703 = 3 \cdot 41 \cdot 461$

Problem F. Fence Job

Input file:	standard input
Output file:	standard output
Time limit:	1 second
Memory limit:	256 megabytes

Fred the Farmer wants to redesign the fence of his house. Fred's fence is composed of n vertical wooden planks of various heights. The i-th plank's height is h_i ($1 \le h_i \le n$). **Initially, all heights are distinct.**

In order to redesign the fence, Fred will choose some contiguous segment $[l...r]$ of planks and "level" them, by cutting them in order to make all heights equal to the minimum height on that segment. More specifically, the new heights of the segment become $h'_i = \min\{h_l, h_{l+1}, ..., h_r\}$ for all $l \le i \le r$.

How many different designs can Fred obtain by applying this procedure several (possibly 0) times? Since the answer may be huge, you are required to output it modulo $10^9 + 7$.

Two designs A and B are different if there is some plank that has a different height in A than in B.

Input

The first line of the input contains n ($1 \le n \le 3\,000$), the number of planks of Fred's fence.

The second line contains n **distinct** integers h_i ($1 \le h_i \le n, 1 \le i \le n$), the heights of each of the planks.

Output

Output a single integer, the number of different possible fence designs that can be obtained, modulo $10^9 + 7$.

Examples

standard input	standard output
3 1 3 2	4
5 1 2 3 4 5	42
7 1 4 2 5 3 6 7	124

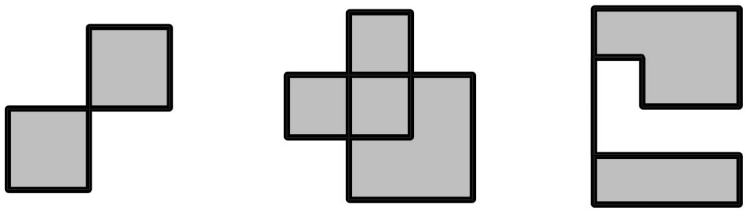
Problem G. Simple Hull

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	1024 megabytes

Gary has been trying to generate simple orthogonal polygons for his geometry homework, but his algorithm seems to be having some issues. After some good hours of debugging, he finally realized what is the problem: apparently, the polygons that he was generating may contain self-intersections, which was not at all what he intended!

More specifically, the "polygons" that Gary has generated are represented by a list of n points $p_i = (x_i, y_i)$, forming a closed polygonal chain. The polygonal chain may contain self-intersections. The segments formed by every two consecutive points (x_i, y_i) and (x_j, y_j) in the chain are either vertical or horizontal.

The polygonal chains for the example test cases are illustrated below (not to scale):



You have decided to help Gary fix this issue, by computing a simple (not self-intersecting) polygon with vertical and horizontal segments that fully contains the chain, and its area is as small as possible. What is the area of such a polygon?

Formally, you have to compute the **infimum** of the areas of all simple orthogonal polygons that contain all the segments $[p_i, p_j]$, for every two adjacent points p_i and p_j.

Input

The first line of the input will contain a positive integer n ($4 \le n \le 100\,000$). The following n lines will contain the points (x_i, y_i) in order ($1 \le x_i, y_i \le 10^6$). No two consecutive points coincide, and there are no two consecutive vertical segments or two consecutive horizontal segments.

Output

Output a single non-negative integer, the infimum of the areas of all simple polygons that enclose the closed polygonal chain. It can be proven that the answer is always integer.

Examples

standard input	standard output
6 1 1 6 1 6 11 11 11 11 6 1 6	50
8 2 4 2 1 4 1 4 3 1 3 1 2 3 2 3 4	6
10 1 1 1 5 4 5 4 3 2 3 2 4 1 4 1 2 4 2 4 1	8

Note

In examples 1 and 3, there are no simple polygons with areas exactly equal to 50 and 8, respecively; however, there exist simple polygons with areas arbitrarily close to these values.

Problem H. AND = OR

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 megabytes

Let's call an array of integers $[b_1, b_2, \ldots, b_m]$ **good**, if we can partition **all** of its elements into 2 **non-empty** groups, such that the bitwise `AND` of the elements in the first group is equal to the bitwise `OR` of the elements in the second group. For example, the array $[1, 7, 3, 11]$ is **good**, as we can partition it into $[1, 3]$ and $[7, 11]$, where 1 `OR` $3 = 3$, and 7 `AND` $11 = 3$.

You are given an array $[a_1, a_2, \ldots, a_n]$, and have to answer q queries of form: is subarray $[a_l, a_{l+1}, \ldots, a_r]$ **good**?

Input

The first line of the input contains two integer n, q ($1 \le n \le 10^5$, $1 \le q \le 10^5$) — the length of the array and the number of the associated queries.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 2^{30} - 1$) — the elements of the array.

The i-th of the next q lines contains 2 integers l_i, r_i ($1 \le l_i \le r_i \le n$) — describing the i-th query.

Output

For each query, output `YES`, if the correspondent subarray is good, and `NO`, if it's not.

Example

standard input	standard output
5 15 0 1 1 3 2 1 1 1 2 1 3 1 4 1 5 2 2 2 3 2 4 2 5 3 3 3 4 3 5 4 4 4 5 5 5	NO NO YES YES YES NO YES YES YES NO NO YES NO NO NO

Problem I. Modulo Permutations

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Given a natural number n, count the number of permutations $(p_1, p_2, \ldots, p_n)$ of the numbers from 1 to n, such that for each i $(1 \le i \le n)$, the following property holds: $p_i \bmod p_{i+1} \le 2$, where $p_{n+1} = p_1$.

As this number can be very big, output it modulo $10^9 + 7$.

Input

The only line of the input contains the integer n $(1 \le n \le 10^6)$.

Output

Output a single integer — the number of the permutations satisfying the condition from the statement, modulo $10^9 + 7$.

Examples

standard input	standard output
1	1
2	2
3	6
4	16
5	40
1000000	581177467

Note

For example, for $n = 4$ you should count the permutation $[4, 2, 3, 1]$, as $4 \bmod 2 = 0 \le 2$, $2 \bmod 3 = 2 \le 2$, $3 \bmod 1 = 0 \le 2$, $1 \bmod 4 = 1 \le 2$. However, you shouldn't count the permutation $[3, 4, 1, 2]$, as $3 \bmod 4 = 3 > 2$ which violates the condition from the statement.

Problem J. One Piece

Input file:	standard input
Output file:	standard output
Time limit:	3 seconds
Memory limit:	256 megabytes

The Goa Kingdom is a network of n islands (identified by numbers from 1 to n), connected by $n - 1$ bidirectional bridges. The network is structured as a tree. Some islands contain valuable treasures, and Luffy is on a quest to find the treasures from all islands.

In order to ease the treasure hunting, he bought a detector from a local merchant. The detector should have shown the distance from each island to the closest treasure (in number of bridges); however, it seems to be horribly broken, and shows the distance from each island to the **farthest** treasure instead!

Nonetheless, he kept the distances that his broken detector showed for each of the islands, hoping that maybe not everything is lost. He now wonders which islands have a higher chance of containing a treasure.

Your task is to help Luffy by arranging the n islands in order, from highest to lowest probability of containing a treasure, given that he now knows the distances shown by the detector for each of the n islands. Initially, you can assume that each of the islands independently had a 50% chance of containing a treasure; in other words, every subset of islands was equally likely to be the subset of the treasure islands.

Input

The first line of the input contains n ($1 \le n \le 250\,000$), the number of islands. The following $n - 1$ lines describe the bridges. Each bridge connects two distinct islands. Finally, the last line contains n non-negative integers, the distances (in number of bridges) shown on Luffy's detector for each of the islands.

It is guaranteed that there is at least one **non-empty** subset that is consistent with the input data.

Output

Output a permutation of size n, the order of the islands from highest to lowest probability of containing a treasure. If two islands have the same probability of containing a treasure, output them in increasing order of their ids.

Examples

standard input	standard output
5 1 2 1 3 2 4 2 5 2 2 3 3 3	3 4 5 1 2
4 2 1 3 2 3 4 1 0 1 2	2 1 3 4

Note

In the first example, island 3 must contain a treasure, as it is the only one at distance 2 from island 2. Islands 4 and 5 have probability 2/3 each, while islands 1 and 2 have probability 1/2.

In the second example, the only possible scenario is that island 2 is the only one containing a treasure.

Problem K. Codenames

Input file:	`standard input`
Output file:	`standard output`
Time limit:	9 seconds
Memory limit:	1024 megabytes

The rules of this game may differ slightly from the official game.

Karen and her friends are competing in a high-stakes board game championship, playing the popular game *Codenames*. Codenames is a game with two opposing teams: the red team and the blue team. Karen is a member of the **red** team.

The game is played on a board of size 5×5 where each of the 25 cells is (secretly) assigned one of four kinds. There is a fixed number of cells of each kind on the board:

- 9 *red* cells (`r`);
- 8 *blue* cells (`b`);
- 7 *neutral* cells (`n`);
- 1 *assassin* cell (`x`).

The true kinds of the cells are known only to one player of each team (called the "spymaster"). The other players initially see only a 5×5 grid full of covered cells. The cells will get revealed as the game progresses. Each covered cell contains the name of an object (which turns out to be irrelevant to this problem).

Luckily, Karen is the spymaster of her team, so she knows the true configuration of the board. Her responsibility is to help her teammates discover the red cells, while keeping them away from all the other cells (especially the *assassin* cell). The way she can do that is by announcing a **hint** consisting of:

- a *valid* word w (from a dictionary of n words);
- a positive number g (the number of guesses that their teammates should make).

Her teammates will then try to guess as many red cells as possible, given the hint. They start by making a first guess, and reveal one of the cells. If the revealed cell is red, they can continue guessing; otherwise, their turn stops, and the other team starts their turn. A team wins the game if all the cells with their corresponding color are revealed, or if the other team revealed the *assassin* cell.

To illustrate this, let's consider the state of the game below (the one corresponding to the example). The left picture shows Karen's view of the board. The middle picture shows her teammates' view of the board. Notice that some of the cells are covered for Karen's teammates, and only Karen knows their true kinds. The meaning of the right picture will be explained later in the statement.

Originally, Karen's goal was to tell hints that relate to the names of objects described in some of the red cells, so that the teammates will know to reveal only those cells. However, she soon realized that the

game is not going great, and that the blue team might beat them in their next turn. Thankfully, she and her friends have devised a secret "emergency cheating scheme" for these situations specifically.

They start by assigning a *letter* to each of the 25 cells, in row-major order (as illustrated above, in the right picture). Then, when Karen announces a word w and a number g, her teammates would do the following:

1. Go through each of the letters w_i of the word w in order;
2. If w_i is not assigned to any cell or the assigned cell of w_i is already revealed, then do nothing; otherwise, guess the cell corresponding to w_i.

The teammates repeat this procedure until they reveal all the correct cells, they make a mistake (reveal a non-red cell), they already made all g guesses, or all the letters of w are revealed.

In the example above, Karen could announce "`actor 2`" to her team. Her team would first guess cell $(1, 1)$ (corresponding to letter `a`), skip letter `c` as the cell $(1, 3)$ is already revealed, and then guess cell $(4, 5)$, winning the game (as the other red cells are already revealed).

Karen wants to use this scheme to win the game **in one turn**. She knows the dictionary of all the n *valid* words, as well as the current state of the game. Find out some hint that she should announce to her team to grant them the victory!

There are q different scenarios that you need to solve. The dictionary is the same for all scenarios, but the board configurations might differ.

Input

The first line of the input contains a positive integer n ($1 \le n \le 100\,000$), the number of valid words. Each of the following n lines contain a single string of at least one and at most 30 letters, representing the words in the dictionary.

The following line contains a positive integer q ($1 \le q \le 100\,000$), the number of scenarios. Then, q lines follow, each describing a board. Each board is represented by a 5×5 grid of letters from the set $\{\mathtt{r}, \mathtt{b}, \mathtt{n}, \mathtt{x}\}$ (red/blue/neutral/assassin). If the letter is uppercase, it means that the cell is already revealed (otherwise the cell is covered). There is at least one blue and one red covered cell, and the assassin cell is always covered; in other words, the state always indicates a game that has not finished yet.

Output

For each of the q scenarios, output the hint consisting of a word w and a number g ($1 \le g \le 9$) which gives Karen's team the victory. If no such hint can be announced for the specific scenario, print a single word "`IMPOSSIBLE`" (without quotes) instead of the hint. If multiple solutions exist, any one is accepted. The answers for different scenarios should be printed on separate lines.

Example

standard input	standard output
3 actor cheat zeta 1 rBBnR NRnbB nRRnR NRxBr nBRbB	actor 2

Note

Note that Karen couldn't have announced, for example, `cheat 3`, as her team would end up revealing the cell at position $(2, 3)$ and ending their turn. Some other correct solutions would be `zeta 2` or `actor 4`.

Problem L. Neo-Robin Hood

Input file:	standard input
Output file:	standard output
Time limit:	4 seconds
Memory limit:	256 megabytes

There are n aspiring politicians in Neverland. They are wealthy, but not wealthy enough to gain political influence. Since Neverland is a financially transparent haven, we know the bank statements of each politician: the i-th politician ($1 \le i \le n$) has m_i dollars, and needs p_i more dollars to achieve his political goals.

You are the infamous modern superhero Neo-Robin Hood. You earn your living by stealing from the rich and wealthy, in order to help... well, whoever promises to help you back. For each of the n politicians you can chose to do one of the following:

1. Steal his m_i dollars;
2. Do nothing to him;
3. Help him gain political influence, by giving him p_i dollars.

But your services don't come for free. Once you help a politician gain political influence, he is bound to help you cover-up one of your thefts so that you won't get in trouble – for instance, by providing an alibi. In turn, you are also bound to not steal his money in the future.

Initially you start with no money. Your task is to rob as many politicians as possible; however, you can't afford to get caught, so you need a politician to account for each crime you commit.

What is the maximum number of people you can rob?

Input

The first line of the input contains a positive integer n ($1 \le n \le 100\,000$), the number of politicians. The second line of the input contains n positive integers m_i ($1 \le m_i \le 10^9$, for all $1 \le i \le n$). The third line of the input contains n positive integers p_i ($1 \le p_i \le 10^9$, for all $1 \le i \le n$).

Output

Output a single non-negative integer, the maximum number of people that you can steal from.

Note that you do not have to maximize your own wealth, but rather the number of people that you are stealing from.

Examples

standard input	standard output
5 2 3 4 5 6 1 2 3 4 5	2
4 1 2 4 2 5 6 9 7	0
4 9 19 6 5 20 3 16 19	1

Problem M. Mistake

Input file:	standard input
Output file:	standard output
Time limit:	3 seconds
Memory limit:	256 megabytes

As an apprentice algorithms enthusiast, it is not a great surprise that Mike struggles to cope with overly complex systems. Unfortunately, this turned out to be a big problem in the company he is currently interning.

Mike's assigned project involves tinkering with the company's Intelligent Cluster for Parallel Computation. This is just a fancy name; in reality, the system is just a simple job scheduler, handling a total of n jobs. Some jobs might depend on successful execution of other jobs before being able to be executed. There are m such dependencies in total.

It is guaranteed that there are no (direct or indirect) circular dependencies between jobs.

When a run is started, the systems intelligently picks an order to execute these jobs so that all the dependencies are met (the order may change between different runs). After picking a valid ordering, it starts executing each of the n jobs in that order. When the system starts executing a job, it prints the id of the job to a log file.

Unfortunately, today was Mike's first day interning at the company and he wasn't very cautious. Consequently, he accidentally ran the system k times in parallel. The system started erratically launching jobs and printing to the log file. Now the log file contains $n \cdot k$ ids of all the jobs that were executed. The job ids from the same run have been printed in the order they were executed, but the outputs from different runs may appear interweaved arbitrarily.

Your task is to figure out which jobs were executed in each of the k runs from the information inside the log file.

Input

The first line of the input will contain three integers n, k, m ($1 \le n, k \le 500\,000$, $0 \le m \le 250\,000$, $n \cdot k \le 500\,000$), the number of jobs in the system, the number of runs Mike had triggered, and the number of dependencies.

The following m lines will contain a pair a_i, b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$, for all $1 \le i \le m$) describing a dependency of kind: "job a_i must be executed before job b_i".

Finally, the last line of the input contains $n \cdot k$ integers c_i ($1 \le c_i \le n$, for all $1 \le i \le n \cdot k$), the job ids that have been printed in the log file, in order.

Output

Output a single line consisting of $n \cdot k$ integers r_i ($1 \le r_i \le k$, for all $1 \le i \le n \cdot k$), the run id corresponding to each of the jobs in the log file. More specifically, r_i should be the run id corresponding to the i-th job, as it appears in the log file.

If multiple solutions are possible, any one is accepted. It is guaranteed that the input data is valid and that a solution always exists.

Example

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