Arranging Products

Statement

Fluffy the Hamster has recently opened a new store. There are a great variety of products sold in his store.

There are N products on display, numbered from 1 to N. The ith product has K_i labels, $L_{i,1}, L_{i,2}, \ldots, L_{i,K_i}$. Each label is a **positive** integer.

Fluffy the Hamster loves seeing many different rearrangements of his products. Today, he has enlisted your help to arrange his products in many different ways. In particular, you need to process Q of the following type of operation:

- $n\ l_1\ c_1\ l_2\ c_2\ \ldots l_n\ c_n$: **Stable** sort the **current arrangement** of products in **ascending** order according to the labels $l_1, l_2, \ldots, l_n$. Label l_i should be ordered according to c_i. If c_i is `<`, then label l_i should be compared according to the ascending order, but if label c_i is `>`, label l_i should be compared according to the descending order. If a product does not have a particular label, the value of that label should be treated as 0.

  (Please refer to the "Examples" section for an illustration.)

Help Fluffy arrange his products!

Constraints

- $1 \leq Q \leq 100$
- $1 \leq N \leq 10^3$
- $1 \leq K_i \leq 100$
- $1 \leq L_{i,j} \leq 10^9$
- For each operation, the values of l_i are all distinct
- For each operation, $1 \leq n \leq 100$

Input

The first line of the input contains a single integer N.

Each of the next N lines contains an integer K_i, followed by K_i space separated integers $L_{i,1}, \ldots, L_{i,K_i}$.

The next line contains a single integer Q.

The next Q lines contain one operation each, in the format above.

Output

For each operation, output the IDs of the products in the correct order after the rearrangement.

Examples

Sample Input	Expected Output
4 5 10 20 30 40 50 3 40 20 10 2 70 40 4 30 40 20 10 3 1 3 < 1 2 > 3 5 > 2 < 1 >	3 2 4 1 3 4 2 1 1 2 3 4

In the example above, there are 4 products. The labels of each of the 4 products are as follows.

Product 1	[10, 20, 30, 40, 50]
Product 2	[40, 20, 10]
Product 3	[70, 40]
Product 4	[30, 40, 20, 10]

In the first query, we should sort products according to the 3rd label of each product, and the 3rd label should be compared according to the ascending order.

Looking only at the 3rd label of each product, we have:

Product 1	[30]
Product 2	[10]
Product 3	[0]
Product 4	[20]

So the sorted order of products should be 3 2 4 1.

In the second query, we should sort products according to the 2nd label of each product, and the 2nd label should be compared according to the descending order.

Continuing from the ordering in the first query and looking only at the 2nd label of of each product, we have

Product 3	[40]
Product 2	[20]
Product 4	[40]
Product 1	[20]

After (stable) sorting, we have the order 3 4 2 1. Note that other orderings such as 4 3 2 1 or 3 4 1 2 are incorrect, as they imply that the sorting is not stable.

In the third query, we should first compare the 5th label in descending order, and if tie, the 2nd label in ascending order, and if still tie, the 1st label in descending order. Continuing from the ordering in the second query and looking at only the 5th, 2nd and 1st labels, we have

Product 3	[0, 40, 70]
Product 4	[0, 40, 30]
Product 2	[0, 20, 40]
Product 1	[50, 20, 10]

After stable sorting, we have the order 1 2 3 4.

Notes

1. A skeleton file has been given to help you. You should not create a new file or rename the file provided. You should develop your program using this skeleton file.
2. You are free to define your own helper methods and classes (or remove existing ones) if it is suitable but you must put all the new classes, if any, in the same skeleton file provided.

Skeleton File

You are given the skeleton file `Products.java`. You should see the following contents when you open the file:

```
/**
 * Name          :
 * Matric. No    :
*/

import java.util.*;

public class Products {
    private void run() {
        // implement your "main" method here
    }

    public static void main(String args[]) {
        Products runner = new Products();
        runner.run();
    }
}
```