LRU Algorithm

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

In computing, cache algorithms (also frequently called cache replacement algorithms or cache replacement policies) are optimizing instructions, or algorithms, that a computer program or a hardware-maintained structure can utilize to manage a cache of information stored on the computer. Caching improves performance by keeping recent or often-used data items in memory locations that are faster or computationally cheaper to access than normal memory stores. When the cache is full, the algorithm must choose which items to discard to make room for the new ones.

One of the most famous cache algorithms is called the *Least Recently Used* (LRU) algorithm. In LRU, items in the cache are typically maintained with a linked list. When an item is accessed, it is removed from the linked list (if present), and then inserted to the front of the list. The front of the linked list is the *Most Recently Used* item. Because used items are continually being moved to the front of the linked list, the least used ones naturally end up at the back of the list. When there's not enough room in the cache, the item at the back of the linked list is discarded.

In this problem, each item in the cache is given a unique positive integer identifier for convenience. For example, let's consider the access sequence {4, 3, 4, 2, 3, 1, 4} and assume that the capacity of the cache is 3. The following table shows the content of the linked list.

Step	**Content of the linked list**	**Operation**
1		Acess 4
2	4	Acess 3
3	$3 \to 4$	Acess 4 (already in cache)
4	$4 \to 3$	Acess 2
5	$2 \to 4 \to 3$	Acess 3 (already in cache)
6	$3 \to 2 \to 4$	Acess 1 (pop 4 from back)
7	$1 \to 3 \to 2$	Acess 4 (pop 2 from back)
8	$4 \to 1 \to 3$	

Now, you want to do some memory optimization for your program. To that end, you are going to run some experiments. Assume that the LRU algorithm is used to manage the cache. The access sequence of your program is known and fixed, and you want to run q experiments about the cache. In the i-th ($1 \le i \le q$) experiment, the capacity of the cache is set to m_i and you have a linked list x_i. You want to find out if at some point during the execution of your program, the linked list maintained by the LRU algorithm is exactly the same as x_i.

Just before you start running your program to experiment, one of your friends challenged you to find out the results without actually running the experiment q times. He asks you to write a simple program to find out the answer, can you do it?

Input

The input contains multiple cases. The first line of the input contains a single positive integer T, the number of cases.

For each case, the first line of the input contains two integers n, q ($1 \le n \le 5\,000, 1 \le q \le 2\,000$), the length of the access sequence and the number of experiments. The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le i \le n, 1 \le a_i \le n$), where the i-th integer denotes the identifier of the item accessed in the i-th operation.

The following q lines each describes an experiment. The i-th ($1 \le i \le q$) line contains an integer m_i ($1 \le m_i \le n$), the capacity of the cache, followed by m_i integers $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$

($1 \le j \le m_i, 0 \le x_{i,j} \le n$), where the j-th integer denotes the identifier of the j-th item in the linked list x_i. Note that the number of items in x_i may be less than m_i. Let L_i be the length of x_i, then the last $m_i - L_i$ integers in the input will be equal to 0, those zeroes should be ignored, while the other integers will be positive.

It's guaranteed that the sum of n over all cases does not exceed 20 000, the sum of q over all cases does not exceed 20 000 and the sum of m_i over all cases does not exceed $2 \cdot 10^6$.

Output

For each case, print q lines, where the i-th line describes the result of the i-th experiment. If at some point, the linked list maintained by the LRU algorithm is equal to the given list x_i, print the string "`Yes`" (without quotes). Otherwise, print the string "`No`" (without quotes).

Example

standard input	standard output
1 7 5 4 3 4 2 3 1 4 1 4 2 2 3 3 3 2 1 4 4 1 3 2 4 3 4 0 0	Yes No No Yes Yes