Sergey recently studied lock-free data structures. He particularly liked the data structure called Lock-Free Stack.

So, lock-free stack is basically an ordinary stack, which can be used by multiple threads of the same program. There are **N** threads, which do push and pop the numbers to this stack simultaneously.

In order to check his knowledge, Sergey implemented this data structure. But he was still unsure, whether his code works correct or not. So he made the test case of the following kind:

- For every thread (say, the $\mathbf{i^{th}}$), there is a list of $\mathbf{A_i}$ numbers, which will be pushed to the stack by this thread in this order (so, first, the first number from this list will be added, then the second one, and so on).

So Sergey runs the program, the numbers get pushed to the stack from all the threads simultaneously. When all the threads are done with it, he wanted to pop all the numbers from the stack and to check the correctness of the output.

But it appeared to be not so easy problem, because in general, there could be a few different correct sequences he could obtain, because the order in which the processes add the numbers is not strictly defined. Moreover, the processes can interrupt in between.

For example, even if he had only two threads and for each of them, there was a list of **1** unique number, then there can be two valid orders: either the first thread added its' number first, or it was the second one.

The another example could be the following: if there are also two thread, the first one having the list **(1, 2)** and the second one having the list **(3, 4)**, then after doing all pops even the sequence **(4, 2, 3, 1)** is correct, because in this case:

- First, the first thread added the first number **1** from its' list;
- Then, the second thread added the first number **3** from its' list;
- Then, the first thread added the second number **2** from its' list;
- Then, the second thread added the second number **4** from its' list;

So since it is a LIFO (last in, first out) stack, after the pops we get the sequence **(4, 2, 3, 1)**.

You are given the number of the threads and the list of integers to be added in order for each of the threads. You are also given a sequence. Determine, whether this sequence could be obtained after the process described above.

Input

The first line of the input contains an integer **T** denoting the number of test cases. The description of **T** test cases follows.

The first line of each test case contains a single integer **N** denoting the number of threads.

Each of the following **N** lines contains the description of the numbers to be pushed in the following way: the first number in the list is A_i; the following numbers are the numbers $B_{i, 1}$, $B_{i, 2}$, ..., B_{i, A_i} denoting the numbers that will be added to the stack (in this order) by the i^{th} thread.

The last line on the test case contains $A_1+A_2+...+A_N$ integer numbers, denoting the sequence Sergey got after he popped all the numbers from the stack.

Output

For each test case, output a single line containing **Yes**, if Sergey could have got this sequence of numbers and **No** otherwise.

Constraints

- $1 \le T \le 15$
- $1 \le A_i$
- $1 \le B_{i, j} \le 1000$
- Let's denote $P = (A_1 + 1) \times (A_2 + 1) \times ... \times (A_N + 1)$
- $1 \le$ sum of $P \le 10^6$

Subtaks

- **Subtask #1 (33 points)**: $1 \le$ sum of $P \le 1000$
- **Subtask #2 (11 points)**: $N = 1$
- **Subtask #3 (56 points)**: no additional constraints

Example

```
Input:
2
2
2 1 2
2 3 4
4 3 2 1
2
2 1 2
2 3 4
4 1 2 3

Output:
Yes
No
```

Explanation

Example case 1. First, the integers of the first thread were added: **1, 2**. Then, the integers of the second thread were added: **3, 4**. So the sequence of the additions is **1, 2, 3, 4**. Since it is a LIFO stack, when we pop, we get **4, 3, 2, 1**. This way, the given sequence is reachable.

Example case 2. We need to get the following sequence of pushes to the stack: **3, 2, 1, 4**. So, **2** should be added before **1**. But only the first thread in the example can add these integers, and it will add **2** only **after** it has added **1**. So, the given configuration is impossible to obtain.