Lock-Free Stack

Sergey đang học về cấu trúc dữ liệu lock-free. Anh ta đặc biệt thích thú với kiểu cấu trúc Lock-Free Stack.

Lock-free stack là một stack bình thường, nó có thể được sử dụng bởi nhiều luồng của cùng một chương trình. Có **N** luồng, chúng có thể đẩy hoặc lấy các số trong stack cùng một lúc.

Để kiểm tra kiến thức của mình, Sergey cài đặt cấu trúc dữ liệu này. Nhưng anh ta vẫn không chắc chắn liệu code của mình có chính xác hay không. Nên anh ta tạo ra loại test sau:

- Với mỗi luồng (gọi là luồng thứ i), sẽ có danh sách $\mathbf{A_i}$ số được đẩy vào stack theo thứ tự xuất hiện (đầu tiên, số thứ nhất trong list sẽ được thêm, sau đó là số thứ hai rồi cứ tiếp tục như vậy).

Sergey chạy chương trình, các con số được đẩy vào stack từ tất cả các luồng cùng một lúc. Khi tất cả các luồng đã thực hiện xong, anh ta muốn lấy ra tất cả các số trong stack để kiểm tra tính đúng đắn của output.

Nhưng nó lại xuất hiện một vấn đề không hề đơn giản, bởi vì thông thường, có vài dãy số khác nhau đều đúng, do thứ tự các luồng đẩy số vào không được định nghĩa rõ ràng. Ngoài ra, các luồng có thể gián đoạn lúc đang chạy.

Ví dụ, nếu anh ta chỉ có hai luồng và mỗi luồng có một danh sách gồm **1** số riêng biệt, khi đó có hai thứ tự đúng: luồng đầu tiên đẩy số vào đầu hoặc nó có thể đẩy vào thứ hai.

Một ví dụ khác: nếu có hai luồng, luồng thứ nhất có danh sách **(1, 2)** và luồng thứ hai có danh sách **(3, 4)**, khi đó lấy ra tất cả các phần tử được dãy **(4, 2, 3, 1)** là đúng, bởi trong trường hợp:

- Ban đầu, luồng đầu tiên thêm số đầu tiên là **1** từ danh sách của nó.
- Sau đó, luồng thứ hai thêm số đầu tiên là **3** từ danh sách của nó.
- Sau đó, luồng đầu tiên thêm số thứ hai là **2** từ danh sách của nó
- Sau đó, luồng thứ hai thêm số thứ hai là **4** từ danh sách của nó.

Bởi tính chất LIFO (vào đầu ra cuối), khi lấy các phần tử ra ta được dãy **(4, 2, 3, 1)**.

Bạn được cho số lượng các luồng và danh sách các số trong luồng được thêm vào theo thứ tự. Bạn cũng được cho một dãy số. Xác định xem dãy số đó có thể được tạo ra từ chương trình được miêu tả như trên hay không.

Dữ liệu vào:

- Dòng đầu tiên chứa một số nguyên **T** – số lượng test. Các test được miêu tả như sau:
- Dòng đầu tiên của mỗi test chứa một số nguyên **N** là sô lượng luồng.

- **N** dòng tiếp theo, mỗi dòng miêu tả các số được đẩy vào: số đầu tiên là $\mathbf{A_i}$; theo sau là các số $\mathbf{B_{i,1}}$, $\mathbf{B_{i,2}}$, ..., $\mathbf{B_{i,Ai}}$ thể hiện các số được thêm vào stack (theo thứ tự) của luồng thứ **i**
- Dòng cuối cùng chứa $\mathbf{A_1+A_2+...+A_N}$ số nguyên thể hiện dãy số Sergey có được sau khi lấy tất cả các số trong stack.

Dữ liệu ra:

- Với mỗi test, in ra một dòng chứa **Yes** nếu Sergey có thể lấy được dãy số đó, ngược lại in ra **No**.

Ràng buộc:

- $\mathbf{1 \le T \le 15}$
- $\mathbf{1 \le A_i}$
- $\mathbf{1 \le B_{i,j} \le 1000}$
- Gọi $\mathbf{P = (A_1 + 1) \times (A_2 + 1) \times ... \times (A_N + 1)}$
- $\mathbf{1 \le}$ tổng $\mathbf{P \le 10^6}$

Subtasks:

- **Subtask #1 (33 điểm)**: $\mathbf{1 \le}$ tổng $\mathbf{P \le 1000}$
- **Subtask #2 (11 điểm)**: $\mathbf{N = 1}$
- **Subtask #3 (56 điểm)**: không có ràng buộc thêm.

Ví dụ:

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
```

Giải thích:

Ví dụ 1. Đầu tiên, các số trong luồng thứ nhất được thêm vào: **1, 2**. Sau đó, các số trong luồng thứ hai được thêm vào: **3, 4**. Do đó dãy được thêm vào là **1, 2, 3, 4**. Bởi tính chất LIFO của stack, khi lấy ra, ta được **4, 3, 2, 1**. Bằng cách này, có thể đạt được dãy số đã cho.

Ví dụ 2. Ta cần dãy được thêm vào stack như sau: **3, 2, 1, 4**. Do đó, **2** cần được thêm vào trước **1**. Nhưng chỉ có luồng thứ nhất có thể thêm những số đó, và nó thêm **2** sau khi thêm **1**. Vì vậy, dãy số đã cho không thể đạt được.