Game of Stacks

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

You have n stacks $r_1, r_2, \ldots, r_n$. Each stack contains some positive integers ranging from 1 to n.

Define the following functions:

```
function init(pos):
    stacks := an array that contains n stacks r[1], r[2], ..., r[n]
    return get(stacks, pos)

function get(stacks, pos):
    if stacks[pos] is empty:
        return pos
    else:
        new\_pos := the top element of stacks[pos]
        pop the top element of stacks[pos]
        return get(stacks, new\_pos)
```

You want to know the values returned by `init(1)`, `init(2)`, ..., `init(n)`.

Note that, during these calls, the stacks $r_1, r_2, \ldots, r_n$ don't change, so the calls `init(1)`, `init(2)`, ..., `init(n)` are independent.

Input

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

Each of the following n lines contains several integers. The first integer k_i ($0 \le k_i \le 10^5$) represents the number of elements in the i-th stack, and the following k_i positive integers $c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}$ ($1 \le c_{i,j} \le n$) represent the elements in the i-th stack. $c_{i,1}$ is the bottom element.

In each test, $\sum k_i \le 10^6$.

Output

You need to output n values, the i-th of which is the value returned by `init(i)`.

Examples

standard input	standard output
3 3 1 2 2 3 3 1 2 3 1 2 1	1 2 2
5 5 1 2 4 3 4 6 1 2 5 3 3 4 6 1 1 4 4 4 2 9 3 1 4 2 3 5 5 1 2 4 4 4 1 3	1 1 1 1 1

Note

In the first example:

- When you call `init(1)`, set `stacks := [[1,2,2],[3,1,2],[1,2,1]]`, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1,2],[3,1,2],[1,2,1]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2],[3,1],[1,2,1]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2],[3],[1,2,1]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1],[3],[1,2,1]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 3`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1],[],[1,2,1]]$, and then call `get(stacks, 3)`.
  - `stacks[3]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[3]`, which makes `stacks` become $[[1],[],[1,2]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[],[],[1,2]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is empty, return 1.
- When you call `init(2)`, set `stacks := [[1,2,2],[3,1,2],[1,2,1]]`, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2,2],[3,1],[1,2,1]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2,2],[3],[1,2,1]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1,2],[3],[1,2,1]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 3`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2],[],[1,2,1]]$, and then call `get(stacks, 3)`.
  - `stacks[3]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[3]`, which makes `stacks` become $[[1,2],[],[1,2]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1],[],[1,2]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is empty, return 2.
- When you call `init(3)`, set `stacks := [[1,2,2],[3,1,2],[1,2,1]]`, and then call `get(stacks, 3)`.
  - `stacks[3]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[3]`, which makes `stacks` become $[[1,2,2],[3,1,2],[1,2]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1,2],[3,1,2],[1,2]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2],[3,1],[1,2]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 1`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1,2],[3],[1,2]]$, and then call `get(stacks, 1)`.
  - `stacks[1]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[1]`, which makes `stacks` become $[[1],[3],[1,2]]$, and then call `get(stacks, 2)`.
  - `stacks[2]` is not empty, set `new_pos := 3`, and pop the top element of `stacks[2]`, which makes `stacks` become $[[1],[],[1,2]]$, and then call `get(stacks, 3)`.

– `stacks[3]` is not empty, set `new_pos := 2`, and pop the top element of `stacks[3]`, which makes `stacks` become [[1], [], [1]], and then call `get(stacks, 2)`.
– `stacks[2]` is empty, return 2.