Buggy Keyboard

Gabrielle has an assignment for her Algorithmic Problem Solving course due in a couple of hours. She opened up her computer which she purchased recently and started working using her favorite editor when she discovered something strange. Generally, pressing the backspace key should erase a character to the left of the cursor. But on her new computer, pressing that key produced a character `<`. Since the assignment is due in couple of hours, she does not have time to call customer support or replace the computer so she decides to temporarily find a way around the problem with the help of a program.

Help Gabrielle write a program which takes the string written on her computer and outputs the string that Gabrielle actually intended to write. It can be assumed that she never needs to output the character `<`. Also you can be assured that she will never press backspace on an empty line.

Input

A string `s` containing text written on Gabrielle's computer. The length of `s` is less than 10^6 The string will contain only lower case letters, spaces, and the character `<`.

Output

A string containing text that was actually intended. Note that there should be no spaces or new line after the last character in the output

Example 1

```
Input:
a<bcd<

Output:
bc
```

Example 2

```
Input:
prind<tf

Output:
printf
```

Example 3

```
Input:
as<d<<

Output:
```

Daily Prize

Trader Jane's has a new promotion to attract even more customers to already crowded supermarket.

To participate in a daily drawing of the prize a customer needs to fill out a form with their name, phone number and the amount that they paid for their groceries.

At the end of the day, Trader Jane's manager picks two forms from among the completed ones:

- first is the one that has the highest amount paid
- second is the one that has the lowest amount paid The person who paid the highest amount gets the prize equal to the difference between their bill and the lowest bill.

Given how busy Trader Jane's supermarket is, you can be certain that at the end of each day there are at least two bills to select (usually there are many many more).

The selected forms are discarded, but the other ones remain in the pool for the next day, so each customer has a chance to be selected as the prize winner on the day of their purchase, or any day after.

Your task is to compute how much money Trader Jane's pays out in prizes.

Input

The input contains an integer `n`, `1 <= n <= 5,000` on the first line - this is the number of days in the promotion. Each of the next `n` lines contains a sequence of non-negative integers separated by whitespace. The first number on each line indicates the number of forms submitted on that day, `0 <= k <= 100,000`. The next `k` numbers specify the bill amounts on the forms entered for the daily drawing on that day. Each amount is guaranteed to be no larger than `1,000,000`. The total amount of all bills is no larger than `1,000,000`.

Output

Print one number that is the sum of all the prizes that Trader Jane's pays out during the promotion followed by a newline.

Example 1

```
Input:
5
3 1 2 3
2 1 1
4 10 5 5 1
0
1 2

Output:
19
```

Example 2

```
Input:
2
2 1 2
2 1 2

Output:
2
```

Queuing With Friends

Standard *Queues* are well known to most computer science students. It's a first-in-first-out data structure. Michelle is already familiar with the standard queue. But she is assigned to implement a variant of the standard queue that allows all friends to queue together, even if they do not arrive at the same time to join the queue.

In this friends-queue each person belongs to exactly one fried-group. If a person arrives to be added to the queue and none of their friends are already in the queue, then they join the queue at the very end. Otherwise, if some of their friends are already in the queue, they join that friend-group (within the friend-group, they queue up after the last of their friends).

Dequeuing works the same as in the standard queue.

The other change from the standard queue is the naming convention. For some unknown reason, Michelle is supposed to call it *push* when the new element/friend is added to the queue (rather than the typical *enqueue*), an to use *pop* as the name of the operation that dequeues an element/friend.

Help Michelle by implementing the friend-queue.

Input

The 1st line contains an integer `1 <= n <= 1,000` equal to the number of friend-groups.

Then `n` group descriptions follow. Each friend-group is described on one line by an integer `k` (the number of friends in this friend-group) followed by `k` integers identifying all the friends.

Friends in different groups are identified by their unique id's in the range `[0 , 999,999]` . A single friend-group may consist of up to `1,000` friends.

Finally, a list of test instructions follows. There are 3 different kinds of instructions:

- `Push id` : add the person with the given id to the friend-queue.
- `Pop` : print the id of the first element/friend and remove then from the friend-queue.
- `Shutdown` : stop your processing the friend-queue (end of input).

There may be up to `200,000` test instructions in the input file, so the implementation of the group queue should be efficient: both adding and removing of an element should only take a constant time.

Output

For each `Pop` instruction, print an integer, the id of the element/friend which is removed, followed by a newline.

Example 1

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 4
Push 2
Push 5
Push 3
Push 6
Pop
Pop
Pop
Pop
Pop
```

```
Pop
Shutdown

Output:
1
2
3
4
5
6
```

Example 2

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 2
Push 4
Pop
Pop
Pop
Push 3
Push 5
Pop
Pop
Push 6
Pop
Pop
Shutdown

Output:
1
2
4
5
3
6
```

Sia's Box

Sia has a mysterious box and she wants to play a game with you. The box supports two types of operations:

- `1 x` : Put the number x into the box.
- `2` : Take out a number from the box.

Sia will give you a sequence of operations and the results of the above `2` operations. Your task is to determine what is really hidden in the box: a stack, a queue, a max priority queue or something else.

Input

The first line of the input contains a single integer N (1 <= N <= 1000), indicating the number of operations. Each of the next N lines contains a single operation described above. For operation `2` , there is an additional number x indicating the result of that operation. The value of x in both operations satisfies 1 <= x <= 100.

Output

You should print one of the following answers on a line by itself:

- `stack` if it is certain that the box is a stack.
- `queue` if it is certain that the box is a queue.
- `priority queue` if it is certain that the box is a max priority queue.
- `impossible` if it is certain that the box cannot be any of those three data structures.
- `not sure` if the box could possibly be more than one of those three data structures.

Example 1

```
Input:
6
1 1
1 2
1 3
2 1
2 2
2 3

Output:
queue
```

Example 2

```
Input:
6
1 1
1 2
1 3
2 3
2 2
2 1

Output:
not sure
```

Example 3

```
Input:
2
1 1
2 2

Output:
impossible
```

Example 4

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

Output:
stack
```

Example 5

```
Input:
7
1 2
1 5
1 1
1 3
2 5
1 4
2 4

Output:
priority queue
```

Unique Subarray

A subarray is the sequence of consecutive elements in an array. A *unique* subarray is a subarray in which all elements are unique (i.e., no repeated values). Given an array, your task is to calculate the maximum length of a *unique* subarray.

For example [4, 3, 2, 2, 1], the maximum length of of a *unique* subarray is 3. The *unique* subarray is [4,3,2]. [3,2,2,1] is a subarray of length 4, but since 2 appears twice in this subarray, it is not a *unique* subarray.

Input

The 1st line contains an integer n (1 <= n <= 10^6), specifying the length of the array.

The following line contains n integers, in range [1, 10^9], denoting the elements of the array.

Output

The maximum length of a *unique* subarray.

Example 1

```
Input:
5
4 3 2 2 1

Output:
3
```

Example 2

```
Input:
6
1 1 1 1 1 1

Output:
1
```