Treino para Iniciantes #12

A. Registration system

5 seconds, 64 megabytes

A new e-mail service "Berlandesk" is going to be opened in Berland in the near future. The site administration wants to launch their project as soon as possible, that's why they ask you to help. You're suggested to implement the prototype of site registration system. The system should work on the following principle.

Each time a new user wants to register, he sends to the system a request with his `name`. If such a `name` does not exist in the system database, it is inserted into the database, and the user gets the response `OK`, confirming the successful registration. If the `name` already exists in the system database, the system makes up a new user name, sends it to the user as a prompt and *also inserts the prompt into the database*. The new name is formed by the following rule. Numbers, starting with 1, are appended one after another to `name` (`name1`, `name2`, ...), among these numbers the least i is found so that `name`i does not yet exist in the database.

Input

The first line contains number n ($1 \le n \le 10^5$). The following n lines contain the requests to the system. Each request is a non-empty line, and consists of not more than 32 characters, which are all lowercase Latin letters.

Output

Print n lines, which are system responses to the requests: `OK` in case of successful registration, or a prompt with a new name, if the requested name is already taken.

input
```
4
abacaba
acaba
abacaba
acab
```
output
```
OK
OK
abacaba1
OK
```

input
```
6
first
first
second
second
third
third
```
output
```
OK
first1
OK
second1
OK
third1
```

B. Plug-in

1 second, 256 megabytes

Polycarp thinks about the meaning of life very often. He does this constantly, even when typing in the editor. Every time he starts brooding he can no longer fully concentrate and repeatedly presses the keys that need to be pressed only once. For example, instead of the phrase "how are you" he can type "hhoow aaaare yyoouu".

Polycarp decided to automate the process of correcting such errors. He decided to write a plug-in to the text editor that will remove pairs of identical consecutive letters (if there are any in the text). Of course, this is not exactly what Polycarp needs, but he's got to start from something!

Help Polycarp and write the main plug-in module. Your program should remove from a string all pairs of identical letters, which are consecutive. If after the removal there appear new pairs, the program should remove them as well. Technically, its work should be equivalent to the following: while the string contains a pair of consecutive identical letters, the pair should be deleted. Note that deleting of the consecutive identical letters can be done in any order, as any order leads to the same result.

Input

The input data consists of a single line to be processed. The length of the line is from 1 to $2 \cdot 10^5$ characters inclusive. The string contains only lowercase Latin letters.

Output

Print the given string after it is processed. It is guaranteed that the result will contain at least one character.

input
```
hhoowaaaareyyoouu
```
output
```
wre
```

input
```
reallazy
```
output
```
rezy
```

input
```
abacabaabacabaa
```
output
```
a
```

C. Tea Queue

1 second, 256 megabytes

Recently n students from city S moved to city P to attend a programming camp.

They moved there by train. In the evening, all students in the train decided that they want to drink some tea. Of course, no two people can use the same teapot simultaneously, so the students had to form a queue to get their tea.

i-th student comes to the end of the queue at the beginning of l_i-th second. If there are multiple students coming to the queue in the same moment, then the student with greater index comes after the student with lesser index. Students in the queue behave as follows: if there is nobody in the queue before the student, then he uses the teapot for exactly one second and leaves the queue with his tea; otherwise the student waits for the people before him to get their tea. If at the beginning of r_i-th second student i still cannot get his tea (there is someone before him in the queue), then he leaves the queue without getting any tea.

For each student determine the second he will use the teapot and get his tea (if he actually gets it).

Input

The first line contains one integer t — the number of test cases to solve ($1 \le t \le 1000$).

Then t test cases follow. The first line of each test case contains one integer n ($1 \le n \le 1000$) — the number of students.

Then n lines follow. Each line contains two integer l_i, r_i ($1 \le l_i \le r_i \le 5000$) — the second i-th student comes to the end of the queue, and the second he leaves the queue if he still cannot get his tea.

It is guaranteed that for every $i \in [2;\ n]$ condition $l_{i-1} \le l_i$ holds.

The sum of n over all test cases doesn't exceed 1000.

Note that in hacks you have to set $t = 1$.

Output

For each test case print n integers. i-th of them must be equal to the second when i-th student gets his tea, or 0 if he leaves without tea.

input

```
2
2
1 3
1 4
3
1 5
1 1
2 3
```

output

```
1 2
1 0 2
```

The example contains 2 tests:

1. During 1-st second, students 1 and 2 come to the queue, and student 1 gets his tea. Student 2 gets his tea during 2-nd second.
2. During 1-st second, students 1 and 2 come to the queue, student 1 gets his tea, and student 2 leaves without tea. During 2-nd second, student 3 comes and gets his tea.

D. Radio Station

2 seconds, 256 megabytes

As the guys fried the radio station facilities, the school principal gave them tasks as a punishment. Dustin's task was to add comments to nginx configuration for school's website. The school has n servers. Each server has a name and an ip (names aren't necessarily unique, but ips are). Dustin knows the ip and name of each server. For simplicity, we'll assume that an nginx command is of form "`command ip;`" where `command` is a string consisting of English lowercase letter only, and `ip` is the ip of one of school servers.

Each ip is of form "`a.b.c.d`" where a, b, c and d are non-negative integers less than or equal to 255 (with no leading zeros). The nginx configuration file Dustin has to add comments to has m commands. Nobody ever memorizes the ips of servers, so to understand the configuration better, Dustin has to comment the name of server that the ip belongs to at the end of each line (after each command). More formally, if a line is "`command ip;`" Dustin has to replace it with "`command ip; #name`" where `name` is the name of the server with ip equal to `ip`.

Dustin doesn't know anything about nginx, so he panicked again and his friends asked you to do his task for him.

Input

The first line of input contains two integers n and m ($1 \le n, m \le 1000$).

The next n lines contain the names and ips of the servers. Each line contains a string `name`, name of the server and a string `ip`, ip of the server, separated by space ($1 \le |name| \le 10$, $name$ only consists of English lowercase letters). It is guaranteed that all ip are distinct.

The next m lines contain the commands in the configuration file. Each line is of form "`command ip;`" ($1 \le |command| \le 10$, `command` only consists of English lowercase letters). It is guaranteed that ip belongs to one of the n school servers.

Output

Print m lines, the commands in the configuration file after Dustin did his task.

input

```
2 2
main 192.168.0.2
replica 192.168.0.1
block 192.168.0.1;
proxy 192.168.0.2;
```

output

```
block 192.168.0.1; #replica
proxy 192.168.0.2; #main
```

input

```
3 5
google 8.8.8.8
codeforces 212.193.33.27
server 138.197.64.57
redirect 138.197.64.57;
block 8.8.8.8;
cf 212.193.33.27;
unblock 8.8.8.8;
check 138.197.64.57;
```

output

```
redirect 138.197.64.57; #server
block 8.8.8.8; #google
cf 212.193.33.27; #codeforces
unblock 8.8.8.8; #google
check 138.197.64.57; #server
```

E. Boxes Packing

1 second, 256 megabytes

Mishka has got n empty boxes. For every i ($1 \le i \le n$), i-th box is a cube with side length a_i.

Mishka can put a box i into another box j if the following conditions are met:

- i-th box is not put into another box;
- j-th box doesn't contain any other boxes;
- box i is smaller than box j ($a_i < a_j$).

Mishka can put boxes into each other an arbitrary number of times. He wants to minimize the number of *visible* boxes. A box is called *visible* iff it is not put into some another box.

Help Mishka to determine the minimum possible number of *visible* boxes!

Input

The first line contains one integer n ($1 \le n \le 5000$) — the number of boxes Mishka has got.

The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^9$), where a_i is the side length of i-th box.

Output

Print the minimum possible number of *visible* boxes.

input

```
3
1 2 3
```

output

```
1
```

input

```
4
4 2 4 3
```

output

```
2
```

In the first example it is possible to put box 1 into box 2, and 2 into 3.

In the second example Mishka can put box 2 into box 3, and box 4 into box 1.

F. Okabe and Boxes

3 seconds, 256 megabytes

Okabe and Super Hacker Daru are stacking and removing boxes. There are n boxes numbered from 1 to n. Initially there are no boxes on the stack.

Okabe, being a control freak, gives Daru $2n$ commands: n of which are to add a box to the top of the stack, and n of which are to remove a box from the top of the stack and throw it in the trash. Okabe wants Daru to throw away the boxes in the order from 1 to n. Of course, this means that it might be impossible for Daru to perform some of Okabe's *remove* commands, because the required box is not on the top of the stack.

That's why Daru can decide to wait until Okabe looks away and then reorder the boxes in the stack in any way he wants. He can do it at any point of time between Okabe's commands, but he can't add or remove boxes while he does it.

Tell Daru the minimum number of times he needs to reorder the boxes so that he can successfully complete all of Okabe's commands. It is guaranteed that every box is added before it is required to be removed.

Input

The first line of input contains the integer n ($1 \le n \le 3 \cdot 10^5$) — the number of boxes.

Each of the next $2n$ lines of input starts with a string "`add`" or "`remove`". If the line starts with the "`add`", an integer x ($1 \le x \le n$) follows, indicating that Daru should add the box with number x to the top of the stack.

It is guaranteed that exactly n lines contain "`add`" operations, all the boxes added are distinct, and n lines contain "`remove`" operations. It is also guaranteed that a box is always added before it is required to be removed.

Output

Print the minimum number of times Daru needs to reorder the boxes to successfully complete all of Okabe's commands.

input

```
3
add 1
remove
add 2
add 3
remove
remove
```

output

```
1
```

input

```
7
add 3
add 2
add 1
remove
add 4
remove
remove
remove
add 6
add 7
add 5
remove
remove
remove
```

output

```
2
```

In the first sample, Daru should reorder the boxes after adding box 3 to the stack.

In the second sample, Daru should reorder the boxes after adding box 4 and box 7 to the stack.