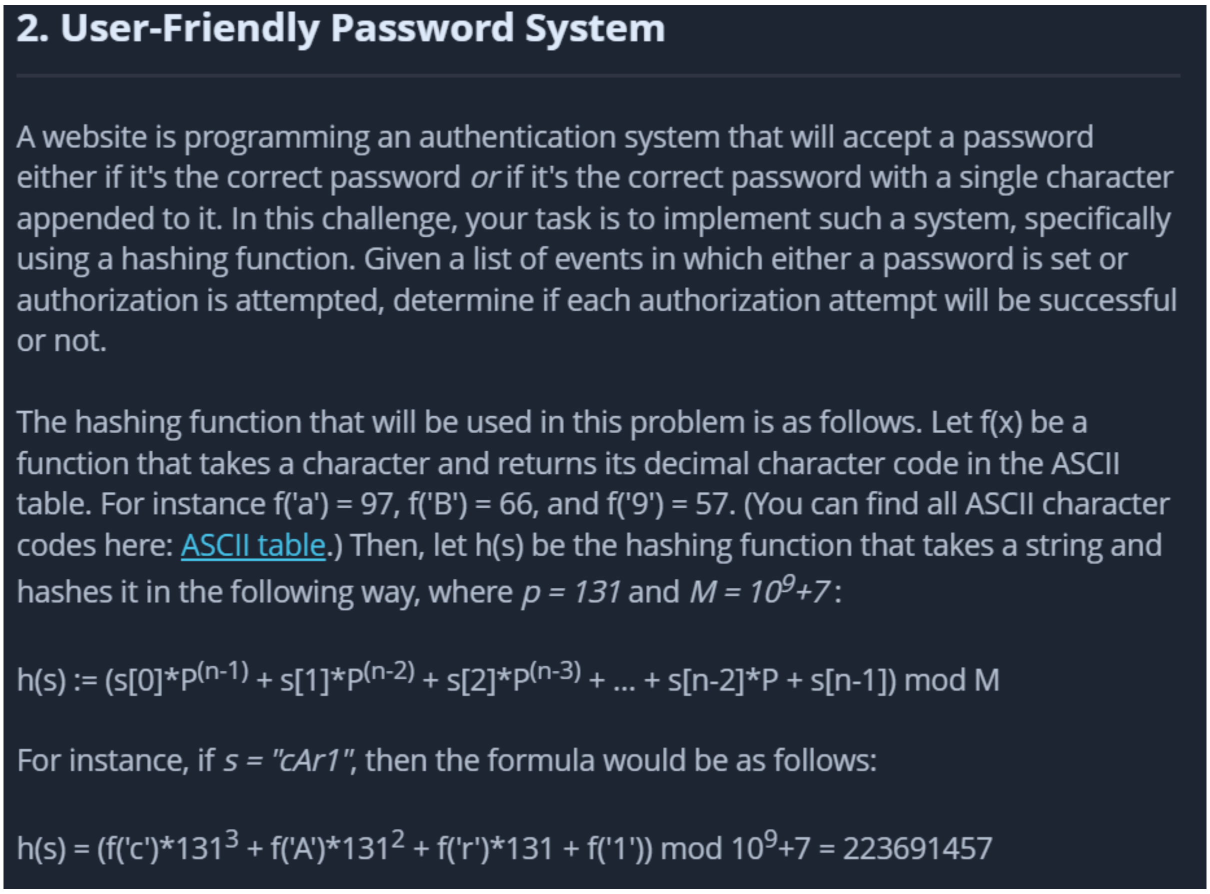


2. User-Friendly Password System

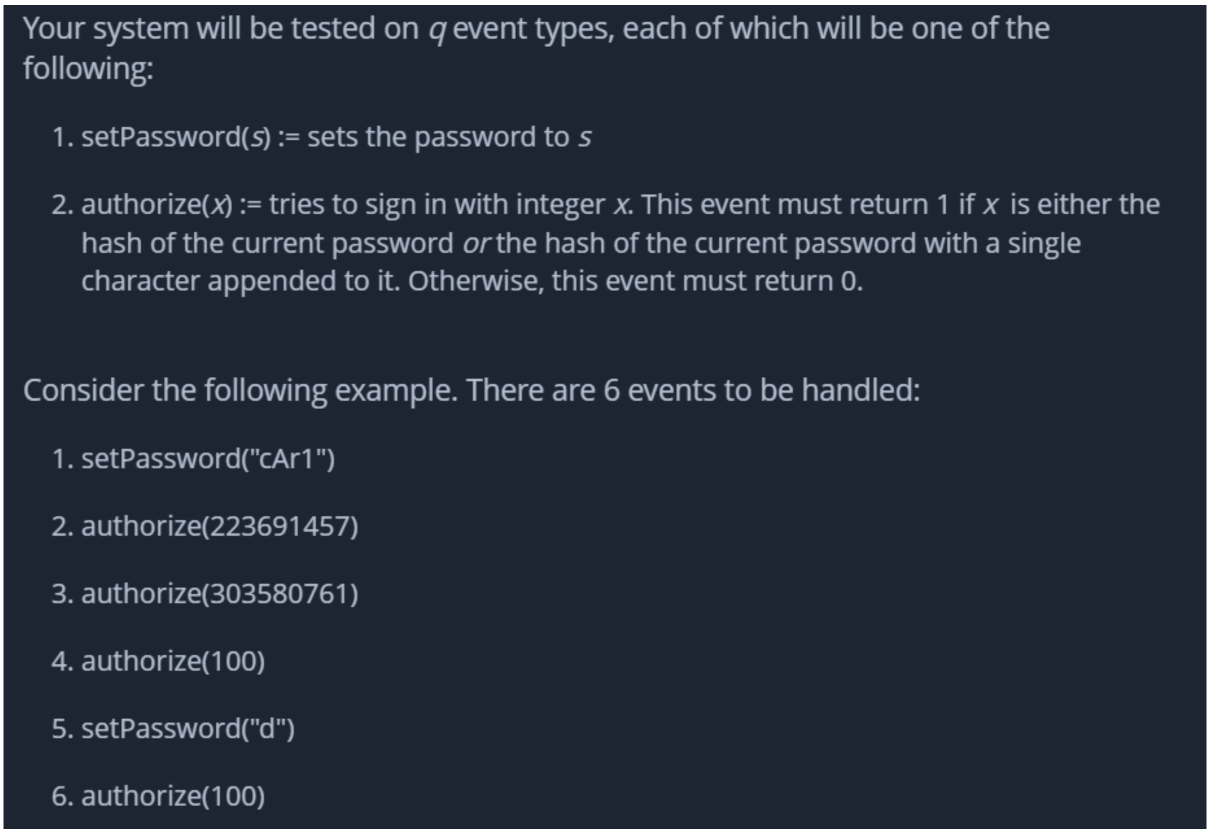
A website is programming an authentication system that will accept a password either if it's the correct password *or* if it's the correct password with a single character appended to it. In this challenge, your task is to implement such a system, specifically using a hashing function. Given a list of events in which either a password is set or authorization is attempted, determine if each authorization attempt will be successful or not.

The hashing function that will be used in this problem is as follows. Let f(x) be a function that takes a character and returns its decimal character code in the ASCII table. For instance f('a') = 97, f('B') = 66, and f('9') = 57. (You can find all ASCII character codes here: ASCII table.) Then, let h(s) be the hashing function that takes a string and hashes it in the following way, where $p = 131$ and $M = 10^9+7$:

h(s) := (s[0]*$P^{(n-1)}$ + s[1]*$P^{(n-2)}$ + s[2]*$P^{(n-3)}$ + ... + s[n-2]*P + s[n-1]) mod M

For instance, if *s = "cAr1"*, then the formula would be as follows:

h(s) = (f('c')*131^3 + f('A')*131^2 + f('r')*131 + f('1')) mod 10^9+7 = 223691457

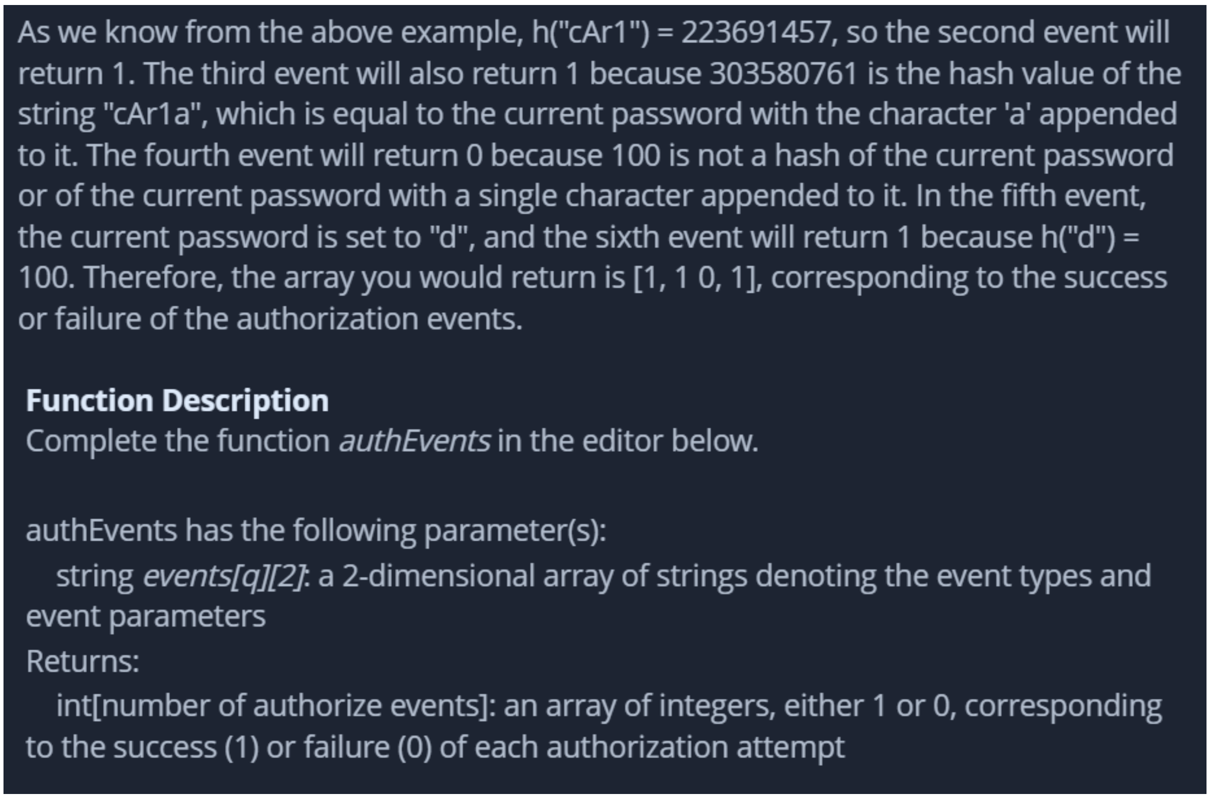


Your system will be tested on *q* event types, each of which will be one of the following:

1. setPassword(*s*) := sets the password to *s*
2. authorize(*x*) := tries to sign in with integer *x*. This event must return 1 if *x* is either the hash of the current password *or* the hash of the current password with a single character appended to it. Otherwise, this event must return 0.

Consider the following example. There are 6 events to be handled:

1. setPassword("cAr1")
2. authorize(223691457)
3. authorize(303580761)
4. authorize(100)
5. setPassword("d")
6. authorize(100)



As we know from the above example, h("cAr1") = 223691457, so the second event will return 1. The third event will also return 1 because 303580761 is the hash value of the string "cAr1a", which is equal to the current password with the character 'a' appended to it. The fourth event will return 0 because 100 is not a hash of the current password or of the current password with a single character appended to it. In the fifth event, the current password is set to "d", and the sixth event will return 1 because h("d") = 100. Therefore, the array you would return is [1, 1 0, 1], corresponding to the success or failure of the authorization events.

Function Description

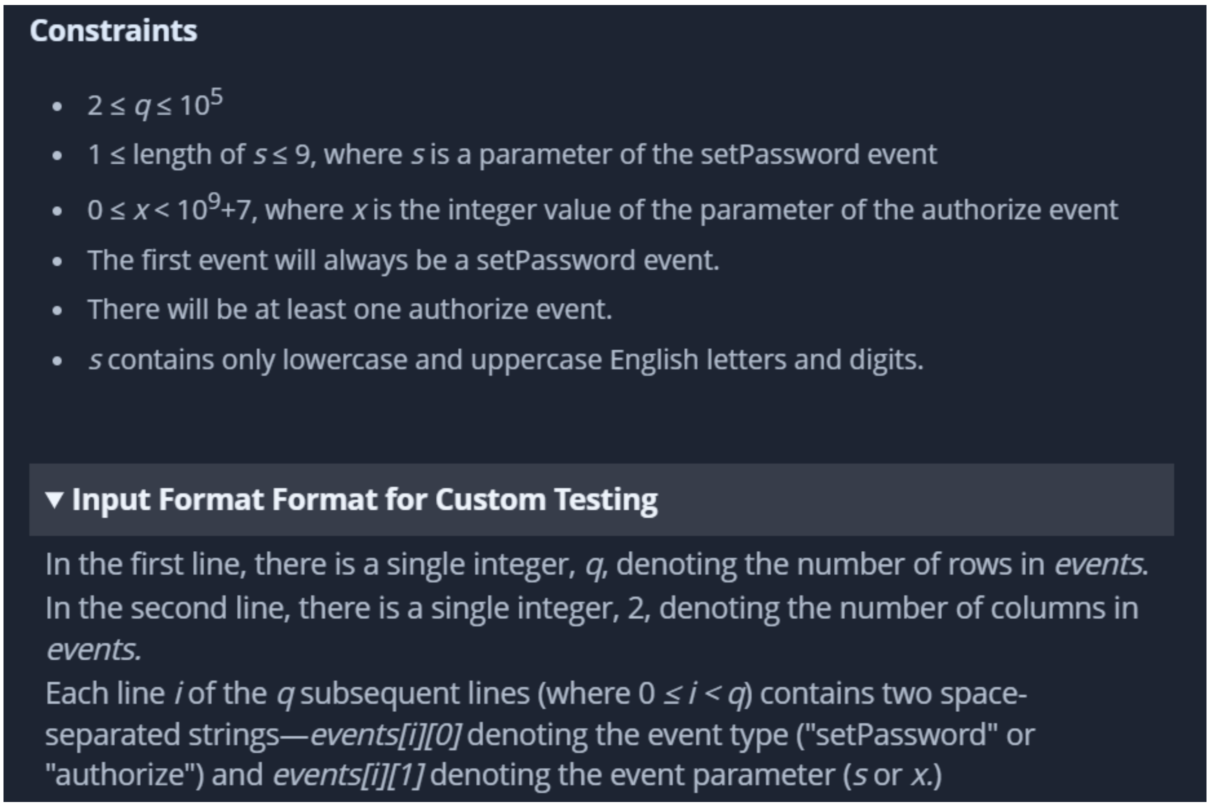
Complete the function *authEvents* in the editor below.

authEvents has the following parameter(s):

string *events[q][2]*: a 2-dimensional array of strings denoting the event types and event parameters

Returns:

int[number of authorize events]: an array of integers, either 1 or 0, corresponding to the success (1) or failure (0) of each authorization attempt

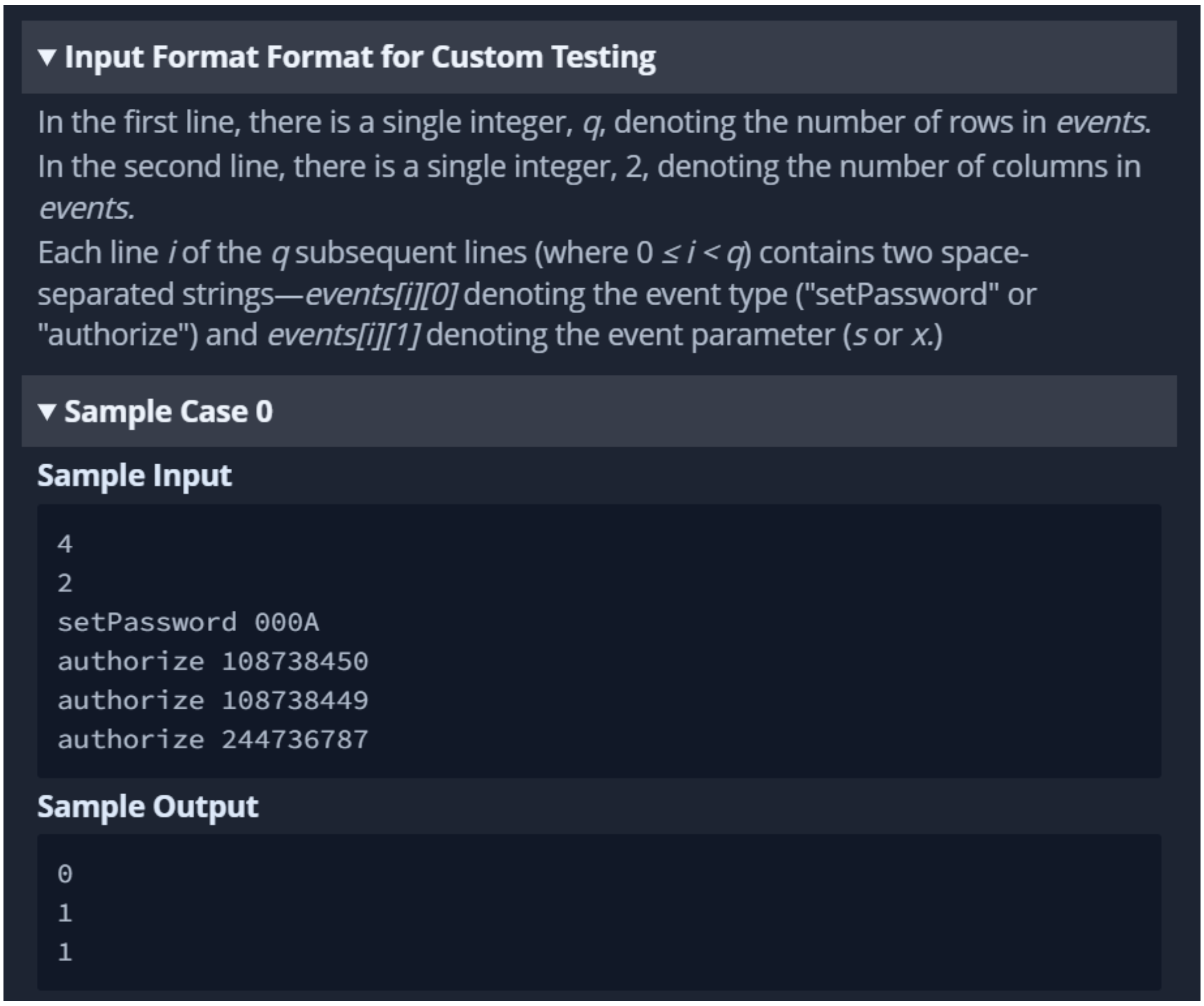


Constraints

- $2 \le q \le 10^5$
- 1 ≤ length of $s \le 9$, where *s* is a parameter of the setPassword event
- $0 \le x < 10^9+7$, where *x* is the integer value of the parameter of the authorize event
- The first event will always be a setPassword event.
- There will be at least one authorize event.
- *s* contains only lowercase and uppercase English letters and digits.

▼ Input Format Format for Custom Testing

In the first line, there is a single integer, *q*, denoting the number of rows in *events.*
In the second line, there is a single integer, 2, denoting the number of columns in *events.*
Each line *i* of the *q* subsequent lines (where $0 \le i < q$) contains two space-separated strings—*events[i][0]* denoting the event type ("setPassword" or "authorize") and *events[i][1]* denoting the event parameter (*s* or *x*.)



▾ Input Format Format for Custom Testing

In the first line, there is a single integer, *q*, denoting the number of rows in *events*.
In the second line, there is a single integer, 2, denoting the number of columns in *events.*
Each line *i* of the *q* subsequent lines (where 0 ≤ *i* < *q*) contains two space-separated strings—*events[i][0]* denoting the event type ("setPassword" or "authorize") and *events[i][1]* denoting the event parameter (*s* or *x*.)

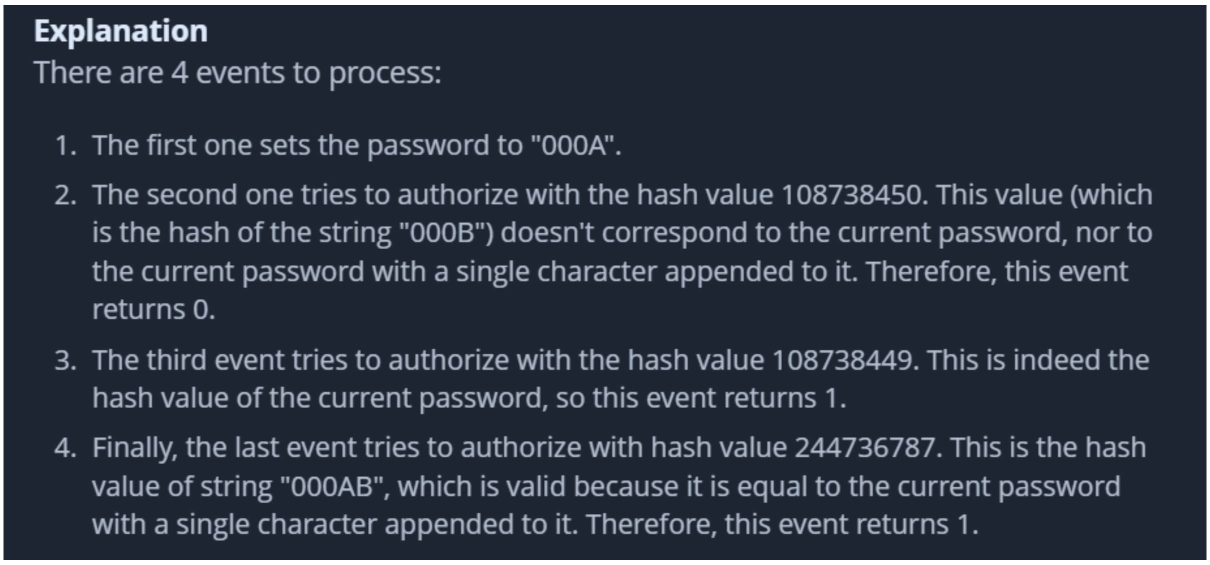
▾ Sample Case 0

Sample Input

```
4
2
setPassword 000A
authorize 108738450
authorize 108738449
authorize 244736787
```

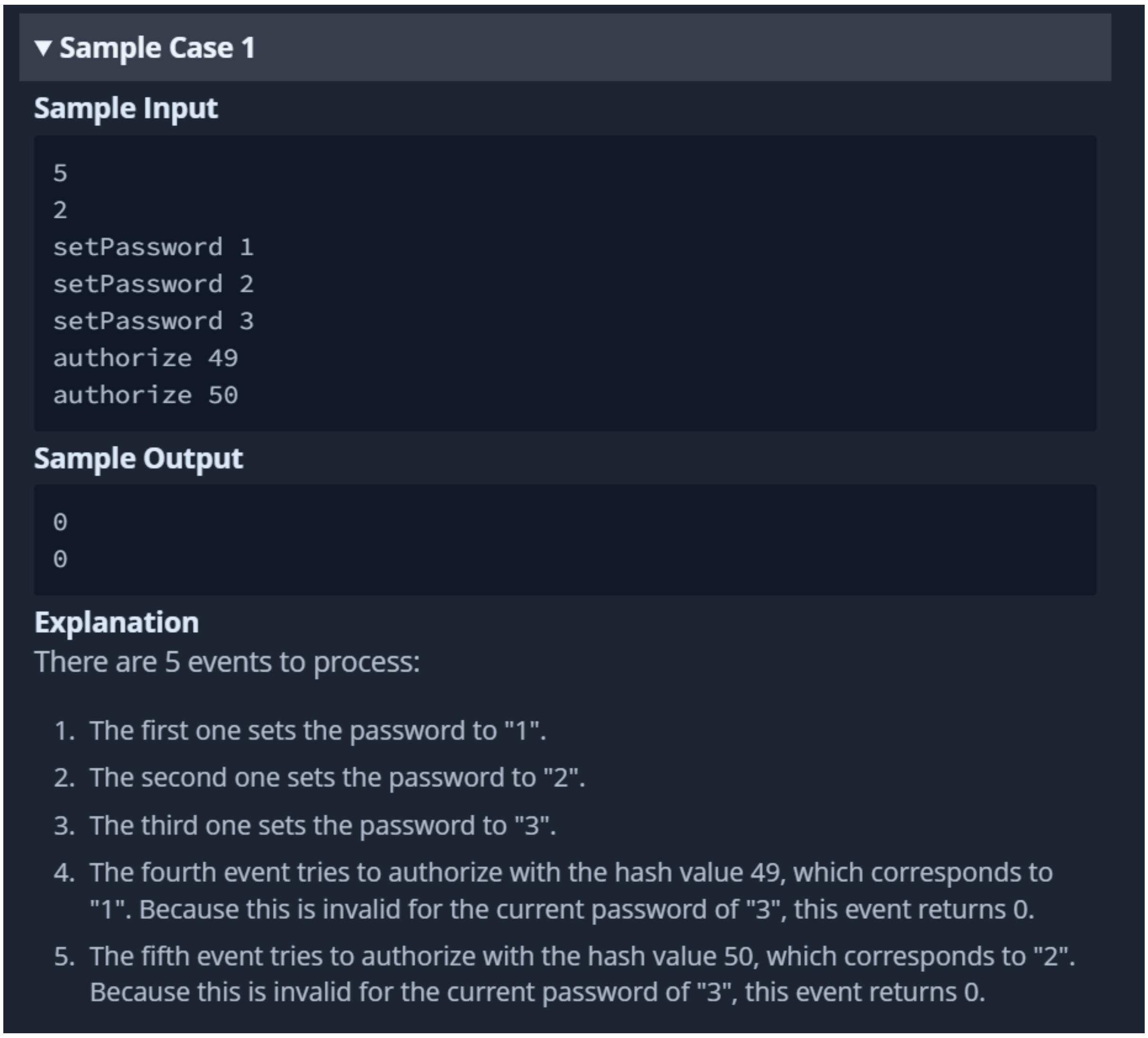
Sample Output

```
0
1
1
```



Explanation
There are 4 events to process:

1. The first one sets the password to "000A".
2. The second one tries to authorize with the hash value 108738450. This value (which is the hash of the string "000B") doesn't correspond to the current password, nor to the current password with a single character appended to it. Therefore, this event returns 0.
3. The third event tries to authorize with the hash value 108738449. This is indeed the hash value of the current password, so this event returns 1.
4. Finally, the last event tries to authorize with hash value 244736787. This is the hash value of string "000AB", which is valid because it is equal to the current password with a single character appended to it. Therefore, this event returns 1.



▼ **Sample Case 1**

Sample Input

```
5
2
setPassword 1
setPassword 2
setPassword 3
authorize 49
authorize 50
```

Sample Output

```
0
0
```

Explanation

There are 5 events to process:

1. The first one sets the password to "1".
2. The second one sets the password to "2".
3. The third one sets the password to "3".
4. The fourth event tries to authorize with the hash value 49, which corresponds to "1". Because this is invalid for the current password of "3", this event returns 0.
5. The fifth event tries to authorize with the hash value 50, which corresponds to "2". Because this is invalid for the current password of "3", this event returns 0.