Safety

Time limit: *2000 ms*
Memory limit: *512 MB*

Vangelis the bear wants to create a tool that will make his passwords stronger. In order to do so, he thought of some transformations, that should make his passwords stronger when applied, and a verification method to check if his tool is doing its job as expected.

Vangelis improvised three kinds of commands for his tool:

1. Check if the substring that starts at position i and ends at position j (inclusive) of the current password is equal to the substring that starts at position k of the current password and has length $j - i + 1$ (it is guaranteed that this substring exists). If the answer is yes, print Y, else print N. The input format of the command is: $1\ i\ j\ k$.
2. Replace the substring that starts at position i and ends at position j (inclusive) of the current password, with the substring that starts at position k of the **original** password and has length $j - i + 1$ (it is guaranteed that this substring exists). The input format of the command is: $2\ i\ j\ k$.
3. Replace each letter in the string that starts at position i and ends at position j (inclusive) of the current password with the next letter of the Latin alphabet, except if the input letter is `z` where it would be replaced with `a`. Examples, `a` will be replaced by `b`, `b` will replaced by `c`, `z` will be replaced by `a` etc. The format of the command is: $3\ i\ j$.

Please note that these operations do not increase the size of the password and that all indices start from 1.

Before he starts coding, Vangelis wants you to create a draft application that will perform the basic functionality of his ideas.

Given a password that is composed from N lowercase Latin characters, you will be given a series of operations to apply on the password and print the result of command type 1.

Standard input

The first line contains the original password.

The second line is an integer M, that represents the number of operations that will be given to your program.

Lines 3 to $M + 2$ contain the input information for one of the command types.

Note:

Some of the test cases are very large, and may require you to speed up input handling in some languages.

In C++, for example, you can include the following line as the first line in your main function to speed up the reading from input:

```
    std::ios_base::sync_with_stdio (false);

```

And in Java, you can use a BufferedReader to greatly speed up reading from input, e.g.:

```
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    // Read next line of input which contains an integer:
    int T = Integer.valueOf(reader.readLine());

```

Standard output

For each type 1 command, print, on a line by itself, the output of the command.

Constraints and notes

- $1 \leq N \leq 3 * 10^5$
- $1 \leq M \leq 3 * 10^5$

Input	Output	Explanation
bbbbxrzbzcj 6 1 1 4 2 2 2 5 7 1 2 6 2 1 2 4 8 3 2 5 1 1 3 9	N Y N N	The first command compares the `bbbb` with `bbbx`, and since they are not equal, the program should output N. The second command replaces the substring from position 2 to 5 with the substring from position 7 to 10 **in the original password**, and thus the password is now `bzbzcrzbzcj`. The third command compares the substring from position 2 to 6 with itself, and thus the expected output is Y. The fourth command compares the substring `zbz` with the substring `bzc`, and thus the output should be N. The fifth command shifts the characters in the substring from position 2 to 5, changing the password to `bacadrzbzcj`. The last command compares the substring `bac` with the substring `zcj`, and outputs N