Hacker: solution

We will use a segment tree that keeps the hashes of intervals of the new password. We also have to keep the interval hashes of the old password, but this can be done simply with prefix sums.

To support updates of the second type (substitutions), we can use lazy propagation of the information on the segment tree. The information that we keep in each "lazy" node is the substring (its hash and where it comes from) of the old string that the new string in the respective interval should be substituted with. To move the lazy information from a node to its children, we can find the two hashes (for left and right child respectively) using the prefix sums information of the old password.

To support updates of the third type, consider a string $w = w_{n-1} \dots w_1 w_0$ and its hash $w_{n-1} \cdot p^{n-1} + \dots + w_1 \cdot p + w_0$. We can group these terms by w_i (so 26 groups total), so we have $1 \cdot (p^{a_1} + \dots + p^{a_{n_1}}) + 2 \cdot (p^{b_1} + \dots + p^{b_{n_2}}) + \dots + 26 \cdot (p^{z_1} + \dots + p^{z_{n_{26}}})$, and we name it $1 \cdot f_1 + 2 \cdot f_2 + \dots + 26 \cdot f_{26}$. Keeping the f's we can recover the hash. Now it is very easy to support the update, because it is simply a shift of f, namely $f'_1 = f_{26}, f'_2 = f_1, \dots$. Of course, we also have to remember the number of shifts that have been done in every node modulo 26, in order to do lazy propagation.

Total complexity $O(N \cdot A + Q \cdot \log N \cdot A)$, where A is the size of the alphabet.