C2. Game with Two Trees

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The Smart Beaver from ABBYY has come up with a new developing game for children. The Beaver thinks that this game will help children to understand programming better.

The main object of the game is finite rooted trees, each of their edges contains some lowercase English letter. Vertices on any tree are always numbered sequentially from 1 to m, where m is the number of vertices in the tree. Before describing the actual game, let's introduce some definitions.

We'll assume that the sequence of vertices with numbers $v_1, v_2, ..., v_k$ ($k \geq 1$) is a forward path, if for any integer i from 1 to k - 1 vertex v_i is a direct ancestor of vertex v_{i+1}. If we sequentially write out all letters from the the edges of the given path from v_1 to v_k, we get some string ($k = 1$ gives us an empty string). We'll say that such string corresponds to forward path $v_1, v_2, ..., v_k$.

We'll assume that the sequence of tree vertices with numbers $v_1, v_2, ..., v_k$ ($k \geq 1$) is a backward path if for any integer i from 1 to k - 1 vertex v_i is the direct descendant of vertex v_{i+1}. If we sequentially write out all the letters from the edges of the given path from v_1 to v_k, we get some string ($k = 1$ gives us an empty string). We'll say that such string corresponds to backward path $v_1, v_2, ..., v_k$.

Now let's describe the game that the Smart Beaver from ABBYY has come up with. The game uses two rooted trees, each of which initially consists of one vertex with number 1. The player is given some sequence of operations. Each operation is characterized by three values (t, v, c) where:

- t is the number of the tree on which the operation is executed (1 or 2);
- v is the vertex index in this tree (it is guaranteed that the tree contains a vertex with this index);
- c is a lowercase English letter.

The actual operation is as follows: vertex v of tree t gets a new descendant with number $m + 1$ (where m is the current number of vertices in tree t), and there should be letter c put on the new edge from vertex v to vertex $m + 1$.

We'll say that an ordered group of three integers (i, j, q) is a good combination if:

- $1 \leq i \leq m_1$, where m_1 is the number of vertices in the first tree;
- $1 \leq j, q \leq m_2$, where m_2 is the number of vertices in the second tree;
- there exists a forward path $v_1, v_2, ..., v_k$ such that $v_1 = j$ and $v_k = q$ in the second tree;
- the string that corresponds to the forward path in the second tree from vertex j to vertex q equals the string that corresponds to the backward path in the first tree from vertex i to vertex 1 (note that both paths are determined uniquely).

Your task is to calculate the number of existing good combinations after each operation on the trees.

Input

The first line contains integer n — the number of operations on the trees. Next n lines specify the operations in the order of their execution. Each line has form "t v c", where t is the number of the tree, v is the vertex index in this tree, and c is a lowercase English letter.

To get the full points for the first group of tests it is sufficient to solve the problem with $1 \leq n \leq 700$.

To get the full points for the second group of tests it is sufficient to solve the problem with $1 \leq n \leq 7000$.

To get the full points for the third group of tests it is sufficient to solve the problem with $1 \leq n \leq 100000$.

Output

Print exactly n lines, each containing one integer — the number of existing good combinations after the corresponding operation from the input.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Examples

input
5 1 1 a 2 1 a 1 2 b 2 1 b 2 3 a

output
1 3 3 4 7

Note

After the first operation the only good combination was $(1, 1, 1)$. After the second operation new good combinations appeared, $(2, 1, 2)$ and $(1, 2, 2)$. The third operation didn't bring any good combinations. The fourth operation added good combination $(1, 3, 3)$. Finally, the fifth operation resulted in as much as three new good combinations — $(1, 4, 4)$, $(2, 3, 4)$ and $(3, 1, 4)$.