CODE Processing

The Coalition Of Development Efficiency (CODE) non-profit group has been attempting to create a processor that far supersedes all others in terms of performance. They put dozens of brains to work, and they struggled with their inner logic for months until they realized... it can't get any better. What a disappointment! Looks like CODE has done its job and is going to disband... until you come up with a brilliant idea! "Why stop at the processor? Let’s optimize machine code to use the most efficient loading mechanisms and make these programs even faster!" Everybody loved the idea of it so much, they let you write the program to do it. Shucks...

Machine code is really simple. There are only three operations that can happen (in this heavily simplified version of a processor):

ADD: Takes the sum of two registers or values, and stores the result in a register (possibly using the same register more than once). The registers need to be created before being used.

STORE: Stores a register or value into another register. Not too time-consuming, but the register has to be created before it is used.

CREATE: Creates a new register. Most time-consuming, and what we want to optimize on (The CREATE commands are inserted explicitly by the compiler, and **WILL NOT** be included in the input).

The programs you are going to optimize are already written, tested, and are not going to change. Your job is to determine the least amount of registers a program needs to CREATE while maintaining the program’s functionality.

To reduce the number of CREATE statements, you will have to check if you can reuse created registers instead of making a new one. A register is considered available for reuse if its value is not ‘used’ again in a following operation. Here we consider a register being ‘used’ if it is referenced in a **source** register for either STORE or ADD operations.

Do not consider any other optimizations for this problem; leave it for the other members of CODE!

Consider the registers used completely local to the program, so specific register numbers do not matter.

Input

Input is the number of operations followed by a list of operations to perform. Each operation is as mentioned above.

[R# | V] : Source register

R#: Destination register

An ADD operation is given as such: ADD **[R# | V] [R# | V] R#**

E.g. ADD R1 1 R2 -> Adds 1 to the value in register 1 and stores it in register 2.

A STORE operation is given as such: STORE **[R# | V] R#**

E.g. STORE 3 R2 -> stores the value 3 into register 2.

E.g. STORE R1 R3 -> stores the value of register 1 into register 2

Output

Output a single line with a single integer giving the minimum number of CREATE statements the compiler must insert to optimize the program, but not change functionality.

Sample Input	Sample Output
2	2
STORE R1 R2	
STORE R2 R3	
4	2
STORE 1 R2	
ADD R1 1 R2	
STORE 3 R2	
ADD R2 R2 R5	
8	4
ADD R1 1 R2	
STORE R2 R3	
ADD R2 R3 R4	
ADD R1 1 R2	
STORE R3 R5	
ADD R5 5 R1	
STORE R1 R5	
ADD R6 R3 R1	