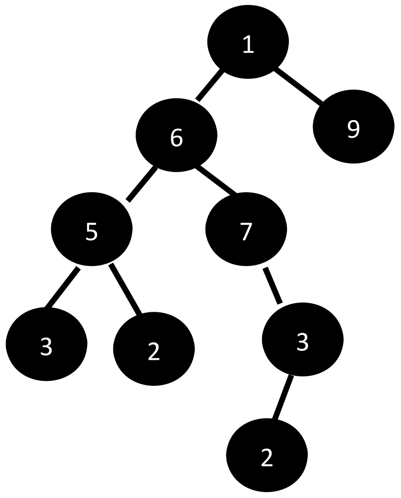
Special nodes considering antecessors and descendants products

In this practice, we denote a node as “special” if the product of its antecessors is equal to the product of its descendants. Notice that we consider that the product of an empty set of nodes is one.

For example, in the following tree:



- The root node “1” is not special because the product of its antecessors (i.e. 1 because empty set) is different from the product of all its descendants (i.e. 6 x 9 x 5 x 7 x 3 x 2 x 3 x 2).
- The node “9” is special because the product of its antecessors (i.e. 1) is equal to the product of its descendants (1 because empty set).
- The node “5” is special because the product of its antecessors (i.e. 6 x 1 = 6) is equal to the product of is descendants (i.e. 3 x 2 = 6).
- The node “7” is also special because the product of its antecessors (i.e. 6 x 1 = 6) is equal to the product of is descendants (i.e. 3 x 2 = 6).
- None of the remaining nodes are special, as you can check by comparing the products of its antecessors and descendants.

Implement an **efficient** program in C++ that receives input from a binary tree “t” and returns the number of special nodes.

In the presented binary tree example, this program should return 3 as there are three special nodes as we have previously mentioned.

Indicate and discuss the **computational complexity** of your proposed solution inside a comment before the definition of the function.

Input

The first line will indicate the number of cases. Each case will be defined with a line, which respectively includes a binary tree of integers.

Each binary tree is represented with a string recursively, in which:

- # represents an empty tree
- [n] represents a tree with just one element on the root with the “n” number

- (left n right) represents a tree with “n” number as root element, the left subtree represented by “left” recursively and the right subtree represented by “right” recursively.

The presented binary tree example is represented as follows with this notation:

((([3] 5 [2]) 6 (# 7 ([2] 3 #))) 1 [9])

Output

The output of each case should be printed in one line. The output of each case will be the number of special nodes.

Implementation Details

In the virtual campus, there is some supporting material for helping you in reading binary trees from the standard input.

Example of input

```
4
((([3] 5 [2]) 6 (# 7 ([2] 3 #))) 1 [9])
#
(((([1] 1 #) 1 #) 1 #) 1 #)
(((([2] 1 #) 1 #) 1 #) 2 #)
```

Example of output

```
3
0
5
3
```