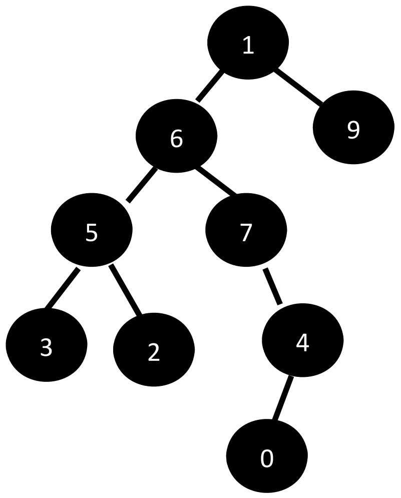
Special nodes considering the number of odd elements in antecessors and descendants

In this practice, we denote a node as “special” in a binary tree if the number of odd numbers in its antecessors is greater to the number of odd numbers in its descendants.

For example, in the following tree:



- The root node “1” is not special because the number of odd numbers of its antecessors (i.e. 0 because empty set) is not greater than the number of odd numbers of all descendants, i.e. 4 odd numbers (5, 3, 7 and 9).
- Node 6 is not special (1 odd antecessors and 3 odd descendants)
- Node 5 is not special (1 odd antecessor and 1 odd descendant)
- Node 3 is special (2 odd antecessors and 0 odd descendant)

In fact, in this tree the special nodes are 3, 2, 7, 4, 0, and 9.

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 6, as there are six 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 ([0] 4 #))) 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 a template shared by both versions of this control. This template reads the tree of each case and prints the output. In this template, the student needs to implement the "solveCase" function that receives an input from a binary tree of integers and should return a number. Notice that students will probably need to define a new generalized function to implement this function. Using this template is optional, so students can either use it or not as they prefer. This template also includes the binary tree file.

Example of input

```
4
((([3] 5 [2]) 6 (# 7 ([0] 4 #))) 1 [9])
#
(((([1] 1 #) 1 #) 1 #) 1 #)
(((([2] 1 #) 1 #) 1 #) 2 #)
```

Example of output

```
6
0
2
2
```