Description Solution Discuss (147) Submissions

1628. Design an Expression Tree With Evaluate Function

Medium 267 46 Add to List Share

Given the `postfix` tokens of an arithmetic expression, build and return *the binary expression tree that represents this expression.*

Postfix notation is a notation for writing arithmetic expressions in which the operands (numbers) appear before their operators. For example, the postfix tokens of the expression `4*(5-(7+2))` are represented in the array `postfix = ["4","5","7","2","+","-","*"]`.

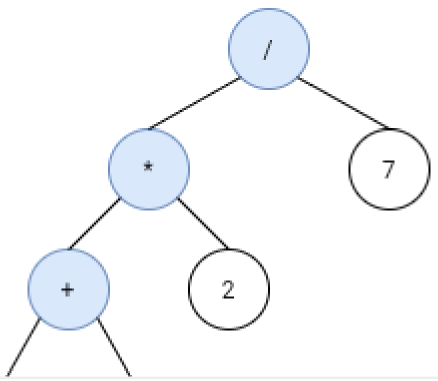
The class `Node` is an interface you should use to implement the binary expression tree. The returned tree will be tested using the `evaluate` function, which is supposed to evaluate the tree's value. You should not remove the `Node` class; however, you can modify it as you wish, and you can define other classes to implement it if needed.

A **binary expression tree** is a kind of binary tree used to represent arithmetic expressions. Each node of a binary expression tree has either zero or two children. Leaf nodes (nodes with 0 children) correspond to operands (numbers), and internal nodes (nodes with two children) correspond to the operators `'+'` (addition), `'-'` (subtraction), `'*'` (multiplication), and `'/'` (division).

It's guaranteed that no subtree will yield a value that exceeds 10^9 in absolute value, and all the operations are valid (i.e., no division by zero).

Follow up: Could you design the expression tree such that it is more modular? For example, is your design able to support additional operators without making changes to your existing `evaluate` implementation?

Example 1:



C#

```
InvalidOperationExcepti
    }
}


/**
 * This is the TreeBuil
class.
 * You can treat it as
driver code that takes
postfix input
 * and returns the expr
tree represnting it as
 */

public class TreeBuilde

    public Node
buildTree(string[] post
        var stack = new
Stack<Node>();
        foreach(string
postfix)
        {
            Node node;

 if(char.IsNumber(s[0])
            {
                node =
ValueNode(int.Parse(s))
            }
            else{
                var ri
stack.Pop();
                var lef
stack.Pop();

                node =
```

Testcase Run Code Result

Accepted Runtime: 110 ms

Your input `["3","4","+","2",`

Output `2`

Expected `2`

Console Use Example Testcase