Description | Solution | Discuss (128) | Submissions

1628. Design an Expression Tree With Evaluate Function

Medium 224 40 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","+","-","*"]`.

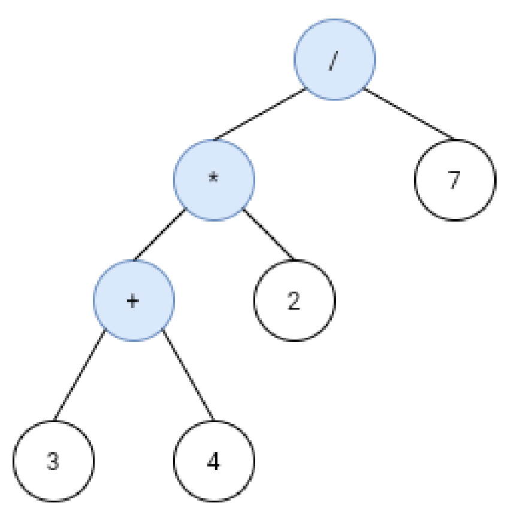
The class `Node` is an interface you should use to implement the binary expression tree. The returned tree will be tested using the `evaluate` function, which is supposed to evaluate the tree's value. You should not remove the `Node` class; however, you can modify it as you wish, and you can define other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

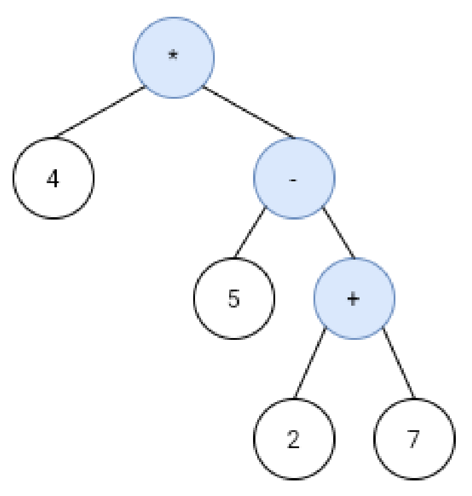
Follow up: Could you design the expression tree such that it is more modular? For example, is your design able to support additional operators without making changes to your existing `evaluate` implementation?

Example 1:



```
Input: s = ["3","4","+","2","*","7","/"]
Output: 2
Explanation: this expression evaluates to the above binary tree with expression
((3+4)*2)/7) = 14/7 = 2.
```

Example 2:



```
Input: s = ["4","5","2","7","+","-","*"]
Output: -16
Explanation: this expression evaluates to the above binary tree with expression 4*
(5-(2+7)) = 4*(-4) = -16.
```

Example 3:

```
Input: s = ["4","2","+","3","5","1","-","*","+"]
Output: 18
```

Example 4:

```
Input: s = ["100","200","+","2","/","5","*","7","+"]
Output: 757
```

Constraints:

- `1 <= s.length < 100`
- `s.length` is odd.
- `s` consists of numbers and the characters `'+'`, `'-'`, `'*'`, and `'/'`.
- If `s[i]` is a number, its integer representation is no more than 10^5.
- It is guaranteed that `s` is a valid expression.
- The absolute value of the result and intermediate values will not exceed 10^9.
- It is guaranteed that no expression will include division by zero.

Accepted 9,903 | Submissions 12,218

Seen this question in a real interview before? Yes No

Companies i

0 ~ 6 months | 6 months ~ 1 year | 1 year ~ 2 years

Amazon 9

Related Topics

Math Stack Tree Design Binary Tree

Hide Hint 1

Apply the concept of Polymorphism to get a good design

Hide Hint 2

Implement the Node class using NumericNode and OperatorNode classes.

Hide Hint 3

NumericNode only maintains the value and evaluate returns this value.

Hide Hint 4

OperatorNode Maintains the left and right nodes representing the left and right operands, and the evaluate function applies the operator to them.

Java | Autocomplete

```java
/**
 * This is the interface for the expression tree Node.
 * You should not remove it, and you can define some classes to implement it.
 */

abstract class Node {
    public abstract int evaluate();
    // define your fields here
};


/**
 * This is the TreeBuilder class.
 * You can treat it as the driver code that takes the postinfix input
 * and returns the expression tree represnting it as a Node.
 */

class TreeBuilder {
    Node buildTree(String[] postfix) {

    }
};


/**
 * Your TreeBuilder object will be instantiated and called as such:
 * TreeBuilder obj = new TreeBuilder();
 * Node expTree = obj.buildTree(postfix);
 * int ans = expTree.evaluate();
 */
```