Problem 1628: Design an Expression Tree With Evaluate Function

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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).

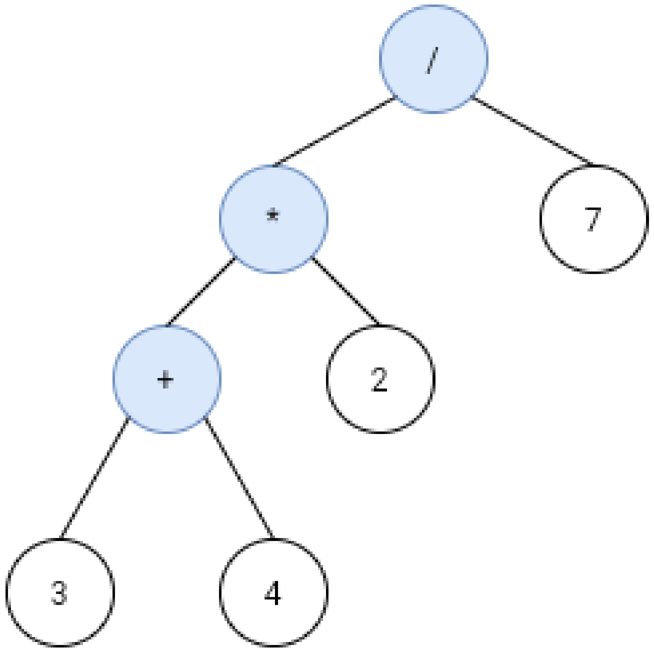
Follow up:

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:

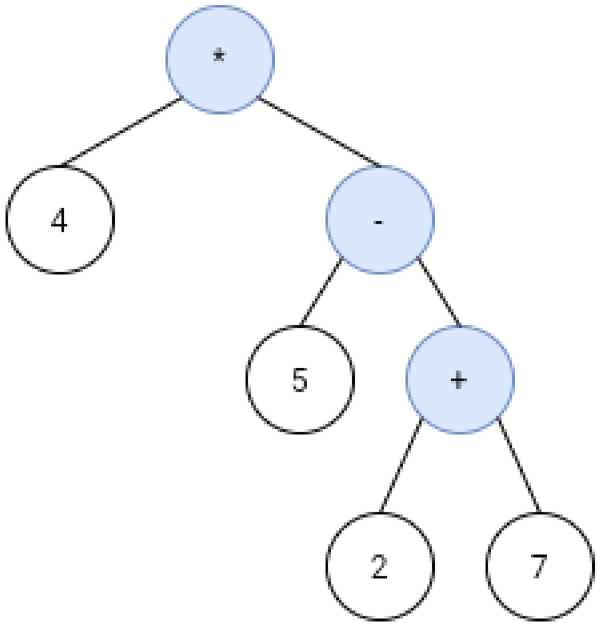
2

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.

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.

Code Snippets

C++:

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

class Node {
public:
virtual ~Node () {};
virtual int evaluate() const = 0;
protected:
// 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 representing it as a Node.
*/

class TreeBuilder {
```

```
public:
Node* buildTree(vector<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();
*/
```

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 representing 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();
*/
```

Python3:

```
import abc
from abc import ABC, abstractmethod
"""
This is the interface for the expression tree Node.
You should not remove it, and you can define some classes to implement it.
"""

class Node(ABC):
@abstractmethod
# define your fields here
def evaluate(self) -> int:
pass



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

class TreeBuilder(object):
def buildTree(self, postfix: List[str]) -> 'Node':



"""
Your TreeBuilder object will be instantiated and called as such:
obj = TreeBuilder();
expTree = obj.buildTree(postfix);
ans = expTree.evaluate();
"""
```

Python:

```
import abc
from abc import ABCMeta, abstractmethod
```

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

class Node:
__metaclass__ = ABCMeta
# define your fields here
@abstractmethod
def evaluate(self):
pass


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

class TreeBuilder(object):
def buildTree(self, postfix):
"""
:type s: List[str]
:rtype: int
"""

"""
Your TreeBuilder object will be instantiated and called as such:
obj = TreeBuilder();
expTree = obj.buildTree(postfix);
ans = expTree.evaluate();
"""
```

JavaScript:

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

var Node = function () {
if (this.constructor === Node) {
```

```
throw new Error('Cannot instanciate abstract class');
}
};

Node.prototype.evaluate = function () {
throw new Error('Cannot call abstract method')
};

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

class TreeBuilder{
/**
* @param {string[]} s
* @return {Node}
*/
buildTree(postfix) {

}

}

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

C#:

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

public 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 representing it as a Node.
*/

public class TreeBuilder {
public 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();
*/
```

Solutions

C++ Solution:

```
/*
* Problem: Design an Expression Tree With Evaluate Function
* Difficulty: Medium
* Tags: array, tree, math, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

```
*/

class Node {
public:
virtual ~Node () {
// TODO: Implement optimized solution
return 0;
};
virtual int evaluate() const = 0;
protected:
// 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 representing it as a Node.
*/

class TreeBuilder {
public:
Node* buildTree(vector<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();
*/
```

Java Solution:

```
/**
* Problem: Design an Expression Tree With Evaluate Function
* Difficulty: Medium
* Tags: array, tree, math, stack
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* 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 representing 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();
*/
```

Python3 Solution:

```
"""
Problem: Design an Expression Tree With Evaluate Function
```

```
Difficulty: Medium
Tags: array, tree, math, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

import abc
from abc import ABC, abstractmethod
"""
This is the interface for the expression tree Node.
You should not remove it, and you can define some classes to implement it.
"""

class Node(ABC):
@abstractmethod
# define your fields here
def evaluate(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
import abc
from abc import ABCMeta, abstractmethod
"""
This is the interface for the expression tree Node.
You should not remove it, and you can define some classes to implement it.
"""

class Node:
__metaclass__ = ABCMeta
# define your fields here
@abstractmethod
def evaluate(self):
pass



"""
This is the TreeBuilder class.
```

```
You can treat it as the driver code that takes the postinfix input
and returns the expression tree representing it as a Node.
"""

class TreeBuilder(object):
def buildTree(self, postfix):
"""
:type s: List[str]
:rtype: int
"""

"""
Your TreeBuilder object will be instantiated and called as such:
obj = TreeBuilder();
expTree = obj.buildTree(postfix);
ans = expTree.evaluate();
"""
```

JavaScript Solution:

```
/**
* Problem: Design an Expression Tree With Evaluate Function
* Difficulty: Medium
* Tags: array, tree, math, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

var Node = function () {
if (this.constructor === Node) {
throw new Error('Cannot instanciate abstract class');
}
};
```

```
Node.prototype.evaluate = function () {
throw new Error('Cannot call abstract method')
};

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

class TreeBuilder{
/**
* @param {string[]} s
* @return {Node}
*/
buildTree(postfix) {

}

}

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

C# Solution:

```
/*
* Problem: Design an Expression Tree With Evaluate Function
* Difficulty: Medium
* Tags: array, tree, math, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
```

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

public 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 representing it as a Node.
*/

public class TreeBuilder {
public 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();
*/
```