Problem 563: Binary Tree Tilt

Problem Information

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

Problem Description

Given the

root

of a binary tree, return

the sum of every tree node's

tilt

.

The

tilt

of a tree node is the

absolute difference

between the sum of all left subtree node

values

and all right subtree node

values

. If a node does not have a left child, then the sum of the left subtree node

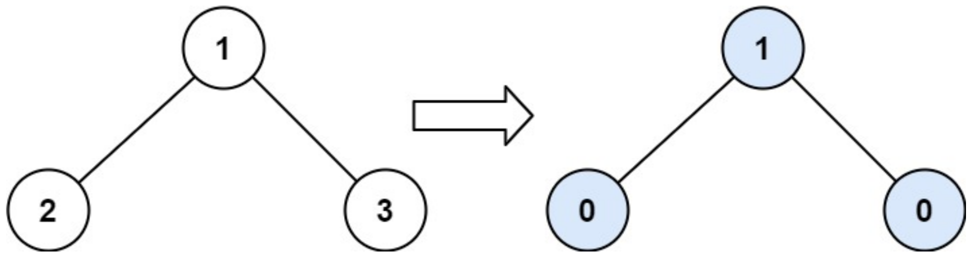
values

is treated as

0

. The rule is similar if the node does not have a right child.

Example 1:



Input:

root = [1,2,3]

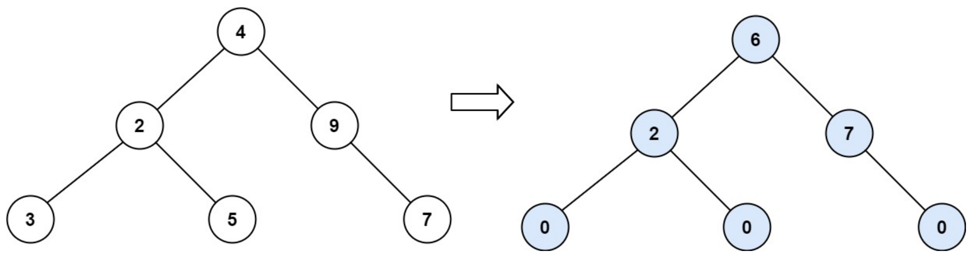
Output:

1

Explanation:

Tilt of node 2 : |0-0| = 0 (no children) Tilt of node 3 : |0-0| = 0 (no children) Tilt of node 1 : |2-3| = 1 (left subtree is just left child, so sum is 2; right subtree is just right child, so sum is 3) Sum of every tilt : 0 + 0 + 1 = 1

Example 2:



Input:

root = [4,2,9,3,5,null,7]

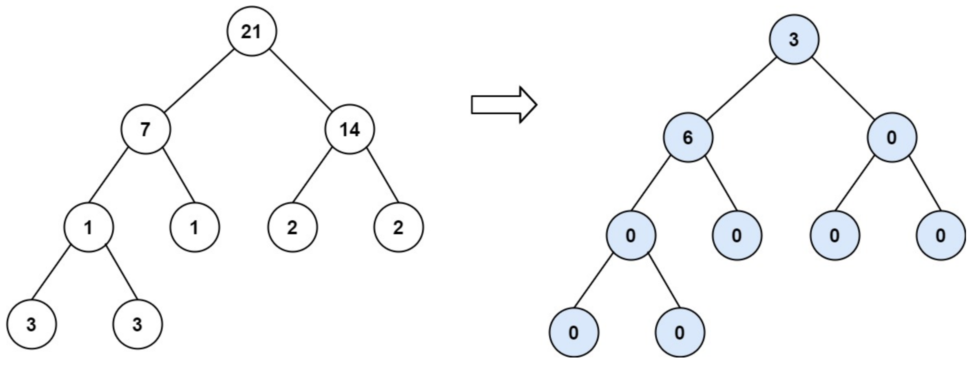
Output:

15

Explanation:

Tilt of node 3 : |0-0| = 0 (no children) Tilt of node 5 : |0-0| = 0 (no children) Tilt of node 7 : |0-0| = 0 (no children) Tilt of node 2 : |3-5| = 2 (left subtree is just left child, so sum is 3; right subtree is just right child, so sum is 5) Tilt of node 9 : |0-7| = 7 (no left child, so sum is 0; right subtree is just right child, so sum is 7) Tilt of node 4 : |(3+5+2)-(9+7)| = |10-16| = 6 (left subtree values are 3, 5, and 2, which sums to 10; right subtree values are 9 and 7, which sums to 16) Sum of every tilt : 0 + 0 + 0 + 2 + 7 + 6 = 15

Example 3:



Input:

root = [21,7,14,1,1,2,2,3,3]

Output:

9

Constraints:

The number of nodes in the tree is in the range

[0, 10

4

]

.

-1000 <= Node.val <= 1000

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
int findTilt(TreeNode* root) {

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public int findTilt(TreeNode root) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def findTilt(self, root: Optional[TreeNode]) -> int:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def findTilt(self, root):
"""
```

```
:type root: Optional[TreeNode]
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @return {number}
*/
var findTilt = function(root) {

};
```

TypeScript:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function findTilt(root: TreeNode | null): number {

};
```

C#:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public int FindTilt(TreeNode root) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
int findTilt(struct TreeNode* root) {

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func findTilt(root *TreeNode) int {

}
```

Kotlin:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun findTilt(root: TreeNode?): Int {

}
}
```

Swift:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
```

```
class Solution {
func findTilt(_ root: TreeNode?) -> Int {

}
}
```

Rust:

```
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn find_tilt(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

}
}
```

Ruby:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer}
def find_tilt(root)

end
```

PHP:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @return Integer
*/
function findTilt($root) {

}
}
```

Dart:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
int findTilt(TreeNode? root) {

}
}
```

Scala:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def findTilt(root: TreeNode): Int = {

}
}
```

Elixir:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec find_tilt(root :: TreeNode.t | nil) :: integer
```

```
def find_tilt(root) do

end
end
```

Erlang:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec find_tilt(Root :: #tree_node{} | null) -> integer().
find_tilt(Root) ->
.
```

Racket:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (find-tilt root)
(-> (or/c tree-node? #f) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
class Solution {
public:
int findTilt(TreeNode* root) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public int findTilt(TreeNode root) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Tree Tilt
Difficulty: Easy
Tags: tree, search

Approach: DFS or BFS traversal
```

```
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def findTilt(self, root: Optional[TreeNode]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def findTilt(self, root):
"""
:type root: Optional[TreeNode]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @return {number}
*/
var findTilt = function(root) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function findTilt(root: TreeNode | null): number {

};
```

C# Solution:

```
/*
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public int FindTilt(TreeNode root) {

}
}
```

C Solution:

```
/*
* Problem: Binary Tree Tilt
* Difficulty: Easy
* Tags: tree, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
int findTilt(struct TreeNode* root) {

}
```

Go Solution:

```
// Problem: Binary Tree Tilt
// Difficulty: Easy
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func findTilt(root *TreeNode) int {
}
```

Kotlin Solution:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun findTilt(root: TreeNode?): Int {

}
}
```

Swift Solution:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func findTilt(_ root: TreeNode?) -> Int {
}
}
```

Rust Solution:

```
// Problem: Binary Tree Tilt
// Difficulty: Easy
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn find_tilt(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

}
}
```

Ruby Solution:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer}
def find_tilt(root)

end
```

PHP Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @return Integer
*/
function findTilt($root) {

}
}
```

Dart Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
int findTilt(TreeNode? root) {

}
}
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def findTilt(root: TreeNode): Int = {

}
}
```

Elixir Solution:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec find_tilt(root :: TreeNode.t | nil) :: integer
def find_tilt(root) do

end
end
```

Erlang Solution:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec find_tilt(Root :: #tree_node{} | null) -> integer().
find_tilt(Root) ->
.
```

Racket Solution:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (find-tilt root)
(-> (or/c tree-node? #f) exact-integer?)
)
```