Problem 2313: Minimum Flips in Binary Tree to Get Result

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given the

root

of a

binary tree

with the following properties:

Leaf nodes

have either the value

0

or

1

, representing

false

and

,

true

respectively.

Non-leaf nodes

have either the value

2

,

3

,

4

, or

5

, representing the boolean operations

OR

,

AND

,

XOR

, and

NOT

, respectively.

You are also given a boolean

result

, which is the desired result of the

evaluation

of the

root

node.

The evaluation of a node is as follows:

If the node is a leaf node, the evaluation is the

value

of the node, i.e.

true

or

false

.

Otherwise,

evaluate

the node's children and

apply

the boolean operation of its value with the children's evaluations.

In one operation, you can

flip

a leaf node, which causes a

false

node to become

true

, and a

true

node to become

false

.

Return

the minimum number of operations that need to be performed such that the evaluation of

root

yields

result

. It can be shown that there is always a way to achieve

result

.

A

leaf node

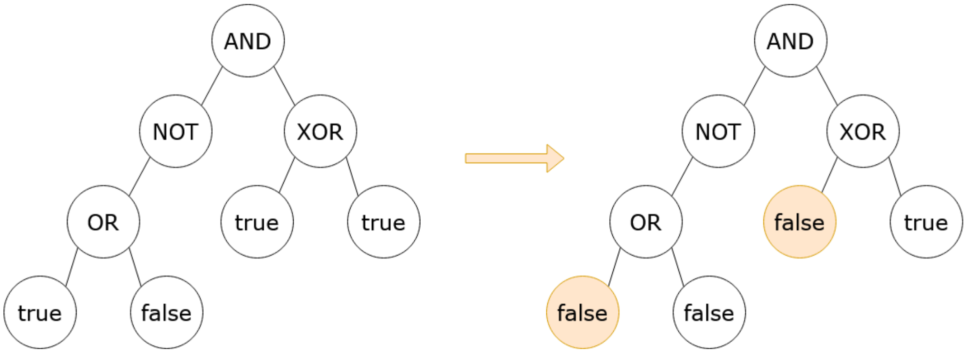
is a node that has zero children.

Note:

NOT

nodes have either a left child or a right child, but other non-leaf nodes have both a left child and a right child.

Example 1:



Input:

root = [3,5,4,2,null,1,1,1,0], result = true

Output:

2

Explanation:

It can be shown that a minimum of 2 nodes have to be flipped to make the root of the tree evaluate to true. One way to achieve this is shown in the diagram above.

Example 2:

Input:

root = [0], result = false

Output:

0

Explanation:

The root of the tree already evaluates to false, so 0 nodes have to be flipped.

Constraints:

The number of nodes in the tree is in the range

[1, 10

5

]

.

0 <= Node.val <= 5

OR

,

AND

, and

XOR

nodes have

2

children.

NOT

nodes have

1

child.

Leaf nodes have a value of

0

or

1

.

Non-leaf nodes have a value of

2

,

3

,

4

, or

5

.

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 minimumFlips(TreeNode* root, bool result) {

}
};
```

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 minimumFlips(TreeNode root, boolean result) {

}
}
```

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 minimumFlips(self, root: Optional[TreeNode], result: bool) -> 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 minimumFlips(self, root, result):
"""
:type root: Optional[TreeNode]
:type result: bool
: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
* @param {boolean} result
* @return {number}
*/
var minimumFlips = function(root, result) {

};
```

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 minimumFlips(root: TreeNode | null, result: boolean): 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 MinimumFlips(TreeNode root, bool result) {

}
}
```

C:

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

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func minimumFlips(root *TreeNode, result bool) 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 minimumFlips(root: TreeNode?, result: Boolean): 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 minimumFlips(_ root: TreeNode?, _ result: Bool) -> 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 minimum_flips(root: Option<Rc<RefCell<TreeNode>>>, result: bool) ->
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
# @param {Boolean} result
# @return {Integer}
def minimum_flips(root, result)

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
* @param Boolean $result
* @return Integer
*/
function minimumFlips($root, $result) {

}
}
```

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 minimumFlips(TreeNode? root, bool result) {

}

}
```

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 minimumFlips(root: TreeNode, result: Boolean): 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 minimum_flips(root :: TreeNode.t | nil, result :: boolean) :: integer
def minimum_flips(root, result) 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 minimum_flips(Root :: #tree_node{} | null, Result :: boolean()) ->
integer().
minimum_flips(Root, Result) ->
.
```

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 (minimum-flips root result)
(-> (or/c tree-node? #f) boolean? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
*
```

```
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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 minimumFlips(TreeNode* root, bool result) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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 minimumFlips(TreeNode root, boolean result) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Flips in Binary Tree to Get Result
Difficulty: Hard
Tags: tree, dp, search

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(n) or O(n * m) for DP table
"""

# 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 minimumFlips(self, root: Optional[TreeNode], result: bool) -> 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 minimumFlips(self, root, result):
"""
:type root: Optional[TreeNode]
:type result: bool
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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
* @param {boolean} result
* @return {number}
*/
var minimumFlips = function(root, result) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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 minimumFlips(root: TreeNode | null, result: boolean): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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 MinimumFlips(TreeNode root, bool result) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Flips in Binary Tree to Get Result
* Difficulty: Hard
* Tags: tree, dp, search
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Minimum Flips in Binary Tree to Get Result
// Difficulty: Hard
// Tags: tree, dp, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func minimumFlips(root *TreeNode, result bool) 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 minimumFlips(root: TreeNode?, result: Boolean): 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 minimumFlips(_ root: TreeNode?, _ result: Bool) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Flips in Binary Tree to Get Result
// Difficulty: Hard
// Tags: tree, dp, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

// 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 minimum_flips(root: Option<Rc<RefCell<TreeNode>>>, result: bool) ->
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
# @param {Boolean} result
# @return {Integer}
def minimum_flips(root, result)

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
* @param Boolean $result
* @return Integer
*/
function minimumFlips($root, $result) {

}
}
```

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 minimumFlips(TreeNode? root, bool result) {

}
}
```

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 minimumFlips(root: TreeNode, result: Boolean): 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 minimum_flips(root :: TreeNode.t | nil, result :: boolean) :: integer
def minimum_flips(root, result) 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 minimum_flips(Root :: #tree_node{} | null, Result :: boolean()) ->
integer().
minimum_flips(Root, Result) ->
.
```

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 (minimum-flips root result)
(-> (or/c tree-node? #f) boolean? exact-integer?)
)
```