

Problem 663: Equal Tree Partition

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the

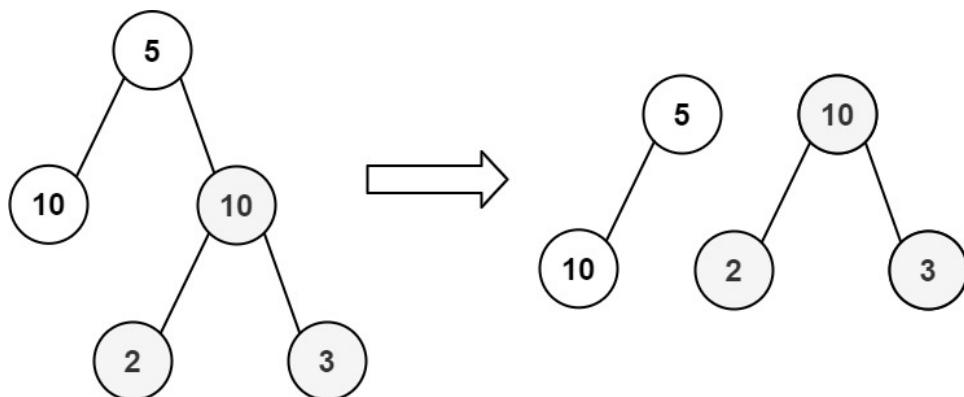
root

of a binary tree, return

true

if you can partition the tree into two trees with equal sums of values after removing exactly one edge on the original tree

Example 1:



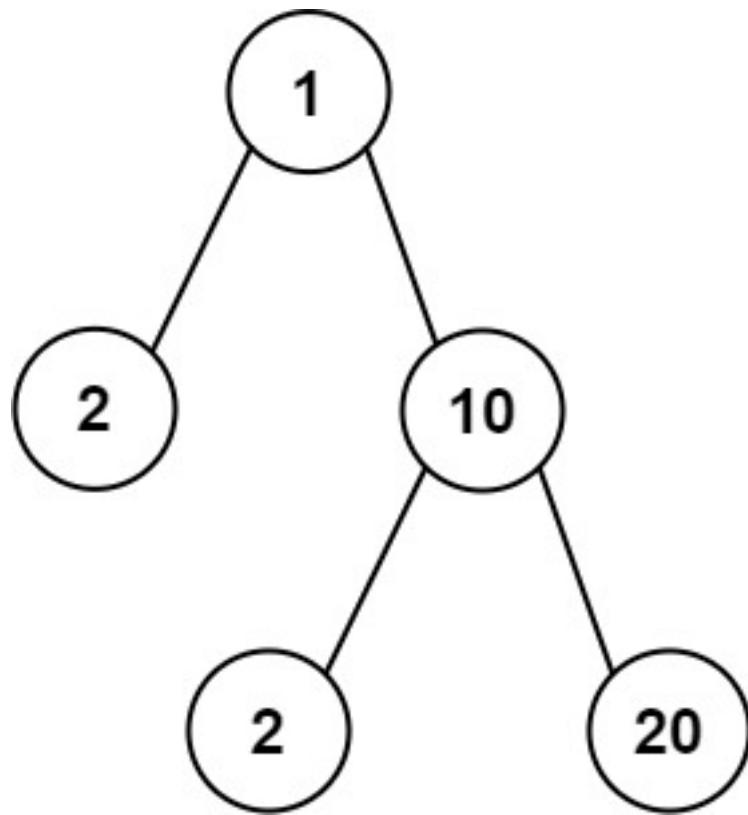
Input:

```
root = [5,10,10,null,null,2,3]
```

Output:

true

Example 2:



Input:

```
root = [1,2,10,null,null,2,20]
```

Output:

false

Explanation:

You cannot split the tree into two trees with equal sums after removing exactly one edge on the tree.

Constraints:

The number of nodes in the tree is in the range

[1, 10]

4

]

.

-10

5

`<= Node.val <= 10`

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:
    bool checkEqualTree(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 boolean checkEqualTree(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 checkEqualTree(self, root: Optional[TreeNode]) -> bool:
```

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 checkEqualTree(self, root):
        """
:type root: Optional[TreeNode]
:rtype: bool
"""

```

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 {boolean}
 */
var checkEqualTree = 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 checkEqualTree(root: TreeNode | null): boolean {
}

```

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 bool CheckEqualTree(TreeNode root) {
        }
    }
}

```

C:

```

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

```

Go:

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

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 checkEqualTree(root: TreeNode?): Boolean {
}
```

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 checkEqualTree(_ root: TreeNode?) -> Bool {
}
}

```

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 check_equal_tree(root: Option<Rc<RefCell<TreeNode>>>) -> bool {
}
}

```

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 {Boolean}
def check_equal_tree(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 $root
 * @return Boolean
 */
function checkEqualTree($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 {  
bool checkEqualTree(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 checkEqualTree(root: TreeNode): Boolean = {  
  
}  
}  
}
```

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 check_equal_tree(root :: TreeNode.t | nil) :: boolean
def check_equal_tree(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 check_equal_tree(Root :: #tree_node{} | null) -> boolean().
check_equal_tree(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 (check-equal-tree root)
  (-> (or/c tree-node? #f) boolean?))

```

```
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Equal Tree Partition
 * Difficulty: Medium
 * 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:
    bool checkEqualTree(TreeNode* root) {
```

```
}
```

```
} ;
```

Java Solution:

```
/**  
 * Problem: Equal Tree Partition  
 * Difficulty: Medium  
 * 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 boolean checkEqualTree(TreeNode root) {  
  
    }  
}
```

Python3 Solution:

```

"""
Problem: Equal Tree Partition
Difficulty: Medium
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 checkEqualTree(self, root: Optional[TreeNode]) -> bool:
        # 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 checkEqualTree(self, root):
        """
:type root: Optional[TreeNode]
:rtype: bool
"""

```

JavaScript Solution:

```

/**
 * Problem: Equal Tree Partition
 * Difficulty: Medium
 * 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 {boolean}
 */
var checkEqualTree = function(root) {

};


```

TypeScript Solution:

```

/**
 * Problem: Equal Tree Partition
 * Difficulty: Medium
 * 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 checkEqualTree(root: TreeNode | null): boolean {
}

```

C# Solution:

```

/*
* Problem: Equal Tree Partition
* Difficulty: Medium
* 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 bool CheckEqualTree(TreeNode root) {
}
}

```

C Solution:

```
/*
 * Problem: Equal Tree Partition
 * Difficulty: Medium
 * 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;
 * };
 */
bool checkEqualTree(struct TreeNode* root) {

}
```

Go Solution:

```
// Problem: Equal Tree Partition
// Difficulty: Medium
// 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 checkEqualTree(root *TreeNode) bool {  
    }  
}
```

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 checkEqualTree(root: TreeNode?): Boolean {  
        }  
    }
```

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 checkEqualTree(_ root: TreeNode?) -> Bool {  
}  
}  
}
```

Rust Solution:

```
// Problem: Equal Tree Partition  
// Difficulty: Medium  
// 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>>,<br/>  
//     pub right: Option<Rc<RefCell<TreeNode>>,<br/>  
// }  
//  
// 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 check_equal_tree(root: Option<Rc<RefCell<TreeNode>>>) -> bool {  
        }  
    }  
}
```

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 {Boolean}
def check_equal_tree(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 $root
 * @return Boolean
 */
function checkEqualTree($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 {  
bool checkEqualTree(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 checkEqualTree(root: TreeNode): Boolean = {  
  
}  
}
```

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 check_equal_tree(root :: TreeNode.t | nil) :: boolean
def check_equal_tree(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 check_equal_tree(Root :: #tree_node{} | null) -> boolean().
check_equal_tree(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 (check-equal-tree root)
  (-> (or/c tree-node? #f) boolean?)
    )
```