

Problem 2641: Cousins in Binary Tree II

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the

root

of a binary tree, replace the value of each node in the tree with the

sum of all its cousins' values

.

Two nodes of a binary tree are

cousins

if they have the same depth with different parents.

Return

the

root

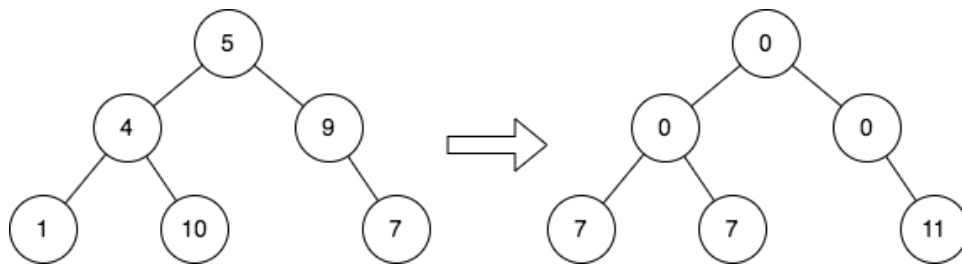
of the modified tree

.

Note

that the depth of a node is the number of edges in the path from the root node to it.

Example 1:



Input:

root = [5,4,9,1,10,null,7]

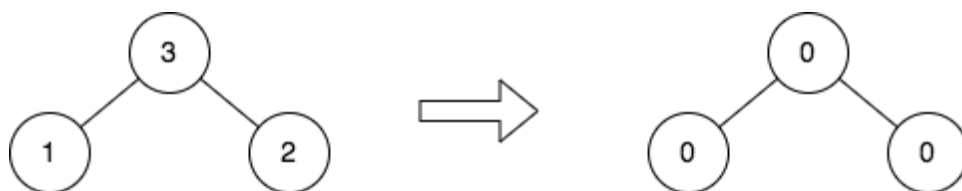
Output:

[0,0,0,7,7,null,11]

Explanation:

The diagram above shows the initial binary tree and the binary tree after changing the value of each node. - Node with value 5 does not have any cousins so its sum is 0. - Node with value 4 does not have any cousins so its sum is 0. - Node with value 9 does not have any cousins so its sum is 0. - Node with value 1 has a cousin with value 7 so its sum is 7. - Node with value 10 has a cousin with value 7 so its sum is 7. - Node with value 7 has cousins with values 1 and 10 so its sum is 11.

Example 2:



Input:

root = [3,1,2]

Output:

[0,0,0]

Explanation:

The diagram above shows the initial binary tree and the binary tree after changing the value of each node. - Node with value 3 does not have any cousins so its sum is 0. - Node with value 1 does not have any cousins so its sum is 0. - Node with value 2 does not have any cousins so its sum is 0.

Constraints:

The number of nodes in the tree is in the range

[1, 10

5

]

.

$1 \leq \text{Node.val} \leq 10$

4

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:
TreeNode* replaceValueInTree(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:
    TreeNode replaceValueInTree(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 replaceValueInTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

```

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

```

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

};

```

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 TreeNode ReplaceValueInTree(TreeNode root) {

}

}

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {

```

```

* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
struct TreeNode* replaceValueInTree(struct TreeNode* root) {

}

```

Go:

```

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

}

```

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 replaceValueInTree(root: TreeNode?): TreeNode? {

}

}

```

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 replaceValueInTree(_ root: TreeNode?) -> TreeNode? {

}

}

```

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 replace_value_in_tree(root: Option<Rc<RefCell<TreeNode>>>) ->
    Option<Rc<RefCell<TreeNode>>> {

    }

}

```

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 {TreeNode}

def replace_value_in_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 $root
 * @return TreeNode
 */
function replaceValueInTree($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 {
  TreeNode? replaceValueInTree(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 replaceValueInTree(root: TreeNode): TreeNode = {

  }
}

```

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 replace_value_in_tree(root :: TreeNode.t | nil) :: TreeNode.t | nil
  def replace_value_in_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 replace_value_in_tree(Root :: #tree_node{} | null) -> #tree_node{} |
null.
replace_value_in_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 (replace-value-in-tree root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f))
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Cousins in Binary Tree II
 * Difficulty: Medium
 * Tags: tree, hash, 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) {}
 *   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:
  TreeNode* replaceValueInTree(TreeNode* root) {

```

```
}  
};
```

Java Solution:

```
/**  
 * Problem: Cousins in Binary Tree II  
 * Difficulty: Medium  
 * Tags: tree, hash, 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 TreeNode replaceValueInTree(TreeNode root) {  
  
 }  
}
```

Python3 Solution:

```

"""
Problem: Cousins in Binary Tree II
Difficulty: Medium
Tags: tree, hash, 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 replaceValueInTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
# 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 replaceValueInTree(self, root):
"""
:type root: Optional[TreeNode]
:rtype: Optional[TreeNode]
"""

```

JavaScript Solution:

```

/**
 * Problem: Cousins in Binary Tree II
 * Difficulty: Medium
 * Tags: tree, hash, 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 {TreeNode}
*/
var replaceValueInTree = function(root) {

};

```

TypeScript Solution:

```

/**
* Problem: Cousins in Binary Tree II
* Difficulty: Medium
* Tags: tree, hash, 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 replaceValueInTree(root: TreeNode | null): TreeNode | null {

};

```

C# Solution:

```

/*
* Problem: Cousins in Binary Tree II
* Difficulty: Medium
* Tags: tree, hash, 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 TreeNode ReplaceValueInTree(TreeNode root) {

}

}

```


C Solution:

```
/*
 * Problem: Cousins in Binary Tree II
 * Difficulty: Medium
 * Tags: tree, hash, 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;
 * };
 */
struct TreeNode* replaceValueInTree(struct TreeNode* root) {

}
```

Go Solution:

```
// Problem: Cousins in Binary Tree II
// Difficulty: Medium
// Tags: tree, hash, 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 replaceValueInTree(root *TreeNode) *TreeNode {

}

```

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 replaceValueInTree(root: TreeNode?): TreeNode? {

    }
}

```

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 replaceValueInTree(_ root: TreeNode?) -> TreeNode? {

}

}

```

Rust Solution:

```

// Problem: Cousins in Binary Tree II
// Difficulty: Medium
// Tags: tree, hash, 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 replace_value_in_tree(root: Option<Rc<RefCell<TreeNode>>>) ->
    Option<Rc<RefCell<TreeNode>>> {

    }

}

```

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 {TreeNode}
def replace_value_in_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 $root
 * @return TreeNode
 */
function replaceValueInTree($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 {
  TreeNode? replaceValueInTree(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 replaceValueInTree(root: TreeNode): TreeNode = {

  }
}
```

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 replace_value_in_tree(root :: TreeNode.t | nil) :: TreeNode.t | nil
  def replace_value_in_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 replace_value_in_tree(Root :: #tree_node{} | null) -> #tree_node{} |
null.
replace_value_in_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 (replace-value-in-tree root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f))
)
```