

Problem 1609: Even Odd Tree

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

A binary tree is named

Even-Odd

if it meets the following conditions:

The root of the binary tree is at level index

0

, its children are at level index

1

, their children are at level index

2

, etc.

For every

even-indexed

level, all nodes at the level have

odd

integer values in

strictly increasing

order (from left to right).

For every

odd-indexed

level, all nodes at the level have

even

integer values in

strictly decreasing

order (from left to right).

Given the

root

of a binary tree,

return

true

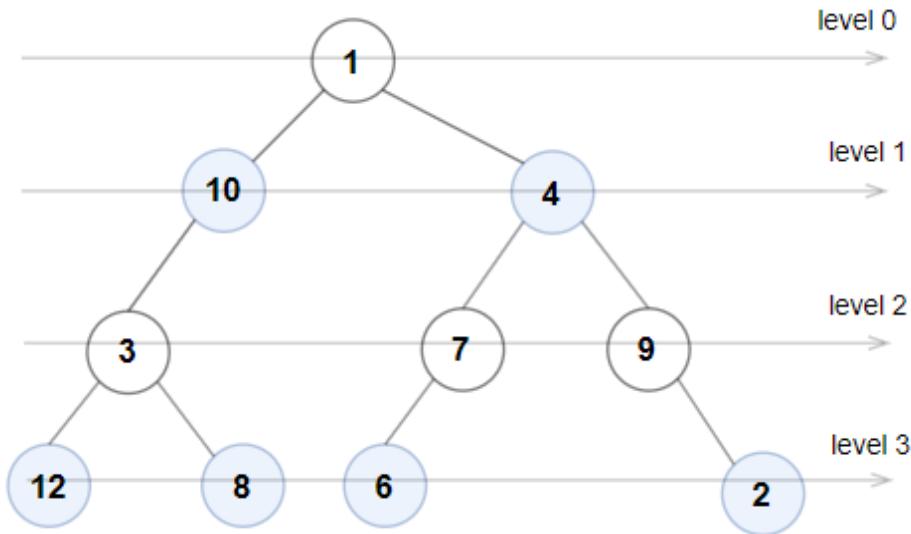
if the binary tree is

Even-Odd

, otherwise return

false

Example 1:



Input:

```
root = [1,10,4,3,null,7,9,12,8,6,null,null,2]
```

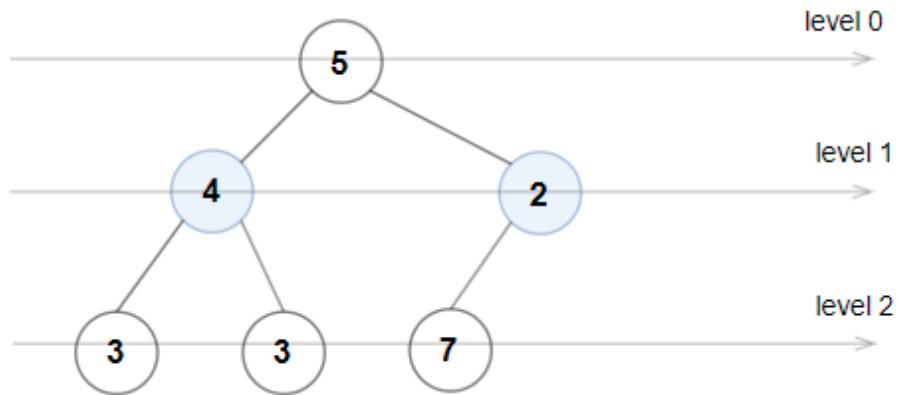
Output:

```
true
```

Explanation:

The node values on each level are: Level 0: [1] Level 1: [10,4] Level 2: [3,7,9] Level 3: [12,8,6,2] Since levels 0 and 2 are all odd and increasing and levels 1 and 3 are all even and decreasing, the tree is Even-Odd.

Example 2:



Input:

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

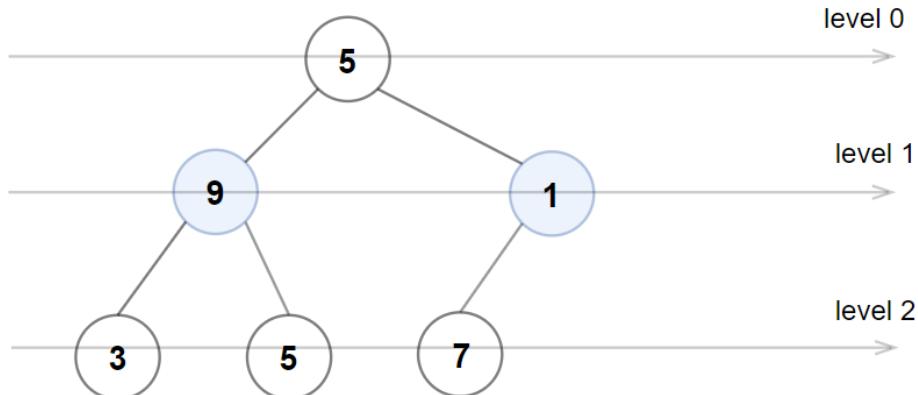
Output:

false

Explanation:

The node values on each level are: Level 0: [5] Level 1: [4,2] Level 2: [3,3,7] Node values in level 2 must be in strictly increasing order, so the tree is not Even-Odd.

Example 3:



Input:

root = [5,9,1,3,5,7]

Output:

false

Explanation:

Node values in the level 1 should be even integers.

Constraints:

The number of nodes in the tree is in the range

[1, 10]

5

]

.

1 <= Node.val <= 10

6

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 isEvenOddTree(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 isEvenOddTree(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 isEvenOddTree(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 isEvenOddTree(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 isEvenOddTree = 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 isEvenOddTree(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 IsEvenOddTree(TreeNode root) {
        }
    }
}

```

C:

```

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

```

```
bool isEvenOddTree(struct TreeNode* root) {  
}  
}
```

Go:

```
/**  
 * Definition for a binary tree node.  
 * type TreeNode struct {  
 *     Val int  
 *     Left *TreeNode  
 *     Right *TreeNode  
 * }  
 */  
func isEvenOddTree(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 isEvenOddTree(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 isEvenOddTree(_ 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 is_even_odd_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 is_even_odd_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 isEvenOddTree($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 isEvenOddTree(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 isEvenOddTree(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 is_even_odd_tree(root :: TreeNode.t | nil) :: boolean
def is_even_odd_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 is_even_odd_tree(Root :: #tree_node{} | null) -> boolean().
is_even_odd_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 (is-even-odd-tree root)
  (-> (or/c tree-node? #f) boolean?))
)

```

Solutions

C++ Solution:

```

/*
 * Problem: Even Odd Tree
 * 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 isEvenOddTree(TreeNode* root) {

}
};


```

Java Solution:

```

/**
 * Problem: Even Odd Tree
 * 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 isEvenOddTree(TreeNode root) {  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Even Odd Tree  
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 isEvenOddTree(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 isEvenOddTree(self, root):  
        """  
        :type root: Optional[TreeNode]  
        :rtype: bool
```

```
'''
```

JavaScript Solution:

```
/**  
 * Problem: Even Odd Tree  
 * 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 isEvenOddTree = function(root) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Even Odd Tree  
 * 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 isEvenOddTree(root: TreeNode | null): boolean {
}

```

C# Solution:

```

/*
 * Problem: Even Odd Tree
 * 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 IsEvenOddTree(TreeNode root) {
    }
}

```

C Solution:

```

/*
 * Problem: Even Odd Tree
 * 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 isEvenOddTree(struct TreeNode* root) {
}

```

Go Solution:

```

// Problem: Even Odd Tree
// 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 isEvenOddTree(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 isEvenOddTree(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 isEvenOddTree(_ root: TreeNode?) -> Bool {

}
}

```

Rust Solution:

```

// Problem: Even Odd Tree
// 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>>,
//   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 is_even_odd_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 is_even_odd_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 $root
 * @return Boolean
 */
function isEvenOddTree($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 isEvenOddTree(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 isEvenOddTree(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 is_even_odd_tree(root :: TreeNode.t | nil) :: boolean
  def is_even_odd_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 is_even_odd_tree(Root :: #tree_node{} | null) -> boolean().
is_even_odd_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 (is-even-odd-tree root)  
(-> (or/c tree-node? #f) boolean?)  
)
```