

# Problem 2415: Reverse Odd Levels of Binary Tree

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given the

root

of a

perfect

binary tree, reverse the node values at each

odd

level of the tree.

For example, suppose the node values at level 3 are

[2,1,3,4,7,11,29,18]

, then it should become

[18,29,11,7,4,3,1,2]

Return

the root of the reversed tree

A binary tree is

perfect

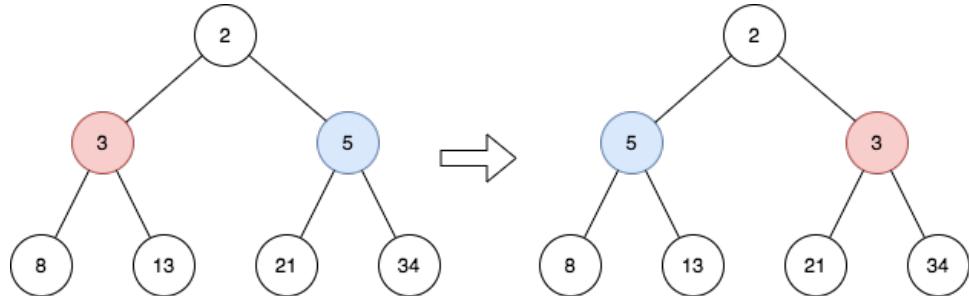
if all parent nodes have two children and all leaves are on the same level.

The

level

of a node is the number of edges along the path between it and the root node.

Example 1:



Input:

root = [2,3,5,8,13,21,34]

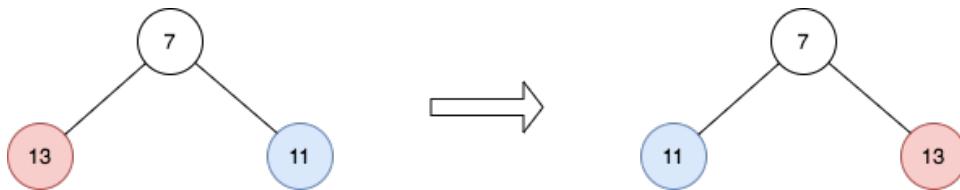
Output:

[2,5,3,8,13,21,34]

Explanation:

The tree has only one odd level. The nodes at level 1 are 3, 5 respectively, which are reversed and become 5, 3.

Example 2:



Input:

root = [7,13,11]

Output:

[7,11,13]

Explanation:

The nodes at level 1 are 13, 11, which are reversed and become 11, 13.

Example 3:

Input:

root = [0,1,2,0,0,0,0,1,1,1,1,2,2,2,2]

Output:

[0,2,1,0,0,0,0,2,2,2,2,1,1,1,1]

Explanation:

The odd levels have non-zero values. The nodes at level 1 were 1, 2, and are 2, 1 after the reversal. The nodes at level 3 were 1, 1, 1, 1, 2, 2, 2, 2, and are 2, 2, 2, 2, 1, 1, 1, 1 after the reversal.

Constraints:

The number of nodes in the tree is in the range

[1, 2

14

]

0 <= Node.val <= 10

5

root

is a

perfect

binary tree.

## 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* reverseOddLevels(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 reverseOddLevels(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 reverseOddLevels(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 reverseOddLevels(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 reverseOddLevels = 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 reverseOddLevels(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 ReverseOddLevels(TreeNode root) {
        }
    }
}

```

## C:

```

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

```

**Go:**

```
/**  
 * Definition for a binary tree node.  
 * type TreeNode struct {  
 *     Val int  
 *     Left *TreeNode  
 *     Right *TreeNode  
 * }  
 */  
func reverseOddLevels(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 reverseOddLevels(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 reverseOddLevels(_ 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 reverse_odd_levels(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 reverse_odd_levels(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 $root
 * @return TreeNode
 */
function reverseOddLevels($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? reverseOddLevels(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 reverseOddLevels(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 reverse_odd_levels(root :: TreeNode.t | nil) :: TreeNode.t | nil
def reverse_odd_levels(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 reverse_odd_levels(Root :: #tree_node{} | null) -> #tree_node{} | null.
reverse_odd_levels(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 (reverse-odd-levels root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f)))

```

```
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Reverse Odd Levels of Binary 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) {}
 *     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* reverseOddLevels(TreeNode* root) {
        }

    };
}
```

### Java Solution:

```
/**
 * Problem: Reverse Odd Levels of Binary 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 TreeNode reverseOddLevels(TreeNode root) {

}
}

```

### Python3 Solution:

```

"""
Problem: Reverse Odd Levels of Binary 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 reverseOddLevels(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 reverseOddLevels(self, root):
        """
:type root: Optional[TreeNode]
:rtype: Optional[TreeNode]
"""

```

## JavaScript Solution:

```

/**
 * Problem: Reverse Odd Levels of Binary 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 {TreeNode}
*/
var reverseOddLevels = function(root) {

};

```

### TypeScript Solution:

```

/**
* Problem: Reverse Odd Levels of Binary 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 reverseOddLevels(root: TreeNode | null): TreeNode | null {  
};
```

### C# Solution:

```
/*  
 * Problem: Reverse Odd Levels of Binary 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 TreeNode ReverseOddLevels(TreeNode root) {  
        }  
    }
```

### C Solution:

```
/*  
 * Problem: Reverse Odd Levels of Binary 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;
* };
*/
struct TreeNode* reverseOddLevels(struct TreeNode* root) {

}

```

## Go Solution:

```

// Problem: Reverse Odd Levels of Binary 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 reverseOddLevels(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 reverseOddLevels(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 reverseOddLevels(_ root: TreeNode?) -> TreeNode? {
        }

    }
}
```

## Rust Solution:

```
// Problem: Reverse Odd Levels of Binary 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 reverse_odd_levels(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 reverse_odd_levels(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 $root
     * @return TreeNode
     */
    function reverseOddLevels($root) {

    }
}

```

### Dart Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int val) { this.val = val; }
 * }
 */

```

```

* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
TreeNode? reverseOddLevels(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 reverseOddLevels(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 reverse_odd_levels(root :: TreeNode.t | nil) :: TreeNode.t | nil
def reverse_odd_levels(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 reverse_odd_levels(Root :: #tree_node{} | null) -> #tree_node{} | null.
reverse_odd_levels(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 (reverse-odd-levels root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f)))
)
```

