

Problem 968: Binary Tree Cameras

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given the

root

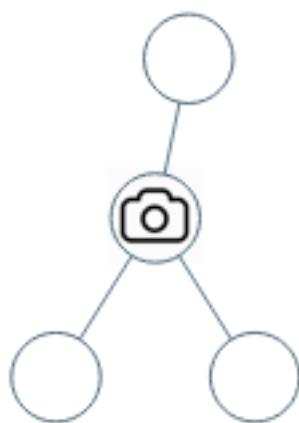
of a binary tree. We install cameras on the tree nodes where each camera at a node can monitor its parent, itself, and its immediate children.

Return

the minimum number of cameras needed to monitor all nodes of the tree

.

Example 1:



Input:

root = [0,0,null,0,0]

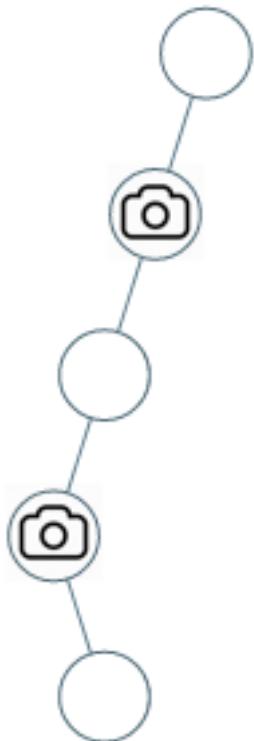
Output:

1

Explanation:

One camera is enough to monitor all nodes if placed as shown.

Example 2:



Input:

root = [0,0,null,0,null,0,null,0]

Output:

2

Explanation:

At least two cameras are needed to monitor all nodes of the tree. The above image shows one of the valid configurations of camera placement.

Constraints:

The number of nodes in the tree is in the range

[1, 1000]

Node.val == 0

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 minCameraCover(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 int minCameraCover(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 minCameraCover(self, root: Optional[TreeNode]) -> 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 minCameraCover(self, root):
        """

```

```
:type root: Optional[TreeNode]
: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
 * @return {number}
 */
var minCameraCover = 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 minCameraCover(root: TreeNode | null): 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 MinCameraCover(TreeNode root) {
        }
    }
}
```

C:

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

Go:

```
/*
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
```

```

* }
*/
func minCameraCover(root *TreeNode) 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 minCameraCover(root: TreeNode?): 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 minCameraCover(_ root: TreeNode?) -> 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 min_camera_cover(root: Option<Rc<RefCell<TreeNode>>>) -> 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
# @return {Integer}
def min_camera_cover(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 $root
 * @return Integer
 */
function minCameraCover($root) {

}

}
}

```

Dart:

```

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

```

```

* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
int minCameraCover(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 minCameraCover(root: TreeNode): 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 min_camera_cover(root :: TreeNode.t() | nil) :: integer

```

```

def min_camera_cover(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 min_camera_cover(Root :: #tree_node{} | null) -> integer().
min_camera_cover(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 (min-camera-cover root)
  (-> (or/c tree-node? #f) exact-integer?))

```

Solutions

C++ Solution:

```
/*
 * Problem: Binary Tree Cameras
 * 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 minCameraCover(TreeNode* root) {

    }
};

}
```

Java Solution:

```

/**
 * Problem: Binary Tree Cameras
 * 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 minCameraCover(TreeNode root) {
        }

    }
}

```

Python3 Solution:

```

"""
Problem: Binary Tree Cameras
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 minCameraCover(self, root: Optional[TreeNode]) -> 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 minCameraCover(self, root):
        """
:type root: Optional[TreeNode]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Binary Tree Cameras
 * 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
 * @return {number}
 */
var minCameraCover = function(root) {

};

```

TypeScript Solution:

```

/**
 * Problem: Binary Tree Cameras
 * 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 minCameraCover(root: TreeNode | null): number {
}

```

C# Solution:

```

/*
 * Problem: Binary Tree Cameras
 * 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 MinCameraCover(TreeNode root) {
 *
 *     }
 * }

```

C Solution:

```

/*
 * Problem: Binary Tree Cameras
 * 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 minCameraCover(struct TreeNode* root) {

}

```

Go Solution:

```

// Problem: Binary Tree Cameras
// 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 minCameraCover(root *TreeNode) 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 minCameraCover(root: TreeNode?): Int {  
        // Implementation  
    }  
}
```

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 minCameraCover(_ root: TreeNode?) -> Int {  
        // Implementation  
    }  
}
```

```
}
```

```
}
```

Rust Solution:

```
// Problem: Binary Tree Cameras
// 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 min_camera_cover(root: Option<Rc<RefCell<TreeNode>>>) -> 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
# @return {Integer}
def min_camera_cover(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 $root
 * @return Integer
 */
function minCameraCover($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 {
    int minCameraCover(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 minCameraCover(root: TreeNode): 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 min_camera_cover(root :: TreeNode.t | nil) :: integer
def min_camera_cover(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 min_camera_cover(Root :: #tree_node{} | null) -> integer().
min_camera_cover(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 (min-camera-cover root)
  (-> (or/c tree-node? #f) exact-integer?))

```

