Minimum number of cameras required to monitor all nodes of a Binary Tree

Difficulty Level : Hard • Last Updated : 18 Jun, 2021

Given a Binary Tree consisting of **N** nodes, the task is to find the minimum number of cameras required to monitor the entire tree such that every camera placed at any node can monitor the node itself, its parent, and its immediate children.

Examples:

Input:

```
      0
     /
    0
   /\
  0  0
```

Output: *1*

Explanation:

```
      0
     /
    0 <----- Camera
   / \
```

In the above tree, the nodes which are bold are the nodes having the camera. Placing the camera at the level 1 of the Tree can monitor all the nodes of the given Binary Tree.
Therefore, the minimum count of camera needed is 1.

Input:

```
      0
     /
    0
   /
  0
   \
    0
```

***Output:** 2*

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

Approach: The given problem can be solved by storing the states of the nodes whether the camera has been placed or not or the node is monitored by any other node having the camera or not. The idea is to perform the DFS Traversal on the given tree and return the states of each node in each recursive call. Consider the following conversion as the states returned by the function:

- If the value is **1**, the node is monitored.
- If the value is **2**, the node is not monitored.
- If the value is **3**, the node has the camera.

- Initialize a variable, say **count** to store the minimum number of the camera required to monitor all the nodes of the given tree.
- Create a function, say **dfs(root)** that takes the root of the given tree and returns the status of each node whether the camera has been placed or not, or the node is monitored by any other node having the camera and perform the following steps:
  - If the value of the node is **NULL**, then return **1** as the **NULL** node is always monitored.
  - Recursively call for the left and the right subtree and store the value return by them in the variables **L** and **R**.
  - If the value of **L** and **R** is **1** then return **2** from the current recursive call as the current root node is not monitored.
  - If the value of **L** or **R** is **2** then increment the value of **count** by **1** as one of the left and the right node is not monitored and return **3**.
  - Otherwise, return **1**.
- Call the above recursive function from the root and if the value returned by it is **2**, then increment the value of **count** by **1**.
- After completing the above steps, print the value of **count** as the resultant number of cameras.

Below is the implementation of the above approach:

C++

```
// C++ program for the above approach

#include <bits/stdc++.h>
using namespace std;

// Structure for a Tree node
struct Node {
    int key;
    struct Node *left, *right;
```

```
// Function to create a new node
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;

    // Return the newly created node
    return (temp);
}

// Stores the minimum number of
// cameras required
int cnt = 0;

// Utility function to find minimum
// number of cameras required to
// monitor the entire tree
int minCameraSetupUtil(Node* root)
{
    // If root is NULL
    if (root == NULL)
        return 1;

    int L = minCameraSetupUtil(
        root->left);
    int R = minCameraSetupUtil(
        root->right);

    // Both the nodes are monitored
    if (L == 1 && R == 1)
        return 2;

    // If one of the left and the
    // right subtree is not monitored
    else if (L == 2 || R == 2) {
        cnt++;
        return 3;
    }

    // If the root node is monitored
    return 1;
```

```
// of cameras required to monitor
// entire tree
void minCameraSetup(Node* root)
{
    int value = minCameraSetupUtil(root);

    // Print the count of cameras
    cout << cnt + (value == 2 ? 1 : 0);
}

// Driver Code
int main()
{
    // Given Binary Tree
    Node* root = newNode(0);
    root->left = newNode(0);
    root->left->left = newNode(0);
    root->left->left->left = newNode(0);
    root->left->left->left->right = newNode(0);

    minCameraSetup(root);

    return 0;
}
```

Java

```
// Java program for the above approach
public class GFG {
    // TreeNode class
    static class Node {
        public int key;
        public Node left, right;
    };

    static Node newNode(int key)
    {
        Node temp = new Node();
        temp.key = key;
        temp.left = temp.right = null;
        return temp;
    }
```

```
// cameras required
static int cnt = 0;

// Utility function to find minimum
// number of cameras required to
// monitor the entire tree
static int minCameraSetupUtil(Node root)
{
    // If root is NULL
    if (root == null)
        return 1;

    int L = minCameraSetupUtil(root.left);
    int R = minCameraSetupUtil(root.right);

    // Both the nodes are monitored
    if (L == 1 && R == 1)
        return 2;

    // If one of the left and the
    // right subtree is not monitored
    else if (L == 2 || R == 2) {
        cnt++;
        return 3;
    }

    // If the root node is monitored
    return 1;
}

// Function to find the minimum number
// of cameras required to monitor
// entire tree
static void minCameraSetup(Node root)
{
    int value = minCameraSetupUtil(root);

    // Print the count of cameras
    System.out.println(cnt + (value == 2 ? 1 : 0));
}

// Driver code
```

```
        Node root = newNode(0);
        root.left = newNode(0);
        root.left.left = newNode(0);
        root.left.left.left = newNode(0);
        root.left.left.left.right = newNode(0);

        minCameraSetup(root);
    }
}
// This code is contributed by abhinavjain194
```

Python3

```
# Python3 program for the above approach

# Structure for a Tree node
class Node:

    def __init__(self, k):

        self.key = k
        self.left = None
        self.right = None

# Stores the minimum number of
# cameras required
cnt = 0

# Utility function to find minimum
# number of cameras required to
# monitor the entire tree
def minCameraSetupUtil(root):

    global cnt

    # If root is None
    if (root == None):
        return 1

    L = minCameraSetupUtil(root.left)
    R = minCameraSetupUtil(root.right)
```

```
            return 2

        # If one of the left and the
        # right subtree is not monitored
        elif (L == 2 or R == 2):
            cnt += 1
            return 3

        # If the root node is monitored
        return 1

# Function to find the minimum number
# of cameras required to monitor
# entire tree
def minCameraSetup(root):

    value = minCameraSetupUtil(root)

    # Print the count of cameras
    print(cnt + (1 if value == 2 else 0))

# Driver Code
if __name__ == '__main__':

    # Given Binary Tree
    root = Node(0)
    root.left = Node(0)
    root.left.left = Node(0)
    root.left.left.left = Node(0)
    root.left.left.left.right = Node(0)

    minCameraSetup(root)

# This code is contributed by mohit kumar 29
```

C#

```
// C# program for the above approach
using System;
using System.Collections.Generic;
public class GFG
{
```

```
class Node {
    public int key;
    public Node left, right;
};

static Node newNode(int key)
{
    Node temp = new Node();
    temp.key = key;
    temp.left = temp.right = null;
    return temp;
}

// Stores the minimum number of
// cameras required
static int cnt = 0;

// Utility function to find minimum
// number of cameras required to
// monitor the entire tree
static int minCameraSetupUtil(Node root)
{

    // If root is NULL
    if (root == null)
        return 1;

    int L = minCameraSetupUtil(root.left);
    int R = minCameraSetupUtil(root.right);

    // Both the nodes are monitored
    if (L == 1 && R == 1)
        return 2;

    // If one of the left and the
    // right subtree is not monitored
    else if (L == 2 || R == 2) {
        cnt++;
        return 3;
    }

    // If the root node is monitored
```

```
    // Function to find the minimum number
    // of cameras required to monitor
    // entire tree
    static void minCameraSetup(Node root)
    {
        int value = minCameraSetupUtil(root);

        // Print the count of cameras
        Console.WriteLine(cnt + (value == 2 ? 1 : 0));
    }

    // Driver code
    public static void Main(String[] args)
    {
        // Given Binary Tree
        Node root = newNode(0);
        root.left = newNode(0);
        root.left.left = newNode(0);
        root.left.left.left = newNode(0);
        root.left.left.left.right = newNode(0);

        minCameraSetup(root);
    }
}

// This code is contributed by Amit Katiyar
```

Javascript

```
<script>
    // Javascript program for the above approach

    // TreeNode class
    class Node
    {
        constructor(key) {
           this.left = null;
           this.right = null;
           this.key = key;
        }
    }
```

```
    let temp = new Node(key);
    return temp;
}

// Stores the minimum number of
// cameras required
let cnt = 0;

// Utility function to find minimum
// number of cameras required to
// monitor the entire tree
function minCameraSetupUtil(root)
{

    // If root is NULL
    if (root == null)
        return 1;

    let L = minCameraSetupUtil(root.left);
    let R = minCameraSetupUtil(root.right);

    // Both the nodes are monitored
    if (L == 1 && R == 1)
        return 2;

    // If one of the left and the
    // right subtree is not monitored
    else if (L == 2 || R == 2) {
        cnt++;
        return 3;
    }

    // If the root node is monitored
    return 1;
}

// Function to find the minimum number
// of cameras required to monitor
// entire tree
function minCameraSetup(root)
{
    let value = minCameraSetupUtil(root);
```

```
    }

    // Given Binary Tree
    let root = newNode(0);
    root.left = newNode(0);
    root.left.left = newNode(0);
    root.left.left.left = newNode(0);
    root.left.left.left.right = newNode(0);

    minCameraSetup(root);

// This code is contributed by suresh07.
</script>
```

Output:

```
2
```

Time Complexity: *O(N)*

Auxiliary Space: *O(H), where H is the [height of the given tree](https://www.geeksforgeeks.org/).*

Like 7

Previous

Smallest index that splits an array into two subarrays with equal product

Next

Check if sum of digits of a number exceeds the product of digits of that number

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Minimum swap required to convert binary tree to binary search tree**
28, Jan 17

02 **Convert given Binary Tree to Symmetric Tree by adding minimum number of nodes**
15, Oct 21

03 **Minimum time required to visit all the special nodes of a Tree**
12, May 20

04 **Count the nodes of the tree which make a pangram when concatenated with the sub-tree nodes**
28, May 19

05 **Count of leaf nodes required to be removed at each step to empty a given Binary Tree**
18, Dec 20

06 **Count nodes from all lower levels smaller than minimum valued node of current level for every level in a Binary Tree**
14, Sep 20

07 **Sum of nodes in a binary tree having only the left child nodes**
09, Nov 21

08 **Count of nodes in a binary tree having their nodes in range [L, R]**
22, Dec 21

Article Contributed By :

rutvikchandla3
@rutvikchandla3

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : mohit kumar 29, abhinavjain194, amit143katiyar, suresh07

Article Tags : Binary Tree, DFS, PostOrder Traversal, tree-traversal, Dynamic Programming, Mathematical, Recursion, Tree

Practice Tags : Dynamic Programming, Mathematical, Recursion, DFS, Tree

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Learn

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

Contribute

Write an Article

Improve an Article

NodeJS