Problem 1379: Find a Corresponding Node of a Binary Tree in a Clone of That Tree

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given two binary trees

original

and

cloned

and given a reference to a node

target

in the original tree.

The

cloned

tree is a

copy of

the

original

tree.

Return

a reference to the same node

in the

cloned

tree.

Note

that you are

not allowed

to change any of the two trees or the

target

node and the answer

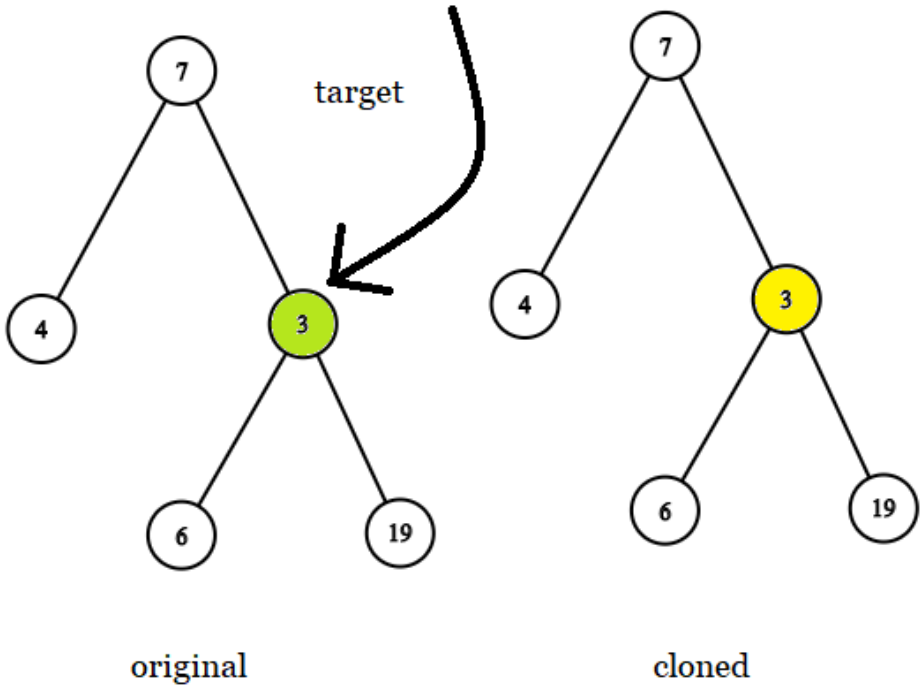
must be

a reference to a node in the

cloned

tree.

Example 1:



Input:

tree = [7,4,3,null,null,6,19], target = 3

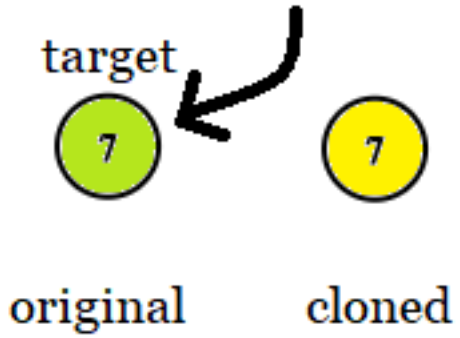
Output:

3

Explanation:

In all examples the original and cloned trees are shown. The target node is a green node from the original tree. The answer is the yellow node from the cloned tree.

Example 2:



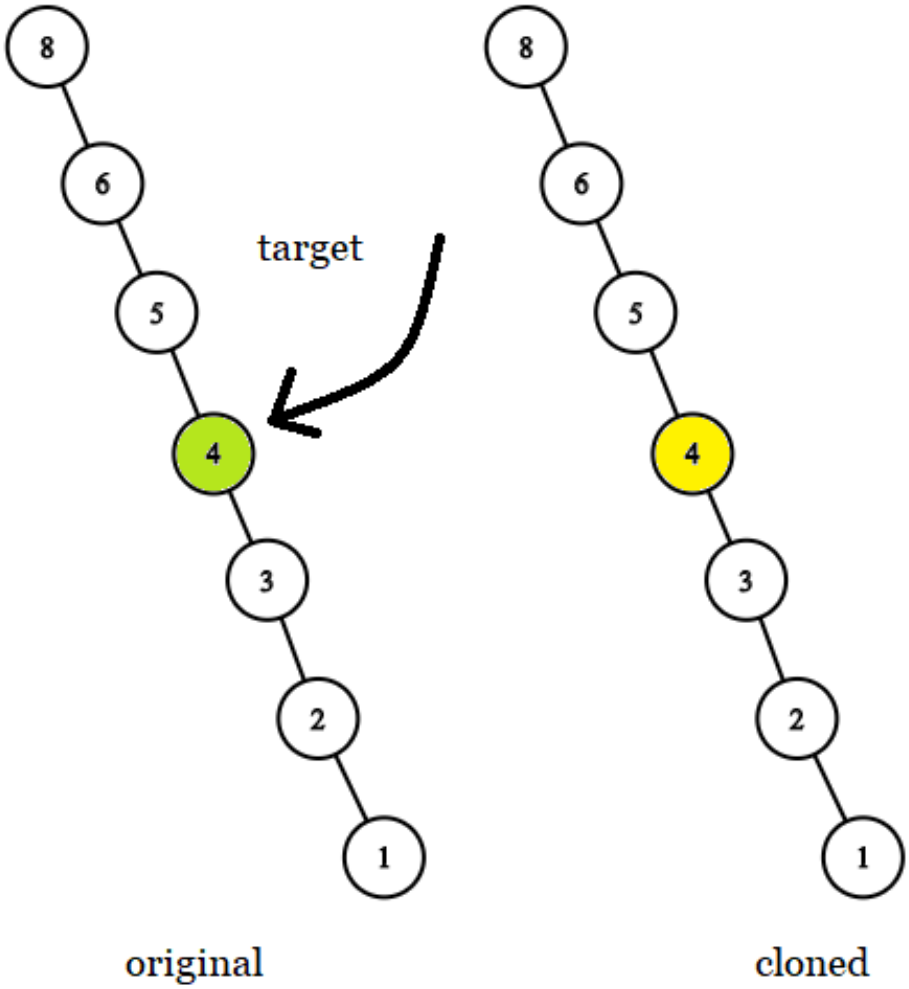
Input:

tree = [7], target = 7

Output:

7

Example 3:



Input:

tree = [8,null,6,null,5,null,4,null,3,null,2,null,1], target = 4

Output:

4

Constraints:

The number of nodes in the

tree

is in the range

[1, 10

4

]

.

The values of the nodes of the

tree

are unique.

target

node is a node from the

original

tree and is not

null

.

Follow up:

Could you solve the problem if repeated values on the tree are allowed?

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
```

```
* TreeNode *left;
* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/

class Solution {
public:
TreeNode* getTargetCopy(TreeNode* original, TreeNode* cloned, TreeNode*
target) {

}
};
```

Java:

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

class Solution {
public final TreeNode getTargetCopy(final TreeNode original, final TreeNode
cloned, final TreeNode target) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None
```

```
class Solution:
def getTargetCopy(self, original: TreeNode, cloned: TreeNode, target:
TreeNode) -> TreeNode:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def getTargetCopy(self, original, cloned, target):
"""
:type original: TreeNode
:type cloned: TreeNode
:type target: TreeNode
:rtype: TreeNode
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} original
* @param {TreeNode} cloned
* @param {TreeNode} target
* @return {TreeNode}
*/

var getTargetCopy = function(original, cloned, target) {

};
```

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 getTargetCopy(original: TreeNode | null, cloned: TreeNode | null,
target: 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 x) { val = x; }
* }
*/

public class Solution {
public TreeNode GetTargetCopy(TreeNode original, TreeNode cloned, TreeNode
target) {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
* Difficulty: Easy
* 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(int x) : val(x), left(NULL), right(NULL) {}
* };
*/

class Solution {
public:
TreeNode* getTargetCopy(TreeNode* original, TreeNode* cloned, TreeNode*
target) {

}
};
```

Java Solution:

```
/**
* Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
* Difficulty: Easy
* 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(int x) { val = x; }
* }
*/

class Solution {
public final TreeNode getTargetCopy(final TreeNode original, final TreeNode
cloned, final TreeNode target) {

}
}
```

Python3 Solution:

```
"""
Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
Difficulty: Easy
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, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def getTargetCopy(self, original: TreeNode, cloned: TreeNode, target:
TreeNode) -> TreeNode:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def getTargetCopy(self, original, cloned, target):
"""
:type original: TreeNode
:type cloned: TreeNode
:type target: TreeNode
:rtype: TreeNode
"""
```

JavaScript Solution:

```
/**
* Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
* Difficulty: Easy
* 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) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} original
```

```
* @param {TreeNode} cloned
* @param {TreeNode} target
* @return {TreeNode}
*/

var getTargetCopy = function(original, cloned, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
* Difficulty: Easy
* 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 getTargetCopy(original: TreeNode | null, cloned: TreeNode | null,
target: TreeNode | null): TreeNode | null {

};
```

C# Solution:

```
/*
* Problem: Find a Corresponding Node of a Binary Tree in a Clone of That Tree
* Difficulty: Easy
* 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 x) { val = x; }
* }
*/

public class Solution {
public TreeNode GetTargetCopy(TreeNode original, TreeNode cloned, TreeNode
target) {

}
}
```