Problem 1660: Correct a Binary Tree

Problem Information

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

Problem Description

You have a binary tree with a small defect. There is

exactly one

invalid node where its right child incorrectly points to another node at the

same depth

but to the

invalid node's right

.

Given the root of the binary tree with this defect,

root

, return

the root of the binary tree after

removing

this invalid node

and every node underneath it

(minus the node it incorrectly points to).

Custom testing:

The test input is read as 3 lines:

TreeNode root

int fromNode

(

not available to

correctBinaryTree

)

int toNode

(

not available to

correctBinaryTree

)

After the binary tree rooted at

root

is parsed, the

TreeNode

with value of

fromNode

will have its right child pointer pointing to the

TreeNode

with a value of

toNode

. Then,

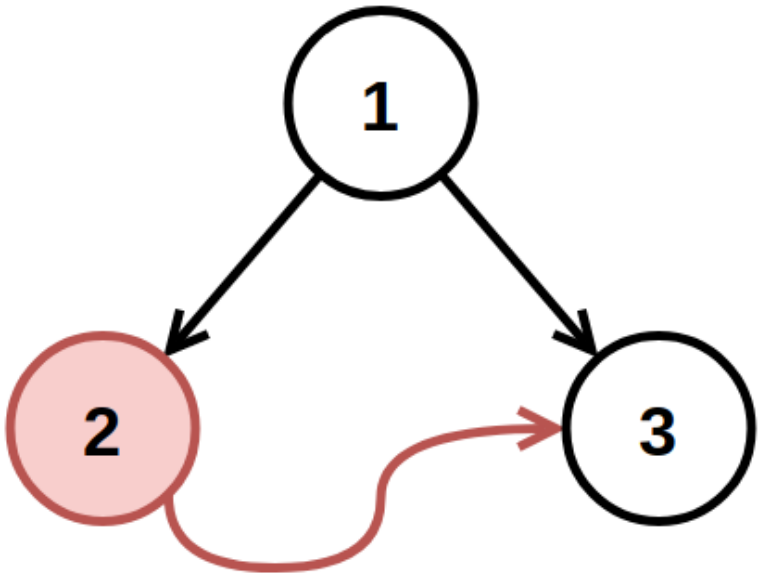
root

is passed to

correctBinaryTree

.

Example 1:



Input:

root = [1,2,3], fromNode = 2, toNode = 3

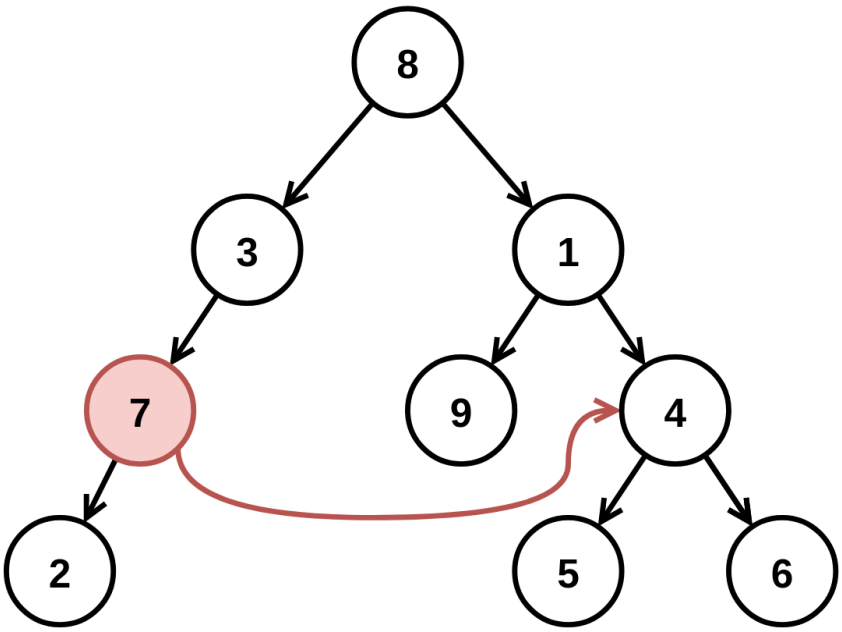
Output:

[1,null,3]

Explanation:

The node with value 2 is invalid, so remove it.

Example 2:



Input:

root = [8,3,1,7,null,9,4,2,null,null,null,5,6], fromNode = 7, toNode = 4

Output:

[8,3,1,null,null,9,4,null,null,5,6]

Explanation:

The node with value 7 is invalid, so remove it and the node underneath it, node 2.

Constraints:

The number of nodes in the tree is in the range

[3, 10

4

]

.

-10

9

<= Node.val <= 10

9

All

Node.val

are

unique

.

fromNode != toNode

fromNode

and

toNode

will exist in the tree and will be on the same depth.

toNode

is to the

right

of

fromNode

.

fromNode.right

is

null

in the initial tree from the test data.

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* correctBinaryTree(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 correctBinaryTree(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 correctBinaryTree(self, root: TreeNode) -> 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 correctBinaryTree(self, root):
"""
:type root: TreeNode
:rtype: 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
* @param {number} from
* @param {number} to
* @return {TreeNode}
*/
var correctBinaryTree = function(root) {

};
```

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 CorrectBinaryTree(TreeNode root) {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Correct a Binary Tree
* Difficulty: Medium
* Tags: tree, hash, 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* correctBinaryTree(TreeNode* root) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Correct a Binary Tree
* Difficulty: Medium
* Tags: tree, hash, 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 correctBinaryTree(TreeNode root) {

}
}
```

Python3 Solution:

```
"""
Problem: Correct a Binary Tree
Difficulty: Medium
Tags: tree, hash, 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 correctBinaryTree(self, root: TreeNode) -> 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 correctBinaryTree(self, root):
"""
:type root: TreeNode
:rtype: TreeNode
"""
```

JavaScript Solution:

```
/**
* Problem: Correct a Binary Tree
* Difficulty: Medium
* Tags: tree, hash, 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
* @param {number} from
* @param {number} to
* @return {TreeNode}
*/
var correctBinaryTree = function(root) {

};
```

C# Solution:

```
/*
* Problem: Correct a Binary Tree
* Difficulty: Medium
* Tags: tree, hash, 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 CorrectBinaryTree(TreeNode root) {

}
}
```