Problem 1261: Find Elements in a Contaminated Binary Tree

Problem Information

Difficulty: Medium
Acceptance Rate: 84.08%
Paid Only: No
Tags: Hash Table, Tree, Depth-First Search, Breadth-First Search, Design, Binary Tree

Problem Description

Given a binary tree with the following rules:

1. `root.val == 0` 2. For any `treeNode`: 1. If `treeNode.val` has a value `x` and `treeNode.left != null`, then `treeNode.left.val == 2 * x + 1` 2. If `treeNode.val` has a value `x` and `treeNode.right != null`, then `treeNode.right.val == 2 * x + 2`

Now the binary tree is contaminated, which means all `treeNode.val` have been changed to `-1`.

Implement the `FindElements` class:

* `FindElements(TreeNode* root)` Initializes the object with a contaminated binary tree and recovers it. * `bool find(int target)` Returns `true` if the `target` value exists in the recovered binary tree.

Example 1:

![](https://assets.leetcode.com/uploads/2019/11/06/untitled-diagram-4-1.jpg)

Input ["FindElements","find","find"] [[[-1,null,-1]],[1],[2]] **Output** [null,false,true] **Explanation** FindElements findElements = new FindElements([-1,null,-1]); findElements.find(1); // return False findElements.find(2); // return True

Example 2:

![](https://assets.leetcode.com/uploads/2019/11/06/untitled-diagram-4.jpg)

Input ["FindElements","find","find","find"] [[[-1,-1,-1,-1,-1]],[1],[3],[5]] **Output** [null,true,true,false] **Explanation** FindElements findElements = new FindElements([-1,-1,-1,-1,-1]); findElements.find(1); // return True findElements.find(3); // return True findElements.find(5); // return False

Example 3:

![](https://assets.leetcode.com/uploads/2019/11/07/untitled-diagram-4-1-1.jpg)

Input ["FindElements","find","find","find","find"] [[[-1,null,-1,-1,null,-1]],[2],[3],[4],[5]] **Output** [null,true,false,false,true] **Explanation** FindElements findElements = new FindElements([-1,null,-1,-1,null,-1]); findElements.find(2); // return True findElements.find(3); // return False findElements.find(4); // return False findElements.find(5); // return True

Constraints:

* `TreeNode.val == -1` * The height of the binary tree is less than or equal to `20` * The total number of nodes is between `[1, 104]` * Total calls of `find()` is between `[1, 104]` * `0 <= target <= 106`

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 FindElements {
```

```
public:
FindElements(TreeNode* root) {

}

bool find(int target) {

}
};

/**
* Your FindElements object will be instantiated and called as such:
* FindElements* obj = new FindElements(root);
* bool param_1 = obj->find(target);
*/
```

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 FindElements {

public FindElements(TreeNode root) {

}

public boolean find(int target) {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* FindElements obj = new FindElements(root);
* boolean param_1 = obj.find(target);
*/
```

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 FindElements:

def __init__(self, root: Optional[TreeNode]):


def find(self, target: int) -> bool:



# Your FindElements object will be instantiated and called as such:
# obj = FindElements(root)
# param_1 = obj.find(target)
```