1261. Find Elements in a Contaminated Binary Tree

Given a binary tree with the following rules:

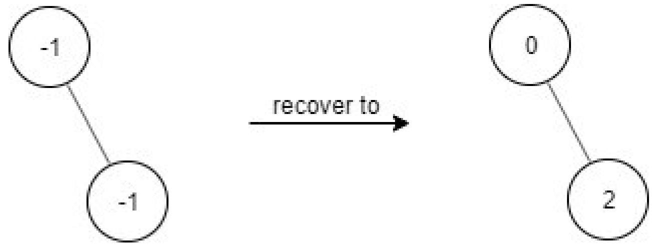
1. `root.val == 0`
2. For any `treeNode`:
   a. If `treeNode.val` has a value `x` and `treeNode.left != null`, then `treeNode.left.val == 2 * x + 1`
   b. If `treeNode.val` has a value `x` and `treeNode.right != null`, then `treeNode.right.val == 2 * x + 2`

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

Implement the `FindElements` class:

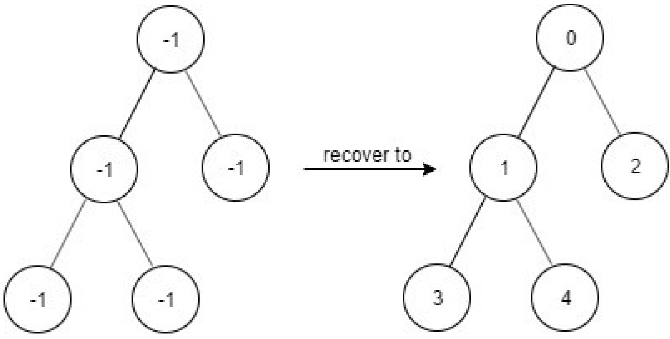
- `FindElements(TreeNode* root)` Initializes the object with a contaminated binary tree and recovers it.
- `bool find(int target)` Returns `true` if the `target` value exists in the recovered binary tree.

Example 1:

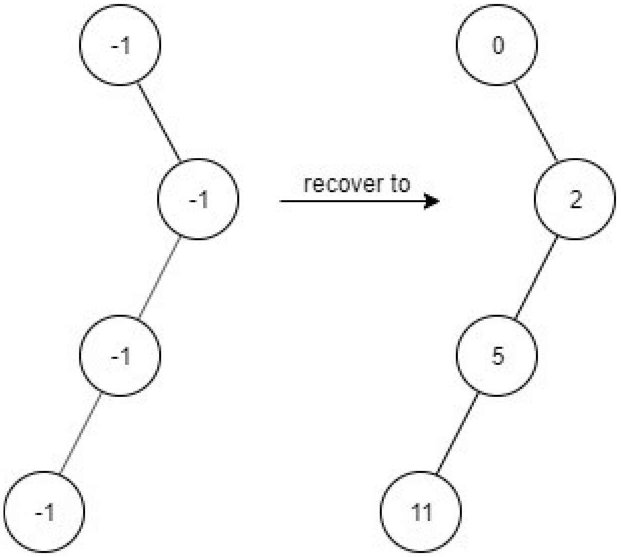


```
Input
["FindElements","find","find"]
[[[-1,null,-1]],[1],[2]]
Output
[null,false,true]
Explanation
FindElements findElements = new
FindElements([-1,null,-1]);
findElements.find(1); // return False
findElements.find(2); // return True
```

Example 2:



```
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
```



```
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`

Overview

We are given a binary tree $root$ which follows the following 3 rules:

1. The value of the root node $root$ is always 0
2. Given a node in the tree with value x , the value of its left child (if it exists) is always $2x+1$
3. Given a node in the tree with value x , the value of its right child (if it exists) is always $2x+2$

This tree is then "contaminated", which means the values of all nodes are overwritten to -1 . We now have to find out what values existed in the tree before it was contaminated. We do this by implementing two functions:

1. $FindElements(TreeNode*root)$ is our constructor that gives us the contaminated binary tree $root$, and recovers the tree.
2. $bool\, find(\text{int}\, target)$ should return whether or not $target$ is one of the original values in $root$ before contamination

1261. Find Elements in a Contaminated Binary Tree

```
/*
    DFS preorder traversal
*/
/**
 * 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 {
    private:
        std::unordered_set<int> values;
    public:
        // Time complexity: O(n)
        // Space complexity: O(2n)
        FindElements(TreeNode* root) {
            auto dfs=[&](TreeNode* node,int val,auto& self)->void{
                if(!node) return;
                node->val=val;
                values.insert(val);
                self(node->left,2*node->val+1,self);
                self(node->right,2*node->val+2,self);
            };

            dfs(root,0,dfs);
        }

        // Time complexity: Ω(1),O(n)
        // Space complexity: O(1)
        bool find(int target) {
            return values.find(target)!=values.end();
        }
};

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

1261. Find Elements in a Contaminated Binary Tree

```
/*
    BFS
*/
/**
 * 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 {
    private:
        std::unordered_set<int> values;
    public:
        // Time complexity: O(n)
        // Space complexity: O(3n)
        FindElements(TreeNode* root) {
            auto bfs=[&](TreeNode* node)->void{
                std::queue<std::pair<TreeNode*,int>> q;
                q.push({root,0});
                while(!q.empty()){
                    auto [node,node_val]=q.front();
                    q.pop();
                    node->val=node_val;
                    values.insert(node_val);
                    if(node->left) q.push({node->left,2*node_val+1});
                    if(node->right) q.push({node->right,2*node_val+2});
                }
            };
            bfs(root);
        }
```

```
     // Time complexity: Ω(1),O(n)
    // Space complexity: O(1)
    bool find(int target) {
        return values.find(target)!=values.end();
    }
};

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