```
/*
给出一个满足下述规则的二叉树：

    root.val == 0
    如果 treeNode.val == x 且 treeNode.left != null，那么 treeNode.left.val == 2 * x + 1
    如果 treeNode.val == x 且 treeNode.right != null，那么 treeNode.right.val == 2 * x + 2

现在这个二叉树受到「污染」，所有的 treeNode.val 都变成了 -1。

请你先还原二叉树，然后实现 FindElements 类：

    FindElements(TreeNode* root) 用受污染的二叉树初始化对象，你需要先把它还原。
    bool find(int target) 判断目标值 target 是否存在于还原后的二叉树中并返回结果。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/find-elements-in-a-contaminated-binary-tree
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

- 方法一:递归还原+DFS遍历查找
- 方法二:递归还原+map去重查找
- 方法三:优化递归还原+map去重查找

方法一:C++_递归还原+DFS遍历查找

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class FindElements
{
    private:

        TreeNode* rroot     =   NULL;
        bool      ret_val   =   false;

        void __recoverTree(TreeNode* node, int parent_val)
        {

            if(node==NULL)
            {
                return;
```

```
            }

            if(node->left)
            {
                node->left->val = 2*parent_val + 1;
                __recoverTree(node->left,2*parent_val + 1);
            }

            if(node->right)
            {
                node->right->val = 2*parent_val + 2;
                __recoverTree(node->right,2*parent_val + 2);
            }
        }


        void __dfs(TreeNode* node, int target)
        {
            if(node==NULL)
            {
                return;
            }

            if(node->val == target)
            {
                ret_val = true;
                return;
            }



            __dfs(node->left,target);
            __dfs(node->right,target);
        }


    public:
        FindElements(TreeNode* root)
        {
            rroot = root;
            if(root)
            {
                root->val = 0;
                __recoverTree(root,0);
            }

            return;

        }

        bool find(int target)
        {
            ret_val = false;

            __dfs(rroot,target);

            return ret_val;

```

```
        }
};

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



/*
执行结果：
通过
显示详情
执行用时 :1288 ms, 在所有 cpp 提交中击败了6.53% 的用户
内存消耗 :17.8 MB, 在所有 cpp 提交中击败了100.00%的用户
 */
```

方法二:C++_递归还原+map去重查找

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class FindElements
{
    private:

        TreeNode*       rroot     =   NULL  ;
        bool            ret_val   =   false ;
        map<int,int>    mii                 ;

        void __recoverTree(TreeNode* node, int parent_val)
        {

            if(node==NULL)
            {
                return;
            }

            if(node->left)
            {
                node->left->val = 2*parent_val + 1;
                __recoverTree(node->left,2*parent_val + 1);
                mii[2*parent_val + 1] = 0;
            }

            if(node->right)
            {
                node->right->val = 2*parent_val + 2;
```

```
                __recoverTree(node->right,2*parent_val + 2);
                mii[2*parent_val + 2] = 0;
            }
        }


    public:
        FindElements(TreeNode* root)
        {
            rroot = root;
            if(root)
            {
                root->val = 0;
                mii[0] = 0;
                __recoverTree(root,0);
            }

            return;

        }

        bool find(int target)
        {
           return (bool)(mii.count(target));
        }
};


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



/*
执行结果:
通过
显示详情
执行用时 :80 ms, 在所有 cpp 提交中击败了37.76% 的用户
内存消耗 :30 MB, 在所有 cpp 提交中击败了100.00%的用户
 */
```

方法三:优化递归还原+map去重查找

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class FindElements
```

```
{
    private:

        bool            ret_val   =   false ;
        map<int,int>    mii                 ;

        void __recoverTree(TreeNode* node, int parent_val)
        {

            if(node==NULL)
            {
                return;
            }

            node->val = parent_val;
            mii[parent_val] = 0;
            __recoverTree(node->left,2*parent_val+1);
            __recoverTree(node->right,2*parent_val+2);
        }


    public:
        FindElements(TreeNode* root)
        {
            if(root)
            {
                mii[0] = 0;
                __recoverTree(root,0);
            }

            return;

        }

        bool find(int target)
        {
           return (bool)(mii.count(target));
        }
};


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



/*
执行结果：
通过
显示详情
执行用时 :56 ms, 在所有 cpp 提交中击败了62.94% 的用户
内存消耗 :30.2 MB, 在所有 cpp 提交中击败了100.00%的用户
 */
```

AlimyBreak
2019.11.18