

```

1  /*
2  给出一个满足下述规则的二叉树：
3
4      root.val == 0
5      如果 treeNode.val == x 且 treeNode.left != null, 那么 treeNode.left.val
== 2 * x + 1
6      如果 treeNode.val == x 且 treeNode.right != null, 那么 treeNode.right.val
== 2 * x + 2
7
8  现在这个二叉树受到「污染」，所有的 treeNode.val 都变成了 -1。
9
10 请你先还原二叉树，然后实现 FindElements 类：
11
12      FindElements(TreeNode* root) 用受污染的二叉树初始化对象，你需要先把它还原。
13      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。
14
15 来源：力扣（LeetCode）
16 链接：https://leetcode-cn.com/problems/find-elements-in-a-contaminated-
binary-tree
17 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。
18 */

```

- 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

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10 class FindElements
11 {
12     private:
13
14         TreeNode* rroot      = NULL;
15         bool      ret_val    = false;
16
17         void __recoverTree(TreeNode* node, int parent_val)
18         {
19
20             if(node==NULL)
21             {
22                 return;

```

```

23     }
24
25     if(node->left)
26     {
27         node->left->val = 2*parent_val + 1;
28         __recoverTree(node->left, 2*parent_val + 1);
29     }
30
31     if(node->right)
32     {
33         node->right->val = 2*parent_val + 2;
34         __recoverTree(node->right, 2*parent_val + 2);
35     }
36 }
37
38
39 void __dfs(TreeNode* node, int target)
40 {
41     if(node==NULL)
42     {
43         return;
44     }
45
46     if(node->val == target)
47     {
48         ret_val = true;
49         return;
50     }
51
52
53
54     __dfs(node->left, target);
55     __dfs(node->right, target);
56 }
57
58
59 public:
60 FindElements(TreeNode* root)
61 {
62     rroot = root;
63     if(root)
64     {
65         root->val = 0;
66         __recoverTree(root, 0);
67     }
68
69     return;
70
71 }
72
73 bool find(int target)
74 {
75     ret_val = false;
76
77     __dfs(rroot, target);
78
79     return ret_val;
80

```

```

81     }
82 };
83
84 /**
85  * Your FindElements object will be instantiated and called as such:
86  * FindElements* obj = new FindElements(root);
87  * bool param_1 = obj->find(target);
88  */
89
90
91
92 /*
93 执行结果:
94 通过
95 显示详情
96 执行用时 :1288 ms, 在所有 cpp 提交中击败了6.53% 的用户
97 内存消耗 :17.8 MB, 在所有 cpp 提交中击败了100.00%的用户
98  */

```

## 方法二:C++\_递归还原+map去重查找

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10 class FindElements
11 {
12 private:
13
14     TreeNode*      rroot      =    NULL   ;
15     bool           ret_val    =    false   ;
16     map<int,int>    mii        ;
17
18     void __recoverTree(TreeNode* node, int parent_val)
19     {
20
21         if(node==NULL)
22         {
23             return;
24         }
25
26         if(node->left)
27         {
28             node->left->val = 2*parent_val + 1;
29             __recoverTree(node->left, 2*parent_val + 1);
30             mii[2*parent_val + 1] = 0;
31         }
32
33         if(node->right)
34         {
35             node->right->val = 2*parent_val + 2;

```

```

36         __recoverTree(node->right, 2*parent_val + 2);
37         mii[2*parent_val + 2] = 0;
38     }
39 }
40
41
42 public:
43     FindElements(TreeNode* root)
44     {
45         root = root;
46         if(root)
47         {
48             root->val = 0;
49             mii[0] = 0;
50             __recoverTree(root, 0);
51         }
52
53         return;
54
55     }
56
57     bool find(int target)
58     {
59         return (bool)(mii.count(target));
60     }
61 };
62
63
64 /**
65  * Your FindElements object will be instantiated and called as such:
66  * FindElements* obj = new FindElements(root);
67  * bool param_1 = obj->find(target);
68  */
69
70
71
72 /*
73 执行结果:
74 通过
75 显示详情
76 执行用时 :80 ms, 在所有 cpp 提交中击败了37.76% 的用户
77 内存消耗 :30 MB, 在所有 cpp 提交中击败了100.00%的用户
78 */

```

### 方法三:优化递归还原+map去重查找

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10 class FindElements

```

```

11 {
12     private:
13
14         bool        ret_val    =    false ;
15         map<int,int>   mii          ;
16
17         void __recoverTree(TreeNode* node, int parent_val)
18         {
19
20             if(node==NULL)
21             {
22                 return;
23             }
24
25             node->val = parent_val;
26             mii[parent_val] = 0;
27             __recoverTree(node->left,2*parent_val+1);
28             __recoverTree(node->right,2*parent_val+2);
29         }
30
31
32     public:
33     FindElements(TreeNode* root)
34     {
35         if(root)
36         {
37             mii[0] = 0;
38             __recoverTree(root,0);
39         }
40
41         return;
42
43     }
44
45     bool find(int target)
46     {
47         return (bool)(mii.count(target));
48     }
49 };
50
51
52 /**
53  * Your FindElements object will be instantiated and called as such:
54  * FindElements* obj = new FindElements(root);
55  * bool param_1 = obj->find(target);
56  */
57
58
59
60 /*
61 执行结果:
62 通过
63 显示详情
64 执行用时 :56 ms, 在所有 cpp 提交中击败了62.94% 的用户
65 内存消耗 :30.2 MB, 在所有 cpp 提交中击败了100.00%的用户
66  */

```

