Problem 1261: Find Elements in a Contaminated Binary Tree

Problem Information

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

Problem Description

Given a binary tree with the following rules:

root.val == 0

For any

treeNode

:

If

treeNode.val

has a value

x

and

treeNode.left != null

, then

treeNode.left.val == 2 * x + 1

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

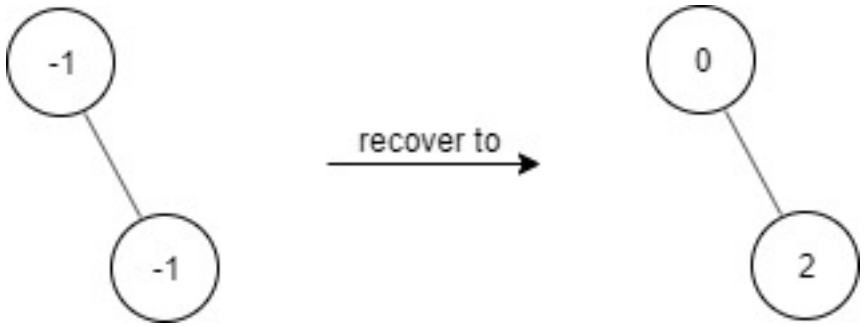
true

if the

target

value exists in the recovered binary tree.

Example 1:



Input

["FindElements","find","find"] [[[-1,null,-1]],[1],[2]]

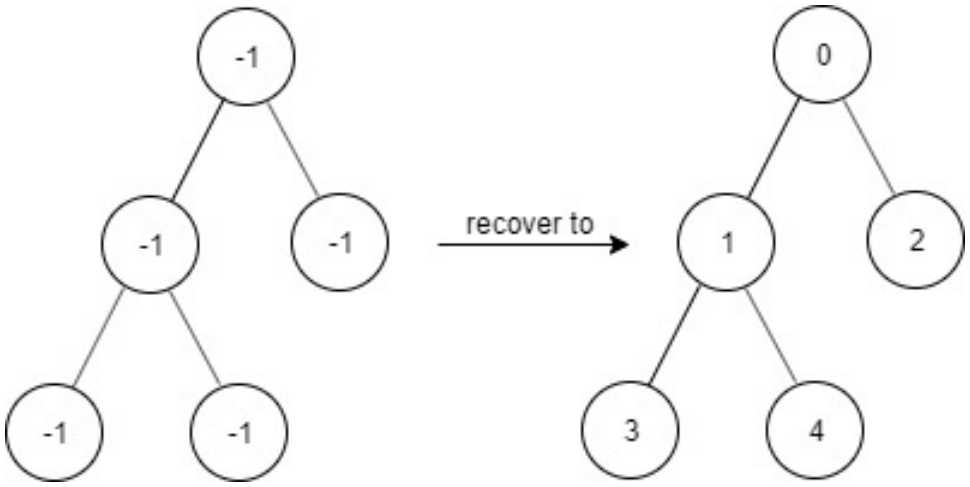
Output

[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]]

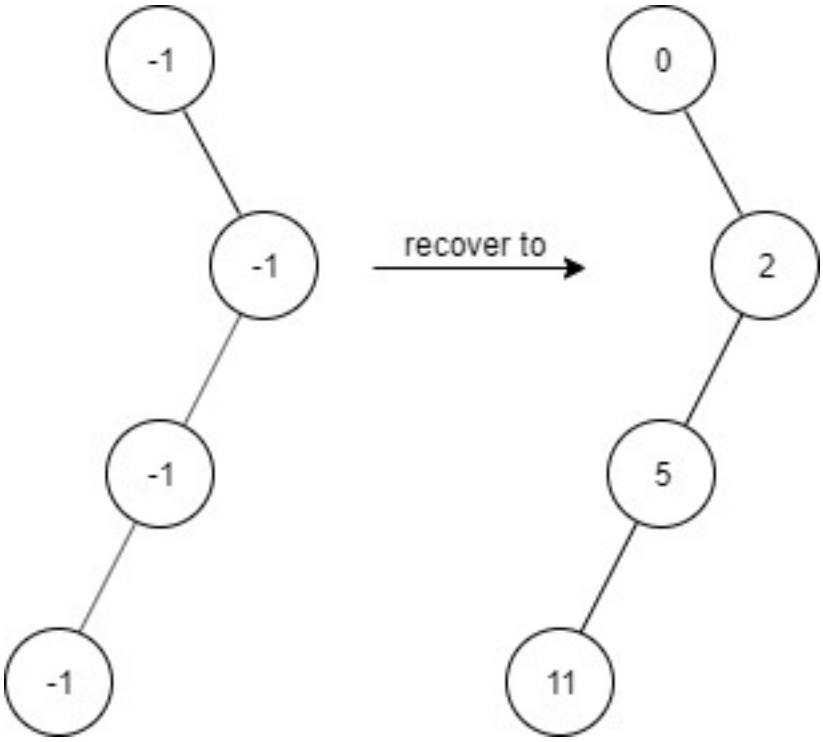
Output

[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

Example 3:



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, 10

4

]

Total calls of

find()

is between

[1, 10

4

]

0 <= target <= 10

6

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)
```

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 FindElements(object):

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

```
"""



def find(self, target):
"""
:type target: int
:rtype: bool
"""



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

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
*/
var FindElements = function(root) {

};

/**
* @param {number} target
* @return {boolean}
*/
FindElements.prototype.find = function(target) {

};

/**
* Your FindElements object will be instantiated and called as such:
```

```
* var obj = new FindElements(root)
* var param_1 = obj.find(target)
*/
```

TypeScript:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

class FindElements {
constructor(root: TreeNode | null) {

}

find(target: number): boolean {

}
}

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

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 FindElements {

public FindElements(TreeNode root) {

}

public 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);
*/
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/



typedef struct {
```

```
} FindElements;


FindElements* findElementsCreate(struct TreeNode* root) {

}

bool findElementsFind(FindElements* obj, int target) {

}

void findElementsFree(FindElements* obj) {

}

/**
* Your FindElements struct will be instantiated and called as such:
* FindElements* obj = findElementsCreate(root);
* bool param_1 = findElementsFind(obj, target);

* findElementsFree(obj);
*/
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
type FindElements struct {

}


func Constructor(root *TreeNode) FindElements {

}
```

```
func (this *FindElements) Find(target int) bool {

}


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

Kotlin:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class FindElements(root: TreeNode?) {

fun find(target: Int): Boolean {

}

}

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

Swift:

```
/**
* Definition for a binary tree node.
```

```
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/

class FindElements {

init(_ root: TreeNode?) {

}

func find(_ target: Int) -> Bool {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* let obj = FindElements(root)
* let ret_1: Bool = obj.find(target)
*/
```

Rust:

```
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
```

```
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
struct FindElements {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FindElements {

fn new(root: Option<Rc<RefCell<TreeNode>>>) -> Self {

}

fn find(&self, target: i32) -> bool {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* let obj = FindElements::new(root);
* let ret_1: bool = obj.find(target);
*/
```

Ruby:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
```

```
# @val = val
# @left = left
# @right = right
# end
# end
class FindElements

=begin
:type root: TreeNode
=end
def initialize(root)

end


=begin
:type target: Integer
:rtype: Boolean
=end
def find(target)

end


end

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

PHP:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
```

```
* }
* }
*/
class FindElements {
/**
* @param TreeNode $root
*/
function __construct($root) {

}

/**
* @param Integer $target
* @return Boolean
*/
function find($target) {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* $obj = FindElements($root);
* $ret_1 = $obj->find($target);
*/
```

Dart:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class FindElements {

FindElements(TreeNode? root) {

}
```

```
bool find(int target) {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* FindElements obj = FindElements(root);
* bool param1 = obj.find(target);
*/
```

Scala:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
class FindElements(_root: TreeNode) {

def find(target: Int): Boolean = {

}

}

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

Elixir:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
```

```
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule FindElements do
@spec init_(root :: TreeNode.t | nil) :: any
def init_(root) do

end

@spec find(target :: integer) :: boolean
def find(target) do

end
end

# Your functions will be called as such:
# FindElements.init_(root)
# param_1 = FindElements.find(target)

# FindElements.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec find_elements_init_(Root :: #tree_node{} | null) -> any().
find_elements_init_(Root) ->
.

-spec find_elements_find(Target :: integer()) -> boolean().
find_elements_find(Target) ->
.
```

```
%% Your functions will be called as such:
%% find_elements_init_(Root),
%% Param_1 = find_elements_find(Target),

%% find_elements_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define find-elements%
(class object%
(super-new)

; root : (or/c tree-node? #f)
(init-field
root)

; find : exact-integer? -> boolean?
(define/public (find target)
)))

;; Your find-elements% object will be instantiated and called as such:
;; (define obj (new find-elements% [root root]))
;; (define param_1 (send obj find target))
```

Solutions

C++ Solution:

```
/*
* Problem: Find Elements in a Contaminated 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) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
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 Solution:

```
/**
* Problem: Find Elements in a Contaminated 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 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 Solution:

```
"""
Problem: Find Elements in a Contaminated 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 FindElements:

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


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

```
# 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 FindElements(object):

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



def find(self, target):
"""
:type target: int
:rtype: bool
"""




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

JavaScript Solution:

```
/**
* Problem: Find Elements in a Contaminated 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
*/
var FindElements = function(root) {

};

/**
* @param {number} target
* @return {boolean}
*/
FindElements.prototype.find = function(target) {

};

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

TypeScript Solution:

```
/**
* Problem: Find Elements in a Contaminated 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 {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

class FindElements {
constructor(root: TreeNode | null) {

}

find(target: number): boolean {

}
}

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

C# Solution:

```
/*
* Problem: Find Elements in a Contaminated 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 FindElements {

public FindElements(TreeNode root) {

}

public 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);
*/
```

C Solution:

```
/*
* Problem: Find Elements in a Contaminated 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;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/



typedef struct {

} FindElements;


FindElements* findElementsCreate(struct TreeNode* root) {

}

bool findElementsFind(FindElements* obj, int target) {

}

void findElementsFree(FindElements* obj) {

}

/**
* Your FindElements struct will be instantiated and called as such:
* FindElements* obj = findElementsCreate(root);
* bool param_1 = findElementsFind(obj, target);

* findElementsFree(obj);
*/
```

Go Solution:

```
// Problem: Find Elements in a Contaminated 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.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
type FindElements struct {

}


func Constructor(root *TreeNode) FindElements {

}


func (this *FindElements) Find(target int) bool {

}


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

Kotlin Solution:

```
/**
* Example:
```

```
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class FindElements(root: TreeNode?) {

fun find(target: Int): Boolean {

}

}

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

Swift Solution:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
```

```
class FindElements {

init(_ root: TreeNode?) {

}

func find(_ target: Int) -> Bool {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* let obj = FindElements(root)
* let ret_1: Bool = obj.find(target)
*/
```

Rust Solution:

```
// Problem: Find Elements in a Contaminated 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.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
```

```
// right: None
// }
// }
// }
struct FindElements {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FindElements {

fn new(root: Option<Rc<RefCell<TreeNode>>>) -> Self {

}

fn find(&self, target: i32) -> bool {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* let obj = FindElements::new(root);
* let ret_1: bool = obj.find(target);
*/
```

Ruby Solution:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
```

```
class FindElements

=begin
:type root: TreeNode
=end
def initialize(root)

end


=begin
:type target: Integer
:rtype: Boolean
=end
def find(target)

end


end

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

PHP Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class FindElements {
```

```
/**
* @param TreeNode $root
*/
function __construct($root) {

}

/**
* @param Integer $target
* @return Boolean
*/
function find($target) {

}
}

/**
* Your FindElements object will be instantiated and called as such:
* $obj = FindElements($root);
* $ret_1 = $obj->find($target);
*/
```

Dart Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class FindElements {

FindElements(TreeNode? root) {

}

bool find(int target) {
```

```
}
}

/**
* Your FindElements object will be instantiated and called as such:
* FindElements obj = FindElements(root);
* bool param1 = obj.find(target);
*/
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
class FindElements(_root: TreeNode) {

def find(target: Int): Boolean = {

}

}

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

Elixir Solution:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
```

```
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule FindElements do
@spec init_(root :: TreeNode.t | nil) :: any
def init_(root) do

end

@spec find(target :: integer) :: boolean
def find(target) do

end
end

# Your functions will be called as such:
# FindElements.init_(root)
# param_1 = FindElements.find(target)

# FindElements.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec find_elements_init_(Root :: #tree_node{} | null) -> any().
find_elements_init_(Root) ->
.

-spec find_elements_find(Target :: integer()) -> boolean().
find_elements_find(Target) ->
.
```

```
%% Your functions will be called as such:
%% find_elements_init_(Root),
%% Param_1 = find_elements_find(Target),

%% find_elements_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define find-elements%
(class object%
(super-new)

; root : (or/c tree-node? #f)
(init-field
root)

; find : exact-integer? -> boolean?
(define/public (find target)
)))

;; Your find-elements% object will be instantiated and called as such:
;; (define obj (new find-elements% [root root]))
;; (define param_1 (send obj find target))
```