Problem 998: Maximum Binary Tree II

Problem Information

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

Problem Description

A

maximum tree

is a tree where every node has a value greater than any other value in its subtree.

You are given the

root

of a maximum binary tree and an integer

val

.

Just as in the

previous problem

, the given tree was constructed from a list

a

(

root = Construct(a)

) recursively with the following

Construct(a)

routine:

If

a

is empty, return

null

.

Otherwise, let

a[i]

be the largest element of

a

. Create a

root

node with the value

a[i]

.

The left child of

root

will be

Construct([a[0], a[1], ..., a[i - 1]])

.

The right child of

root

will be

Construct([a[i + 1], a[i + 2], ..., a[a.length - 1]])

.

Return

root

.

Note that we were not given

a

directly, only a root node

root = Construct(a)

.

Suppose

b

is a copy of

a

with the value

val

appended to it. It is guaranteed that

b

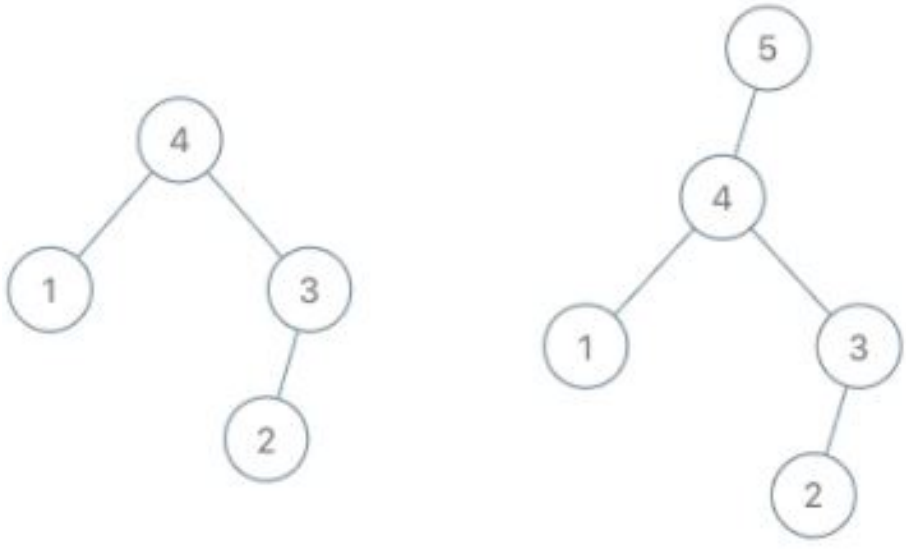
has unique values.

Return

Construct(b)

.

Example 1:



Input:

root = [4,1,3,null,null,2], val = 5

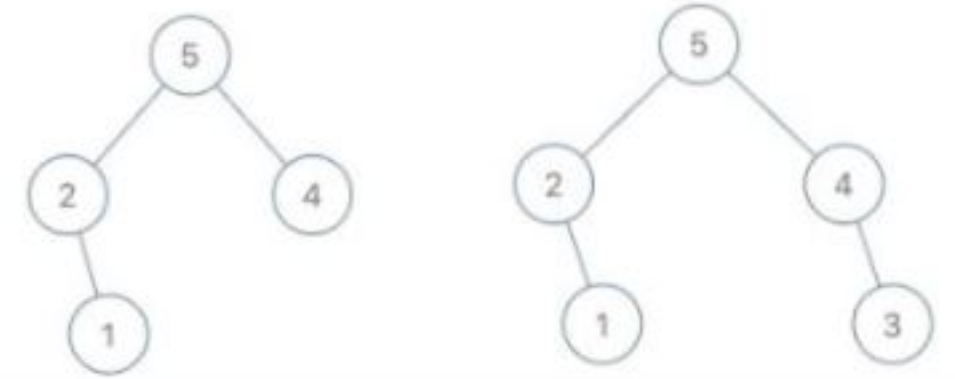
Output:

[5,4,null,1,3,null,null,2]

Explanation:

a = [1,4,2,3], b = [1,4,2,3,5]

Example 2:



Input:

root = [5,2,4,null,1], val = 3

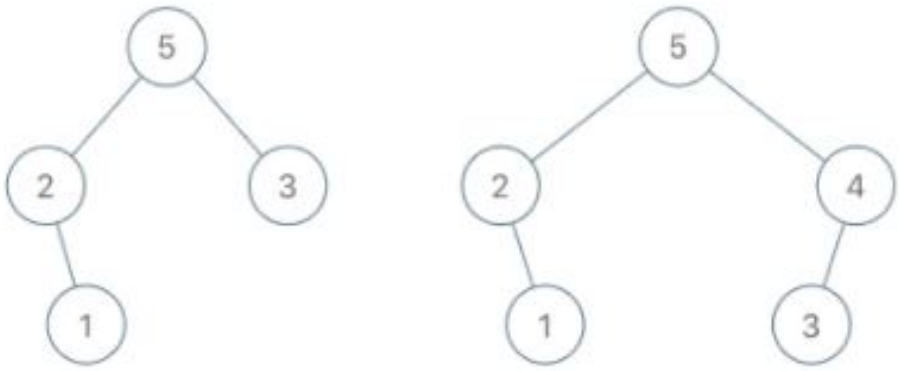
Output:

[5,2,4,null,1,null,3]

Explanation:

a = [2,1,5,4], b = [2,1,5,4,3]

Example 3:



Input:

root = [5,2,3,null,1], val = 4

Output:

[5,2,4,null,1,3]

Explanation:

a = [2,1,5,3], b = [2,1,5,3,4]

Constraints:

The number of nodes in the tree is in the range

[1, 100]

.

1 <= Node.val <= 100

All the values of the tree are

unique

.

1 <= val <= 100

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 Solution {
public:
TreeNode* insertIntoMaxTree(TreeNode* root, int val) {

}
};
```

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 Solution {
```

```
public TreeNode insertIntoMaxTree(TreeNode root, int val) {

}
}
```

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 Solution:
def insertIntoMaxTree(self, root: Optional[TreeNode], val: int) ->
Optional[TreeNode]:
```

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 Solution(object):
def insertIntoMaxTree(self, root, val):
"""
:type root: Optional[TreeNode]
:type val: int
:rtype: Optional[TreeNode]
"""
```

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
* @param {number} val
* @return {TreeNode}
*/
var insertIntoMaxTree = function(root, val) {

};
```

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)
* }
* }
*/

function insertIntoMaxTree(root: TreeNode | null, val: number): TreeNode |
null {

};
```

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 Solution {
public TreeNode InsertIntoMaxTree(TreeNode root, int val) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
struct TreeNode* insertIntoMaxTree(struct TreeNode* root, int val) {

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func insertIntoMaxTree(root *TreeNode, val int) *TreeNode {

}
```

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 Solution {
fun insertIntoMaxTree(root: TreeNode?, `val`: Int): TreeNode? {

}
}
```

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 Solution {
func insertIntoMaxTree(_ root: TreeNode?, _ val: Int) -> TreeNode? {

}
}
```

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
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn insert_into_max_tree(root: Option<Rc<RefCell<TreeNode>>>, val: i32) ->
Option<Rc<RefCell<TreeNode>>> {

}
}
```

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
# @param {TreeNode} root
# @param {Integer} val
# @return {TreeNode}
def insert_into_max_tree(root, val)
```

```
end
```

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 Solution {

/**
* @param TreeNode $root
* @param Integer $val
* @return TreeNode
*/
function insertIntoMaxTree($root, $val) {

}
}
```

Dart:

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

```
}
}
```

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
* }
*/
object Solution {
def insertIntoMaxTree(root: TreeNode, `val`: Int): TreeNode = {

}
}
```

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 Solution do
@spec insert_into_max_tree(root :: TreeNode.t | nil, val :: integer) ::
TreeNode.t | nil
def insert_into_max_tree(root, val) do

end
end
```

Erlang:

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

-spec insert_into_max_tree(Root :: #tree_node{} | null, Val :: integer()) ->
#tree_node{} | null.
insert_into_max_tree(Root, Val) ->
.
```

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/contract (insert-into-max-tree root val)
(-> (or/c tree-node? #f) exact-integer? (or/c tree-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Binary Tree II
* Difficulty: Medium
```

```
* Tags: tree
*
* 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) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
TreeNode* insertIntoMaxTree(TreeNode* root, int val) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Binary Tree II
* Difficulty: Medium
* Tags: tree
*
* 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 Solution {
public TreeNode insertIntoMaxTree(TreeNode root, int val) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Binary Tree II
Difficulty: Medium
Tags: tree

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 Solution:
def insertIntoMaxTree(self, root: Optional[TreeNode], val: int) ->
```

```
Optional[TreeNode]:
# 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 Solution(object):
def insertIntoMaxTree(self, root, val):
"""
:type root: Optional[TreeNode]
:type val: int
:rtype: Optional[TreeNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Binary Tree II
* Difficulty: Medium
* Tags: tree
*
* 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
```

```
* @param {number} val
* @return {TreeNode}
*/
var insertIntoMaxTree = function(root, val) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Binary Tree II
* Difficulty: Medium
* Tags: tree
*
* 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)
* }
* }
*/

function insertIntoMaxTree(root: TreeNode | null, val: number): TreeNode |
null {

};
```

C# Solution:

```
/*
* Problem: Maximum Binary Tree II
* Difficulty: Medium
* Tags: tree
*
* 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 Solution {
public TreeNode InsertIntoMaxTree(TreeNode root, int val) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Binary Tree II
* Difficulty: Medium
* Tags: tree
*
* 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;
* };
*/
struct TreeNode* insertIntoMaxTree(struct TreeNode* root, int val) {

}
```

Go Solution:

```
// Problem: Maximum Binary Tree II
// Difficulty: Medium
// Tags: tree
//
// 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
* }
*/
func insertIntoMaxTree(root *TreeNode, val int) *TreeNode {

}
```

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 Solution {
fun insertIntoMaxTree(root: TreeNode?, `val`: Int): TreeNode? {

}
}
```

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 Solution {
func insertIntoMaxTree(_ root: TreeNode?, _ val: Int) -> TreeNode? {

}
}
```

Rust Solution:

```
// Problem: Maximum Binary Tree II
// Difficulty: Medium
// Tags: tree
//
// 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
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn insert_into_max_tree(root: Option<Rc<RefCell<TreeNode>>>, val: i32) ->
Option<Rc<RefCell<TreeNode>>> {

}
}
```

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
```

```
# @param {TreeNode} root
# @param {Integer} val
# @return {TreeNode}
def insert_into_max_tree(root, val)

end
```

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 Solution {

/**
* @param TreeNode $root
* @param Integer $val
* @return TreeNode
*/
function insertIntoMaxTree($root, $val) {

}
}
```

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 Solution {
TreeNode? insertIntoMaxTree(TreeNode? root, int val) {

}
}
```

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
* }
*/
object Solution {
def insertIntoMaxTree(root: TreeNode, `val`: Int): TreeNode = {

}
}
```

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 Solution do
```

```
@spec insert_into_max_tree(root :: TreeNode.t | nil, val :: integer) ::
TreeNode.t | nil
def insert_into_max_tree(root, val) do

end
end
```

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 insert_into_max_tree(Root :: #tree_node{} | null, Val :: integer()) ->
#tree_node{} | null.
insert_into_max_tree(Root, Val) ->
.
```

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/contract (insert-into-max-tree root val)
(-> (or/c tree-node? #f) exact-integer? (or/c tree-node? #f))
)
```