

Problem 1038: Binary Search Tree to Greater Sum Tree

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the

root

of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a

binary search tree

is a tree that satisfies these constraints:

The left subtree of a node contains only nodes with keys

less than

the node's key.

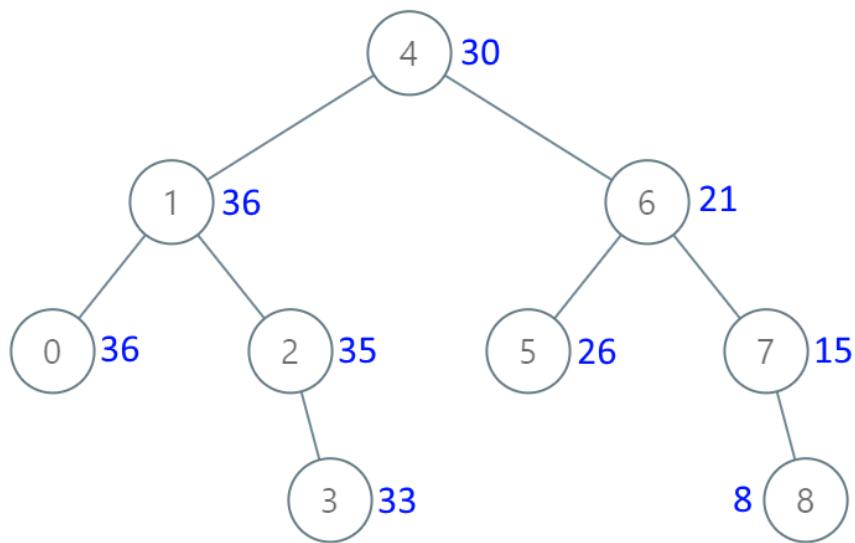
The right subtree of a node contains only nodes with keys

greater than

the node's key.

Both the left and right subtrees must also be binary search trees.

Example 1:



Input:

```
root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
```

Output:

```
[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

Example 2:

Input:

```
root = [0,null,1]
```

Output:

```
[1,null,1]
```

Constraints:

The number of nodes in the tree is in the range

[1, 100]

.

$0 \leq \text{Node.val} \leq 100$

All the values in the tree are

unique

.

Note:

This question is the same as 538:

<https://leetcode.com/problems/convert-bst-to-greater-tree/>

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* bstToGst(TreeNode* root) {
```

```
}
```

```
};
```

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 bstToGst(TreeNode root) {  
  
    }  
}
```

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 bstToGst(self, root: Optional[TreeNode]) -> 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 bstToGst(self, root):
        """
:type root: Optional[TreeNode]
: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
 * @return {TreeNode}
 */
var bstToGst = function(root) {

};

```

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 bstToGst(root: TreeNode | null): 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 BstToGst(TreeNode root) {
        }
    }
}

```

C:

```

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

```

Go:

```
/**  
 * Definition for a binary tree node.  
 * type TreeNode struct {  
 *     Val int  
 *     Left *TreeNode  
 *     Right *TreeNode  
 * }  
 */  
func bstToGst(root *TreeNode) *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 bstToGst(root: TreeNode?): 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 bstToGst(_ root: TreeNode?) -> 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 bst_to_gst(root: Option<Rc<RefCell<TreeNode>>>) -> 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  
# @return {TreeNode}  
def bst_to_gst(root)  
  
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 $root  
 * @return TreeNode  
 */  
function bstToGst($root) {  
  
}  
}
```

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? bstToGst(TreeNode? root) {  
  
}  
}
```

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 bstToGst(root: TreeNode): 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 bst_to_gst(root :: TreeNode.t | nil) :: TreeNode.t | nil
def bst_to_gst(root) 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 bst_to_gst(Root :: #tree_node{} | null) -> #tree_node{} | null.
bst_to_gst(Root) ->
.
.
```

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 (bst-to-gst root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f)))

```

```
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Binary Search Tree to Greater Sum Tree
 * Difficulty: Medium
 * Tags: tree, 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) {}
 *     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* bstToGst(TreeNode* root) {

    }
};
```

Java Solution:

```
/**
 * Problem: Binary Search Tree to Greater Sum Tree
 * Difficulty: Medium
```

```

* Tags: tree, 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 Solution {
public TreeNode bstToGst(TreeNode root) {

}
}

```

Python3 Solution:

```

"""
Problem: Binary Search Tree to Greater Sum Tree
Difficulty: Medium
Tags: tree, 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 Solution:
    def bstToGst(self, root: Optional[TreeNode]) -> 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 bstToGst(self, root):
        """
:type root: Optional[TreeNode]
:rtype: Optional[TreeNode]
"""

```

JavaScript Solution:

```

/**
 * Problem: Binary Search Tree to Greater Sum Tree
 * Difficulty: Medium
 * Tags: tree, 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
* @return {TreeNode}
*/
var bstToGst = function(root) {

};

```

TypeScript Solution:

```

/**
* Problem: Binary Search Tree to Greater Sum Tree
* Difficulty: Medium
* Tags: tree, 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)
*   }
* }
*/

```

```
function bstToGst(root: TreeNode | null): TreeNode | null {  
};
```

C# Solution:

```
/*  
 * Problem: Binary Search Tree to Greater Sum Tree  
 * Difficulty: Medium  
 * Tags: tree, 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 Solution {  
    public TreeNode BstToGst(TreeNode root) {  
        }  
    }
```

C Solution:

```
/*  
 * Problem: Binary Search Tree to Greater Sum Tree  
 * Difficulty: Medium
```

```

* Tags: tree, 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;
* };
*/
struct TreeNode* bstToGst(struct TreeNode* root) {

}

```

Go Solution:

```

// Problem: Binary Search Tree to Greater Sum Tree
// Difficulty: Medium
// Tags: tree, 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
* }
*/
func bstToGst(root *TreeNode) *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 bstToGst(root: TreeNode?): 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 bstToGst(_ root: TreeNode?) -> TreeNode? {
        ...
    }
}
```

Rust Solution:

```
// Problem: Binary Search Tree to Greater Sum Tree
// Difficulty: Medium
// Tags: tree, 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
//     }
//   }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
  pub fn bst_to_gst(root: Option<Rc<RefCell<TreeNode>>>) ->
    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
# @return {TreeNode}
def bst_to_gst(root)

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 $root
     * @return TreeNode
     */
    function bstToGst($root) {

    }
}

```

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? bstToGst(TreeNode? root) {

}
}

```

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 bstToGst(root: TreeNode): 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 bst_to_gst(root :: TreeNode.t | nil) :: TreeNode.t | nil
def bst_to_gst(root) 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 bst_to_gst(Root :: #tree_node{} | null) -> #tree_node{} | null.
bst_to_gst(Root) ->
.
.
```

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 (bst-to-gst root)
  (-> (or/c tree-node? #f) (or/c tree-node? #f)))
)
```

