

Problem 333: Largest BST Subtree

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the root of a binary tree, find the largest subtree

, which is also a Binary Search Tree (BST), where the largest means subtree has the largest number of nodes.

A

Binary Search Tree (BST)

is a tree in which all the nodes follow the below-mentioned properties:

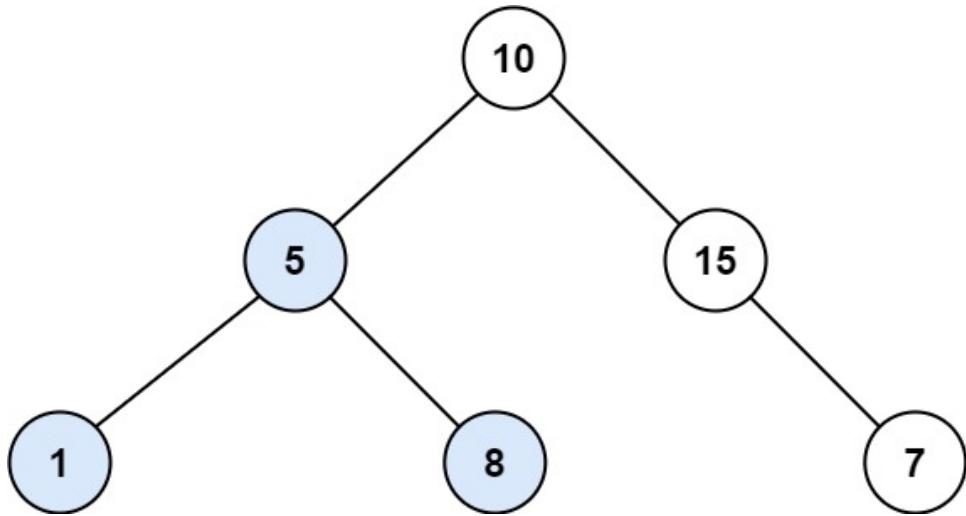
The left subtree values are less than the value of their parent (root) node's value.

The right subtree values are greater than the value of their parent (root) node's value.

Note:

A subtree must include all of its descendants.

Example 1:



Input:

```
root = [10,5,15,1,8,null,7]
```

Output:

3

Explanation:

The Largest BST Subtree in this case is the highlighted one. The return value is the subtree's size, which is 3.

Example 2:

Input:

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

Output:

2

Constraints:

The number of nodes in the tree is in the range

[0, 10

4

]

-10

4

<= Node.val <= 10

4

Follow up:

Can you figure out ways to solve it with

O(n)

time complexity?

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:
int largestBSTSubtree(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 int largestBSTSubtree(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 largestBSTSubtree(self, root: Optional[TreeNode]) -> int:

```

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 largestBSTSubtree(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: int
        """

```

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 {number}
 */
var largestBSTSubtree = 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 largestBSTSubtree(root: TreeNode | null): number {
}

```

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 int LargestBSTSubtree(TreeNode root) {
        }
    }
}

```

C:

```

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

```

```
int largestBSTSubtree(struct TreeNode* root) {  
    }  
}
```

Go:

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

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 largestBSTSubtree(root: TreeNode?): Int {  
  
    }  
}
```

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 largestBSTSubtree(_ root: TreeNode?) -> Int {
}
}

```

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 largest_bst_subtree(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

```

```
}
```

```
}
```

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 {Integer}
def largest_bst_subtree(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 Integer
 */
```

```
function largestBSTSubtree($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 {  
int largestBSTSubtree(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 largestBSTSubtree(root: TreeNode): Int = {  
  
}  
}
```

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 largest_bst_subtree(root :: TreeNode.t() | nil) :: integer
def largest_bst_subtree(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 largest_bst_subtree(Root :: #tree_node{} | null) -> integer().
largest_bst_subtree(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 (largest-bst-subtree root)
  (-> (or/c tree-node? #f) exact-integer?))
)

```

Solutions

C++ Solution:

```

/*
 * Problem: Largest BST Subtree
 * Difficulty: Medium
 * Tags: tree, dp, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * 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 Solution {
public:
int largestBSTSubtree(TreeNode* root) {
}
};


```

Java Solution:

```

/**
 * Problem: Largest BST Subtree
 * Difficulty: Medium
 * Tags: tree, dp, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * 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 int largestBSTSubtree(TreeNode root) {  
    }  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Largest BST Subtree  
Difficulty: Medium  
Tags: tree, dp, search  
  
Approach: DFS or BFS traversal  
Time Complexity: O(n) where n is number of nodes  
Space Complexity: O(n) or O(n * m) for DP table  
"""  
  
# 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 largestBSTSubtree(self, root: Optional[TreeNode]) -> int:  
        # 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 largestBSTSubtree(self, root):  
        """  
        :type root: Optional[TreeNode]  
        :rtype: int
```

```
"""
```

JavaScript Solution:

```
/**  
 * Problem: Largest BST Subtree  
 * Difficulty: Medium  
 * Tags: tree, dp, search  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */  
  
/**  
 * 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 {number}  
 */  
var largestBSTSubtree = function(root) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Largest BST Subtree  
 * Difficulty: Medium  
 * Tags: tree, dp, search  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */
```

```

/**
 * 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 largestBSTSubtree(root: TreeNode | null): number {
}

```

C# Solution:

```

/*
 * Problem: Largest BST Subtree
 * Difficulty: Medium
 * Tags: tree, dp, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * 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 int LargestBSTSubtree(TreeNode root) {
    }
}

```

C Solution:

```

/*
 * Problem: Largest BST Subtree
 * Difficulty: Medium
 * Tags: tree, dp, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

}

```

Go Solution:

```

// Problem: Largest BST Subtree
// Difficulty: Medium
// Tags: tree, dp, search
//
// Approach: DFS or BFS traversal

```

```

// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

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

}

```

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 largestBSTSubtree(root: TreeNode?): Int {
        }
    }
}

```

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 largestBSTSubtree(_ root: TreeNode?) -> Int {
}
}

```

Rust Solution:

```

// Problem: Largest BST Subtree
// Difficulty: Medium
// Tags: tree, dp, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

// 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 largest_bst_subtree(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {
    }

}
}

```

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 {Integer}
def largest_bst_subtree(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 $root
 * @return Integer
 */
function largestBSTSubtree($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 {
int largestBSTSubtree(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 largestBSTSubtree(root: TreeNode): Int = {

}
}

```

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 largest_bst_subtree(TreeNode.t() | nil) :: integer
def largest_bst_subtree(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 largest_bst_subtree(tree_node() | null) -> integer().
largest_bst_subtree(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 (largest-bst-subtree root)  
(-> (or/c tree-node? #f) exact-integer?)  
)
```