Problem 776: Split BST

Problem Information

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

Problem Description

Given the

root

of a binary search tree (BST) and an integer

target

, split the tree into two subtrees where the first subtree has nodes that are all smaller or equal to the target value, while the second subtree has all nodes that are greater than the target value. It is not necessarily the case that the tree contains a node with the value

target

.

Additionally, most of the structure of the original tree should remain. Formally, for any child

c

with parent

p

in the original tree, if they are both in the same subtree after the split, then node

c

should still have the parent

p

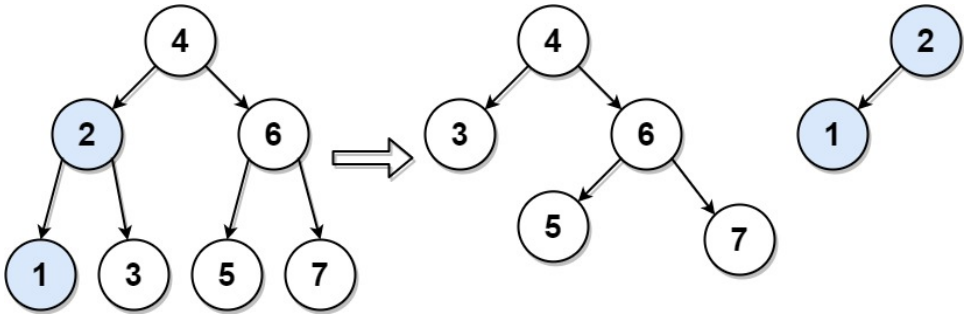
.

Return

an array of the two roots of the two subtrees in order

.

Example 1:



Input:

root = [4,2,6,1,3,5,7], target = 2

Output:

[[2,1],[4,3,6,null,null,5,7]]

Example 2:

Input:

root = [1], target = 1

Output:

[[1],[]]

Constraints:

The number of nodes in the tree is in the range

[1, 50]

.

0 <= Node.val, target <= 1000

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:
vector<TreeNode*> splitBST(TreeNode* root, int 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 Solution {
public TreeNode[] splitBST(TreeNode root, int 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 Solution:
def splitBST(self, root: Optional[TreeNode], target: int) ->
List[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 splitBST(self, root, target):
"""
:type root: Optional[TreeNode]
```

```
:type target: int
:rtype: List[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} target
* @return {TreeNode[]}
*/
var splitBST = function(root, 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)
* }
* }
*/

function splitBST(root: TreeNode | null, target: number): Array<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[] SplitBST(TreeNode root, int target) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct TreeNode** splitBST(struct TreeNode* root, int target, int*
returnSize) {

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func splitBST(root *TreeNode, target 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 splitBST(root: TreeNode?, target: Int): Array<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 splitBST(_ root: TreeNode?, _ target: 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 split_bst(root: Option<Rc<RefCell<TreeNode>>>, target: i32) ->
Vec<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} target
# @return {TreeNode[]}
def split_bst(root, target)

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 $target
* @return TreeNode[]
*/
function splitBST($root, $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 Solution {
List<TreeNode?> splitBST(TreeNode? root, int 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
* }
*/
object Solution {
def splitBST(root: TreeNode, target: Int): Array[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 split_bst(root :: TreeNode.t | nil, target :: integer) :: [TreeNode.t |
nil]
def split_bst(root, target) 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 split_bst(Root :: #tree_node{} | null, Target :: integer()) ->
[#tree_node{} | null].
split_bst(Root, Target) ->
.
```

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 (split-bst root target)
(-> (or/c tree-node? #f) exact-integer? (listof (or/c tree-node? #f)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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:
vector<TreeNode*> splitBST(TreeNode* root, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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() {}
* 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[] splitBST(TreeNode root, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Split BST
Difficulty: Medium
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
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 splitBST(self, root: Optional[TreeNode], target: int) ->
List[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 splitBST(self, root, target):
"""
:type root: Optional[TreeNode]
:type target: int
:rtype: List[Optional[TreeNode]]
"""
```

JavaScript Solution:

```
/**
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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} target
* @return {TreeNode[]}
*/
var splitBST = function(root, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 splitBST(root: TreeNode | null, target: number): Array<TreeNode |
null> {

};
```

C# Solution:

```
/*
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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[] SplitBST(TreeNode root, int target) {

}
}
```

C Solution:

```
/*
* Problem: Split BST
* Difficulty: Medium
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct TreeNode** splitBST(struct TreeNode* root, int target, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Split BST
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 splitBST(root *TreeNode, target 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 splitBST(root: TreeNode?, target: Int): Array<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 splitBST(_ root: TreeNode?, _ target: Int) -> [TreeNode?] {

}
}
```

Rust Solution:

```
// Problem: Split BST
// Difficulty: Medium
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 split_bst(root: Option<Rc<RefCell<TreeNode>>>, target: i32) ->
Vec<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} target
# @return {TreeNode[]}
def split_bst(root, target)

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 $target
* @return TreeNode[]
*/
function splitBST($root, $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 Solution {
List<TreeNode?> splitBST(TreeNode? root, int 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
* }
*/
object Solution {
def splitBST(root: TreeNode, target: Int): Array[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 split_bst(root :: TreeNode.t | nil, target :: integer) :: [TreeNode.t |
nil]
def split_bst(root, target) 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 split_bst(Root :: #tree_node{} | null, Target :: integer()) ->
[#tree_node{} | null].
split_bst(Root, Target) ->
.
```

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 (split-bst root target)
(-> (or/c tree-node? #f) exact-integer? (listof (or/c tree-node? #f)))
)
```