Problem 285: Inorder Successor in BST

Problem Information

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

Problem Description

Given the

root

of a binary search tree and a node

p

in it, return

the in-order successor of that node in the BST

. If the given node has no in-order successor in the tree, return

null

.

The successor of a node

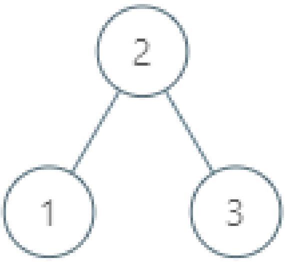
p

is the node with the smallest key greater than

p.val

.

Example 1:



Input:

root = [2,1,3], p = 1

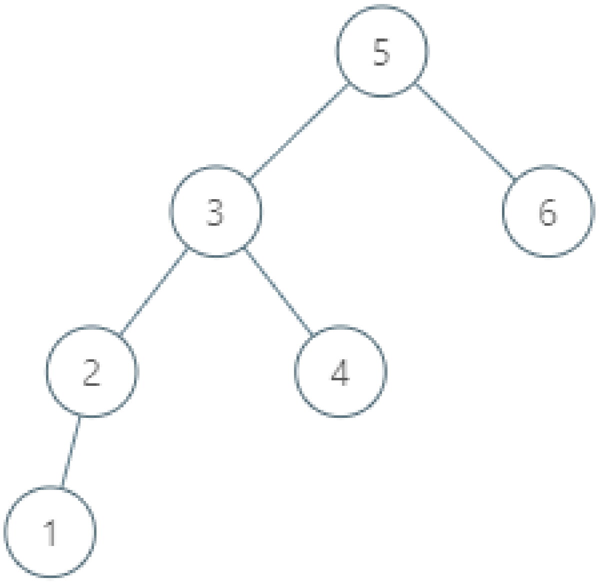
Output:

2

Explanation:

1's in-order successor node is 2. Note that both p and the return value is of TreeNode type.

Example 2:



Input:

root = [5,3,6,2,4,null,null,1], p = 6

Output:

null

Explanation:

There is no in-order successor of the current node, so the answer is

null

.

Constraints:

The number of nodes in the tree is in the range

[1, 10

4

]

.

-10

5

<= Node.val <= 10

5

All Nodes will have unique values.

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode(int x) { val = x; }
* }
*/
class Solution {
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
```

```
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def inorderSuccessor(self, root: TreeNode, p: TreeNode) ->
Optional[TreeNode]:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def inorderSuccessor(self, root, p):
"""
:type root: TreeNode
:type p: TreeNode
:rtype: TreeNode
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {TreeNode} p
* @return {TreeNode}
*/
var inorderSuccessor = function(root, p) {

};
```

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 inorderSuccessor(root: TreeNode | null, p: 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 x) { val = x; }
* }
*/
public class Solution {
public TreeNode InorderSuccessor(TreeNode root, TreeNode p) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
/**
* Definition for a binary tree node.
* class TreeNode(var `val`: Int = 0) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/

class Solution {
fun inorderSuccessor(root: TreeNode?, p: 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(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* }
* }
*/

class Solution {
func inorderSuccessor(_ root: TreeNode?, _ p: 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 inorder_successor(root: Option<Rc<RefCell<TreeNode>>>, p:
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)
# @val = val
# @left, @right = nil, nil
# end
# end

# @param {TreeNode} root
# @param {TreeNode} p
# @return {TreeNode}
def inorder_successor(root, p)

end
```

PHP:

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

class Solution {
/**
* @param TreeNode $root
* @param TreeNode $p
* @return TreeNode
*/
function inorderSuccessor($root, $p) {

}
}
```

Scala:

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

object Solution {
def inorderSuccessor(root: TreeNode, p: TreeNode): TreeNode = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Inorder Successor in BST
* 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(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {

}
};
```

Java Solution:

```
/**
* Problem: Inorder Successor in BST
* 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(int x) { val = x; }
* }
*/
class Solution {
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {

}
}
```

Python3 Solution:

```
"""
Problem: Inorder Successor in BST
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, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def inorderSuccessor(self, root: TreeNode, p: TreeNode) ->
Optional[TreeNode]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def inorderSuccessor(self, root, p):
"""
:type root: TreeNode
:type p: TreeNode
:rtype: TreeNode
"""
```

JavaScript Solution:

```
/**
* Problem: Inorder Successor in BST
* 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) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {TreeNode} p
* @return {TreeNode}
*/
var inorderSuccessor = function(root, p) {

};
```

TypeScript Solution:

```
/**
* Problem: Inorder Successor in BST
* 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 inorderSuccessor(root: TreeNode | null, p: TreeNode | null):
TreeNode | null {

};
```

C# Solution:

```
/*
* Problem: Inorder Successor in BST
* 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 x) { val = x; }
* }
*/
public class Solution {
public TreeNode InorderSuccessor(TreeNode root, TreeNode p) {
}
}
```

C Solution:

```
/*
* Problem: Inorder Successor in BST
* 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* inorderSuccessor(struct TreeNode* root, struct TreeNode* p)
{

}
```

Go Solution:

```
// Problem: Inorder Successor in BST
// 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 inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {

}
```

Kotlin Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(var `val`: Int = 0) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/

class Solution {
fun inorderSuccessor(root: TreeNode?, p: 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(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* }
* }
*/
```

```
class Solution {
func inorderSuccessor(_ root: TreeNode?, _ p: TreeNode?) -> TreeNode? {

}
}
```

Rust Solution:

```
// Problem: Inorder Successor in BST
// 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 inorder_successor(root: Option<Rc<RefCell<TreeNode>>>, p:
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)
# @val = val
# @left, @right = nil, nil
# end
# end

# @param {TreeNode} root
# @param {TreeNode} p
# @return {TreeNode}
def inorder_successor(root, p)

end
```

PHP Solution:

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

class Solution {
/**
* @param TreeNode $root
* @param TreeNode $p
* @return TreeNode
*/
function inorderSuccessor($root, $p) {

}

}
```

Scala Solution:

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

object Solution {
def inorderSuccessor(root: TreeNode, p: TreeNode): TreeNode = {

}
}
```