Problem 510: Inorder Successor in BST II

Problem Information

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

Problem Description

Given a

node

in a binary search tree, return

the in-order successor of that node in the BST

. If that node has no in-order successor, return

null

.

The successor of a

node

is the node with the smallest key greater than

node.val

.

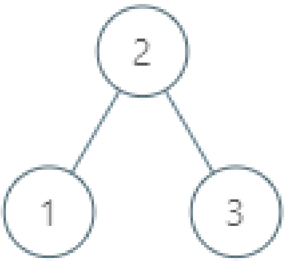
You will have direct access to the node but not to the root of the tree. Each node will have a reference to its parent node. Below is the definition for

Node

:

class Node { public int val; public Node left; public Node right; public Node parent; }

Example 1:



Input:

tree = [2,1,3], node = 1

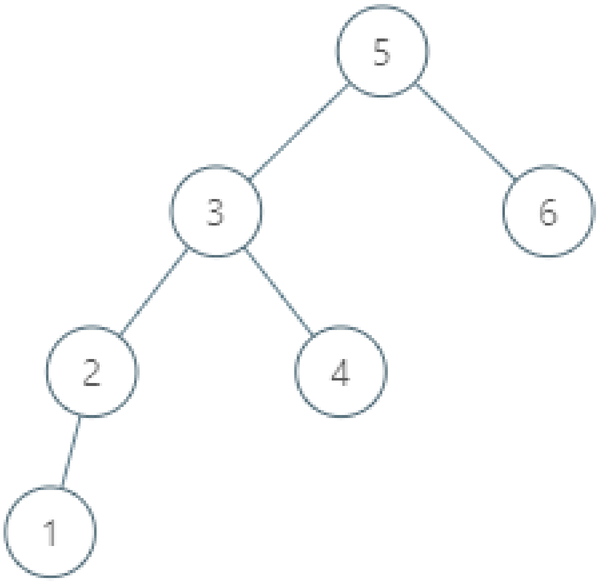
Output:

2

Explanation:

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

Example 2:



Input:

tree = [5,3,6,2,4,null,null,1], node = 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.

Follow up:

Could you solve it without looking up any of the node's values?

Code Snippets

C++:

```
/*
// Definition for a Node.
class Node {
public:
int val;
Node* left;
Node* right;
Node* parent;
};
*/

class Solution {
public:
Node* inorderSuccessor(Node* node) {

}
};
```

Java:

```
/*
// Definition for a Node.
class Node {
public int val;
public Node left;
public Node right;
public Node parent;
};
*/

class Solution {
public Node inorderSuccessor(Node node) {

}
}
```

Python3:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val):
self.val = val
self.left = None
self.right = None
self.parent = None
"""

class Solution:
def inorderSuccessor(self, node: 'Node') -> 'Optional[Node]':
```

Python:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val):
self.val = val
self.left = None
self.right = None
self.parent = None
"""

class Solution(object):
def inorderSuccessor(self, node):
"""
```

```
:type node: Node
:rtype: Node
"""
```

JavaScript:

```
/**
* // Definition for a _Node.
* function _Node(val) {
* this.val = val;
* this.left = null;
* this.right = null;
* this.parent = null;
* };
*/

/**
* @param {_Node} node
* @return {_Node}
*/
var inorderSuccessor = function(node) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: number
* left: _Node | null
* right: _Node | null
* parent: _Node | null
*
* constructor(v: number) {
* this.val = v;
* this.left = null;
* this.right = null;
* this.parent = null;
* }
* }
*/
```

```
}
```

```
function inorderSuccessor(node: _Node | null): _Node | null {

};
```

C#:

```
/*
// Definition for a Node.
public class Node {
public int val;
public Node left;
public Node right;
public Node parent;
}
*/

public class Solution {
public Node InorderSuccessor(Node x) {

}
}
```

C:

```
/*
// Definition for a Node.
struct Node {
int val;
struct Node* left;
struct Node* right;
struct Node* parent;
};
*/

struct Node* inorderSuccessor(struct Node* node) {

}
```

Go:

```
/**
* Definition for Node.
* type Node struct {
* Val int
* Left *Node
* Right *Node
* Parent *Node
* }
*/

func inorderSuccessor(node *Node) *Node {

}
```

Kotlin:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var left: Node? = null
* var right: Node? = null
* var parent: Node? = null
* }
*/

class Solution {
fun inorderSuccessor(node: Node?): Node? {

}
}
```

Swift:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var left: Node?
* public var right: Node?
* public var parent: Node?
* public init(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* self.parent = nil
* }
* }
*/

class Solution {
func inorderSuccessor(_ node: Node?) -> Node? {

}
}
```

Ruby:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :left, :right, :parent
# def initialize(val=0)
# @val = val
# @left, @right, parent = nil, nil, nil
# end
# end

# @param {Node} root
# @return {Node}
def inorderSuccessor(node)

end
```

PHP:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $left = null;
* public $right = null;
* public $parent = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->left = null;
* $this->right = null;
* $this->parent = null;
* }
* }
*/

class Solution {
/**
* @param Node $node
* @return Node
*/
function inorderSuccessor($node) {

}
}
```

Scala:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var left: Node = null
* var right: Node = null
* var parent: Node = null
* }
*/

object Solution {
def inorderSuccessor(node: Node): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Inorder Successor in BST II
* 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 Node.
class Node {
public:
int val;
Node* left;
Node* right;
Node* parent;
};
*/

class Solution {
public:
Node* inorderSuccessor(Node* node) {

}
};
```

Java Solution:

```
/**
* Problem: Inorder Successor in BST II
* 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 Node.
class Node {
public int val;
public Node left;
public Node right;
public Node parent;
};
*/

class Solution {
public Node inorderSuccessor(Node node) {

}
}
```

Python3 Solution:

```
"""
Problem: Inorder Successor in BST II
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 Node.
class Node:
def __init__(self, val):
self.val = val
self.left = None
self.right = None
self.parent = None
"""

class Solution:
def inorderSuccessor(self, node: 'Node') -> 'Optional[Node]':
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val):
self.val = val
self.left = None
self.right = None
self.parent = None
"""

class Solution(object):
def inorderSuccessor(self, node):
"""
:type node: Node
:rtype: Node
"""
```

JavaScript Solution:

```
/**
* Problem: Inorder Successor in BST II
* 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 _Node.
* function _Node(val) {
* this.val = val;
* this.left = null;
* this.right = null;
* this.parent = null;
* };
*/

/**
* @param {_Node} node
* @return {_Node}
```

```
*/
var inorderSuccessor = function(node) {

};
```

TypeScript Solution:

```
/**
* Problem: Inorder Successor in BST II
* 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 _Node.
* class _Node {
* val: number
* left: _Node | null
* right: _Node | null
* parent: _Node | null
*
* constructor(v: number) {
* this.val = v;
* this.left = null;
* this.right = null;
* this.parent = null;
* }
* }
*/

function inorderSuccessor(node: _Node | null): _Node | null {

};
```

C# Solution:

```
/*
* Problem: Inorder Successor in BST II
* 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 Node.
public class Node {
public int val;
public Node left;
public Node right;
public Node parent;
}
*/

public class Solution {
public Node InorderSuccessor(Node x) {

}
}
```

C Solution:

```
/*
* Problem: Inorder Successor in BST II
* 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 Node.
struct Node {
int val;
struct Node* left;
struct Node* right;
struct Node* parent;
};
*/

struct Node* inorderSuccessor(struct Node* node) {

}
```

Go Solution:

```
// Problem: Inorder Successor in BST II
// 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 Node.
* type Node struct {
* Val int
* Left *Node
* Right *Node
* Parent *Node
* }
*/

func inorderSuccessor(node *Node) *Node {

}
```

Kotlin Solution:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var left: Node? = null
* var right: Node? = null
* var parent: Node? = null
* }
*/

class Solution {
fun inorderSuccessor(node: Node?): Node? {

}
}
```

Swift Solution:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var left: Node?
* public var right: Node?
* public var parent: Node?
* public init(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* self.parent = nil
* }
* }
*/

class Solution {
func inorderSuccessor(_ node: Node?) -> Node? {

}
}
```

Ruby Solution:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :left, :right, :parent
# def initialize(val=0)
# @val = val
# @left, @right, parent = nil, nil, nil
# end
# end

# @param {Node} root
# @return {Node}
def inorderSuccessor(node)

end
```

PHP Solution:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $left = null;
* public $right = null;
* public $parent = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->left = null;
* $this->right = null;
* $this->parent = null;
* }
* }
*/

class Solution {
/**
* @param Node $node
* @return Node
*/
function inorderSuccessor($node) {

}
}
```

Scala Solution:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var left: Node = null
* var right: Node = null
* var parent: Node = null
* }
*/

object Solution {
def inorderSuccessor(node: Node): Node = {

}
}
```