Problem 117: Populating Next Right Pointers in Each Node II

Problem Information

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

Problem Description

Given a binary tree

struct Node { int val; Node *left; Node *right; Node *next; }

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to

NULL

.

Initially, all next pointers are set to

NULL

.

Example 1:

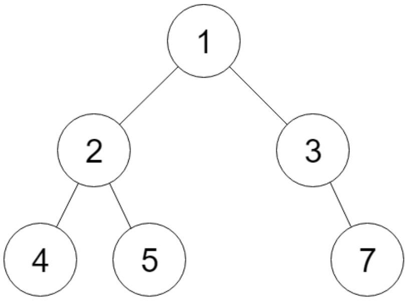

Figure A

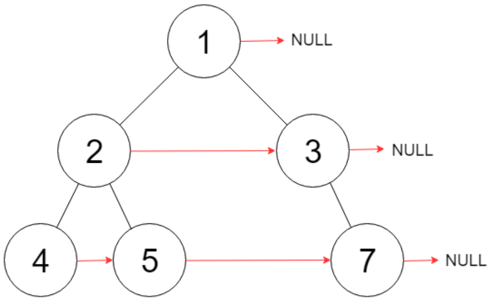

Figure B

Input:

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

Output:

[1,#,2,3,#,4,5,7,#]

Explanation:

Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.

Example 2:

Input:

root = []

Output:

[]

Constraints:

The number of nodes in the tree is in the range

[0, 6000]

.

-100 <= Node.val <= 100

Follow-up:

You may only use constant extra space.

The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

Code Snippets

C++:

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

Node() : val(0), left(NULL), right(NULL), next(NULL) {}

Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}

Node(int _val, Node* _left, Node* _right, Node* _next)
: val(_val), left(_left), right(_right), next(_next) {}
};
*/

class Solution {
public:
Node* connect(Node* root) {

}
};
```

Java:

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

public Node() {}

public Node(int _val) {
val = _val;
}

public Node(int _val, Node _left, Node _right, Node _next) {
val = _val;
left = _left;
right = _right;
next = _next;
}
};
*/

class Solution {
public Node connect(Node root) {

}
}
```

Python3:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None,
next: 'Node' = None):
self.val = val
self.left = left
self.right = right
self.next = next
"""

class Solution:
```

```
def connect(self, root: 'Node') -> 'Node':
```

Python:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=0, left=None, right=None, next=None):
self.val = val
self.left = left
self.right = right
self.next = next
"""

class Solution(object):
def connect(self, root):
"""
:type root: Node
:rtype: Node
"""
```

JavaScript:

```
/**
* // Definition for a _Node.
* function _Node(val, left, right, next) {
* this.val = val === undefined ? null : val;
* this.left = left === undefined ? null : left;
* this.right = right === undefined ? null : right;
* this.next = next === undefined ? null : next;
* };
*/

/**
* @param {_Node} root
* @return {_Node}
*/
var connect = function(root) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: number
* left: _Node | null
* right: _Node | null
* next: _Node | null
*
* constructor(val?: number, left?: _Node, right?: _Node, next?: _Node) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* this.next = (next===undefined ? null : next)
* }
* }
*/



function connect(root: _Node | null): _Node | null {

};
```

C#:

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

public Node() {}

public Node(int _val) {
val = _val;
}

public Node(int _val, Node _left, Node _right, Node _next) {
val = _val;
left = _left;
right = _right;
next = _next;
}
}
*/

public class Solution {
public Node Connect(Node root) {

}
}
```

C:

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

struct Node* connect(struct Node* root) {

}
```

Go:

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

func connect(root *Node) *Node {

}
```

Kotlin:

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

class Solution {
fun connect(root: Node?): Node? {

}
}
```

Swift:

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

class Solution {
func connect(_ root: Node?) -> Node? {

}
}
```

Ruby:

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

# @param {Node} root
# @return {Node}
def connect(root)

end
```

PHP:

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

class Solution {
/**
* @param Node $root
* @return Node
*/
public function connect($root) {

}
}
```

Scala:

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

object Solution {
def connect(root: Node): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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* next;

Node() : val(0), left(NULL), right(NULL), next(NULL) {}

Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}
Node(int _val, Node* _left, Node* _right, Node* _next)
: val(_val), left(_left), right(_right), next(_next) {}
};
*/

class Solution {
public:
Node* connect(Node* root) {

}
};
```

Java Solution:

```
/**
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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 next;

public Node() {
// TODO: Implement optimized solution
return 0;
}

public Node(int _val) {
val = _val;
}
public Node(int _val, Node _left, Node _right, Node _next) {
val = _val;
left = _left;
right = _right;
next = _next;
}
};
*/

class Solution {
public Node connect(Node root) {

}
}
```

Python3 Solution:

```
"""
Problem: Populating Next Right Pointers in Each Node II
Difficulty: Medium
Tags: tree, search, linked_list, stack

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: int = 0, left: 'Node' = None, right: 'Node' = None,
next: 'Node' = None):
self.val = val
self.left = left
self.right = right
self.next = next
"""

class Solution:
def connect(self, root: 'Node') -> 'Node':
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=0, left=None, right=None, next=None):
self.val = val
self.left = left
self.right = right
self.next = next
"""

class Solution(object):
def connect(self, root):
"""
:type root: Node
:rtype: Node
"""
```

JavaScript Solution:

```
/**
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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, left, right, next) {
* this.val = val === undefined ? null : val;
* this.left = left === undefined ? null : left;
* this.right = right === undefined ? null : right;
* this.next = next === undefined ? null : next;
* };
*/

/**
* @param {_Node} root
* @return {_Node}
*/
var connect = function(root) {

};
```

TypeScript Solution:

```
/**
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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
* next: _Node | null
*
* constructor(val?: number, left?: _Node, right?: _Node, next?: _Node) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* this.next = (next===undefined ? null : next)
* }
* }
*/
```

```
function connect(root: _Node | null): _Node | null {

};
```

C# Solution:

```
/*
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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 next;

public Node() {}

public Node(int _val) {
val = _val;
}

public Node(int _val, Node _left, Node _right, Node _next) {
val = _val;
left = _left;
right = _right;
next = _next;
}
}
*/

public class Solution {
public Node Connect(Node root) {
}
}
```

C Solution:

```
/*
* Problem: Populating Next Right Pointers in Each Node II
* Difficulty: Medium
* Tags: tree, search, linked_list, stack
*
* 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 *next;
* };
*/

struct Node* connect(struct Node* root) {

}
```

Go Solution:

```
// Problem: Populating Next Right Pointers in Each Node II
// Difficulty: Medium
// Tags: tree, search, linked_list, stack
//
// 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.
* type Node struct {
* Val int
* Left *Node
* Right *Node
* Next *Node
* }
*/

func connect(root *Node) *Node {

}
```

Kotlin Solution:

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

class Solution {
fun connect(root: 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 next: Node?
* public init(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* self.next = nil
* }
* }
*/

class Solution {
func connect(_ root: Node?) -> Node? {

}
}
```

Ruby Solution:

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

# @param {Node} root
# @return {Node}
def connect(root)

end
```

PHP Solution:

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

class Solution {
/**
* @param Node $root
* @return Node
*/
public function connect($root) {

}
}
```

Scala Solution:

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

object Solution {
def connect(root: Node): Node = {

}
}
```