Problem 1485: Clone Binary Tree With Random Pointer

Problem Information

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

Problem Description

A binary tree is given such that each node contains an additional random pointer which could point to any node in the tree or null.

Return a

deep copy

of the tree.

The tree is represented in the same input/output way as normal binary trees where each node is represented as a pair of

[val, random_index]

where:

val

: an integer representing

Node.val

random_index

: the index of the node (in the input) where the random pointer points to, or

null

if it does not point to any node.

You will be given the tree in class

Node

and you should return the cloned tree in class

NodeCopy

.

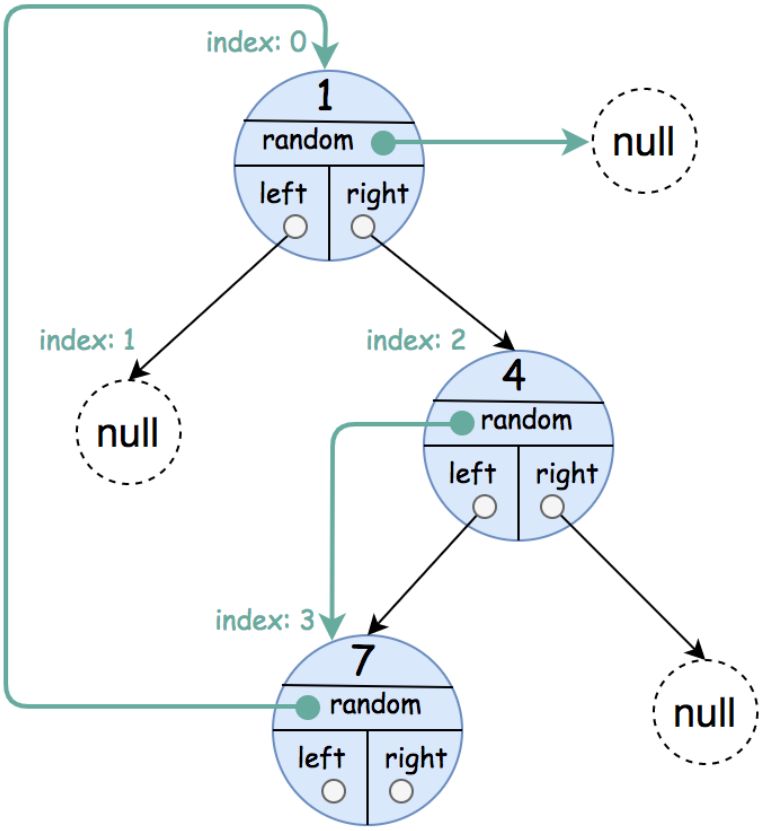
NodeCopy

class is just a clone of

Node

class with the same attributes and constructors.

Example 1:



Input:

root = [[1,null],null,[4,3],[7,0]]

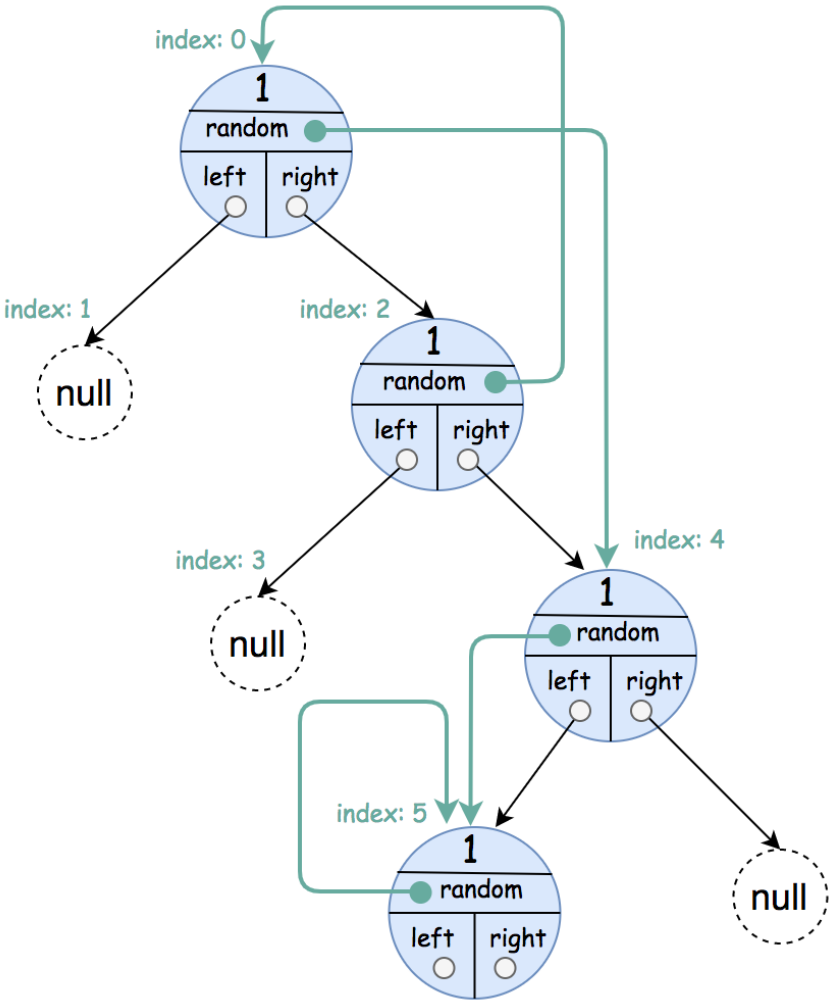
Output:

[[1,null],null,[4,3],[7,0]]

Explanation:

The original binary tree is [1,null,4,7]. The random pointer of node one is null, so it is represented as [1, null]. The random pointer of node 4 is node 7, so it is represented as [4, 3] where 3 is the index of node 7 in the array representing the tree. The random pointer of node 7 is node 1, so it is represented as [7, 0] where 0 is the index of node 1 in the array representing the tree.

Example 2:



Input:

root = [[1,4],null,[1,0],null,[1,5],[1,5]]

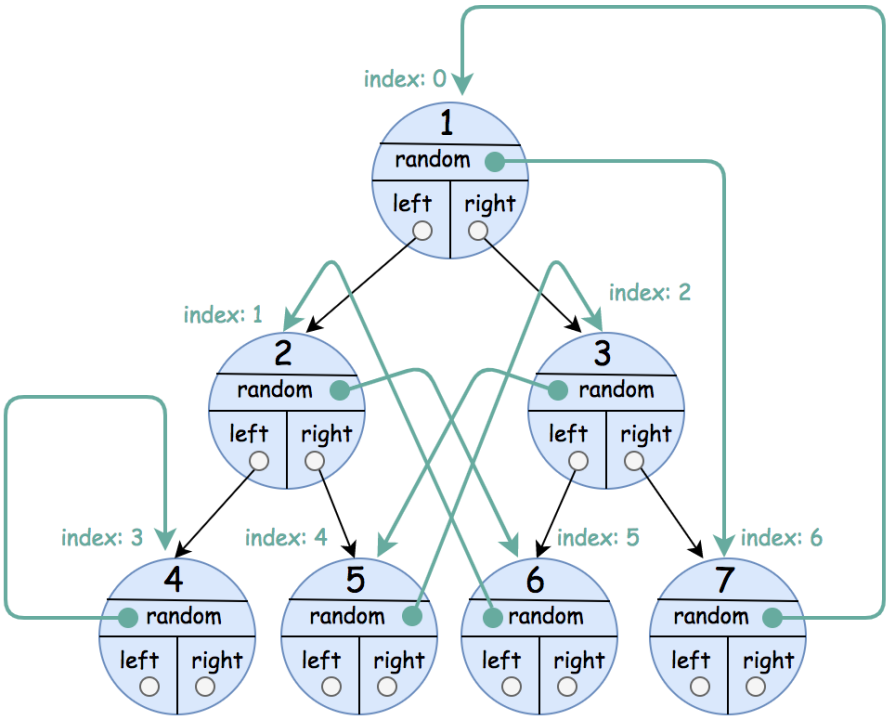
Output:

[[1,4],null,[1,0],null,[1,5],[1,5]]

Explanation:

The random pointer of a node can be the node itself.

Example 3:



Input:

root = [[1,6],[2,5],[3,4],[4,3],[5,2],[6,1],[7,0]]

Output:

[[1,6],[2,5],[3,4],[4,3],[5,2],[6,1],[7,0]]

Constraints:

The number of nodes in the

tree

is in the range

[0, 1000].

1 <= Node.val <= 10

6

Code Snippets

C++:

```
/**
* Definition for a Node.
* struct Node {
* int val;
* Node *left;
* Node *right;
* Node *random;
* Node() : val(0), left(nullptr), right(nullptr), random(nullptr) {}
* Node(int x) : val(x), left(nullptr), right(nullptr), random(nullptr) {}
* Node(int x, Node *left, Node *right, Node *random) : val(x), left(left),
right(right), random(random) {}
* };
*/

class Solution {
public:
NodeCopy* copyRandomBinaryTree(Node* root) {

}
};
```

Java:

```
/**
* Definition for Node.
* public class Node {
* int val;
* Node left;
* Node right;
* Node random;
* Node() {}
* Node(int val) { this.val = val; }
* Node(int val, Node left, Node right, Node random) {
* this.val = val;
* this.left = left;
* this.right = right;
* this.random = random;
```

```
* }
* }
*/

class Solution {
public NodeCopy copyRandomBinaryTree(Node root) {

}
}
```

Python3:

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

class Solution:
def copyRandomBinaryTree(self, root: 'Optional[Node]') ->
'Optional[NodeCopy]':
```

Python:

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

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

JavaScript:

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

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

};
```

TypeScript:

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


function copyRandomBinaryTree(root: _Node | null): NodeCopy | null {

};
```

C#:

```
/**
* Definition for Node.
* public class Node {
* public int val;
* public Node left;
* public Node right;
* public Node random;
* public Node(int val=0, Node left=null, Node right=null, Node random=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* this.random = random;
* }
* }
*/

public class Solution {
public NodeCopy CopyRandomBinaryTree(Node root) {

}
}
```

Go:

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

func copyRandomBinaryTree(root *Node) *NodeCopy {

}
```

Kotlin:

```
/**
* Example:
* var ti = Node(5)
* var v = ti.`val`
* Definition for a Node.
* class Node(var `val`: Int) {
* var left: Node? = null
* var right: Node? = null
* var random: Node? = null
* }
*/

class Solution {
fun copyRandomBinaryTree(root: Node?): NodeCopy? {

}
}
```

Swift:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var left: Node?
* public var right: Node?
* public var random: Node?
* public init() { self.val = 0; self.left = nil; self.right = nil;
self.random = nil; }
* public init(_ val: Int) {self.val = val; self.left = nil; self.right = nil;
self.random = nil; }
* public init(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* self.random = nil
* }
* }
*/

class Solution {
func copyRandomBinaryTree(_ root: Node?) -> NodeCopy? {
```

```
}
}
```

Ruby:

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

# @param {Node} root
# @return {NodeCopy}
def copy_random_binary_tree(root)

end
```

PHP:

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

class Solution {
```

```
/**
* @param Node $root
* @return NodeCopy
*/
public function copyRandomBinaryTree($root) {

}
}
```

Scala:

```
/**
* Definition for a Node.
* class Node(var _value: Int, _left: Node = null, _right: Node = null,
_random: Node = null) {
* var value: Int = _value
* var left: Node = _left
* var right: Node = _right
* var random: Node = _random
* }
*/

object Solution {
def copyRandomBinaryTree(root: Node): NodeCopy = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Clone Binary Tree With Random Pointer
* Difficulty: Medium
* Tags: array, tree, hash, 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 Node.
* struct Node {
* int val;
* Node *left;
* Node *right;
* Node *random;
* Node() : val(0), left(nullptr), right(nullptr), random(nullptr) {}
* Node(int x) : val(x), left(nullptr), right(nullptr), random(nullptr) {}
* Node(int x, Node *left, Node *right, Node *random) : val(x), left(left),
right(right), random(random) {}
* };
*/

class Solution {
public:
NodeCopy* copyRandomBinaryTree(Node* root) {

}
};
```

Java Solution:

```
/**
* Problem: Clone Binary Tree With Random Pointer
* Difficulty: Medium
* Tags: array, tree, hash, 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 Node.
* public class Node {
* int val;
* Node left;
* Node right;
```

```
* Node random;
* Node() {
// TODO: Implement optimized solution
return 0;
}
* Node(int val) { this.val = val; }
* Node(int val, Node left, Node right, Node random) {
* this.val = val;
* this.left = left;
* this.right = right;
* this.random = random;
* }
* }
*/

class Solution {
public NodeCopy copyRandomBinaryTree(Node root) {

}
}
```

Python3 Solution:

```
"""
Problem: Clone Binary Tree With Random Pointer
Difficulty: Medium
Tags: array, tree, hash, 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 Node.
# class Node:
# def __init__(self, val=0, left=None, right=None, random=None):
# self.val = val
# self.left = left
# self.right = right
# self.random = random
```

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

Python Solution:

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

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

JavaScript Solution:

```
/**
* Problem: Clone Binary Tree With Random Pointer
* Difficulty: Medium
* Tags: array, tree, hash, 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 _Node.
* function _Node(val, left, right, random) {
* this.val = val === undefined ? null : val;
* this.left = left === undefined ? null : left;
* this.right = right === undefined ? null : right;
```

```
* this.random = random === undefined ? null : random;
* };
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Clone Binary Tree With Random Pointer
* Difficulty: Medium
* Tags: array, tree, hash, 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 _Node.
* class _Node {
* val: number
* left: _Node | null
* right: _Node | null
* random: _Node | null
*
* constructor(val?: number, left?: _Node | null, right?: _Node | null,
random?: _Node | null) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* this.random = (random===undefined ? null : random)
* }
* }
*/
```

```
function copyRandomBinaryTree(root: _Node | null): NodeCopy | null {

};
```

C# Solution:

```
/*
* Problem: Clone Binary Tree With Random Pointer
* Difficulty: Medium
* Tags: array, tree, hash, 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 Node.
* public class Node {
* public int val;
* public Node left;
* public Node right;
* public Node random;
* public Node(int val=0, Node left=null, Node right=null, Node random=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* this.random = random;
* }
* }
*/

public class Solution {
public NodeCopy CopyRandomBinaryTree(Node root) {

}
}
```

Go Solution:

```
// Problem: Clone Binary Tree With Random Pointer
// Difficulty: Medium
// Tags: array, tree, hash, 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 Node.
* type Node struct {
* Val int
* Left *Node
* Right *Node
* Random *Node
* }
*/

func copyRandomBinaryTree(root *Node) *NodeCopy {

}
```

Kotlin Solution:

```
/**
* Example:
* var ti = Node(5)
* var v = ti.`val`
* Definition for a Node.
* class Node(var `val`: Int) {
* var left: Node? = null
* var right: Node? = null
* var random: Node? = null
* }
*/

class Solution {
fun copyRandomBinaryTree(root: Node?): NodeCopy? {

}
}
```

Swift Solution:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var left: Node?
* public var right: Node?
* public var random: Node?
* public init() { self.val = 0; self.left = nil; self.right = nil;
self.random = nil; }
* public init(_ val: Int) {self.val = val; self.left = nil; self.right = nil;
self.random = nil; }
* public init(_ val: Int) {
* self.val = val
* self.left = nil
* self.right = nil
* self.random = nil
* }
* }
*/

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

}
}
```

Ruby Solution:

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

# @param {Node} root
# @return {NodeCopy}
```

```
def copy_random_binary_tree(root)

end
```

PHP Solution:

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

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

}
}
```

Scala Solution:

```
/**
* Definition for a Node.
* class Node(var _value: Int, _left: Node = null, _right: Node = null,
_random: Node = null) {
* var value: Int = _value
* var left: Node = _left
```

```
* var right: Node = _right
* var random: Node = _random
* }
*/

object Solution {
def copyRandomBinaryTree(root: Node): NodeCopy = {

}
}
```