# Problem 138: Copy List with Random Pointer

## Problem Information

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

## Problem Description

A linked list of length

n

is given such that each node contains an additional random pointer, which could point to any node in the list, or

null

.

Construct a

deep copy

of the list. The deep copy should consist of exactly

n

brand new

nodes, where each new node has its value set to the value of its corresponding original node. Both the

next

and

random

pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state.

None of the pointers in the new list should point to nodes in the original list

.

For example, if there are two nodes

X

and

Y

in the original list, where

X.random --> Y

, then for the corresponding two nodes

x

and

y

in the copied list,

x.random --> y

.

Return

the head of the copied linked list

.

The linked list is represented in the input/output as a list of

n

nodes. 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 (range from

0

to

n-1

) that the

random

pointer points to, or

null

if it does not point to any node.

Your code will

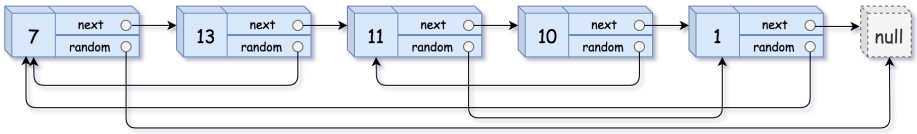
only

be given the

head

of the original linked list.

Example 1:



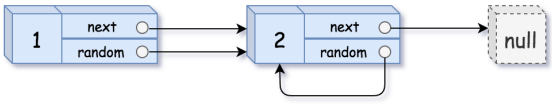

Input:

head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

Output:

[[7,null],[13,0],[11,4],[10,2],[1,0]]

Example 2:



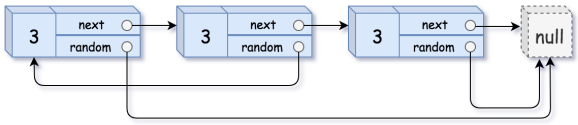

Input:

head = [[1,1],[2,1]]

Output:

[[1,1],[2,1]]

Example 3:




Input:

head = [[3,null],[3,0],[3,null]]

Output:

[[3,null],[3,0],[3,null]]

Constraints:

0 <= n <= 1000

-10

4

<= Node.val <= 10

4

Node.random

is

null

or is pointing to some node in the linked list.

## Code Snippets

**C++:**

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

Node(int _val) {
val = _val;
next = NULL;
random = NULL;
}
};
*/

class Solution {
public:
Node* copyRandomList(Node* head) {

}
};
```

**Java:**

```
/*
// Definition for a Node.
class Node {
int val;
Node next;
Node random;

public Node(int val) {
this.val = val;
this.next = null;
this.random = null;
}
}
*/

class Solution {
public Node copyRandomList(Node head) {
```

```
}
}
```

**Python3:**

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

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

**Python:**

```
"""
# Definition for a Node.
class Node:
def __init__(self, x, next=None, random=None):
self.val = int(x)
self.next = next
self.random = random
"""

class Solution(object):
def copyRandomList(self, head):
"""
:type head: Node
:rtype: Node
"""
```

**JavaScript:**

```
/**
* // Definition for a _Node.
* function _Node(val, next, random) {
* this.val = val;
* this.next = next;
```

```
* this.random = random;
* };
*/

/**
* @param {_Node} head
* @return {_Node}
*/
var copyRandomList = function(head) {

};
```

**TypeScript:**

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


function copyRandomList(head: _Node | null): _Node | null {

};
```

**C#:**

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

```
public Node(int _val) {
val = _val;
next = null;
random = null;
}
}
*/

public class Solution {
public Node CopyRandomList(Node head) {

}
}
```

**C:**

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

struct Node* copyRandomList(struct Node* head) {

}
```

**Go:**

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

func copyRandomList(head *Node) *Node {
```

```
}
```

**Kotlin:**

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

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

}
}
```

**Swift:**

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

class Solution {
func copyRandomList(_ head: Node?) -> Node? {

}
```

```
}
```

**Ruby:**

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

# @param {Node} node
# @return {Node}
def copyRandomList(head)

end
```

**PHP:**

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

class Solution {
/**
* @param Node $head
* @return Node
*/
function copyRandomList($head) {
```

```
}
}
```

**Scala:**

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

object Solution {
def copyRandomList(head: Node): Node = {

}
}
```

## Solutions

### C++ Solution:

```
/*
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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

```
Node* next;
Node* random;

Node(int _val) {
val = _val;
next = NULL;
random = NULL;
}
};
*/

class Solution {
public:
Node* copyRandomList(Node* head) {

}
};
```

**Java Solution:**

```
/**
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/*
// Definition for a Node.
class Node {
int val;
Node next;
Node random;

public Node(int val) {
this.val = val;
this.next = null;
this.random = null;
```

```
}
}
*/

class Solution {
public Node copyRandomList(Node head) {

}
}
```

### Python3 Solution:

```
"""
Problem: Copy List with Random Pointer
Difficulty: Medium
Tags: hash, linked_list

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

"""
# Definition for a Node.
class Node:
def __init__(self, x: int, next: 'Node' = None, random: 'Node' = None):
self.val = int(x)
self.next = next
self.random = random
"""

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

### Python Solution:

```
"""
# Definition for a Node.
class Node:
```

```
def __init__(self, x, next=None, random=None):
self.val = int(x)
self.next = next
self.random = random
"""

class Solution(object):
def copyRandomList(self, head):
"""
:type head: Node
:rtype: Node
"""
```

**JavaScript Solution:**

```
/**
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* // Definition for a _Node.
* function _Node(val, next, random) {
* this.val = val;
* this.next = next;
* this.random = random;
* };
*/

/**
* @param {_Node} head
* @return {_Node}
*/
var copyRandomList = function(head) {

};
```

### TypeScript Solution:

```
/**
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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


function copyRandomList(head: _Node | null): _Node | null {

};
```

### C# Solution:

```
/*
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
```

```
*/

/*
// Definition for a Node.
public class Node {
public int val;
public Node next;
public Node random;

public Node(int _val) {
val = _val;
next = null;
random = null;
}
}
*/

public class Solution {
public Node CopyRandomList(Node head) {

}
}
```

### C Solution:

```
/*
* Problem: Copy List with Random Pointer
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for a Node.
* struct Node {
* int val;
* struct Node *next;
* struct Node *random;
```

```
* };
*/

struct Node* copyRandomList(struct Node* head) {

}
```

### Go Solution:

```
// Problem: Copy List with Random Pointer
// Difficulty: Medium
// Tags: hash, linked_list
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

/**
* Definition for a Node.
* type Node struct {
* Val int
* Next *Node
* Random *Node
* }
*/

func copyRandomList(head *Node) *Node {

}
```

### Kotlin Solution:

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

```
*/

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

}
}
```

**Swift Solution:**

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

class Solution {
func copyRandomList(_ head: Node?) -> Node? {

}
}
```

**Ruby Solution:**

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

```
# @param {Node} node
# @return {Node}
def copyRandomList(head)

end
```

**PHP Solution:**

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

class Solution {
/**
* @param Node $head
* @return Node
*/
function copyRandomList($head) {

}
}
```

**Scala Solution:**

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

```
* }
*/

object Solution {
def copyRandomList(head: Node): Node = {

}
}
```