138. Copy List with Random Pointer

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

Return a **deep copy** of the 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`) where random pointer points to, or `null` if it does not point to any node.

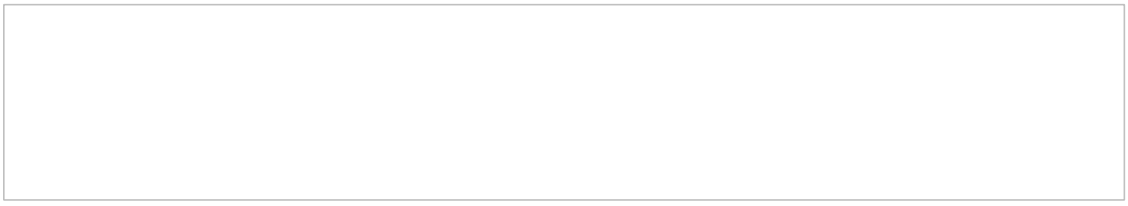
Example 1:


```
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]]
```

Example 4:

```
Input: head = []
Output: []
Explanation: Given linked list is empty (null pointer), so return null.
```

Constraints:

- `-10000 <= Node.val <= 10000`
- `Node.random` is null or pointing to a node in the linked list.
- Number of Nodes will not exceed 1000.

There are many ways to do this question, I will discuss two ways :
First of all, Deep Copy means creating exactly same copy as the original list

Deep copy

A shallow copy means constructing a new collection object and then populating it with references to the child objects found in the original.

---

1. **By modifying the original lists next pointer**

- here we first create a copy node (**node**)
- set **node->random = original node**
- preserve **next_original = original node->next**
- now change **original node->next = node**
- continue constructing copy node this way, U can see we are modifying next pointers of original list
- Now Your lists wil look something like :

- Now we need to set the random pointers of copy list
- use following formula :

```
node->random = node->random->random
if(node->random != NULL)
node->random = node->random->next
```

- **CODE:**

```
class Solution {
  public:
  Node* copyRandomList(Node* head) {
    if(!head || !head->next) return head;

    Node* temp_original = head;
    Node* root = new Node(temp_original->val);
    root->random = temp_original;

    Node* temp_copy = root;
    Node* next_original = temp_original->next;
    temp_original->next = temp_copy;

    while(temp_original){
       temp_original = next_original;
       Node* node = new Node(next_original->val);
       node->random = temp_original;
       temp_copy->next = node;
       temp_copy = node;

       next_original = temp_original->next;
       temp_original->next = temp_copy;
       if(!next_original)
       {
          temp_copy->next = NULL;
          break;
       }
    }
     Node* temp = root;
```

```
        while(temp){
            temp->random = temp->random->random;
            if(temp->random)
                temp->random = temp->random->next;
            temp = temp->next;
        }
        return root;
    }
};
```

This method doesn't work in the cases where you are not allowed to modify the original list

---

2. **By using map**

Create a map<Node*,Node*> m

- where first parameter contains original nodes, and second one contains copy nodes
- traverse through the original list and create copy nodes (without any links just with values)

  m[temp] = new Node(temp->val)

- Traverse original list again and this time set the links :

  m[temp]->next = m[temp->next]
  m[temp]->random = m[temp->random]
  take care of NULL values

- **return m[head]**
- **CODE:**

```
Node* copyRandomList(Node* head) {
        if(!head)
            return head;
        map<Node*,Node*> clever;
        Node *temp = head;
        while(temp){
            clever[temp] = new Node(temp->val);
            temp = temp->next;
        }
        temp = head;
        while(temp){
            if(temp->next)
            clever[temp]->next = clever[temp->next];
            else
                clever[temp]->next = NULL;
            if(temp->random)
            clever[temp]->random = clever[temp->random];
            else
                clever[temp]->random = NULL;
            temp = temp->next;
        }
        return clever[head];
    }
```

this solution is much simpler and clever!!!