Clone the L.L.

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`.

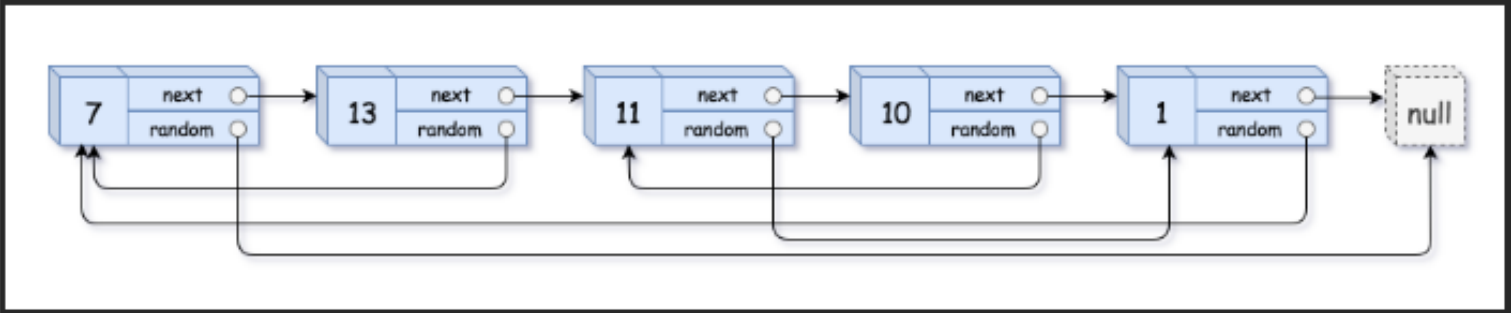
Return *the head of the copied linked list.*

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:

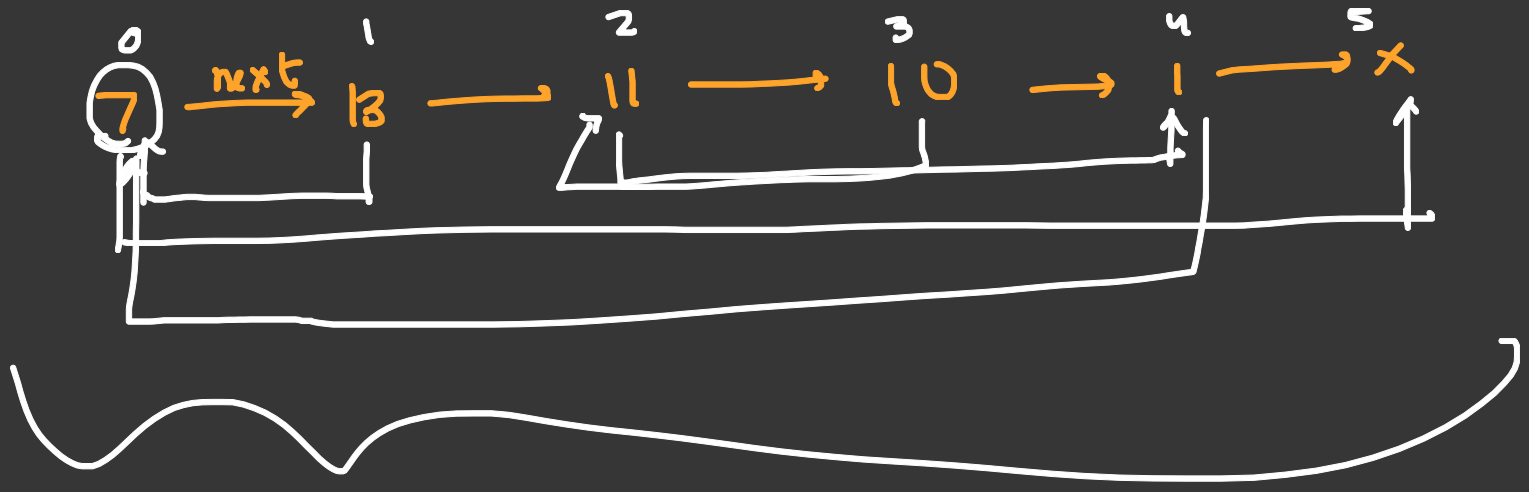
- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) that the `random` pointer points to, or `null` if it does not point to any node.

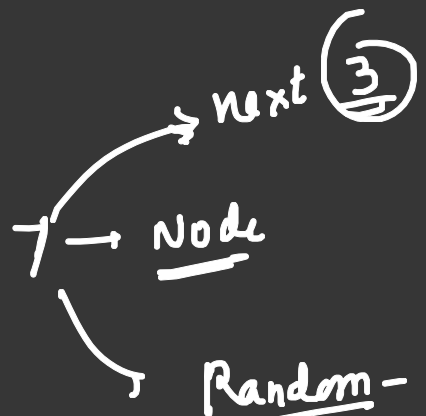
Your code will **only** be given the `head` of the original linked list.

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]]
```

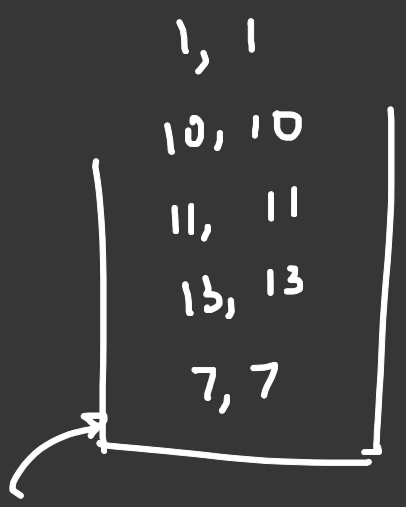


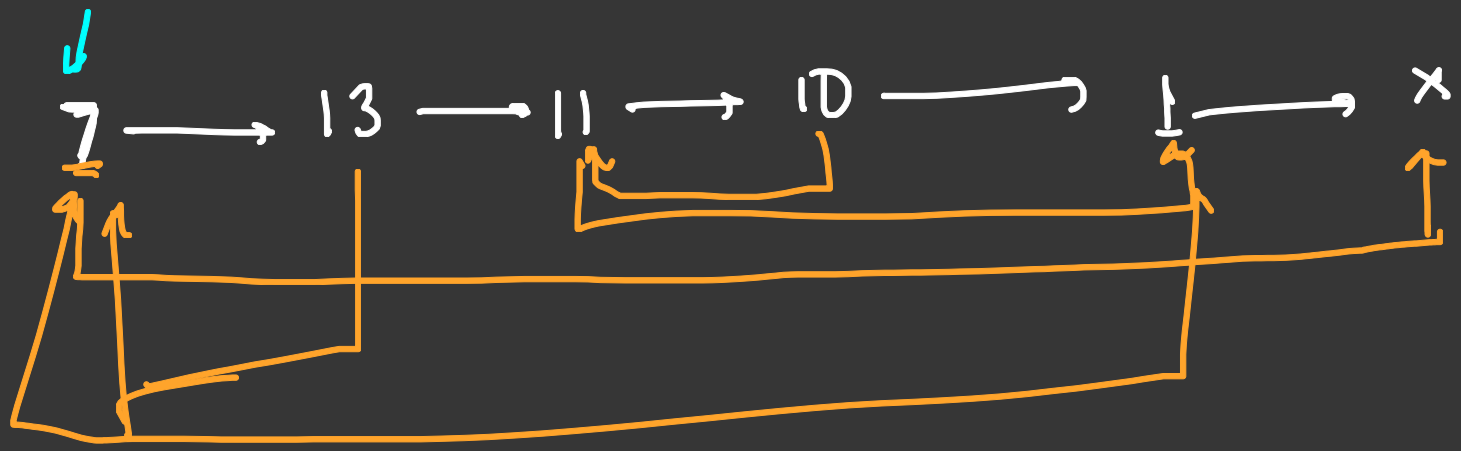


Store → new node → copy

Brute

hashmap < node*, node* >





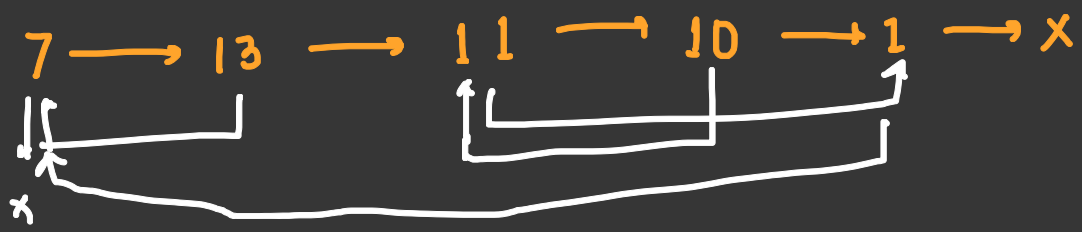
Node* node = hash[t]

node→next = hash[t→next]

node→random = hash[t→random]

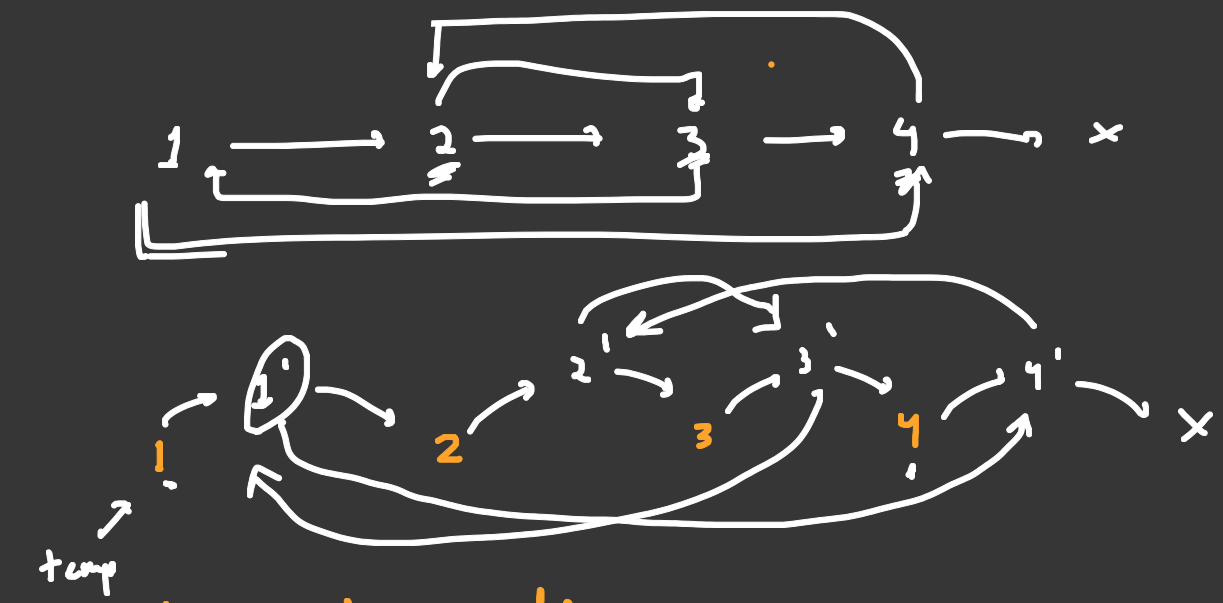
t = t→next;

Optimal Approach:



Idea is to insert a copy in the between the two 2 nodes.

- We create a copy node of node to its next.



temp → next → random

= temp → random
↓
next;

temp = temp → next → next.

- We connect random node

T.C ~ O(N)

S.C ~ O(1)