✍️ LeetCode Daily – Problem #138: Copy List with Random Pointer (Medium)

🚀 A deep copy challenge that sharpens understanding of recursive structures and pointer references. It's not just about copying nodes — it's about **preserving the structure** including random pointers that can point anywhere or even form cycles.

📌 Problem Overview

🧩 **Problem:**
Given a linked list where each node has a `next` pointer and a `random` pointer (which could point to any node or `null`), create a **deep copy** of the list.

🪆 **Example:**

```
Node 1 → Node 2 → Node 3 → Node 4 → Node 5 → None
Randoms: 1→None, 2→1, 3→5, 4→3, 5→1
```

🎯 Output: A **completely new list** where each node has the same value and `next`/`random` structure as the original — but no shared references.

🔍 Thought Process / Approach

🔁 **Approach: Recursion + Hash Map (Memoization)**

✅ **Step 1:** Use a dictionary `visited` to map original nodes to their copied counterparts.
✅ **Step 2:** For each node:

- If `None`: return `None`
- If already copied: return from `visited`
- Else: create a new node, store in map, recursively copy `next` and `random`

📌 **Why It Works:**
This handles both cycles (avoids infinite recursion) and shared references (avoids duplicating nodes pointed to by multiple random pointers).

📌 Steps:

1. Base case: If head is `None`, return `None`
2. If node already exists in map, return it
3. Create new node and store mapping
4. Recursively assign:
    - `node.next = copyRandomList(head.next)`
    - `node.random = copyRandomList(head.random)`

🧠 Key Concepts / Tags

🧩 **Concepts:** Hash Map, Recursion, Deep Copy
🏷️ **Tags:** Linked List, Hash Table, Recursion, Clone

⏰ Time & Space Complexity

🕐 **Time:** O(n) — Each node is visited once
💾 **Space:** O(n) — For hashmap and recursion stack

💬 Learning / Takeaway

🧠 **Takeaway:**
Learned how to make a **true deep copy** of a linked list structure, especially when it includes **complex random references**. Gained clarity on using **memoization** to manage already-seen nodes and prevent duplication or infinite loops.

This type of pattern is often used in **graph cloning**, **tree deep copies**, and object replication in real-world software.

✅ Clean Code

```
class Solution(object):
    def __init__(self):
        self.visited = {}

    def copyRandomList(self, head):
        if head is None:
            return None

        if head in self.visited:
            return self.visited[head]

        node = Node(head.val, None, None)
        self.visited[head] = node

        node.next = self.copyRandomList(head.next)
        node.random = self.copyRandomList(head.random)

        return node
```