leetcode-138

Problem Summary

Hume ek linked list di gayi hai jisme har node ke paas ek `next` pointer ke alawa ek `random` pointer bhi hota hai, jo kisi bhi node (ya `NULL`) ki taraf point kar sakta hai.

Hume iska **deep copy** banana hai jisme:

- Har copied node bilkul nayi memory me ho.
- `next` aur `random` pointers copied list ke andar hi point karein (original list ke nodes me nahi).
- Original aur copy list ka **structure and relation same** rahe.

Tere Code ka Step-by-Step Explanation

1. Base Case Handle Karna

```
if(head == NULL) return head;
```

Agar list khali hai (`head == NULL`), toh sidha `NULL` return kar do.

2. Copy Linked List Nodes Using `next` Pointers

```
Node* temp1 = head;
Node* copy_head = new Node(temp1->val);
temp1 = temp1->next;
Node* temp2 = copy_head;

while(temp1 != NULL) {
    Node* to_append = new Node(temp1->val);
    temp2->next = to_append;
    temp1 = temp1->next;
    temp2 = temp2->next;
}
```

- `temp1` → original list traverse karne ke liye.
- `temp2` → copied list traverse karne ke liye.
- Pehle ek `copy_head` banaya jo first copied node hai.
- Fir loop me original list ka har node copy karke nayi list me jod diya, bas `val` copy kiya aur `next` pointer set kiya.

📌 **Important**: Abhi `random` pointers set nahi hue, sirf `next` ka kaam hua hai.

---

3. Mapping Original Nodes to Copied Nodes

```
temp1 = head;
temp2 = copy_head;
unordered_map<Node*, Node*> m;

while(temp1 != NULL) {
    m[temp1] = temp2;
    temp1 = temp1->next;
    temp2 = temp2->next;
}
```

- Yahaan ek **hash map** (`unordered_map<Node*, Node*>`) banaya jisme:
  - **Key** = original node ka address
  - **Value** = uska corresponding copied node ka address

📌 Ye map isliye banaya gaya hai taaki jab `random` pointers set karein, toh pata ho ki original node ka `random` pointer kis copied node se map hota hai.

---

4. Random Pointers Set Karna

```
for(auto x : m) {
    Node* main_tree_node = x.first;
    Node* copy_tree_node = x.second;
    if(main_tree_node->random != NULL) {
        copy_tree_node->random = m[main_tree_node->random];
    }
}
```

- Har mapping ke liye:
  - Original node ka `random` pointer dekha.
  - Agar wo `NULL` nahi hai, toh map se uske corresponding copied node ka address nikal ke copied list ke node ka `random` set kar diya.

📌 **Logic**:

Original ka `random` → kisi original node ko point karega

Map ka use karke → us original node ka copied version nikal lo → copied ka `random` usko point kara do.

---

5. Final Answer

```
return copy_head;
```

- Copied linked list ka head return kiya.

Time & Space Complexity Analysis

- **Time Complexity**:
  - Step 2 (copy nodes) → **O(n)**
  - Step 3 (map ban[illegible]ना) → **O(n)**
  - Step 4 (random set करना) → **O(n)**
  - **Total → O(n)**
- **Space Complexity**:
  - `unordered_map` me n entries → **O(n)**

Dry Run Example

Input

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

Original List Structure:

```
7 → 13 → 11 → 10 → 1
     |    |     |
    (7)  (1)   (11)
```

Step 2 Output (only next pointers copy):

```
7' → 13' → 11' → 10' → 1'
(random abhi NULL)
```

Step 3 Map (original → copy):

```
7   → 7'
13  → 13'
11  → 11'
10  → 10'
1   → 1'
```

Step 4 Setting Random:

- 13 ka random = 7 → map → 13' ka random = 7'
- 11 ka random = 1 → map → 11' ka random = 1'
- 10 ka random = 11 → map → 10' ka random = 11'
- 1 ka random = 7 → map → 1' ka random = 7'

Final Copied List:

```
7' → 13' → 11' → 10' → 1'
     |      |     |
    (7')   (1')  (11')
```

Bhai, ye explanation future me tu padhega to easily samajh jayega ki

- Pehle `next` pointer ka copy hota hai
- Fir ek map banake original aur copy nodes ka relation store hota hai
- Fir `random` pointers set hote hain map ka use karke
