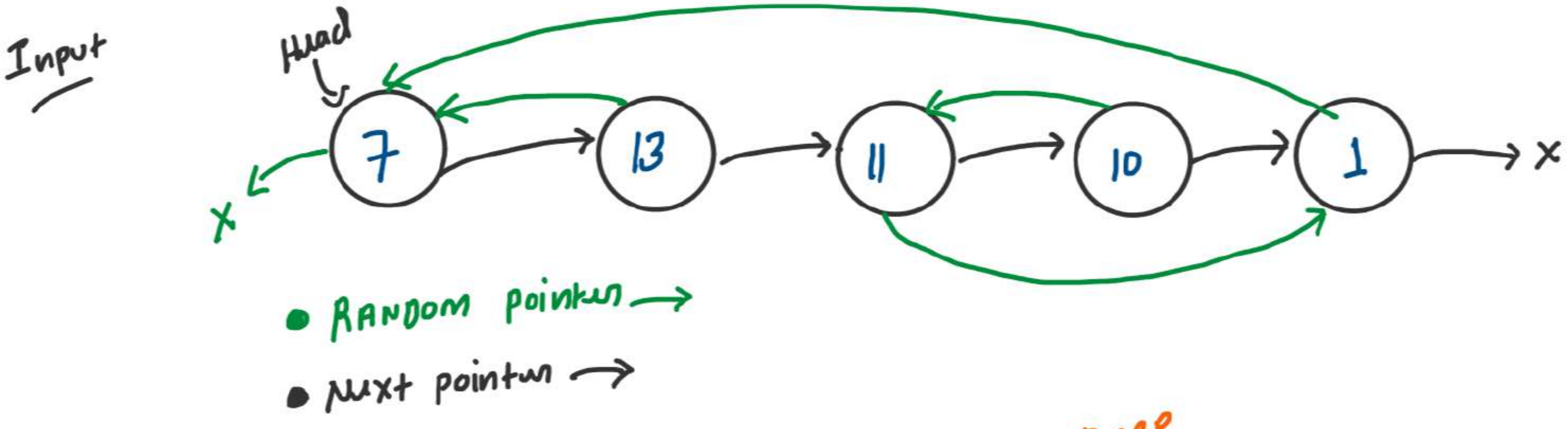
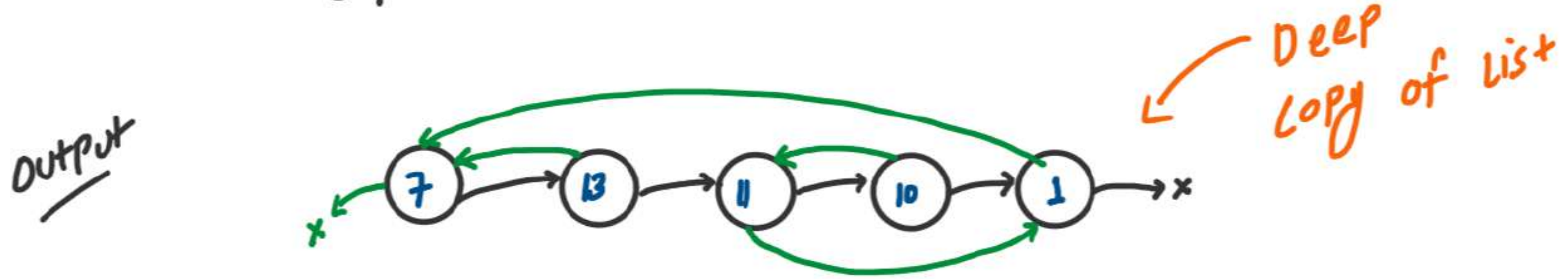
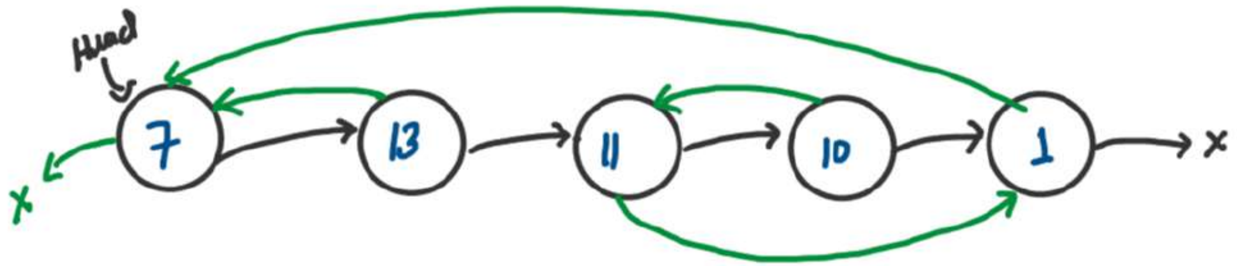
HW 07: Copy List with Random Pointer (Leetcode-138)





Approach 1 Using map

step1 Copy List in map
using Recursion

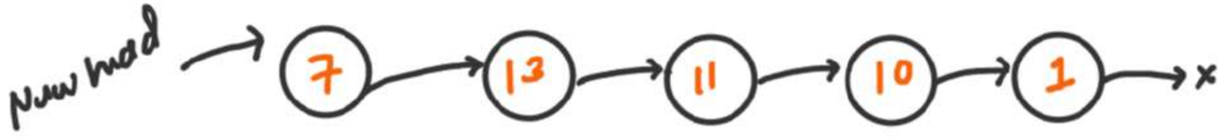


MP

Old ptr	New ptr
7	7
13	13
11	11
10	10
1	1

Base case
if(!head) return Null;

```
Node* newNode = new Node(head->val);
MP[head] = newhead;
newNode->next = f(head->next, MP);
```



Step2 Allocate the Random pointer

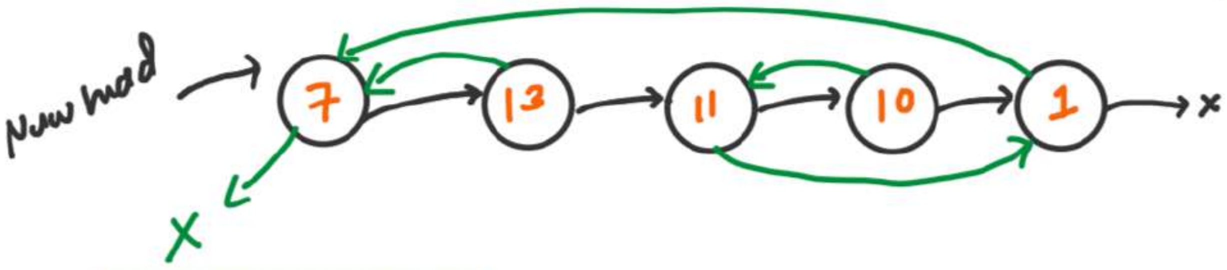
```
if( head→Random) {
    Newhead→Random = mp[head→Random];
}
```

**

Map[old ptr] = New pointer

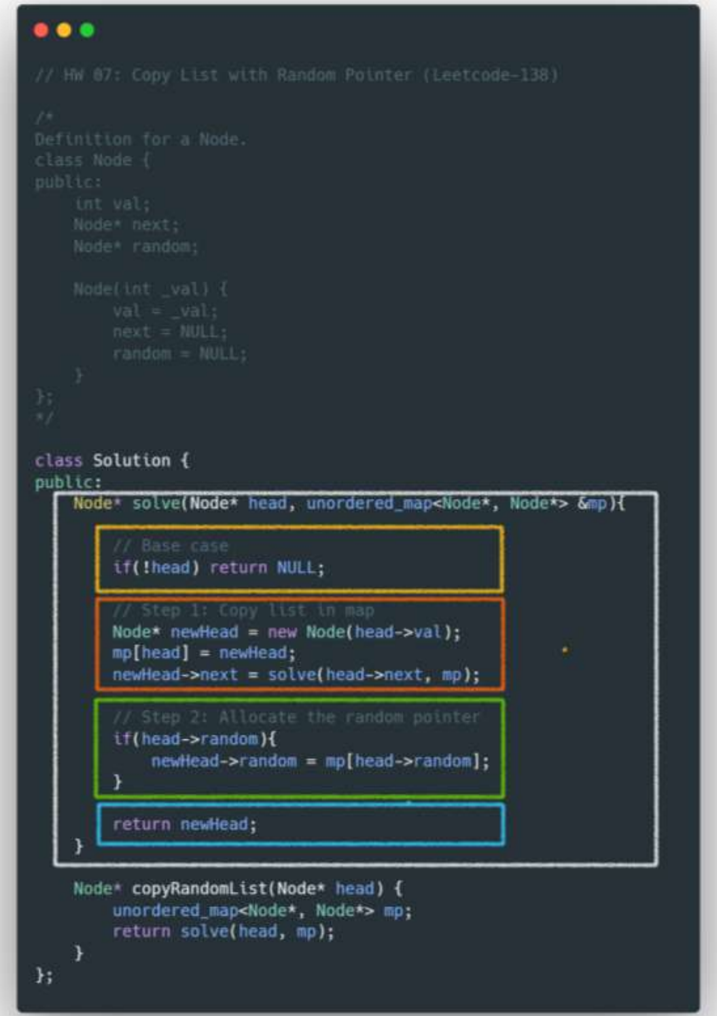
key of map

value of map



```
Return Newhead
```

Output



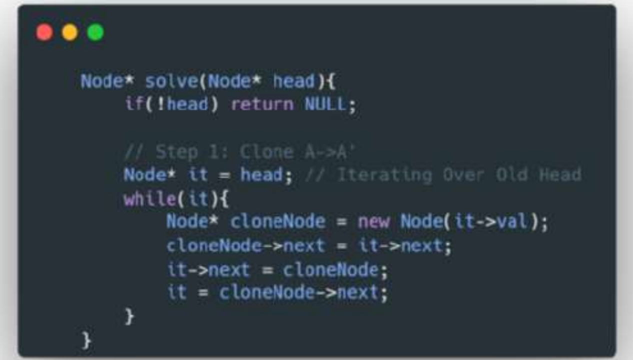
```cpp
// HW 07: Copy List with Random Pointer (Leetcode-138)

/*
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* solve(Node* head, unordered_map<Node*, Node*> &mp){

        // Base case
        if(!head) return NULL;

        // Step 1: Copy list in map
        Node* newHead = new Node(head->val);
        mp[head] = newHead;
        newHead->next = solve(head->next, mp);

        // Step 2: Allocate the random pointer
        if(head->random){
            newHead->random = mp[head->random];
        }

        return newHead;
    }

    Node* copyRandomList(Node* head) {
        unordered_map<Node*, Node*> mp;
        return solve(head, mp);
    }
};
```

Time Complexity: *O(N),*
Where N is number of nodes in list

Space Complexity: *O(N),*
where N is number of elements (Nodes) stored in map

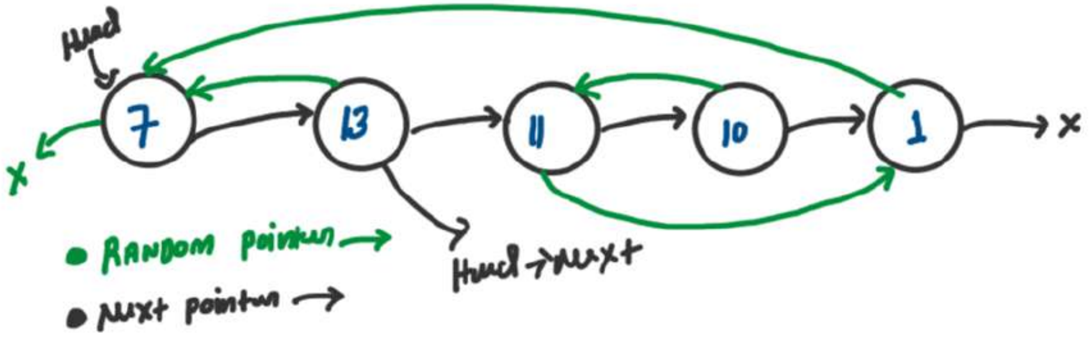
Approach 2 without using map

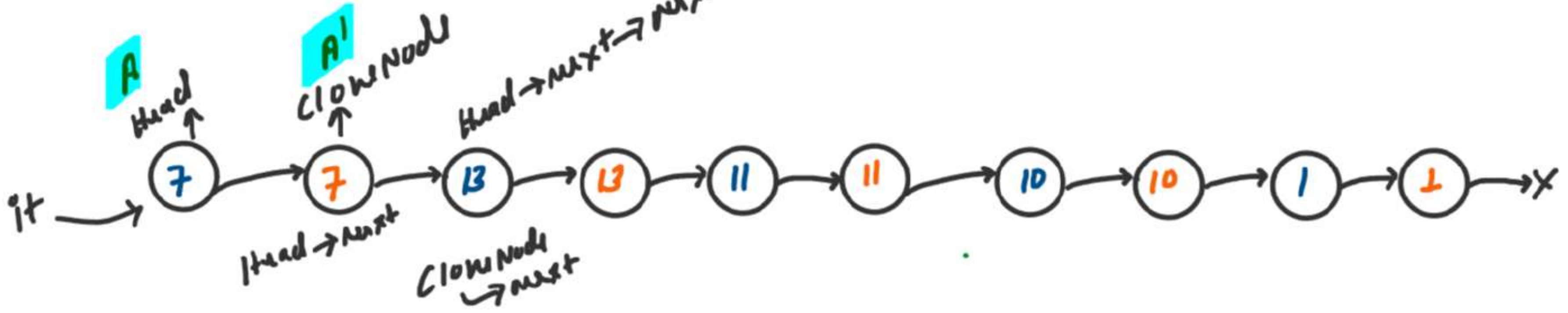
Step:1



```
Node* solve(Node* head){
    if(!head) return NULL;

    // Step 1: Clone A->A'
    Node* it = head; // Iterating Over Old Head
    while(it){
        Node* cloneNode = new Node(it->val);
        cloneNode->next = it->next;
        it->next = cloneNode;
        it = cloneNode->next;
    }
}
```

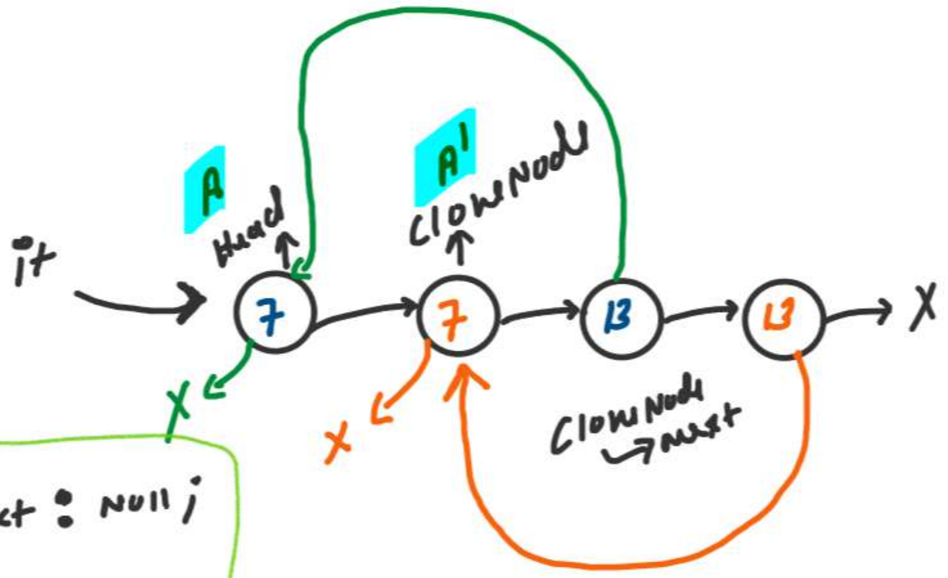
Input





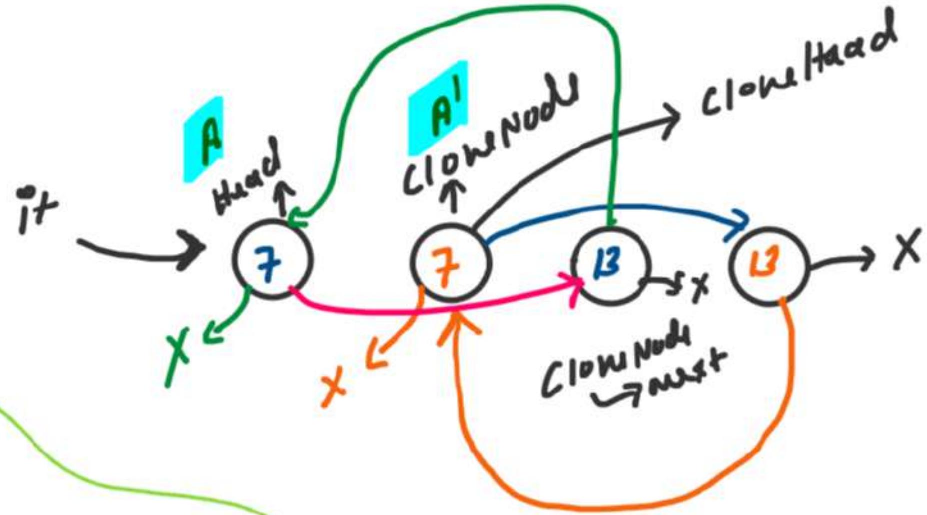
step2 Assign Random pointer A'
with the help of A

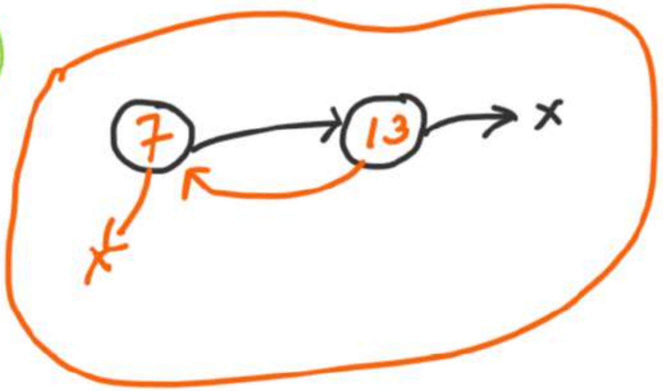
```
it = head ;
while ( it) {
    Node* cloneNode = it->next;
    cloneNode->Random =
        it->Random ? it->Random->next : Null ;
    it = it->next->next ;
}
```

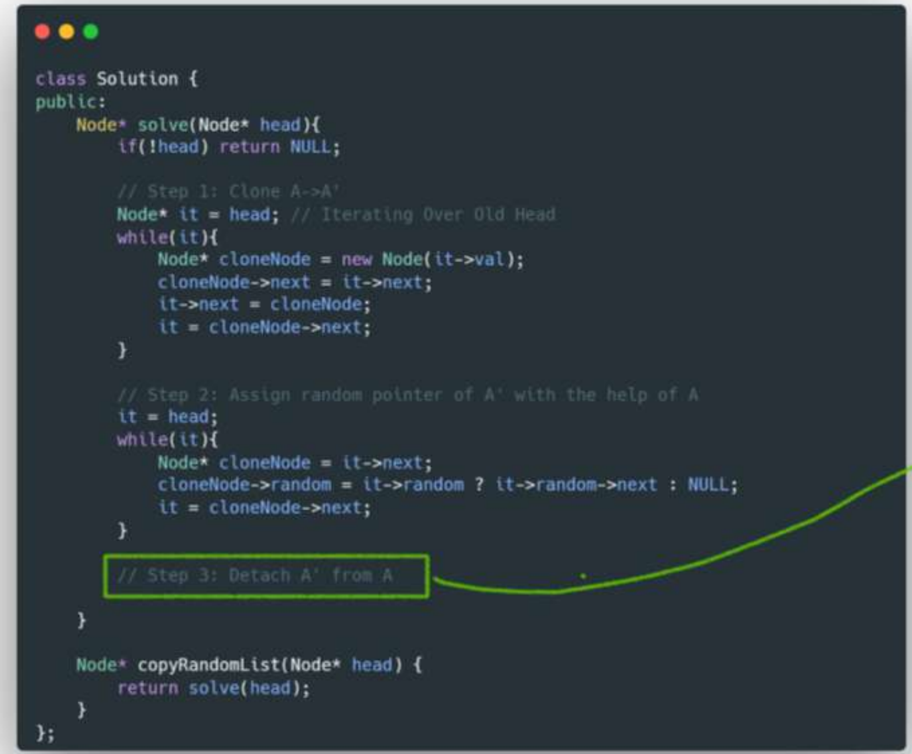


Step3 Detach A' from A

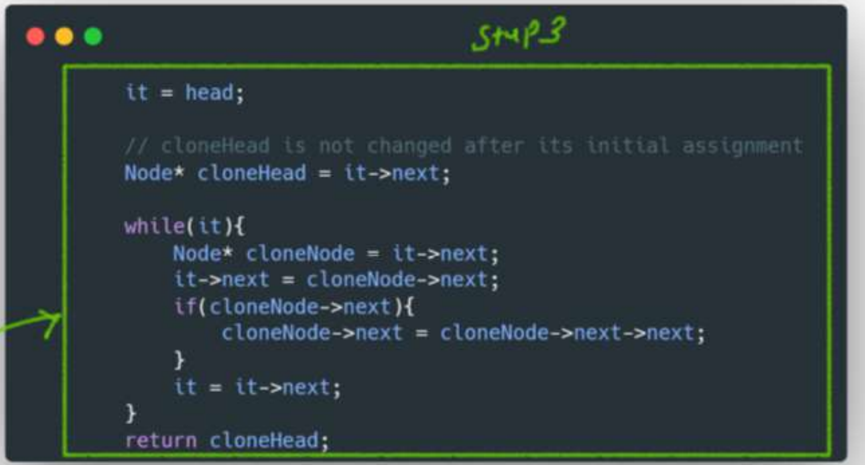
```
it = head;
Node* cloneHead = it->next;
while(it) {
    Node* cloneNode = it->next;
    it->next = cloneNode->next;
    if(cloneNode->next) {
        cloneNode->next = cloneNode->next->next;
    }
    it = it->next;
}
return cloneHead;
```







```cpp
class Solution {
public:
    Node* solve(Node* head){
        if(!head) return NULL;

        // Step 1: Clone A->A'
        Node* it = head; // Iterating Over Old Head
        while(it){
            Node* cloneNode = new Node(it->val);
            cloneNode->next = it->next;
            it->next = cloneNode;
            it = cloneNode->next;
        }

        // Step 2: Assign random pointer of A' with the help of A
        it = head;
        while(it){
            Node* cloneNode = it->next;
            cloneNode->random = it->random ? it->random->next : NULL;
            it = cloneNode->next;
        }

        // Step 3: Detach A' from A

    }

    Node* copyRandomList(Node* head) {
        return solve(head);
    }
};
```



```cpp
        it = head;

        // cloneHead is not changed after its initial assignment
        Node* cloneHead = it->next;

        while(it){
            Node* cloneNode = it->next;
            it->next = cloneNode->next;
            if(cloneNode->next){
                cloneNode->next = cloneNode->next->next;
            }
            it = it->next;
        }
        return cloneHead;
```

T.C. ⇒ O(N)

S.C. ⇒ O(1)