Linked List_1

@kbbhatt04

@September 5, 2023

Copy List with Random Pointer

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. Return head of new list.

- Approach
  - Brute-force
    - Use hash table to store the mapping between each node in the original list and its corresponding node in the copied list
    - In first pass, create a new node corresponding to a node in given list
    - In second pass, set its corresponding new node's `next` and `random` pointers based on the hash map
    - Time Complexity: $O(2n)$
    - Space Complexity: $O(n)$
  - Optimal
    - For each node, create a corresponding new node and place it between the current node and the current node's `next`
    - Traverse the interweaved list. For each old node, set its corresponding new node's `random` pointer.
    - Traverse the interweaved list again to separate the old and new lists
    - Time Complexity: $O(3n)$
    - Space Complexity: $O(1)$

```
# Python3
# Brute-force Solution
class Solution:
    def copyRandomList(self, head):
        if not head:
            return None
        old_to_new = {}

        curr = head
        while curr:
            old_to_new[curr] = Node(curr.val)
            curr = curr.next

        curr = head
        while curr:
            old_to_new[curr].next = old_to_new.get(curr.next)
            old_to_new[curr].random = old_to_new.get(curr.rando
            curr = curr.next

        return old_to_new[head]
```

```
# Python3
# Optimal Solution
class Solution:
    def copyRandomList(self, head):
        ll_iter = head
        while ll_iter:
            cur_next = ll_iter.next
            new_node = Node(ll_iter.val, cur_next)
            ll_iter.next = new_node
            ll_iter = cur_next

        ll_iter = head
        while ll_iter:
            if ll_iter.random:
                ll_iter.next.random = ll_iter.random.next
            ll_iter = ll_iter.next.next

        ll_iter = head
        new_list = Node(0)
        new_list_head = new_list
        while ll_iter:
            new_list.next = ll_iter.next
            ll_iter.next = ll_iter.next.next
            ll_iter = ll_iter.next
            new_list = new_list.next

        return new_list_head.next
```

```
// C++
// Optimal Solution
class Solution {
public:
    Node* copyRandomList(Node* head) {
                if (!head) return nullptr;

        Node* curr = head;
        while (curr) {
            Node* new_node = new Node(curr->val);
            new_node->next = curr->next;
            curr->next = new_node;
            curr = new_node->next;
        }

        curr = head;
        while (curr) {
            if (curr->random) {
                curr->next->random = curr->random->next;
            }
            curr = curr->next->next;
        }

        Node* old_head = head;
        Node* new_head = head->next;
        Node* curr_old = old_head;
        Node* curr_new = new_head;

        while (curr_old) {
            curr_old->next = curr_old->next->next;
            curr_new->next = curr_new->next ? curr_new->next->ne
            curr_old = curr_old->next;
            curr_new = curr_new->next;
        }

        return new_head;
    }
};
```

Template

- Approach
    - Brute-force
        -
        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$
    - Better
        -
        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$

    - Optimal
        -
        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$

```
# Python3
# Brute-force Solution
```

```
# Python3
# Better Solution
```

```
# Python3
# Optimal Solution
```

```
// C++
// Optimal Solution
```