Wednesday, January 19, 2022 12:02 PM

Clone a linked list with next and random pointer

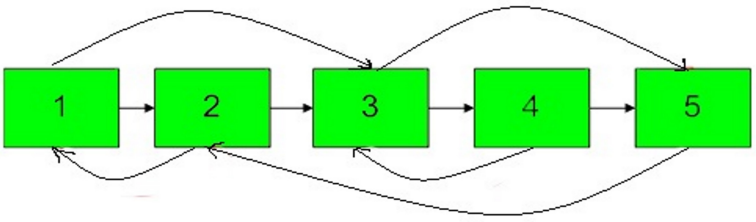
Hard Accuracy: 49.62% Submissions: 35797 Points: 8

You are given a special linked list with **N** nodes where each node has a next pointer pointing to its next node. You are also given **M** random pointers, where you will be given **M** number of pairs denoting two nodes **a** and **b** **i.e. a->arb = b.**

Construct a copy of the given list. The copy should consist of exactly **N** 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.



Example 1:

Input:
N = 4, M = 2
value = {1,2,3,4}
pairs = {{1,2},{2,4}}
Output: 1
Explanation: In this test case, there
are 4 nodes in linked list. Among these
4 nodes, 2 nodes have arbitrary pointer
set, rest two nodes have arbitrary pointer
as NULL. Second line tells us the value
of four nodes. The third line gives the
information about arbitrary pointers.
The first node arbitrary pointer is set to
node 2. The second node arbitrary pointer

is set to node 4.
Example 2:

Input:
N = 4, M = 2
value[] = {1,3,5,9}
pairs[] = {{1,1},{3,4}}
Output: 1
Explanation: In the given testcase , applying the method as stated in the above example, the output will be 1.
Your Task:

The task is to complete the function **copyList**() which takes one argument the head of the linked list to be cloned and should **return** the head of the cloned linked list.

NOTE :

1. If there is any node whose arbitrary pointer is not given then it's by default NULL.

2. Your solution return an output **1** if your clone linked list is correct, else it returns **0**.

Expected Time Complexity : O(n)

Expected Auxilliary Space : O(1)

Constraints:

1 <= N <= 100

1 <= M <= N

1 <= a, b <= 100

Expected Time Complexity : O(n)
Expected Auxilliary Space : O(1)

Constraints:
1 <= N <= 100
1 <= M <= N
1 <= a, b <= 100

View Bookmarked Problems

Company Tags

Adobe Amazon BankBazaar D-E-Shaw
MakeMyTrip Microsoft Morgan Stanley
Ola Cabs OYO Rooms Snapdeal Walmart
Flipkart

Related Interview Experiences

- Bankbazaar com interview experience set 7 senior android developer
- Ola cabs interview experience set 2 android 1 5 years
- Makemytrip interview experience set 4
- Makemytrip interview experience set 8 on campus

138. Copy List with Random Pointer

Medium

6972907Add to ListShare

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`.

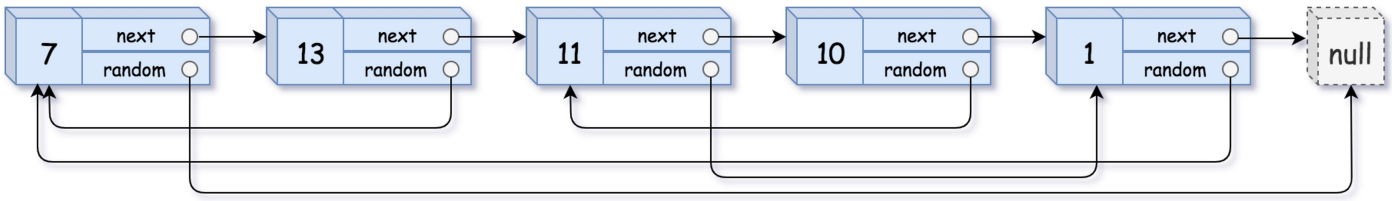
Return *the head of the copied linked list*.

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:

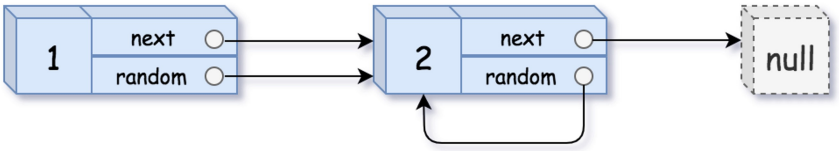
- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) that the `random` pointer points to, or `null` if it does not point to any node.

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

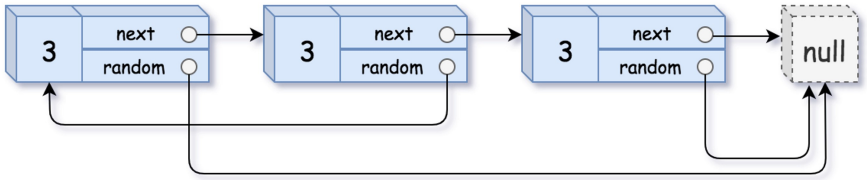
Example 1:



Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]
Example 2:



Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]
Example 3:



Input: head = [[3,null],[3,0],[3,null]]
Output: [[3,null],[3,0],[3,null]]

Constraints:

- `0 <= n <= 1000`
- `-10⁴ <= Node.val <= 10⁴`
- `Node.random` is `null` or is pointing to some node in the linked list.

From <https://leetcode.com/problems/copy-list-with-random-pointer/>

Using Hash Map

```
import java.util.*;
public class CloneWithRandomPointers {
    class Node {
        int val;
        Node next;
        Node random;
        public Node(int val) {
            this.val = val;
            this.next = null;
            this.random = null;
```

```
        }
    }
    public Node copyRandomList(Node head) {
        if (head == null)
            return head;
        Map<Node, Node> map = new HashMap<>();
        Node temp=head;
        while(temp!=null)
        {
            map.put(temp, new Node (temp.val));
            temp=temp.next;

        }
        temp=head;
        while(temp!=null)
        {
            Node tempsClone=map.get(temp);
            //  placing clones next at its position
            tempsClone.next=map.get(temp.next);

            //  now place the random of the clones
            Node random=temp.random;
            Node randomsClone=map.get(random);
            tempsClone.random=randomsClone;
            // now move ahead
            temp=temp.next;

        }









        return null;
    }
}
```

Using Constant Space

An intuitive solution is to keep a hash table for each node in the list, via which we just need to iterate the list in 2 rounds respectively to create nodes and assign the values for their random pointers. As a result, the space complexity of this solution is O(N), although with a linear time complexity.
Note: if we do not consider the space reversed for the output, then we could say that the algorithm does not consume any additional memory during the processing, i.e. O(1) space complexity
As an optimised solution, we could reduce the space complexity into constant. ***The idea is to associate the original node with its copy node in a single linked list. In this way, we don't need extra space to keep track of the new nodes.***
The algorithm is composed of the follow three steps which are also 3 iteration rounds.

1. Iterate the original list and duplicate each node. The duplicate of each node follows its original immediately.
2. Iterate the new list and assign the random pointer for each duplicated node.
3. Restore the original list and extract the duplicated nodes.

The algorithm is implemented as follows:

From <https://leetcode.com/problems/copy-list-with-random-pointer/discuss/43491/A-solution-with-constant-space-complexity-O(1)-and-linear-time-complexity-O(N)>

```
public RandomListNode copyRandomList(RandomListNode head) {
  RandomListNode iter = head, next;
  // First round: make copy of each node,
  // and link them together side-by-side in a single list.
  while (iter != null) {
    next = iter.next;
    RandomListNode copy = new RandomListNode(iter.label);
    iter.next = copy;
    copy.next = next;
    iter = next;
  }
  // Second round: assign random pointers for the copy nodes.
  iter = head;
  while (iter != null) {
    if (iter.random != null) {
      iter.next.random = iter.random.next;
    }
    iter = iter.next.next;
  }
  // Third round: restore the original list, and extract the copy list.
  iter = head;
  RandomListNode pseudoHead = new RandomListNode(0);
  RandomListNode copy, copyIter = pseudoHead;
  while (iter != null) {
    next = iter.next.next;
    // extract the copy
    copy = iter.next;
    copyIter.next = copy;
    copyIter = copy;
    // restore the original list
    iter.next = next;
    iter = next;
  }
  return pseudoHead.next;
}
```