**138. Copy List with Random Pointer**

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

1. **Listen**

**Problem Description:**

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.

**Input**:

**head** of original singly **linked list** with extra random pointer

**Goal**:

Deep copy the **original linked list** with all new nodes

**Return**:

head of the deep copied linked list

1. **Example**

**Constraints:**

* 0 <= n <= 1000
* -104 <= Node.val <= 104
* Node.random is null or is pointing to some node in the linked list.
* None of the pointers in the new list should point to nodes in the original list.

**Test Cases:**

* linked list is even length
* linked list is odd length

**Edge Cases:**

* linked list is empty
* linked list has single node

**Questions & Assumptions:**

* Can a node’s random pointer point to itself?

**Example 1**

Diagram

Description automatically generated

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

**Output:** [[7,null],[13,0],[11,4],[10,2],[1,0]]

1. **Brute Force**

**Solution 1:**

Make two passes over the original linked list

Pass 1: Create a deep copy linked list of the original’s values. Store the original node as a key to the newly created copy node in a HashMap.

Pass 2: Iterate over the deep copy and assign the next and random pointers according to values from the HashMap.

We can accomplish this in O(1n + 2n) = O(n) time and O(n + n) space (n for hashmap and n for newly copied nodes).

1. **Optimize**

**Solution 2:**

We can remove the need for a HashMap by performing the copying in place.

This can be accomplished in three passes:

1. Iterate over and copy each original node’s value and next pointer. The duplicate  
   of each node follows its original immediately.
2. Iterate over and copy each original nodes random pointer.
3. Restore the original list and extract the duplicate nodes into their own list.

Solution 1: O(n+n) = O(n) time

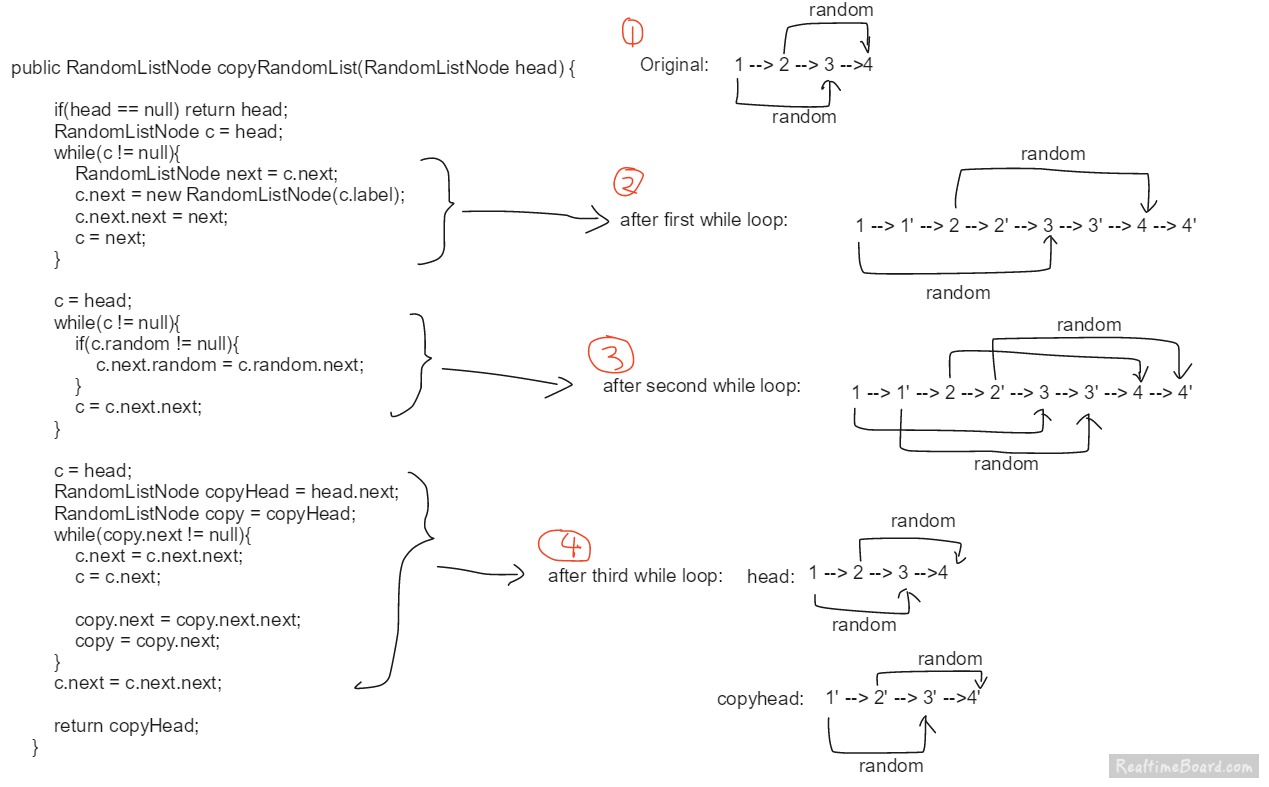
Solution 2: O(n+n+n) = O(n) time

Solution 1: O(n+n) = O(n) space

Solution 2: O(n) = O(n) space

We can sacrifice an extra time constant to save on a space constant.

1. **Implement**



1. **Test**