Copy List with Random Pointer

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Solution 1

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.

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:

```
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;
}
```

written by liaison original link here

Solution 2

The idea is: Step 1: create a new node for each existing node and join them together eg: A->B->C will be A->A'->B->B'->C->C'

Step2: copy the random links: for each new node n', n'.random = n.random.next

Step3: detach the list: basically n.next = n.next.next; n'.next = n'.next.next

Here is the code:

```java
/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *     RandomListNode(int x) { this.label = x; }
 * };
 */
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {
        if(head==null){
            return null;
        }
        RandomListNode n = head;
        while (n!=null){
            RandomListNode n2 = new RandomListNode(n.label);
            RandomListNode tmp = n.next;
            n.next = n2;
            n2.next = tmp;
            n = tmp;
        }

        n=head;
        while(n != null){
            RandomListNode n2 = n.next;
            if(n.random != null)
                n2.random = n.random.next;
            else
                n2.random = null;
            n = n.next.next;
        }

        //detach list
        RandomListNode n2 = head.next;
        n = head;
        RandomListNode head2 = head.next;
        while(n2 != null && n != null){
            n.next = n.next.next;
            if (n2.next == null){
                break;
            }
            n2.next = n2.next.next;

            n2 = n2.next;
            n = n.next;
        }
        return head2;

    }
}
```

written by sharon2 original link here

Solution 3

```
//
// Here's how the 1st algorithm goes.
// Consider l1 as a node on the 1st list and l2 as the corresponding node on 2nd l
ist.
// Step 1:
// Build the 2nd list by creating a new node for each node in 1st list.
// While doing so, insert each new node after it's corresponding node in the 1st l
ist.
// Step 2:
// The new head is the 2nd node as that was the first inserted node.
// Step 3:
// Fix the random pointers in the 2nd list: (Remember that l1->next is actually l2
)
// l2->random will be the node in 2nd list that corresponds l1->random,
// which is next node of l1->random.
// Step 4:
// Separate the combined list into 2: Splice out nodes that are part of second lis
t.
// Return the new head that we saved in step 2.
//

RandomListNode *copyRandomList(RandomListNode *head) {
    RandomListNode *newHead, *l1, *l2;
    if (head == NULL) return NULL;
    for (l1 = head; l1 != NULL; l1 = l1->next->next) {
        l2 = new RandomListNode(l1->label);
        l2->next = l1->next;
        l1->next = l2;
    }

    newHead = head->next;
    for (l1 = head; l1 != NULL; l1 = l1->next->next) {
        if (l1->random != NULL) l1->next->random = l1->random->next;
    }

    for (l1 = head; l1 != NULL; l1 = l1->next) {
        l2 = l1->next;
        l1->next = l2->next;
        if (l2->next != NULL) l2->next = l2->next->next;
    }

    return newHead;
}


//
// Here's how the 2nd algorithm goes.
// Consider l1 as a node on the 1st list and l2 as the corresponding node on 2nd l
ist.
// Step 1:
// Build the 2nd list by creating a new node for each node in 1st list.
// While doing so, set the next pointer of the new node to the random pointer
// of the corresponding node in the 1st list.  And set the random pointer of the
// 1st list's node to the newly created node.
```

```
// Step 2:
// The new head is the node pointed to by the random pointer of the 1st list.
// Step 3:
// Fix the random pointers in the 2nd list: (Remember that l1->random is l2)
// l2->random will be the node in 2nd list that corresponds to the node in the
// 1st list that is pointed to by l2->next,
// Step 4:
// Restore the random pointers of the 1st list and fix the next pointers of the
// 2nd list. random pointer of the node in 1st list is the next pointer of the
// corresponding node in the 2nd list.  This is what we had done in the
// 1st step and now we are reverting back. next pointer of the node in
// 2nd list is the random pointer of the node in 1st list that is pointed to
// by the next pointer of the corresponding node in the 1st list.
// Return the new head that we saved in step 2.
//

RandomListNode *copyRandomList(RandomListNode *head) {
    RandomListNode *newHead, *l1, *l2;
    if (head == NULL) return NULL;

    for (l1 = head; l1 != NULL; l1 = l1->next) {
        l2 = new RandomListNode(l1->label);
        l2->next = l1->random;
        l1->random = l2;
    }

    newHead = head->random;
    for (l1 = head; l1 != NULL; l1 = l1->next) {
        l2 = l1->random;
        l2->random = l2->next ? l2->next->random : NULL;
    }

    for (l1 = head; l1 != NULL; l1 = l1->next) {
        l2 = l1->random;
        l1->random = l2->next;
        l2->next = l1->next ? l1->next->random : NULL;
    }

    return newHead;
}
```

written by satyakam original link here