


题目一：

138. 随机链表的复制

已解答 

中等 相关标签 相关企业 提示 Aa

给你一个长度为 `n` 的链表，每个节点包含一个额外增加的随机指针 `random`，该指针可以指向链表中的任何节点或空节点。

构造这个链表的 **深拷贝**。深拷贝应该正好由 `n` 个 **全新** 节点组成，其中每个新节点的值都设为其对应的原节点的值。新节点的 `next` 指针和 `random` 指针也都应指向复制链表中的新节点，并使原链表和复制链表中的这些指针能够表示相同的链表状态。**复制链表中的指针都不应指向原链表中的节点。**

例如，如果原链表中有 `x` 和 `y` 两个节点，其中 `x.random --> y`。那么在复制链表中对应的两个节点 `x` 和 `y`，同样有 `x.random --> y`。

返回复制链表的头节点。

用一个由 `n` 个节点组成的链表来表示输入/输出中的链表。每个节点用一个 `[val, random_index]` 表示：

- `val`：一个表示 `Node.val` 的整数。
- `random_index`：随机指针指向的节点索引（范围从 `0` 到 `n-1`）；如果不指向任何节点，则为 `null`。

你的代码 **只** 接受原链表的头节点 `head` 作为传入参数。

示例 1：

解题思路一：

使用哈希表记录原链表中已经拷贝过的数据，由于是使用回溯的方法对数据进行拷贝，因此，有的数据可能会出现不止一次，如果能在哈希表中找到对应的节点，说明这个节点已经拷贝过，直接return即可，对于链表中的next和val依次遍历即可

```
class Solution {
public:
    //定义哈希表存储新链表数据
    unordered_map<Node*, Node*> cachedNode;

    Node* copyRandomList(Node* head) {
        if (head == nullptr) {
            return nullptr;
        }
        if (!cachedNode.count(head)) {
            //当前链表的值为原链表对应节点的值
            Node* headNew = new Node(head->val);
            //当前节点入哈希表，便于之后检测
            cachedNode[head] = headNew;
        }
        headNew->next = copyRandomList(head->next);
        headNew->random = copyRandomList(head->random);
        return headNew;
    }
};
```

```

        //递归实现next和random节点的复制
        headNew->next = copyRandomList(head->next);
        headNew->random = copyRandomList(head->random);
    }
    return cachedNode[head];
}
};

```

解题思路二：

我们可以在原链表的基础上，每个节点依次加入一个相同的节点，即x->y->z变成x->x->y->y那么只要再把这个链表拆开成为两条链表即可，那就一定有一个是从head开始，一个从head->next，head开始的指向head->next->next即可。在添加过程中，我们需要使用双指针来处理下一个节点的链接，对于random节点，我们只需要在next节点添加后，判断当前节点的random是否为空，为空就变成next->random=random->next

```

class Solution {
public:
    Node* copyRandomList(Node* head) {
        if (!head) return nullptr; // 空链表直接返回nullptr
        //首先得处理原链表的下一个节点
        Node* cur = head;
        while (cur)
        {
            Node* tmp = new Node(cur->val);
            //获取下一个节点
            tmp->next = cur->next;
            //把当前节点加入进去
            cur->next = tmp;
            //跳转到原链表的下一个节点
            cur = tmp->next;
        }
        //恢复至原链表的表头
        cur = head;
        while (cur)
        {
            if (cur->random)
            {
                //复制原链表的随机指向
                cur->next->random = cur->random->next;
            }
            //回到原链表下一个
            cur = cur->next->next;
        }
        cur = head;
        Node* newhead = head->next;
        Node* newtmp = newhead;
        while (cur)
        {
            //恢复原链表next
            cur->next = newtmp->next;

```

```
        if (newtmp->next)
        {
            //恢复复制链表下一个链表节点
            newtmp->next = cur->next->next;
        }
        //新链表是原链表下一个节点的下一个
        cur = cur->next;
        newtmp = newtmp->next;
    }
    return newhead;
}
};
```