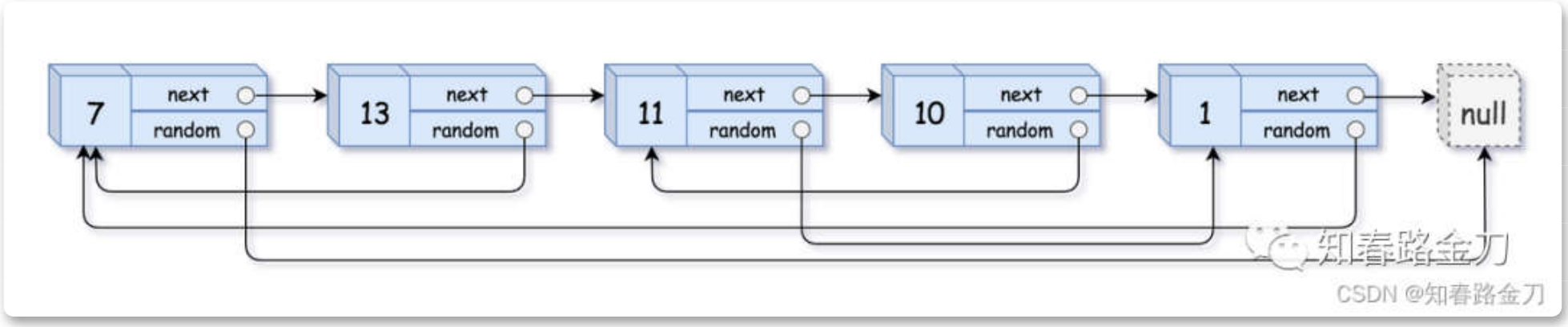
题目介绍[1]

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 。

示例 1



> **「输入」**：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
>
> **「输出」**：[[7,null],[13,0],[11,4],[10,2],[1,0]]

题目解答

方法一：哈希表

思路和算法

本题重点是对这个链表进行**「深拷贝」**，也就是说不能改变原有链表，且复制一个新链表出来。

首先想到的一个简单的思路就是使用哈希表，这个哈希表的 key 存储原有链表的节点，哈希表的 value 存储新链表的节点。

- 首先遍历整个链表来构建哈希表，遇到的每一个原有链表中的节点就构建一个新节点，同时原有链表的节点作为 key，新节点作为 value 放进哈希表中。
  - 这个步骤操作完之后，哈希表中就有了原有链表节点和新节点的对应关系了。
- 再遍历一次链表，从哈希表中获取新节点，以及新节点的 next 和 random 节点，这样就完成了链表的深拷贝。

这个算法有一个坏处就是需要使用哈希表来存储新旧节点的对应关系，浪费了额外的内存空间，后面还会讲解一个最优的算法思路。

代码实现

```
class Solution {
    Map<Node, Node> cachedNode = new HashMap<Node, Node>();
    public Node copyRandomList(Node head) {
        for (Node p = head; p != null; p = p.next) {
            cachedNode.put(p, new Node(p.val));
        }
        for (Node p = head; p != null; p = p.next) {
            Node newNode = cachedNode.get(p);
            newNode.next = cachedNode.get(p.next);
            newNode.random =  cachedNode.get(p.random);
        }
        return cachedNode.get(head);
    }
}
```

复杂度分析

- 时间复杂度：O(n)
- 空间复杂度：O(n)，需要构造一个哈希表来存储所有节点

方法二：迭代法

思路和算法

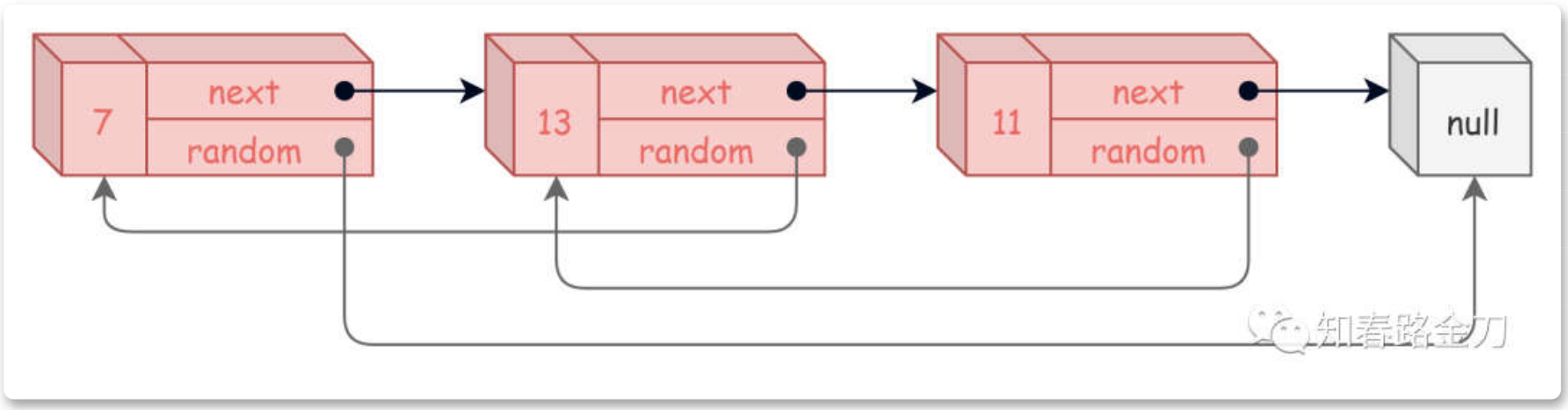
这个思路不会使用额外的存储空间来保存新旧节点的对应关系，而是把新旧节点的对应关系保存在**「原有链表」**之上，最后再对这个链表进行拆分。

- 首先遍历一次链表，每遇到一个旧链表的节点，我们就复制一个新的节点出来，再把这个新的节点插在这个旧链表节点的后面。
  - 这次遍历完成之后，链表的长度会变成原来的两倍。链表中奇数节点为旧链表中的节点，偶数节点为新链表中的节点。
  - 而且我们可以通过旧节点，很容易地找到所对应地新节点，新节点也就是这个旧节点的下一个节点。
- 再次遍历链表，把新节点中的 random 指针指向正确的新节点。
- 最后对整个链表进行拆分。

这个方法的思路理解起来还是有点儿复杂的，同学们可以看下面金刀亲手制作的图解。

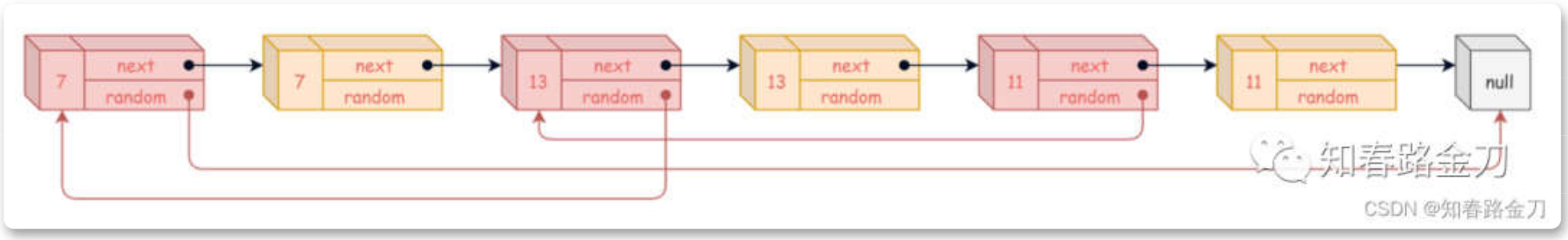
图解算法

「链表初始状态」



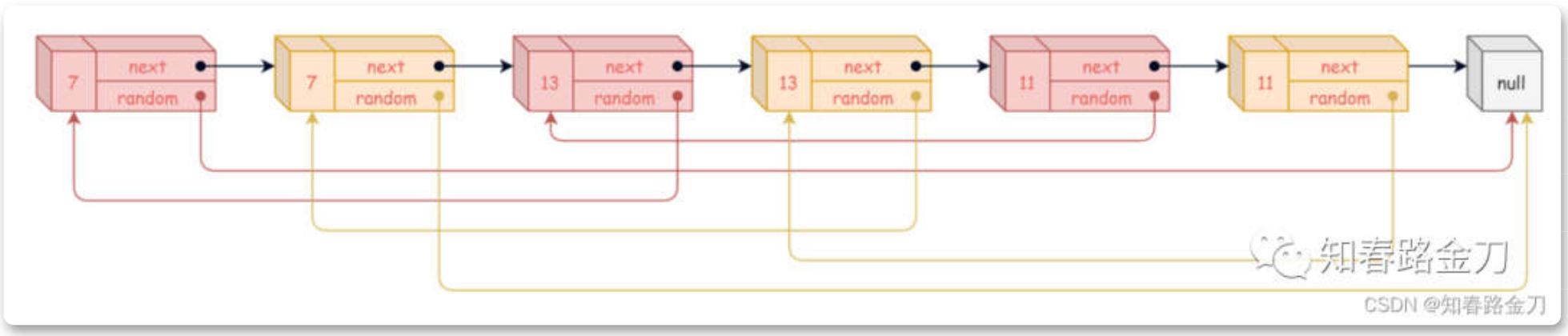
「第一次遍历链表之后的状态」

其中红色节点表示旧链表中的节点，黄色节点表示新链表中的节点。 每个红色节(旧节点)点的下一个节点都是所对应的新链表中的节点。



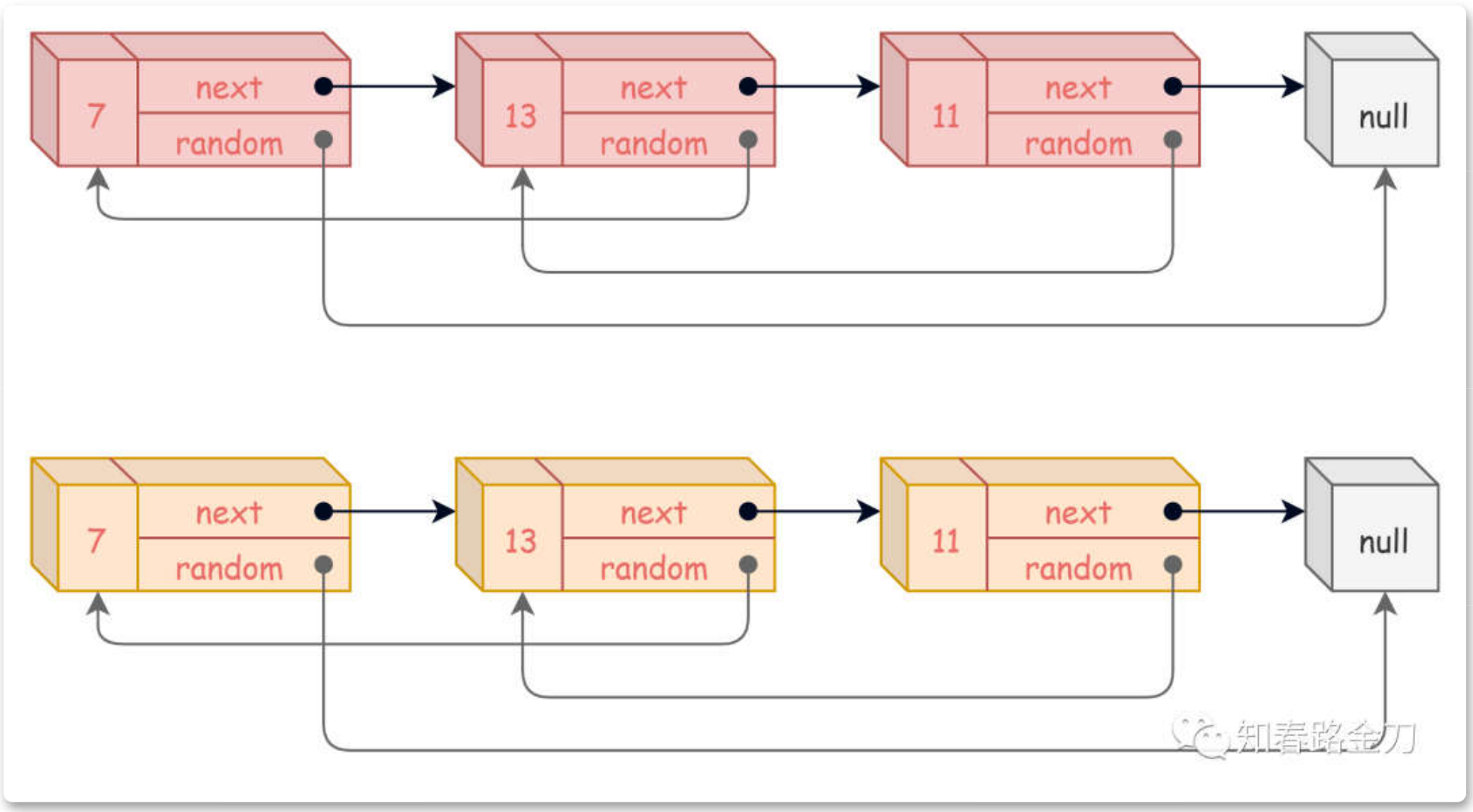
「第二次遍历链表之后的状态」

在这次遍历中，依次把黄色节点的 random 指为新的节点。



「拆分链表」

最后在对链表进行拆分，这样我们就完成了深拷贝。



代码实现

```
class Solution {
    public Node copyRandomList(Node head) {
        if (head == null) {
            return null;
        }
        Node p = head;
        while (p != null) { // 复制链表节点
            Node newNode = new Node(p.val);
            newNode.next = p.next;
            p.next = newNode;
            p = newNode.next;
        }
        p = head;
        while (p != null) { // 修正新节点的random指针
            p.next.random = (p.random == null) ? null : p.random.next;
            p = p.next.next;
        }
        Node dummp = new Node(0);
        Node last = dummp;
        p = head;
        while (p != null) {
            last.next = p.next;
            p.next = p.next.next;
            last = last.next;
            p = p.next;
        }
        return dummp.next;
    }
}
```

复杂度分析

- 时间复杂度：O(n)
- 空间复杂度：O(1)

参考资料

[1] **原题链接:** https://leetcode-cn.com/problems/copy-list-with-random-pointer/