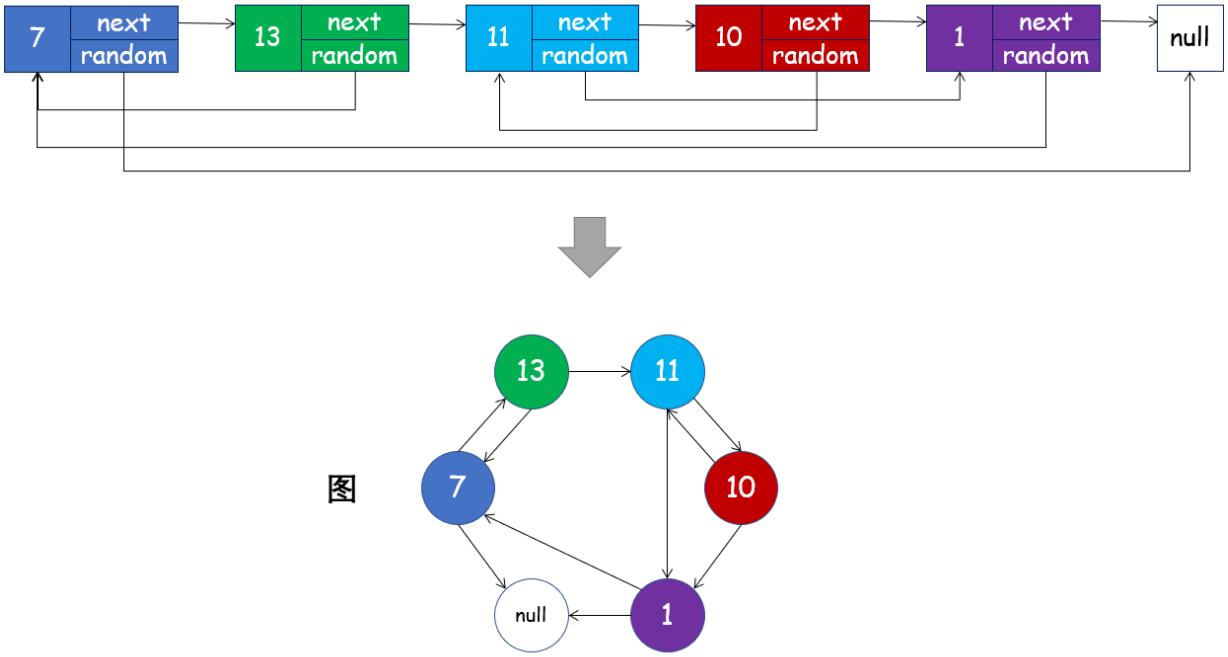
剑指35 复杂链表的复制

hash表想到了，但是花了很久时间，且链表复制操作不熟练(虽然最终优化结果无需链表的复制操作)。

方法一：哈希表 map<原节点地址，新节点地址>

方法二：拼接与拆分 //巧妙方法
　拓展链表，在每个链表结点的旁边拷贝，然后将拷贝的节点分离出来。

方法三/四：图论的DFS/BFS



图

```
// 法一：k神
// 牛逼，第一次遍历只需要new出节点，建立map。第二次遍历直接给2个指针元素赋值。

/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
```

```
};
*/
class Solution {
public:
    Node* copyRandomList(Node* head) {
        if(head == nullptr) return nullptr;
        Node* cur = head;
        unordered_map<Node*, Node*> map;
        // 3. 复制各节点，并建立 “原节点 -> 新节点” 的 Map 映射
        while(cur != nullptr) {
            map[cur] = new Node(cur->val);
            cur = cur->next;
        }
        cur = head;
        // 4. 构建新链表的 next 和 random 指向
        while(cur != nullptr) {
            map[cur]->next = map[cur->next];
            map[cur]->random = map[cur->random];
            cur = cur->next;
        }
        // 5. 返回新链表的头节点
        return map[head];
    }
};


```

```
// 法二 nb
class Solution {
public:
    Node* copyRandomList(Node* head) {
        if(head == nullptr) return nullptr;
        Node* cur = head;
        // 1. 复制各节点，并构建拼接链表
        while(cur != nullptr) {
            Node* tmp = new Node(cur->val);
            tmp->next = cur->next;
            cur->next = tmp;
            cur = tmp->next;
        }
        // 2. 构建各新节点的 random 指向
        cur = head;
        while(cur != nullptr) {
            if(cur->random != nullptr)
                cur->next->random = cur->random->next;
            cur = cur->next->next;
        }
        // 3. 拆分两链表
        cur = head->next;
        Node* pre = head, *res = head->next;
        while(cur->next != nullptr) {
            pre->next = pre->next->next;
            cur->next = cur->next->next;
            pre = pre->next;
            cur = cur->next;
        }
        pre->next = nullptr; // 单独处理原链表尾节点
```

```
        return res;       // 返回新链表头节点
    }
};

```

```
// 法三：DFS还真行
// map保存已经遍历过的。

class Solution {
public:
    unordered_map<Node*, Node*> dic;
    Node* dfs(Node* p) {
        if(p == NULL) return NULL;

        if(dic.find(p) != dic.end()) return dic[p];

        Node* tmp = new Node(p->val);
        dic[p] = tmp;
        tmp->next = dfs(p->next);
        tmp->random = dfs(p->random);
        return tmp;
    }
    Node* copyRandomList(Node* head) {
        return dfs(head);
    }
};
```