面试题35：复杂链表的复制

题目描述

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）。

解题思路

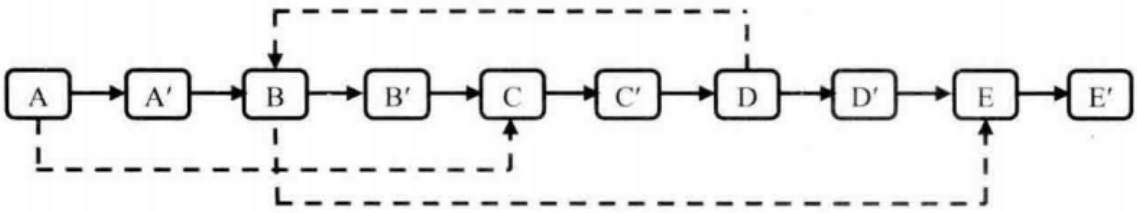

图 4.12　复制复杂链表的第一步

注：复制原始链表的任意节点 N 并创建新节点 N'，再把 N'链接到 N 的后面。

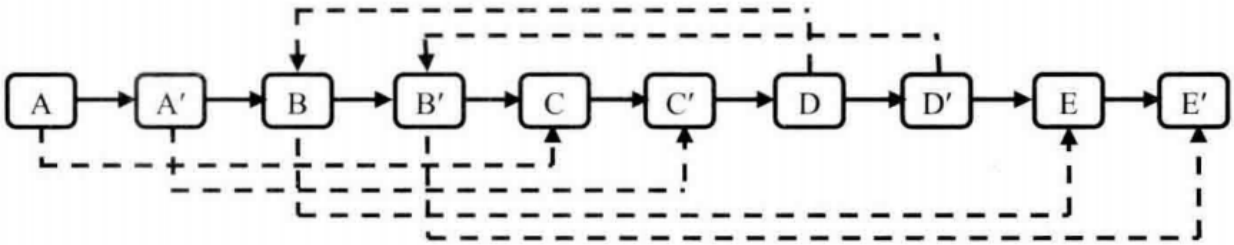

图 4.13　复制复杂链表的第二步

注：如果原始链表上的节点 N 的 m_pSibling 指向 S，则它对应的复制节点 N'的 m_pSibling 指向 S 的复制节点 S'。

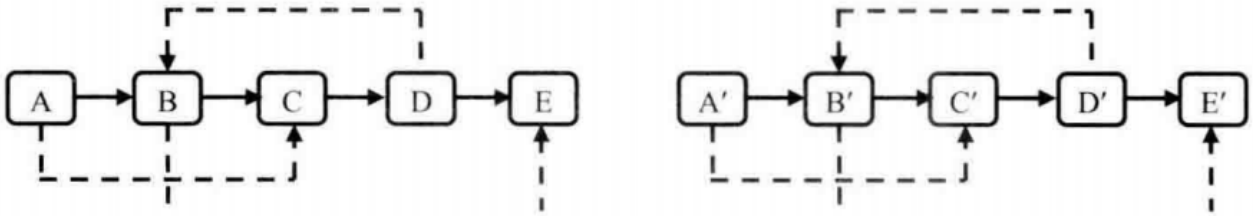

图 4.14　复制复杂链表的第三步

注：把第二步得到的链表拆分成两个链表，奇数位置上的节点组成原始链表，偶数位置上的节点组成复制出来的链表。

C++

```
/*
struct RandomListNode {
    int label;
    struct RandomListNode *next, *random;
```

```
5      RandomListNode(int x) :
6              label(x), next(NULL),
   random(NULL) {
7      }
8  };
9  */
10 class Solution {
11 public:
12     RandomListNode* Clone(RandomListNode*
   pHead)
13     {
14         CloneNode(pHead);
15         CloneRandom(pHead);
16         return ReconnectNode(pHead);
17     }
18     //第一步：根据原始链表的每个节点N创建对应的N',
   N'链接在N的后面
19     void CloneNode(RandomListNode* head){
20         RandomListNode* pNode = head;
21         while(pNode != nullptr){
22             RandomListNode* pClone = new
   RandomListNode(0);
23             pClone -> label = pNode ->
   label;
24             pClone -> next = pNode -> next;
25             //pCloned -> m_pSibling =
   nullptr;
26             pNode -> next = pClone;
```

```
27              pNode = pClone -> next;
28              }
29          }
30      //第二步：设置复制出来的节点的m_pSibling
31      void CloneRandom(RandomListNode* head){
32          RandomListNode* pNode = head;
33          while(pNode != nullptr){
34              RandomListNode* pClone = pNode -
> next;
35              if(pNode -> random != nullptr){
36                  pClone -> random = pNode ->
random -> next;
37              }
38              pNode = pClone -> next;
39          }
40      }
41      //第三步：把这个长链表拆分成两个链表，奇数位置
原始链表，偶数位置复制链表
42      RandomListNode*
ReconnectNode(RandomListNode* head){
43          RandomListNode* pNode = head;
44          RandomListNode* pCloneHead =
nullptr;
45          RandomListNode* pCloneNode =
nullptr;
46          if(pNode != nullptr){
47              pCloneHead = pCloneNode = pNode
-> next;
```

```
            pNode -> next = pCloneNode -> next;
            pNode = pNode -> next;
        }
        while(pNode != nullptr){
            pCloneNode -> next = pNode -> next;
            pCloneNode = pCloneNode -> next;
            pNode -> next = pCloneNode -> next;
            pNode = pNode -> next;
        }
        return pCloneHead;
    }
};
```

Java

```
/*
public class RandomListNode {
    int label;
    RandomListNode next = null;
    RandomListNode random = null;

    RandomListNode(int label) {
        this.label = label;
    }
}
```

```
*/
public class Solution {
    public RandomListNode Clone(RandomListNode pHead)
    {
        if(pHead==null)
            return null;
        RandomListNode pCur = pHead;
        //复制next 如原来是A->B->C 变成A->A'->B->B'->C->C'
        while(pCur!=null){
            RandomListNode node = new RandomListNode(pCur.label);
            node.next = pCur.next;
            pCur.next = node;
            pCur = node.next;
        }
        pCur = pHead;
        //复制random pCur是原来链表的结点 pCur.next是复制pCur的结点
        while(pCur!=null){
            if(pCur.random!=null)
                pCur.next.random = pCur.random.next;
            pCur = pCur.next.next;
        }
        RandomListNode head = pHead.next;
        RandomListNode cur = head;
```

```
        pCur = pHead;
        //拆分链表
        while(pCur!=null){
            pCur.next = pCur.next.next;
            if(cur.next!=null)
                cur.next = cur.next.next;
            cur = cur.next;
            pCur = pCur.next;
        }
        return head;
    }
}
```

Python 2.7.3

```
# -*- coding:utf-8 -*-
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # 返回 RandomListNode
    def Clone(self, pHead):
        # write code here
        #递归法
        if not pHead: return
```

```
        newNode = RandomListNode(pHead.label)
        newNode.random = pHead.random
        newNode.next = self.Clone(pHead.next)
        return newNode
```

```
# -*- coding:utf-8 -*-
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # 返回 RandomListNode
    def Clone(self, head):
        # write code here
        #哈希表法
        nodeList = []      #存放各个节点
        randomList = []    #存放各个节点指向的random节点。没有则为None
        labelList = []     #存放各个节点的值
        while head:
            randomList.append(head.random)
            nodeList.append(head)
            labelList.append(head.label)
            head = head.next
        #random节点的索引，如果没有则为1
```

```
        labelIndexList = map(lambda c: nodeList.index(c) if c else -1, randomList)
        dummy = RandomListNode(0)
        pre = dummy
        #节点列表，只要把这些节点的random设置好，顺序串起来就ok了。
        nodeList=map(lambda c:RandomListNode(c),labelList)
        #把每个节点的random绑定好，根据对应的index来绑定
        for i in range(len(nodeList)):
            if labelIndexList[i]!=-1:

 nodeList[i].random=nodeList[labelIndexList[i]]
        for i in nodeList:
            pre.next=i
            pre=pre.next
        return dummy.next
```