查阅更多的题解，请点击

Problem

138. Copy List with Random Pointer(Medium)

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.

Example 1:

```
Input:
{"$id":"1","next":{"$id":"2","next":null,"random":{"$ref":"2"},"val":2},"random":{"$ref":"2"},'

Explanation:
Node 1's value is 1, both of its next and random pointer points to Node 2.
Node 2's value is 2, its next pointer points to null and its random pointer points to itself.
```

Note: you must return the **copy of the given head** as a reference to the cloned list.

Solution

Solution One

O(n^2) time O(n) space

这道题是实现对链表的深拷贝，实际上，对链表的拷贝并不是一件复杂的事情，该问题的难点在于存在random pointer

对于一般的链表，一次遍历即可以实现对链表的拷贝。由于引入了random pointer，其可以指向任意结点，则在构建新链表的时候不能对所有的random pointer赋值（对于Node a，其random pointer指向的节点可能为a之后）。在新的链表构建完成之后，需要根据原链表中random指向的节点，将新链表中对应random pointer指向新链表中对应的节点，此时需要对指向的节点进行标识，一个直观的策略是depth：即链表的头结点到random pointer指向节点的距离）

由于每次depth需要从head开始遍历，在最坏情况下（小概率事件），原链表中每个节点的random pointer均不为空，指向链表尾，则需要遍历n*n次，所以该算法的时间复杂度为$O(n^2)$

Solution Two

O(n) time O(n) space

在solution one中，已经得出了一个该问题一般性的解（实际上该解已经faster than 99.97% of C++ online submissions），**那么对于一般性的解，我们尝试对其优化以得到更好的解**

观察solution one中的解法，分为两部分：构建新链表和random pointer赋值。第一部分很难去优化，而第二部分的查找node非常耗时，可以尝试优化该部分->对于查找的优化，一般想到用一些高级的数据结构来实现更高的查找效率，如红黑树，hash表等，而在该问题中，对random pointer赋值：

- random pointer指向的为nullptr或是链表的一个node，而已知的是原链表的node，需要查找的为在新链表中的node（注意新链表尾原链表的拷贝，两个链表是一一对应的），这里不妨引入hash table<key=原链表中的node，value=新链表中对应的node>

由于hash table查找效率为O(1)，则对于最耗时的查找操作，我们给出了比较好的优化策略

Note:
实际上由于hash table的创建和hash值的计算都需要一定的时间开销，在链表长度不长的情况下，实际上solution one的运行效率也较好
GitHub传送门

Solution One

```
class Solution
{
    int getDepth(Node *root, Node *node)
    {
        int depth = 0;
        Node *temp = root;
        while (temp != node)
        {
            temp = temp->next;
            depth++;
        }
        return depth;
    }
    Node *getNode(Node *root, int depth)
    {
        Node *node = root;
        while (depth--)
            node = node->next;
        return node;
    }

  public:
    Node *copyRandomList(Node *head)
    {
        if (head == nullptr)
            return nullptr;

        //copy val and next
        Node *root = new Node(head->val, head->next, nullptr);
        Node *node = root;
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
            node->next = new Node(temp->val, temp->next, nullptr);
            node = node->next;
        }

        //copy random pointer
        node = root;
        temp = head;
        while (temp != nullptr)
        {
            node->random = temp->random == nullptr ? nullptr : getNode(root, getDepth(head, temp->random));
            temp = temp->next;
            node = node->next;
        }
        return root;
    }
};
```

Solution Two

```
class Solution
{
  public:
    Node *copyRandomList(Node *head)
    {
        if (head == nullptr)
            return nullptr;

        unordered_map<Node *, Node *> hashTable;
        Node *node = head;
        while (node != nullptr)
        {
            hashTable[node]=new Node(node->val, nullptr, nullptr);
            node = node->next;
        }
        node = head;
        while (node != nullptr)
        {
            hashTable[node]->next = hashTable[node->next];
            if(node->random!=nullptr)
                 hashTable[node]->random = hashTable[node->random];
            node = node->next;
        }
        return hashTable[head];
    }
};
```