【门徒计划】第一周刷题代码



# 链表的访问

## Leetcode-141-环形链表

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (head==NULL) return false;
        ListNode *slow=head, *quick=head->next;
        // while 的结束条件
        // 无环. quick = NULL or quick->next = NULL
        // 有环. slow = quick, slow ,quick !=NULL
        while(quick && quick->next && slow!=quick){
            slow = slow->next;
            quick = quick->next->next;
        }
        return quick && quick->next;
    }
};
```

## Leetcode-142-环形链表II

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        if(head==nullptr)return nullptr;
        if(head->next==nullptr)return nullptr;
        ListNode *slow=head,*quick=head,*other=head;
        do{
            slow = slow->next;
            quick = quick->next->next;
        }while(slow && quick && quick->next && slow != quick);

        if(quick && quick->next){
            while(other!=quick){
                other=other->next;
                quick=quick->next;
            }
            return quick;
        }else{
            return nullptr;
        }
    }
};
```

## Leetcode-202-快乐数

```
class Solution {
public:
    int getNext(int x){
        int ret=0;
        while(x>0){
            int c = x%10;
            ret = ret + c * c;
            x = x/10;
        }
```

```
        return ret;
    }
    bool isHappy(int n) {
        int slow = n;
        int fast = n;
        do{
            slow = getNext(slow);
            fast = getNext(getNext(fast));
        }while(slow!=fast && fast != 1);
        return fast==1;
    }
};
```

# 链表的反转

## Leetcode-206-反转链表

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(head == nullptr || head->next == nullptr) return head;
        ListNode* tail =head->next;
        ListNode* ret=reverseList(head->next);
        head->next = tail->next;
        tail->next = head;
        return ret;
    }
};
```

## Leetcode-92-反转链表II

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseN(ListNode* head, int n){
        if(n==1) return head;
        ListNode *tail = head->next;
        ListNode *ret = reverseN(head->next, n-1);
        head->next = tail->next;
        tail->next = head;
        return ret;
    }
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        ListNode ret(0,head);
        ListNode *p=&ret;
        int cnt=n-m+1;
        while(--m)p=p->next;
        p->next = reverseN(p->next, cnt);
        return ret.next;
    }
};
```

## Leetcode-25-K个一组翻转链表

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
```

```
        ListNode* _reserseN_(ListNode* head, int n){
            if(n==1) return head;
            ListNode *tail = head->next;
            ListNode *ret = _reserseN_(head->next, n-1);
            head->next = tail->next;
            tail->next = head;
            return ret;
        }
        ListNode* reserseN(ListNode * head, int n){
            ListNode *p = head;
            int cnt = n;
            while(--n && p){
                p = p->next;
            }
            if(p==nullptr) return head;
            return _reserseN_(head,cnt);
        }
        ListNode* reverseKGroup(ListNode* head, int k) {
            ListNode hair(0,head);
            ListNode *p=&hair;
            ListNode *q=p->next;
            while( (p->next = reserseN(q,k)) != q){
                p = q;
                q = p->next;
            }
            return hair.next;
        }
};
```

## Leetcode-61-旋转链表

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(head==nullptr)return nullptr;
        int n=1;
```

```
        ListNode* p=head;
        while(p->next){
            p=p->next;
            n=n+1;
        }
        p->next = head;
        k = k%n;
        k = n-k;
        while(k--){
            p=p->next;
        }
        head=p->next;
        p->next = nullptr;
        return head;
    }
};
```

## 链表的删除操作

### Leetcode-19-删除链表的倒数第 N 个节点

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode hair(0,head);
        ListNode *p=&hair;
        ListNode *hand=head;
        while(n--)hand=hand->next;
        while(hand){
            p=p->next;
            hand=hand->next;
        }
        p->next = p->next->next;
        return hair.next;
```

```
    }
};
```

## Leetcode-83-删除排序链表中的重复元素

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head==nullptr)return nullptr;
        ListNode *fast=head;
        while(fast->next){
            if(fast->val==fast->next->val){
                fast->next = fast->next->next;
            }else{
                fast=fast->next;
            }
        }
        return head;
    }
};
```

## Leetcode-82-删除排序链表中的重复元素II

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
```

```
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode hair(0,head);
        ListNode *slow=&hair;
        ListNode *fast;
        while(slow->next){
            if (slow->next->next &&
                slow->next->val == slow->next->next->val){
                    fast = slow->next->next;
                    while(fast && fast->val == slow->next->val){
                        fast=fast->next;
                    }
                    slow->next=fast;
            }else{
                slow = slow->next;
            }
        }
        return hair.next;
    }
};
```