**1 Problem: Reverse Linked List**

这个题目要注意边界条件，特别是只有一个节点或者没有节点的时候。用三个指针可以解出

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

struct ListNode \*last,\*current,\*next,\*tmp;

last=nullptr;

current=head;

if(head==nullptr) return head;

next=current->next;

while(next!=nullptr){

current->next=last;

tmp=next->next;

next->next=current;

last=current;

current=next;

next=tmp;

}

current->next=last;

return current;

}

};

**2 Problem: Remove Nth Node from End of List**

这个题目要注意如果删除的节点是头结点要做特殊处理，用两个指针。

class Solution {

public:

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

int count=1;

ListNode \*tmp=head,\*last;

while(tmp->next!=nullptr){

tmp=tmp->next;

count++;

}

if(n>count) return nullptr;

tmp=head;

last=head;

for(int i=0;i<count-n;i++){

last=tmp;

tmp=tmp->next;

}

if(tmp==head) head=head->next;

else{

last->next=tmp->next;

}

free(tmp);

return head;

}

};

**3 Problem: Remove Linked List Elements**

要考虑删除的元素是头结点的情况，而且可以一直是头结点。优化方面可以考虑如果被删除的元素连在一起时可以先不断开链，找到不是相同元素的时候在断开。

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode \*last,\*tmp;

last=head;

tmp=head;

while(tmp!=nullptr){

if(tmp->val==val){

if(head==tmp) head=head->next;

last->next=tmp->next;

free(tmp);

tmp=last->next;

}

else{

last=tmp;

tmp=tmp->next;

}

}

return head;

}

};

**4** Problem: Remove Duplicates from Sorted List

这个题不存在投指针删除的情况，可以用一个值保留上一个链表的值，如果不一样就继续，一样就删除当前的，考虑到优化，如果一样先保留，碰到不一样是发现一样的计数大于1就删除，否则继续。

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

if(head==nullptr) return head;

struct ListNode \*last,\*next;

int current=head->val;

next=head->next;

last=head;

while(next!=nullptr){

if(next->val==current){//remove

last->next=next->next;

free(next);

next=last->next;

}

else{

current=next->val;

last=next;

next=next->next;

}

}

return head;

}

};

**5 Problem: Merge Two Sorted Lists**

用两个指针，最后把结果放到第一个里面，如果第一个指针大于第二个，第二个往下走，否则第一个往下走，如果有一个指针为空，发现是第二个直接返回，发现是第一个需要把第二个的剩下的接过去。

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

ListNode \*head, \*last;

if(l1==nullptr) return l2;

if(l2==nullptr) return l1;

if(l1->val<l2->val){

head=l1;

last=head;

l1=l1->next;

}

else{

head=l2;

last=head;

l2=l2->next;

}

while(l1!=nullptr&&l2!=nullptr){

if(l1->val<l2->val){

last->next=l1;

last=l1;

l1=l1->next;

}

else{

last->next=l2;

last=l2;

l2=l2->next;

}

}

while(l1!=nullptr){

last->next=l1;

last=l1;

l1=l1->next;

}

while(l2!=nullptr){

last->next=l2;

last=l2;

l2=l2->next;

}

return head;

}

};

**6** Problem: Intersection of Two Linked Lists

这个题开始想了各种办法没思路，后面看了一下网友的提示豁然开朗，求两个链表长度差，长的一个走那么多然后两个一起走，如果相遇返回，到了末端返回空。

class Solution {

public:

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

ListNode \*tmp1,\*tmp2;

int count1,count2;

count1=count2=0;

tmp1=headA;tmp2=headB;

while(tmp1!=nullptr){

tmp1=tmp1->next;

count1++;

}

while(tmp2!=nullptr){

tmp2=tmp2->next;

count2++;

}

tmp1=headA;tmp2=headB;

if(count1>count2){

int value=count1-count2;

for(int i=0;i<value;i++) tmp1=tmp1->next;

}

else{

int value=count2-count1;

for(int i=0;i<value;i++) tmp2=tmp2->next;

}

while(true){

if(tmp1==tmp2) return tmp1;

if(tmp1==nullptr) return nullptr;

tmp1=tmp1->next;

tmp2=tmp2->next;

}

}

};

**7 Problem: Add Two Numbers**

比较繁琐，要考虑进位的问题，开始就一直两个指针加，一个没了要把另外一个加完，但容易错在最后加完了所有指针一定要考进位是否为一（两个数加不可能为二），如果是还要新建一个为一的节点。

class Solution {

public:

ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {

if(l1==nullptr) return l2;

if(l2==nullptr) return l1;

int in=0;

ListNode \*next1,\*next2,\*head,\*next3;

next1=l1;next2=l2;

head=new ListNode(0);

next3=head;

while(next1!=nullptr&&next2!=nullptr){

next3->val=next1->val+next2->val+in;

in=0;

if(next3->val>=10){

in=1;

next3->val-=10;

}

next3->next=new ListNode(0);

next1=next1->next;

next2=next2->next;

next3=next3->next;

}

ListNode \*l;

if(next1!=nullptr) l=next1;

else l=next2;

while(l!=nullptr){

next3->val=l->val+in;

in=0;

if(next3->val>=10){

in=1;

next3->val-=10;

}

next3->next=new ListNode(0);

l=l->next;

next3=next3->next;

}

if(in==1){

next3->val=1;

next3->next=nullptr;

}

else{

ListNode \*tmp=head;

while(tmp->next!=next3) tmp=tmp->next;

free(next3);

tmp->next=nullptr;

}

return head;

}

};

### 8 Problem: Swap Nodes in Pairs

用四个指针，两个保存上一个链接和链接到下一个的，然后中间两个互换以后用另外两个连起来。要考虑为空和只有一个的情况。

class Solution {

public:

ListNode\* swapPairs(ListNode\* head) {

ListNode \*first,\*second,\*tmp,\*last;

first=head;

if(first==nullptr) return head;

second=first->next;

if(second==nullptr) return head;

head=second;

last=first;

while(second!=nullptr){

last->next=second;

tmp=second->next;

second->next=first;

first->next=tmp;

last=first;

first=tmp;

if(first==nullptr) return head;

second=first->next;

}

return head;

}

};

**9** Problem: Rotate List

第一次做没有考虑K大于等于链表长度的情况，其实只要取摩就好。然后把收尾链接起来，找到旋转的哪个元素，再把它和上一个断开就行了。（和写的方法不同，带我觉得前者更简单）

class Solution {

public:

ListNode\* rotateRight(ListNode\* head, int k) {

ListNode \*tmp=head,\*current;

if(head==nullptr) return nullptr;

int count=0;

while(tmp!=nullptr){

tmp=tmp->next;

count++;

}

k%=count;

tmp=head;

for(int i=0;i<count-k-1;i++) tmp=tmp->next;

current=tmp->next;

tmp->next=nullptr;

tmp=current;

if(current==nullptr) return head;

while(current->next!=nullptr) current=current->next;

current->next=head;

head=tmp;

return head;

}

};

**10 Problem: Sort List**

代码时间复杂度过不了，让我看看其他人的大作在修改。不过思路应该是用快排的方法做，我应该是用的快排的方法，但不知道为什么过不了大数据测试。

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

if(head==nullptr) return nullptr;

return quicksort(nullptr,head,nullptr);

}

ListNode \*quicksort(ListNode \*last,ListNode \*begin,ListNode \*next){

if(begin!=next){

ListNode \*head=quickpass(begin,last,next);

quicksort(head,head->next,next);

if(last!=nullptr) return quicksort(last,last->next,head);

else return quicksort(last,begin,head);

}

else return last;

}

ListNode\* quickpass(ListNode \*head,ListNode \*last,ListNode \*next){

ListNode \*sHead,\*lHead,\*key,\*current,\*lCurrent,\*sCurrent;

sHead=lHead=nullptr;

lCurrent=sCurrent=nullptr;

key=head;

current=head->next;

while(current!=next){

if(current->val>=key->val){

if(lHead==nullptr){

lHead=current;

lCurrent=current;

}

else{

lCurrent->next=current;

lCurrent=lCurrent->next;

}

}

else{

if(sHead==nullptr){

sHead=current;

sCurrent=current;

}

else{

sCurrent->next=current;

sCurrent=sCurrent->next;

}

}

current=current->next;

}

if(sCurrent!=nullptr){

sCurrent->next=key;

if(last!=nullptr) last->next=sHead;

}

else if(last!=nullptr) last->next=key;

if(lHead!=nullptr){

key->next=lHead;

lCurrent->next=next;

}

else key->next=next;

return key;

}

};