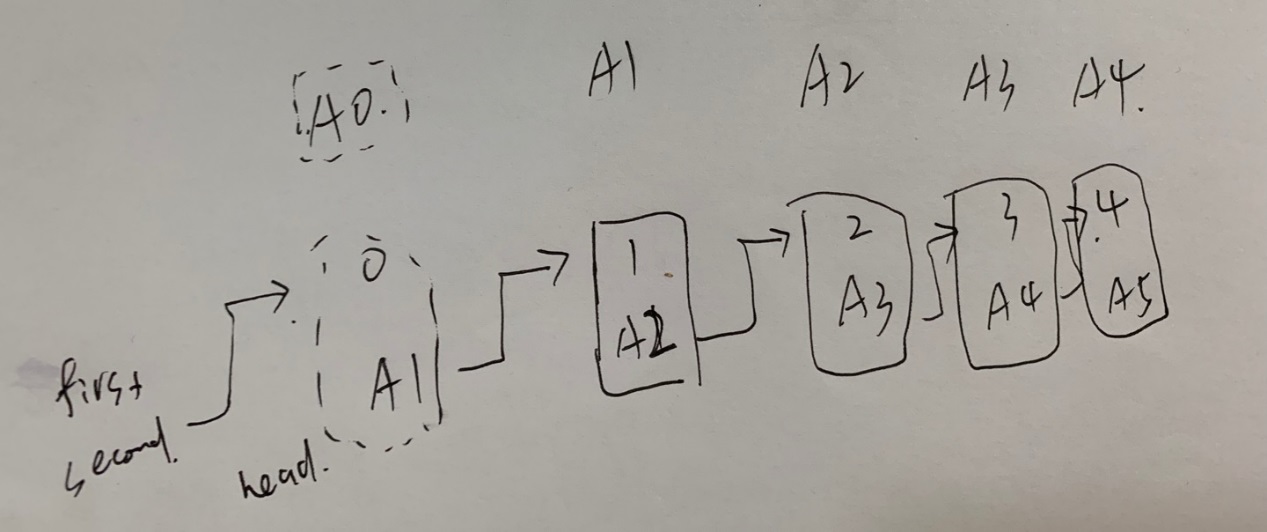
#19

思路就是有两个指针，让第二个指针上前移动n步

链表梳理



ListNode \*head是指head是一个指针，指到一个ListNode

指针利用地址指到值，引用从值查地址。

Head是一个指针。head ->next指到的地方的next

newhead是一个listnode newhead.next 该节点的next

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

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

ListNode\* first = head;

ListNode\* second = head;

for (int i = 1; i <= n; i++){

second = second->next;

}

while (second->next){

first = first->next;

second = second->next;

}

first->next = first->next->next;

return head;

}

};

这样写没有增加新节点，但经常timeout而且不适合特殊情况。为特殊情况考虑，在前面再加一个节点即可包含所有特殊情况。如在head加上val，和head一起组成一个新节点。然后\*first = &newhead的意思是，让newhead的地址赋值给first。First指向newhead的地址。First是一个指针。First->next是head。

static int x = [](){ios::sync\_with\_stdio(false); cin.tie(nullptr); return 0; }();

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

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

ListNode newhead (0);

newhead.next = head;

ListNode\* first = &newhead;

ListNode\* second = &newhead;

for (int i = 1; i <= n; i++){

second = second->next;

}

while (second->next){

first = first->next;

second = second->next;

}

first->next = first->next->next;

return newhead.next;

}

};