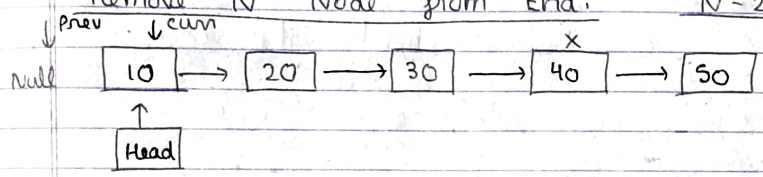
Date: / /
Page:

Day - 116

Linked List - 4

* **Remove N^{th} Node from End;** $N = 2$



⇒ First, we count the nodes then minus the value of k from it.

=1 After that again traverse the LL and reduce the value of count by 1.

=1 when the value becomes 0 then remove that current node.

=1 So, we require two pointers curr & prev.

Code

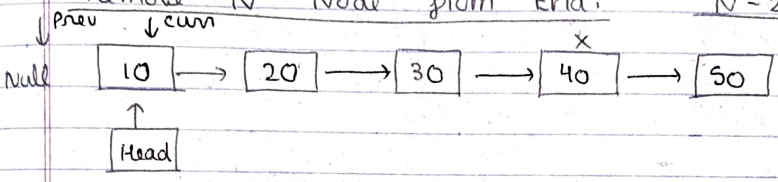
```
int count = 0;
ListNode * temp = head;
while ( temp != NULL) {
    count++;
    temp = temp -> next;
}
count -= N;
List Node * curr = head, * prev = NULL;
while( count--){
    prev = curr;
    curr = curr -> next;
}
prev -> next = curr -> next;
delete curr;
```

Date: / /
Page:

Day - 116

Linked List - 4

* **Remove N^{th} Node from End;** $N = 2$



⇒ First, we count the nodes then minus the value of k from it.

⇒ After that again traverse the LL and reduce the value of count by 1.

⇒ when the value becomes 0 then remove that current node.

⇒ So, we require two pointers curr & prev.

Code

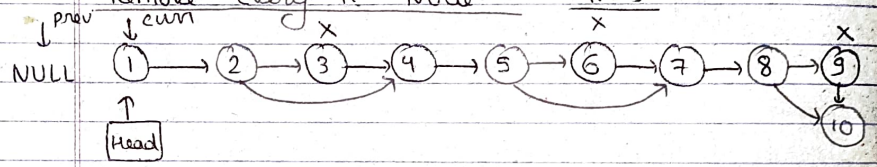
```
int count = 0;
ListNode * temp = head;
while ( temp != NULL) {
    count++;
    temp = temp -> next;
}
count -= N;
List Node * curr = head, * prev = NULL;
while( count--){
    prev = curr;
    curr = curr -> next;
}
prev -> next = curr -> next;
delete curr;
```

Date: / /
Page:

There will be a edge case here when N = 5,
Because our code will not work for N = 5,
Because ~~prev~~ next of prev will not exist,

So, we also have to handle this case.

* Remove every k^th Node : k = 3



So, in this question, we also require two pointers — prev & curr.

Code

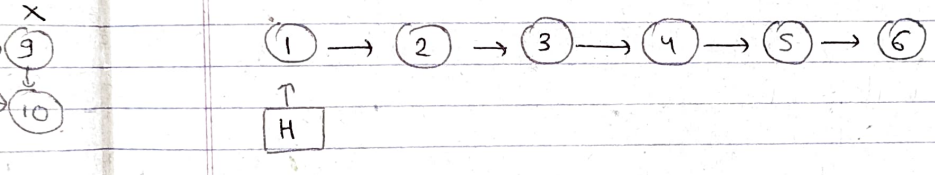
```
Node *curr = head, *prev = NULL;
int count = 1;
while(curr){
    if(k == count){
        prev->next = curr->next;
        delete curr;
        curr = prev->next;
        count = 1;
    } else{
        prev = curr;
        curr = curr->next;
        count++;
    }
}
return head;
```

Date: / /
Page:

⇒ When the value of k = 1, then this is a edge case —

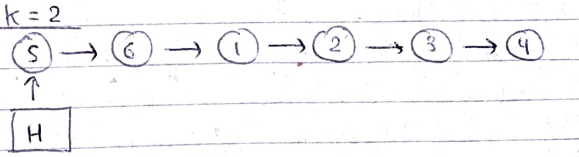
⇒ if(k == 1)
return & NULL;

* Rotate List; k = 2



⇒ Rotate the List k times i.e. —

k = 2



⇒ Our first task is to reach that k node from last.

⇒ For that, first we have to count the nodes then simply minus the value of k then traverse to that count value.

⇒ Then, we will break the bond by using curr & prev pointers.

⇒ We will ~~do~~ make prev's next to NULL.

⇒ After that traverse to the end of LL by tail pointer.

⇒ Then connect tail with first pointer & make head equal to curr.

⇒ And return head.

Date: / /
Page:

Code

```
int count = 0;
ListNode *temp = head;
while(temp){
    count++;
    temp = temp -> next;
}; k = k % count;  // For edge case no. 4.
                   → if(k==0) return head;
count -= k;
ListNode * curr = head, *prev = NULL;
while (count--){
    prev = curr;
    curr = curr -> next;
}
prev -> next = NULL;
ListNode * tail = curr;
while( tail -> next != NULL){
    tail = tail -> next;
}
tail -> next = head;
head = curr;
```

⇒ Edgecases for this question are →
if head is NULL then return head.
if k = 1 then return head.
if k = 0 then return head.
⇒ And if the value of k is more than no. of nodes then —
Rotate the list by 'k % count' times.

Date: / /
Page:

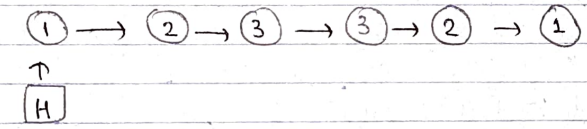
⇒ We can also solve it by another method —

⇒ First we traverse the node ~~then~~ and store all the values in a array.

⇒ Then, the k values from end, we write in the starting. And the remaining values we write in the remaining linked list after starting.

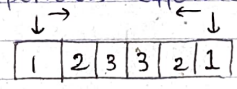
⇒ But, it's also possible that this solution don't pass on the Leetcode or GFG.

* Check Palindrome:



⇒ First, we traverse the array and store all the elements in the array.

⇒ After that, by using two pointers approach, check for palindrome.

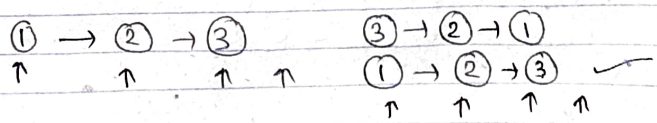


⇒ Here, we are using extra space.

⇒ But, we want to solve it without using extra space.

⇒ Now, first we break it into two pieces then reverse the second part.

⇒ And then check both the parts are same or not.



⇒ Then Palindrome.

Date: / /
Page:

⇒

```
ListNode *temp = head;
int count = 0;
while(temp){
    count++;
    temp = temp -> next;
}
count /= 2;
ListNode *curr = head, *prev = NULL;
while(count--){
    prev = curr;
    curr = curr -> next;
}
prev -> next = NULL;
prev = NULL;
ListNode * front = NULL;
while(curr){
    front = curr -> next;
    curr -> next = prev;
    prev = curr;
    curr = front;
}

ListNode *head1 = head, *head2 = prev;
while(head1){
    if(head1 -> val != head2 -> val){
        return 0;
    }
    head1 = head1 -> next;
    head2 = head2 -> next;
}
return 1;
```