Lecture 27 LinkedList

1. <https://leetcode.com/problems/remove-nth-node-from-end-of-list/>

class Solution {

public:

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

ListNode \*dummy = new ListNode();

dummy->next = head;

ListNode \*slow = dummy;

ListNode \*fast = dummy;

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

{

fast=fast->next;

}

if(fast==NULL)

{

return head->next;

}

while(fast->next!=NULL)

{

fast = fast->next;

slow = slow->next;

}

slow->next = slow->next->next;

return dummy->next;

}

};

1. <https://leetcode.com/problems/odd-even-linked-list/>

class Solution {

public:

ListNode\* oddEvenList(ListNode\* head) {

if(head==NULL)

{

return head;

}

ListNode \*oddHead = head;

ListNode \*odd = head;

ListNode \*evenHead = head->next;

ListNode \*even = head->next;

while(even!=NULL && even->next!=NULL)

{

odd->next = odd->next->next;

even->next = even->next->next;

odd = odd->next;

even = even->next;

}

odd->next = evenHead;

return oddHead;

}

};

1. <https://practice.geeksforgeeks.org/problems/reverse-a-linked-list-in-groups-of-given-size/1#>

struct node \*reverse (struct node \*head, int k)

{

// Complete this method

node \*p=NULL;

node \*c=head;

node \*n;

int count = k;

while(count-- && c!=NULL)

{

n=c->next;

c->next=p;

p=c;

c=n;

}

if(n!=NULL)

{

head->next = reverse(n, k);

}

return p;

}

1. <https://leetcode.com/problems/reorder-list/>

class Solution {

ListNode \*reverse(ListNode \*head)

{

ListNode \*p =NULL;

ListNode \*c = head;

ListNode \*n;

while(c!=NULL)

{

n=c->next;

c->next=p;

p=c;

c=n;

}

return p;

}

void merge(ListNode \*head1, ListNode \*head2)

{

while(head1!=NULL)

{

ListNode \*a = head1->next;

ListNode \*b = head2->next;

head1->next = head2;

if(a==NULL)

{

break;

}

head2->next=a;

head1=a;

head2=b;

}

}

public:

void reorderList(ListNode\* head) {

if(head==NULL || head->next==NULL)

{

return;

}

ListNode \*head1 = head;

ListNode \*slow = head;

ListNode \*fast = head;

ListNode \*prev;

while(fast!=NULL && fast->next!=NULL)

{

prev = slow;

fast = fast->next->next;

slow = slow->next;

}

prev->next = NULL;

ListNode\* head2 = reverse(slow);

merge(head1, head2);

}

};