**Design Linked List**

class MyLinkedList {

Node head;

class Node{

int data;

Node next;

Node(int data){

this.data=data;

}

}

/\*\* Initialize your data structure here. \*/

public MyLinkedList() {

}

/\*\* Get the value of the index-th node in the linked list. If the index is invalid, return -1. \*/

public int get(int index) {

if(head==null){

return -1;

}

int length=0;

Node current=head;

while(current!=null){

current=current.next;

length++;

}

if(index>=length){

return -1;

}

current=head;

int i=0;

while(i<index){

current=current.next;

i++;

}

return current.data;

}

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

public void addAtHead(int val) {

if(head==null){

head=new Node(val);

return;

}

Node current=new Node(val);

current.next=head;

head=current;

}

/\*\* Append a node of value val to the last element of the linked list. \*/

public void addAtTail(int val) {

if(head==null){

head=new Node(val);

return;

}

Node current=head;

while(current.next!=null){

current=current.next;

}

current.next=new Node(val);

}

/\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/

public void addAtIndex(int index, int val) {

if(index==0){

Node add=new Node(val);

add.next=head;

head=add;

return;

}

int length=0;

Node current=head;

while(current!=null){

current=current.next;

length++;

}

if(index>length){

return;

}

current=head;

int i=0;

while(i<index-1){

current=current.next;

i++;

}

Node add=new Node(val);

add.next=current.next;

current.next=add;

}

/\*\* Delete the index-th node in the linked list, if the index is valid. \*/

public void deleteAtIndex(int index) {

if(index==0){

if(head.next==null){

head=null;

}

else

head=head.next;

return;

}

int length=0;

Node current=head;

while(current!=null){

current=current.next;

length++;

}

if(index>=length)

return;

current=head;

int i=0;

while(i<index-1){

current=current.next;

i++;

}

Node del=current.next;

current.next=del.next;

}

}

/\*\*

\* Your MyLinkedList object will be instantiated and called as such:

\* MyLinkedList obj = new MyLinkedList();

\* int param\_1 = obj.get(index);

\* obj.addAtHead(val);

\* obj.addAtTail(val);

\* obj.addAtIndex(index,val);

\* obj.deleteAtIndex(index);

\*/

**Linked List Cycle**

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public boolean hasCycle(ListNode head) {

if(head==null)

return false;

ListNode slow=head;

ListNode fast=head.next;

while(true){

if(fast==null || fast.next==null){

return false;

}

if(fast==slow){

return true;

}

slow=slow.next;

fast=fast.next.next;

}

}

}

Learning:

Taught me two pointer technique: keeping a slow pointer(1 step) and fast pointer(2 steps), once they meet we know that a cycle is formed.

**Linked List Cycle II(Node where cycle starts)**

/\*\*

\* Definition for singly-linked list.

\* class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public ListNode detectCycle(ListNode head) {

if(head==null)

return null;

ListNode current=head;

ListNode slow=head;

ListNode fast=head.next;

while(true){

if(fast==null || fast.next==null){

return null;

}

if(fast==slow){

current=current.next;

}

if(fast==current || fast.next==current)

return current;

slow=slow.next;

fast=fast.next.next;

}

}

}

**Remove Nth Node From End of List**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

int i=1;

ListNode slow=head;

ListNode fast=head;

while(i<n){

fast=fast.next;

i++;

}

if(fast.next==null){

head=head.next;

}

else{

while(fast.next.next!=null){

slow=slow.next;

fast=fast.next;

}

slow.next=slow.next.next;

} return head;

}

}

**Intersection of Two Linked Lists**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) {

\* val = x;

\* next = null;

\* }

\* }

\*/

public class Solution {

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

int smal=0;

int larg=0;

ListNode small=headA;

ListNode large=headB;

if(small==null || large ==null)

return null;

if(large==small)

return large;

while(small!=null){

small=small.next;

smal++;

}

while(large!=null){

large=large.next;

larg++;

}

if(smal<larg){

small=headA;

large=headB;

}

else{

small=headB;

large=headA;

}

int i=1;

System.out.println(larg-smal);

while(i<=Math.abs(larg-smal)){

large=large.next;

i++;

}

while(true){

if(large==null)

return null;

if(large==small)

return small;

System.out.println(small.val+" "+large.val);

small=small.next;

large=large.next;

}

}

}

**Reverse Linked List**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode reverseList(ListNode head) {

if(head==null || head.next==null)

return head;

ListNode current=head.next;

ListNode prev=head;

while(current!=null){

prev.next=current.next;

current.next=head;

head=current;

current=prev.next;

}

return head;

}

}

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

class Solution {

public ListNode removeElements(ListNode head, int val) {

if(head==null){

return null;

}

else if(head.next==null && head.val==val){

return null;

}

else{

ListNode current=head.next;

ListNode prev=head;

while(current!=null){

if(head.val==val){

head=prev.next;

prev=head;

current=head.next;

}

else if(current.val==val){

prev.next=current.next;

current=current.next;

}

else{

current=current.next;

prev=prev.next;

}

}

current =head;

while(current!=null){

System.out.print(current.val+" ");

current=current.next;

}

if(head.next==null && head.val==val){

return null;

}

else

return head;

}

}

}

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode oddEvenList(ListNode head) {

if(head==null || head.next==null)

return head;

ListNode odd=head;

ListNode even\_head=head.next;

ListNode even=even\_head;

ListNode current=even\_head;

int i=2;

while(current!=null){

if(i%2!=0){

even.next=current.next;

odd.next=current;

current.next=even\_head;

odd=odd.next;

current=even;

}

else{

even=current;

}

i++;

current=current.next;

}

return head;

}

}