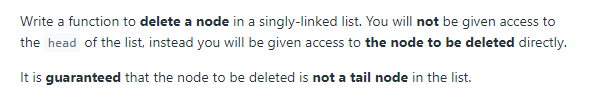
**Lab 5**

Solve these problems for the linked list.

Deadline: Week 7

<https://leetcode.com/problems/delete-node-in-a-linked-list/>



class Solution {

public void deleteNode(ListNode node) {

node.val = node.next.val;

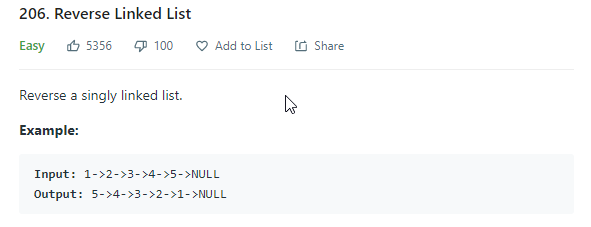
node.next = node.next.next;

}

}

Node value will be equal to the next node value also node next will be next next node. In that way we will be delete the node.

<https://leetcode.com/problems/reverse-linked-list/>



class Solution {

public ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode cur = head;

while(cur != null){

ListNode nextTemp = cur.next;

cur.next = prev;

prev = cur;

cur = nextTemp;

}

return prev;

}

}

Previous node equal to the null, because current node starts from head. The system iterates the until the current node will nor equal to the null. Also nextTemp node will equal to the next of the current. Replace the values of current next to the previous, previous to the current, current to the nextTemp. In the end, the system will show reverse order linkedList.

https://leetcode.com/problems/merge-two-sorted-lists/



class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode finalList = new ListNode(0);

ListNode temp = finalList;

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

if(l1.val <= l2.val){

temp.next = l1;

l1 = l1.next;

}else{

temp.next = l2;

l2 = l2.next;

}

temp = temp.next;

}

if(l1 != null) temp.next = l1;

if(l2 != null) temp.next = l2;

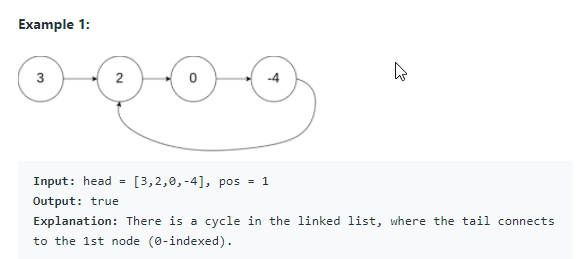
return finalList.next;

}

}

In This task we should to sort using merge sort. Firstly, created new list finalList and temp equal to the finallist first element. It’s iterate until the both of the lists not equal to the null. Then compare two values of each lists. If list 1 less than l 2 , temp next value will equal to the l1, else temp next will equal to the to the l2. Also temp will equal to the temp next element. In the end, the program will compare if list 1 and list 2 with null, if it is correct, it temp next value equal to the l1 and 12.

<https://leetcode.com/problems/linked-list-cycle/>



public class Solution {

public boolean hasCycle(ListNode head) {

Set<ListNode> nodeSeen = new HashSet<>();

while(head != null){

if(nodeSeen.contains(head))

return true;

else

nodeSeen.add(head);

head = head.next;

}

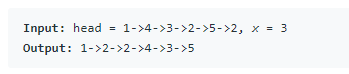
return false;

}

}

In this task we should determine has a cycle or not. Create the set nodeSeen. The iterate until head equal to the null. If nodeSeen contains head it show true, else it add to the set. Otherwise it shows false.

<https://leetcode.com/problems/partition-list/>



class Solution {

public ListNode partition(ListNode head, int x) {

ListNode before\_head = new ListNode(0);

ListNode before = before\_head;

ListNode after\_head = new ListNode(0);

ListNode after = after\_head;

while(head != null){

if(head.val < x){

before.next = head;

before = before.next;

}else{

after.next = head;

after = after.next;

}

head = head.next;

}

after.next = null;

before.next = after\_head.next;

return before\_head.next;

}

}

Before and after are the two pointers used to create the two list, before\_head and after\_head are used to save the heads of the two lists. All of these are initialized with the dummy nodes created.

If the original list node is lesser than the given x, assign it to the before list.

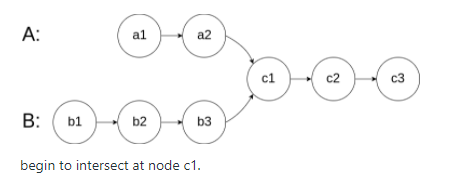
If the original list node is greater or equal to the given x, assign it to the after list.

Then move ahead in the original list

Last node of "after" list would also be ending node of the reformed list

Once all the nodes are correctly assigned to the two lists, combine them to form a single list which would be returned.

<https://leetcode.com/problems/intersection-of-two-linked-lists/>



public class Solution {

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

Set<ListNode> nodes = new HashSet<>();

ListNode pa = headA;

ListNode pb = headB;

while(pa != null){

nodes.add(pa);

pa = pa.next;

}

if(nodes.isEmpty())

return null;

while(pb != null){

if(nodes.contains(pb))

return pb;

pb = pb.next;

}

return null;

}

}

Traverse list A and store the reference to each node in a hash set. Then check every node b in list B: if b contains in the hash set, then b is the intersection node.

<https://leetcode.com/problems/palindrome-linked-list/>

class Solution {

public boolean isPalindrome(ListNode head) {

if(head==null || head.next==null)return true;

ListNode slow=head, fast=head.next;

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

fast=fast.next.next;

slow = slow.next;

}

ListNode sec = slow.next;

slow.next=null;

sec=reverse(sec,null);

while(sec!=null){

if(sec.val!=head.val)return false;

sec=sec.next;

head=head.next;

}

return true;

}

private ListNode reverse(ListNode curr , ListNode prev){

ListNode future = curr.next;

curr.next = prev;

if(future == null) return curr;

return reverse(future , curr);

}

}

Firstly, find middle node. Then reversing the second half of list. Then check first half and second half lists are same or not.