

Laboratory work report

Submitted by:

Madiyar Zhumabekov

IS-1812K

Submitted to:

Yeskendir Sultanov

**Lab 5**

Solve these problems for the linked list.

Deadline: Week 7

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

**Task:** Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list, instead you will be given access to **the node to be deleted** directly.

It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

**Answer:** Just change the node value to the node value after the required node.

**My code:**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

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

\* }

\*/

class Solution {

public void deleteNode(ListNode node) {

node.val = node.next.val;

node.next = node.next.next;

}

}

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

**Task:** Reverse a singly linked list.

**Answer:** While traversing the list, change the next pointer of the current node to point to the previous element. Since the node doesn't have a reference to its previous node, I pre-save its previous element. Used another pointer to store the next node before changing the link.

**My code:**

/\*\*

\* 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) {

ListNode prevNode = null;

ListNode nextNode = null;

while(head != null){

nextNode = head.next;

head.next = prevNode;

prevNode = head;

head = nextNode;

}

return prevNode;

}

}

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

**Task:** Merge two sorted linked lists and return it as a new **sorted** list. The new list should be made by splicing together the nodes of the first two lists.

**Answer:** Combines 2 Listnodes in ascending order.

**My code:**

/\*\*

\* 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 mergeTwoLists(ListNode l1, ListNode l2) {

ListNode result = new ListNode(0);

ListNode prevNode = result;

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

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

prevNode.next = l1;

l1 = l1.next;

} else {

prevNode.next = l2;

l2 = l2.next;

}

prevNode = prevNode.next;

}

if (l1 != null) {

prevNode.next = l1;

}

if (l2 != null) {

prevNode.next = l2;

}

return result.next;

}

}

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

**Task:** Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter**.

Return true if there is a cycle in the linked list. Otherwise, return false.

**Answer:**

**My code:**

/\*\*

\* 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 || head.next == null){

return false;}

ListNode n1 = head;

ListNode n2 = head.next;

while(true){

if(n1 == n2){

return true;}

if(n2.next == null || n2.next.next == null){

return false; }

n1 = n1.next;

n2 = n2.next.next;

}

}

}

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

**Task:** Given a linked list and a value *x*, partition it such that all nodes less than *x* come before nodes greater than or equal to *x*.

You should preserve the original relative order of the nodes in each of the two partitions.

**Answer:** Before and after are two pointers used to create two lists. Before head and after head are used to save the headers of two lists. All of them are initialized using the created dummy nodes. If the source node of the list is smaller than the specified x, assign it before the list. Move forward in the original list. The last node of the "after" list will also be the final node of the reformed list. Once all nodes are correctly assigned to the two lists, combine them to form a single list that will be returned.

**My code:** /\*\*

\* 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 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;

}

}

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

**Task:** Write a program to find the node at which the intersection of two singly linked lists begins.

**Answer:** If lists are of unequal length, we set pointer of longer list at 'move' distance from its head. Now,from this point of longer list, and from head of smaller list, both Lists are equal. And thus we simply traverse the lists and if both pointers are equal at any point, just return that node. If at no point are the two pointers equal, the lists have no intersection and thus return null.

**My code:**

/\*\*

\* 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 lenA = 0;

int lenB = 0;

ListNode currA = headA;

ListNode currB = headB;

if (headA == null || headB == null){

return null;}

while(currA != null) {

lenA++;

currA = currA.next;

}

while(currB != null) {

lenB++;

currB = currB.next;

}

currA = headA;

currB = headB;

if(lenA > lenB) {

for(int i=0; i<lenA-lenB; i++){

currA = currA.next;}

} else {

for(int i=0; i<lenB-lenA; i++){

currB = currB.next;}

}

while(currA != currB) {

currA = currA.next;

currB = currB.next;

}

return currA;

}

}

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

**Task:** Given a singly linked list, determine if it is a palindrome.

**Answer:** Get two nodes, one node moves faster than the other. If we find that the same cycle of nodes exists. If there is a zero child, the loop is not possible. One node moves faster than the other, if the cycle exists, they will meet at some point

**My code:**

/\*\*

\* 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 boolean isPalindrome(ListNode head) {

ListNode temp = head;

ArrayList<Integer> arr = new ArrayList();

int i=0;

int j;

while(temp!=null){

arr.add(temp.val);

temp=temp.next;

}

j=arr.size()-1;

while(i<j){

if(Integer.compare(arr.get(i),arr.get(j)) !=0){

return false;

}

i++;

j--;

}

return true;

}

}

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

**Task:** Given the head of a linked list, return the list after sorting it in ***ascending order***.

**Answer:**

**My code:**