Q1.class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def deleteMiddleNode(head):

if head is None or head.next is None:

return head

slow = head

fast = head

prev = None

while fast and fast.next:

prev = slow

slow = slow.next

fast = fast.next.next

prev.next = slow.next

return head

# Example usage

# Create the linked list: 1 -> 2 -> 3 -> 4 -> 5

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

print("Before deletion:")

current = head

while current:

print(current.val, end=" ")

current = current.next

print()

head = deleteMiddleNode(head)

print("After deletion:")

current = head

while current:

print(current.val, end=" ")

current = current.next

print()

Q2.class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def hasLoop(head):

slow = head

fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

return True

return False

# Example usage

# Create the linked list: 1 -> 3 -> 4 -> 3 (loop)

head = ListNode(1)

head.next = ListNode(3)

head.next.next = ListNode(4)

head.next.next.next = head.next

has\_loop = hasLoop(head)

print(has\_loop)

Q3class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def findNthFromEnd(head, N):

if not head:

return None

# Move the first pointer N nodes ahead

first = head

for \_ in range(N):

if not first:

return None # N is greater than the length of the linked list

first = first.next

# Move both pointers simultaneously

second = head

while first:

first = first.next

second = second.next

return second.val

# Example usage

# Create the linked list: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

head.next.next.next.next.next = ListNode(6)

head.next.next.next.next.next.next = ListNode(7)

head.next.next.next.next.next.next.next = ListNode(8)

head.next.next.next.next.next.next.next.next = ListNode(9)

N = 2

nth\_node = findNthFromEnd(head, N)

print(nth\_node)

Q4.class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def isPalindrome(head):

# Convert linked list to a list

current = head

values = []

while current:

values.append(current.val)

current = current.next

# Check if the list is a palindrome

start = 0

end = len(values) - 1

while start < end:

if values[start] != values[end]:

return False

start += 1

end -= 1

return True

# Example usage

# Create the linked list: R -> A -> D -> A -> R

head = ListNode('R')

head.next = ListNode('A')

head.next.next = ListNode('D')

head.next.next.next = ListNode('A')

head.next.next.next.next = ListNode('R')

result = isPalindrome(head)

print(result) # Output: True

Q5.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class LinkedListLoopRemoval {

public static ListNode detectAndRemoveLoop(ListNode head) {

ListNode slow = head;

ListNode fast = head;

// Find the meeting point of slow and fast pointers

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

slow = slow.next;

fast = fast.next.next;

if (slow == fast) {

break;

}

}

// If there is no loop, return the head of the linked list

if (slow != fast) {

return head;

}

// Move the slow pointer back to the head of the linked list

slow = head;

// Find the start of the loop

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

slow = slow.next;

fast = fast.next;

}

// Remove the loop

fast.next = null;

return head;

}

public static void main(String[] args) {

int N = 3;

int[] value = {1, 3, 4};

int X = 2;

// Create the linked list

ListNode[] nodes = new ListNode[N];

for (int i = 0; i < N; i++) {

nodes[i] = new ListNode(value[i]);

if (i > 0) {

nodes[i - 1].next = nodes[i];

}

}

if (X > 0) {

nodes[N - 1].next = nodes[X - 1];

}

// Detect and remove the loop

ListNode head = detectAndRemoveLoop(nodes[0]);

// Print the linked list

while (head != null) {

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

head = head.next;

}

}

}

Q6.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class LinkedListTraversal {

public static ListNode deleteNodes(ListNode head, int M, int N) {

ListNode curr = head;

ListNode prev = null;

while (curr != null) {

// Traverse M nodes

for (int i = 1; i <= M && curr != null; i++) {

prev = curr;

curr = curr.next;

}

// Delete N nodes

for (int i = 1; i <= N && curr != null; i++) {

curr = curr.next;

}

// Connect the previous node to the next node

prev.next = curr;

}

return head;

}

public static void printLinkedList(ListNode head) {

ListNode curr = head;

while (curr != null) {

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

curr = curr.next;

}

System.out.println();

}

public static void main(String[] args) {

// Create the linked list

ListNode head = new ListNode(1);

ListNode node2 = new ListNode(2);

ListNode node3 = new ListNode(3);

ListNode node4 = new ListNode(4);

ListNode node5 = new ListNode(5);

ListNode node6 = new ListNode(6);

ListNode node7 = new ListNode(7);

ListNode node8 = new ListNode(8);

ListNode node9 = new ListNode(9);

ListNode node10 = new ListNode(10);

head.next = node2;

node2.next = node3;

node3.next = node4;

node4.next = node5;

node5.next = node6;

node6.next = node7;

node7.next = node8;

node8.next = node9;

node9.next = node10;

int M = 1;

int N = 1;

System.out.println("Original Linked List:");

printLinkedList(head);

ListNode modifiedList = deleteNodes(head, M, N);

System.out.println("Modified Linked List:");

printLinkedList(modifiedList);

}

}

Q7class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class LinkedListInsertion {

public static void insertAtAlternatePositions(ListNode first, ListNode second) {

ListNode firstCurr = first;

ListNode secondCurr = second;

while (firstCurr != null && secondCurr != null) {

ListNode firstNext = firstCurr.next;

ListNode secondNext = secondCurr.next;

// Insert the second list node after the current first list node

firstCurr.next = secondCurr;

secondCurr.next = firstNext;

// Move to the next pair of nodes

firstCurr = firstNext;

secondCurr = secondNext;

}

// If the second list still has remaining nodes, append them to the first list

if (secondCurr != null) {

firstCurr.next = secondCurr;

}

// Empty the second list

second = null;

}

public static void printLinkedList(ListNode head) {

ListNode curr = head;

while (curr != null) {

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

curr = curr.next;

}

System.out.println();

}

public static void main(String[] args) {

// Create the first linked list

ListNode first = new ListNode(5);

ListNode node2 = new ListNode(7);

ListNode node3 = new ListNode(17);

ListNode node4 = new ListNode(13);

ListNode node5 = new ListNode(11);

first.next = node2;

node2.next = node3;

node3.next = node4;

node4.next = node5;

// Create the second linked list

ListNode second = new ListNode(12);

ListNode node6 = new ListNode(10);

ListNode node7 = new ListNode(2);

ListNode node8 = new ListNode(4);

ListNode node9 = new ListNode(6);

second.next = node6;

node6.next = node7;

node7.next = node8;

node8.next = node9;

System.out.println("First Linked List:");

printLinkedList(first);

System.out.println("Second Linked List:");

printLinkedList(second);

insertAtAlternatePositions(first, second);

System.out.println("Modified First Linked List:");

printLinkedList(first);

System.out.println("Modified Second Linked List:");

printLinkedList(second);

}

}

Q8.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class CircularLinkedList {

public static boolean isCircular(ListNode head) {

if (head == null) {

return false;

}

ListNode slow = head;

ListNode fast = head.next;

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

if (slow == fast) {

return true;

}

slow = slow.next;

fast = fast.next.next;

}

return false;

}

public static void main(String[] args) {

// Create a circular linked list

ListNode head = new ListNode(1);

ListNode node2 = new ListNode(2);

ListNode node3 = new ListNode(3);

ListNode node4 = new ListNode(4);

head.next = node2;

node2.next = node3;

node3.next = node4;

node4.next = head; // Circular connection

// Check if the linked list is circular

boolean isCircular = isCircular(head);

if (isCircular) {

System.out.println("The linked list is circular.");

} else {

System.out.println("The linked list is not circular.");

}

}

}