**Assignment Lists in Java**

Q1. Q1. Given a linked list and a key ‘X‘ in, the task is to check if X is present in the linked list or not.

Examples:

Input:14->21->11->30->10, X = 14

Output:Yes

Explanation: 14 is present in the linked list.

Input:6->21->17->30->10->8, X = 13

Output:No

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class LinkedListSearch {

// Function to check if the key X is present in the linked list

public static String search(Node head, int X) {

Node current = head;

// Traverse the linked list

while (current != null) {

// If key X is found, return "Yes"

if (current.data == X) {

return "Yes";

}

current = current.next;

}

// If we reach here, key X was not found, return "No"

return "No";

}

public static void main(String[] args) {

// Create a linked list: 14->21->11->30->10

Node head = new Node(14);

head.next = new Node(21);

head.next.next = new Node(11);

head.next.next.next = new Node(30);

head.next.next.next.next = new Node(10);

int X1 = 14;

int X2 = 13;

// Search for X1

System.out.println("Is " + X1 + " present? " + search(head, X1)); // Output: Yes

// Search for X2

System.out.println("Is " + X2 + " present? " + search(head, X2)); // Output: No

}

}

Q2. Insert a node at the given position in a linked list. We are given a pointer to a node, and the new node is inserted after the given node.

Input:LL = 1 -> 2 -> 4 -> 5 -> 6 pointer = 2 value = 3.

Output:1 -> 2 -> 3 -> 4 -> 5 -> 6

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class LinkedListInsert {

// Function to insert a new node after a given node

public static void insertAfter(Node prevNode, int newData) {

// Check if the given node is null

if (prevNode == null) {

System.out.println("The previous node cannot be null.");

return;

}

// Create a new node with the given data

Node newNode = new Node(newData);

// Point the new node's next to the previous node's next

newNode.next = prevNode.next;

// Point the previous node's next to the new node

prevNode.next = newNode;

}

// Function to print the linked list

public static void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data + " -> ");

current = current.next;

}

System.out.println("null");

}

public static void main(String[] args) {

// Create a linked list: 1 -> 2 -> 4 -> 5 -> 6

Node head = new Node(1);

head.next = new Node(2);

head.next.next = new Node(4);

head.next.next.next = new Node(5);

head.next.next.next.next = new Node(6);

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

printList(head);

// Insert node with value 3 after the node with value 2

insertAfter(head.next, 3); // Insert after node 2

System.out.println("\nLinked List after insertion:");

printList(head);

}

}

Q3. Given the head of a sorted linked list, delete all duplicates such that each element

appears only once. Return the linked list sorted as well.

1🡪1🡪2

1🡪2

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class RemoveDuplicates {

// Function to remove duplicates from a sorted linked list

public static Node removeDuplicates(Node head) {

if (head == null) {

return null;

}

Node current = head;

// Traverse the list

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

// If current node and the next node have the same data, skip the next node

if (current.data == current.next.data) {

current.next = current.next.next;

} else {

current = current.next;

}

}

return head;

}

// Function to print the linked list

public static void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data + " -> ");

current = current.next;

}

System.out.println("null");

}

public static void main(String[] args) {

// Create a sorted linked list: 1 -> 1 -> 2

Node head = new Node(1);

head.next = new Node(1);

head.next.next = new Node(2);

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

printList(head);

// Remove duplicates

head = removeDuplicates(head);

System.out.println("\nLinked List after removing duplicates:");

printList(head);

}

}

Q4. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example 1:

Input : head = [1,2,2,1]

Output : true

Example 2:

Input:head = [1,2]

Output:false

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class LinkedListPalindrome {

// Function to check if the linked list is a palindrome

public static boolean isPalindrome(Node head) {

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

return true;

}

// Step 1: Find the middle of the linked list using the slow-fast pointer approach

Node slow = head, fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

// Step 2: Reverse the second half of the linked list

Node secondHalf = reverseList(slow);

// Step 3: Compare the first half with the reversed second half

Node firstHalf = head;

Node copySecondHalf = secondHalf; // To restore the list later

boolean isPalindrome = true;

while (secondHalf != null) {

if (firstHalf.data != secondHalf.data) {

isPalindrome = false;

break;

}

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

// Step 4: Restore the original list (optional)

reverseList(copySecondHalf);

return isPalindrome;

}

// Helper function to reverse a linked list

private static Node reverseList(Node head) {

Node prev = null;

Node current = head;

while (current != null) {

Node nextTemp = current.next;

current.next = prev;

prev = current;

current = nextTemp;

}

return prev;

}

// Function to print the linked list

public static void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data + " -> ");

current = current.next;

}

System.out.println("null");

}

public static void main(String[] args) {

// Example 1: head = [1, 2, 2, 1]

Node head1 = new Node(1);

head1.next = new Node(2);

head1.next.next = new Node(2);

head1.next.next.next = new Node(1);

System.out.println("Linked List 1:");

printList(head1);

System.out.println("Is Palindrome? " + isPalindrome(head1)); // Output: true

// Example 2: head = [1, 2]

Node head2 = new Node(1);

head2.next = new Node(2);

System.out.println("\nLinked List 2:");

printList(head2);

System.out.println("Is Palindrome? " + isPalindrome(head2)); // Output: false

}

}

Q5. Given two numbers represented by two lists, write a function that returns the sum list. The

sum list is a list representation of the addition of two input numbers.

Example:

Input:

List1: 5->6->3

List2: 8->4->2

Output:

Resultant list: 1->4->0->5

Explanation: 563 + 842 = 1405

Input:

List1: 7->5->9->4->6

List2: 8->4

Output:

Resultant list: 7->6->0->3->0

Explanation: 75946+84=76030

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class LinkedListAddition {

// Function to reverse a linked list

private static Node reverseList(Node head) {

Node prev = null;

Node current = head;

while (current != null) {

Node nextTemp = current.next;

current.next = prev;

prev = current;

current = nextTemp;

}

return prev;

}

// Function to add two numbers represented by linked lists

public static Node addLists(Node l1, Node l2) {

// Reverse both lists

l1 = reverseList(l1);

l2 = reverseList(l2);

Node dummyHead = new Node(0);

Node p = l1, q = l2, current = dummyHead;

int carry = 0;

// Traverse both lists and add corresponding digits

while (p != null || q != null) {

int x = (p != null) ? p.data : 0;

int y = (q != null) ? q.data : 0;

int sum = carry + x + y;

carry = sum / 10;

current.next = new Node(sum % 10);

current = current.next;

if (p != null) p = p.next;

if (q != null) q = q.next;

}

// If there's a carry left, add a new node with carry value

if (carry > 0) {

current.next = new Node(carry);

}

// Reverse the result list to get the final answer

return reverseList(dummyHead.next);

}

// Function to print the linked list

public static void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data + " -> ");

current = current.next;

}

System.out.println("null");

}

public static void main(String[] args) {

// Example 1: 563 + 842

Node l1 = new Node(5);

l1.next = new Node(6);

l1.next.next = new Node(3);

Node l2 = new Node(8);

l2.next = new Node(4);

l2.next.next = new Node(2);

Node result = addLists(l1, l2);

System.out.println("Resultant list:");

printList(result); // Output: 1 -> 4 -> 0 -> 5 -> null

// Example 2: 75946 + 84

Node l3 = new Node(7);

l3.next = new Node(5);

l3.next.next = new Node(9);

l3.next.next.next = new Node(4);

l3.next.next.next.next = new Node(6);

Node l4 = new Node(8);

l4.next = new Node(4);

result = addLists(l3, l4);

System.out.println("\nResultant list:");

printList(result); // Output: 7 -> 6 -> 0 -> 3 -> 0 -> null

}

}