**Assignment 4**

Following things to be added in each question:

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 3/10/2024

**1. Implement a singly linked list with basic operations: insert, delete, search.**

* **Test Case 1**:  
  Input: Insert 3 → Insert 7 → Insert 5 → Delete 7 → Search 5  
  Output: List = [3, 5], Found = True
* **Test Case 2**:  
  Input: Insert 9 → Insert 4 → Delete 4 → Search 10  
  Output: List = [9], Found = False

**class SinglyLinkedList {**

**class Node {**

**int data;**

**Node next;**

**Node(int d) { data = d; next = null; }**

**}**

**Node head;**

**public void insert(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node temp = head;**

**while (temp.next != null) temp = temp.next;**

**temp.next = newNode;**

**}**

**}**

**public void delete(int key) {**

**if (head == null) return;**

**if (head.data == key) {**

**head = head.next;**

**return;**

**}**

**Node temp = head;**

**while (temp.next != null && temp.next.data != key) {**

**temp = temp.next;**

**}**

**if (temp.next != null) {**

**temp.next = temp.next.next;**

**}**

**}**

**public boolean search(int key) {**

**Node temp = head;**

**while (temp != null) {**

**if (temp.data == key) return true;**

**temp = temp.next;**

**}**

**return false;**

**}**

**public void printList() {**

**Node temp = head;**

**while (temp != null) {**

**System.out.print(temp.data + " ");**

**temp = temp.next;**

**}**

**System.out.println();**

**}**

**public static void main(String[] args) {**

**SinglyLinkedList list = new SinglyLinkedList();**

**list.insert(3); list.insert(7); list.insert(5);**

**list.delete(7);**

**System.out.print("List: ");**

**list.printList();**

**System.out.println("Found 5: " + list.search(5)); // True**

**}**

**}**

**2. Reverse a singly linked list.**

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: List = [5, 4, 3, 2, 1]
* **Test Case 2**:  
  Input: List = [10, 20, 30]  
  Output: List = [30, 20, 10]

**public void reverse() {**

**Node prev = null, current = head, next = null;**

**while (current != null) {**

**next = current.next;**

**current.next = prev;**

**prev = current;**

**current = next;**

**}**

**head = prev;**

**}**

**3. Detect a cycle in a linked list.**

* **Test Case 1**:  
  Input: List = [1 → 2 → 3 → 4 → 5 → 3 (cycle)]  
  Output: Cycle Detected
* **Test Case 2**:  
  Input: List = [6 → 7 → 8 → 9]  
  Output: No Cycle

**public boolean detectCycle() {**

**Node slow = head, fast = head;**

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

**slow = slow.next;**

**fast = fast.next.next;**

**if (slow == fast) return true;**

**}**

**return false;**

**}**

**4. Merge two sorted linked lists.**

* **Test Case 1**:  
  Input: List1 = [1, 3, 5], List2 = [2, 4, 6]  
  Output: Merged List = [1, 2, 3, 4, 5, 6]
* **Test Case 2**:  
  Input: List1 = [10, 15, 20], List2 = [12, 18, 25]  
  Output: Merged List = [10, 12, 15, 18, 20, 25]

**public static Node mergeSortedLists(Node l1, Node l2) {**

**Node dummy = new Node(0);**

**Node current = dummy;**

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

**if (l1.data < l2.data) {**

**current.next = l1;**

**l1 = l1.next;**

**} else {**

**current.next = l2;**

**l2 = l2.next;**

**}**

**current = current.next;**

**}**

**if (l1 != null) current.next = l1;**

**if (l2 != null) current.next = l2;**

**return dummy.next;**

**}**

**5. Find the nth node from the end of a linked list.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], n = 2  
  Output: 40
* **Test Case 2**:  
  Input: List = [5, 15, 25, 35], n = 4  
  Output: 5

**public int findNthFromEnd(int n) {**

**Node first = head, second = head;**

**for (int i = 0; i < n; i++) {**

**if (second == null) return -1; // n is larger than the size of the list**

**second = second.next;**

**}**

**while (second != null) {**

**first = first.next;**

**second = second.next;**

**}**

**return first.data;**

**}**

**6. Remove duplicates from a sorted linked list.**

* **Test Case 1**:  
  Input: List = [1, 1, 2, 3, 3, 4]  
  Output: List = [1, 2, 3, 4]
* **Test Case 2**:  
  Input: List = [7, 7, 8, 9, 9, 10]  
  Output: List = [7, 8, 9, 10]

**public void removeDuplicates() {**

**Node current = head;**

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

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

**current.next = current.next.next;**

**} else {**

**current = current.next;**

**}**

**}**

**}**

**7. Implement a doubly linked list with insert, delete, and traverse operations.**

* **Test Case 1**:  
  Input: Insert 10 → Insert 20 → Insert 30 → Delete 20  
  Output: List = [10, 30]
* **Test Case 2**:  
  Input: Insert 1 → Insert 2 → Insert 3 → Delete 1  
  Output: List = [2, 3]

**class DoublyLinkedList {**

**class Node {**

**int data;**

**Node next, prev;**

**Node(int d) { data = d; next = prev = null; }**

**}**

**Node head;**

**public void insert(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**} else {**

**Node temp = head;**

**while (temp.next != null) temp = temp.next;**

**temp.next = newNode;**

**newNode.prev = temp;**

**}**

**}**

**public void delete(int key) {**

**if (head == null) return;**

**Node temp = head;**

**while (temp != null && temp.data != key) temp = temp.next;**

**if (temp != null) {**

**if (temp.prev != null) temp.prev.next = temp.next;**

**if (temp.next != null) temp.next.prev = temp.prev;**

**if (temp == head) head = temp.next;**

**}**

**}**

**public void traverse() {**

**Node temp = head;**

**while (temp != null) {**

**System.out.print(temp.data + " ");**

**temp = temp.next;**

**}**

**System.out.println();**

**}**

**}**

**8. Reverse a doubly linked list.**

* **Test Case 1**:  
  Input: List = [5, 10, 15, 20]  
  Output: List = [20, 15, 10, 5]
* **Test Case 2**:  
  Input: List = [4, 8, 12]  
  Output: List = [12, 8, 4]

**public void reverseDoubly() {**

**Node temp = null, current = head;**

**while (current != null) {**

**temp = current.prev;**

**current.prev = current.next;**

**current.next = temp;**

**current = current.prev;**

**}**

**if (temp != null) head = temp.prev;**

**}**

**9. Add two numbers represented by linked lists.**

* **Test Case 1**:  
  Input: List1 = [2 → 4 → 3], List2 = [5 → 6 → 4] (243 + 465)  
  Output: Sum List = [7 → 0 → 8]
* **Test Case 2**:  
  Input: List1 = [9 → 9 → 9], List2 = [1] (999 + 1)  
  Output: Sum List = [0 → 0 → 0 → 1]

**public Node addTwoNumbers(Node l1, Node l2) {**

**Node dummy = new Node(0);**

**Node p = l1, q = l2, current = dummy;**

**int carry = 0;**

**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 (carry > 0) {**

**current.next = new Node(carry);**

**}**

**return dummy.next;**

**}**

**10. Rotate a linked list by k places.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], k = 2  
  Output: List = [30, 40, 50, 10, 20]
* **Test Case 2**:  
  Input: List = [5, 10, 15, 20], k = 3  
  Output: List = [20, 5, 10, 15]

**public void rotate(int k) {**

**if (k == 0 || head == null) return;**

**Node current = head;**

**int length = 1;**

**// Get the length of the list**

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

**current = current.next;**

**length++;**

**}**

**// Make the list circular**

**current.next = head;**

**// Find the node where rotation should happen**

**k = k % length;**

**for (int i = 0; i < length - k; i++) {**

**current = current.next;**

**}**

**// Set the new head and break the circular link**

**head = current.next;**

**current.next = null;**

**}**

**public void printList() {**

**Node temp = head;**

**while (temp != null) {**

**System.out.print(temp.data + " ");**

**temp = temp.next;**

**}**

**System.out.println();**

**}**

**11. Flatten a multilevel doubly linked list.**

* **Test Case 1**:  
  Input: List = [1 → 2 → 3, 3 → 7 → 8, 8 → 10 → 12]  
  Output: Flattened List = [1 → 2 → 3 → 7 → 8 → 10 → 12]
* **Test Case 2**:  
  Input: List = [1 → 2 → 3, 2 → 5 → 6, 6 → 7 → 9]  
  Output: Flattened List = [1 → 2 → 5 → 6 → 7 → 9 → 3]

**public Node flatten(Node head) {**

**if (head == null) return head;**

**Node curr = head;**

**while (curr != null) {**

**if (curr.child != null) {**

**Node next = curr.next;**

**curr.next = curr.child;**

**curr.child.prev = curr;**

**curr.child = null;**

**Node last = curr.next;**

**while (last.next != null) last = last.next;**

**last.next = next;**

**if (next != null) next.prev = last;**

**}**

**curr = curr.next;**

**}**

**return head;**

**}**

**12. Split a circular linked list into two halves.**

* **Test Case 1**:  
  Input: Circular List = [1 → 2 → 3 → 4 → 5 → 6 → (back to 1)]  
  Output: List1 = [1 → 2 → 3], List2 = [4 → 5 → 6]
* **Test Case 2**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)]  
  Output: List1 = [10 → 20], List2 = [30 → 40]

**public void splitCircularList() {**

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

**Node slow = head, fast = head;**

**// Use the slow-fast pointer technique to find the middle**

**while (fast.next != head && fast.next.next != head) {**

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**Node head2 = slow.next;**

**slow.next = head;**

**Node temp = head2;**

**while (temp.next != head) {**

**temp = temp.next;**

**}**

**temp.next = head2;**

**// Print the two halves**

**System.out.print("First half: ");**

**printList(head);**

**System.out.print("Second half: ");**

**printList(head2);**

**}**

**13. Insert a node in a sorted circular linked list.**

* **Test Case 1**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)], Insert 25  
  Output: Circular List = [10 → 20 → 25 → 30 → 40 → (back to 10)]
* **Test Case 2**:  
  Input: Circular List = [5 → 15 → 25 → (back to 5)], Insert 10  
  Output: Circular List = [5 → 10 → 15 → 25 → (back to 5)]

**public void insertInSortedCircular(int data) {**

**Node newNode = new Node(data);**

**if (head == null) {**

**head = newNode;**

**newNode.next = head;**

**return;**

**}**

**Node current = head;**

**while (current.next != head && current.next.data < data) {**

**current = current.next;**

**}**

**newNode.next = current.next;**

**current.next = newNode;**

**if (newNode.data < head.data) {**

**head = newNode;**

**}**

**}**

**public void printCircularList() {**

**if (head == null) return;**

**Node temp = head;**

**do {**

**System.out.print(temp.data + " ");**

**temp = temp.next;**

**} while (temp != head);**

**System.out.println();**

**}**

**14. Check if two linked lists intersect, and find the intersection point if they do.**

* **Test Case 1**:  
  Input: List1 = [1 → 2 → 3 → 4 → 5], List2 = [6 → 7 → 4 → 5]  
  Output: Intersection Point = 4
* **Test Case 2**:  
  Input: List1 = [10 → 20 → 30 → 40], List2 = [15 → 25 → 35]  
  Output: No Intersection

**public Node getIntersectionNode(Node headA, Node headB) {**

**if (headA == null || headB == null) return null;**

**Node a = headA, b = headB;**

**while (a != b) {**

**a = (a == null) ? headB : a.next;**

**b = (b == null) ? headA : b.next;**

**}**

**return a;**

**}**

**15. Find the middle element of a linked list in one pass.**

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: Middle = 3
* **Test Case 2**:  
  Input: List = [11, 22, 33, 44, 55, 66]  
  Output: Middle = 44

**public Node findMiddle() {**

**Node slow = head, fast = head;**

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

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**return slow;**

**}**