**C-DAC Mumbai Date 01/10/2024**

**Subject: Algorithm and Data Structure**

**Assignment 3**

**Solve the assignment with following thing 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

Program:

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

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;

}

}

void delete(int data) {

if (head == null) {

System.out.println("List is empty");

return;

}

if (head.data == data) {

head = head.next;

return;

}

Node temp = head;

while (temp.next != null) {

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

temp.next = temp.next.next;

return;

}

temp = temp.next;

}

System.out.println("Element not found");

}

boolean search(int data) {

Node temp = head;

while (temp != null) {

if (temp.data == data) {

return true;

}

temp = temp.next;

}

return false;

}

void display() {

Node temp = head;

System.out.print("List = [");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedList1 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.insert(3);

list.insert(7);

list.insert(5);

list.delete(7);

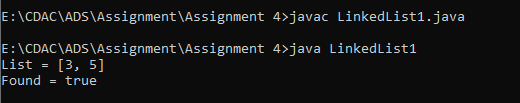
list.display();

System.out.println("Found = " + list.search(5));

}

}

Output:



**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]

Program:

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void reverse() {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}

void display() {

Node temp = head;

System.out.print("List = [");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ2 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

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

list.display();

list.reverse();

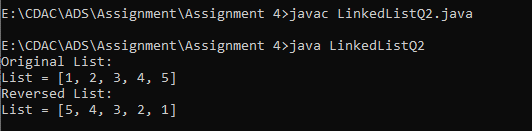
System.out.println("Reversed List:");

list.display();

}

}

Output:



**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

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void createCycle(int data) {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = head; // Create cycle at head

}

boolean detectCycle() {

Node slow = head;

Node fast = head;

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

slow = slow.next;

fast = fast.next.next;

if (slow == fast) {

//System.out.println("Cycle detected");

return true ;

}

}

//System.out.println("No Cycle");

return false;

}

void display() {

Node temp = head;

int count = 0;

while (temp != null && count < 10) { // Limit display to 10 nodes

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

temp = temp.next;

count++;

}

System.out.println();

}

}

public class LinkedListQ3 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

list.createCycle(1); // Create cycle at node 1

if (list.detectCycle()) {

System.out.println("Cycle Detected");

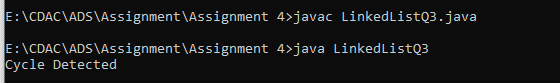
} else {

System.out.println("No Cycle");

}

}

}



**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]

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class Main {

public static void main(String[] args) {

LinkedList list1 = new LinkedList();

list1.append(1);

list1.append(3);

list1.append(5);

LinkedList list2 = new LinkedList();

list2.append(2);

list2.append(4);

list2.append(6);

LinkedList mergedList = mergeLists(list1, list2);

System.out.println("List1:");

list1.display();

System.out.println("List2:");

list2.display();

System.out.println("Merged List:");

mergedList.display();

}

static LinkedList mergeLists(LinkedList list1, LinkedList list2) {

LinkedList mergedList = new LinkedList();

Node node1 = list1.head;

Node node2 = list2.head;

while (node1 != null && node2 != null) {

if (node1.data < node2.data) {

mergedList.append(node1.data);

node1 = node1.next;

} else {

mergedList.append(node2.data);

node2 = node2.next;

}

}

// Append remaining nodes

while (node1 != null) {

mergedList.append(node1.data);

node1 = node1.next;

}

while (node2 != null) {

mergedList.append(node2.data);

node2 = node2.next;

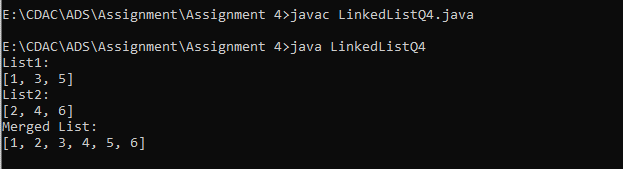
}

return mergedList;

}

}

Output:



**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

1. Create a linked list with append method.

2. Create a nthNodeFromEnd function to find the nth node from the end.

3. Use two pointers, mainPtr and refPtr, to traverse the list.

4. Move refPtr n nodes ahead.

5. Move both pointers until refPtr reaches the end.

6. mainPtr will be at the nth node from the end.

Program:

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

int nthNodeFromEnd(int n) {

Node mainPtr = head;

Node refPtr = head;

int count = 0;

if (head == null) {

return -1; // List is empty

}

while (count < n) {

if (refPtr == null) {

return -1; // n is greater than list length

}

refPtr = refPtr.next;

count++;

}

while (refPtr != null) {

mainPtr = mainPtr.next;

refPtr = refPtr.next;

}

return mainPtr.data;

}

}

public class LinkedListQ5 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(10);

list.append(20);

list.append(30);

list.append(40);

list.append(50);

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

list.display();

int n = 2;

int result = list.nthNodeFromEnd(n);

if (result != -1) {

System.out.println(n + "th node from end: " + result);

} else {

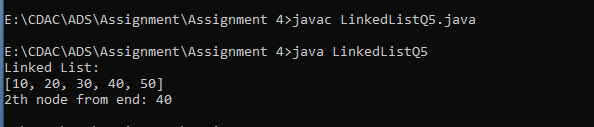
System.out.println("Invalid input");

}

}

}

Output:



**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]

1. Create a linked list with append method.

2. Create a removeDuplicates function to remove duplicates.

3. check the list, comparing adjacent nodes.

4. If duplicates are found, skip the next node.

5. Display original and updated lists using display method.

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

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;

}

}

}

}

public class LinkedListQ6 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(1);

list.append(1);

list.append(2);

list.append(3);

list.append(3);

list.append(4);

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

list.display();

list.removeDuplicates();

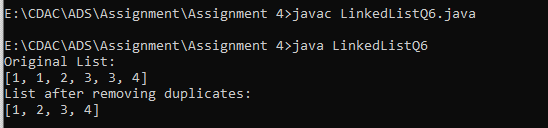
System.out.println("List after removing duplicates:");

list.display();

}

}

Output:



**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 Node {

int data;

Node prev;

Node next;

Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

class DoublyLinkedList {

Node head;

Node tail;

void insertAtBeginning(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head.prev = newNode;

head = newNode;

}

}

void insertAtEnd(int data) {

Node newNode = new Node(data);

if (tail == null) {

head = newNode;

tail = newNode;

} else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

void deleteNode(int data) {

Node temp = head;

while (temp != null) {

if (temp.data == data) {

if (temp.prev == null) {

head = temp.next;

if (head != null) {

head.prev = null;

} else {

tail = null;

}

} else if (temp.next == null) {

tail = temp.prev;

tail.next = null;

} else {

temp.prev.next = temp.next;

temp.next.prev = temp.prev;

}

return;

}

temp = temp.next;

}

}

void traverseForward() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

void traverseBackward() {

Node temp = tail;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.prev != null) {

System.out.print(", ");

}

temp = temp.prev;

}

System.out.println("]");

}

void display() {

Node temp = head;

System.out.print("List = [");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ7 {

public static void main(String[] args) {

DoublyLinkedList dll = new DoublyLinkedList();

dll.insertAtEnd(10);

dll.insertAtEnd(20);

dll.insertAtEnd(30);

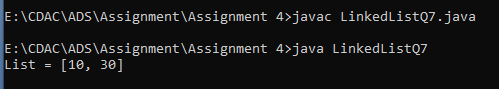
dll.deleteNode(20);

dll.display();

}

}

Output:



**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]

class Node {

int data;

Node next;

Node prev;

Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

class DoublyLinkedList {

Node head;

Node tail;

void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

void reverse() {

Node temp = null;

Node current = head;

while (current != null) {

temp = current.prev;

current.prev = current.next;

current.next = temp;

current = current.prev;

}

if (temp != null) {

head = temp.prev;

}

}

void display() {

Node temp = head;

System.out.print("List = [");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ8 {

public static void main(String[] args) {

DoublyLinkedList dll = new DoublyLinkedList();

dll.append(5);

dll.append(10);

dll.append(15);

dll.append(20);

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

dll.display();

dll.reverse();

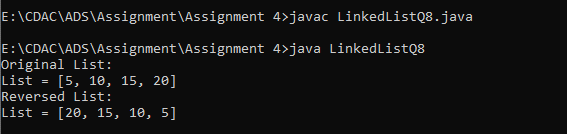
System.out.println("Reversed List:");

dll.display();

}

}

Output:



**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]

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print("->");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ9 {

public static void main(String[] args) {

LinkedList list1 = new LinkedList();

list1.append(3);

list1.append(4);

list1.append(2);

LinkedList list2 = new LinkedList();

list2.append(4);

list2.append(6);

list2.append(5);

System.out.println("List1:");

list1.display();

System.out.println("List2:");

list2.display();

LinkedList sumList = addLists(list1, list2);

System.out.println("Sum List:");

sumList.display();

}

static LinkedList addLists(LinkedList list1, LinkedList list2) {

LinkedList sumList = new LinkedList();

Node node1 = list1.head;

Node node2 = list2.head;

int carry = 0;

while (node1 != null || node2 != null) {

int sum = carry;

if (node1 != null) {

sum += node1.data;

node1 = node1.next;

}

if (node2 != null) {

sum += node2.data;

node2 = node2.next;

}

carry = sum / 10;

sum %= 10;

sumList.append(sum);

}

if (carry > 0) {

sumList.append(carry);

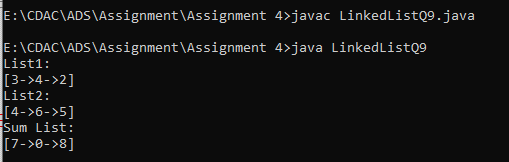
}

return sumList;

}

}

Output:



**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]

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

void rotate(int k) {

if (head == null || k == 0) {

return;

}

Node temp = head;

int length = 1;

while (temp.next != null) {

temp = temp.next;

length++;

}

temp.next = head; // Make it circular

k = k % length;

int newTailIndex = length - k - 1;

Node newTail = head;

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

newTail = newTail.next;

}

head = newTail.next;

newTail.next = null; // Break the circle

}

}

public class LinkedListQ10 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(10);

list.append(20);

list.append(30);

list.append(40);

list.append(50);

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

list.display();

int k = 2;

list.rotate(k);

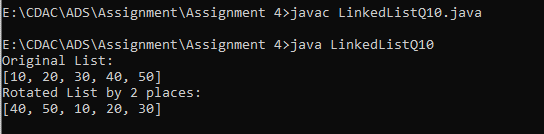
System.out.println("Rotated List by " + k + " places:");

list.display();

}

}

Output:



**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]

class Node {

int data;

Node next;

Node prev;

Node child;

Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

this.child = null;

}

}

class DoublyLinkedList {

Node head;

void append(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;

}

}

void flatten() {

Node temp = head;

while (temp != null) {

if (temp.child != null) {

Node childTemp = temp.child;

while (childTemp.next != null) {

childTemp = childTemp.next;

}

childTemp.next = temp.next;

if (temp.next != null) {

temp.next.prev = childTemp;

}

temp.next = temp.child;

temp.child.prev = temp;

temp.child = null;

}

temp = temp.next;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(" -> ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ11{

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

list.append(1);

list.append(2);

list.append(3);

Node child1 = new Node(7);

child1.next = new Node(8);

list.head.next.next.child = child1;

Node child2 = new Node(10);

child2.next = new Node(12);

child1.next.child = child2;

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

list.display();

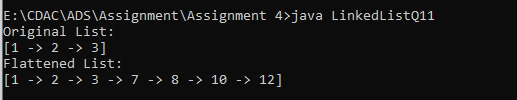
list.flatten();

System.out.println("Flattened List:");

list.display();

}

}



**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]

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class CircularLinkedList {

Node head;

void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

head.next = head; // Circular

} else {

Node temp = head;

while (temp.next != head) {

temp = temp.next;

}

temp.next = newNode;

newNode.next = head;

}

}

void display(Node head) {

Node temp = head;

System.out.print("[");

while (true) {

System.out.print(temp.data);

if (temp.next == head) {

break;

}

System.out.print(" -> ");

temp = temp.next;

}

System.out.println("]");

}

void split() {

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

return;

}

Node slow = head;

Node fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

Node secondHead = slow.next;

slow.next = head;

Node temp = secondHead;

while (temp.next != head) {

temp = temp.next;

}

temp.next = secondHead;

System.out.println("List1:");

display(head);

System.out.println("List2:");

display(secondHead);

}

}

public class LinkedListQ12 {

public static void main(String[] args) {

CircularLinkedList list = new CircularLinkedList();

list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

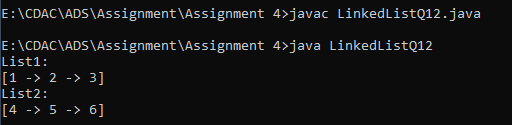
list.append(6);

list.split();

}

}

Output:



**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)]

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class CircularLinkedList {

Node head;

void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

head.next = head; // Circular

} else {

Node temp = head;

while (temp.next != head) {

temp = temp.next;

}

temp.next = newNode;

newNode.next = head;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (true) {

System.out.print(temp.data);

if (temp.next == head) {

break;

}

System.out.print(" -> ");

temp = temp.next;

}

System.out.println("]");

}

void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

head.next = head; // Circular

} else if (data <= head.data) {

Node temp = head;

while (temp.next != head) {

temp = temp.next;

}

temp.next = newNode;

newNode.next = head;

head = newNode;

} else {

Node temp = head;

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

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

}

}

public class LinkedListQ13 {

public static void main(String[] args) {

CircularLinkedList list = new CircularLinkedList();

list.append(10);

list.append(20);

list.append(30);

list.append(40);

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

list.display();

int data = 25;

list.insert(data);

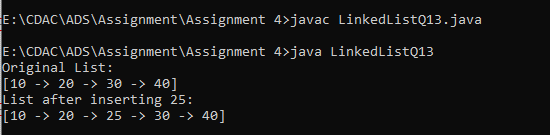
System.out.println("List after inserting " + data + ":");

list.display();

}

}

Output:



**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

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void append(Node node) {

if (head == null) {

head = node;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = node;

}

}

int length() {

int length = 0;

Node temp = head;

while (temp != null) {

length++;

temp = temp.next;

}

return length;

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(" -> ");

}

temp = temp.next;

}

System.out.println("]");

}

}

public class LinkedListQ14 {

public static void main(String[] args) {

LinkedList list1 = new LinkedList();

list1.append(1);

list1.append(2);

list1.append(3);

Node intersection = new Node(4);

intersection.next = new Node(5);

list1.append(intersection);

LinkedList list2 = new LinkedList();

list2.append(6);

list2.append(7);

list2.append(intersection);

System.out.println("List1:");

list1.display();

System.out.println("List2:");

list2.display();

Node intersectPoint = findIntersection(list1, list2);

if (intersectPoint != null) {

System.out.println("Intersection Point = " + intersectPoint.data);

} else {

System.out.println("No intersection");

}

}

static Node findIntersection(LinkedList list1, LinkedList list2) {

int len1 = list1.length();

int len2 = list2.length();

int diff = Math.abs(len1 - len2);

Node longer = len1 > len2 ? list1.head : list2.head;

Node shorter = len1 > len2 ? list2.head : list1.head;

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

longer = longer.next;

}

while (longer != null && shorter != null) {

if (longer == shorter) {

return longer;

}

longer = longer.next;

shorter = shorter.next;

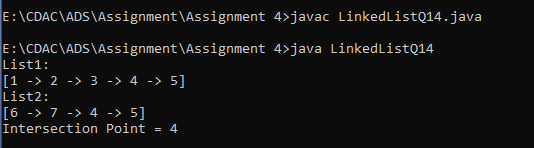
}

return null;

}

}

Output:



**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

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

Node head;

void append(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;

}

}

void display() {

Node temp = head;

System.out.print("[");

while (temp != null) {

System.out.print(temp.data);

if (temp.next != null) {

System.out.print(", ");

}

temp = temp.next;

}

System.out.println("]");

}

int findMiddle() {

if (head == null) {

return -1; // List is empty

}

Node slow = head;

Node fast = head;

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

slow = slow.next; // Move one step at a time

fast = fast.next.next; // Move two steps at a time

}

return slow.data; // Return middle element's data

}

}

public class LinkedListQ15 {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

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

list.display();

int middle = list.findMiddle();

System.out.println("Middle Element = " + middle);

}

}

Output:

