Assignment 14 Solution – Linked List | DSA

Question 1

Given a linked list of N nodes such that it may contain a loop.
A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.
Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

Example 1:
Input:
N = 3
value[] = {1,3,4}
X = 2
Output:1
Explanation: The link list looks like
1 -> 3 -> 4
^ |
|___|
A loop is present. If you remove it successfully, the answer will be 1.

Example 2:
Input:
N = 4
value[] = {1,8,3,4}
X = 0
Output:1
Explanation: The Linked list does not contain any loop.

Example 3:
Input:
N = 4
value[] = {1,2,3,4}
X = 1
Output:1
Explanation: The link list looks like
1 -> 2 -> 3 -> 4
^ |
|_____|
A loop is present. If you remove it successfully, the answer will be 1.

Solution Code:

```
package in.ineuron.pptAssignment14;

class Node {
	int data;
	Node next;

	Node(int data) {
		this.data = data;
		next = null;
	}
}

public class DetectAndRemoveLoop_1 {
	Node head;

	void addNode(int data) {
		Node newNode = new Node(data);
		if (head == null) {
			head = newNode;
		} else {
			Node curr = head;
			while (curr.next != null) {
				curr = curr.next;
			}
			curr.next = newNode;
		}
	}

	void detectAndRemoveLoop() {
		if (head == null || head.next == null) {
			return;
		}

		Node slow = head;
		Node fast = head;

		// Detect the loop using Floyd's cycle detection algorithm
		while (fast != null && fast.next != null) {
			slow = slow.next;
			fast = fast.next.next;
			if (slow == fast) {
				break;
			}
		}
```

```
            // If loop exists, find the start of the loop
            if (slow == fast) {
                slow = head;
                while (slow.next != fast.next) {
                    slow = slow.next;
                    fast = fast.next;
                }
                // Remove the loop by setting the next of the last node to null
                fast.next = null;
            }
        }

        void printList() {
            Node curr = head;
            while (curr != null) {
                System.out.print(curr.data + " ");
                curr = curr.next;
            }
            System.out.println();
        }

        public static void main(String[] args) {
            DetectAndRemoveLoop_1 list = new DetectAndRemoveLoop_1();
            list.addNode(1);
            list.addNode(8);
            list.addNode(3);
            list.addNode(4);

            // Creating a loop by connecting the last node to the node at position X
            // (1-based index)
            list.head.next.next.next = list.head.next;

            // Detect and remove the loop
            list.detectAndRemoveLoop();

            // Print the modified linked list
            list.printList();
        }
    }
```

Question 2

A number N is represented in Linked List such that each digit corresponds to a node in linked list. You need to add 1 to it.

Example 1:
Input:
LinkedList: 4->5->6
Output:457

Example 2:
Input:
LinkedList: 1->2->3
Output:124

Solution Code :

```
package in.ineuron.pptAssignment14;

class Node {
	int data;
	Node next;

	Node(int data) {
		this.data = data;
		next = null;
	}
}

public class AddOneLinkedList_2 {
	Node head;

	void addNode(int data) {
		Node newNode = new Node(data);
		if (head == null) {
			head = newNode;
		} else {
			Node curr = head;
			while (curr.next != null) {
				curr = curr.next;
			}
			curr.next = newNode;
		}
	}

	Node reverseList(Node node) {
		Node prev = null;
		Node curr = node;
		Node next = null;
```

```
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }

        return prev;
    }

    void addOne() {
        head = reverseList(head);
        Node curr = head;
        int carry = 1;

        while (curr != null) {
            int sum = curr.data + carry;
            curr.data = sum % 10;
            carry = sum / 10;
            curr = curr.next;
        }

        if (carry > 0) {
            Node newNode = new Node(carry);
            newNode.next = head;
            head = newNode;
        }

        head = reverseList(head);
    }

    void printList() {
        Node curr = head;
        while (curr != null) {
            System.out.print(curr.data);
            curr = curr.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        AddOneLinkedList_2 list = new AddOneLinkedList_2();
        list.addNode(4);
        list.addNode(5);
        list.addNode(6);
```

```
                // Add 1 to the linked list
                list.addOne();

                // Print the modified linked list
                list.printList();
        }
    }
```

Question 3

Given a Linked List of size N, where every node represents a sub-linked-list and contains two pointers:(i) a next pointer to the next node,(ii) a bottom pointer to a linked list where this node is head. Each of the sub-linked-list is in sorted order. Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order. Note: The flattened list will be printed using the bottom pointer instead of next pointer.

Example 1:
Input:

```
5 -> 10 -> 19 -> 28
|    |       |      |
7    20    22     35
|            |      |
8           50     40
|                   |
30                 45
```

Output: 5-> 7-> 8- > 10 -> 19-> 20->
22-> 28-> 30-> 35-> 40-> 45-> 50.
Explanation:
The resultant linked lists has every
node in a single level.(Note:| represents the bottom pointer.)

Example 2:
Input:

```
5 -> 10 -> 19 -> 28
|                 |
7        22
|        |
8        50
|
30
```

Output: 5->7->8->10->19->22->28->30->50
Explanation: The resultant linked lists has every node in a single level.

(Note:| represents the bottom pointer.)

Solution Code:

```
package in.ineuron.pptAssignment14;

// Java program for flattening a Linked List
public class FlattenLinkedList_3 {
	Node head; // head of list

	/* Linked list Node */
	class Node {
		int data;
		Node right, down;

		Node(int data) {
			this.data = data;
			right = null;
			down = null;
		}
	}

	// An utility function to merge two sorted linked lists
	Node merge(Node a, Node b) {
		// if first linked list is empty then second
		// is the answer
		if (a == null)
			return b;

		// if second linked list is empty then first
		// is the result
		if (b == null)
			return a;

		// compare the data members of the two linked lists
		// and put the larger one in the result
		Node result;

		if (a.data < b.data) {
			result = a;
			result.down = merge(a.down, b);
		}

		else {
			result = b;
			result.down = merge(a, b.down);
		}

		result.right = null;
```

```
        return result;
    }

    Node flatten(Node root) {
        // Base Cases
        if (root == null || root.right == null)
            return root;

        // recur for list on right
        root.right = flatten(root.right);

        // now merge
        root = merge(root, root.right);

        // return the root
        // it will be in turn merged with its left
        return root;
    }

    /*
     * Utility function to insert a node at beginning of the linked list
     */
    Node push(Node head_ref, int data) {
        /*
         * 1 & 2: Allocate the Node & Put in the data
         */
        Node new_node = new Node(data);

        /* 3. Make next of new Node as head */
        new_node.down = head_ref;

        /* 4. Move the head to point to new Node */
        head_ref = new_node;

        /* 5. return to link it back */
        return head_ref;
    }

    void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.down;
        }
        System.out.println();
    }
```

```
        // Driver's code
        public static void main(String args[]) {
                FlattenLinkedList_3 L = new FlattenLinkedList_3();

                /*
                 * Let us create the following linked list 5 -> 10 -> 19 -> 28 | | | | V V V V 7
                 * 20 22 35 | | | V V V 8 50 40 | | V V 30 45
                 */

                L.head = L.push(L.head, 30);
                L.head = L.push(L.head, 8);
                L.head = L.push(L.head, 7);
                L.head = L.push(L.head, 5);

                L.head.right = L.push(L.head.right, 20);
                L.head.right = L.push(L.head.right, 10);

                L.head.right.right = L.push(L.head.right.right, 50);
                L.head.right.right = L.push(L.head.right.right, 22);
                L.head.right.right = L.push(L.head.right.right, 19);

                L.head.right.right.right = L.push(L.head.right.right.right, 45);
                L.head.right.right.right = L.push(L.head.right.right.right, 40);
                L.head.right.right.right = L.push(L.head.right.right.right, 35);
                L.head.right.right.right = L.push(L.head.right.right.right, 20);

                // Function call
                L.head = L.flatten(L.head);

                L.printList();
        }
}
```

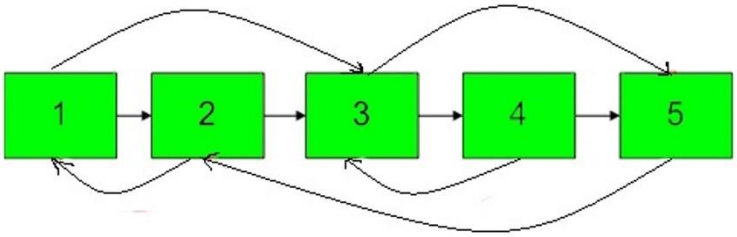
Question 4

You are given a special linked list with N nodes where each node has a next pointer pointing to its next node. You are also given M random pointers, where you will be given M number of pairs denoting two nodes a and b i.e. a->arb = b (arb is pointer to random node).

Construct a copy of the given list. The copy should consist of exactly N new nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. None of the pointers in the new list should point to nodes in the original list.

For example, if there are two nodes X and Y in the original list, where X.arb --> Y, then for the corresponding two nodes x and y in the copied list, x.arb --> y.

Return the head of the copied linked list.



Note :- The diagram isn't part of any example, it just depicts an example of how the linked list may look like.

Example 1:
Input: N = 4, M = 2
value = {1,2,3,4}
pairs = {{1,2},{2,4}}
Output: 1
Explanation: In this test case, there are 4 nodes in linked list. Among these 4 nodes, 2 nodes have arbitrary pointer set, rest two nodes have arbitrary pointer as NULL. Second line tells us the value of four nodes. The third line gives the information about arbitrary pointers. The first node arbitrary pointer is set to node 2. The second node arbitrary pointer is set to node 4.

Example 2:
Input: N = 4, M = 2
value[] = {1,3,5,9}
pairs[] = {{1,1},{3,4}}
Output:1
Explanation: In the given testcase ,applying the method as stated in the above example, the output will be 1.

Solution Code :

```
package in.ineuron.pptAssignment14;

//Java code to implement the approach

import java.util.HashMap;

class Node04 {
	int val;
	Node04 next;
	Node04 arbit;

//Constructor
	Node04(int x) {
		this.val = x;
		this.next = null;
		this.arbit = null;
	}
}

public class CopyLinkedListWithRandomPointers_4 {

	static Node04 cloneLinkedList(Node04 head) {
		// Map to store the mapping of
		// old nodes with new ones
		HashMap<Node04, Node04> mp = new HashMap<>();
		Node04 temp, nhead;

		// Duplicate of the first node
		temp = head;
		nhead = new Node04(temp.val);
		mp.put(temp, nhead);

		// Loop to create duplicates of nodes
		// with only next pointer
		while (temp.next != null) {
			nhead.next = new Node04(temp.next.val);
			temp = temp.next;
			nhead = nhead.next;
			mp.put(temp, nhead);
		}
		temp = head;

		// Loop to clone the arbit pointers
		while (temp != null) {
			mp.get(temp).arbit = mp.get(temp.arbit);
```

```
                temp = temp.next;
            }

            // Return the head of the clone
            return mp.get(head);
        }

        static void printList(Node04 head) {
            System.out.print(head.val + "(" + head.arbit.val + ")");
            head = head.next;
            while (head != null) {
                System.out.print(" -> " + head.val + "(" + head.arbit.val + ")");
                head = head.next;
            }
            System.out.println();
        }

        public static void main(String[] args) {
            // Creating a linked list with random pointer
            Node04 head = new Node04(1);
            head.next = new Node04(2);
            head.next.next = new Node04(3);
            head.next.next.next = new Node04(4);
            head.next.next.next.next = new Node04(5);
            head.arbit = head.next.next;
            head.next.arbit = head;
            head.next.next.arbit = head.next.next.next.next;
            head.next.next.next.arbit = head.next.next;
            head.next.next.next.next.arbit = head.next;

            // Print the original list
            System.out.println("The original linked list:");
            printList(head);

            // Function call
            Node04 sol = cloneLinkedList(head);

            System.out.println("The cloned linked list:");
            printList(sol);
        }
    }
```

Question 5

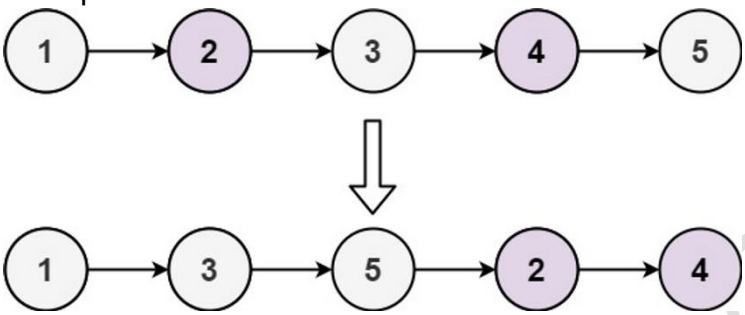
Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.

The first node is considered odd, and the second node is even, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

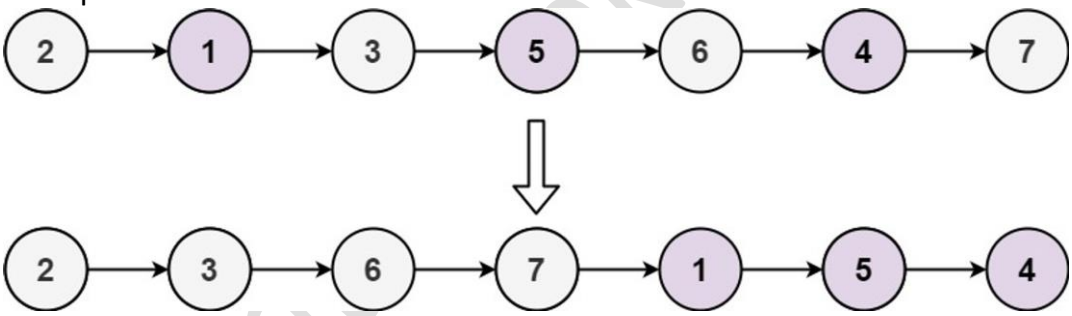
You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:



Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]

Example 2:



Input: head = [2,1,3,5,6,4,7]
Output: [2,3,6,7,1,5,4]

Solution Code:

```
package in.ineuron.pptAssignment14;

//Java program to rearrange a linked list
//in such a way that all odd positioned node are stored before all even positioned nodes
public class RearrangeLinkedList_5 {

//Linked List Node
	static class Node {
		int data;
		Node next;
	}
```

```
	//A utility function to create a new node
		static Node newNode(int key) {
			Node temp = new Node();
			temp.data = key;
			temp.next = null;
			return temp;
		}

	//Rearranges given linked list
	//such that all even positioned
	//nodes are before odd positioned.
	//Returns new head of linked List.
		static Node rearrangeEvenOdd(Node head) {
			// Corner case
			if (head == null)
				return null;

			// Initialize first nodes of even and
			// odd lists
			Node odd = head;
			Node even = head.next;

			// Remember the first node of even list so
			// that we can connect the even list at the
			// end of odd list.
			Node evenFirst = even;

			while (1 == 1) {
				// If there are no more nodes,
				// then connect first node of even
				// list to the last node of odd list
				if (odd == null || even == null || (even.next) == null) {
					odd.next = evenFirst;
					break;
				}

				// Connecting odd nodes
				odd.next = even.next;
				odd = even.next;

				// If there are NO more even nodes
				// after current odd.
				if (odd.next == null) {
					even.next = null;
					odd.next = evenFirst;
					break;
				}
```

```
                // Connecting even nodes
                even.next = odd.next;
                even = odd.next;
            }
            return head;
        }

    //A utility function to print a linked list
        static void printlist(Node node) {
            while (node != null) {
                System.out.print(node.data + "->");
                node = node.next;
            }
            System.out.println("NULL");
        }

    //Driver code
        public static void main(String[] args) {
            Node head = newNode(1);
            head.next = newNode(2);
            head.next.next = newNode(3);
            head.next.next.next = newNode(4);
            head.next.next.next.next = newNode(5);

            System.out.println("Given Linked List");
            printlist(head);

            head = rearrangeEvenOdd(head);

            System.out.println("Modified Linked List");
            printlist(head);
        }
    }
```

Question 6

Given a singly linked list of size N. The task is to left-shift the linked list by k nodes, where k is a given positive integer smaller than or equal to length of the linked list.

Example 1:

Input: N = 5

value[] = {2, 4, 7, 8, 9}

k = 3

Output:8 9 2 4 7

Explanation: Rotate 1:4 -> 7 -> 8 -> 9 -> 2

Rotate 2: 7 -> 8 -> 9 -> 2 -> 4

Rotate 3: 8 -> 9 -> 2 -> 4 -> 7

Example 2:

Input: N = 8

value[] = {1, 2, 3, 4, 5, 6, 7, 8}

k = 4

Output:5 6 7 8 1 2 3 4

Solution Code :

```
package in.ineuron.pptAssignment14;

//Java program to rotate a linked list

class Node06 {
	int data;
	Node06 next;

	Node06(int data) {
		this.data = data;
		next = null;
	}
}

public class RotateLinkedList_6 {
	Node06 head;

	void addNode(int data) {
		Node06 newNode = new Node06(data);
		if (head == null) {
			head = newNode;
		} else {
			Node06 curr = head;
			while (curr.next != null) {
				curr = curr.next;
			}
			curr.next = newNode;
		}
```

```
    }

    void leftShift(int k) {
        if (head == null || k <= 0) {
            return;
        }

        int length = getLength();
        k = k % length; // Adjust k if it is greater than the length of the linked list

        if (k == 0) {
            return; // No need to shift
        }

        Node06 curr = head;
        Node06 newHead = null;
        Node06 tail = null;

        // Traverse to the kth node from the beginning
        for (int i = 1; i < k && curr != null; i++) {
            curr = curr.next;
        }

        if (curr == null) {
            return; // Invalid k value
        }

        newHead = curr.next;
        curr.next = null;
        tail = newHead;

        // Traverse to the last node
        while (tail.next != null) {
            tail = tail.next;
        }

        // Connect the last node to the original head
        tail.next = head;
        head = newHead;
    }

    int getLength() {
        int length = 0;
        Node06 curr = head;
        while (curr != null) {
            length++;
            curr = curr.next;
```

```
            }
            return length;
        }

        void printList() {
            Node06 curr = head;
            while (curr != null) {
                System.out.print(curr.data + " ");
                curr = curr.next;
            }
            System.out.println();
        }

        public static void main(String[] args) {
            RotateLinkedList_6 list = new RotateLinkedList_6();
            list.addNode(2);
            list.addNode(4);
            list.addNode(7);
            list.addNode(8);
            list.addNode(9);

            int k = 3;

            // Left-shift the linked list by k nodes
            list.leftShift(k);

            // Print the modified linked list
            list.printList();
        }
    }
```

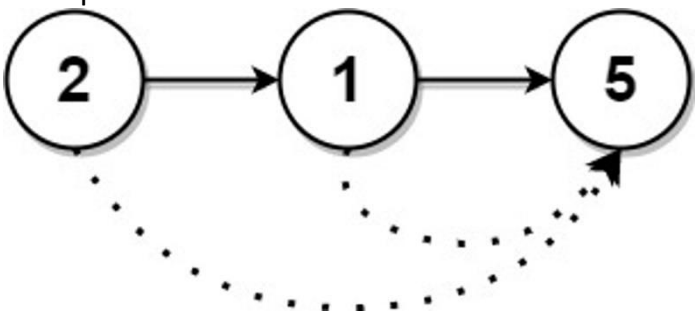
Question 7

You are given the `head` of a linked list with `n` nodes.

For each node in the list, find the value of the next greater node. That is, for each node, find the value of the first node that is next to it and has a strictly larger value than it.

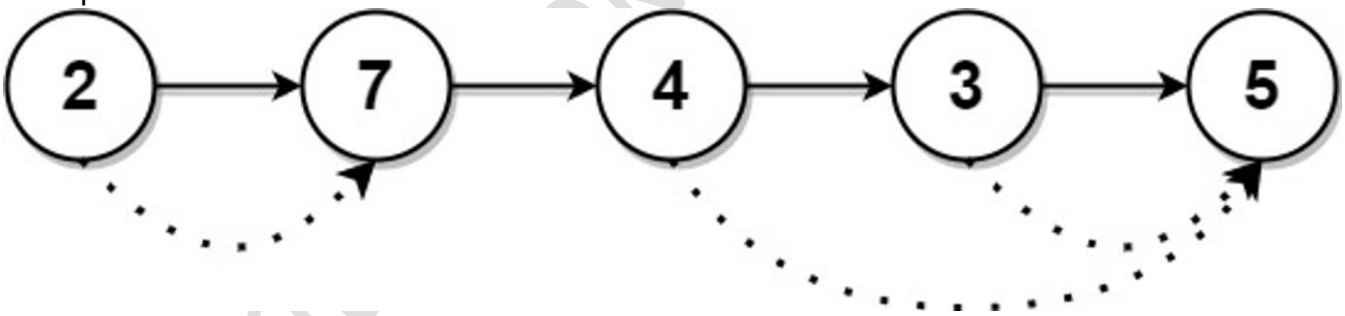
Return an integer array `answer` where `answer[i]` is the value of the next greater node of the `ith` node (1-indexed). If the `ith` node does not have a next greater node, set `answer[i] = 0`.

Example 1:



Input: head = [2,1,5]
Output: [5,5,0]

Example 2:



Input: head = [2,7,4,3,5]
Output: [7,0,5,5,0]

Solution Code:

```
package in.ineuron.pptAssignment14;

import java.util.*;

class ListNode {
	int val;
	ListNode next;

	ListNode(int val) {
		this.val = val;
		this.next = null;
	}
}
```

```
class Solution {
    public int[] nextLargerNodes(ListNode head) {
        List<Integer> values = new ArrayList<>();
        ListNode curr = head;

        // Extract the values from the linked list
        while (curr != null) {
            values.add(curr.val);
            curr = curr.next;
        }

        int[] result = new int[values.size()];
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < values.size(); i++) {
            int currValue = values.get(i);

            // Process nodes with larger values
            while (!stack.isEmpty() && currValue > values.get(stack.peek())) {
                int index = stack.pop();
                result[index] = currValue;
            }

            stack.push(i);
        }

        return result;
    }
}

public class NextLargerNodes_6 {
    public static void main(String[] args) {
        ListNode head = new ListNode(2);
        head.next = new ListNode(1);
        head.next.next = new ListNode(5);

        Solution solution = new Solution();
        int[] result = solution.nextLargerNodes(head);

        // Print the result
        for (int val : result) {
            System.out.print(val + " ");
        }
        System.out.println();
    }
}
```

Question 8

Given the `head` of a linked list, we repeatedly delete consecutive sequences of nodes that sum to `0` until there are no such sequences.

After doing so, return the head of the final linked list. You may return any such answer.

(Note that in the examples below, all sequences are serializations of `ListNode` objects.)

Example 1:
Input: head = [1,2,-3,3,1]
Output: [3,1]
Note: The answer [1,2,1] would also be accepted.

Example 2:
Input: head = [1,2,3,-3,4]
Output: [1,2,4]

Example 3:
Input: head = [1,2,3,-3,-2]
Output: [1]

Solution Code:

```
package in.ineuron.pptAssignment14;

import java.util.HashMap;
import java.util.Map;

class ListNode08 {
	int val;
	ListNode08 next;

	ListNode08(int val) {
		this.val = val;
		this.next = null;
	}
}

class Solution08 {
	public ListNode08 removeZeroSumSublists(ListNode08 head) {
		ListNode08 dummy = new ListNode08(0);
		dummy.next = head;
		ListNode08 curr = dummy;
		int sum = 0;
		Map<Integer, ListNode08> map = new HashMap<>();

		while (curr != null) {
```

```
                sum += curr.val;

                if (map.containsKey(sum)) {
                    ListNode08 prev = map.get(sum).next;
                    int tempSum = sum + prev.val;
                    while (prev != curr) {
                        map.remove(tempSum);
                        prev = prev.next;
                        tempSum += prev.val;
                    }
                    map.get(sum).next = curr.next;
                } else {
                    map.put(sum, curr);
                }

                curr = curr.next;
            }

            return dummy.next;
        }
    }

    public class RemoveZeroSumSublists_8 {
        public static void main(String[] args) {
            ListNode08 head = new ListNode08(1);
            head.next = new ListNode08(2);
            head.next.next = new ListNode08(-3);
            head.next.next.next = new ListNode08(3);
            head.next.next.next.next = new ListNode08(1);

            Solution08 solution = new Solution08();
            ListNode08 result = solution.removeZeroSumSublists(head);

            // Print the result
            while (result != null) {
                System.out.print(result.val + " ");
                result = result.next;
            }
            System.out.println();
        }
    }
```

GITHUB link:
https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-14