Assignment_04_ADS

1.Doubly Linked List Insertion in java

```
class Node {
    int data;
    Node prev;
    Node next;

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

class DoublyLinkedList {
    Node head;

    public DoublyLinkedList() {
        this.head = null;
    }



    public void insertAtBeginning(int data) {
        Node newNode = new Node(data);
```

```
    if (head == null) {
        head = newNode;
    } else {
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }
}

// Method to insert a new node at the end of the doubly linked list
public void insertAtEnd(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 printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insertAtBeginning(5);
        dll.insertAtEnd(10);
        dll.insertAtEnd(15);
        dll.insertAtBeginning(2);
        dll.printList();
    }
}
```

2.Reverse a Doubly Linked List in java

Assignment_04_ADS

```
class Node {
    int data;
    Node prev;
    Node next;

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

class DoublyLinkedList {
    Node head;



    public DoublyLinkedList() {
        this.head = null;
    }

    public void insertAtEnd(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 reverse() {
    Node current = head;
    Node temp = null;



    while (current != null) {
        temp = current.prev;
        current.prev = current.next;
        current.next = temp;
        current = current.prev;
    }



    if (temp != null) {
```

```
            head = temp.prev;
        }
    }



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

public class Main {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insertAtEnd(5);
        dll.insertAtEnd(10);
        dll.insertAtEnd(15);

        System.out.println("Original list:");
        dll.printList();
```

```
        dll.reverse();

        System.out.println("Reversed list:");
        dll.printList();
    }
}
```

3.Delete a node in a Doubly Linked List in java

```
class Node {
    int data;
    Node prev;
    Node next;

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

class DoublyLinkedList {
    Node head;
```

```
public DoublyLinkedList() {
   this.head = null;
}



public void insertAtEnd(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 deleteNode(int key) {
   Node temp = head;
```

```
    if (temp != null && temp.data == key) {
        head = temp.next;
        head.prev = null;
        return;
    }
    while (temp != null && temp.data != key) {
        temp = temp.next;
    }


    if (temp == null) {
        return;
    }


    if (temp.next != null) {
        temp.next.prev = temp.prev;
    }

    if (temp.prev != null) {
        temp.prev.next = temp.next;
    }
}
```

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

public class Main {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insertAtEnd(5);
        dll.insertAtEnd(10);
        dll.insertAtEnd(15);

        System.out.println("Original list:");
        dll.printList();

        dll.deleteNode(10);

        System.out.println("List after deleting node with value 10:");
```

```
        dll.printList();
    }
}
```

4.Program to find length of Doubly Linked List in java

```
class Node {
    int data;
    Node prev;
    Node next;

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

class DoublyLinkedList {
    Node head;
    public DoublyLinkedList() {
        this.head = null;
    }
```

```
public void insertAtEnd(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;
    }
}
```

```
    int count = 0;
    Node current = head;
    while (current != null) {
        count++;
        current = current.next;
    }
    return count;
}
```

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

public class Main {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insertAtEnd(5);
        dll.insertAtEnd(10);
        dll.insertAtEnd(15);

        System.out.println("Original list:");
        dll.printList();
        int length = dll.length();
        System.out.println("Length of the list: " + length);
    }
```

```
}
```

7.Write tree traversals in java

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class TreeTraversal {

    public static void preorder(TreeNode root) {
        if (root == null) return;
        System.out.print(root.val + " ");
        preorder(root.left);
        preorder(root.right);
    }
```

```
public static void inorder(TreeNode root) {
    if (root == null) return;
    inorder(root.left);
    System.out.print(root.val + " ");
    inorder(root.right);
}



public static void postorder(TreeNode root) {
    if (root == null) return;
    postorder(root.left);
    postorder(root.right);
    System.out.print(root.val + " ");
}



public static void levelOrder(TreeNode root) {
    if (root == null) return;
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    while (!queue.isEmpty()) {
        TreeNode node = queue.poll();
        System.out.print(node.val + " ");
```

```
            if (node.left != null) queue.offer(node.left);
            if (node.right != null) queue.offer(node.right);
        }
    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);

        System.out.println("Preorder traversal:");
        preorder(root);
        System.out.println("\nInorder traversal:");
        inorder(root);
        System.out.println("\nPostorder traversal:");
        postorder(root);
        System.out.println("\nLevel order traversal:");
        levelOrder(root);
    }
}
```

8.Search a node in Binary Tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {

    public static TreeNode search(TreeNode root, int target) {
        if (root == null || root.val == target) {
            return root;
        }


        TreeNode leftResult = search(root.left, target);
        if (leftResult != null) {
```

```
        return leftResult;
    }



    TreeNode rightResult = search(root.right, target);
    return rightResult;
}

public static void main(String[] args) {

    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.left = new TreeNode(4);
    root.left.right = new TreeNode(5);
    root.right.left = new TreeNode(6);
    root.right.right = new TreeNode(7);

    int target = 5;
    TreeNode result = search(root, target);

    if (result != null) {
        System.out.println("Node " + target + " found in the binary tree.");
    } else {
```

```
            System.out.println("Node " + target + " not found in the binary tree.");
        }
    }
}
```

9.Inorder Successor of a node in Binary Tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode parent;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
        this.parent = null;
    }
}

public class BinaryTree {
```

```
public static TreeNode inorderSuccessor(TreeNode root, TreeNode node) {
   if (node == null)
      return null;



   if (node.right != null) {
      return minValue(node.right);
   }



   TreeNode parent = node.parent;
   while (parent != null && node == parent.right) {
      node = parent;
      parent = parent.parent;
   }
   return parent;
}



private static TreeNode minValue(TreeNode node) {
   TreeNode current = node;
   while (current.left != null) {
      current = current.left;
   }
```

```
    return current;
}

public static void main(String[] args) {

    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.left = new TreeNode(4);
    root.left.right = new TreeNode(5);
    root.right.left = new TreeNode(6);
    root.right.right = new TreeNode(7);


    root.parent = null;
    root.left.parent = root;
    root.right.parent = root;
    root.left.left.parent = root.left;
    root.left.right.parent = root.left;
    root.right.left.parent = root.right;
    root.right.right.parent = root.right;


    TreeNode node = root.left;
```

```
        TreeNode successor = inorderSuccessor(root, node);


        if (successor != null) {
            System.out.println("Inorder successor of node " + node.val + " is " + successor.val);
        } else {
            System.out.println("Node " + node.val + " does not have an inorder successor.");
        }
    }
}
```