Assignment 4

----------------------

----------------------

1.Doubly Linked List Insertion in java

---------------------------------------

class DoublyNode {

int data;

DoublyNode prev;

DoublyNode next;

DoublyNode(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

class DoublyLinkedList {

DoublyNode head;

DoublyLinkedList() {

this.head = null;

}

// Method to insert a node at the end of the linked list

public void insert(int data) {

DoublyNode newNode = new DoublyNode(data);

if (head == null) {

head = newNode;

} else {

DoublyNode temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

newNode.prev = temp;

}

}

// Method to display the linked list

public void display() {

DoublyNode temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

---------------------------------------

---------------------------------------

2.Reverse a Doubly Linked List in java

---------------------------------------

class ReverseDoublyLinkedList {

public static DoublyNode reverse(DoublyNode head) {

DoublyNode temp = null;

DoublyNode current = head;

while (current != null) {

temp = current.prev;

current.prev = current.next;

current.next = temp;

current = current.prev;

}

if (temp != null) {

head = temp.prev;

}

return head;

}

}

---------------------------------------

---------------------------------------

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

---------------------------------------

class DeleteNode {

public static DoublyNode delete(DoublyNode head, int key) {

DoublyNode temp = head;

if (head == null) return null;

if (head.data == key) {

head = head.next;

head.prev = null;

return head;

}

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

temp = temp.next;

}

if (temp == null) return head;

if (temp.next != null) {

temp.next.prev = temp.prev;

}

if (temp.prev != null) {

temp.prev.next = temp.next;

}

return head;

}

}

---------------------------------------

---------------------------------------

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

---------------------------------------

class LengthDoublyLinkedList {

public static int length(DoublyNode head) {

int len = 0;

DoublyNode temp = head;

while (temp != null) {

len++;

temp = temp.next;

}

return len;

}

}

---------------------------------------

---------------------------------------

5.Find the largest node in Doubly linked list in java

---------------------------------------

class LargestNode {

public static int findLargest(DoublyNode head) {

int max = Integer.MIN\_VALUE;

DoublyNode temp = head;

while (temp != null) {

if (temp.data > max) {

max = temp.data;

}

temp = temp.next;

}

return max;

}

}

---------------------------------------

---------------------------------------

6.Insert value in sorted way in a sorted doubly linked list in java

---------------------------------------

class InsertSorted {

public static DoublyNode insertSorted(DoublyNode head, int data) {

DoublyNode newNode = new DoublyNode(data);

if (head == null) {

return newNode;

}

if (data < head.data) {

newNode.next = head;

head.prev = newNode;

return newNode;

}

DoublyNode current = head;

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

current = current.next;

}

newNode.next = current.next;

if (current.next != null) {

current.next.prev = newNode;

}

current.next = newNode;

newNode.prev = current;

return head;

}

}

---------------------------------------

---------------------------------------

7.Write tree traversals in java

---------------------------------------

class TreeNode {

int data;

TreeNode left;

TreeNode right;

TreeNode(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

class TreeTraversal {

public static void inorder(TreeNode root) {

if (root != null) {

inorder(root.left);

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

inorder(root.right);

}

}

public static void preorder(TreeNode root) {

if (root != null) {

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

preorder(root.left);

preorder(root.right);

}

}

public static void postorder(TreeNode root) {

if (root != null) {

postorder(root.left);

postorder(root.right);

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

}

}

}

---------------------------------------

---------------------------------------

8.Search a node in Binary Tree

---------------------------------------

class SearchNode {

public static boolean search(TreeNode root, int key) {

if (root == null) {

return false;

}

if (root.data == key) {

return true;

}

return search(root.left, key) || search(root.right, key);

}

}

---------------------------------------

---------------------------------------

9.Inorder Successor of a node in Binary Tree

---------------------------------------

class InorderSuccessor {

public static TreeNode inorderSuccessor(TreeNode root, TreeNode node) {

if (node.right != null) {

return minValue(node.right);

}

TreeNode successor = null;

while (root != null) {

if (node.data < root.data) {

successor = root;

root = root.left;

} else if (node.data > root.data) {

root = root.right;

} else {

break;

}

}

return successor;

}

private static TreeNode minValue(TreeNode node) {

TreeNode current = node;

while (current.left != null) {

current = current.left;

}

return current;

}

}

---------------------------------------

---------------------------------------

10.Print Head node of every node in Binary Tree

---------------------------------------

class PrintHeadNode {

public static void printHeadNodes(TreeNode root) {

if (root != null) {

System.out.println("Head Node: " + root.data);

printHeadNodes(root.left);

printHeadNodes(root.right);

}

}

}

---------------------------------------

---------------------------------------