ASSIGNMENT 4

Q1. Doubly Linked List Insertion in java

class DLL{

Node head;

static class Node{

int data;

Node prev;

Node next;

Node(int d)

{

data = d;

next = null;

prev = null;

}

}

void insert(int new\_data)

{

Node new\_node = new Node(new\_data);

new\_node.next = head;

new\_node.prev = null;

if( head != null)

head.prev = new\_node;

head = new\_node;

}

void insertAfter(Node prev, int new\_data)

{

if(prev == null)

return;

Node new\_node = new Node(new\_data);

new\_node.next = prev.next;

prev.next = new\_node;

new\_node.prev = prev;

Node p = new\_node.next;

p.prev = new\_node;

}

void display(Node n)

{

Node p=null;

System.out.println("Forward printing:");

while(n != null)

{

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

p=n;

n=n.next;

}

System.out.println("--------");

System.out.println("Backward printing:");

while(p != null)

{

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

p=p.prev;

}

}

public static void main(String args[])

{

DLL d1 = new DLL();

d1.insert(5);

d1.insert(10);

d1.insert(15);

d1.display(d1.head);

System.out.println();

d1.insertAfter(d1.head, 7);

d1.display(d1.head);

}

}

OUPUT:-

Forward printing:

15 10 5 --------

Backward printing:

5 10 15 --------

Forward printing:

15 7 10 5 --------

Backward printing:

5 10 7 15 --------

Q2. Reverse 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;

}

}

public class DoublyLinkedList {

Node head;

Node tail;

public DoublyLinkedList() {

this.head = null;

this.tail = null;

}

public void insertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

}

}

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; // Move to the next node in the original order

}

if (temp != null) {

head = temp.prev; // Update the head to the new first node

}

}

public void printForward() {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtEnd(4);

list.insertAtEnd(5);

System.out.println("Original doubly linked list from head to tail:");

list.printForward();

list.reverse();

System.out.println("Reversed doubly linked list from head to tail:");

list.printForward();

}

}

OUTPUT :-

Original doubly linked list from head to tail:

1 2 3 4 5

Reversed doubly linked list from head to tail:

5 4 3 2 1

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

public void deleteNode(int key) {

Node current = head;

if (head != null && head.data == key) {

head = head.next;

if (head != null) {

head.prev = null;

}

return;

}

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

current = current.next;

}

if (current == null) {

System.out.println("Node with key " + key + " not found.");

return;

} if (current.prev != null) {

current.prev.next = current.next;

}

if (current.next != null) {

current.next.prev = current.prev;

} if (current == tail) {

tail = current.prev;

}

current = null; }

public void printForward() {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

OUTPUT:-

Original list:

15 10 5

Deleting node with data 10

List after deletion:

15 5

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

class DLL {

Node head;

static class Node {

int data;

Node prev;

Node next;

Node(int d) {

data = d;

next = null;

prev = null;

}

}

void insert(int new\_data) {

Node new\_node = new Node(new\_data);

new\_node.next = head;

new\_node.prev = null;

if (head != null)

head.prev = new\_node;

head = new\_node;

}

int length() {

int len = 0;

Node temp = head;

while (temp != null) {

len++;

temp = temp.next;

}

return len;

}

void display() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String args[]) {

DLL d1 = new DLL();

d1.insert(5);

d1.insert(10);

d1.insert(15);

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

d1.display();

int length = d1.length();

System.out.println("Length of the doubly linked list: " + length);

}

}

OUTPUT:-

Original list:

15 10 5

Length of the doubly linked list: 3

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

class DLL {

Node head;

static class Node {

int data;

Node prev;

Node next;

Node(int d) {

data = d;

next = null;

prev = null;

}

}

void insert(int new\_data) {

Node new\_node = new Node(new\_data);

new\_node.next = head;

new\_node.prev = null;

if (head != null)

head.prev = new\_node;

head = new\_node;

}

Node findLargest() {

if (head == null)

return null;

Node current = head;

Node largest = head;

while (current != null) {

if (current.data > largest.data) {

largest = current;

}

current = current.next;

}

return largest;

}

void display() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String args[]) {

DLL d1 = new DLL();

d1.insert(5);

d1.insert(10);

d1.insert(15);

d1.insert(7);

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

d1.display();

Node largestNode = d1.findLargest();

if (largestNode != null) {

System.out.println("Larg

est node in the doubly linked list: " + largestNode.data);

} else {

System.out.println("Doubly linked list is empty.");

}

}

}

OUTPUT:-

Original list:

7 15 10 5

Largest node in the doubly linked list: 15

Q 6.Insert value in sorted way in a sorted 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;

}

}

public class SortedDoublyLinkedList {

Node head;

Node tail;

public SortedDoublyLinkedList() {

this.head = null;

this.tail = null;

}

public void insertSorted(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

return;

}

if (value <= head.data) {

newNode.next = head;

head.prev = newNode;

head = newNode;

return;

}

Node current = head;

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

current = current.next;

}

if (current == null) { tail.next = newNode;

newNode.prev = tail;

tail = newNode;

} else {

newNode.next = current;

newNode.prev = current.prev;

current.prev.next = newNode;

current.prev = newNode;

}

}

public void printForward() {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

SortedDoublyLinkedList list = new SortedDoublyLinkedList();

list.insertSorted(5);

list.insertSorted(2);

list.insertSorted(9);

list.insertSorted(1);

list.insertSorted(7);

System.out.println("Sorted doubly linked list from head to tail:");

list.printForward();

}

}

OUTPUT:-

Sorted doubly linked list from head to tail:

1 2 5 7 9

Q 7. Write tree traversals in java

class BT1{

Node root;

static class Node{

int data;

Node left,right;

Node(int d){

data = d;

left = right= null;

}

}

BT1(){

root = null;

}

BT1(int d){

root = new Node(d);

}

void printInorder(Node n)

{

if(n == null)

return;

printInorder(n.left);

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

printInorder(n.right);

}

void inorder()

{

printInorder(root);

}

void printPreorder(Node n)

{

if(n == null)

return;

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

printPreorder(n.left);

printPreorder(n.right);

}

void preorder()

{

printPreorder(root);

}

void printPostorder(Node n)

{

if(n == null)

return;

printPostorder(n.left);

printPostorder(n.right);

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

}

void postorder()

{

printPostorder(root);

}

public static void main(String args[])

{

BT1 t1 = new BT1();

t1.root = new Node(20);

t1.root.left = new Node(25);

t1.root.right = new Node(56);

t1.root.right.right = new Node(12);

System.out.println("Inorder---->");

t1.inorder();

System.out.println();

System.out.println("Preorder---->");

t1.preorder();

System.out.println();

System.out.println("Postorder---->");

t1.postorder();

}

}

OUTPUT :-

Inorder---->

25 20 56 12

Preorder---->

20 25 56 12

Postorder---->

25 12 56 20

Q 8. Search a node in Binary Tree

public boolean search(TreeNode root, int target) {

if (root == null) {

return false; // Empty tree

}

if (root.val == target) {

return true; }

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

}

OUTPUT:-

Node 3 found in the binary tree.

Q 9. Inorder Successor of a node in Binary Tree

public TreeNode inorderSuccessor(TreeNode root, TreeNode target) {

if (root == null || target == null) {

return null; }

if (target.right != null) {

TreeNode successor = target.right;

while (successor.left != null) {

successor = successor.left;

}

return successor;

}

TreeNode successor = null;

TreeNode current = root;

while (current != null) {

if (target.val < current.val) {

successor = current;

current = current.left;

} else if (target.val > current.val) {

current = current.right;

} else {

break; }

}

return successor;

}

OUTPUT:-

Inorder successor of 15 is 20

Q 10. Print Head node of every 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 BinaryTreeHeadNodes {

public void printHeadNodes(TreeNode root) {

if (root == null) {

return;

}

System.out.println("Head node of subtree rooted at " + root.val + ": " + getHeadNode(root).val);

printHeadNodes(root.left);

printHeadNodes(root.right);

}

private TreeNode getHeadNode(TreeNode node) {

while (node.left != null || node.right != null) {

if (node.left != null) {

node = node.left;

} else {

node = node.right;

}

}

return node;

}

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);

BinaryTreeHeadNodes tree = new BinaryTreeHeadNodes();

tree.printHeadNodes(root);

}

}

OUTPUT:-

Head node of subtree rooted at 1: 4

Head node of subtree rooted at 2: 4

Head node of subtree rooted at 4: 4

Head node of subtree rooted at 5: 5

Head node of subtree rooted at 3: 6

Head node of subtree rooted at 6: 6

Head node of subtree rooted at 7: 7