1.Doubly Linked List Insertion in java

class DLL1{

Node head;

static class Node{

int data;

Node next,prev;

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 display(Node n)

{

Node p = null;

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

while(n != null)

{

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

p=n;

n=n.next;

}

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

System.out.println();

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

while(p != null)

{

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

p=p.prev;

}

}

public static void main(String args[])

{

DLL1 d1 = new DLL1();

d1.display(d1.head);

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

d1.insert(10);

d1.display(d1.head);

System.out.println();

d1.insert(5);

d1.display(d1.head);

}

}

Forward direction:

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

Backward direction:

------------------------------\*\*

Forward direction:

10---> ------------------------------

Backward direction:

10--->

Forward direction:

5---> 10---> ------------------------------

Backward direction:

10---> 5--->

2. Reverse a Doubly Linked List in java

class LinkedList {

static Node head;

static class Node {

int data;

Node next, prev;

Node(int d)

{

data = d;

next = prev = null;

}

}

void reverse()

{

Node temp = null;

Node current = head;

doubly linked list \*/

while (current != null) {

temp = current.prev;

current.prev = current.next;

current.next = temp;

current = current.prev;

}

if (temp != null) {

head = temp.prev;

}

}

void push(int new\_data)

{

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

prev is always NULL \*/

new\_node.prev = null;

new\_node.next = head;

if (head != null) {

head.prev = new\_node;

}

head = new\_node;

}

void printList(Node node)

{

while (node != null) {

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

node = node.next;

}

}

public static void main(String[] args)

{

LinkedList list = new LinkedList();

list.push(2);

list.push(4);

list.push(8);

list.push(10);

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

list.printList(head);

list.reverse();

System.out.println("");

System.out.println("The reversed Linked List is ");

list.printList(head);

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

Original linked list

10 8 4 2

The reversed Linked List is

2 4 8 10

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

class DLL1{

Node head;

static class Node{

int data;

Node prev;

Node next;

Node(int d)

{

data = d;

next = null;

prev = null;

}

}

void deletenode(Node n)

{

//base condition

if(head == null || n == null)

{

return;

}

//deletion at the begining

if(head == n)

head=n.next;

//head.prev=null;

// deletion in between two elements

if(n.next != null)

n.next.prev = n.prev;

if(n.prev != null)

n.prev.next=n.next;

return;

}

void display(Node n)

{

Node p = null;

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

while(n != null)

{

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

p=n;

n=n.next;

}

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

System.out.println();

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

while(p != null)

{

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

p=p.prev;

}

}

public static void main(String args[])

{

DLL1 d1 = new DLL1 ();

//d1.display(d1.head);

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

System.out.println();

d1.deletenode(d1.head.next);

d1.display(d1.head);

}}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java DLL1

------------------------------\*\*

Exception in thread "main" java.lang.NullPointerException

at DLL1.main(DLL1.java:69)

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

class Node

{

int data;

Node next, prev;

Node(int val)

{

data = val;

next = null;

prev = null;

}

}

class LinkedList

{

/\* Function to add a node to front of doubly

linked list \*/

static Node push(Node head, int data)

{

Node new\_node = new Node(data);

new\_node.next = head;

new\_node.prev = null;

if (head != null)

head.prev = new\_node;

head = new\_node;

return head;

}

// This function returns size of doubly linked list

static int findSize(Node node)

{

int res = 0;

while (node != null)

{

res++;

node = node.next;

}

return res;

}

public static void main(String args[])

{

Node head = null;

head = push(head, 4);

head = push(head, 3);

head = push(head, 2);

head = push(head, 1);

System.out.println(findSize(head));

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

4

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

class LinkedList

{

static class Node {

int data;

Node next;

Node prev;

};

// UTILITY FUNCTIONS

// Function to insert a node at the

// beginning of the Doubly Linked List

static Node push(Node head\_ref, int new\_data)

{

// allocate node

Node new\_node = new Node();

// put in the data

new\_node.data = new\_data;

// since we are adding at the

// beginning, prev is always null

new\_node.prev = null;

// link the old list of the new node

new\_node.next = (head\_ref);

// change prev of head node to new node

if ((head\_ref) != null)

(head\_ref).prev = new\_node;

// move the head to point to the new node

(head\_ref) = new\_node;

return head\_ref;

}

// Function to find the largest

// nodes in Doubly Linked List

static int LargestInDLL(Node head\_ref)

{

Node max, temp;

// initialize two pointer temp

// and max on the head node

temp = max = head\_ref;

// traverse the whole doubly linked list

while (temp != null) {

// if temp's data is greater than

// max's data, then put max = temp

// and return max.data

if (temp.data > max.data)

max = temp;

temp = temp.next;

}

return max.data;

}

public static void main(String args[])

{

// Start with the empty list

Node head = null;

// Let us create a linked list

head = push(head, 20);

head = push(head, 14);

head = push(head, 181);

head = push(head, 100);

System.out.printf("%d", LargestInDLL(head));

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

181

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

class Node

{

int data;

Node next, prev;

}

class LinkedList

{

// function to create and return a new node

// of a doubly linked list

static Node getNode(int data)

{

// allocate node

Node newNode = new Node();

// put in the data

newNode.data = data;

newNode.prev = newNode.next = null;

return newNode;

}

// function to insert a new node in sorted way in

// a sorted doubly linked list

static Node sortedInsert(Node head, Node newNode)

{

Node current;

// if list is empty

if (head == null)

head = newNode;

// if the node is to be inserted at the beginning

// of the doubly linked list

else if (head.data >= newNode.data)

{

newNode.next = head;

newNode.next.prev = newNode;

head = newNode;

}

else

{

current = head;

// locate the node after which the new node

// is to be inserted

while (current.next != null &&

current.next.data < newNode.data)

current = current.next;

/\* Make the appropriate links \*/

newNode.next = current.next;

// if the new node is not inserted

// at the end of the list

if (current.next != null)

newNode.next.prev = newNode;

current.next = newNode;

newNode.prev = current;

}

return head;

}

// function to print the doubly linked list

static void printList(Node head)

{

while (head != null)

{

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

head = head.next;

}

}

// Driver code

public static void main(String args[])

{

/\* start with the empty doubly linked list \*/

Node head = null;

// insert the following nodes in sorted way

Node new\_node = getNode(8);

head = sortedInsert(head, new\_node);

new\_node = getNode(5);

head = sortedInsert(head, new\_node);

new\_node = getNode(3);

head = sortedInsert(head, new\_node);

new\_node = getNode(10);

head = sortedInsert(head, new\_node);

new\_node = getNode(12);

head = sortedInsert(head, new\_node);

new\_node = getNode(9);

head = sortedInsert(head, new\_node);

System.out.println("Created Doubly Linked List");

printList(head);

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

Created Doubly Linked List

3 5 8 9 10 12

7.Write tree traversals in java

class Node {

int key;

Node left, right;

public Node(int item)

{

key = item;

left = right = null;

}

}

class LinkedList {

// Root of Binary Tree

Node root;

LinkedList() { root = null; }

// Given a binary tree, print its nodes in inorder

void printInorder(Node node)

{

if (node == null)

return;

// First recur on left child

printInorder(node.left);

// Then print the data of node

System.out.print(node.key + " ");

// Now recur on right child

printInorder(node.right);

}

// Driver code

public static void main(String[] args)

{

LinkedList tree = new LinkedList();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

tree.root.left.left = new Node(4);

tree.root.left.right = new Node(5);

// Function call

System.out.println(

"Inorder traversal of binary tree is ");

tree.printInorder(tree.root);

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

Inorder traversal of binary tree is

4 2 5 1 3

8. Search a node in Binary Tree

class LinkedList

{

static class Node

{

int data;

Node left, right;

Node(int data)

{

this.data = data;

left = right = null;

}

};

static boolean ifNodeExists( Node node, int key)

{

if (node == null)

return false;

if (node.data == key)

return true;

boolean res1 = ifNodeExists(node.left, key);

if(res1) return true;

boolean res2 = ifNodeExists(node.right, key);

return res2;

}

public static void main(String args[])

{

Node root = new Node(0);

root.left = new Node(1);

root.left.left = new Node(3);

root.left.left.left = new Node(7);

root.left.right = new Node(4);

root.left.right.left = new Node(8);

root.left.right.right = new Node(9);

root.right = new Node(2);

root.right.left = new Node(5);

root.right.right = new Node(6);

int key = 4;

if (ifNodeExists(root, key))

System.out.println("Node is present");

else

System.out.println("NOde is not present");

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

Node is present

9.Inorder Successor of a node in Binary Tree

class Node {

int data;

Node left, right, parent;

Node(int d)

{

data = d;

left = right = parent = null;

}

}

class LinkedList {

static Node head;

Node insert(Node node, int data)

{

if (node == null) {

return (new Node(data));

}

else {

Node temp = null;

if (data <= node.data) {

temp = insert(node.left, data);

node.left = temp;

temp.parent = node;

}

else {

temp = insert(node.right, data);

node.right = temp;

temp.parent = node;

}

return node;

}

}

Node inOrderSuccessor(Node root, Node n)

{

if (n.right != null) {

return minValue(n.right);

}

Node p = n.parent;

while (p != null && n == p.right) {

n = p;

p = p.parent;

}

return p;

}

Node minValue(Node node)

{

Node current = node;

while (current.left != null) {

current = current.left;

}

return current;

}

public static void main(String[] args)

{

LinkedList tree = new LinkedList();

Node root = null, temp = null, suc = null, min = null;

root = tree.insert(root, 20);

root = tree.insert(root, 8);

root = tree.insert(root, 22);

root = tree.insert(root, 4);

root = tree.insert(root, 12);

root = tree.insert(root, 10);

root = tree.insert(root, 14);

temp = root.left.right.right;

suc = tree.inOrderSuccessor(root, temp);

if (suc != null) {

System.out.println(

"Inorder successor of "

+ temp.data + " is " + suc.data);

}

else {

System.out.println(

"Inorder successor does not exist");

}

}

}

C:\Users\mujah\Downloads\Algorithm and Data Structure>java LinkedList

Inorder successor of 14 is 20

10.Print Head node of every node in Binary Tree