Assignment 5

Some basic Question of linkedList

Q.1 : Insert the element at beginning of the linked list

Adding nodes to the start of the list: 1

Adding nodes to the start of the list: 2 1

Adding nodes to the start of the list: 3 2 1

Adding nodes to the start of the list: 4 3 2 1

Sol :

Code - >

```
public class InsertBeginLinkedList {
   Node head;
   class Node
   {
      int data;
      Node next;
      Node(int data) {this.data=data; next = null; }
   }
   public void insertAtBegin(int x)
   {
      Node newNode = new Node(x);
      newNode.next = head;
      head = newNode;
   }

   public void print()
   {
      Node temp = head;
      while (temp != null)
      {
         System.out.print(temp.data+" ");
         temp = temp.next;
      }
      System.out.println();
   }
   public static void main(String[] args) {
      InsertBeginLinkedList node = new InsertBeginLinkedList();
      node.insertAtBegin(1);
      System.out.print("Adding nodes to the start of list : ");
      node.print();
      node.insertAtBegin(2);
      System.out.print("Adding nodes to the start of list : ");
      node.print();
```

```
        node.insertAtBegin(3);
        System.out.print("Adding nodes to the start of list : ");
        node.print();
        node.insertAtBegin(4);
        System.out.print("Adding nodes to the start of list : ");
        node.print();
    }
}
```

Output ->

Adding nodes to the start of list : 1

Adding nodes to the start of list : 2 1

Adding nodes to the start of list : 3 2 1

Adding nodes to the start of list : 4 3 2 1

Process finished with exit code 0

Q.2 : Insert the element at end of the linked list

Adding nodes to the End of the list: 1

Adding nodes to the End of the list: 1 2

Adding nodes to the End of the list: 1 2 3

Adding nodes to the End of the list: 1 2 3 4 Sol :

Code - >

```
public class InsertEndLinkedList {
    Node head;
    class Node
    {
        int data;
        Node next;
        Node(int data) {this.data=data; next = null; }
    }
public void insertAtEnd(int x)
  {
    Node newNode = new Node(x);
    if (head == null)
```

```
        {
            head = new Node(x);
            return;
        }
        Node tail = head;
        while (tail.next != null){
            tail = tail.next;
        }
        tail.next = newNode;
        return;
    }
    public void print()
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        InsertBeginLinkedList node = new InsertBeginLinkedList();
        node.insertAtEnd(1);
        System.out.print("Adding nodes to the end of list : ");
        node.print();
        node.insertAtEnd(2);
        System.out.print("Adding nodes to the end of list : ");
        node.print();
        node.insertAtEnd(3);
        System.out.print("Adding nodes to the end of list : ");
        node.print();
        node.insertAtEnd(4);
        System.out.print("Adding nodes to the end of list : ");
        node.print();
    }
}
```

Output ->

Adding nodes to the end of list : 1

Adding nodes to the end of list : 1 2

Adding nodes to the end of list : 1 2 3

Adding nodes to the end of list : 1 2 3 4

Process finished with exit code 0

Q.3 : Insert the element at Given position of the linked list

Input: 3->5->8->10, data = 2, position = 2

Output: 3->2->5->8->10

Sol :

Code - >

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}
public class LinkedListNode {
    public static Node insertAtPos(Node head, int pos, int data){
        Node temp = new  Node (data);
        if(pos == 1){
            temp.next = head;
            return temp;
        }
        Node curr = head;
        for(int i=1; i<pos-2 && curr != null; i++){
            curr = curr.next;
        }
        if(curr==null){
            return head;
        }
        temp.next = curr.next;
        curr.next = temp;
        return head;
    }
    public static void print(Node head)
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input Linked List : ");
```

```
        Node head = new Node(sc.nextInt());
        Node node1 = new Node(sc.nextInt());
        Node node2 = new Node(sc.nextInt());
        Node node3 = new Node(sc.nextInt());
        head.next=node1;
        node1.next=node2;
        node2.next=node3;
        insertAtPos(head,2,2);
        System.out.println("List after insertion");
        print(head);
    }
}
```

Output ->

Input Linked List : 3 5 8 10

List after insertion

3 2 5 8 10

Process finished with exit code 0

Q.4 : Remove the element at Given position of the linked list

Input: position = 2, Linked List = 8->2->3->1->7

Output: Linked List = 8->3->1->7

Sol :

Code - >

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}
public class LinkedListNode {
    public static Node deleteAtPos(Node head, int pos) {
        if(head == null){
```

```
            return null;
        }
        if(pos == 1){
            return head.next;
        }
        Node curr = head;
        for(int i=1; i<pos-2 && curr != null; i++){
            curr = curr.next;
        }
        if(curr==null || curr.next == null){

            return head;
        }
        curr.next = curr.next.next;
        return head;
    }
    public static void print(Node head)
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input Linked List : ");
        Node head = new Node(sc.nextInt());
        Node node1 = new Node(sc.nextInt());
        Node node2 = new Node(sc.nextInt());
        Node node3 = new Node(sc.nextInt());
        Node node4 = new Node(sc.nextInt());
        System.out.print("Enter position : ");
        int pos = sc.nextInt();
        head.next=node1;
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
        System.out.println("List after deletion");
        deleteAtPos(head,pos);
        print(head);
    }
}
```

Output-1 ->

Input Linked List : 8 2 3 1 7

Enter position : 2

List after deletion

8 3 1 7

Process finished with exit code 0

Q.5 : Search the element of the linked list

Input: = [10->20->30->12->0->23->2->12] element = 23

Output: 5

Sol :

Code - >

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}
public class LinkedListNode {
    public static int search(Node head, int n) {
        if(head == null){
            return -1;
        }
        if(head.data == n){
            return 0;
        }
        int res = search(head.next,n);
        if(res == -1){
            return -1;
        }
        return res+1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input Linked List : ");
        Node head = new Node(sc.nextInt());
        Node node1 = new Node(sc.nextInt());
        Node node2 = new Node(sc.nextInt());
        Node node3 = new Node(sc.nextInt());
```

```
        Node node4 = new Node(sc.nextInt());
        Node node5 = new Node(sc.nextInt());
        Node node6 = new Node(sc.nextInt());
        Node node7 = new Node(sc.nextInt());
        head.next=node1;
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
        node4.next=node5;
        node5.next=node6;
        node6.next=node4;
        System.out.print("Enter element : ");
        int element = sc.nextInt();
        int findNode = search(head,element);
        System.out.println("Position of the element is : " + findNode);
    }
}
```

Output ->

Input Linked List : 10 20 30 12 0 23 2 12

Enter element : 23

Position of the element is : 5

Process finished with exit code 0

Q.6. Find the Starting point of loop

Sol :

Code - >

```
import java.util.HashSet;
import java.util.Scanner;
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}
public class LinkedListNode {
    public static Node startingPointOfLoop(Node head) {
        if(head == null || head.next == null){
```

```
            return null;
        }
        Node temp = head;
        HashSet<Node> hs = new HashSet<>();
        while(!hs.contains(temp)){
            hs.add(temp);
            if(temp.next == null){
                return null;
            }
            temp=temp.next;
        }
        return temp;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input Linked List : ");
        Node head = new Node(sc.nextInt());
        Node node1 = new Node(sc.nextInt());
        Node node2 = new Node(sc.nextInt());
        Node node3 = new Node(sc.nextInt());
        Node node4 = new Node(sc.nextInt());
        head.next=node1;
        node1.next=node2;
        node2.next=node3;
        node3.next=node4;
       node4.next=node2;
        Node loopStart = startingPointOfLoop(head);
        if (loopStart != null){
            System.out.println("Starting point of loop is at node : " + loopStart.data);
        }
    }
}
```

Output ->

Input Linked List : 1 2 3 4 5

Starting point of loop is at node : 3

Process finished with exit code 0

Q.7. Implement the tree

Sol :

Code - >

```java
import java.util.*;
import java.util.Scanner;

public class TreeClass {
    static class TreeNode<T> {
        T data;
        ArrayList<TreeNode<T>> children;

        TreeNode(T data) {
            this.data = data;
            children = new ArrayList<TreeNode<T>>();
        }
    }

    public static TreeNode<Integer> takeInput(){
        Scanner s= new Scanner(System.in);
        Queue<TreeNode<Integer>> pendingNodes= new LinkedList<>();
        System.out.println("Enter the root data ");
        int rootData= s.nextInt();
        if(rootData==-1)
            return null;

        TreeNode<Integer> root= new TreeNode<Integer>(rootData);
        pendingNodes.add(root);

        while(!pendingNodes.isEmpty()){
            TreeNode<Integer> front= pendingNodes.poll();
            System.out.println("Enter no. of children "+ front.data);
            int numChild= s.nextInt();
            for(int i=0;i<numChild;i++){
                System.out.println("Enter the  "+i+" th child data"+ front.data);
                int childData= s.nextInt();
                TreeNode<Integer> childNode= new TreeNode<>(childData);
                front.children.add(childNode);
                pendingNodes.add(childNode);
            }
        }
        return root;
    }


    public static void printTree(TreeNode<Integer> root){

        if(root==null){
            return ;
        }
        System.out.print(root.data+": ");
        for(int i=0; i<root.children.size(); i++){
            System.out.print(root.children.get(i).data+ " ");
        }
```

```
        System.out.println();
        for(int i=0;i<root.children.size();i++){
            TreeNode<Integer> child= root.children.get(i);
            printTree(child);
        }
    }
    public static void main(String[] args){
        TreeNode<Integer> root = takeInput();
        printTree(root);
    }
}
```

Output ->

Enter the root data

1

Enter no. of children 1

3

Enter the 0 th child data1

4

Enter the 1 th child data1

5

Enter the 2 th child data1

6

Enter no. of children 4

0

Enter no. of children 5

0

Enter no. of children 6

0

1: 4 5 6

4:

5:

6:

Process finished with exit code 0

Q.8. Print the PreOrder Traversal of a binary tree

Output => Preorder (Root, Left, Right) : 1 2 4 5 3

Sol :

Code - >

```
import java.util.LinkedList;
import java.util.Queue;

public class Trees {
    public static class  BinaryTreeNode<T> {
        T data;
        BinaryTreeNode<T> left;
        BinaryTreeNode<T> right;
        public BinaryTreeNode(T data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static void preOrder(BinaryTreeNode<Integer> root) {
        //Your code goes here
        if(root == null){
            return;
        }
        System.out.print(root.data+" ");
        preOrder(root.left);
        preOrder(root.right);
    }
    public static void main(String[] args) {
        BinaryTreeNode<Integer> root = new BinaryTreeNode<>(1);
        BinaryTreeNode<Integer> node1 = new BinaryTreeNode<>(2);
        BinaryTreeNode<Integer> node2 = new BinaryTreeNode<>(3);
        BinaryTreeNode<Integer> node3 = new BinaryTreeNode<>(4);
        BinaryTreeNode<Integer> node4 = new BinaryTreeNode<>(5);
        root.left=node1;
        root.right=node2;
        node1.left=node3;
        node1.right=node4;
        node2.left=null;
        node2.right=null;
        node3.left=null;
        node3.right=null;
        node4.left=null;
        node4.right=null;
        System.out.print("Preorder Traversal : ");
        preOrder(root);
```

```
        System.out.println();
    }
}
```

Output ->

Preorder Traversal : 1 2 4 5 3

Process finished with exit code 0

Q.9. Print the InOrder Traversal of a binary tree

Output => Inorder (Left, Root, Right) : 4 2 5 1 3

Sol :

Code - >

```
import java.util.LinkedList;
import java.util.Queue;

public class Trees {
    public static class  BinaryTreeNode<T> {
        T data;
        BinaryTreeNode<T> left;
        BinaryTreeNode<T> right;
        public BinaryTreeNode(T data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static void inOrder(BinaryTreeNode<Integer> root) {
        if(root == null){
            return;
        }
        inOrder(root.left);
        System.out.print(root.data+" ");
        inOrder(root.right);
    }


    public static void main(String[] args) {
        BinaryTreeNode<Integer> root = new BinaryTreeNode<>(1);
        BinaryTreeNode<Integer> node1 = new BinaryTreeNode<>(2);
        BinaryTreeNode<Integer> node2 = new BinaryTreeNode<>(3);
        BinaryTreeNode<Integer> node3 = new BinaryTreeNode<>(4);
        BinaryTreeNode<Integer> node4 = new BinaryTreeNode<>(5);
        root.left=node1;
```

```
        root.right=node2;
        node1.left=node3;
        node1.right=node4;
        node2.left=null;
        node2.right=null;
        node3.left=null;
        node3.right=null;
        node4.left=null;
        node4.right=null;
        System.out.print("Inorder Traversal : ");
        inOrder(root);
        System.out.println();
    }
}
```

Output ->

Inorder Traversal : 4 2 5 1 3

Process finished with exit code 0

Q.10. Print the PostOrder Traversal of a binary tree

Output=> Postorder (Left, Right, Root) : 4 5 2 3 1

Sol :

Code - >

```
import java.util.LinkedList;
import java.util.Queue;

public class Trees {
    public static class  BinaryTreeNode<T> {
        T data;
        BinaryTreeNode<T> left;
        BinaryTreeNode<T> right;
        public BinaryTreeNode(T data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static void postOrder(BinaryTreeNode<Integer> root) {
        //Your code goes here
        if(root == null){
            return;
        }
```

```
        postOrder(root.left);
        postOrder(root.right);
        System.out.print(root.data+" ");
    }

    public static void main(String[] args) {
        BinaryTreeNode<Integer> root = new BinaryTreeNode<>(1);
        BinaryTreeNode<Integer> node1 = new BinaryTreeNode<>(2);
        BinaryTreeNode<Integer> node2 = new BinaryTreeNode<>(3);
        BinaryTreeNode<Integer> node3 = new BinaryTreeNode<>(4);
        BinaryTreeNode<Integer> node4 = new BinaryTreeNode<>(5);
        root.left=node1;
        root.right=node2;
        node1.left=node3;
        node1.right=node4;
        node2.left=null;
        node2.right=null;
        node3.left=null;
        node3.right=null;
        node4.left=null;
        node4.right=null;
        System.out.print("Postorder Traversal : ");
        postOrder(root);
        System.out.println();
    }
}
```

Output ->

Postorder Traversal : 4 5 2 3 1

Process finished with exit code 0

Q.11. Print the LevelOrder Traversal of a binary tree

Level Order => 1 2 3 4 5

Sol :

Code - >

```
import java.util.LinkedList;
import java.util.Queue;

public class Trees {
    public static class  BinaryTreeNode<T> {
        T data;
        BinaryTreeNode<T> left;
        BinaryTreeNode<T> right;
```

```
        public BinaryTreeNode(T data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static void levelOrder(BinaryTreeNode<Integer> root){
        Queue q = new LinkedList();
        q.add(root);
        while (!q.isEmpty()){
            BinaryTreeNode<Integer> curr = (BinaryTreeNode<Integer>) q.poll();
            if (curr.left != null){
                q.add(curr.left);
            }
            if (curr.right != null){
                q.add(curr.right);
            }
            System.out.print(curr.data+" ");
        }
    }

    public static void main(String[] args) {
        BinaryTreeNode<Integer> root = new BinaryTreeNode<>(1);
        BinaryTreeNode<Integer> node1 = new BinaryTreeNode<>(2);
        BinaryTreeNode<Integer> node2 = new BinaryTreeNode<>(3);
        BinaryTreeNode<Integer> node3 = new BinaryTreeNode<>(4);
        BinaryTreeNode<Integer> node4 = new BinaryTreeNode<>(5);
        root.left=node1;
        root.right=node2;
        node1.left=node3;
        node1.right=node4;
        node2.left=null;
        node2.right=null;
        node3.left=null;
        node3.right=null;
        node4.left=null;
        node4.right=null;
        System.out.print("LevelOrder Traversal : ");
        levelOrder(root);
        System.out.println();
    }
}
```

Output ->

LevelOrder Traversal : 1 2 3 4 5

Process finished with exit code 0