Student ID: 202112047

IT623 - Lab Assignment 9

1. Implement BFS on a given graph.

Code:

```
import java.util.*;

public class Program1
{
    int node;
    LinkedList<Integer> list[];
    Queue<Integer> q;

    Program1(int v)
    {
        node = v;
        list = new LinkedList[node];

        for(int i = 0; i < v; i++)
        {
            list[i] = new LinkedList<>();
        }
        q = new LinkedList<Integer>();
    }

    void insertEdge(int v, int w)
    {
        list[v].add(w);
    }
```

```
void BFS(int n)
{
    boolean nodes[] = new boolean[node];
    int a = 0;
    nodes[n]=true;
    q.add(n);

    while (q.size() != 0)
    {
        n = q.poll();
        System.out.print(n+" ");

        for (int i = 0; i < list[n].size(); i++)
        {
            a = list[n].get(i);
            if (!nodes[a])
            {
                nodes[a] = true;
                q.add(a);
            }
        }
    }
}

public static void main(String args[])
{
     Program1 p1 = new Program1(14);

    p1.insertEdge(1, 2);
    p1.insertEdge(1, 3);
    p1.insertEdge(3, 6);
    p1.insertEdge(3, 7);
    p1.insertEdge(7, 10);
```

```
        p1.insertEdge(7, 11);
        p1.insertEdge(2, 4);
        p1.insertEdge(2, 5);
        p1.insertEdge(5, 9);
        p1.insertEdge(4, 8);
        p1.insertEdge(8, 12);

        System.out.println("Breadth First Traversal for the graph is ");
        p1.BFS(1);
    }
}
```

Output Snapshot:

2. Print largest value of each row in tree.

Code:

```
import java.util.*;

public class Program2 {
        static class Node {
                int value;
                Node left, right;
        };
```

```
static void findByLevel(Vector<Integer> res, Node root, int d) {
        if (root == null)
                return;

        if (d == res.size())
                res.add(root.value);
        else
                res.set(d, Math.max(res.get(d), root.value));

        findByLevel(res, root.left, d + 1);
        findByLevel(res, root.right, d + 1);

}

static Vector<Integer> largestValue(Node root) {
        Vector<Integer> res = new Vector<>();
        findByLevel(res, root, 0);
        return res;
}

static Node newNode(int data) {
        Node temp = new Node();
        temp.value = data;
        temp.left = temp.right = null;
        return temp;
}

public static void main(String[] args) {
        Node root = null;

        root = newNode(1);
        root.left = newNode(3);
        root.right = newNode(2);
```

```
        root.left.left = newNode(5);
        root.left.right = newNode(3);
        root.right.right = newNode(9);

        Vector<Integer> res = largestValue(root);

        for (int i = 0; i < res.size(); i++)
            System.out.print(res.get(i) + " ");
    }
}
```

Output Snapshot:

3. Populate the next pointer of each node as follows.

Code:

```
class Node {
    int data;
    Node left, right, nextRight;

    Node(int item) {
        data = item;
        left = right = nextRight = null;
    }
```

```
}

public class Program3 {

	static class BinaryTree {
		Node root;

		void connectRecur(Node p) {
			if (p == null)
				return;
			if (p.nextRight != null)
				connectRecur(p.nextRight);
			if (p.left != null) {
				if (p.right != null) {
					p.left.nextRight = p.right;
					p.right.nextRight = getNextRight(p);
				} else
					p.left.nextRight = getNextRight(p);
				connectRecur(p.left);
			} else if (p.right != null) {
				p.right.nextRight = getNextRight(p);
				connectRecur(p.right);
			} else
				connectRecur(getNextRight(p));
		}

		Node getNextRight(Node p) {
			Node temp = p.nextRight;
			while (temp != null) {
				if (temp.left != null)
					return temp.left;
				if (temp.right != null)
					return temp.right;
```

```
                temp = temp.nextRight;
            }

            return null;
        }

        public static void main(String[] args) {

            BinaryTree tree = new BinaryTree();

            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);
            tree.root.right.right = new Node(7);
            tree.root.right.left = new Node(6);
            tree.connectRecur(tree.root);

            if (tree.root.nextRight == null && tree.root.right.nextRight
            == null && tree.root.right.right.nextRight == null)
                System.out.println(tree.root.data + " null " +
                tree.root.left.data + " " + tree.root.left.nextRight.data
                + " null " + tree.root.left.left.data + " " +
                tree.root.left.left.nextRight.data + " "+
                tree.root.right.left.data + " " +
                tree.root.right.left.nextRight.data + " null ");

        }
    }
}
```

Student ID: 202112047

Output Snapshot: