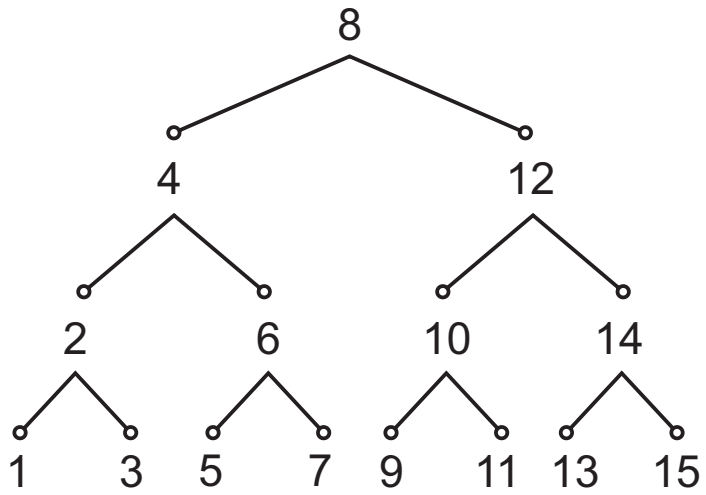
TUGAS MODUL- 09

Nama : Muhammad Ruchbi Ahadian
NPM : 1942448
Kelas : IF - C
Mata Kuliah : *Artificial Intelligence*



Reverse Level Order Traversal :

1									1		15
1	3								3		14
1	3	5							5		13
1	3	5	7						7		12
1	3	5	7	9					9		11
1	3	5	7	9	11				11		10
1	3	5	7	9	11	13			13		9
1	3	5	7	9	11	13	15		15		8
5	7	9	11	13	15	2			2		7
9	11	13	15	2	6				6		6
13	15	2	6	10					10		5
2	6	10	14						14		4
10	14	4							4		3
4	12								12		2
8									8		1

Queue → **Stack ↑** **Iteration ↑**

Hasil *traverse BT in reverse order* :

A. Perhatikan deret angka pada hasil untuk level-3, sebutkan ciri-cirinya

Jawab:

Berisi bilangan ganjil, hasil dari level-3 dimulai dari angka 1, diakhiri oleh angka 15. Terdapat perbedaan nilai sebanyak 2 dari suatu angka ke angka selanjutnya.

B. Perhatikan deret angka pada hasil untuk level-2, sebutkan ciri-cirinya

Jawab:

Berisi bilangan genap, hasil dari level-2 dimulai dari angka 2, diakhiri oleh angka 14. Terdapat perbedaan nilai sebanyak 4 dari suatu angka ke angka selanjutnya.

C. Perhatikan deret angka pada hasil untuk level-1, sebutkan ciri-cirinya

Jawab:

Berisi bilangan genap, hasil dari level-1 dimulai dari angka 4, diakhiri oleh angka 12. Terdapat perbedaan nilai sebanyak 8 dari suatu angka ke angka selanjutnya.

D. Apakah ada ke terkaitan hasil level-1 dan level-0

Jawab:

Ada, pada struktur pohon biner seimbang jika jumlah angka yang ada pada suatu level dibagi dengan dua maka akan menghasilkan dua jenis angka, yaitu angka-angka yang lebih kecil dari angka yang ada di level-0 (sisi kiri) dan angka-angka yang lebih besar dari level-0 (sisi kanan). Jika level-0 dan 1 dibaca secara in-order maka akan Terdapat perbedaan nilai sebanyak 4 dari suatu angka ke angka selanjutnya.

E. Sintesa (gabungkan) contoh program modul sebelumnya (**BinaryTreeTraversal**) dengan contoh program di atas menjadi program dengan 5 moda *traverse* (in, pre, post, level dan reverse level order) dan diberi nama : ***BinaryTreeTraversal2***

Jawab:

```
D:\smt4\Artificial Integelence\P9
λ javac BinaryTreeTraversal2.java

D:\smt4\Artificial Integelence\P9
λ java -cp . BinaryTreeTraversal2

Pre Order Traversal :
8 4 2 1 3 6 5 7 12 10 9 11 14 13 15
In Order Traversal :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Post Order Traversal :
1 3 2 5 7 6 4 9 11 10 13 15 14 12 8
Level Order Traversal :
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
Reverse Level Order Traversal :
1 3 5 7 9 11 13 15 2 6 10 14 4 12 8

D:\smt4\Artificial Integelence\P9
λ
```

BinaryTreeTraversal2.java

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

public class BinaryTreeTraversal2 {
	private Node rootNode;

	public static void main(String[] args) {
		new BinaryTreeTraversal2();
	}

	public BinaryTreeTraversal2(){

		addNode(rootNode, 8);
		addNode(rootNode, 4);
		addNode(rootNode, 2);
		addNode(rootNode, 1);
		addNode(rootNode, 3);
		addNode(rootNode, 6);
		addNode(rootNode, 5);
		addNode(rootNode, 7);
		addNode(rootNode, 12);
		addNode(rootNode, 10);
		addNode(rootNode, 9);
		addNode(rootNode, 11);
		addNode(rootNode, 14);
		addNode(rootNode, 13);
		addNode(rootNode, 15);

		System.out.println("\nPre Order Traversal :");
		printTreePreOrder(rootNode);

		System.out.println("\nIn Order Traversal :");
		printTreeInOrder(rootNode);

		System.out.println("\nPost Order Traversal :");
		printTreePostOrder(rootNode);

		System.out.println("\nLevel Order Traversal :");
		printTreeLevelOrder(rootNode);

		System.out.println("\nReverse Level Order Traversal :");
		printTreeReverseLevelOrder(rootNode);
		}

		//Preorder Printing.
		private void printTreePreOrder(Node rootNode){
			if(rootNode==null)
				return;
			System.out.print(rootNode.getData() + " ");
			printTreePreOrder(rootNode.getLeft());
			printTreePreOrder(rootNode.getRight());
		}

		//Inorder Printing.
		private void printTreeInOrder(Node rootNode){
			if(rootNode==null)
				return;
			printTreeInOrder(rootNode.getLeft());
			System.out.print(rootNode.getData() + " ");
			printTreeInOrder(rootNode.getRight());
		}

		//Postorder Printing.
		private void printTreePostOrder(Node rootNode){
			if(rootNode==null)
				return;
			printTreePostOrder(rootNode.getLeft());
			printTreePostOrder(rootNode.getRight());
			System.out.print(rootNode.getData() + " ");
		}

		//Level order Printing.
		private void printTreeLevelOrder(Node rootNode){
			if(rootNode==null)
				return;
```

```
                Queue<Node> queue = new LinkedList<Node>();
                queue.add(rootNode);

                while(!queue.isEmpty()){
                        Node obj = (Node)queue.poll();
                        System.out.print(obj.getData() + " ");

                        if(obj.getLeft()!=null)
                                queue.add(obj.getLeft());
                        if(obj.getRight()!=null)
                                queue.add(obj.getRight());
                }
        }

        //Reverse level order Printing.
        private void printTreeReverseLevelOrder(Node rootNode) {
                if (rootNode == null) {
                        return;
                }
                Stack<Node> stack = new Stack<Node>();
                Queue<Node> queue = new LinkedList<Node>();
                queue.add(rootNode);

                while(!queue.isEmpty()) {
                        Node node = queue.poll();
                        stack.add(node);

                        if (node.getRight() != null) {
                                queue.add(node.getRight());
                        }
                        if (node.getLeft() != null) {
                                queue.add(node.getLeft());
                        }
                }

                while(!stack.empty()) {
                        System.out.print(stack.pop().getData() + " ");
                }  System.out.println();
        }

        private void addNode(Node rootNode, int data){
                if(rootNode==null){
                        Node temp1 = new Node(data);
                        this.rootNode=temp1;
                }else{
                        addNodeInProperPlace(rootNode, data);
                }
        }

        private void addNodeInProperPlace(Node rootNode, int data){
                if(data>rootNode.getData()){
                        if(rootNode.getRight()!=null){
                                addNode(rootNode.getRight(), data);
                        }else{
                                Node temp1 = new Node(data);
                                rootNode.setRight(temp1);
                        }
                }else if(data<rootNode.getData()){
                        if(rootNode.getLeft()!=null){
                                addNode(rootNode.getLeft(), data);
                        }else{
                                Node temp1 = new Node(data);
                                rootNode.setLeft(temp1);
                        }
                }
        }
}

class Node{
        private int data;
        private Node left;
        private Node right;

        public Node(int data) {
                this.data=data;
        }

        public int getData() {
                return data;
```

```
	public void setData(int data) {
		this.data = data;
	}

	public Node getLeft() {
		return left;
	}

	public void setLeft(Node left) {
		this.left = left;
	}

	public Node getRight() {
		return right;
	}

	public void setRight(Node right) {
		this.right = right;
	}
}
```