Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

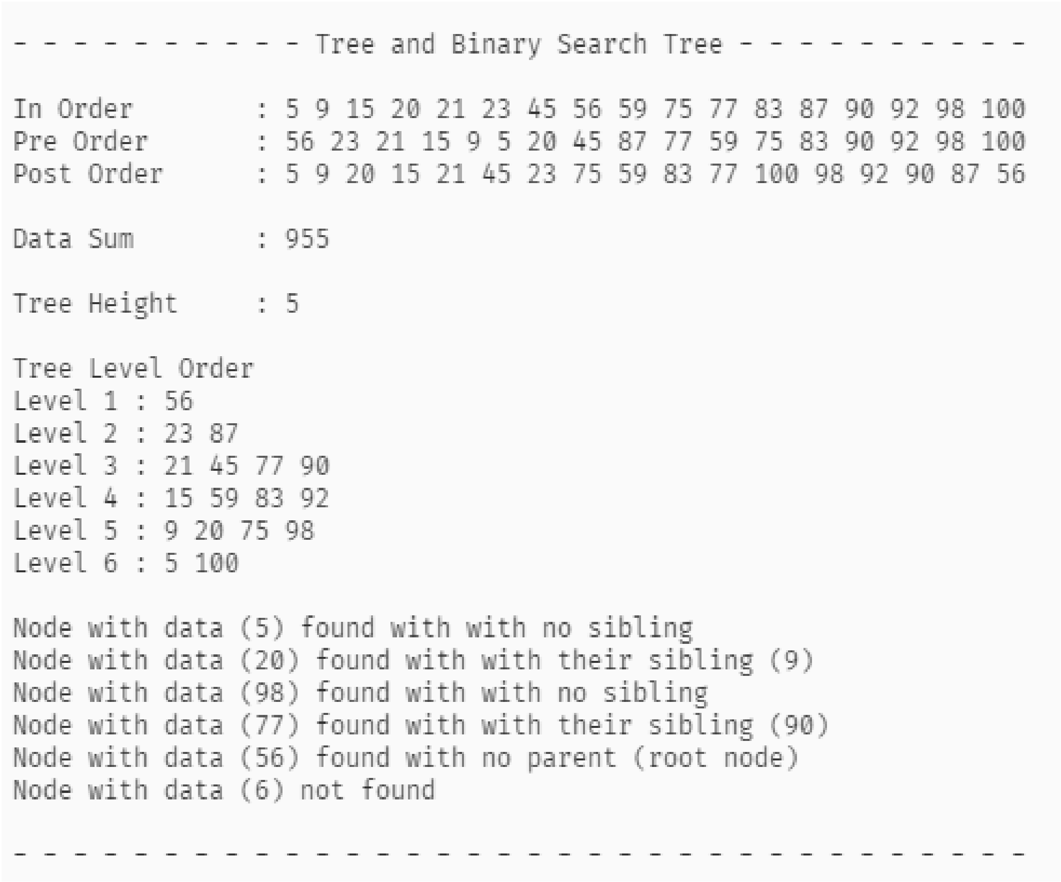
Kelas : KOMB1

Tugas 3 –Tree dan Binary Search Tree

```java
public class Main {
    public static void main(String[] args) {
        int[] data = {56 ,23, 21, 15, 9, 87, 45, 77, 59, 90, 83, 75, 20, 5, 92, 98, 100};

        // Mengimplementasikan binary search tree
        Tree BST = new Tree();

        for(int i = 0; i < data.length; i++) {
            Node node = new Node(data[i]);
            BST.addNode(node);
        }

        System.out.println("\n- - - - - - - - - - Tree and Binary Search Tree - - - - - - - - - -");

        // Menampilkan hasil kunjungan berdasarkan 3 cara
        BST.printInOrder();
        BST.printPreOrder();
        BST.printPostOrder();

        // Menghitung jumlah nilai seluruh elemen
        BST.printSum();

        // Menentukan tinggi binary search tree
        BST.printTreeHeight();

        // Menampilkan node berdasarkan level kedalaman
        BST.printLevelOrder();

        // Menampilkan nilai sibling node
        BST.printSibling(5);
        BST.printSibling(20);
        BST.printSibling(98);
        BST.printSibling(77);
        BST.printSibling(56);
        BST.printSibling(6);

        System.out.println("\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n");
    }
}

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

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

    public int getData() {      // Mengambil value node
        return this.data;
    }

    public Node getLeft() {     // Mengambil left child node
        return this.left;
    }

    public Node getRight() {        // Mengambil right child node
        return this.right;
    }

    public void setLeft(Node node) {    // Mengubah node left child node
        this.left = node;
    }

    public void setRight(Node node) {   // Mengubah node right child node
        this.right = node;
    }
}

class Tree {
    private Node root, parent;
    private int height, sum = 0;

    public Node getRoot() {     // Mengambil value node root
        return this.root;
    }

    public boolean isEmpty() {      // Cek eksistensi tree
        return root == null;
    }

    public void addNode(Node node) {    // Menambahkan node ke tree
        if(isEmpty()) {
            root = node;
        }
        else {
            addNodeFunction(node, root);    // Memanggil fungsi lanjutan untuk menambahkan node
        }
    }

    public void addNodeFunction(Node node, Node parent) {       // Menambahkan node
        if(parent.getData() > node.getData()) {
            if(parent.getLeft() == null) {
                parent.setLeft(node);
            }
            else {
                addNodeFunction(node, parent.getLeft());
            }
        }
              else {
                  if(parent.getRight() == null) {
                      parent.setRight(node);
                  }
                  else {
                      addNodeFunction(node, parent.getRight());
                  }
              }
          }

          public void inOrderFunction(Node node) {        // Menampilkan data secara inorder
              if(node != null) {
                  inOrderFunction(node.getLeft());
                  System.out.print(node.getData() + "  ");
                  inOrderFunction(node.getRight());
              }
          }

          public void printInOrder() {        // Fungsi utama untuk menampilkan data secara inorder
              System.out.print("\nIn Order\t: ");
              this.inOrderFunction(this.getRoot());
          }

          public void preOrderFunction(Node node) {       // Menampilkan data secara preorder
              if(node != null) {
                  System.out.print(node.getData() + "  ");
                  preOrderFunction(node.getLeft());
                  preOrderFunction(node.getRight());
              }
          }

          public void printPreOrder() {         // Fungsi utama untuk menampilkan data secara preorder
              System.out.print("\nPre Order\t: ");
              this.preOrderFunction(this.getRoot());
          }


          public void postOrderFunction(Node node) {      // Menampilkan data secara postorder
              if(node != null) {
                  postOrderFunction(node.getLeft());
                  postOrderFunction(node.getRight());
                  System.out.print(node.getData() + "  ");
              }
          }

          public void printPostOrder() {         // Fungsi utama untuk menampilkan data secara postorder
              System.out.print("\nPost Order\t: ");
              this.postOrderFunction(this.getRoot());
          }

          public void sumNodeData(Node node) {         // Menambahkan seluruh value node di tree
              if(node != null) {
                  sumNodeData(node.getLeft());
                  sum += node.getData();
                  sumNodeData(node.getRight());
              }
          }

          public void printSum() {         // Fungsi utama untuk menjumlahkan semua value node pada tree
              sumNodeData(root);
              System.out.println("\n\nData Sum\t: " + sum + "\n");
              sum = 0;
          }

          public void printTreeHeight() {         // Fungsi utama untuk menghitung tinggi tree
              height = treeHeightFunction(root);
              System.out.println("Tree Height\t: " + (height - 1) + "\n");
          }

          public int treeHeightFunction(Node node) {          // Menghitung tinggi tree
              if(node == null) {
                  return 0;
              }
              else {
                  if(treeHeightFunction(node.getLeft()) > treeHeightFunction(node.getRight())) {
                      return(1 + treeHeightFunction(node.getLeft()));
                  }
                  else {
                      return(1 + treeHeightFunction(node.getRight()));
                  }
              }
          }

          public void printLevelOrder() {         // Fungsi utama untuk menampilkan data tiap level pada tree
              System.out.println("Tree Level Order");
              for(int i = 1; i <= height; i++) {
                  System.out.print("Level " + i + " : ");
                  levelOrderFunction(root, i);
                  System.out.println();
              }
              System.out.println();
          }

          public void levelOrderFunction(Node node, int level) {      // Mencari dan menampilkan data tiap level pada tree
              if(node != null) {
                  if(level == 1){
                      System.out.print(node.getData() + "  ");
                  }
                  else if(level > 1) {
                      levelOrderFunction(node.getLeft(), level - 1);
                      levelOrderFunction(node.getRight(), level - 1);
                  }
```

```java
        }

    }

    public void printSibling(int data) {            // Mencari sibling dari sebuah node dalam tree
        boolean isExist = searchNode(root, data);
        if(isExist) {
            System.out.print("Node with data (" + data + ") found with");
            findParent(root, data);
            if(parent != null) {
                // System.out.print(" with parent (" + parent.getData() + ")");
                if(parent.getLeft() != null && parent.getLeft().getData() == data) {
                    if(parent.getRight() != null) {
                        System.out.println(" with their sibling (" + parent.getRight().getData() + ")");
                    }
                    else {
                        System.out.println(" with no sibling");
                    }
                }
                else if(parent.getRight() != null && parent.getRight().getData() == data) {
                    if(parent.getLeft() != null) {
                        System.out.println(" with their sibling (" + parent.getLeft().getData() + ")");
                    }
                    else {
                        System.out.println(" with no sibling");
                    }
                }
            } else {
                System.out.println(" no parent (root node)");
            }
        }
        else {
            System.out.println("Node with data (" + data + ") not found");
        }
    }

```

```java
        public boolean searchNode(Node node, int data) {        // Cek eksistensi node
            while(node != null) {
                if(node.getData() == data) {
                    return true;
                }
                else {
                    if(node.getData() > data) {
                        return searchNode(node.getLeft(), data);
                    }
                    else {
                        return searchNode(node.getRight(), data);
                    }
                }
            }
            return false;
        }

        public Node findParent(Node node, int data) {       // Mencari parent dari sebuah node dalam tree
            if(node == root) {
                parent = null;
            }
            while(node != null) {
                if((node.getLeft() != null && node.getLeft().getData() == data) || (node.getRight() != null && node.getRight().getData() == data)) {
                    parent = node;
                }

                if(node.getData() > data) {
                    return findParent(node.getLeft(), data);
                }
                else {
                    return findParent(node.getRight(), data);
                }
            }
            return parent;
        }
    }
```



```
- - - - - - - - - - Tree and Binary Search Tree - - - - - - - - - -

In Order        : 5 9 15 20 21 23 45 56 59 75 77 83 87 90 92 98 100
Pre Order       : 56 23 21 15 9 5 20 45 87 77 59 75 83 90 92 98 100
Post Order      : 5 9 20 15 21 45 23 75 59 83 77 100 98 92 90 87 56

Data Sum        : 955

Tree Height     : 5

Tree Level Order
Level 1 : 56
Level 2 : 23 87
Level 3 : 21 45 77 90
Level 4 : 15 59 83 92
Level 5 : 9 20 75 98
Level 6 : 5 100

Node with data (5) found with with no sibling
Node with data (20) found with with their sibling (9)
Node with data (98) found with with no sibling
Node with data (77) found with with their sibling (90)
Node with data (56) found with no parent (root node)
Node with data (6) not found

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```