

Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

Kelas : KOMB1

Tugas 3 –Tree dan Binary Search Tree

```
1 public class Main {
2     public static void main(String[] args) {
3         int[] data = {56 ,23, 21, 15, 9, 87, 45, 77, 59, 90, 83,
4             75, 20, 5, 92, 98, 100};
5         // Mengimplementasikan binary search tree
6         Tree BST = new Tree();
7
8         for(int i = 0; i < data.length; i++) {
9             Node node = new Node(data[i]);
10            BST.addNode(node);
11        }
12
13        System.out.println("\n- - - - - Tree and
14        Binary Search Tree - - - - -");
15
16        // Menampilkan hasil kunjungan berdasarkan 3 cara
17        BST.printInOrder();
18        BST.printPreOrder();
19        BST.printPostOrder();
20
21        // Menghitung jumlah nilai seluruh elemen
22        BST.printSum();
23
24        // Menentukan tinggi binary search tree
25        BST.printTreeHeight();
26
27        // Menampilkan node berdasarkan level kedalaman
28        BST.printLevelOrder();
29
30        // Menampilkan nilai sibling node
31        BST.printSibling(5);
32        BST.printSibling(20);
33        BST.printSibling(98);
34        BST.printSibling(77);
35        BST.printSibling(56);
36        BST.printSibling(6);
```

```

37         System.out.println("\n- - - - -");
38     }
39 }
40
41 class Node {
42     private int data;
43     private Node left, right;
44
45     public Node(int data) {        // Node constructor
46         this.data = data;
47     }
48
49     public int getData() {        // Mengambil value node
50         return this.data;
51     }
52
53     public Node getLeft() {        // Mengambil left child node
54         return this.left;
55     }
56
57     public Node getRight() {        // Mengambil right child
58         node                        // Mengambil right child
59         return this.right;
60     }
61
62     public void setLeft(Node node) {    // Mengubah node left
63         child node
64         this.left = node;
65     }
66
67     public void setRight(Node node) {    // Mengubah node right
68         child node
69         this.right = node;
70     }
71 }
72
73 class Tree {
74     private Node root, parent;
75     private int height, sum = 0;
76
77     public Node getRoot() {        // Mengambil value node root
78         return this.root;
79     }
80
81     public boolean isEmpty() {        // Cek eksistensi tree
82         return root == null;
83     }
84 }

```

```

81
82     public void addNode(Node node) {        // Menambahkan node ke
      tree
83         if(isEmpty()) {
84             root = node;
85         }
86         else {
87             addNodeFunction(node, root);    // Memanggil fungsi
      lanjutan untuk menambahkan node
88         }
89     }
90
91     public void addNodeFunction(Node node, Node parent)
    {
      // Menambahkan node
92         if(parent.getData() > node.getData()) {
93             if(parent.getLeft() == null) {
94                 parent.setLeft(node);
95             }
96             else {
97                 addNodeFunction(node, parent.getLeft());
98             }
99         }
100         else {
101             if(parent.getRight() == null) {
102                 parent.setRight(node);
103             }
104             else {
105                 addNodeFunction(node, parent.getRight());
106             }
107         }
108     }
109
110     public void inOrderFunction(Node node) {        //
      Menampilkan data secara inorder
111         if(node != null) {
112             inOrderFunction(node.getLeft());
113             System.out.print(node.getData() + " ");
114             inOrderFunction(node.getRight());
115         }
116     }
117
118     public void printInOrder() {        // Fungsi utama
      untuk menampilkan data secara inorder
119         System.out.print("\nIn Order\t: ");
120         this.inOrderFunction(this.getRoot());
121     }
122

```

```

123         public void preOrderFunction(Node node) {           //
            Menampilkan data secara preorder
124             if(node != null) {
125                 System.out.print(node.getData() + " ");
126                 preOrderFunction(node.getLeft());
127                 preOrderFunction(node.getRight());
128             }
129         }
130
131         public void printPreOrder() {           // Fungsi utama
            untuk menampilkan data secara preorder
132             System.out.print("\nPre Order\t: ");
133             this.preOrderFunction(this.getRoot());
134         }
135
136
137         public void postOrderFunction(Node node) {           //
            Menampilkan data secara postorder
138             if(node != null) {
139                 postOrderFunction(node.getLeft());
140                 postOrderFunction(node.getRight());
141                 System.out.print(node.getData() + " ");
142             }
143         }
144
145         public void printPostOrder() {           // Fungsi utama
            untuk menampilkan data secara postorder
146             System.out.print("\nPost Order\t: ");
147             this.postOrderFunction(this.getRoot());
148         }
149
150         public void sumNodeData(Node node) {           //
            Menambahkan seluruh value node di tree
151             if(node != null) {
152                 sumNodeData(node.getLeft());
153                 sum += node.getData();
154                 sumNodeData(node.getRight());
155             }
156         }
157
158         public void printSum() {           // Fungsi utama untuk
            menjumlahkan semua value node pada tree
159             sumNodeData(root);
160             System.out.println("\n\nData Sum\t: " + sum +
            "\n");
161             sum = 0;
162         }
163

```

```

164         public void printTreeHeight() {                // Fungsi
            utama untuk menghitung tinggi tree
165             height = treeHeightFunction(root);
166             System.out.println("Tree Height\t: " + (height -
1) + "\n");
167         }
168
169         public int treeHeightFunction(Node node) {        //
            Menghitung tinggi tree
170             if(node == null) {
171                 return 0;
172             }
173             else {
174                 if(treeHeightFunction(node.getLeft()) >
treeHeightFunction(node.getRight())) {
175                     return(1 +
treeHeightFunction(node.getLeft()));
176                 }
177                 else {
178                     return(1 +
treeHeightFunction(node.getRight()));
179                 }
180             }
181         }
182
183         public void printLevelOrder() {                // Fungsi
            utama untuk menampilkan data tiap level pada tree
184             System.out.println("Tree Level Order");
185             for(int i = 1; i <= height; i++) {
186                 System.out.print("Level " + i + " : ");
187                 levelOrderFunction(root, i);
188                 System.out.println();
189             }
190             System.out.println();
191         }
192
193         public void levelOrderFunction(Node node, int level)
        {
            // Mencari dan menampilkan data tiap level pada tree
194             if(node != null) {
195                 if(level == 1){
196                     System.out.print(node.getData() + " ");
197                 }
198                 else if(level > 1) {
199                     levelOrderFunction(node.getLeft(), level -
1);
200                     levelOrderFunction(node.getRight(), level
- 1);
201                 }

```

```

202         }
203
204     }
205
206     public void printSibling(int data) { //
        Mencari sibling dari sebuah node dalam tree
207         boolean isExist = searchNode(root, data);
208         if(isExist) {
209             System.out.print("Node with data (" + data +
                ") found with");
210             findParent(root, data);
211             if(parent != null) {
212                 // System.out.print(" with parent (" +
                parent.getData() + ")");
213                 if(parent.getLeft() != null &&
                parent.getLeft().getData() == data) {
214                     if(parent.getRight() != null) {
215                         System.out.println(" with their
                sibling (" + parent.getRight().getData() + ")");
216                     }
217                     else {
218                         System.out.println(" with no
                sibling");
219                     }
220                 }
221                 else if(parent.getRight() != null &&
                parent.getRight().getData() == data) {
222                     if(parent.getLeft() != null) {
223                         System.out.println(" with their
                sibling (" + parent.getLeft().getData() + ")");
224                     }
225                     else {
226                         System.out.println(" with no
                sibling");
227                     }
228                 }
229             } else {
230                 System.out.println(" no parent (root
                node)");
231             }
232         }
233         else {
234             System.out.println("Node with data (" + data +
                ") not found");
235         }
236     }
237

```

```

238     public boolean searchNode(Node node, int data)
    {
        // Cek eksistensi node
239         while(node != null) {
240             if(node.getData() == data) {
241                 return true;
242             }
243             else {
244                 if(node.getData() > data) {
245                     return searchNode(node.getLeft(),
data);
246                 }
247                 else {
248                     return searchNode(node.getRight(),
data);
249                 }
250             }
251         }
252         return false;
253     }
254
255     public Node findParent(Node node, int data) {
        //
        Mencari parent dari sebuah node dalam tree
256         if(node == root) {
257             parent = null;
258         }
259         while(node != null) {
260             if((node.getLeft() != null &&
node.getLeft().getData() == data) || (node.getRight() != null &&
node.getRight().getData() == data)) {
261                 parent = node;
262             }
263
264             if(node.getData() > data) {
265                 return findParent(node.getLeft(), data);
266             }
267             else {
268                 return findParent(node.getRight(), data);
269             }
270         }
271         return parent;
272     }
273 }

```

- - - - - 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

- - - - -