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

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

```
81
      public void addNode(Node node) { // Menambahkan node ke
82
   tree
           if(isEmpty()) {
83
84
               root = node;
85
           }
86
           else {
               addNodeFunction(node, root); // Memanggil fungsi
87
   laniutan 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
           }
                else {
100
                     if(parent.getRight() == null) {
101
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
                 if(node != null) {
111
112
                     inOrderFunction(node.getLeft());
113
                     System.out.print(node.getData() + " ");
114
                     inOrderFunction(node.getRight());
115
                 }
116
            }
117
            public void printInOrder() {
118
                                                // Fungsi utama
  untuk menampilkan data secara inorder
119
                 System.out.print("\nIn Order\t: ");
                 this.inOrderFunction(this.getRoot());
120
121
            }
122
```

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

```
public void printTreeHeight() { // Fungsi
  utama untuk menghitung tinggi tree
               height = treeHeightFunction(root);
165
               System.out.println("Tree Height\t: " + (height -
166
  1) + "\n");
167
168
169
           public int treeHeightFunction(Node node) {
 Menghitung tinggi tree
                if(node == null) {
170
171
                   return 0;
172
               else {
173
                   if(treeHeightFunction(node.getLeft()) >
174
  treeHeightFunction(node.getRight())) {
                       return(1 +
  treeHeightFunction(node.getLeft()));
176
177
                   else {
178
                       return(1 +
  treeHeightFunction(node.getRight()));
179
                   }
                }
180
           }
181
182
183
           utama untuk menampilkan data tiap level pada tree
184
               System.out.println("Tree Level Order");
               for(int i = 1; i <= height; i++) {</pre>
185
                   System.out.print("Level " + i + " : ");
186
                   levelOrderFunction(root, i);
187
188
                   System.out.println();
189
190
               System.out.println();
            }
191
192
            public void levelOrderFunction(Node node, int level)
193
         // Mencari dan menampilkan data tiap level pada tree
194
                if(node != null) {
195
                   if(level == 1){
                       System.out.print(node.getData() + " ");
196
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 {
                                 System.out.println(" with no
218
  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) {
                             return searchNode(node.getLeft(),
245
  data);
                         }
246
247
                         else {
                             return searchNode(node.getRight(),
248
  data);
249
                         }
250
                     }
251
                 }
252
                 return false;
            }
253
254
             public Node findParent(Node node, int data) {
255
  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
                     if(node.getData() > data) {
264
265
                         return findParent(node.getLeft(), data);
                     }
266
267
                     else {
268
                         return findParent(node.getRight(), data);
269
                     }
270
                 }
271
                 return parent;
272
            }
273
```

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