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, 75, 20, 5, 92, 98, 100};
4. *// Mengimplementasikan binary search tree*
5. Tree BST = new Tree();
6. for(int i = 0; i < data.length; i++) {
7. Node node = new Node(data[i]);
8. BST.addNode(node);
9. }
11. System.out.println("\n- - - - - - - - - - Tree and Binary Search Tree - - - - - - - - - -");
13. *// Menampilkan hasil kunjungan berdasarkan 3 cara*
14. BST.printInOrder();
15. BST.printPreOrder();
16. BST.printPostOrder();
17. *// Menghitung jumlah nilai seluruh elemen*
18. BST.printSum();
19. *// Menentukan tinggi binary search tree*
20. BST.printTreeHeight();
21. *// Menampilkan node berdasarkan level kedalaman*
22. BST.printLevelOrder();
23. *// Menampilkan nilai sibling node*
24. BST.printSibling(5);
25. BST.printSibling(20);
26. BST.printSibling(98);
27. BST.printSibling(77);
28. BST.printSibling(56);
29. BST.printSibling(6);
30. System.out.println("\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n");
31. }
32. }
33. class Node {
34. private int data;
35. private Node left, right;
36. public Node(int data) {     *// Node constructor*
37. this.data = data;
38. }
39. public int getData() {      *// Mengambil value node*
40. return this.data;
41. }
42. public Node getLeft() {     *// Mengambil left child node*
43. return this.left;
44. }
46. public Node getRight() {        *// Mengambil right child node*
47. return this.right;
48. }
50. public void setLeft(Node node) {    *// Mengubah node left child node*
51. this.left = node;
52. }
54. public void setRight(Node node) {   *// Mengubah node right child node*
55. this.right = node;
56. }
57. }
58. class Tree {
59. private Node root, parent;
60. private int height, sum = 0;
61. public Node getRoot() {     *// Mengambil value node root*
62. return this.root;
63. }
64. public boolean isEmpty() {      *// Cek eksistensi tree*
65. return root == null;
66. }
67. public void addNode(Node node) {    *// Menambahkan node ke tree*
68. if(isEmpty()) {
69. root = node;
70. }
71. else {
72. addNodeFunction(node, root);    *// Memanggil fungsi lanjutan untuk menambahkan node*
73. }
74. }
76. public void addNodeFunction(Node node, Node parent) {       *// Menambahkan node*
77. if(parent.getData() > node.getData()) {
78. if(parent.getLeft() == null) {
79. parent.setLeft(node);
80. }
81. else {
82. addNodeFunction(node, parent.getLeft());
83. }
84. }
85. else {
86. if(parent.getRight() == null) {
87. parent.setRight(node);
88. }
89. else {
90. addNodeFunction(node, parent.getRight());
91. }
92. }
93. }
95. public void inOrderFunction(Node node) {        *// Menampilkan data secara inorder*
96. if(node != null) {
97. inOrderFunction(node.getLeft());
98. System.out.print(node.getData() + " ");
99. inOrderFunction(node.getRight());
100. }
101. }
102. public void printInOrder() {        *// Fungsi utama untuk menampilkan data secara inorder*
103. System.out.print("\nIn Order\t: ");
104. this.inOrderFunction(this.getRoot());
105. }
107. public void preOrderFunction(Node node) {       *// Menampilkan data secara preorder*
108. if(node != null) {
109. System.out.print(node.getData() + " ");
110. preOrderFunction(node.getLeft());
111. preOrderFunction(node.getRight());
112. }
113. }
115. public void printPreOrder() {        *// Fungsi utama untuk menampilkan data secara preorder*
116. System.out.print("\nPre Order\t: ");
117. this.preOrderFunction(this.getRoot());
118. }
120. public void postOrderFunction(Node node) {      *// Menampilkan data secara postorder*
121. if(node != null) {
122. postOrderFunction(node.getLeft());
123. postOrderFunction(node.getRight());
124. System.out.print(node.getData() + " ");
125. }
126. }
128. public void printPostOrder() {        *// Fungsi utama untuk menampilkan data secara postorder*
129. System.out.print("\nPost Order\t: ");
130. this.postOrderFunction(this.getRoot());
131. }
132. public void sumNodeData(Node node) {        *// Menambahkan seluruh value node di tree*
133. if(node != null) {
134. sumNodeData(node.getLeft());
135. sum += node.getData();
136. sumNodeData(node.getRight());
137. }
138. }
139. public void printSum() {        *// Fungsi utama untuk menjumlahkan semua value node pada tree*
140. sumNodeData(root);
141. System.out.println("\n\nData Sum\t: " + sum + "\n");
142. sum = 0;
143. }
144. public void printTreeHeight() {         *// Fungsi utama untuk menghitung tinggi tree*
145. height = treeHeightFunction(root);
146. System.out.println("Tree Height\t: " + (height - 1) + "\n");
147. }
148. public int treeHeightFunction(Node node) {         *// Menghitung tinggi tree*
149. if(node == null) {
150. return 0;
151. }
152. else {
153. if(treeHeightFunction(node.getLeft()) > treeHeightFunction(node.getRight())) {
154. return(1 + treeHeightFunction(node.getLeft()));
155. }
156. else {
157. return(1 + treeHeightFunction(node.getRight()));
158. }
159. }
160. }
161. public void printLevelOrder() {         *// Fungsi utama untuk menampilkan data tiap level pada tree*
162. System.out.println("Tree Level Order");
163. for(int i = 1; i <= height; i++) {
164. System.out.print("Level " + i + " : ");
165. levelOrderFunction(root, i);
166. System.out.println();
167. }
168. System.out.println();
169. }
171. public void levelOrderFunction(Node node, int level) {      *// Mencari dan menampilkan data tiap level pada tree*
172. if(node != null) {
173. if(level == 1){
174. System.out.print(node.getData() + " ");
175. }
176. else if(level > 1) {
177. levelOrderFunction(node.getLeft(), level - 1);
178. levelOrderFunction(node.getRight(), level - 1);
179. }
180. }
182. }
183. public void printSibling(int data) {            *// Mencari sibling dari sebuah node dalam tree*
184. boolean isExist = searchNode(root, data);
185. if(isExist) {
186. System.out.print("Node with data (" + data + ") found with");
187. findParent(root, data);
188. if(parent != null) {
189. *// System.out.print(" with parent (" + parent.getData() + ")");*
190. if(parent.getLeft() != null && parent.getLeft().getData() == data) {
191. if(parent.getRight() != null) {
192. System.out.println(" with their sibling (" + parent.getRight().getData() + ")");
193. }
194. else {
195. System.out.println(" with no sibling");
196. }
197. }
198. else if(parent.getRight() != null && parent.getRight().getData() == data) {
199. if(parent.getLeft() != null) {
200. System.out.println(" with their sibling (" + parent.getLeft().getData() + ")");
201. }
202. else {
203. System.out.println(" with no sibling");
204. }
205. }
206. } else {
207. System.out.println(" no parent (root node)");
208. }
209. }
210. else {
211. System.out.println("Node with data (" + data + ") not found");
212. }
213. }
214. public boolean searchNode(Node node, int data) {        *// Cek eksistensi node*
215. while(node != null) {
216. if(node.getData() == data) {
217. return true;
218. }
219. else {
220. if(node.getData() > data) {
221. return searchNode(node.getLeft(), data);
222. }
223. else {
224. return searchNode(node.getRight(), data);
225. }
226. }
227. }
228. return false;
229. }
230. public Node findParent(Node node, int data) {       *// Mencari parent dari sebuah node dalam tree*
231. if(node == root) {
232. parent = null;
233. }
234. while(node != null) {
235. if((node.getLeft() != null && node.getLeft().getData() == data) || (node.getRight() != null && node.getRight().getData() == data)) {
236. parent = node;
237. }
239. if(node.getData() > data) {
240. return findParent(node.getLeft(), data);
241. }
242. else {
243. return findParent(node.getRight(), data);
244. }
245. }
246. return parent;
247. }
248. }

