

LAPORAN JOBSHHET 13

1. Buatlah class Node, BinaryTree dan BinaryTreeMain
2. Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.
3. Di dalam class BinaryTree, tambahkan atribut root.
4. Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTree
5. Tambahkan method add() di dalam class BinaryTree. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.
6. Tambahkan method find()
7. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order.
8. Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.
9. Tambahkan method delete().
Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.
Kemudian tambahkan proses penghapusan terhadap node current yang telah ditemukan.
10. Buka class BinaryTreeMain dan tambahkan method main().
11. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
12. Amati hasil running tersebut.

Percobaan:

Class Node

```
Source History | 
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7   package tree;
8
9   /**
10    *
11    * @author Hp
12    */
13   public class Node {
14       int data;
15       Node left;
16       Node right;
17
18       public Node () {
19       }
20
21       public Node (int data) {
22           this.left = null;
23           this.data = data;
24           this.right = null;
25       }
26   }
27
```

Class BinaryTree

```
Source History | 
7   package tree;
8
9   /**
10    *
11    * @author Hp
12    */
13   public class BinaryTree {
14       Node root;
15
16       public BinaryTree() {
17           root = null;
18       }
19       boolean isEmpty() {
20           return root == null;
21       }
22       void add (int data) {
23           if (isEmpty()) { // tree is empty
24               root = new Node (data);
25           } else {
26               Node current = root;
27               while (true) {
28                   if (data < current.data) {
29                       if (current.left != null) {
30                           current = current.left;
31                       } else {
32                           current.left = new Node (data);
33                           break;
34                       }
35
36                       } else if (data > current.data) {
37                           if (current.right != null) {
38                               current = current.right;
39                           } else {
40                               current.right = new Node (data);
41                               break;
42                           }
43                       } else { // data is already exist
44                           break;
45                       }
46                   }
47               }
48           }
49       boolean find (int data) {
50           boolean hasil = false;
51           Node current = root;
52           while (current != null) {
53               if (current.data == data) {
54                   hasil = true;
55                   break;
56               } else if (data < current.data) {
57                   current = current.left;
58               } else {
59                   current = current.right;
60               }
61           }
62           return hasil;
63       }
64   }
```

```

65 void traversePreOrder (Node node){
66     if (node != null){
67         System.out.print(" " + node.data);
68         traversePreOrder(node.left);
69         traversePreOrder(node.right);
70     }
71 }
72 void traversePostOrder (Node node){
73     if (node != null){
74         traversePostOrder(node.left);
75         traversePostOrder(node.right);
76         System.out.print(" " + node.data);
77     }
78 }
79 void traverseInOrder (Node node){
80     if (node != null){
81         traverseInOrder(node.left);
82         System.out.print(" " + node.data);
83         traverseInOrder(node.right);
84     }
85 }

```

```

86 Node getSuccessor (Node del){
87     Node successor = del.right;
88     Node successorParent = del;
89     while(successor.left!=null){
90         successorParent = successor;
91         successor = successor.left;
92     }
93     if(successor!=del.right){
94         successorParent.left = successor.right;
95         successor.right = del.right;
96     }
97     return successor;
98 }
99 void delete(int data) {
100     if (isEmpty()) {
101         System.out.println("Tree Is Empty!");
102         return;
103     }

```

```

104 //find node (current) that will be deleted
105 Node parent = root;
106 Node current = root;
107 boolean isLeftChild = false;
108 while (current != null) {
109     if (current.data == data) {
110         break;
111     } else if (data < current.data) {
112         parent = current;
113         current = current.left;
114         isLeftChild = true;
115     } else if (data > current.data) {
116         parent = current;
117         current = current.right;
118         isLeftChild = false;
119     }
120 }

```

```

121 // deletion
122 if (current == null) {
123     System.out.println("Couldn't find data!");
124     return;
125 } else {
126     //if there is no child, simply delete it
127     if (current.left == null && current.right == null) {
128         if (current == root) {
129             root = null;
130         } else {
131             if (isLeftChild) {
132                 parent.left = null;
133             } else {
134                 parent.right = null;
135             }
136         }
137     } else if (current.left == null) { // if there is 1 child (right)
138         if (current == root) {
139             root = current.right;
140         } else {
141             if (isLeftChild) {
142                 parent.left = current.right;
143             } else {
144                 parent.right = current.right;
145             }
146         }
147     }

```

```

147         } else if (current.right == null) { // if there is 1 child (left)
148             if (current == root) {
149                 root = current.left;
150             } else {
151                 if (isLeftChild) {
152                     parent.left = current.left;
153                 } else {
154                     parent.right = current.left;
155                 }
156             }
157         } else { //if there is 2 child
158             Node successor = getSuccessor(current);
159             if (current == root) {
160                 root = successor;
161             } else {
162                 if (isLeftChild) {
163                     parent.left = successor;
164                 }
165                 successor.left = current.left;
166             }
167         }
168     }
169 }
170 }
171

```

BinaryTreeMain

```

Source History
package tree;

/**
 * @author Hp
 */
public class BinaryTreeMain {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        BinaryTree bt = new BinaryTree();

        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find " + bt.find(5));
        bt.delete(8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}

```

Output

Tree - NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

Output - Tree (run)

```

run:
649587971015
94567891015
35471510906
Find true
649587971015
BUILD SUCCESSFUL (total time: 0 seconds)

```

2.1.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

Pada binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa dikarenakan oleh Binary Search Tree mengatur setiap child node sebelah kiri selalu lebih kecil nilainya dari root node. Sedangkan pada child node sebelah kanan mengatur setiap nilai yang lebih besar daripada root node yang memberikan proses efisiensi pada proses searching.

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

kegunaan dari atribut left dan right pada class Node adalah untuk menentukan leftchild dan rightchild dan untuk menyimpan angka di left right selain fungsi dari left dan right sama seperti next dan prev.

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

untuk menentukan nilai paling atas dalam class Binary Tree

b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

null atau kosong

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

Yang terjadi jika tree masih kosong, dan ditambahkan sebuah node baru proses pengisian node ke dalam root baru.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah

ini. Jelaskan secara detil untuk apa baris program tersebut?

jika data baru kurang dari data lama maka di lakukan pengecekan lagi apakah data kiri bernilai sama dengan null, jika iya data lama akan masuk ke dalam data kiri, jika tidak maka data kiri di ganti dengan data yang baru saja di masukkan, setelah itu break

2.2.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2. Buatlah class BinaryTreeArray dan BinaryTreeArrayMain
3. Buat atribut data dan idxLast di dalam class BinaryTreeArray. Buat juga method populateData() dan traverseInOrder().
4. Kemudian dalam class BinaryTreeArrayMain buat method main() seperti gambar berikut ini.
5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

Class BinaryTreeArray:

```
Source History
7 package tree;
8
9 /**
10  *
11  * @author Hp
12  */
13 public class BinaryTreeArray {
14     int[] data;
15     int idxLast;
16
17     public BinaryTreeArray() {
18         data = new int[10];
19     }
20
21     void add(int key) {
22         idxLast++;
23         data[idxLast] = key;
24     }
25
26     void populateData(int data[], int idxLast) {
27         this.data = data;
28         this.idxLast = idxLast;
29     }
30
31     void traverseInOrder(int idxStart) {
32         if (idxStart <= idxLast) {
33             traverseInOrder(2 * idxStart + 1);
34             System.out.print(data[idxStart] + " ");
35             traverseInOrder(2 * idxStart + 2);
36         }
37     }
38
39     void traversePreOrder(int idxStart) {
40         if (idxStart <= idxLast) {
41             System.out.print(data[idxStart] + " ");
42             traversePreOrder(2 * idxStart + 1);
43             traversePreOrder(2 * idxStart + 2);
44         }
45     }
46
47     void traversePostOrder(int idxStart) {
48         if (idxStart <= idxLast) {
49             traversePostOrder(2 * idxStart + 1);
50             traversePostOrder(2 * idxStart + 2);
51             System.out.print(data[idxStart] + " ");
52         }
53     }
54 }
55 }
```

BinaryTreeArrayMain

```

4  * and open the template in the editor.
5  */
6
7  package tree;
8
9  /**
10   *
11   * @author Hp
12   */
13  public class BinaryTreeArrayMain {
14      public static void main(String[] args) {
15          BinaryTreeArray bta = new BinaryTreeArray();
16          int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
17          int idxLast = 6;
18          bta.populateData(data, idxLast);
19          bta.add(25);
20          bta.add(6);
21          bta.add(4);
22          bta.traverseInOrder(0);
23          System.out.println(" ");
24          bta.traversePreOrder(0);
25          System.out.println(" ");
26          bta.traversePostOrder(0);
27          System.out.println(" ");
28      }
29  }
30
31

```

Ouput

```

Output - Tree (run)
run:
25 3 6 4 4 5 6 7 8 9
6 4 2 25 6 5 4 8 7 9
25 6 2 4 5 4 7 9 8 6
BUILD SUCCESSFUL (total time: 0 seconds)

```

13.2.1 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray adalah untuk mendeklarasikan banyaknya nilai array dan IdxLast dalam menentukan nilai terakhir saat add

2. Apakah kegunaan dari method populateData()?

kegunaan dari method populateData() adalah untuk melakukan penginputan data agar dapat dikenali oleh indexnya

3. Apakah kegunaan dari method traverseInOrder()?

kegunaan dari method traverseInOrder() adalah untuk mencetak seluruh data yang ada dalam tree mulai dari sebelah kiri

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks

berapakah posisi left child dan right child masing-masing?

Jika suatu node binary tree disimpan dalam array indeks 2, maka indeks left = 1 dan right = 3

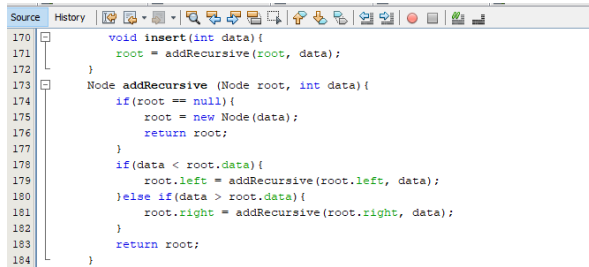
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4 adalah untuk membatasi index agar hanya menjadi 6

13.3 Tugas Praktikum

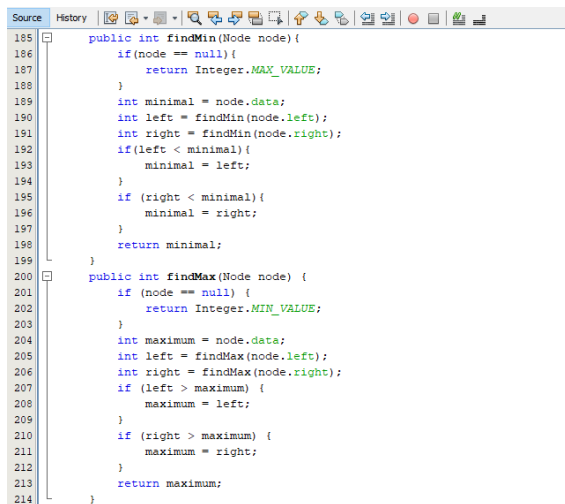
Waktu pengerjaan: 90 menit

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

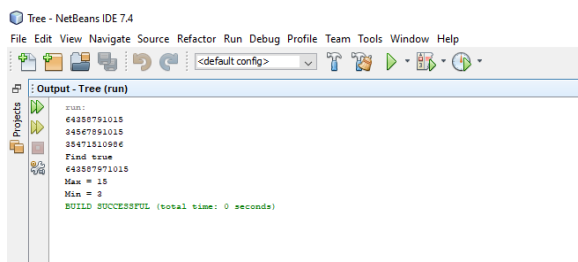


```
170 void insert(int data){
171     root = addRecursive(root, data);
172 }
173 Node addRecursive (Node root, int data){
174     if(root == null){
175         root = new Node(data);
176         return root;
177     }
178     if(data < root.data){
179         root.left = addRecursive(root.left, data);
180     }else if(data > root.data){
181         root.right = addRecursive(root.right, data);
182     }
183     return root;
184 }
```

2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.



```
185 public int findMin(Node node){
186     if(node == null){
187         return Integer.MAX_VALUE;
188     }
189     int minimal = node.data;
190     int left = findMin(node.left);
191     int right = findMin(node.right);
192     if(left < minimal){
193         minimal = left;
194     }
195     if (right < minimal){
196         minimal = right;
197     }
198     return minimal;
199 }
200 public int findMax(Node node) {
201     if (node == null) {
202         return Integer.MIN_VALUE;
203     }
204     int maximum = node.data;
205     int left = findMax(node.left);
206     int right = findMax(node.right);
207     if (left > maximum) {
208         maximum = left;
209     }
210     if (right > maximum) {
211         maximum = right;
212     }
213     return maximum;
214 }
```



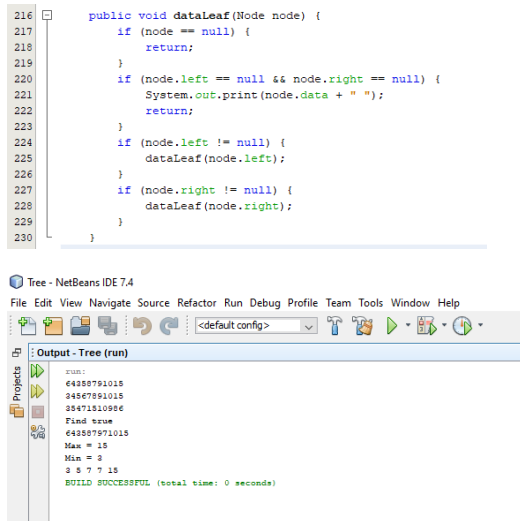
```
Tree - NetBeans IDE 7.4
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Output - Tree (run)
sum:
44388791015
24567891015
25471510906
Find true
44388791015
Max = 15
Min = 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.


```

216     public void dataLeaf(Node node) {
217         if (node == null) {
218             return;
219         }
220         if (node.left == null && node.right == null) {
221             System.out.print(node.data + " ");
222             return;
223         }
224         if (node.left != null) {
225             dataLeaf(node.left);
226         }
227         if (node.right != null) {
228             dataLeaf(node.right);
229         }
230     }

```



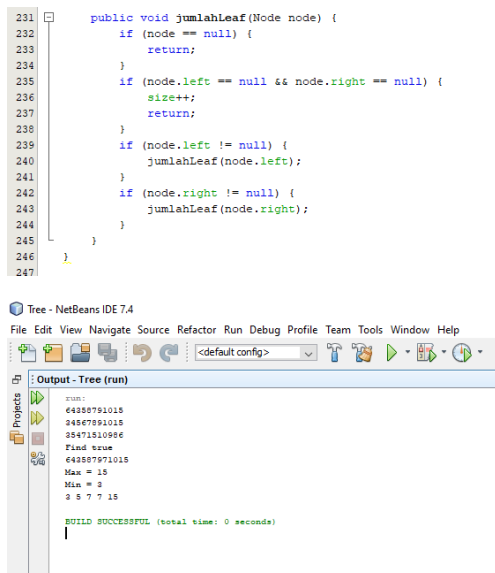
The screenshot shows the NetBeans IDE 7.4 interface. The main editor displays the `dataLeaf` method in a class. The method checks if a node is null, prints its data if it's a leaf node, and recursively calls itself on the left and right children. The bottom panel shows the 'Output - Tree (run)' window with a successful build message: 'BUILD SUCCESSFUL (total time: 0 seconds)'.

4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```

231     public void jumlahLeaf(Node node) {
232         if (node == null) {
233             return;
234         }
235         if (node.left == null && node.right == null) {
236             size++;
237             return;
238         }
239         if (node.left != null) {
240             jumlahLeaf(node.left);
241         }
242         if (node.right != null) {
243             jumlahLeaf(node.right);
244         }
245     }
246 }
247

```



The screenshot shows the NetBeans IDE 7.4 interface. The main editor displays the `jumlahLeaf` method in a class. The method checks if a node is null, increments a `size` variable if it's a leaf node, and recursively calls itself on the left and right children. The bottom panel shows the 'Output - Tree (run)' window with a successful build message: 'BUILD SUCCESSFUL (total time: 0 seconds)'.

5. Modifikasi class BinaryTreeArray, dan tambahkan :

- method `add(int data)` untuk memasukan data ke dalam tree

```

30     public void add(int data, int idx){
31         this.data[idx] = data;
32     }

```

- method `traversePreOrder()` dan `traversePostOrder()`

```

30 public void add(int data, int idx){
31     this.data[idx] = data;
32 }
33 public void traversePreOrder(int idxStart) {
34     if (idxStart <= idxLast) {
35         if (data[idxStart] == 0) {
36             System.out.print(idxLast + " ");
37         } else {
38             System.out.print(data[idxStart] + " ");
39         }
40         traverseInOrder((2 * idxStart) + 1);
41         traverseInOrder((2 * idxStart) + 2);
42     }
43 }
44 public void traversePostOrder(int idxStart) {
45     if (idxStart <= idxLast) {
46         traverseInOrder((2 * idxStart) + 1);
47         traverseInOrder((2 * idxStart) + 2);
48         if (data[idxStart] == 0) {
49             System.out.print(idxLast + " ");
50         } else {
51             System.out.print(data[idxStart] + " ");
52         }
53     }
54 }

```

