



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

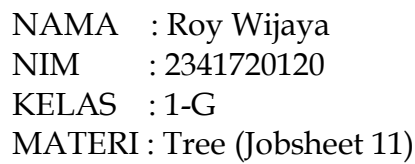
Percobaan 1

```
Preorder Traversal : 6 4 3 5 8 7 9 10 15
Inorder Traversal : 3 4 5 6 7 8 9 10 15
PostOrder Traversal : 3 5 4 7 15 10 9 8 6
Find Node : true
Delete Node 8

PreOrder Traversal : 6 4 3 5 9 7 10 15
```

```
1 package p13;
2
3 public class BinaryTreeMain27 {
4     public static void main(String[] args) {
5         BinaryTree27 bt = new BinaryTree27();
6         bt.add(6);
7         bt.add(4);
8         bt.add(8);
9         bt.add(3);
10        bt.add(5);
11        bt.add(7);
12        bt.add(9);
13        bt.add(10);
14        bt.add(15);
15
16        System.out.print("Preorder Traversal : ");
17        bt.traversePreorder(bt.root);
18        System.out.println("");
19
20        System.out.print("Inorder Traversal : ");
21        bt.traverseInOrder(bt.root);
22        System.out.println("");
23
24        System.out.print("PostOrder Traversal : ");
25        bt.traversePostOrder(bt.root);
26        System.out.println("");
27
28        System.out.println("Find Node : " + bt.find(5));
29        System.out.println("Delete Node 8");
30        bt.delete(8);
31        System.out.println("");
32        System.out.print("PreOrder Traversal : ");
33        bt.traversePreorder(bt.root);
34        System.out.println("");
35    }
36
37 }
```

```
1 package p13;
2
3 public class Node27 {
4     int data;
5     Node27 left;
6     Node27 right;
7
8     Node27(int data) {
9         this.data = data;
10        this.left = null;
11        this.right = null;
12    }
13 }
14
```



NIM : 2341720120

KELAS : 1-G

MATERI : Tree (Jobsheet 11)

```

1 package D3;
2
3 import P41.StructureData.Node;
4
5 public class BinaryTree27 {
6     Node27 root;
7     int size;
8
9     public BinaryTree27() {
10         root = null;
11     }
12
13     boolean isEmpty() {
14         return root == null;
15     }
16
17     void addNode(Node27 data) {
18         if (isEmpty()) {
19             root = new Node27(data);
20         } else {
21             Node27 current = root;
22             if (data.compareTo(current.data) < 0) {
23                 if (current.left == null) {
24                     current.left = new Node27(data);
25                 } else {
26                     current = current.left;
27                 }
28             } else if (data.compareTo(current.data) > 0) {
29                 if (current.right == null) {
30                     current.right = new Node27(data);
31                 } else {
32                     current = current.right;
33                 }
34             } else {
35                 return;
36             }
37         }
38     }
39
40     boolean findNode(Node27 data) {
41         boolean result = false;
42         Node27 current = root;
43         while (current != null) {
44             if (current.data.compareTo(data) == 0) {
45                 result = true;
46                 return result;
47             } else if (data.compareTo(current.data) < 0) {
48                 current = current.left;
49             } else if (data.compareTo(current.data) > 0) {
50                 current = current.right;
51             }
52         }
53         return result;
54     }
55
56     void traverseInOrder(Node27 node) {
57         if (node == null) {
58             return;
59         }
60         traverseInOrder(node.left);
61         traverseInOrder(node.data);
62         traverseInOrder(node.right);
63     }
64
65     void traversePreOrder(Node27 node) {
66         if (node == null) {
67             return;
68         }
69         traversePreOrder(node.data);
70         traversePreOrder(node.left);
71         traversePreOrder(node.right);
72     }
73
74     void traversePostOrder(Node27 node) {
75         if (node == null) {
76             return;
77         }
78         traversePostOrder(node.left);
79         traversePostOrder(node.data);
80         traversePostOrder(node.right);
81     }
82
83     Node27 getInOrder(Node27 node) {
84         if (node == null) {
85             return null;
86         }
87         Node27 successor = null;
88         Node27 predecessor = null;
89         if (current.left != null) {
90             successor = successor.left;
91             predecessor = predecessor.left;
92         } else {
93             successor = null;
94             predecessor = predecessor.right;
95         }
96         return successor;
97     }
98
99     void delete(Node27 data) {
100         if (isEmpty()) {
101             System.out.println("Tree is empty");
102             return;
103         }
104         Node27 parent = root;
105         Node27 current = root;
106         while (current != null) {
107             if (current.data.compareTo(data) < 0) {
108                 current = current.left;
109             } else if (current.data.compareTo(data) > 0) {
110                 current = current.right;
111             } else {
112                 // Node to be deleted is found
113                 if (current.left != null) {
114                     // Node has left child
115                     Node27 leftChild = current.left;
116                     while (leftChild.right != null) {
117                         leftChild = leftChild.right;
118                     }
119                     // Replace current node with its left child
120                     current.data = leftChild.data;
121                     current.left = leftChild.left;
122                     current.right = leftChild.right;
123                     leftChild = null;
124                 } else {
125                     // Node has no left child
126                     // Replace current node with its right child
127                     current.data = current.right.data;
128                     current.left = current.right.left;
129                     current.right = current.right.right;
130                     current.right = null;
131                 }
132             }
133         }
134     }
135
136     boolean findNode(Node27 data) {
137         if (isEmpty()) {
138             System.out.println("Node is not found");
139             return false;
140         }
141         Node27 current = root;
142         while (current != null) {
143             if (current.data.compareTo(data) == 0) {
144                 return true;
145             } else if (current.data.compareTo(data) < 0) {
146                 current = current.left;
147             } else if (current.data.compareTo(data) > 0) {
148                 current = current.right;
149             }
150         }
151         return false;
152     }
153
154     void delete(Node27 data) {
155         if (isEmpty()) {
156             System.out.println("Node is not found");
157             return;
158         }
159         Node27 current = root;
160         while (current != null) {
161             if (current.data.compareTo(data) < 0) {
162                 current = current.left;
163             } else if (current.data.compareTo(data) > 0) {
164                 current = current.right;
165             } else {
166                 // Node to be deleted is found
167                 if (current.left != null) {
168                     // Node has left child
169                     Node27 leftChild = current.left;
170                     while (leftChild.right != null) {
171                         leftChild = leftChild.right;
172                     }
173                     // Replace current node with its left child
174                     current.data = leftChild.data;
175                     current.left = leftChild.left;
176                     current.right = leftChild.right;
177                     leftChild = null;
178                 } else {
179                     // Node has no left child
180                     // Replace current node with its right child
181                     current.data = current.right.data;
182                     current.left = current.right.left;
183                     current.right = current.right.right;
184                     current.right = null;
185                 }
186             }
187         }
188     }
189
190     boolean findNode(Node27 data) {
191         if (isEmpty()) {
192             System.out.println("Node is not found");
193             return false;
194         }
195         Node27 current = root;
196         while (current != null) {
197             if (current.data.compareTo(data) == 0) {
198                 return true;
199             } else if (current.data.compareTo(data) < 0) {
200                 current = current.left;
201             } else if (current.data.compareTo(data) > 0) {
202                 current = current.right;
203             }
204         }
205         return false;
206     }
207
208     void delete(Node27 data) {
209         if (isEmpty()) {
210             System.out.println("Node is not found");
211             return;
212         }
213         Node27 current = root;
214         while (current != null) {
215             if (current.data.compareTo(data) < 0) {
216                 current = current.left;
217             } else if (current.data.compareTo(data) > 0) {
218                 current = current.right;
219             } else {
220                 // Node to be deleted is found
221                 if (current.left != null) {
222                     // Node has left child
223                     Node27 leftChild = current.left;
224                     while (leftChild.right != null) {
225                         leftChild = leftChild.right;
226                     }
227                     // Replace current node with its left child
228                     current.data = leftChild.data;
229                     current.left = leftChild.left;
230                     current.right = leftChild.right;
231                     leftChild = null;
232                 } else {
233                     // Node has no left child
234                     // Replace current node with its right child
235                     current.data = current.right.data;
236                     current.left = current.right.left;
237                     current.right = current.right.right;
238                     current.right = null;
239                 }
240             }
241         }
242     }
243
244     boolean findNode(Node27 data) {
245         if (isEmpty()) {
246             System.out.println("Node is not found");
247             return false;
248         }
249         Node27 current = root;
250         while (current != null) {
251             if (current.data.compareTo(data) == 0) {
252                 return true;
253             } else if (current.data.compareTo(data) < 0) {
254                 current = current.left;
255             } else if (current.data.compareTo(data) > 0) {
256                 current = current.right;
257             }
258         }
259         return false;
260     }
261
262     void delete(Node27 data) {
263         if (isEmpty()) {
264             System.out.println("Node is not found");
265             return;
266         }
267         Node27 current = root;
268         while (current != null) {
269             if (current.data.compareTo(data) < 0) {
270                 current = current.left;
271             } else if (current.data.compareTo(data) > 0) {
272                 current = current.right;
273             } else {
274                 // Node to be deleted is found
275                 if (current.left != null) {
276                     // Node has left child
277                     Node27 leftChild = current.left;
278                     while (leftChild.right != null) {
279                         leftChild = leftChild.right;
280                     }
281                     // Replace current node with its left child
282                     current.data = leftChild.data;
283                     current.left = leftChild.left;
284                     current.right = leftChild.right;
285                     leftChild = null;
286                 } else {
287                     // Node has no left child
288                     // Replace current node with its right child
289                     current.data = current.right.data;
290                     current.left = current.right.left;
291                     current.right = current.right.right;
292                     current.right = null;
293                 }
294             }
295         }
296     }
297
298     boolean findNode(Node27 data) {
299         if (isEmpty()) {
300             System.out.println("Node is not found");
301             return false;
302         }
303         Node27 current = root;
304         while (current != null) {
305             if (current.data.compareTo(data) == 0) {
306                 return true;
307             } else if (current.data.compareTo(data) < 0) {
308                 current = current.left;
309             } else if (current.data.compareTo(data) > 0) {
310                 current = current.right;
311             }
312         }
313         return false;
314     }
315
316     void delete(Node27 data) {
317         if (isEmpty()) {
318             System.out.println("Node is not found");
319             return;
320         }
321         Node27 current = root;
322         while (current != null) {
323             if (current.data.compareTo(data) < 0) {
324                 current = current.left;
325             } else if (current.data.compareTo(data) > 0) {
326                 current = current.right;
327             } else {
328                 // Node to be deleted is found
329                 if (current.left != null) {
330                     // Node has left child
331                     Node27 leftChild = current.left;
332                     while (leftChild.right != null) {
333                         leftChild = leftChild.right;
334                     }
335                     // Replace current node with its left child
336                     current.data = leftChild.data;
337                     current.left = leftChild.left;
338                     current.right = leftChild.right;
339                     leftChild = null;
340                 } else {
341                     // Node has no left child
342                     // Replace current node with its right child
343                     current.data = current.right.data;
344                     current.left = current.right.left;
345                     current.right = current.right.right;
346                     current.right = null;
347                 }
348             }
349         }
350     }
351
352     boolean findNode(Node27 data) {
353         if (isEmpty()) {
354             System.out.println("Node is not found");
355             return false;
356         }
357         Node27 current = root;
358         while (current != null) {
359             if (current.data.compareTo(data) == 0) {
360                 return true;
361             } else if (current.data.compareTo(data) < 0) {
362                 current = current.left;
363             } else if (current.data.compareTo(data) > 0) {
364                 current = current.right;
365             }
366         }
367         return false;
368     }
369
370     void delete(Node27 data) {
371         if (isEmpty()) {
372             System.out.println("Node is not found");
373             return;
374         }
375         Node27 current = root;
376         while (current != null) {
377             if (current.data.compareTo(data) < 0) {
378                 current = current.left;
379             } else if (current.data.compareTo(data) > 0) {
380                 current = current.right;
381             } else {
382                 // Node to be deleted is found
383                 if (current.left != null) {
384                     // Node has left child
385                     Node27 leftChild = current.left;
386                     while (leftChild.right != null) {
387                         leftChild = leftChild.right;
388                     }
389                     // Replace current node with its left child
390                     current.data = leftChild.data;
391                     current.left = leftChild.left;
392                     current.right = leftChild.right;
393                     leftChild = null;
394                 } else {
395                     // Node has no left child
396                     // Replace current node with its right child
397                     current.data = current.right.data;
398                     current.left = current.right.left;
399                     current.right = current.right.right;
400
```



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

PERTANYAAN

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

Jawab:

Metode traverse pada binary tree preorder, postorder, dan inorder mengecek setiap node dalam urutan tertentu, tetapi tidak memberikan keuntungan efisiensi dalam pencarian karena method tersebut mungkin harus mengecek semua node.

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

Jawab:

Left dan right dalam class node digunakan untuk mendefinisikan child dari node yang sekarang, jika left maka untuk left child jika right untuk right child

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

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

Jawab:

- a. atribut root digunakan untuk mendefinisikan sebuah induk/parent dalam suatu tree
- b. ketika object tree pertama kali dibuat, nilai root adalah null

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

Jawab:

Jika tree masih kosong node baru akan ditambahkan dan node baru tersebut akan menjadi root atau parent.



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
if(data<current.data){  
    if(current.left!=null){  
        current = current.left;  
    }else{  
        current.left = new Node(data);  
        break;  
    }  
}
```

Jawab:

Kode pada soal adalah salah, kode yang benar adalah seperti kode dibawah.

```
if (data < current.data) {  
    if (current.left == null) {  
        current.left = new Node27(data);  
        break;  
    } else {  
        current = current.left;  
    }  
}
```

Pada baris pertama terdapat pengecekan apakah data yang akan diinputkan lebih kecil dari data yang dibandingkan saat ini, jika benar akan dicek kembali apakah left child dari current tersebut bernilai null, jika benar maka data baru akan menempati posisi tersebut, jika nilai left dari current tidak null maka nilai current digantikan dengan nilai left dari current.



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

Percobaan 2

InOrder Traversal : 3 4 5 6 7 8 9

```
1 package p14;  
2  
3 public class BinaryTreeArrayMain27 {  
4     public static void main(String[] args) {  
5         BinaryTreeArray27 bta = new BinaryTreeArray27();  
6         int[] data = { 6, 4, 8, 3, 5, 7, 9, 0, 0, 0 };  
7         int idxLast = 6;  
8         bta.populateData(data, idxLast);  
9         System.out.print("\nInOrder Traversal : ");  
10        bta.traverseInOrder(0);  
11        System.out.println("\n");  
12    }  
13 }  
14 }  
15
```

```
1 package p14;  
2  
3 public class BinaryTreeArray27 {  
4     int[] data;  
5     int idxLast;  
6  
7     public BinaryTreeArray27() {  
8         data = new int[10];  
9     }  
10  
11     void populateData(int data[], int idxLast) {  
12         this.data = data;  
13         this.idxLast = idxLast;  
14     }  
15  
16     void traverseInOrder(int idxStart) {  
17         if (idxStart <= idxLast) {  
18             traverseInOrder(2 * idxStart + 1);  
19             System.out.print(data[idxStart] + " ");  
20             traverseInOrder(2 * idxStart + 2);  
21         }  
22     }  
23 }  
24 }  
25
```



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

Pertanyaan

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

Jawab:

Atribut data digunakan untuk menyimpan data dalam array. Atribut idxLast digunakan untuk menentukan batas indeks array tersebut

2. Apakah kegunaan dari method populateData()?

Jawab:

Method tersebut digunakan untuk mengisi data pada objek yang telah dibuat yaitu bta dan batas indeks tersebut juga ditentukan dengan parameter method tersebut.

3. Apakah kegunaan dari method traverseInOrder()?

Jawab:

traverseInOrder adalah method penelusuran seluruh node pada binary tree dengan pengecekan dimulai pada indeks yang dikirimkan dengan argumen pada parameter method tersebut

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan right child masing-masing?

Jawab:

Dengan i adalah 2 maka:

Anak kiri diperoleh dengan $2*i+1 = 5 \Rightarrow$ pada indeks ke-5

Anak kanan diperoleh dengan $2*i+2 = 6 \Rightarrow$ pada indeks ke-6

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

Jawab:

Statement tersebut digunakan untuk menunjukkan indeks valid terakhir yang terisi dalam array data.



NAMA : Roy Wijaya
NIM : 2341720120
KELAS : 1-G
MATERI : Tree (Jobsheet 11)

LINK GITHUB:

https://github.com/RoyW12/AlgoritmaStrukturData_1G_28/tree/main/src/P14