JOBSHEET XIII
TREE

1. Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model *Tree* khususnya *Binary Tree*
2. membuat dan mendeklarasikan struktur algoritma *Binary Tree*.
3. menerapkan dan mengimplementasikan algoritma *Binary Tree* dalam kasus *Binary Search Tree*

2. Praktikum

2.1 Implementasi Binary Search Tree menggunakan Linked List

2.1.1 Tahapan percobaan

Waktu Percobaan (45 menit)

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan array (praktikum 2) dan linked list (praktikum 1). Sebelumnya, akan dibuat class Node, dan Class BinaryTree

Node
data: int left: Node right: Node
Node(left: Node, data:int, right:Node)

BinaryTree
root: Node size : int
DoubleLinkedLists() add(data: int): void find(data: int) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) add(item: int, index:int): void delete(data: int): void

1. Buatlah class **Node**, **BinaryTree** dan **BinaryTreeMain**
2. Di dalam class **Node**, tambahkan atribut **data**, **left** dan **right**, serta konstruktor default dan berparameter.

```
3   public class Node {
4       int data;
5       Node left;
6       Node right;
7
8       public Node(){
9       }
10      public Node(int data){
11          this.left = null;
12          this.data = data;
13          this.right = null;
14      }
15  }
```

3. Di dalam class **BinaryTree**, tambahkan atribut **root**.

```
3   public class BinaryTree {
4       Node root;
5
```

4. Tambahkan konstruktor default dan method **isEmpty()** di dalam class **BinaryTree**

```
6       public BinaryTree(){
7           root = null;
8       }
9       boolean isEmpty(){
10          return root==null;
11      }
```

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.

```
12      void add(int data){
13          if(isEmpty()){//tree is empty
14              root = new Node(data);
15          }else{
16              Node current = root;
17              while(true){
18                  if(data<current.data){
19                      if(current.left!=null){
20                          current = current.left;
21                      }else{
22                          current.left = new Node(data);
23                          break;
24                      }
25                  }else if(data>current.data){
26                      if(current.right!=null){
27                          current = current.right;
28                      }else{
29                          current.right = new Node(data);
30                          break;
31                      }
32                  }else{//data is already exist
33                      break;
34                  }
35              }
36          }
37      }
```

6. Tambahkan method find()

```
38      boolean find(int data){
39          boolean hasil = false;
40          Node current = root;
41          while(current!=null){
42              if(current.data==data){
43                  hasil = true;
44                  break;
45              }else if(data<current.data){
46                  current = current.left;
47              }else{
48                  current = current.right;
49              }
50          }
51          return hasil;
52      }
```

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.

```
53      void traversePreOrder(Node node) {
54          if (node != null) {
55              System.out.print(" " + node.data);
56              traversePreOrder(node.left);
57              traversePreOrder(node.right);
58          }
59      }
60      void traversePostOrder(Node node) {
61          if (node != null) {
62              traversePostOrder(node.left);
63              traversePostOrder(node.right);
64              System.out.print(" " + node.data);
65          }
66      }
67      void traverseInOrder(Node node) {
68          if (node != null) {
69              traverseInOrder(node.left);
70              System.out.print(" " + node.data);
71              traverseInOrder(node.right);
72          }
73      }
```

8. Tambahkan method **getSuccessor()**. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
74      Node getSuccessor(Node del){
75          Node successor = del.right;
76          Node successorParent = del;
77          while(successor.left!=null){
78              successorParent = successor;
79              successor = successor.left;
80          }
81          if(successor!=del.right){
82              successorParent.left = successor.right;
83              successor.right = del.right;
84          }
85          return successor;
86      }
```

9. Tambahkan method **delete()**.

```
87      void delete(int data){
88
89      }
```

Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```
88              if(isEmpty()){
89                  System.out.println("Tree is empty!");
90                  return;
91              }
92              //find node (current) that will be deleted
93              Node parent = root;
94              Node current = root;
95              boolean isLeftChild = false;
96              while(current!=null){
97                  if(current.data==data){
98                      break;
99                  }else if(data<current.data){
100                     parent = current;
101                     current = current.left;
102                     isLeftChild = true;
103                 }else if(data>current.data){
104                     parent = current;
105                     current = current.right;
106                     isLeftChild = false;
107                 }
108             }
```

Kemudian tambahkan proses penghapusan terhadap node current yang telah ditemukan.

```
109             //deletion
110             if(current==null){
111                 System.out.println("Couldn't find data!");
113                 return;
114             }else{
115                 //if there is no child, simply delete it
116                 if(current.left==null&&current.right==null){
117                     if(current==root){
118                         root  = null;
119                     }else{
120                         if(isLeftChild){
121                             parent.left = null;
122                         }else{
123                             parent.right = null;
124                         }
125                     }
126                 }else if(current.left==null){//if there is 1 child (right)
127                     if(current==root){
128                         root = current.right;
129                     }else{
130                         if(isLeftChild){
131                             parent.left = current.right;
132                         }else{
133                             parent.right = current.right;
134                         }
135                     }
136                 }else if(current.right==null){//if there is 1 child (left)
137                     if(current==root){
138                         root = current.left;
139                     }else{
140                         if(isLeftChild){
141                             parent.left = current.left;
142                         }else{
143                             parent.right = current.left;
144                         }
145                     }
146                 }else{//if there is 2 childs
147                     Node successor = getSuccessor(current);
148                     if(current==root){
149                         root = successor;
150                     }else{
151                         if(isLeftChild){
152                             parent.left = successor;
153                         }else{
154                             parent.right = successor;
155                         }
156                         successor.left = current.left;
157                     }
158                 }
```

10. Buka class BinaryTreeMain dan tambahkan method main().

```
3   public class BinaryTreeMain {
4       public static void main(String[] args) {
5           BinaryTree bt = new BinaryTree();
6
7           bt.add(6);
8           bt.add(4);
9           bt.add(8);
10          bt.add(3);
11          bt.add(5);
12          bt.add(7);
13          bt.add(9);
14          bt.add(10);
15          bt.add(15);
16
17          bt.traversePreOrder(bt.root);
18          System.out.println("");
19          bt.traverseInOrder(bt.root);
20          System.out.println("");
21          bt.traversePostOrder(bt.root);
22          System.out.println("");
23          System.out.println("Find "+bt.find(5));
24          bt.delete(8);
25          bt.traversePreOrder(bt.root);
26          System.out.println("");
27      }
28  }
```

11. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
12. Amati hasil running tersebut.

2.1.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
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;
        }
```

}

2.2 Implementasi binary tree dengan array

Waktu percobaan: 45 menit

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().**

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

4. Kemudian dalam class **BinaryTreeArrayMain** buat method main() seperti gambar berikut ini.

```
3   public class BinaryTreeArrayMain {
4       public static void main(String[] args) {
5           BinaryTreeArray bta = new BinaryTreeArray();
6           int[] data = {6,4,8,3,5,7,9,0,0,0};
7           int idxLast = 6;
8           bta.populateData(data, idxLast);
9           bta.traverseInOrder(0);
10      }
11  }
```

5. Jalankan class **BinaryTreeArrayMain** dan amati hasilnya!

13.2.1 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
2. Apakah kegunaan dari method **populateData()**?
3. Apakah kegunaan dari method **traverseInOrder()**?

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

13.3 Tugas Praktikum

Waktu pengerjaan: 90 menit

1. Buat method di dalam class **BinaryTree** yang akan menambahkan node dengan cara rekursif.
2. Buat method di dalam class **BinaryTree** untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.
3. Buat method di dalam class **BinaryTree** untuk menampilkan data yang ada di leaf.
4. Buat method di dalam class **BinaryTree** untuk menampilkan berapa jumlah leaf yang ada di dalam tree.
5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
   - method **add(int data)** untuk memasukan data ke dalam tree
   - method **traversePreOrder()** dan **traversePostOrder()**