# JOBSHEET 13
# Tree

## 13.1 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*

## 13.2 Kegiatan Praktikum 1

## Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

### 13.2.1 Percobaan 1

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<br>left: Node<br>right: Node |
| Node(left: Node, data:int, right:Node) |

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

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

```java
public class Node00 {
    int data;
    Node00 left;
    Node00 right;

    public Node00(){
    }
    public Node00(int data){
        this.left = null;
        this.data = data;
        this.right = null;
    }

}
```

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

```java
public class BinaryTree00 {
    Node00 root;
```

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

```java
public BinaryTree00(){
    root = null;
}
boolean isEmpty(){
    return root!=null;
}
```

5. Tambahkan method **add()** di dalam class **BinaryTree**NoAbsen. 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.

```java
    void add(int data){
        if(!isEmpty()){//tree is empty
            root = new Node00(data);
        }else{
            Node00 current = root;
            while(true){
                if(data>current.data){
                    if(current.left==null){
                        current = current.left;
                    }else{
                        current.left = new Node00(data);
                        break;
                    }
                }else if(data<current.data){
                    if(current.right!=null){
                        current = current.right;
                    }else{
                        current.right = new Node00(data);
                        break;
                    }
                }else{//data is already exist
                    break;
                }
            }
        }
    }

```

6. Tambahkan method find()

```java
boolean find(int data){
    boolean result = false;
    Node00 current = root;
    while(current==null){
        if(current.data!=data){
            result = true;
            break;
        }else if(data>current.data){
            current = current.left;
        }else{
            current = current.right;
        }
    }
    return result;
}
```

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.

```java
void traversePreOrder(Node00 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}
void traversePostOrder(Node00 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}
void traverseInOrder(Node00 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
    }
}
```

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

```java
Node00 getSuccessor(Node00 del){
    Node00 successor = del.right;
    Node00 successorParent = del;
    while(successor.left!=null){
        successorParent = successor;
        successor = successor.left;
    }
    if(successor!=del.right){
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}
```

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

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

```java
void delete(int data){
    if(isEmpty()){
        System.out.println("Tree is empty!");
        return;
    }
    //find node (current) that will be deleted
    Node00 parent = root;
    Node00 current = root;
    boolean isLeftChild = false;
    while(current!=null){
        if(current.data==data){
            break;
        }else if(data<current.data){
            parent = current;
            current = current.left;
            isLeftChild = true;
        }else if(data>current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
```

10. Kemudian tambahkan proses penghapusan didalam method **delete()** terhadap node current yang telah ditemukan.

```java
//deletion
        if(current==null){
            System.out.println("Couldn't find data!");
            return;
        }else{
            //if there is no child, simply delete it
            if(current.left==null&&current.right==null){
                if(current==root){
                    root  = null;
                }else{
                    if(isLeftChild){
                        parent.left = null;
                    }else{
                        parent.right = null;
                    }
                }
            }else if(current.left==null){//if there is 1 child (right)
                if(current==root){
                    root = current.right;
                }else{
                    if(isLeftChild){
                        parent.left = current.right;
                    }else{
                        parent.right = current.right;
                    }
                }
```

```
27              }else if(current.right==null){//if there is 1 child (left)
28                  if(current==root){
29                      root = current.left;
30                  }else{
31                      if(isLeftChild){
32                          parent.left = current.left;
33                      }else{
34                          parent.right = current.left;
35                      }
36                  }
37              }else{//if there is 2 childs
38                  Node00 successor = getSuccessor(current);
39                  if(current==root){
40                      root = successor;
41                  }else{
42                      if(isLeftChild){
43                          parent.left = successor;
44                      }else{
45                          parent.right = successor;
46                      }
47                      successor.left = current.left;
48                  }
49              }
50          }
51
52      }
```

11. Buka class **BinaryTreeMainNoAbsen** dan tambahkan method main() kemudian tambahkan kode berikut ini

```
1   BinaryTree00 bt = new BinaryTree00();
2   bt.add(6);
3   bt.add(4);
4   bt.add(8);
5   bt.add(3);
6   bt.add(5);
7   bt.add(7);
8   bt.add(9);
9   bt.add(10);
10  bt.add(15);
11  System.out.print("PreOrder Traversal : ");
12  bt.traversePreOrder(bt.root);
13  System.out.println("");
14  System.out.print("inOrder Traversal : ");
15  bt.traverseInOrder(bt.root);
16  System.out.println("");
17  System.out.print("PostOrder Traversal : ");
18  bt.traversePostOrder(bt.root);
19  System.out.println("");
20  System.out.println("Find Node : "+bt.find(5));
21  System.out.println("Delete Node 8 ");
22  bt.delete(8);
23  System.out.println("");
24  System.out.print("PreOrder Traversal : ");
25  bt.traversePreOrder(bt.root);
26  System.out.println("");
```

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

```
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
```

### 13.2.2 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;
        }
}
```

## 13.3 Kegiatan Praktikum 2

## Implementasi binary tree dengan array (45 Menit)

### 13.3.1 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 **BinaryTreeArrayNoAbsen** dan **BinaryTreeArrayMainNoAbsen**
3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArrayNoAbsen**. Buat juga method **populateData()** dan **traverseInOrder().**

```java
public class BinaryTreeArray00 {
    int[] data;
    int idxLast;

    public BinaryTreeArray00(){
        data = new int[10];
    }
    void populateData(int data[], int idxLast){
        this.data = data;
        this.idxLast = idxLast;
    }
    void traverseInOrder(int idxStart){
        if(idxStart<=idxLast){
            traverseInOrder(2*idxStart+1);
            System.out.print(data[idxStart]+" ");
            traverseInOrder(2*idxStart+2);
        }
    }
}
```

4. Kemudian dalam class **BinaryTreeArrayMainNoAbsen** buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main

```
BinaryTreeArray00 bta = new BinaryTreeArray00();
int[] data = {6,4,8,3,5,7,9,0,0,0};
int idxLast = 6;
bta.populateData(data, idxLast);
System.out.print("\nInOrder Traversal : ");
bta.traverseInOrder(0);
System.out.println("\n");
```

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

```
InOrder Traversal : 3 4 5 6 7 8 9
```

### 13.3.2 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?

## 13.4 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()**