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 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 **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;
    }

}
```

```java
public class Node26 {
    int data;
    Node26 left, right;

    Node26 (){

    }
    Node26 (int data){
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

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

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

```java
public class BinaryTree26 {
    Node26 root;

}
```

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

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

```java
    BinaryTree26(){
        root = null;
    }

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

5. Tambahkan method **add()** di dalam class **BinaryTreeNoAbsen**. 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.

```
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;
            }
        }
    }
}
```

```
void add(int data){
    if (!isEmpty()) {
        root = new Node26(data);
    } else {
        Node26 current = root;
        while (true) {
            if (data > current.data) {
                if (current.right == null) {
                    current = current.left;
                } else {
                    current.left =new Node26();
                    break;
                }
            } else if (data < current.data) {
                if (current.right != null) {
                    current = current.right;
                    break;
                } else {
                    current.right = new Node26();
                    break;
                }
            } else {
                break;
            }
        }
    }
}
}
```

6. Tambahkan method find()

```
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;
}
```

```
boolean find(int data){
    boolean result = false;
    Node26 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 false;
}
```

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.

```
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);
    }
}
```

```
void traversePreOrder(Node26 node){
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder(Node26 node){
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void traverseInOrder(Node26 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.

```
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;
}
```

```
Node26 getSuccessor(Node26 del){
    Node26 successor = del.right;
    Node26 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;
        }
    }
```

```java
void delete(int data){
    if (isEmpty()) {
        System.out.println(x:"Tree is Empty");
        return;
    }

    Node26 parent = root;
    Node26 current = root;
    boolean isLeftChild = false;

    while (current != null) {
        if (current.data == data) {
            break;
        } else if (data < current.data) {
            parent = current;
            current = cu
            isLeftChild
        } else if (data>current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
```

Node26 current - BinaryTre

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

```
//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;
                    }
                }
            }else if(current.right==null){//if there is 1 child (left)
                if(current==root){
                    root = current.left;
                }else{
                    if(isLeftChild){
                        parent.left = current.left;
                    }else{
                        parent.right = current.left;
                    }
                }
            }else{//if there is 2 childs
                Node00 successor = getSuccessor(current);
                if(current==root){
                    root = successor;
                }else{
                    if(isLeftChild){
                        parent.left = successor;
                    }else{
                        parent.right = successor;
                    }
                    successor.left = current.left;
                }
            }
        }

    }
```

```
if (current == null) {
                    System.out.println("Couldn't find data");
                    return;
                }else{

                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 (current == root) {
                        root = current.right;
                    } else {
                        if (isLeftChild) {
                            parent.left = current.right;
                        } else {
                            parent.right = current.right;
                        }
                    }
                } else if (current.right == null) {
                    if (current == root) {
                        root = current.left;
                    } else {
                        if (isLeftChild) {
                            parent.left = current.left;
                        } else {
                            parent.right = current.left;
                        }
                    }
                } else {
                    Node26 successor = getSuccessor(current);
                    if (current == root) {
                        root = successor;
                    } else {
                        if (isLeftChild) {
                            parent.left = successor;
                        } else {
                            parent.right = successor;
                        }
                    }
                    successor.left = current.left;
                }
            }
            }
```

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

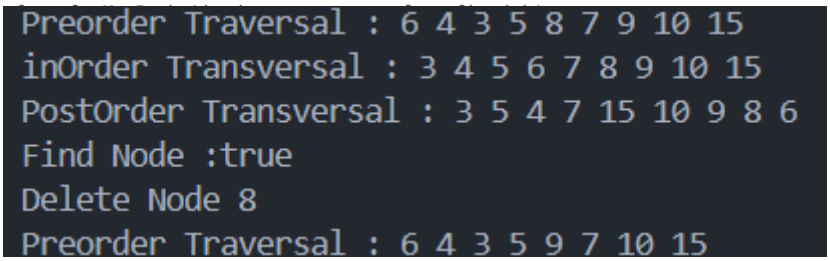
```
BinaryTree00 bt = new BinaryTree00();
bt.add(6);
bt.add(4);
bt.add(8);
bt.add(3);
bt.add(5);
bt.add(7);
bt.add(9);
bt.add(10);
bt.add(15);
System.out.print("PreOrder Traversal : ");
bt.traversePreOrder(bt.root);
System.out.println("");
System.out.print("inOrder Traversal : ");
bt.traverseInOrder(bt.root);
System.out.println("");
System.out.print("PostOrder Traversal : ");
bt.traversePostOrder(bt.root);
System.out.println("");
System.out.println("Find Node : "+bt.find(5));
System.out.println("Delete Node 8 ");
bt.delete(8);
System.out.println("");
System.out.print("PreOrder Traversal : ");
bt.traversePreOrder(bt.root);
System.out.println("");
```

```
public static void main(String[] args) {
    BinaryTree26 bt = new BinaryTree26();
    bt.add(data:6);
    bt.add(data:4);
    bt.add(data:8);
    bt.add(data:3);
    bt.add(data:5);
    bt.add(data:7);
    bt.add(data:9);
    bt.add(data:10);
    bt.add(data:15);
    System.out.print(s:"Preorder Traversal :");
    bt.traversePreOrder(bt.root);
    System.out.println(x:"");
    System.out.print(s:"inOrder Transversal :");
    bt.traverseInOrder(bt.root);
    System.out.println(x:"");
    System.out.print(s:"PostOrder Transversal :");
    bt.traversePostOrder(bt.root);
    System.out.println(x:"");
    System.out.println("Find Node :"+bt.find(data:5));
    System.out.println(x:"Delete Node 8");
    bt.delete(data:8);
    System.out.print(s:"");
    System.out.print(s:"Preorder Traversal :");
    bt.traversePreOrder(bt.root);
    System.out.println(x:"");
```

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?

   Dalam binary search tree (BST), setiap node memiliki sifat bahwa semua nilai di subtree kiri lebih kecil dan semua nilai di subtree kanan lebih besar dari nilai node tersebut. Ini memungkinkan pencarian dilakukan secara efisien dengan membandingkan nilai yang dicari dengan nilai node saat ini dan mengarahkan pencarian ke subtree yang sesuai

2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   Atribut left dan right pada class Node digunakan untuk menunjukkan child node dari suatu node tertentu dalam struktur data pohon

3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   Atribut root dalam class BinaryTree digunakan untuk menyimpan referensi ke node pertama (atau root) dari pohon biner. Ini memungkinkan akses langsung ke seluruh struktur pohon.

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
   Ketika objek tree pertama kali dibuat, nilai dari root adalah null karena pada awalnya pohon belum memiliki node apapun.

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   Ketika tree masih kosong dan akan ditambahkan sebuah node baru, node tersebut akan menjadi root dari pohon tersebut. root akan menunjuk langsung ke node baru tersebut, dan subtree left dan right dari root akan diatur menjadi null

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;
        }
 }
```

   Baris program ini bertanggung jawab untuk menambahkan node baru ke subtree kiri dari node current jika nilai data yang akan ditambahkan lebih kecil dari nilai data dari current.

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);
        }
    }
}
```

```java
public class BinaryTreeArray26 {
    int [] data;
    int idxLast;

    BinaryTreeArray26(){
        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");
```

```
public class BinaryTreeArrayMain26 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray26 bta = new BinaryTreeArray26();

        int[] Data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(Data, idxLast);

        System.out.print(s:"Inorder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}
```

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

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

   data digunakan untuk menyimpan elemen-elemen dari binary tree dalam bentuk array.

   idxLast menyimpan indeks terakhir dari data yang valid dalam array data.

2. Apakah kegunaan dari method **populateData()**?
   Method populateData() digunakan untuk mengisi array data dengan data yang diberikan dan mengatur idxLast untuk menunjukkan indeks terakhir data yang valid dalam array.

3. Apakah kegunaan dari method **traverseInOrder()**?
   Method traverseInOrder() digunakan untuk melakukan traversal inorder pada binary tree yang direpresentasikan dalam bentuk array. Ini mencetak elemen-elemen dalam urutan terurut dari kiri ke kanan.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   Jika node binary tree disimpan dalam array di indeks i, maka left child berada di indeks 2*i + 1 dan right child berada di indeks 2*i + 2.

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Statement int idxLast = 6 mengatur idxLast untuk menunjukkan bahwa elemen terakhir dari array Data yang valid adalah di indeks 6. Hal ini memastikan bahwa traversal hanya dilakukan pada data yang valid dalam array.

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

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

```
Node26 addRecursive(Node26 current, int data) {
    if (current == null) {
        return new Node26(data);
    }
    if (data < current.data) {
        current.left = addRecursive(current.left, data);
    } else if (data > current.data) {
        current.right = addRecursive(current.right, data);
    }
    return current;
}
```

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

```
int findMin() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty");
    }
    Node26 current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current.data;
}

int findMax() {
    if (isEmpty()) {
        System.out.println(x:"Tree Is Empty");
    }
    Node26 current = root;
    while (current.right != null) {
        current = current.right;
    }
    return current.data;
}
```

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

```
void printLeaf(Node26 node) {
    if (node != null) {

        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        } else {
            printLeaf(node.left);
            printLeaf(node.right);
        }

    }
}
```

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

```
int countLeaf(Node26 node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    return countLeaf(node.left) + countLeaf(node.right);
}
```

Binary tree main :

```
package Latihan;

public class BinaryTreeMain26 {
    public static void main(String[] args) {
        BinaryTree26 bt = new BinaryTree26();
        bt.add(5);
        bt.add(9);
        bt.add(8);
        bt.add(11);
        bt.add(5);
        bt.add(7);
        bt.add(12);
        bt.add(10);
        bt.add(15);
        System.out.print("Preorder Traversal :");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("inOrder Transversal :");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("PostOrder Transversal :");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find Node :"+bt.find(5));
        System.out.println("Delete Node 8");
        bt.delete(8);
        System.out.print("");
        System.out.print("Preorder Traversal :");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("Find Min Node :"+bt.findMin());
        System.out.println("");
        System.out.print("Find Max Node :"+bt.findMax());
        System.out.println("");
        System.out.print("Count leaf nodes :"+bt.countLeaf(bt.root));
        System.out.println("");
        bt.addRecursive(bt.root, 19);
        System.out.print("InOrder Transversal :");
        bt.traverseInOrder(bt.root);
    }
}
```

Hasil :

```
Preorder Traversal : 5 9 8 7 11 10 12 15
inOrder Transversal : 5 7 8 9 10 11 12 15
PostOrder Transversal : 7 8 10 15 12 11 9 5
Find Node :true
Delete Node 8
Preorder Traversal : 5 9 7 11 10 12 15
Find Min Node :5
Find Max Node :15
Count leaf nodes :3
InOrder Transversal : 5 7 9 10 11 12 15 19
```

5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
   - method **add(int data)** untuk memasukan data ke dalam tree

```
void add(int value) {
    if (idxLast >= data.length - 1) {
        System.out.println(x:"Tree is full");
        return;
    }
    data[++idxLast] = value;
}
```

   - method **traversePreOrder()** dan **traversePostOrder()**

```
    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(data[idxStart] + " ");
            traversePreOrder(2 * idxStart + 1);
            traversePreOrder(2 * idxStart + 2);
        }
    }

    void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traversePostOrder(2 * idxStart + 1);
            traversePostOrder(2 * idxStart + 2);
            System.out.print(data[idxStart] + " ");
        }
    }
}
```

BinaryTreeArrayMain:

```java
package Latihan;

public class BinaryTreeArrayMain26 {
    public static void main(String[] args) {
        BinaryTreeArray26 bta = new BinaryTreeArray26();

        int[] initialData = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0, 0};
        int idxLast = 9;
        bta.populateData(initialData, idxLast);

        System.out.print("Inorder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("\n");

        System.out.print("Preorder Traversal: ");
        bta.traversePreOrder(0);
        System.out.println("\n");

        System.out.print("Postorder Traversal: ");
        bta.traversePostOrder(0);
        System.out.println("\n");

        bta.add(23);

        System.out.print("Inorder Traversal setelah menambahkan 23: ");
        bta.traverseInOrder(0);
        System.out.println();
    }
}
```

Hasil:

```
Inorder Traversal : 0 3 0 4 0 5 6 7 8 9

Preorder Traversal: 6 4 3 0 0 5 0 8 7 9

Postorder Traversal: 0 0 3 0 5 4 7 9 8 6

Inorder Traversal setelah menambahkan 10: 0 3 0 4 0 5 23 6 7 8 9
```