Algoritma dan Struktur Data 2

Modul 8
Binary Tree (Pohon Biner)

Disusun oleh:

Dwi Intan Af'idah, S.T., M.Kom

PROGRAM STUDI TEKNIK INFORMATIKA
POLITEKNIK HARAPAN BERSAMA
TAHUN AJARAN 2020/2021

Daftar Isi

1 Binary Tree

1.1 Target Pembelajaran

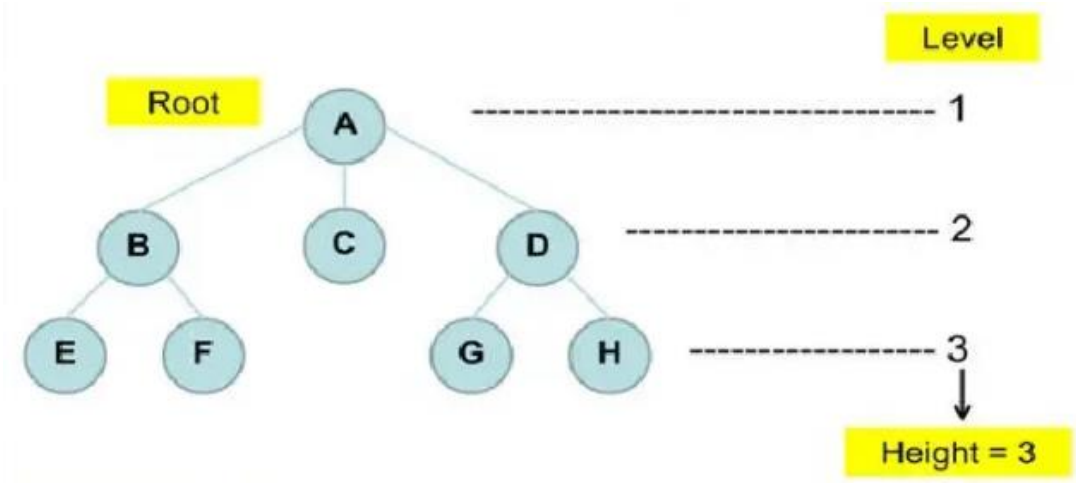
1. Memahami konsep Binary Tree.
2. Memahami konsep pembacaan Binary Tree dengan traversal Inorder, Preorder, dan PostOrder.
3. Mengimplementasikan pembacaan Binary Tree dengan traversal Inorder, Preorder, dan PostOrder.

1.2 Dasar Teori

1.2.1 Binary Tree

Sebuah binary tree adalah sebuah pengorganisasian secara hirarki dari beberapa buah simpul, dimana masing-masing simpul tidak mempunyai anak lebih dari 2. Simpul yang berada di bawah sebuah simpul dinamakan anak dari simpul tersebut. Simpul yang berada di atas sebuah simpul dinamakan induk dari simpul tersebut.

Masing-masing simpul dalam binary tree terdiri dari tiga bagian yaitu sebuah data dan dua buah pointer yang dinamakan pointer kiri dan kanan. Ilustrasi dari binary tree ditunjukkan pada gambar di bawah ini.



Keterangan Ilustrasi Binary Tree:

1. Predesesor

   Kode yang berada diatas node tertentu.

   Contoh: B merupakan predesesor E dan F.

2. Succesor

   Kode yang berada dibawah node tertentu.

   Contoh: E dan F merupakan succesor dari B.

3. Ancestor

   Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.

   Contoh: A dan B merupakan ancestor dari F.

4. Descendant

   Seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.

   Contoh: F dan B merupakan ancestor dari A.

5. Parent

   Predesesor satu level di atas satu node.

   Contoh: B merupakan parent dari F.

6. Child

   Succesor satu level di bawah satu node.

   Contoh: F merupakan child dari B.

7. Sibling

   Node yang memiliki parent yang sama dengan satu node.

   Contoh: E dan F adalah sibling.

8. Subtree

   Bagian tree yang berupa satu node besert descendant-nya (contoh: Subtree B, E, F dan Subtree D, G, H)

9. Size

   Banyaknya node dalam suatu tree.

   Contoh: gambar tree di atas memiliki siza=8.

10. Height

    Banyaknya tingkat/level dalam suatu tree.

    Contoh: gambar tree di atas memiliki height=3.

11. Root (Akar)

    Node khusus dalam tree yang tidak memiliki predesesor.

    Contoh: A

12. Leaf (Daun)

    Node-node dalam tree yang tiak memiliki daun.

    Contoh: Node E, F, C, G, H

13. Degreee (Derajat)

    Banyaknya child yang dimiliki oleh suatu node

    Contoh: Node A memiliki derajat 3, node B memiliki derajat 2

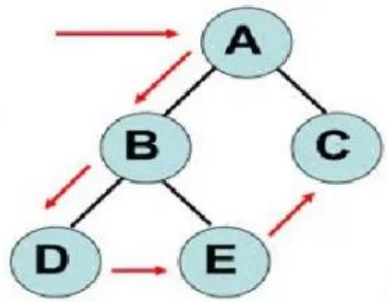
1.2.2 Binary Tree Traversal (Kunjungan Binary Tree)

Proses traversal adalah proses melakukan kunjungan pada setiap node pada suatu binary tree tepat satu kali. Dengan melakukan kunjungan secara lengkap, maka akan didapatkan urutan informasi secara linier yang tersimpan dalam sebuah binary tree.

Terdapat tiga teknik rekursif untuk binary tree traversal, yaitu:

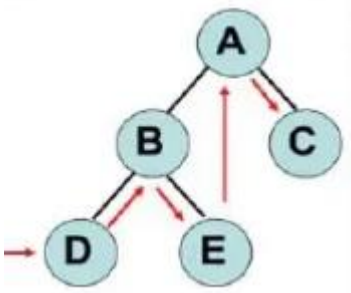
1. Kunjungan secara preorder (Deep First Order)
   a. Cetak isi simpul yang dikunjungi (simpul akar),
   b. Kunjungi cabang kiri.
   c. Kunjungi cabang kanan,

   Ditunjukkan melalui gambar di bawah ini:

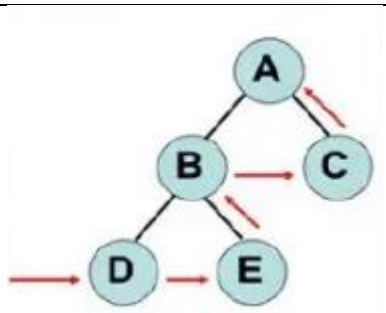
	Urutan: A – B – D – E – C

2. Kunjungan secara inorder (symetric order), mempunyai urutan:
   a. Kunjungi cabang kiri,
   b. Cetak isi simpul yag dikunjungi (simpul akar),
   c. Kunjungi cabang kanan.

   Ditunjukkan melalui gambar di bawah ini:

	Urutan: D – B – E – A – C

3. Kunjungan secara postorder, mempunyai urutan:
   a. Kunjungi cabang kiri,
   b. Kunjungi cabang kanan,
   c. Cetak isi simpul yang dikunjungi (simpul akar).

	Urutan: D – E – B – C – A

1.3 Latihan

1.3.1 Source Code BinaryTree

1. Membuat class TreeNode

```
2   public class TreeNode {
3       int data;
4       TreeNode left;
5       TreeNode right;
6
7       public TreeNode(int data) {
8           this.data = data;
9       }
10  }
```

2. Membuat class BinaryTree

```
public class BinaryTree {
    TreeNode root;

    public boolean isEmpty(){
        return (root==null);
    }

    //method insert data
    public void insert(TreeNode input) {
        if (isEmpty()) {
            root = input;
        } else {
            // cari parent yg sesuai dan (kiri/kanan)
            TreeNode current = root;
            TreeNode parent = null;
            boolean diKiri = true;
                while (current != null) {
                    parent = current;
                    // kalau data yang akan diinputkan lebih besar,
                    // bergerak ke kanan
                    if (current.data < input.data) {
                        current = current.right;
                        diKiri = false;
                    // else gerak ke kiri
                    } else if(current.data > input.data){
                        current = current.left;
                        diKiri = true;
                    }else{
                        System.out.println("data "+input.data+" sudah
ada");
                        break;
                    }
```

```
                }
            // hubungkan ke parent
            if (diKiri) {
                parent.left = input;
            } else {
                parent.right = input;
            }
        }
    }
    public void preOrder(){
        preOrder(root);
    }
    public void inOrder(){
        inOrder(root);
    }
    public void postOrder(){
        postOrder(root);
    }

    public void preOrder(TreeNode akar){
if(akar != null){
            System.out.print(akar.data+" ");
            preOrder(akar.left);
            preOrder(akar.right);
}
    }
    public void inOrder(TreeNode akar){
if(akar != null){
            inOrder(akar.left);
            System.out.print(akar.data+" ");
            inOrder(akar.right);
}
    }

    public void postOrder(TreeNode akar){
if(akar != null){
            postOrder(akar.left);
            postOrder(akar.right);
            System.out.print(akar.data+" ");
}
    }

    //method mencari data
    public TreeNode search(int key) {
        TreeNode node = null;
        TreeNode current = root;
        // lakukan pencarian selama current bukan null
        while (current != null) {
            if (current.data == key) {
                return node;
            } else {
                if (current.data < key) {
                    current = current.right;
                } else {
                    current = current.left;
                }
```

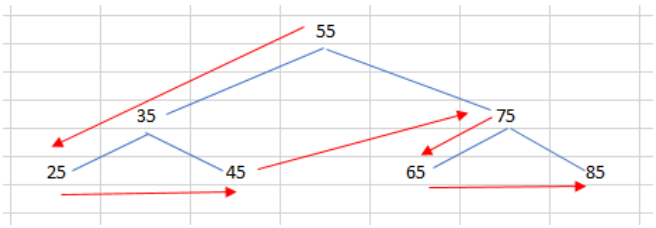
```
            }
        }
        return node;
    }
}
```

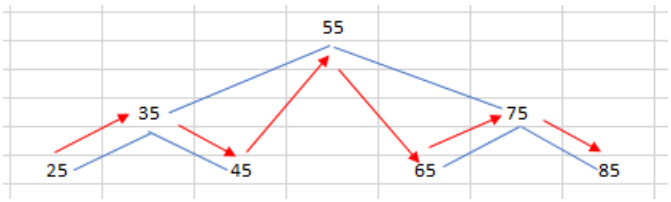
3. Membuat class BinaryTreeApp

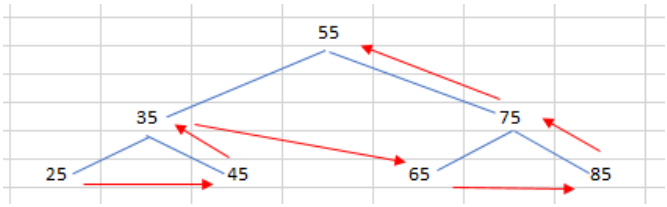
```
3   public class BinaryTreeApp {
4       public static void main(String[] args) {
5           BinaryTree tree = new BinaryTree();
6
7           TreeNode node;
8
9           node = new TreeNode(55);
10          tree.insert(node);
11
12          node = new TreeNode(35);
13          tree.insert(node);
14
15          node = new TreeNode(25);
16          tree.insert(node);
17
18          node = new TreeNode(45);
19          tree.insert(node);
20
21          node = new TreeNode(75);
22          tree.insert(node);
23
24          node = new TreeNode(65);
25          tree.insert(node);
26
27          node = new TreeNode(85);
28          tree.insert(node);
29
30          System.out.print("Traversal dengan preorder :");
31          tree.preOrder();
32          System.out.print("\nTraversal dengan inorder :");
33          tree.inOrder();
34          System.out.print("\nTraversal dengan postorder :");
35          tree.postOrder();
36          System.out.println();
37      }
38  }
39
```

1.3.2 Ilustrasi

Ilustrasi dari stuktur data binary tree yang sesuai dengan sub bab 1.3.2 adalah sebagai berikut:

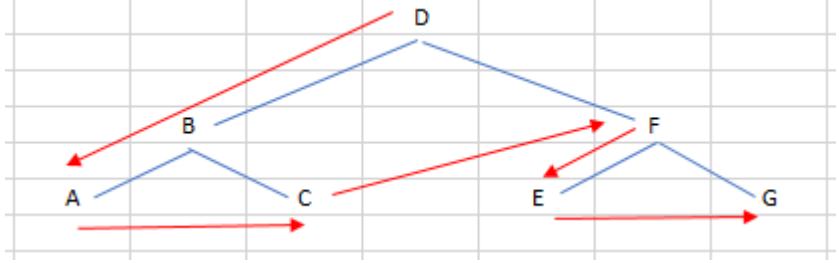
Preorder 	Urutan: 55 – 35 – 25 – 45 – 75 – 65 – 85

Indorder 	Urutan: 25 – 35 – 45 – 55 – 65 – 75 – 85

Postorder 	Urutan: 25 – 45 – 35 – 65 – 85 – 75 – 55

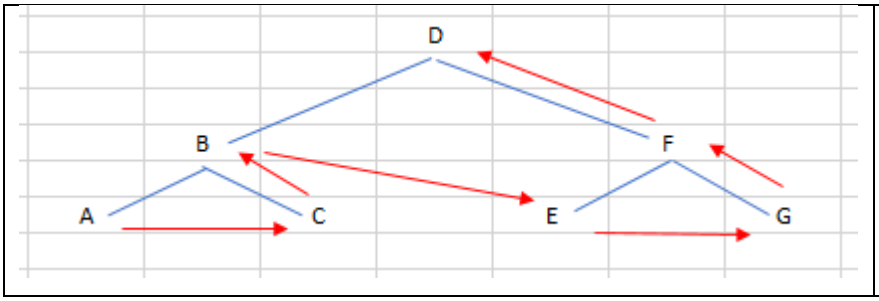
1.4 Tugas Praktikum 8

1. Buatlah **kode program** secara **berturut-turut** dari ilustrasi di bawah ini:



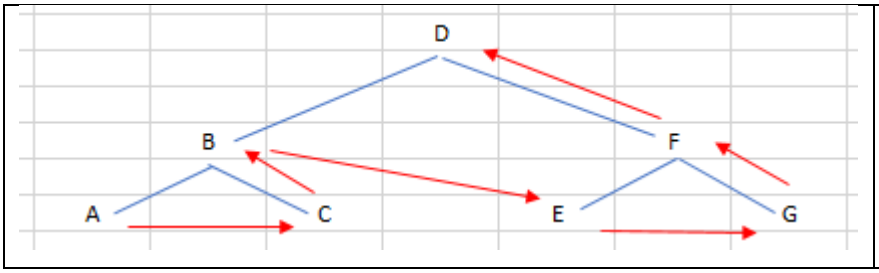
Urutan:

D – B – A – C – F – E – G



Urutan:

A – C – B – E – G – F – D



Urutan:

A – B – C – D – E – F – G

2. Sebutkan contoh nama-nama elemen berdasarkan ilustras pada nomor 1

a Predesesor	e Parent
b Succesor	f Child
c Ancestor	g Sibling
d Descendant	h Subtree