# Penjelasan Jurnal 13

Pekan ke-13

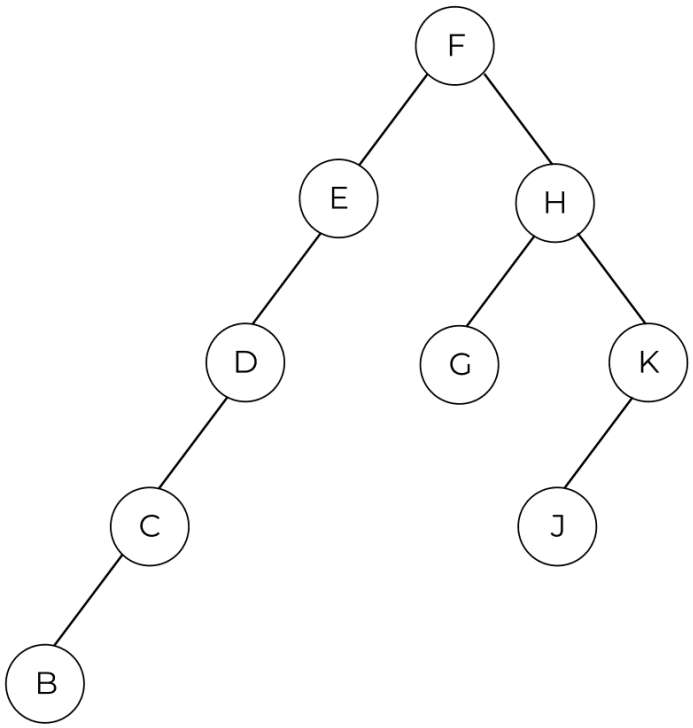

Dibuat oleh :

Muh Daffah Putra Muharsyah 607062330131

## Implementasi Struktur Data

D3 Rekayasa Perangkat Lunak Aplikasi
Fakultas Ilmu Terapan
Tahun 2024

**Binary Search Tree yang terbentuk dari**
F, E, H, D, G, C, B, H, K, J




F dimasukkan pertama otomatis menjadi root dari Tree

Kemudian E dimasukkan, karena E < F maka E dimasukkan disebelah kiri dari F

Kemudian H dimasukkan, karena H > F maka H dimasukkan disebelah kanan dari F

Kemudian D dimasukkan, karena D < F maka E dimasukkan disebelah kiri dari F, namun karena kiri F diisi oleh E maka akan dilakukan perbandingan E dan D, kemudian karena D < E, maka D dimasukkan disebelah kiri E

Kemudian G dimasukkan, karena G > F maka G dimasukkan disebelah kanan dari F, namun karena kanan F diisi oleh H maka akan dilakukan perbandingan H dan G, kemudian karena G < H, maka G dimasukkan disebelah kiri H

Kemudian C dimasukkan, karena C < F, maka C dimasukkan disebelah kiri dari F, namun karena kiri F diisi oleh E, maka akan dibandingkan E dan C, kemudian karena C < E, maka C dimasukkan disebelah kiri E, namun karena kiri E diisi oleh D, maka akan dibandingkan D dan C, kemudian karena C < D, maka C dimasukkan disebelah kiri D

Kemudian B dimasukkan, karena B < F, maka B dimasukkan disebelah kiri dari F, namun karena kiri F diisi oleh E, maka akan dibandingkan E dan B, kemudian karena B < E, maka B dimasukkan disebelah kiri E, namun karena kiri E diisi oleh D, maka akan dibandingkan D dan B, kemudian karena B < D, maka

B dimasukkan disebelah kiri D namun karena kiri D diisi oleh C, maka akan dibandingkan B dan C, kemudian karena B < C, maka B dimasukkan disebelah kiri C

Kemudian H dimasukkan, karena H > F maka H dimasukkan disebelah kanan dari F, namun karena sudah ada H sebelumnya maka H yang kedua tidak dimasukkan dalam Tree

Kemudian K dimasukkan, karena K > F maka K dimasukkan disebelah kanan dari F, namun karena kanan F diisi oleh H, maka dibandingkan H dan K, karena K > H maka K dimasukkan dikanan dari H.

Kemudian J dimasukkan, karena J > F, maka J dimasukkan disebelah kanan dari F, namun karena kanan F diisi oleh H, maka dibandingkan H dan J, karena J > H maka J dimasukkan dikanan dari H, namun karena kanan H diisi oleh K maka akan dibandingkan K dan J, karena J < K maka J dimasukkan disebelah kiri dari K

Kemudian instruksi dari jurnal untuk **Mencari K dan A** menghasilkan output sebagai berikut:

```
Searching for K
Data ditemukan
Searching for A
Data tidak ditemukan
```

Dengan algoritma membandingkan antara data yang dicari dengan node sekarang berada, saat data yang dicari bukan node sekarang maka akan dilakukan perbandingan jika data yang dicari < dari node sekarang akan memeriksa node kiri dari node sekarang lalu jika masih bukan akan melakukan perulangan dengan kondisi yang sudah dibuat sebelumnya jika data yang dicari > dari node sekarang maka akan memerika node kanan dari node sekarang.

```java
public boolean searchBSTHelper(TreeNode<E> node, E key) {
    boolean result = false;
    if (node != null) {
        if (key.equals(node.getData()))
            result = true;
        else if (key.compareTo(node.getData()) < 0)
            result = searchBSTHelper(node.getLeftNode(), key);
        else
            result = searchBSTHelper(node.getRightNode(), key);
    }
    return result;
}
```

Selanjutnya output akan diproses oleh method yang akan mencetak hasil dari pencarian data yang telah dilakukan jika perbandingan data menghasilkan true maka akan mencetak data ditemukan jika tidak maka akan mencetak data tidak ditemukan

```java
public void searchBST(E key) {
    boolean hasil = searchBSTHelper(root, key);
    if (hasil)
        System.out.println(x:"Data ditemukan");
    else
        System.out.println(x:"Data tidak ditemukan");
}
```

**Penjelasan Main Java**

Pada awal baris kita membuat tree bertipe data string terlebih dahulu

```
Tree<String> tree = new Tree<>();
```

Kemudian kita memasukkan data – data sesuai yang ada di Jurnal dengan memanggil method yang sudah kita buat sebelumnya yaitu “insertNode”. Data yang akan dimasukkan adalah sebagai berikut :

F, E, H, D, G, C, B, H, K, J

Kemudian kita memasukkannya kedalam tree

```
tree.insertNode(insertValue:"F");
tree.insertNode(insertValue:"E");
tree.insertNode(insertValue:"H");
tree.insertNode(insertValue:"D");
tree.insertNode(insertValue:"G");
tree.insertNode(insertValue:"C");
tree.insertNode(insertValue:"B");
tree.insertNode(insertValue:"H");
tree.insertNode(insertValue:"K");
tree.insertNode(insertValue:"J");
```

Algoritma dari method insertNode ini adalah node pertama masuk akan menjadi root kemudian data selanjutnya akan dibandingkan dengan root jika lebih kecil dari root maka akan masuk disebelah kiri root namun jika lebih besar maka akan masuk disebelah kanan dari root. Adapun kondisi jika sebelah kiri/kanan root telah terisi oleh data lain maka akan melakukan perulangan untuk membandingkan data yang akan masuk dengan data yang telah terisi.

```
// insert a new node in the binary search tree
public void insertNode(E insertValue) {
    if (root == null) {
        root = new TreeNode<E>(insertValue); // create root node
    } else {
        root.insert(insertValue); // call the insert method
    }
}
```

```
// locate insertion point and insert new node; ignore duplicate values
public void insert(E insertValue) {
    // insert in left subtree
    if (insertValue.compareTo(data) < 0) {
        // insert new TreeNode
        if (leftNode == null) {
            leftNode = new TreeNode<E>(insertValue);
        } else { // continue traversing left subtree recursively
            leftNode.insert(insertValue);
        }
    }
    // insert in right subtree
    else if (insertValue.compareTo(data) > 0) {
        // insert new TreeNode
        if (rightNode == null) {
            rightNode = new TreeNode<E>(insertValue);
        } else { // continue traversing right subtree recursively
            rightNode.insert(insertValue);
        }
    }
}
```

Selanjutnya mencetak **Preorder Traversal**

```
System.out.println(x:"Preorder traversal");
tree.preorderTraversal();
System.out.println();
```

Algoritma dari Preorder Traversal adalah mencetak terlebih dahulu kemudian pergi ke Node Sebelah kiri lalu sebelah kanan, yang mana jika kita terapkan dalam sebuah codingan akan sebagai berikut

```
// begin preorder traversal
public void preorderTraversal() {
    preorderHelper(root);
}

// recursive method to perform preorder traversal
private void preorderHelper(TreeNode<E> node) {
    if (node == null) {
        return;
    }
    System.out.printf(format:"%s ", node.getData()); // output node data
    preorderHelper(node.getLeftNode()); // traverse left subtree
    preorderHelper(node.getRightNode()); // traverse right subtree
}
```

Nah codingan diatas akan menghasilkan data hasil travers menggunakan preorder sebagai berikut

```
Preorder traversal
F E D C B H G K J
```

Masuk mencetak **Inorder Traversal**

```
System.out.println(x:"Inorder traversal");
tree.inorderTraversal();
System.out.println();
```

Algoritma dari Inorder Traversal adalah pergi ke Node sebelah kiri terlebih dahulu sampai ujung kemudian mencetak, setelah itu baru pergi ke Node sebelah kanan yang mana jika kita terapkan dalam sebuah codingan akan sebagai berikut

```
// begin inorder traversal
public void inorderTraversal() {
    inorderHelper(root);
}

// recursive method to perform inorder traversal
private void inorderHelper(TreeNode<E> node) {
    if (node == null) {
        return;
    }
    inorderHelper(node.getLeftNode()); // traverse left subtree
    System.out.printf(format:"%s ", node.getData()); // output node data
    inorderHelper(node.getRightNode()); // traverse right subtree
}
```

Nah codingan diatas akan menghasilkan hasil data travers sebagai berikut

```
Inorder traversal
B C D E F G H J K
```

Terakhir dalam program ini untuk mencetak **Postorder Traversal**

```
System.out.println(x:"Postorder traversal");
tree.postorderTraversal();
System.out.println();
System.out.println();
```

Algoritma dari Postorder Traversal adalah pergi ke sebelah kiri terlebih dahulu sampai ujung kemudian kesebelah kanan jika tidak ada lagi node baru mencetak, yang jika kita terapkan dalam sebuah codingan maka akan menghasilkan sebagai berikut

```
// begin postorder traversal
public void postorderTraversal() {
    postorderHelper(root);
}

// recursive method to perform postorder traversal
private void postorderHelper(TreeNode<E> node) {
    if (node == null) {
        return;
    }
    postorderHelper(node.getLeftNode()); // traverse left subtree
    postorderHelper(node.getRightNode()); // traverse right subtree
    System.out.printf(format:"%s ", node.getData()); // output node data
}
```

Nah codingan diatas akan menghasilkan data hasil travers menggunakan preorder sebagai berikut

```
Postorder traversal
B C D E G J K H F
```