# LAPORAN PRAKTIKUM

**Modul 9**

**Tree**


**Disusun Oleh :**

**Fauzan Rofif Ardiyanto/2211104036**

**SE 07 2**

**Asisten Praktikum :**

**Aldi Putra**

**Andini Nur Hidayah**

**Dosen Pengampu :**

**Wahyu Andi Saputra**

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**


**2024**

## 1. Tujuan

a. Mahasiswa mampu menjelaskan definisi dan konsep dari tree.
b. Mahasiswa mampu menerapkan operasi tambah, menghapus, pada tree.
c. Mahasiswa mampu menerapkan operasi tampil data pada tree.

## 2. Landasan Teori

a. Tree

Struktur data pohon (tree) adalah suatu struktur hierarkis yang terdiri dari simpul-simpul (nodes) yang terhubung oleh sisi (edges). Sebuah pohon memiliki struktur yang mirip dengan pohon pada dunia nyata, di mana terdapat satu akar (root) dan cabang-cabang yang berkembang dari akar tersebut. Setiap simpul pada pohon dapat memiliki anak (child) yang lebih banyak, dan simpul-simpul tersebut dapat membentuk lebih banyak cabang.

**Elemen utama Tree, antara lain:**

- Akar (Root): Simpul pertama pada pohon yang menjadi titik awal dari struktur pohon. Akar hanya memiliki satu jalur untuk mengakses seluruh pohon.
- Simpul (Node): Setiap elemen dalam pohon, yang berisi data dan dapat memiliki satu atau lebih anak.
- Anak (Child): Simpul yang terhubung langsung dengan simpul lainnya yang berada di atasnya (parent).
- Induk (Parent): Simpul yang menghubungkan dengan satu atau lebih simpul di bawahnya (children).
- Daun (Leaf): Simpul yang tidak memiliki anak.
- Level: Menunjukkan kedalaman simpul dalam pohon, dimulai dari akar.
- Kedalaman (Depth): Jarak antara simpul tertentu dengan akar pohon.
- Tinggi (Height): Panjang jalur terpanjang dari akar ke daun terjauh dalam pohon.

**Jenis jenis Tree, di antaranya:**

- Pohon Biner (Binary Tree): Pohon di mana setiap simpul memiliki paling banyak dua anak (kiri dan kanan).

- Pohon Biner Pencarian (Binary Search Tree): Pohon biner yang disusun sedemikian rupa sehingga nilai simpul anak kiri lebih kecil daripada nilai simpul induk, sedangkan nilai simpul anak kanan lebih besar daripada induknya.
- Heap: Pohon biner khusus yang memenuhi sifat heap, di mana setiap simpul induk lebih besar (untuk heap maksimal) atau lebih kecil (untuk heap minimal) daripada anak-anaknya.
- Pohon AVL: Pohon biner pencarian yang memiliki aturan keseimbangan tertentu, yaitu selisih tinggi antara subpohon kiri dan subpohon kanan setiap simpul tidak lebih dari satu.
- Pohon Trie: Digunakan dalam pencarian string, dengan setiap simpul mewakili karakter dalam string.

**Operasi dasar pada Tree, antara lain:**

- Traversal: Proses mengunjungi setiap simpul dalam pohon. Traversal pada pohon dapat dilakukan dengan tiga cara utama:
- Pre-order: Mengunjungi akar terlebih dahulu, kemudian anak kiri, dan anak kanan.
- In-order: Mengunjungi anak kiri, kemudian akar, dan anak kanan. Pada pohon biner pencarian, traversal in-order menghasilkan urutan data yang terurut.
- Post-order: Mengunjungi anak kiri, anak kanan, dan terakhir akar.
- Level-order: Mengunjungi simpul-simpul pada setiap level secara berurutan.
- Pencarian (Search): Mencari elemen tertentu dalam pohon dengan cara traversal atau menggunakan algoritma khusus pada pohon biner pencarian.
- Penyisipan (Insertion): Menambahkan simpul baru ke dalam pohon.
- Penghapusan (Deletion): Menghapus simpul dari pohon, dengan memperhatikan struktur pohon setelah penghapusan.

## 3. Guided

### a. Guided 1

Source Code

```
#include <iostream>
using namespace std;

struct Pohon {
    char data;
    Pohon *left, *right;
    Pohon *parent;
};

Pohon *root, *baru;

void init() {
    root = NULL;
}
bool isEmpty(){
    return root == NULL;
}

void buatNode(char data) {
    if (isEmpty()){
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl
;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

Pohon* insertLeft(char data, Pohon *node) {
    if (node-> left != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kiri! " << endl
;
        return NULL;
    }

    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data <<endl;
    return baru;
}

Pohon* insertRight(char data, Pohon *node){
    if (node->right != NULL){
        cout << "\nNode " << node->data << " sudah ada child kanan " << endl;
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
```

```
    cout << "\nNode " << data << " berhasil ditambahkan ke cild kanan " <<
node->data <<endl;
    return baru;
}

void find(char data, Pohon *node){
    if(!node) return;

    if(node->data == data){
        cout << "\nNode ditemukan." << data << endl;
        return;
    }
    find(data, node->left);
    find(data, node->right);
}

void preOrder(Pohon *node){
    if (!node) return;
    cout << node->data << " " ;
    preOrder(node->left);
    preOrder(node->right);
}

void inOrder(Pohon *node){
    if(!node) return;
    inOrder(node->left);
    cout << node->data << " " ;
    inOrder(node->right);
}

void postOrder(Pohon *node){
    if (!node) return;
    postOrder(node->left);
    postOrder(node->right);
    cout << node->data << " " ;
}

Pohon* deleteNode(Pohon *node, char data){
    if (!node) return NULL;

    if (data < node ->data) {
        node->left = deleteNode(node->left, data);
    } else if (data > node->data) {
        node->right = deleteNode(node->right, data);
    } else {
        if (!node->left) {
            Pohon *temp = node->right;
            delete node;
            return temp;
        } else if (!node->right) {
            Pohon *temp = node->left;
            delete node;
            return temp;
```

```
        }

        Pohon *successor = node->right;
        while (successor->left) successor = successor->left;
        node->data = successor->data;
        node->right = deleteNode(node->right, successor->data);
    }
    return node;
}

Pohon* mostLeft(Pohon *node){
    if(!node) return NULL;
    while (node->left) node = node->left;
    return node;
}

Pohon* mostRight(Pohon *node) {
    if (!node) return NULL;
    while (node->right) node = node->right;
    return node;
}

int main(){
    init();
    buatNode('F');
    insertLeft('B', root);
    insertRight('G', root);
    insertLeft('A', root)->left;
    insertRight('D', root->left);
    insertLeft('C', root->left->right);
    insertRight('E', root->left->right);

    cout << "\nPre-Order Transversal: ";
    preOrder(root);
    cout << "\nIn-Order Tranversal: ";
    inOrder(root);
    cout << "\nPost-order Transversal: ";
    postOrder(root);

    cout << "\nMost Left Node: " << mostLeft(root)->data;
    cout << "\nMost Right Node: " << mostRight(root)->data;

    cout << "\nMenghapus node D.";
    root = deleteNode(root,'D');
    cout << "\nIn-Order Transversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}
```

Output

```
Node F berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri F

Node G berhasil ditambahkan ke cild kanan F

Node F sudah ada child kiri!

Node D berhasil ditambahkan ke cild kanan B

Node C berhasil ditambahkan ke child kiri D

Node E berhasil ditambahkan ke cild kanan D

Pre-Order Transversal: F B D C E G
In-Order Tranversal: B C D E F G
Post-order Transversal: C E D B G F
Most Left Node: B
Most Right Node: G
Menghapus node D.
In-Order Transversal setelah penghapusan: B C E F G
```

Deskripsi

Program di atas mengimplementasikan sebuah pohon biner dengan operasi dasar seperti pembuatan node, penambahan anak kiri dan kanan, pencarian node, serta traversal pohon menggunakan metode pre-order, in-order, dan post-order. Pohon ini juga mendukung penghapusan node dengan menggunakan algoritma penghapusan pada pohon biner pencarian (BST). Dalam program ini, setiap node menyimpan data berupa karakter dan memiliki pointer ke anak kiri, anak kanan, dan induknya. Fungsi mostLeft dan mostRight digunakan untuk mencari node paling kiri dan paling kanan dalam

pohon, sedangkan fungsi deleteNode digunakan untuk menghapus node tertentu dan mempertahankan struktur pohon. Program ini kemudian menampilkan traversal pohon, menemukan node paling kiri dan kanan, serta menghapus node tertentu dan menampilkan hasil traversal setelah penghapusan.

## 4. Unguided

a. Soal 1

Source Code

```
#include <iostream>
using namespace std;

struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};

// Root pohon
Pohon *root = NULL, *baru;

// Fungsi untuk menginisialisasi pohon
void init() {
    root = NULL;
}

// Fungsi untuk memeriksa apakah pohon kosong
bool isEmpty() {
    return root == NULL;
}

// Fungsi untuk membuat node root
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

// Fungsi untuk menambahkan anak kiri
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    }

    baru = new Pohon{data, NULL, NULL, node};
```

```
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data << endl;
    return baru;
}

// Fungsi untuk menambahkan anak kanan
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }

    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " <<
node->data << endl;
    return baru;
}

// Fungsi untuk menampilkan anak dari node
void displayChildren(Pohon *node) {
    if (node == NULL) {
        cout << "Node tidak ditemukan." << endl;
        return;
    }

    cout << "Children dari node " << node->data << ": ";

    // Cek jika anak kiri ada
    if (node->left != NULL) {
        cout << node->left->data << " ";
    }

    // Cek jika anak kanan ada
    if (node->right != NULL) {
        cout << node->right->data << " ";
    }

    // Jika tidak ada anak kiri dan kanan
    if (node->left == NULL && node->right == NULL) {
        cout << "Tidak ada anak." << endl;
    } else {
        cout << endl;
    }
}

// Fungsi untuk menampilkan keturunan dari node
void displayDescendants(Pohon *node) {
    if (node == NULL) {
        cout << "Node tidak ditemukan." << endl;
        return;
    }
```

```
    cout << "Descendants dari node " << node->data << ": ";
    if (node->left != NULL) {
        cout << node->left->data << " ";
        displayDescendants(node->left);
    }
    if (node->right != NULL) {
        cout << node->right->data << " ";
        displayDescendants(node->right);
    }

    cout << endl;
}

// Fungsi rekursif untuk mencari node
Pohon* findNode(char data, Pohon* node) {
    if (node == NULL) return NULL;
    if (node->data == data) return node;
    Pohon* leftSearch = findNode(data, node->left);
    if (leftSearch) return leftSearch;
    return findNode(data, node->right);
}

// Fungsi untuk memeriksa apakah pohon adalah BST
bool is_valid_bst(Pohon* node, char min_val, char max_val) {
    if (node == NULL) return true;

    if (node->data < min_val || node->data > max_val)
        return false;

    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}

// Fungsi untuk menghitung jumlah simpul daun
int countLeafNodes(Pohon* node) {
    if (node == NULL) return 0;
    if (node->left == NULL && node->right == NULL) return 1;
    return countLeafNodes(node->left) + countLeafNodes(node->right);
}

int main() {
    init();

    // Deklarasi variabel yang dipakai untuk menyimpan node
    Pohon* parentLeft;
    Pohon* parentRight;
    Pohon* node;
    Pohon* nodeDescendants;
    Pohon* foundNode;

    // Menu untuk memasukkan data pohon dan memilih operasi
    int choice;
```

```
    char data, parentData;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Buat Root\n";
        cout << "2. Tambah Anak Kiri\n";
        cout << "3. Tambah Anak Kanan\n";
        cout << "4. Tampilkan Anak dari Node\n";
        cout << "5. Tampilkan Keturunan dari Node\n";
        cout << "6. Cari Node\n";
        cout << "7. Cek apakah Pohon Valid sebagai BST\n";
        cout << "8. Hitung Jumlah Simpul Daun\n";
        cout << "9. Keluar\n";
        cout << "Masukkan pilihan: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data root: ";
                cin >> data;
                buatNode(data);
                break;

            case 2:
                cout << "Masukkan data node yang akan diberi anak kiri: ";
                cin >> parentData;
                cout << "Masukkan data anak kiri: ";
                cin >> data;
                parentLeft = findNode(parentData, root);
                if (parentLeft != NULL) {
                    insertLeft(data, parentLeft);
                } else {
                    cout << "Node " << parentData << " tidak ditemukan.\n";
                }
                break;

            case 3:
                cout << "Masukkan data node yang akan diberi anak kanan: ";
                cin >> parentData;
                cout << "Masukkan data anak kanan: ";
                cin >> data;
                parentRight = findNode(parentData, root);
                if (parentRight != NULL) {
                    insertRight(data, parentRight);
                } else {
                    cout << "Node " << parentData << " tidak ditemukan.\n";
                }
                break;

            case 4:
                cout << "Masukkan data node untuk melihat anak-anaknya: ";
                cin >> data;
                node = findNode(data, root);
```

```
                displayChildren(node);
                break;

            case 5:
                cout << "Masukkan data node untuk melihat keturunannya: ";
                cin >> data;
                nodeDescendants = findNode(data, root);
                displayDescendants(nodeDescendants);
                break;

            case 6:
                cout << "Masukkan data node yang akan dicari: ";
                cin >> data;
                foundNode = findNode(data, root);
                if (foundNode != NULL) {
                    cout << "Node " << data << " ditemukan.\n";
                } else {
                    cout << "Node " << data << " tidak ditemukan.\n";
                }
                break;

            case 7:
                if (is_valid_bst(root, 'A', 'Z')) {
                    cout << "Pohon ini adalah Binary Search Tree.\n";
                } else {
                    cout << "Pohon ini bukan Binary Search Tree.\n";
                }
                break;

            case 8:
                cout << "Jumlah simpul daun: " << countLeafNodes(root) <<
endl;
                break;

            case 9:
                cout << "Keluar dari program.\n";
                return 0;

            default:
                cout << "Pilihan tidak valid.\n";
        }
    }

    return 0;
}
```

Output

- Membuat Pohon (Root)

```
Menu:
1. Buat Pohon
2. Masukkan Anak Kiri
3. Masukkan Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Periksa apakah pohon adalah Binary Search Tree (BST)
7. Hitung jumlah simpul daun
8. Pre-order Traversal
9. In-order Traversal
10. Post-order Traversal
11. Keluar
Pilihan: 4
Masukkan data node untuk tampilkan anaknya: A
Children dari node A :
```

- Menambahan Children

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 2
Masukkan data node yang akan diberi anak kiri: A
Masukkan data anak kiri: B

Node B berhasil ditambahkan ke child kiri A
```

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 3
Masukkan data node yang akan diberi anak kanan: A
Masukkan data anak kanan: C

Node C berhasil ditambahkan ke child kanan A
```

- Menampilkan Node

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 4
Masukkan data node untuk melihat anak-anaknya: A
Children dari node A: B C
```

- Menampilkan Keturunan

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 4
Masukkan data node untuk melihat anak-anaknya: B
Children dari node B: D E

Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 5
Masukkan data node untuk melihat keturunannya: B
Descendants dari node B: D Descendants dari node D:
E Descendants dari node E:
```

- Cari Node

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 6
Masukkan data node yang akan dicari: A
Node A ditemukan.
```

- Cek Jenis Pohon

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 7
Pohon ini bukan Binary Search Tree.
```

- Hitung Jumlah Leaf

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 8
Jumlah simpul daun: 3
```

Deskripsi

Program ini merupakan implementasi pohon biner yang memungkinkan pengguna untuk membangun dan mengelola pohon biner dengan berbagai operasi. Pengguna dapat membuat root pohon, menambahkan anak kiri dan kanan pada node yang ada, serta menampilkan anak atau keturunan dari suatu node. Selain itu, program juga menyediakan fitur untuk mencari node tertentu, memeriksa apakah pohon tersebut valid sebagai Binary Search Tree (BST), dan menghitung jumlah simpul daun dalam pohon. Program menggunakan menu interaktif di mana pengguna dapat memilih berbagai operasi melalui input, dan mendukung pencarian node, traversal, serta validasi struktur pohon.