

**LAPORAN PRAKTIKUM  
STRUKTUR DATA  
Modul 9  
“TREE”**



**Disusun Oleh:  
MUHAMMAD RALFI - 2211104054  
SE-07-2**

**Dosen :  
Wahyu Andi Saputra S.Pd, M.Eng**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY  
PURWOKERTO  
2024**

## 1. Tujuan

- Mahasiswa dapat memahami konsep Tree
- Mahasiswa mampu mendefinisikann tentang Tree pada pemrograman
- Mahasiswa dapat mengimplementasikan konsep Tree pada pemrograman

## 2. Landasan Teori

### Tree

Tree merupakan salah satu struktur data non-linear yang digunakan untuk merepresentasikan hubungan hierarkis antara elemen-elemen data. Struktur ini terdiri dari simpul (nodes) yang saling terhubung melalui hubungan parent dan child. Sebuah tree memiliki satu simpul utama yang disebut root, yang berfungsi sebagai titik awal. Setiap simpul dapat memiliki sejumlah child nodes, tetapi hanya memiliki satu parent node, kecuali root yang tidak memiliki parent. Struktur ini bersifat rekursif, di mana setiap sub-tree dari sebuah tree juga berbentuk tree.

Tree memiliki berbagai jenis, seperti binary tree, binary search tree (BST), AVL tree, dan B-tree, yang masing-masing memiliki karakteristik dan tujuan penggunaan yang berbeda. Pada binary tree, setiap simpul hanya dapat memiliki maksimal dua child nodes, sedangkan pada binary search tree, elemen-elemen diatur sedemikian rupa sehingga simpul di sebelah kiri memiliki nilai lebih kecil daripada simpul induknya, dan simpul di sebelah kanan memiliki nilai lebih besar

## 3. Guided

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

struct pohon {
    char data;
    pohon *kiri;
    pohon *kanan;
    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 << "Node " << data << " berhasil dibuat sebagai root" << endl;
    } else {
        cout << "Pohon sudah dibuat" << endl;
    }
}

pohon* insertKiri(char data, pohon *node) {
```

```
        if(node->kiri != NULL){
            cout << "\nNode " << data << " sudah memiliki anak kiri" << node-
>data << endl;
        }
        baru = new pohon{data, NULL, NULL, node};
        node->kiri = baru;
        cout << "\nNode " << data << "berhasil ditambahkan" << endl;
        return baru;
    }

    pohon *insertKanan(char data, pohon *node) {
        if(node->kanan != NULL){
            cout << "\nNode " << data << " sudah memiliki anak kanan" <<
node->data << endl;
        }
        baru = new pohon{data, NULL, NULL, node};
        node->kanan = baru;
        cout << "\nNode " << data << "berhasil ditambahkan" << endl;
        return baru;
    }

    void updateNode(char data, pohon *node) {
        if(!node) {
            cout << "\nNode " << data << " tidak ditemukan" << endl;
            return;
        }
        char temp = node->data;
        node->data = data;
        cout << "\nNode " << temp << " berhasil diupdate menjadi " << data <<
endl;
    }

    void find(char data, pohon *node) {
        if(!node)return;
        if(node->data == data) {
            cout << "\nNode " << data << " ditemukan" << endl;
            return;
        }
        find(data, node->kiri);
        find(data, node->kanan);
    }

    void preOrder(pohon *node) {
        if (!node) return;
        cout << node->data << " ";
        preOrder(node->kiri);
        preOrder(node->kanan);
    }

    void inOrder(pohon *node) {
        if (!node) return;
        inOrder(node->kiri);
        cout << node->data << " ";
        inOrder(node->kanan);
    }

    void postOrder(pohon *node) {
        if (!node) return;
```

```
        postOrder(node->kiri);
        postOrder(node->kanan);
        cout << node->data << " ";
    }

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

        if (data < node->data){
            node->kiri = deleteNode(node->kiri, data);
        } else if (data > node->data){
            node->kanan = deleteNode(node->kanan, data);
        } else {
            if (!node->kiri) {
                pohon *temp = node->kanan;
                delete node;
                return temp;
            } else if (!node->kanan) {
                pohon *temp = node->kiri;
                delete node;
                return temp;
            }

            pohon *successor = node->kanan;
            while (successor->kiri) successor = successor->kiri;
            node->data = successor->data;
            node->kanan = deleteNode(node->kanan, successor->data);
        }
        return node;
    }

    pohon* mostKiri(pohon *node) {
        if(!node) return NULL;
        while(node->kiri) node = node->kiri;
        return node;
    }

    pohon* mostKanan(pohon *node) {
        if(!node) return NULL;
        while(node->kanan) node = node->kanan;
        return node;
    }

    int main() {
        init();
        buatNode('F');
        insertKiri('B', root);
        insertKanan('G', root);
        insertKiri('A', root ->kiri);
        insertKanan('D', root ->kiri);
        insertKiri('C', root -> kiri ->kanan);
        insertKanan('E', root->kiri->kanan);

        cout << "\nPre-order Traversal: ";
        preOrder(root);
        cout << "\nPost-order Traversal: ";
        postOrder(root);
    }
```

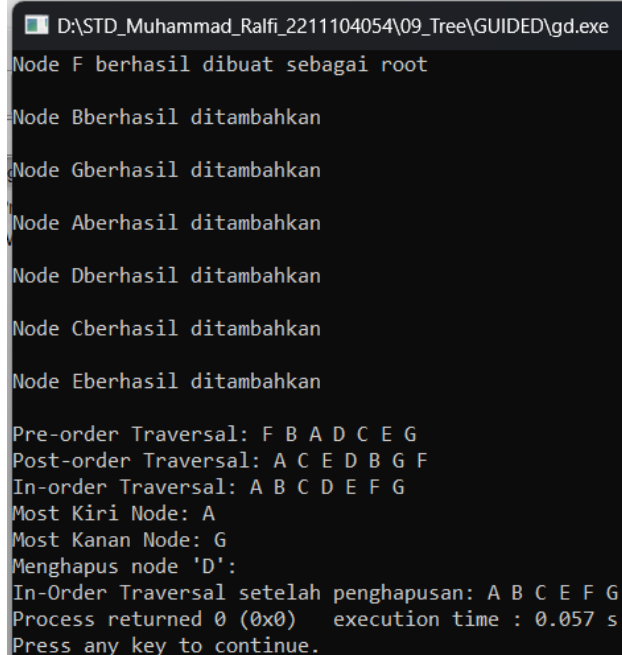
```
    cout << "\nIn-order Traversal: ";
    inOrder(root);

    cout << "\nMost Kiri Node: " << mostKiri(root)->data;
    cout << "\nMost Kanan Node: " << mostKanan(root)->data;

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

    return 0;
}
```

Output:



```
D:\STD_Muhammad_Ralfi_2211104054\09_Tree\GUIDED\gd.exe
Node F berhasil dibuat sebagai root
Node B berhasil ditambahkan
Node G berhasil ditambahkan
Node A berhasil ditambahkan
Node D berhasil ditambahkan
Node C berhasil ditambahkan
Node E berhasil ditambahkan
Pre-order Traversal: F B A D C E G
Post-order Traversal: A C E D B G F
In-order Traversal: A B C D E F G
Most Kiri Node: A
Most Kanan Node: G
Menghapus node 'D':
In-Order Traversal setelah penghapusan: A B C E F G
Process returned 0 (0x0)   execution time : 0.057 s
Press any key to continue.
```

#### 4. Unguided

File kode

```
#include <iostream>
#include <queue>
#include <climits>
using namespace std;
struct pohon {
    int data; // Ganti char menjadi int
    pohon *kiri;
    pohon *kanan;
    pohon *parent;
};
pohon *root, *baru;
void init() {
    root = NULL;
}
bool isEmpty() {
    return root == NULL;
}
```

```
}
void buatNode(int data) {
    if (isEmpty()) {
        root = new pohon{data, NULL, NULL, NULL};
        cout << "Node " << data << " berhasil dibuat sebagai root" << endl;
    } else {
        cout << "Pohon sudah dibuat" << endl;
    }
}

pohon* insertKiri(int data, pohon *node) {
    if (node->kiri != NULL) {
        cout << "\nNode " << node->data << " sudah memiliki anak kiri" << endl;
        return NULL;
    }
    baru = new pohon{data, NULL, NULL, node};
    node->kiri = baru;
    cout << "\nNode " << data << " berhasil ditambahkan sebagai anak kiri dari " << node->data << endl;
    return baru;
}

pohon* insertKanan(int data, pohon *node) {
    if (node->kanan != NULL) {
        cout << "\nNode " << node->data << " sudah memiliki anak kanan" << endl;
        return NULL;
    }
    baru = new pohon{data, NULL, NULL, node};
    node->kanan = baru;
    cout << "\nNode " << data << " berhasil ditambahkan sebagai anak kanan dari " << node->data << endl;
    return baru;
}

void displayTree(pohon* node, int level = 0) {
    if (node == NULL) return;

    displayTree(node->kanan, level + 1);

    for (int i = 0; i < level; i++) cout << "    ";
    cout << node->data << endl;

    displayTree(node->kiri, level + 1);
}

pohon* findNode(int data) {
    if (isEmpty()) return NULL;

    queue<pohon*> q;
```

```
q.push(root);

while (!q.empty()) {
    pohon* temp = q.front();
    q.pop();

    if (temp->data == data) return temp;

    if (temp->kiri) q.push(temp->kiri);
    if (temp->kanan) q.push(temp->kanan);
}

return NULL;
}

// Fungsi rekursif untuk memeriksa properti Binary Search Tree
bool is_valid_bst(pohon* node, int min_val, int max_val) {
    if (!node) return true;

    int nilai = node->data; // Anggap data sudah berupa int
    if (nilai <= min_val || nilai >= max_val) return false;

    return is_valid_bst(node->kiri, min_val, nilai) &&
           is_valid_bst(node->kanan, nilai, max_val);
}

void periksaBST() {
    if (is_valid_bst(root, INT_MIN, INT_MAX)) {
        cout << "Pohon ini adalah Binary Search Tree yang valid.\n";
    } else {
        cout << "Pohon ini BUKAN Binary Search Tree yang valid.\n";
    }
}

// Fungsi rekursif untuk menghitung jumlah simpul daun
int cari_simpul_daun(pohon* node) {
    if (!node) return 0; // Jika node kosong, kembalikan 0

    // Jika node tidak memiliki anak kiri dan kanan, maka itu adalah simpul
    daun
    if (!node->kiri && !node->kanan) return 1;

    // Traversal ke kiri dan kanan
    return cari_simpul_daun(node->kiri) + cari_simpul_daun(node->kanan);
}

void hitungSimpulDaun() {
    int jumlah_daun = cari_simpul_daun(root);
    cout << "Jumlah simpul daun: " << jumlah_daun << endl;
}
```

```
int main() {
    init();
    int pilihan;
    int data, parent_data;

    while (true) {
        cout << "--- Menu Pohon Biner ---" << endl;
        cout << "1. Tambah Root Node" << endl;
        cout << "2. Tambah Child Kiri" << endl;
        cout << "3. Tambah Child Kanan" << endl;
        cout << "4. Tampilkan Struktur Pohon" << endl;
        cout << "5. Periksa apakah pohon adalah BST" << endl;
        cout << "6. Hitung Jumlah Simpul Daun" << endl;
        cout << "7. Keluar" << endl;
        cout << "Masukkan pilihan Anda: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                if (isEmpty()) {
                    cout << "Masukkan data root: ";
                    cin >> data;
                    buatNode(data);
                } else {
                    cout << "Root sudah ada!" << endl;
                }
                break;

            case 2: // Tambah Child Kiri
                cout << "Masukkan data node induk: ";
                cin >> parent_data;
                cout << "Masukkan data anak kiri: ";
                cin >> data;
                if (pohon* parent = findNode(parent_data)) {
                    insertKiri(data, parent);
                } else {
                    cout << "Node induk tidak ditemukan!" << endl;
                }
                break;

            case 3: // Tambah Child Kanan
                cout << "Masukkan data node induk: ";
                cin >> parent_data;
                cout << "Masukkan data anak kanan: ";
                cin >> data;
                if (pohon* parent = findNode(parent_data)) {
                    insertKanan(data, parent);
                } else {
                    cout << "Node induk tidak ditemukan!" << endl;
                }
            }
        }
    }
}
```



```
        }  
        break;  
  
    case 4:  
        cout << "Struktur Pohon:" << endl;  
        displayTree(root);  
        break;  
  
    case 5:  
        periksaBST();  
        break;  
  
    case 6:  
        hitungSimpulDaun();  
        break;  
  
    case 7:  
        cout << "Keluar dari program." << endl;  
        return 0;  
  
    default:  
        cout << "Pilihan tidak valid. Silakan coba lagi." << endl;  
    }  
}  
  
return 0;  
}
```

Output:

a. Menambahkan node

<pre> D:\STD_Muhammad_Ralfi_2211104054\09_Tree\UNGUIDED\ugd.exe --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 1 Masukkan data root: 10 Node 10 berhasil dibuat sebagai root --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 2 Masukkan data node induk: 10 Masukkan data anak kiri: 7  Node 7 berhasil ditambahkan sebagai anak kiri dari 10 --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 2 Masukkan data node induk: 7 Masukkan data anak kiri: 3  Node 3 berhasil ditambahkan sebagai anak kiri dari 7 </pre>	<pre> D:\STD_Muhammad_Ralfi_2211104054\09_Tree\UNGUIDED\ugd.exe --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 3 Masukkan data node induk: 10 Masukkan data anak kanan: 12  Node 12 berhasil ditambahkan sebagai anak kanan dari 10 --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 3 Masukkan data node induk: 12 Masukkan data anak kanan: 16  Node 16 berhasil ditambahkan sebagai anak kanan dari 12 --- Menu Pohon Biner --- 1. Tambah Root Node 2. Tambah Child Kiri 3. Tambah Child Kanan 4. Tampilkan Struktur Pohon 5. Periksa apakah pohon adalah BST 6. Hitung Jumlah Simpul Daun 7. Keluar Masukkan pilihan Anda: 4 Struktur Pohon:     16    12   10    7     3 </pre>
---	--

b. Mengecek BST apakah valid atau tidak

```

Struktur Pohon:
    16
   12
  10
   7
    3
--- Menu Pohon Biner ---
1. Tambah Root Node
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Struktur Pohon
5. Periksa apakah pohon adalah BST
6. Hitung Jumlah Simpul Daun
7. Keluar
Masukkan pilihan Anda: 5
Pohon ini adalah Binary Search Tree yang valid.
--- Menu Pohon Biner ---
1. Tambah Root Node
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Struktur Pohon
5. Periksa apakah pohon adalah BST
6. Hitung Jumlah Simpul Daun
7. Keluar
Masukkan pilihan Anda:

```

c. Mencari simpul daun

```
Pohon ini adalah Binary Search Tree yang valid.
--- Menu Pohon Biner ---
1. Tambah Root Node
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Struktur Pohon
5. Periksa apakah pohon adalah BST
6. Hitung Jumlah Simpul Daun
7. Keluar
Masukkan pilihan Anda: 6
Jumlah simpul daun: 2
--- Menu Pohon Biner ---
1. Tambah Root Node
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Struktur Pohon
5. Periksa apakah pohon adalah BST
6. Hitung Jumlah Simpul Daun
7. Keluar
Masukkan pilihan Anda: _
```

## 5. Kesimpulan

Tree merupakan struktur data yang efisien untuk merepresentasikan data secara hierarkis dan mendukung operasi seperti pencarian, penyisipan, dan penghapusan. Melalui praktikum, dipahami cara kerja binary tree dan binary search tree (BST), termasuk teknik traversal seperti preorder, inorder, dan postorder. Tree juga memiliki aplikasi luas, seperti struktur direktori file, yang menunjukkan fleksibilitas dan manfaatnya dalam pengelolaan data.