LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X
TREE

Disusun Oleh :
NAMA : ABYAN RAHMAN AL FARIZ
NIM : 103112430021

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Tree adalah struktur data yang menyusun informasi dalam bentuk hierarki melalui hubungan antar node yang saling terhubung oleh edge. Struktur ini berawal dari sebuah node awal yang disebut root, kemudian bercabang ke node-node lain yang berperan sebagai child, sementara node yang tidak memiliki cabang lebih lanjut dikenal sebagai leaf.

Sifatnya yang tidak terikat pada urutan linear membuat tree sangat fleksibel dalam merepresentasikan data bertingkat dan kompleks. Karena itu, struktur ini banyak digunakan dalam berbagai bidang, seperti penyusunan folder pada sistem operasi, proses parsing pada compiler, hingga pembuatan model pohon keputusan pada kecerdasan buatan.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1 (tree.cpp)

```cpp
#include "tree.h"
#include <iostream>
using namespace std;

BinaryTree::BinaryTree() {
    root = nullptr;
}

int BinaryTree::getHeight(Node* n) {
    return (n == nullptr) ? 0 : n->height;
}

int BinaryTree::getBalance(Node* n) {
    return (n == nullptr) ? 0 :
    getHeight(n->left) - getHeight(n->right);
}

Node* BinaryTree::rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;
```

```
    x->right = y;
    y->left = T2;

    y->height = max(getHeight(y->left),
     getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left),
     getHeight(x->right)) + 1;

    return x;
}

Node* BinaryTree::rotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(getHeight(x->left),
     getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left),
     getHeight(y->right)) + 1;

    return y;
}

Node* BinaryTree::insertNode(Node* node, int value) {
    if (node == nullptr) {
        Node* newNode = new Node{value, nullptr, nullptr, 1};
        return newNode;
    }

    if (value < node->data)
        node->left = insertNode(node->left, value);
    else if (value > node->data)
        node->right = insertNode(node->right, value);
    else
        return node;

    node->height = 1 + max(getHeight(node->left),
    getHeight(node->right));

    int balance = getBalance(node);
```

```
    if (balance > 1 && value < node->left->data)
        return rotateRight(node);

    if (balance < -1 && value > node->right->data)
        return rotateLeft(node);

    if (balance > 1 && value > node->left->data) {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }

    if (balance < -1 && value < node->right->data) {
        node->right = rotateRight(node->right);
        return rotateLeft(node);
    }

    return node;
}

void BinaryTree::insert(int value) {
    root = insertNode(root, value);
}

Node* BinaryTree::minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

Node* BinaryTree::deleteNode(Node* root, int key) {
    if (root == nullptr)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if ((root->left == nullptr) || (root->right == nullptr)) {
            Node* temp = root->left ? root->left : root->right;
```

```
            if (temp == nullptr) {
                temp = root;
                root = nullptr;
            } else {
                *root = *temp;
            }
            delete temp;
        } else {
            Node* temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }

    if (root == nullptr)
        return root;

    root->height = 1 + max(getHeight(root->left),
getHeight(root->right));

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)
        return rotateRight(root);

    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = rotateLeft(root->left);
        return rotateRight(root);
    }

    if (balance < -1 && getBalance(root->right) <= 0)
        return rotateLeft(root);

    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rotateRight(root->right);
        return rotateLeft(root);
    }

    return root;
}

void BinaryTree::deleteValue(int value) {
    root = deleteNode(root, value);
```

```
}

void BinaryTree::update(int oldVal, int newVal) {
    deleteValue(oldVal);
    insert(newVal);
}

void BinaryTree::inorder(Node* node) {
    if (node == nullptr) return;
    inorder(node->left);
    cout << node->data << " ";
    inorder(node->right);
}

void BinaryTree::preorder(Node* node) {
    if (node == nullptr) return;
    cout << node->data << " ";
    preorder(node->left);
    preorder(node->right);
}

void BinaryTree::postorder(Node* node) {
    if (node == nullptr) return;
    postorder(node->left);
    postorder(node->right);
    cout << node->data << " ";
}

void BinaryTree::inorder() { inorder(root); cout << endl; }
void BinaryTree::preorder() { preorder(root); cout << endl; }
void BinaryTree::postorder() { postorder(root); cout << endl; }
```

Guided 1 (tree.h)

```
#ifndef TREE_H
#define TREE_H

struct Node {
    int data;
    Node *left, *right;
    int height;
```

```
};

class BinaryTree {
private:
    Node* root;

    Node* insertNode(Node* node, int value);
    Node* deleteNode(Node* node, int value);

    int getHeight(Node* node);
    int getBalance(Node* node);

    Node* rotateLeft(Node* node);
    Node* rotateRight(Node* node);

    Node* minValueNode(Node* node);

    void inorder(Node* node);
    void preorder(Node* node);
    void postorder(Node* node);

public:
    BinaryTree();
    void insert(int value);
    void deleteValue(int value);
    void update(int oldVal, int newVal);

    void inorder();
    void preorder();
    void postorder();
};
#endif
```

Guided 1 (main.cpp)

```
#include <iostream>
#include "tree.h"
#include "tree.cpp"

using namespace std;
```

```cpp
int main() {
    BinaryTree tree;

    cout <<"=== INSERT DATA ===" << endl;
    tree.insert(10);
    tree.insert(15);
    tree.insert(20);
    tree.insert(30);
    tree.insert(35);
    tree.insert(40);
    tree.insert(50);

    cout << "Data yang diinsert: 10,15,20,30,45,40,50" << endl;

    cout << "\nTraversal setelah insert: " << endl;
    cout << "Inorder: "; tree.inorder();
    cout << "Preorder: "; tree.preorder();
    cout << "Postorder: "; tree.postorder();

    cout << "\n=== UPDATE DATA ===" << endl;
    cout << "Sebelum update (20 -> 25):" << endl;
    cout << "Inorder: "; tree.inorder();

    tree.update(20, 25);

    cout << "Setelah update (20 -> 25):" << endl;
    cout << "Inorder: "; tree.inorder();

    cout << "\n=== DELETE DATA ===" << endl;
    cout << "Sebelum delete (hapus subtree dengan root = 30):" <<
endl;
    cout << "Inorder: "; tree.inorder();

    tree.deleteValue(30);

    cout << "Setelah delete (subtree root = 30 dihapus):" << endl;
    cout << "Inorder: "; tree.inorder();

    return 0;
}
```

Screenshots Output:

```
PS D:\TelkomUniversity\Mata Kuliah\Semester 3\Struktur Data\Praktikum>  & 'c:\Users\ACER\.vscode\extensio
ns\ms-vscode.cpptools-1.29.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIE
ngine-In-f5rh0apj.mur' '--stdout=Microsoft-MIEngine-Out-sp10zoyi.kvs' '--stderr=Microsoft-MIEngine-Error-
opucbwmf.el3' '--pid=Microsoft-MIEngine-Pid-gzgoexy0.swh' '--dbgExe=C:\Users\ACER\mingw32\bin\gdb.exe' '-
-interpreter=mi'
=== INSERT DATA ===
Data yang diinsert: 10,15,20,30,45,40,50

Traversal setelah insert:
Inorder: 10 15 20 30 35 40 50
Preorder: 30 15 10 20 40 35 50
Postorder: 10 20 15 35 50 40 30

=== UPDATE DATA ===
Sebelum update (20 -> 25):
Inorder: 10 15 20 30 35 40 50
Setelah update (20 -> 25):
Inorder: 10 15 25 30 35 40 50

=== DELETE DATA ===
Sebelum delete (hapus subtree dengan root = 30):
Inorder: 10 15 25 30 35 40 50
Setelah delete (subtree root = 30 dihapus):
Inorder: 10 15 25 35 40 50
```

Deskripsi:

Program ini merupakan implementasi dari AVL Tree, yaitu bentuk khusus dari Binary Search Tree yang selalu menjaga keseimbangan tinggi antar subpohon setelah setiap operasi insert maupun delete. Struktur data diatur melalui tiga file utama: tree.h, tree.cpp, dan main.cpp. Pada file tree.h, didefinisikan struktur Node yang menyimpan data, pointer kiri dan kanan, serta tinggi node, serta kelas BinaryTree yang berisi deklarasi fungsi-fungsi penting seperti insert, delete, update, perhitungan tinggi, balance factor, rotasi, dan traversal. File tree.cpp berisi implementasi seluruh fungsi tersebut, termasuk mekanisme rotasi (LL, RR, LR, RL) yang digunakan untuk menyeimbangkan tree secara otomatis, pemrosesan insert dan delete secara rekursif, serta fungsi traversal inorder, preorder, dan postorder. Pada file main.cpp, program melakukan simulasi dengan memasukkan beberapa nilai ke dalam AVL Tree, menampilkan hasil traversal, melakukan operasi update dengan menghapus dan menambahkan ulang nilai tertentu, serta menghapus salah satu node dalam tree. Setelah setiap operasi, AVL Tree secara otomatis menyesuaikan strukturnya agar tetap seimbang.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1 (bstree.cpp)

```cpp
#include "bstree.h"

address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}

void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else {
        if (x < root->info) {
            insertNode(root->left, x);
        } else if (x > root->info) {
            insertNode(root->right, x);
        }
        // jika sama, dilewati (BST tidak boleh duplikat)
    }
}

address findNode(address root, infotype x) {
    if (root == Nil) return Nil;
    if (x == root->info) return root;
    if (x < root->info) return findNode(root->left, x);
    return findNode(root->right, x);
}

void printInOrder(address root) {
    if (root != Nil) {
        printInOrder(root->left);
        cout << root->info << " ";
        printInOrder(root->right);
    }
}

/* ======== Tambahan Soal 2 ======== */

int hitungJumlahNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungJumlahNode(root->left) +
```

```
hitungJumlahNode(root->right);
}

int hitungTotalInfo(address root) {
    if (root == Nil) return 0;
    return root->info + hitungTotalInfo(root->left) +
hitungTotalInfo(root->right);
}

int hitungKedalaman(address root) {
    if (root == Nil) return 0;
    int L = hitungKedalaman(root->left);
    int R = hitungKedalaman(root->right);
    return 1 + max(L, R);
}

/* Tambahan soal 3 */
void printPreOrder(address root) {
    if (root != Nil) {
        cout << root->info << " ";
        printPreOrder(root->left);
        printPreOrder(root->right);
    }
}

void printPostOrder(address root) {
    if (root != Nil) {
        printPostOrder(root->left);
        printPostOrder(root->right);
        cout << root->info << " ";
    }
}



    infotype x = Q.info[Q.head];

    if (Q.head == Q.tail) {
        // kembali jadi kosong
        Q.head = -1;
        Q.tail = -1;
    } else {
        // geser elemen ke kiri (ALTERNATIF 1)
        for (int i = Q.head; i < Q.tail; i++) {
```

```
            Q.info[i] = Q.info[i + 1];
        }
        Q.tail--;
    }

    return x;
}

void printInfo(Queue Q) {
    cout << Q.head << " - " << Q.tail << " | ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    for (int i = Q.head; i <= Q.tail; i++) {
        cout << Q.info[i] << " ";
    }
    cout << endl;
}
```

Unguided 1 (bstree.h)

```
#ifndef BSTREE_H_INCLUDED
#define BSTREE_H_INCLUDED

#include <iostream>
using namespace std;

#define Nil NULL

typedef int infotype;
typedef struct Node *address;

struct Node {
    infotype info;
    address left;
    address right;
};
```

```
// Primitif dasar
address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(address root, infotype x);
void printInOrder(address root);

// Tambahan soal nomor 2
int hitungJumlahNode(address root);
int hitungTotalInfo(address root);
int hitungKedalaman(address root);

#endif
```

Unguided 1 (main.cpp)

```
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"

using namespace std;

int main() {
    cout << "Hello World\n";

    address root = Nil;

    insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 6);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 3);
    insertNode(root, 6);  // duplikat (tidak masuk BST)
    insertNode(root, 7);

    cout << "Inorder Traversal : ";
    printInOrder(root);
    cout << endl;

    cout << "Preorder Traversal: ";
    printPreOrder(root);
```

```
    cout << endl;

    cout << "Postorder Traversal: ";
    printPostOrder(root);
    cout << endl;

    cout << "\nKedalaman Tree : " << hitungKedalaman(root) <<
endl;
    cout << "Jumlah Node    : " << hitungJumlahNode(root) << endl;
    cout << "Total Info     : " << hitungTotalInfo(root) << endl;

    return 0;
}
```

Screenshots Output:

```
PS D:\TelkomUniversity\Mata Kuliah\Semester 3\Struktur Data\Praktikum>  & 'c:\Users\ACER\.vscode\extensio
ns\ms-vscode.cpptools-1.29.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIE
ngine-In-sksydzsw.gxm' '--stdout=Microsoft-MIEngine-Out-antwwvtf.mtx' '--stderr=Microsoft-MIEngine-Error-
xvn2kauf.tq1' '--pid=Microsoft-MIEngine-Pid-zq2kwnbu.fk0' '--dbgExe=C:\Users\ACER\mingw32\bin\gdb.exe' '-
-interpreter=mi'
Hello World
Inorder Traversal : 1 2 3 4 5 6 7
Preorder Traversal: 1 2 6 4 3 5 7
Postorder Traversal: 3 5 4 7 6 2 1

Kedalaman Tree : 5
Jumlah Node    : 7
Total Info     : 28
```

Deskripsi:

Program ini mengimplementasikan Binary Search Tree (BST) menggunakan struktur linked list. Setiap node memiliki info (bilangan integer) serta dua pointer yaitu left dan right.

Fungsi insertNode() digunakan untuk memasukkan data baru ke dalam BST sesuai aturan: nilai lebih kecil masuk ke subtree kiri dan nilai lebih besar ke subtree kanan. Program juga menyediakan fitur pencarian (findNode), serta traversal InOrder, PreOrder, dan PostOrder.

Selain itu, tiga fungsi tambahan dibuat sesuai soal:

- hitungJumlahNode() → menghitung total node dalam BST.
- hitungTotalInfo() → menjumlahkan seluruh nilai info.
- hitungKedalaman() → menentukan kedalaman maksimum tree.

Program dijalankan melalui main.cpp dengan memasukkan beberapa nilai dan menampilkan hasil traversal serta perhitungan tadi.

D. Kesimpulan

Tree merupakan struktur data hierarkis yang sangat penting dalam pemrograman karena mampu merepresentasikan hubungan data secara lebih alami dibandingkan struktur linear seperti array atau linked list. Pada tree, setiap elemen (node) dapat memiliki banyak anak, sehingga cocok untuk menggambarkan data bertingkat seperti struktur folder, organisasi, ekspresi matematika, hingga basis data.

Struktur data tree memungkinkan operasi pencarian, penyisipan, dan penghapusan dilakukan lebih efisien, terutama pada varian yang teratur seperti Binary Search Tree (BST). Selain itu, berbagai jenis tree seperti AVL Tree, Red-Black Tree, Heap, dan Trie memanfaatkan konsep keseimbangan untuk menjaga performa tetap optimal. Tree juga mendukung berbagai metode traversal (inorder, preorder, postorder) yang digunakan untuk menelusuri dan memproses data sesuai kebutuhan.

E. Referensi

https://bse.telkomuniversity.ac.id/tree-data-structure/

https://terapan-ti.vokasi.unesa.ac.id/post/struktur-data-pengertian-fungsi-dan-penerapannya

https://bpmbkm.uma.ac.id/2025/10/08/membongkar-cara-kerja-tree-dalam-struktur-data/