LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL IX
TREE

Disusun Oleh :

NAMA : RISKY CAHAYU

NIM : 103112430121

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Binary Search Tree (BST) adalah salah satu struktur data berbasis pohon biner yang digunakan untuk menyimpan data secara terurut sehingga proses pencarian, penyisipan, dan penghapusan dapat dilakukan secara efisien. Setiap node pada BST memiliki maksimal dua anak, yaitu left child dan right child, dengan aturan utama bahwa nilai pada left subtree selalu lebih kecil dari nilai pada node induk, sedangkan nilai pada right subtree selalu lebih besar. Aturan ini membuat BST sangat efektif untuk operasi pencarian karena memungkinkan proses divide and conquer dalam menelusuri data.

Dalam implementasinya, BST dibangun menggunakan dynamic memory allocation dengan setiap node menyimpan info, left pointer, dan right pointer. Fungsi insertNode digunakan untuk menambahkan elemen baru dengan mengikuti aturan BST, sedangkan findNode digunakan untuk mencari elemen dengan menelusuri cabang kiri atau kanan sesuai nilai yang dicari. Traversal pohon seperti in-order, pre-order, dan post-order digunakan untuk membaca isi pohon dalam urutan tertentu, di mana in-order traversal menghasilkan data dalam bentuk terurut menaik. Selain itu, pohon dapat dianalisis lebih lanjut melalui fungsi seperti hitungJumlahNode (menghitung total node), hitungTotalInfo (menjumlahkan seluruh nilai), dan hitungKedalaman (mengukur tinggi pohon). Semua operasi tersebut menggunakan pendekatan rekursif untuk memanfaatkan struktur hierarki dari tree. Secara keseluruhan, BST menjadi dasar penting dalam implementasi berbagai struktur data yang lebih kompleks seperti AVL Tree, Red-Black Tree, dan Binary Heap.

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

Guided 1

Kode main.cpp

```cpp
// Risky Cahayu
// 103112430121

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

using namespace std;

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, 35, 40, 50" << endl;

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

    cout << "\n=== UPDATE DATA ===" << endl;
    cout << "Sebelum update (20 -> 25):" << endl;
    cout << "Inprder    : "; 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;
}
```

Kode tree.cpp

```
// Risky Cahayu
// 103112430121

#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; }
```

Kode tree.h

```
// Risky Cahayu
// 103112430121

#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* rotateRight(Node* y);
    Node* rotateLeft(Node* x);

    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
```

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\modul 10\Guided> cd "c:\Users\ASUS\Videos\strukdat\modul 10\Guided\" ;
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
=== INSERT DATA ===
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

Traversal setelah insert:
Inroder    : 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):
Inprder    : 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
PS C:\Users\ASUS\Videos\strukdat\modul 10\Guided>
```

Deskripsi:

Kode di atas merupakan implementasi AVL Tree, yaitu Binary Search Tree (BST) yang selalu menjaga keseimbangan tinggi (height) pada setiap node agar operasi insert, delete, dan update tetap efisien. Program ini melakukan tiga operasi utama: insert untuk menambah data sambil menyeimbangkan tree menggunakan rotasi kiri/kanan, update

dengan cara menghapus nilai lama dan memasukkan nilai baru, serta delete yang menghapus node kemudian menyeimbangkan kembali strukturnya. Selain itu tersedia fungsi traversal (inorder, preorder, postorder) untuk menampilkan isi tree. Pada fungsi main, program memasukkan beberapa nilai, menampilkan traversal, melakukan update nilai 20 menjadi 25, kemudian menghapus subtree dengan root 30, lalu menampilkan hasil akhirnya. Dengan demikian, kode ini bertujuan membangun pohon AVL yang stabil dan efisien sambil mendemonstrasikan cara memasukkan, memperbarui, dan menghapus data beserta output traversalnya.

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

Unguided 1 #Alternatif 1 (head diam, tail bergerak)

Kode main.cpp

```cpp
// Risky Cahayu
// 103112430121

#include <iostream>
#include "bstree.h"
using namespace std;

int main() {
    cout << "Hello world!" << endl;

    address root = NULL;

    insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 6);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 3);
    insertNode(root, 7);

    inOrder(root);
    cout << endl << endl;

    cout << "kedalaman : " << hitungKedalaman(root) << endl;
    cout << "jumlah node : " << hitungJumlahNode(root) << endl;
    cout << "total : " << hitungTotalInfo(root) << endl << endl;

    cout << "PreOrder  : ";
    preOrder(root);
    cout << endl;

    cout << "PostOrder : ";
    postOrder(root);
```

```
    cout << endl;

    return 0;
}
```

Kode bstree.h

```
// Risky Cahayu
// 103112430121

#ifndef BSTREE_H
#define BSTREE_H

#include <iostream>
using namespace std;

typedef int infotype;

typedef struct Node* address;

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

address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(address root, infotype x);
void inOrder(address root);
void preOrder(address root);
void postOrder(address root);

int hitungJumlahNode(address root);
int hitungTotalInfo(address root);
int hitungKedalaman(address root);

#endif
```

Kode bstree.cpp

```
// Risky Cahayu
// 103112430121

#include "bstree.h"

address alokasi(infotype x) {
```

```
    address p = new Node;
    p->info = x;
    p->left = NULL;
    p->right = NULL;
    return p;
}

void insertNode(address &root, infotype x) {
    if (root == NULL) {
        root = alokasi(x);
    } else {
        if (x < root->info)
            insertNode(root->left, x);
        else
            insertNode(root->right, x);
    }
}

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

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

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

void postOrder(address root) {
    if (root != NULL) {
        postOrder(root->left);
        postOrder(root->right);
        cout << root->info << " - ";
    }
}
```

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

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

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

Screenshots Output

```
Hello world!
1 - 2 - 3 - 4 - 5 - 6 - 7 -

kedalaman : 5
jumlah node : 7
total : 28

PreOrder  : 1 - 2 - 6 - 4 - 3 - 5 - 7 -
PostOrder : 3 - 5 - 4 - 7 - 6 - 2 - 1 -
PS C:\Users\ASUS\Videos\strukdat\modul 10\Unguided>
```

Deskripsi:

Kode di atas merupakan implementasi sederhana dari Binary Search Tree (BST) yang digunakan untuk menyimpan data secara terurut dan melakukan berbagai operasi dasar pada struktur data tree. Program menyediakan fungsi untuk alokasi node, insertNode untuk memasukkan data ke tree sesuai aturan BST (nilai lebih kecil ke kiri, lebih besar ke kanan), serta findNode untuk mencari nilai tertentu. Selain itu terdapat fungsi traversal yaitu in-order, pre-order, dan post-order untuk menampilkan isi tree dengan urutan yang berbeda. Program juga menghitung statistik tree, seperti jumlah node, total nilai info, dan kedalaman (height) tree menggunakan rekursi. Pada fungsi main, program membangun BST dengan memasukkan beberapa angka, lalu menampilkan hasil traversal dan informasi penting mengenai struktur tree tersebut. Tujuan keseluruhannya adalah mendemonstrasikan cara membuat, mengelola, dan menganalisis BST menggunakan operasi dasar dan rekursi.

D. Kesimpulan

Kode program yang diimplementasikan pada praktikum ini bertujuan untuk membangun dan memanipulasi struktur data Binary Search Tree (BST) dengan berbagai operasi dasar seperti penyisipan (insert), pencarian (find), dan tiga jenis traversal (in-order, pre-order, dan post-order). Melalui penggunaan pointer dan rekursi, program mampu membentuk hubungan hierarkis antar-node sehingga setiap operasi dilakukan dengan mengikuti aturan BST, yaitu penempatan elemen lebih kecil di subtree kiri dan elemen lebih besar di subtree kanan. Selain itu, program juga menyediakan fungsi analisis seperti menghitung jumlah node, total nilai seluruh elemen, serta kedalaman pohon yang memberikan gambaran mengenai kompleksitas dan karakteristik struktur tree yang terbentuk.

Secara keseluruhan, kode ini menunjukkan bagaimana BST dapat digunakan sebagai struktur data yang efisien dan terorganisasi untuk menyimpan serta mengolah data secara terurut. Penggunaan traversal membantu menampilkan isi pohon dalam berbagai bentuk urutan, sedangkan perhitungan statistik seperti kedalaman dan jumlah node memperlihatkan bagaimana kondisi pohon mempengaruhi kinerja operasi lainnya. Dengan memahami implementasi ini, mahasiswa dapat mengenali konsep fundamental tree dan rekursi yang menjadi dasar bagi struktur data yang lebih kompleks seperti AVL Tree, Red-Black Tree, dan berbagai algoritma pencarian yang banyak diterapkan di bidang informatika.

E. Referensi

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3rd ed.)*. MIT Press, 2009.
(Bab mengenai Trees dan Binary Search Trees)

Weiss, Mark Allen.
Data Structures and Algorithm Analysis in C++ (4th ed.). Pearson, 2014.
(Pembahasan Binary Trees dan Binary Search Trees)

TutorialsPoint. *Data Structures – Binary Search Tree*.
https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm