LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X
TREE

Disusun Oleh :
NAMA : ZAKI MAULA DHIA
NIM : 103112400127

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

TREE adalah struktur data non-linier terpenting dan sangat fleksibel dalam komputasi TREE. Terminologi utama dalam struktur data pohon berasal dari pohon keluarga, dengan istilah “parent,” “child,” “ancestor,” dan “descendant”. Susunan dari satu atau lebih simpul (node) yang terdiri dari satu simpul khusus yang disebut akar (root) sedangkan sisanya membentuk seubtree dari akar.

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

Guided 1

```
// file 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* 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
```

```
// file tree.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);
}
```

```cpp
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; }
```

```cpp
// file main.cpp
#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 << "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 (hapus subtree dengan root = 30):" << endl;
    cout << "Inorder : "; tree.inorder();

    return 0;
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>  &
pters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-mz
Error-dmhrkjzz.ufh' '--pid=Microsoft-MIEngine-Pid-fgri2lxq.2av' '--db
=== INSERT DATA ===
Data yang diinsert: 10, 15, 20, 30, 35, 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 (hapus subtree dengan root = 30):
Inorder : 10 15 25 35 40 50
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>
```

Deskripsi:

Program ini membuat dan mengelola AVL Tree yang memiliki kemampuan self-blancing. Dengan function insertNode() yang akan memasukan semua angka yang diinput dengan mengikuti aturan, nilai lebih kecil masuk ke kiri, dan nilai lebih besar masuk ke kanan. Dan kemudian akan menampilkan hasil dari TREE yang dibuat program dalam bentuk tranversal Inorder, Preorder, dan Postorder. Setelah hasil transversal diketahui maka tree yang dibuat dari program tersebut dapat diketahui.

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

Unguided 1

```cpp
// main.cpp
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"
using namespace std;

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

    address root = nullptr;

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

    printInOrder(root);

    return 0;
}
```

```h
// bstree.h
#ifndef BSTREE_H
#define BSTREE_H

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(infotype x, address root);

void printInOrder(address root);

#endif
```

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

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

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

```
        insertNode(root->left, x);
    }
    else if (x > root->info) {
        insertNode(root->right, x);
    }
}

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

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

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>  & '
pters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-dv
Error-1ye0hcgb.cvv' '--pid=Microsoft-MIEngine-Pid-sbv3saf4.1je' '--db
Hello world!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>
```

Deskripsi:

Program tersebut dibuat untuk menunjukkan cara membuat struktur data tree yang simpel dan hanya menggunakan insert, dan beberapa function penting dalam struktur data tree. Output yang didapat adalah hasil dari tranversal inorder sehingga hasilnya urut mulai dari yang terkecil sampai dengan yang terbesar.

Unguided 2

```cpp
// main.cpp
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"
using namespace std;

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

    address root = nullptr;

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

    printInOrder(root);

    cout << "\n";
    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah node : " << hitungJumlahNode(root) << endl;
    cout << "total : " << hitungTotalInfo(root) << endl;

    return 0;
}
```

```cpp
// bstree.h
#ifndef BSTREE_H
#define BSTREE_H

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(infotype x, address root);

void printInOrder(address root);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root, int start);

#endif
```

```
// stack.cpp
#include <iostream>
#include "bstree.h"
using namespace std;

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

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

address findNode(infotype x, address root) {
    if (root == nullptr) return nullptr;
```

```
    if (x == root->info) return root;
    if (x < root->info) return findNode(x, root->left);
    return findNode(x, root->right);
}

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

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

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

int hitungKedalaman(address root, int start) {
    if (root == nullptr) return 0;

    int leftDepth  = hitungKedalaman(root->left, start);
    int rightDepth = hitungKedalaman(root->right, start);

    return 1 + (leftDepth > rightDepth ? leftDepth : rightDepth);
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>  & '
pters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ku
Error-s1pkavj5.ukk' '--pid=Microsoft-MIEngine-Pid-ltsrruom.ma2' '--db
Hello world!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah node : 7
total : 28
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>
```

Deskripsi:

Program unguided 1 yang ditambah dengan beberapa function untuk mencari kedalaman dari tree yang dibuat, berapa banyak node yang dibuat, dan berapa total angka yang ada disetiap node. Function tersebut adalah hitungJumlahNode() yang berfungsi mengembalikan integer banyak node yang ada di dalam BST, hitungTotalInfo() berfungsi mengembalikan jumlah (total) info dari node-node yang ada di dalam BST, dan hitungKedalaman() yang memiliki fungsi rekursif mengembalikan integer kedalaman maksimal dari binary tree.

Unguided 3

```cpp
// main.cpp
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"
using namespace std;

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

    address root = nullptr;

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

    printInOrder(root);

    cout << "\n";
    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah node : " << hitungJumlahNode(root) << endl;
    cout << "total : " << hitungTotalInfo(root) << endl;

    cout << "\nPreOrder : ";
    printPreOrder(root);

    cout << "\nPostOrder : ";
    printPostOrder(root);
```

```
    return 0;
}
```

```
// bstree.h
#ifndef BSTREE_H
#define BSTREE_H

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(infotype x, address root);

void printInOrder(address root);

void printPreOrder(address root);

void printPostOrder(address root);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root, int start);

#endif
```

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

```cpp
using namespace std;

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

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

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

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

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

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

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

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

int hitungKedalaman(address root, int start) {
    if (root == nullptr) return 0;

    int leftDepth  = hitungKedalaman(root->left, start);
    int rightDepth = hitungKedalaman(root->right, start);

    return 1 + (leftDepth > rightDepth ? leftDepth : rightDepth);
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata\modul 10test>  &
pters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-k
Error-y1amk0ct.xgr' '--pid=Microsoft-MIEngine-Pid-l1ueb55w.3tu' '--d
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\MyBook Hype AMD\Documents\strukturdata\modul 10test>
```

Deskripsi:

Program yang sama seperti di nomor 1 dan 2 tetapi di soal 3 ditambah function untuk menamilkan tranversal preorder dan postorder, kedua tranversal tersebut memiliki sistem pengurutan yang berbeda sehingga hasilnya juga berbeda.

D. Kesimpulan

kesimpulan dari materi TREE adalah bahwa struktur data TREE dapat menjadi lebih efisien daripada singel linked list atapun yang lainnya karena struktur data TREE mengurutkan dan memastikan angka disebelah kiri lebih kecil daripada angka disebelah kanannya, sehingga program dapat mengurutkannya dengan lebih mudah dan cepat.

E. Referensi

- Ginting, S. H. N., Effendi, H., Kumar, S., Marsisno, W., Sitanggang, Y. R. U., Anwar, K., ... & Smrti, N. N. E. (2024). Pengantar struktur data. *Penerbit Mifandi Mandiri Digital*, *1*(01).
- Soetanto, H. (2022). Struktur Data.