LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL TREE

Disusun Oleh :

NAMA : Ayu Setyaning Tyas

NIM : 103112430119

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Struktur Data Tree adalah struktur data non-linear di mana kumpulan elemen yang dikenal sebagai node terhubung satu sama lain melalui tepi sehingga ada tepat satu jalur antara dua node. A Pohon biner adalah struktur data hierarkis berbasis pohon di mana setiap node dapat memiliki paling banyak dua anak. Simpul paling atas disebut akar, dan node tanpa anak disebut daun. Untuk mengimplementasikan pohon biner di C ++, kita akan menggunakan pendekatan berbasis node. Setiap simpul pohon biner akan berisi data dan petunjuk ke anak-anak kiri dan kanannya.

Dalam struktur data Tree ada yang di sebut Node adalah struktur yang mungkin berisi data dan koneksi ke node lain, kadang-kadang disebut tepi atau linkslink. Setiap node dalam pohon memiliki nol atau lebih child nodesnode anak, yang di bawahnya di pohon (dengan konvensi, pohon ditarik dengan *keturunan* ke bawah). Sebuah node yang memiliki anak disebut node orang tua anak (atau atasan). Semua node memiliki tepat satu orang tua, kecuali simpul akar paling atas, yang tidak memilikinya. Sebuah node mungkin memiliki banyak node leluhur, seperti orang tua orang tua. Simpul anak dengan orang tua yang sama adalah silly node. Biasanya saudara kandung memiliki perintah, dengan yang pertama secara konvensional ditarik di sebelah kiri. Beberapa definisi memungkinkan pohon untuk tidak memiliki node sama sekali, dalam hal ini disebut *empty*kosong.

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

Guided 1

Code *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);

    Node 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
```

Code *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);
}

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);
```

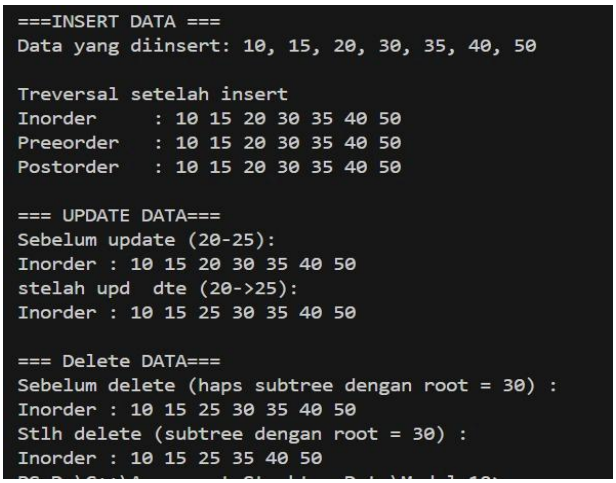
```
    postorder(node->right);
    cout << node->data << " ";
}

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

Code *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 root = 30): " << endl;
    cout << "Inorder: "; tree.inorder();

    tree.deleteValue(30);
```

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

    return 0;
}
```

Screenshoot Output



Deskripsi:

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

Unguided 1

Code *bstree.h*

```
#ifndef BSTREE_H
#define BSTREE_H

#define Nil NULL

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 inorder(address root);
```

```
#endif
```

Code *bstree.cpp*

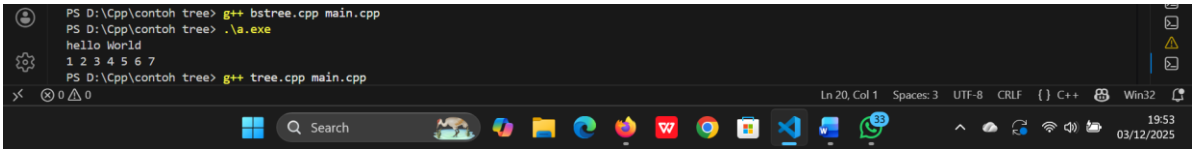
```
#include <iostream>
#include "bstree.h"
using namespace std;

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);
   }
}

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

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

Code *main.cpp*

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

```
using namespace std;

int main() {
  cout << "hello World" << endl;
  address root = Nil;
  insertNode(root, 1);
  insertNode(root, 2);
  insertNode(root, 6);
  insertNode(root, 4);
  insertNode(root, 5);
  insertNode(root, 3);
  insertNode(root, 6);
  insertNode(root, 7);
  inorder(root);
  return 0;
}
```

Screenshots Output



Deskripsi:

Program ini menggunakan Binary search Tree (BST), Program ini digunakan untuk membuat struktur tree yang dinamis dan mengurutkan data dengan traversal inorder dari nilai terkecil di sebelah kiri dan nilai terbesar di sebelah kanan.

Unguided 2

Code *bstree.h*

```
int hitungNode(address root);

int hitungTotal(address root, int start);

int hitungKedalaman(address root, int start);
```

Code *bstree.cpp*

**Sama dengan code sebelumnya hanya saja ada penambahan fungsi baru yang sesuai dengan yang ada di bstree.h*

```
int hitungNode(address root) {
   if (root == Nil) {
      return 0;
   } else {
      return 1 + hitungNode(root->left) + hitungNode(root->right);
```

```
    }
}
```

```
int hitungTotal(address root, int start) {
    if (root == Nil) {
        return start;
    } else {
        start += root->info;
        start = hitungTotal(root->left, start);
        start = hitungTotal(root->right, start);
        return start;
    }
}
```

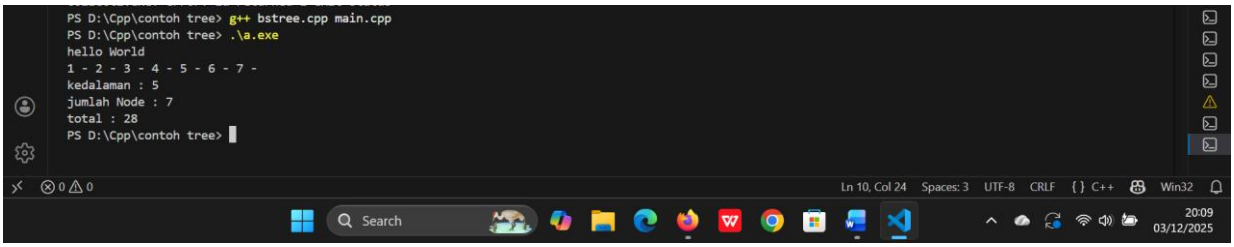
```
int hitungKedalaman(address root, int start) {
    if (root == Nil) {
        return start;
    } else {
        int leftDepth = hitungKedalaman(root->left, start + 1);
        int rightDepth = hitungKedalaman(root->right, start + 1);
        return (leftDepth > rightDepth) ? leftDepth : rightDepth;
    }
}
```

Code *main.cpp*

**tidak ada berubahan hanya ada penambahan*

```
cout<<"\n";
   cout<<"kedalaman : "<< hitungKedalaman(root,0) << endl;
   cout<<"jumlah Node : "<< hitungNode(root) << endl;
   cout<<"total : "<< hitungTotal(root, 0) << endl;
```

Screenshots Output



Deskripsi:

Program ini hamper sama dengan program sebelumnya hanya saja ada tambahan beberapa fungsi dalam code yang berguna untuk menghitung jumlah node, menjumlahkan total info pada node, dan menghitung kelaman maksimumnya.

Unguided 3

Code *bstree.h*

```
void printPreOrder(address root);
void printPostOrder(address root);
```

Code *bstree.cpp*

**Sama dengan code sebelumnya hanya saja ada penambahan fungsi baru yang sesuai dengan yang ada di bstree.h*

```
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 << " - ";
    }
}
```

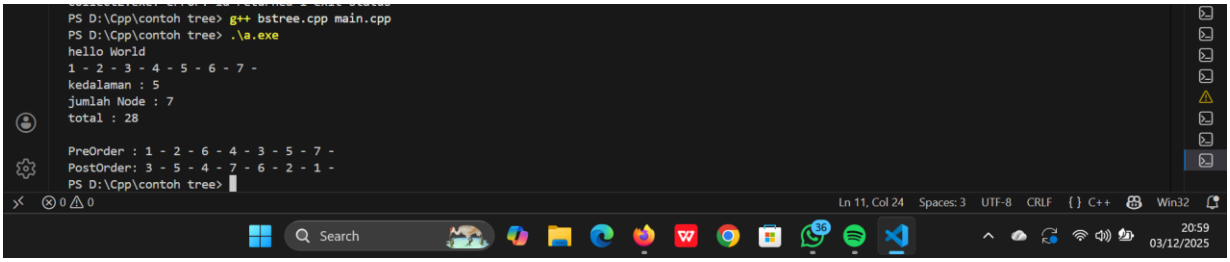
Code *main.cpp*

**tidak ada berubahan hanya ada penambahan*

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

```
cout << "\nPostOrder: ";
printPostOrder(root);
return 0;
```

Screenshoot Output



Deskripsi

Program ini juga kurang lebih sama dengan yang pertama dan kedua hanya saja ada tambahan fungsi baru yang digunakan untuk mengurutkan tree secara preorder dan post order. Pada preorder data atau nilai akan diurutkan dari root lalu ke kiri dan terakhir ke sebelah kanan, sedangkan postorder itu akan dimulai dari sebelah kiri lalu ke kanan dann terakhir kan menuju root. Program ini berguna untuk memahami proses traversal dalam banyak cara dan melihat bagaimana struktur tree memengaruhi urutan output.

D. Kesimpulan

Dari keseluruhan praktikum, mahasiswa mempelajari konsep fundamental mengenai struktur data Binary Search Tree, mulai dari implementasi dasar, operasi insert dan pencarian node, hingga traversal dan analisis struktur tree. Pembuatan ADT membantu mahasiswa memahami pentingnya modularitas dan pemisahan antara definisi serta implementasi. Selain itu, penggunaan rekursi pada perhitungan jumlah node, total nilai, dan kedalaman memperjelas bagaimana tree dapat diproses secara efisien. Tugas ini memperkuat pemahaman mahasiswa tentang struktur data non-linear dan menjadi dasar penting untuk materi lanjutan seperti AVL Tree, Heap, atau algoritma representasi tree lainnya.

E. Referensi

https://www.geeksforgeeks.org/dsa/tree-data-structure/
https://www.geeksforgeeks.org/cpp/binary-tree-in-cpp/
https://en.wikipedia.org/wiki/Tree_(abstract_data_type)