LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X
TREE

Disusun Oleh :
NAMA : HILMI HAKIM RAMADANI
NIM : 103112430016

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Tree dalam bahasa inggris diartikan Pohon. Dalam Struktur Data Konsep Tree mirip seperti pohon yaitu kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Struktur Tree adalah suatu cara merepresentasikan suatu struktur hirarki (one to many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Konsep Tree disebuah struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one to many) dan tidak linier antara elemen-elemennya.

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

Guided

#main.cpp

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

using namespace std;

int main()
{
    BinaryTree tree;

    cout << "=== INSERT DATA ===\n"
         << 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\n"
         << 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 diupdate (20 -> 25): " << endl;
```

```
    cout << "Inorder: ";
    tree.inOrder();

    tree.update(20, 25);

    cout << "sesudah diupdate (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;
}
```

#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::rightRotate(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::leftRotate(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 rightRotate(node);

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

    if (balance > 1 && value > node->left->data)
    {
        node->left = leftRotate(node->left);
```

```
        return rightRotate(node);
    }

    if (balance < -1 && value < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(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 rightRotate(root);

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

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

    if (balance < -1 && getBalance(root->right) > 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(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)
```

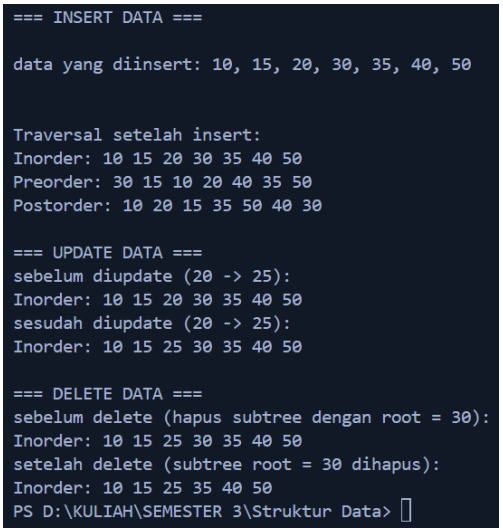
```
        return;
    cout << node->data << " ";
    preOrder(node->left);
    preOrder(node->right);
}

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;
}
```

#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 *rightRotate(Node *y);
```

```
    Node *leftRotate(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 // TREE_H
```

Screenshots Output



Deskripsi:

Tiga file kode yang diberikan membentuk implementasi AVL Tree di C++ dengan operasi dasar insert, update, delete, dan traversal. File tree.h berisi deklarasi struktur Node dan kelas BinaryTree, termasuk semua metode privat untuk manipulasi pohon seperti rotasi kiri/kanan, mencari node minimum, serta metode traversal (inorder, preorder, postorder). File tree.cpp berisi implementasi semua metode dari BinaryTree, termasuk algoritma insert dan delete yang menjaga keseimbangan AVL tree dengan rotasi yang sesuai, serta update node dengan menghapus nilai lama dan menyisipkan nilai baru. File main.cpp berfungsi sebagai program utama yang menggunakan kelas BinaryTree,

melakukan penyisipan beberapa nilai, menampilkan traversal pohon, memperbarui nilai tertentu, dan menghapus subtree tertentu, sambil menampilkan hasil setiap langkah ke layar. Dengan struktur ini, pengguna dapat melihat bagaimana AVL tree secara otomatis menyeimbangkan dirinya dan mendemonstrasikan operasi dasar pohon biner.

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

Soal1

#main.cpp

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

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);

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

   return 0;
}
```

#bstree.cpp

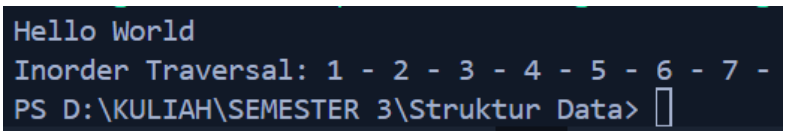
```
#include "bstree.h"

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

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

#bstree.h

```
#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;
};

#define Nil NULL

address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void printInorder(address root);

#endif
```

Screenshots Output



Soal 2

- Menambah code di bstree.h

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

- Menambah code di bstree.cpp

```
int hitungJumlahNode(address root)
{
   if (root == Nil)
   {
      return 0;
   }
   else
   {
```

```
        return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
      }
    }

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

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

- Menambah code di main.cpp

```
cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
cout << "jumlah Node : " << hitungJumlahNode(root) << endl;
cout << "total : " << hitungTotalInfo(root, 0) << endl;
```

- Output

```
Hello World
Inorder Traversal: 1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
PS D:\KULIAH\SEMESTER 3\Struktur Data>
```

Soal 3

- Tambahkan code di bstree.h

```
void printPreorder(address root);
void printPostorder(address root);
```

- Tambahkan code di bstree.cpp

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

- Tambahkan code di main.cpp

```
    cout << "\nPre-order  : ";
      printPreorder(root);
      cout << endl;

      cout << "Post-order : ";
      printPostorder(root);
  cout << endl;
```

- Output

```
Hello World
Inorder Traversal: 1 - 2 - 3 - 4 - 5 - 6 - 7
kedalaman : 5
jumlah Node : 7
total : 28

Pre-order  : 1 - 2 - 6 - 4 - 3 - 5 - 7 -
Post-order : 3 - 5 - 4 - 7 - 6 - 2 - 1 -
PS D:\KULIAH\SEMESTER 3\Struktur Data>
```

Deskripsi:

Program ADT Binary Search Tree (BST) terdiri dari tiga bagian utama yaitu bstree.h, bstree.cpp, dan main.cpp, di mana bstree.h berisi definisi tipe data BST berupa struktur node dengan data integer serta pointer ke anak kiri dan kanan, sekaligus deklarasi seluruh fungsi ADT. Pada bstree.cpp diimplementasikan fungsi-fungsi BST secara rekursif, meliputi pembuatan node baru, penyisipan data sesuai aturan BST, traversal inorder, preorder, dan postorder, serta fungsi untuk menghitung jumlah node, total nilai seluruh node, dan kedalaman maksimum pohon. Sementara itu, main.cpp berfungsi untuk menguji ADT dengan membuat BST, memasukkan beberapa data, menampilkan hasil traversal, serta menampilkan hasil perhitungan kedalaman, jumlah node, dan total nilai node, sehingga seluruh fitur ADT Binary Search Tree dapat berjalan dan terverifikasi dengan baik.

D. Kesimpulan

Berdasarkan praktikum modul Tree ini, dapat disimpulkan bahwa struktur data pohon (Tree) merupakan struktur non-linier yang sangat efektif untuk merepresentasikan data secara hierarkis. Implementasi AVL Tree pada program pertama menunjukkan bagaimana pohon biner dapat dijaga tetap seimbang secara otomatis melalui operasi rotasi saat melakukan penyisipan atau penghapusan node, sehingga menjaga kinerja pencarian tetap optimal. Sementara pada program Binary Search Tree (BST), praktikan dapat memahami operasi dasar seperti penyisipan, pencarian, dan traversal (inorder, preorder, postorder), serta menghitung jumlah node, total nilai, dan kedalaman pohon. Praktikum ini memperkuat pemahaman tentang konsep pohon, implementasi ADT pohon, serta pentingnya keseimbangan dalam pohon biner agar operasi tetap efisien. Dengan demikian, praktikan mampu mengaplikasikan teori tree dalam bentuk program komputer yang dapat melakukan manipulasi data secara dinamis dan terstruktur.

E. Referensi

Medium. 2021, 14 Januari. Memahami Konsep Tree dalam Struktur Data Lengkap dengan Source Code Programnya. Diakses pada 21 Desember. Dari https://daismabali.medium.com/memahami-konsep-tree-dalam-struktur-data-lengkap-dengan-source-code-programnya-acbd0a8733d6