LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 10
TREE (BAGIAN PERTAMA)

Disusun Oleh :

NAMA : Muhammad Luthfi Arrafi Ramadhani

NIM : 103112430043

Dosen Pengampu:

Fahrudin Mukti Wibowo

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Dasar teori pada modul ini membahas rekursif dan struktur data tree, terutama Binary Search Tree (BST). Rekursif adalah teknik ketika sebuah fungsi memanggil dirinya kembali untuk menyelesaikan bagian masalah yang lebih kecil, sehingga proses perhitungan menjadi lebih rapi. Sementara Tree merupakan struktur data yang terdiri dari simpul utama (root), simpul-simpul di dalamnya, serta simpul yang tidak memiliki turunan yang disebut leaf. Salah satu bentuk tree yang digunakan adalah binary tree, dan dari struktur ini lahir BST yang menyimpan data dengan aturan tertentu: nilai yang lebih kecil ditempatkan di bagian kiri dan nilai yang lebih besar di bagian kanan. BST memiliki operasi penting seperti insert, search, delete, serta traversal pre-order, in-order, dan post-order. Modul ini juga membahas cara menemukan nilai terkecil dan terbesar melalui most-left dan most-right, serta perhitungan jumlah simpul, total nilai, dan kedalaman tree menggunakan fungsi rekursif.

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

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 data);
    Node* deleteNode(Node* node, int data);

    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
```

(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; }
```

(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 << "\nTransversal 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 substree dengan root = 30): " << endl;

   cout << "Inorder: "; tree.inorder();

   tree.deleteValue(30);

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

   return 0;

}
```

Screenshots Output

```
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043> cd "e:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi
Arrafi Ramadhani-103112430043\Minggu 10\Guided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
===INSERT DATA ===
Data yang diinsert : 10, 15, 20, 30, 35, 40, 50

Transversal 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 substree dengan root = 30):
Inorder: 10 15 25 30 35 40 50
Setelah delete (substree root = 30 dihapus):
Inorder: 10 15 25 35 40 50
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043\Minggu 10\Guided>
```

Deskripsi:

Program tersebut merupakan implementasi struktur data Binary Search Tree. File **tree.h** digunakan sebagai tempat membuat struktur Node dan daftar fungsi di BinaryTree, fungsinya untuk memasukkan data, menghapus data, memperbarui data, menghitung tinggi node, melakukan rotasi, dan menampilkan isi tree. File **tree.cpp** berisi proses kerja dari semua fungsi itu, ada proses penyisipan dan penghapusan node, rotasi untuk menjaga keseimbangan tree, perhitungan tinggi dan selisih cabang, proses traversal inorder, preorder, dan postorder. Sementara itu, file **main.cpp** berfungsi menjalankan program dengan membuat objek BinaryTree, lalu melakukan operasi insert, update, dan delete, serta menampilkan hasil traversal untuk melihat perubahan yang terjadi pada tree setelah setiap operasi dijalankan.

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

Unguided 1

(bstree.h)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#ifndef BSTREE_H
#define BSTREE_H
#include <iostream>
using namespace std;

#define Nil NULL

typedef int infotype;

struct Node {
    infotype info;
```

```
    Node* left;
    Node* right;
};

typedef Node* address;

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

#endif
```

(bstree.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#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);
        }
    }
}

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);
    }
}
```

(main.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#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);

    InOrder(root);

    return 0;
}
```

Screenshots Output

```
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043> cd "e:\BACKUP092024\VSCODE\3rd sm\
Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043\Minggu 10\Unguided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) {
.\main }
Hello World!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043\Minggu 10\Unguided>
```

Deskripsi:

Program tersebut merupakan implementasi struktur data Binary Search Tree. Pada file **bstree.h**, didefinisikan struktur Node yang memuat nilai info serta pointer *left* dan *right*, dan dideklarasikan fungsi yang akan digunakan seperti *alokasi*, *insertNode, findNode*, dan *InOrder*. File **bstree.cpp** berisi isi dari fungsi-fungsi tersebut, dimulai dari *alokasi* yang membuat node baru, *insertNode* menempatkan data ke kiri atau kanan sesuai aturan BST, *findNode* menelusuri tree secara rekursif untuk menemukan nilai yang dicari, dan *InOrder* mencetak nilai-nilai dari yang paling kecil ke terbesar. Sementara itu, file **main.cpp** menjalankan program dengan membuat variabel *root,* memanggil *insertNode* untuk memasukkan beberapa data, lalu menampilkan hasil akhirnya menggunakan *InOrder* sehingga nilai-nilai dalam tree terlihat berurutan.

Unguided 2

(bstree.h)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#ifndef BSTREE_H
#define BSTREE_H
#include <iostream>
using namespace std;

#define Nil NULL

typedef int infotype;

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

typedef Node* address;

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

// SOAL 2
int hitungNode(address root);
int hitungTotal(address root);
int hitungKedalaman(address root, int start);

#endif
```

(bstree.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#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);
        }
    }
}

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);
    }
}

// SOAL 2
int hitungNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungNode(root->left) + hitungNode(root->right);
}

int hitungTotal(address root) {
    if (root == Nil) return 0;
```

```
    return root->info + hitungTotal(root->left) + hitungTotal(root->right);
}

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

(main.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#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);

    InOrder(root);
    cout << endl;
    // SOAL 2
    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah node : " << hitungNode(root) << endl;
    cout << "total : " << hitungTotal(root) << endl;

    return 0;
}
```

Screenshots Output

Deskripsi:

Program tersebut lanjutan dari implementasi Binary Search Tree dengan menambahkan tiga fungsi baru. Pada file **bstree.h**, selain deklarasi fungsi dasar BST, ditambahkan tiga fungsi yaitu *hitungNode, hitungTotal,* dan *hitungKedalaman*. Ketiga fungsi ini diisi pada **bstree.cpp**, di mana *hitungNode* digunakan untuk menghitung berapa banyak node yang ada dalam tree, *hitungTotal* menjumlahkan seluruh nilai info dari setiap node, dan *hitungKedalaman* menentukan seberapa dalam tree tersebut dengan cara menelusuri cabang kiri dan kanan secara rekursif. Pada file **main.cpp**, fungsi-fungsi tambahan ini dipanggil setelah proses insert untuk menampilkan kedalaman tree, jumlah node, serta total nilai yang tersimpan.

Unguided 3

(bstree.h)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#ifndef BSTREE_H
#define BSTREE_H
#include <iostream>
using namespace std;

#define Nil NULL

typedef int infotype;

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

typedef Node* address;

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

```
address findNode(infotype x, address root);
void InOrder(address root);

// SOAL 2
int hitungNode(address root);
int hitungTotal(address root);
int hitungKedalaman(address root, int start);

// SOAL 3
void PreOrder(address root);
void PostOrder(address root);

#endif
```

(bstree.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

#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);
        }
    }
}

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);
   }
}

// SOAL 2
int hitungNode(address root) {
   if (root == Nil) return 0;
   return 1 + hitungNode(root->left) + hitungNode(root->right);
}

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

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

// SOAL 3
void PreOrder(address root) {
   if (root != Nil) {
      cout << root->info << " - ";
      PreOrder(root->left);
      PreOrder(root->right);
   }
}

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

(main.cpp)

```
// Muhammad Luthfi Arrafi Ramadhani
// 103112430043
// IF 12-06

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

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

    address root = Nil;

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

    // InOrder(root);
    // cout << endl;
    // // SOAL 2
    // cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    // cout << "jumlah node : " << hitungNode(root) << endl;
    // cout << "total : " << hitungTotal(root) << endl;

    // SOAL 3
    cout << "\npre-order: ";
    PreOrder(root);

    cout << "\npost-order: ";
    PostOrder(root);

    return 0;
}
```

Screenshots Output

```
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043> cd "e:\BACKUP092024\VSCODE\3rd sm\
Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043\Minggu 10\Unguided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) {
.\main }
Hello World!

pre-order: 6 - 4 - 2 - 1 - 3 - 5 - 7 -
post-order: 1 - 3 - 2 - 5 - 4 - 7 - 6 -
PS E:\BACKUP092024\VSCODE\3rd sm\Praktikum Muhammad Luthfi Arrafi Ramadhani-103112430043\Minggu 10\Unguided>
```

Deskripsi:

Program tersebut merupakan pengembangan dari Binary Search Tree dengan menambahkan dua fungsi baru untuk menampilkan isi tree, yaitu *PreOrder* dan *PostOrder*. Pada file bstree.h ditambahkan deklarasi kedua fungsi tersebut, sementara proses kerjanya ditulis di bstree.cpp. Fungsi *PreOrder* mencetak data dimulai dari node akar lalu bergerak ke bagian kiri dan kanan, sehingga urutan yang muncul mengikuti struktur tree dari atas ke bawah. Sedangkan fungsi *PostOrder* mencetak node dari bagian kiri dan kanan terlebih dahulu, kemudian menampilkan nilai akarnya di bagian akhir. Pada file main.cpp, urutan insert disesuaikan supaya tree yang dihasilkan sama (seperti modul).

D. Kesimpulan

Dari ketiga latihan ini bisa disimpulkan bahwa Binary Search Tree menyimpan data dengan rapi berdasarkan besar kecilnya nilai. Latihan pertama fokus pada cara memasukkan data, mencari nilai, dan menampilkan isi tree secara inorder. Latihan kedua menambah fungsi untuk menghitung berapa banyak node yang ada, berapa total nilainya, dan seberapa dalam tree tersebut. Lalu latihan ketiga menampilkan isi tree dengan urutan preorder dan postorder untuk menunjukkan cara penelusuran yang berbeda. Secara keseluruhan, ketiga latihan ini membantu memahami bentuk tree, proses memasukkan data, serta cara membaca isi tree.

E. Referensi

Politeknik Harber. (2020). *Implementasi algoritma binary tree dan sequential searching pada sistem informasi MLM*. Jurnal Informatika, *Artikel 2087*. https://ejournal.poltekharber.ac.id/index.php/informatika/article/download/2087/1547

Modul Praktikum. (2025). *Modul praktikum 4: Binary tree & binary search tree* [PDF]. Scribd. https://id.scribd.com/doc/186761873/Modul-Prakt-4-Binary-Tree-Binary-Search-Tree