LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL 10

TREE

Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

AVL Tree adalah bentuk pengembangan dari *Binary Search Tree* (BST) yang memiliki sifat terurut, di mana semua *Left Child* harus lebih kecil dari *Parent*-nya, dan semua *Right Child* harus lebih besar dari *Parent* serta *Left Child*-nya. Karakteristik utama yang membedakan AVL Tree adalah mekanisme penyeimbangan otomatisnya (*self-balancing*), yang menetapkan aturan ketat bahwa perbedaan tinggi (*height*) atau jumlah maksimum node dalam satu cabang antara *subtree* kiri dan *subtree* kanan maksimal hanyalah 1. Jika operasi penyisipan atau penghapusan data menyebabkan perbedaan tinggi ini dilanggar, struktur pohon akan melakukan rotasi otomatis untuk menyeimbangkan kembali dirinya, sehingga mencegah pohon menjadi miring dan menjamin efisiensi waktu pencarian tetap optimal.

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

"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 << " ";
```

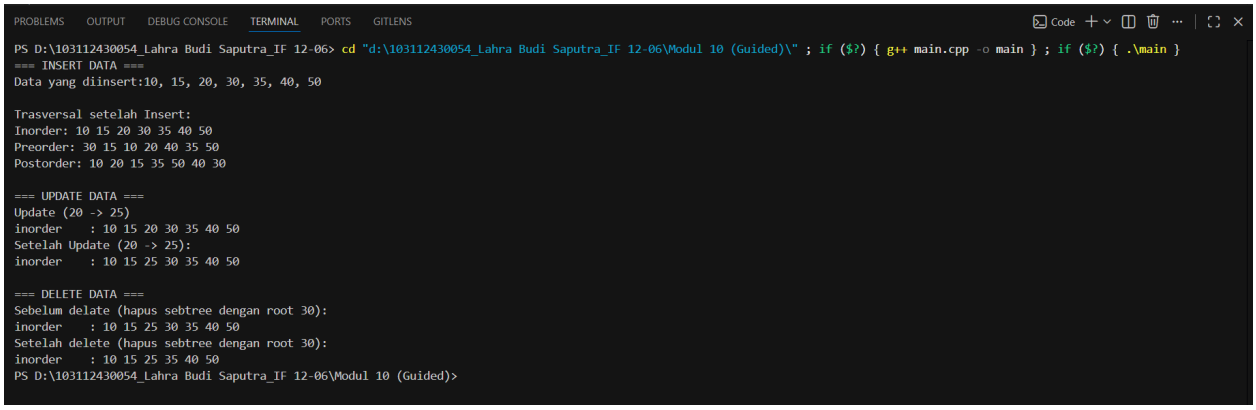
```
}

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 << "\nTrasversal setelah Insert:" <<endl;
   cout << "Inorder: "; tree.inorder();
   cout << "Preorder: "; tree.preorder();
   cout << "Postorder: "; tree.postorder();

   cout << "\n=== UPDATE DATA ===" << endl;
   cout << "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 delate (hapus sebtree dengan root 30):" << endl;
    cout << "inorder    : "; tree.inorder();

    tree.deleteValue(30);
    cout << "Setelah delete (hapus sebtree dengan root 30):" << endl;
    cout << "inorder    : "; tree.inorder();

    return 0;
}
```

Screenshots Output



Deskripsi:

Program di atas merupakan implementasi Abstract Data Type (ADT) dari struktur data AVL Tree (pohon biner penyeimbang otomatis) yang diorganisir ke dalam tiga file terpisah untuk modularitas. File **header tree.h** berfungsi sebagai kontrak antarmuka yang mendefinisikan struktur Node (dilengkapi variabel height untuk melacak ketinggian) serta mendeklarasikan kelas BinaryTree dengan metode publik seperti penyisipan, penghapusan, dan traversal, serta metode privat untuk rotasi dan

pengecekan keseimbangan. Logika inti program terdapat pada **file tree.cpp,** yang menangani algoritma penyeimbangan pohon melalui rotasi (rotateLeft dan rotateRight) setiap kali terjadi operasi insert atau delete yang mengganggu faktor keseimbangan (*balance factor*), serta menangani operasi update dengan cara menghapus nilai lama dan menyisipkan nilai baru. Keseluruhan fungsionalitas ini kemudian diuji dalam **file main.cpp,** yang menjalankan skenario penambahan data urut (10 hingga 50), pengubahan data (update), dan penghapusan data, lalu menampilkan hasilnya ke layar menggunakan metode traversal inorder, preorder, dan postorder untuk membuktikan bahwa pohon tetap terurut dan seimbang.

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

Unguided (SOAL 1)

("bstree.h")

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

typedef int infotype;
typedef struct Node *address;

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

// Fungsi Dasar (Latihan 1)
address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void printInorder(address root);
#endif
```

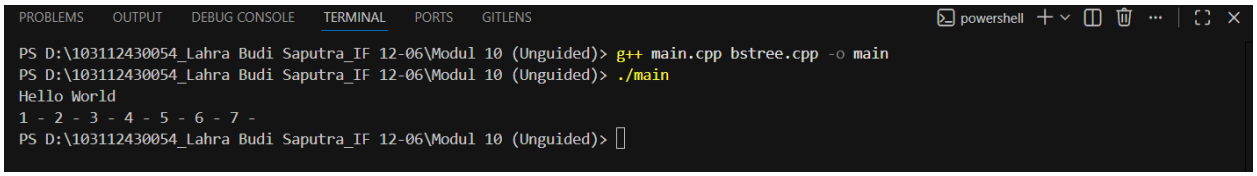
(“bstree.cpp”)

```
#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 || root->info == x) return root;
   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);
   }
}
```

(main.cpp)

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

int main() {
    cout << "Hello World" << endl;
    address root = Nil;

    // Insert data
    insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 6);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 3);
    insertNode(root, 6);
    insertNode(root, 7);
    printInorder(root);
    return 0;
}
```

Screenshots Output:



Deskripsi:

Program ini merupakan implementasi Abstract Data Type (ADT) dari struktur data Binary Search Tree (BST) yang disusun secara modular dalam tiga file terpisah. **File header bstree.h** berfungsi sebagai antarmuka yang mendefinisikan struktur Node (berisi data integer serta pointer ke anak kiri dan kanan) dan mendeklarasikan prototipe fungsi dasar seperti alokasi memori, penyisipan (insertNode), pencarian, dan traversal. Logika operasional dari fungsi-fungsi tersebut dijabarkan dalam **file bstree.cpp**, di mana insertNode menerapkan aturan BST secara rekursif (nilai lebih kecil ke kiri, lebih besar ke kanan) dan printInorder digunakan untuk mengunjungi node dari nilai terkecil ke terbesar. Keseluruhan logika ini diuji melalui **file main.cpp**, yang menyisipkan sekumpulan angka (1, 2, 6, 4, 5, 3, 7) ke dalam tree dan menampilkan hasil pengurutannya ke terminal dengan format traversal In-Order.

Unguided (SOAL 2)

(tree.h)

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

typedef int infotype;
typedef struct Node *address;

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

// Fungsi Dasar (Latihan 1)
address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void printInorder(address root);

// Fungsi Tambahan (Latihan 2)
// Menghitung jumlah node dalam tree
int hitungJumlahNode(address root);

// Menghitung total penjumlahan nilai info dalam tree
int hitungTotalInfo(address root);

// Menghitung kedalaman (tinggi) tree
int hitungKedalaman(address root, int lvl);
#endif
```

(tree.cpp)

```
#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 || root->info == x) return root;
   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);
   }
}

// Implementasi Fungsi Tambahan (Latihan 2 & 3)

// Soal Latihan 2
// Menghitung Jumlah Node
int hitungJumlahNode(address root) {
   if (root == Nil) {
```

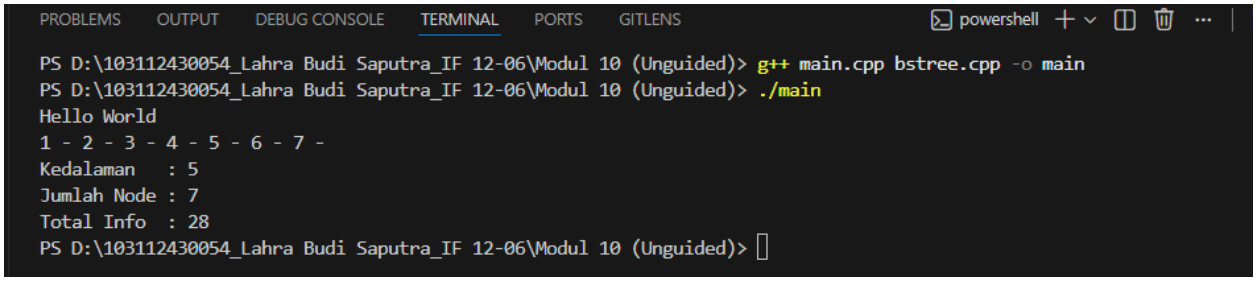
```
    return 0;
  }
  return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
}

// Menghitung Total Info
int hitungTotalInfo(address root) {
  if (root == Nil) {
    return 0;
  }
  return root->info + hitungTotalInfo(root->left) + hitungTotalInfo(root->right);
}

// Menghitung Kedalaman (Max Depth)
int hitungKedalaman(address root, int lvl) {
  if (root == Nil) {
    return lvl;
  }

  int leftDepth = hitungKedalaman(root->left, lvl + 1);
  int rightDepth = hitungKedalaman(root->right, lvl + 1);

  if (leftDepth > rightDepth) return leftDepth;
  else return rightDepth;
}
```

(main.cpp)

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

using namespace std;

int main() {
  cout << "Hello World" << endl;
  address root = Nil;

  // Insert data
  insertNode(root, 1);
  insertNode(root, 2);
  insertNode(root, 6);
  insertNode(root, 4);
  insertNode(root, 5);
```

```
    insertNode(root, 3);
    insertNode(root, 6);
    insertNode(root, 7);
  printInorder(root);
    // Latihan 2: Hitung-hitungan
    cout << "\nKedalaman   : " << hitungKedalaman(root, 0) << endl;
    cout << "Jumlah Node : " << hitungJumlahNode(root) << endl;
    cout << "Total Info  : " << hitungTotalInfo(root) << endl;

    return 0;
}
```

Screenshots Output:



Deskripsi:

Program ini merupakan pengembangan lanjutan dari ADT Binary Search Tree (BST) yang tidak hanya menangani operasi dasar penyisipan dan traversal, tetapi juga menyertakan fungsi analisis properti tree secara rekursif. Dalam **file header tree.h,** dideklarasikan fungsi tambahan untuk menghitung total node, menjumlahkan seluruh nilai data, dan mengukur kedalaman (*height*) pohon. Logika rekursif dari fungsi-fungsi tersebut (hitungJumlahNode, hitungTotalInfo, dan hitungKedalaman) diimplementasikan dalam **tree.cpp** untuk menelusuri setiap cabang pohon hingga ke daun. **File main.cpp** kemudian menguji fungsionalitas ini dengan membangun tree dari sekumpulan data dan menampilkan statistik strukturnya meliputi jumlah node, total nilai, dan kedalaman maksimum tepat setelah hasil traversal In-Order ditampilkan.

Unguided (SOAL 3)

(tree.h)

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

typedef int infotype;
typedef struct Node *address;

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

// Fungsi Dasar (Latihan 1)
address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void printInorder(address root);

// Fungsi Tambahan (Latihan 2 & 3)
// Menghitung jumlah node dalam tree
int hitungJumlahNode(address root);

// Menghitung total penjumlahan nilai info dalam tree
int hitungTotalInfo(address root);

// Menghitung kedalaman (tinggi) tree
int hitungKedalaman(address root, int lvl);
```

```
// Mencetak secara PreOrder dan PostOrder
void printPreOrder(address root);
void printPostOrder(address root);

#endif
```

(tree.cpp)

```
#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 || root->info == x) return root;
   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);
    }
}

// Implementasi Fungsi Tambahan (Latihan 2 & 3)

// Soal Latihan 2
// Menghitung Jumlah Node
int hitungJumlahNode(address root) {
    if (root == Nil) {
        return 0;
    }
    return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
}

// Menghitung Total Info
int hitungTotalInfo(address root) {
    if (root == Nil) {
        return 0;
    }
    return root->info + hitungTotalInfo(root->left) + hitungTotalInfo(root->right);
}

// Menghitung Kedalaman (Max Depth)
int hitungKedalaman(address root, int lvl) {
    if (root == Nil) {
        return lvl;
```

```
    }

    int leftDepth = hitungKedalaman(root->left, lvl + 1);
    int rightDepth = hitungKedalaman(root->right, lvl + 1);

    if (leftDepth > rightDepth) return leftDepth;
    else return rightDepth;
}

// Soal Latihan 3
// PreOrder (Root -> Kiri -> Kanan)
void printPreOrder(address root) {
    if (root != Nil) {
        cout << root->info << " - ";
        printPreOrder(root->left);
        printPreOrder(root->right);
    }
}

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

(main.cpp)

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

using namespace std;
```

```
int main() {
    cout << "Hello World" << endl;
    address root = Nil;

    // Insert data
    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 << "\nInOrder  : ";
    printInorder(root);

    // Latihan 3: Print PreOrder dan PostOrder
    cout << "\nPreOrder : ";
    printPreOrder(root);

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

    // Latihan 2: Hitung-hitungan
    cout << "\nKedalaman   : " << hitungKedalaman(root, 0) << endl;
    cout << "Jumlah Node : " << hitungJumlahNode(root) << endl;
    cout << "Total Info  : " << hitungTotalInfo(root) << endl;

    return 0;
}
```

Screenshots Output:

Deskripsi:

Program ini merupakan implementasi komprehensif dari ADT Binary Search Tree (BST) yang memperluas fungsionalitas dasar dengan kemampuan traversal lengkap dan analisis statistik pohon. **File header tree.h** mendeklarasikan antarmuka untuk tiga metode penelusuran (InOrder, PreOrder, PostOrder) serta fungsi utilitas untuk menghitung jumlah node, akumulasi nilai data, dan kedalaman maksimum pohon. Seluruh logika ini diimplementasikan secara rekursif dalam **tree.cpp,** memungkinkan penelusuran mendalam ke setiap cabang untuk melakukan perhitungan struktural maupun pencetakan data. Kode tersebut kemudian diuji dalam **main.cpp,** yang tidak hanya menyusun data input menjadi struktur pohon, tetapi juga memvalidasi kebenaran logika traversal dan menampilkan laporan statistik (kedalaman, jumlah node, dan total nilai) secara langsung ke terminal.

D. Kesimpulan

praktikum ini berfokus pada pemahaman dan implementasi struktur data **Binary Search Tree (BST)** sebagai bentuk struktur data non-linear yang sangat bergantung pada penerapan fungsi **rekursif**. Mahasiswa mempelajari karakteristik utama BST, yaitu aturan penyisipan data yang ketat di mana *left child* harus selalu bernilai lebih kecil dari *parent* dan *right child* lebih besar dari *parent* , yang bertujuan untuk mengoptimalkan efisiensi pencarian data. Selain operasi dasar tersebut, modul ini juga menekankan pada metode penelusuran (*traversal*) pohon secara *PreOrder*, *InOrder*, dan *PostOrder* , serta kemampuan untuk menganalisis properti pohon melalui perhitungan jumlah node, total nilai, dan kedalaman (*height*) pohon secara algoritmik .

E. Referensi

Wisesty, U. N., Nurrahmi, H., Yunanto, P. E., Rismala, R., & Sthevanie, F. (2025). STRUKTUR DATA MENGGUNAKAN C++. PENERBIT KBM INDONESIA. https://books.google.com/books?hl=en&lr=&id=JwCTEQAAQBAJ&oi=fnd&pg=PA157&dq=tree+bahasa+cpp&ots=Wt5zLl_j_k&sig=NxhKxZJodN_8PxFVplIORN62y78

Anita Sindar, R. M. S. (2019). Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY. https://books.google.com/books?hl=en&lr=&id=GP_ADwAAQBAJ&oi=fnd&pg=PA23&dq=stack+pada+c%2B%2B&ots=86k4Nl2OhV&sig=0KNR8rE2WYaLliEAZmi71x2eU7k

Santoso, L. E. (2004). STANDARD TEMPLATE LIBRARY C++ UNTUK MENGAJARKAN STRUKTUR DATA. Jurnal FASILKOM Vol, 2(2). https://www.academia.edu/download/56411324/standard-template-library-c__-untuk-mengajarkan-struktur-data.pdf