LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL X

TREE

Disusun Oleh :

Nama : Izzah Minkhotun Fannisa

NIM : 103112400198

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

C++ merupakan bahasa pemrograman yang pertama kali dikembangkan oleh Bjarne Stroustrop, seorang ilmuwan komputer asal Denmark. Bahasa pemrograman C++ disempurnakan dari bahasa C yang di temukan oleh Dennis Ritchie pada awal 1970-an. Lalu resmi dirilis pada tahun 1985. C++ cukup banyak digunakan karena kemampuannya dalam memberikan kecepatan eksekusi tinggi serta dukungan terhadap manajemen memori tingkat rendah.

Binary Search Tree (BST) merupakan struktur data berbentuk pohon biner yang dirancang untuk menyimpan dan mengelola data dalam urutan tertentu, sehingga operasi pencarian, penyisipan, dan penghapusan dapat dilakukan dengan efisien. Setiap simpul dalam BST memiliki paling banyak dua anak; nilai-nilai di subpohon kiri selalu lebih kecil daripada nilai simpul induk, sedangkan nilai-nilai di subpohon kanan selalu lebih besar, sehingga traversal inorder menghasilkan urutan nilai yang terurut naik.

BST bersifat rekursif, di mana setiap subpohonnya juga merupakan BST, dengan kompleksitas rata-rata operasi seperti pencarian, penyisipan, dan penghapusan sebesar O(log n). Namun, kompleksitas ini bisa menurun menjadi O(n) jika pohon tidak seimbang. Berkat sifat hierarkis dan terstruktur ini, BST sering digunakan untuk implementasi tabel simbol, indeks dalam database, serta algoritma yang memerlukan akses cepat ke data terurut.

Untuk mengatasi kelemahan pada kasus terburuk BST yang tidak seimbang, dikembangkan varian pohon pencarian seperti self-balancing BST, misalnya AVL dan *Red-Black tree,* yang menjaga ketinggian pohon agar tetap logaritmik.

B. Guide

1. Guide 1

a. Source 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);

    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 << "\nTraversal setelah insert:" << endl;
    cout << "Inoder : "; 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 detele (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;
}
```

b. Screenshot Output

```
=== INSERT DATA ===
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

Traversal setelah insert:
Inoder : 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 detele (hapus subtree dengan root = 30):
Inorder : 10 15 25 30 35 40 50
Setelah delete (subtree root = 30 dihapus):
Inorder : 10 15 25 35 40 50
```

c. Deskripsi Program

Program ini mengimplementasikan Tree menggunakan bahasa C++. Program menyediakan operasi *insert, delete*, dan *update* data, serta menampilkan isi tree melalui traversal *inorder, preorder*, dan *postorder.* Mekanisme rotasi digunakan untuk menjaga keseimbangan tree agar operasi dapat berjalan secara efisien.

C. Unguide

1. Unguide 1

a. Source Code

bstree.h

```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);
void PreOrder(address root);
void PostOrder(address root);

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

#endif
```

bstree.cpp

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

using namespace std;

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

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

```
#include <iostream>
```

```cpp
#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;
}
```

b. Screenshot Output

```
Hello World!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
```

c. Deskripsi Program

Latihan 1 bertujuan untuk mengimplementasikan *Abstract Data Type (ADT) Binary Search Tree (BST)* menggunakan *linked list*. Pada latihan ini dibuat struktur node yang terdiri dari info, *left,* dan *right*, serta fungsi dasar seperti alokasi node, proses penyisipan data ke dalam BST sesuai aturan (nilai lebih kecil ke kiri dan lebih besar ke kanan), dan traversal InOrder. Hasil traversal InOrder digunakan untuk membuktikan bahwa data tersimpan secara terurut di dalam BST.

2. Unguide 2

a. Source Code

bstree.h

```cpp
#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);
void InOrder(address root);

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

#endif
```

bstree.cpp

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

using namespace std;

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);
    }
}

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

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

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

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

    int kiri  = hitungKedalaman(root->left, start + 1);
    int kanan = hitungKedalaman(root->right, start + 1);

    return (kiri > kanan) ? kiri : kanan;
}
```

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);

    InOrder(root);
```

```
    cout << endl;

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

    return 0;
}
```

b. Screenshot Output

```
Hello World!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah node : 7
total : 28
```

c. Deskripsi Program

Latihan 2 merupakan pengembangan dari latihan sebelumnya dengan menambahkan fungsi-fungsi rekursif pada BST. Fungsi yang dibuat meliputi perhitungan jumlah node, total nilai info, dan kedalaman maksimum tree. Seluruh fungsi bekerja secara rekursif dengan menelusuri subtree kiri dan kanan hingga mencapai node kosong. Latihan ini bertujuan untuk memperkuat pemahaman konsep rekursi pada struktur data tree.

3. Unguide 3

a. Source 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);

/* traversal latihan 3 */
```

```
void PreOrder(address root);
void PostOrder(address root);

#endif
```

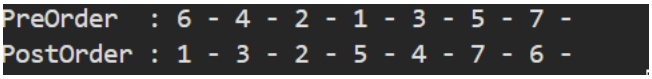
bstree.cpp

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

using namespace std;

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);
    }
}

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

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

using namespace std;

int main() {
    address root = Nil;

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

    cout << "PreOrder  : ";
    PreOrder(root);
    cout << endl;

    cout << "PostOrder : ";
    PostOrder(root);

    return 0;
}
```

b. Screenshot Output



c. Deskripsi Program

Latihan 3 bertujuan untuk memahami proses traversal pada *Binary Search Tree*, khususnya *Pre-Order* dan *Post-Order* traversal. Tree dibentuk sesuai ilustrasi yang diberikan, kemudian dilakukan traversal secara rekursif untuk menghasilkan urutan kunjungan node yang berbeda. Latihan ini menunjukkan perbedaan cara penelusuran tree dan membantu memahami karakteristik masing-masing metode traversal.

D. Kesimpulan

Berdasarkan Latihan 1 sampai dengan Latihan 3, dapat disimpulkan bahwa *Binary Search Tree (BST)* merupakan struktur data non-linear yang efektif untuk menyimpan data secara terurut. Implementasi BST menggunakan linked list mempermudah pengelolaan node dengan relasi parent dan child. Penggunaan fungsi

rekursif sangat sesuai untuk operasi pada tree, seperti penyisipan data, perhitungan jumlah node, total nilai, kedalaman tree, serta proses traversal. Traversal InOrder, PreOrder, dan PostOrder menghasilkan urutan kunjungan node yang berbeda dan menunjukkan cara penelusuran tree sesuai kebutuhan. Dengan demikian, pemahaman BST dan rekursi sangat penting dalam pengolahan struktur data berbasis tree.

E. Referensi

AlgoMap. (n.d.). Binary trees & binary search trees. https://algomap.io/lessons/binary-trees

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

freeCodeCamp. (n.d.). Binary search tree explained with examples. https://www.freecodecamp.org/news/binary-search-trees-bst-explained-with-examples/

GeeksforGeeks. (n.d.). Binary search tree (BST): What is it? https://www.geeksforgeeks.org/what-is-binary-search-tree/

Virginia Tech OpenDSA. (2018). CS 2030 data structures – Binary search trees. https://opendsa.cs.vt.edu/ODSA/Books/uwyo/cosc2030/SECTION-A/html/BST.html

Wikipedia. (n.d.). Binary search tree. https://en.wikipedia.org/wiki/Binary_search_tree

Weiss, M. A. (2014). Data structures and algorithm analysis in C++ (4th ed.). Pearson Education.