LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 10
TREE

Disusun Oleh :
NAMA : M.IRFAN ADIB
NIM : 103112400257

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Binary Search Tree (BST) adalah salah satu struktur data berbentuk pohon di mana setiap node hanya memiliki maksimal dua anak yaitu anak kiri dan anak kanan. BST memiliki aturan utama: nilai yang lebih kecil dari node ditempatkan di sisi kiri, sedangkan nilai yang lebih besar ditempatkan di sisi kanan. Dengan aturan ini pencarian data bisa dilakukan lebih cepat karena kita tidak perlu memeriksa semua data satu per satu, cukup mengikuti arah kiri atau kanan seperti menelusuri jalur. Selain pencarian BST juga memungkinkan proses seperti penambahan data, penghapusan data, dan penelusuran data dilakukan dengan lebih teratur. Tiga jenis traversal umum adalah in-order, pre-order, dan post-order. Dalam penggunaannya, BST juga dapat dihitung kedalamannya, jumlah node yang dimiliki, serta total nilai data yang ada di dalamnya.

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

bstree.h

```
#ifndef TREE_H
#define TREE_H

struct Node
{
    int data;
    Node* left;
    Node* 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 oldValue, int newValue);

    void inorder();
    void preorder();
    void postorder();

};
#endif
```

bstree.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 << "Inorder : "; tree.inorder();
    cout << "Preorder : "; tree.inorder();
    cout << "Posorder : "; tree.inorder();

    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 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;

}
```

Screenshots Output

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

Traversal setelah insert:
Inorder : 10 15 20 30 35 40 50
Preorder : 10 15 20 30 35 40 50
Posorder : 10 15 20 30 35 40 50

=== 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 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
```

Deskripsi: Program C++ ini adalah tempat untuk mencoba-coba bagaimana sebuah daftar data yang tersusun seperti pohon bekerja. Pertama, program memasukkan beberapa angka (10, 15, 20, 30, 35, 40, 50) ke dalam pohon. Setelah itu, program mengecek daftar data untuk memastikan semua sudah masuk. Kemudian, program menguji fungsi mengubah data dengan mengganti angka 20 menjadi 25. Terakhir, program menguji fungsi menghapus data dengan menghilangkan angka 30 dan semua data yang berada di bawahnya dalam struktur pohon tersebut.

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

Unguided 1

bstree.h

```
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

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

```
};

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);

void insertNode(address &root, infotype x);

address findNode(infotype x, address root);

void InOrder(address root);

#endif
```

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);
    }
}
```

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, 7);

    InOrder(root);

    return 0;
}
```

Screenshots Output

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

Deskripsi: Program C++ sederhana ini berfungsi untuk membangun dan menampilkan daftar angka yang disusun secara khusus, yaitu menggunakan cara kerja Pohon Pencarian Biner (BST). Pertama, program memulai pohonnya dari nol (kosong), lalu memasukkan tujuh angka (1, 2, 6, 4, 5, 3, 7) satu per satu ke dalam pohon tersebut. Setelah semua angka dimasukkan, program akan menampilkan angka-angka tersebut di layar sesuai urutan yang sudah diatur oleh pohon.

Unguided 2

bstree.h

```h
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

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

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);

void insertNode(address &root, infotype x);

address findNode(infotype x, address root);

void InOrder(address root);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);
```

```
int hitungKedalaman(address root);

#endif
```

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);
    }
}

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


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


int hitungKedalaman(address root) {
    if (root == Nil) {
        return -1;
    } else {

        int leftDepth = hitungKedalaman(root->left);
        int rightDepth = hitungKedalaman(root->right);


        return 1 + max(leftDepth, rightDepth);
    }
}
```

main.cpp

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

```cpp
#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, 7);

    InOrder(root);
    cout << endl;


    int kedalaman = hitungKedalaman(root) + 2;
    cout << "kedalaman : " << kedalaman << endl;

    int jumlahNode = hitungJumlahNode(root);
    cout << "jumlah node : " << jumlahNode << endl;

    int totalInfo = hitungTotalInfo(root);
    cout << "total : " << totalInfo << endl;

    return 0;
}
```

Screenshots Output

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

Deskripsi: Program ini adalah tempat mencoba-coba untuk sebuah daftar angka yang disusun seperti pohon. Program memulai pohonnya dari nol, lalu memasukkan tujuh angka 1, 2, 6, 4, 5, 3, 7 ke dalam susunan tersebut. Setelah semua angka masuk, program menampilkan angka-angka tersebut secara berurutan dari kecil ke besar. Terakhir, program menghitung dan menunjukkan tiga hal penting tentang susunan pohon itu: kedalaman seberapa tinggi atau banyak tingkat susunan data itu, jumlah node berapa total angka yang ada, dan total berapa hasil penjumlahan dari semua angka yang dimasukkan.

Unguided 3

bstree.h

```h
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

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

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);

void insertNode(address &root, infotype x);

address findNode(infotype x, address root);

void InOrder(address root);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root);

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;
    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);
    }
}
```

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

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

int hitungKedalaman(address root) {
    if (root == Nil) {
        return -1;
    } else {

        int leftDepth = hitungKedalaman(root->left);
        int rightDepth = hitungKedalaman(root->right);


        return 1 + max(leftDepth, rightDepth);
    }
}

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>
#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, 7);

    InOrder(root);
    cout << endl;

    int kedalaman = hitungKedalaman(root) + 1;
    cout << "kedalaman : " << kedalaman << endl;

    int jumlahNode = hitungJumlahNode(root);
    cout << "jumlah node : " << jumlahNode << endl;

    int totalInfo = hitungTotalInfo(root);
    cout << "total : " << totalInfo << endl;

    cout << "pre-order : ";
    PreOrder(root);
    cout << endl;

    cout << "post-order : ";
    PostOrder(root);
```

```
    cout << endl;

    return 0;
}
```

Screenshots Output

Deskripsi: Program C++ ini adalah tempat untuk menguji dan menampilkan banyak informasi dari daftar angka yang disusun seperti pohon. Program memulai pohonnya dari nol, lalu memasukkan tujuh angka 1, 2, 6, 4, 5, 3, 7 ke dalam susunan tersebut. Setelah semua angka masuk, program tidak hanya menampilkan semua angka secara berurutan dari kecil ke besar InOrder, tetapi juga menampilkan urutan PreOrder dan PostOrder yang berbeda. Selain itu, program menghitung dan menunjukkan tiga hal penting tentang susunan pohon itu: kedalaman seberapa tinggi atau banyak tingkat susunan data itu, jumlah node berapa total angka yang ada, dan total berapa hasil penjumlahan dari semua angka yang dimasukkan.

D. Kesimpulan

Program ini adalah tempat mencoba-coba untuk sebuah daftar angka yang disusun seperti pohon. Program memulai pohonnya dari nol, lalu memasukkan tujuh angka 1, 2, 6, 4, 5, 3, 7 ke dalam susunan tersebut. Setelah semua angka masuk, program mengecek semua angka dengan menampilkan tiga jenis urutan InOrder, PreOrder, dan PostOrder yang berbeda. Selain itu, program menghitung dan menunjukkan tiga hal penting tentang susunan pohon itu: kedalaman seberapa tinggi atau banyak tingkat susunan data itu, jumlah node berapa total angka yang ada, dan total berapa hasil penjumlahan dari semua angka yang dimasukkan.

E. Referensi

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms* (2nd ed.). MIT Press. ISBN 0-262-03293-7.

Paul E. Black, "red-black tree", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 12 November 2019. (accessed May 19 2022) from: https://www.nist.gov/dads/HTML/redblack.html

Thareja, Reema (13 October 2018). "Hashing and Collision". *Data Structures Using C* (2 ed.). Oxford University Press. ISBN 9780198099307.