LAPORAN

MODUL 10

TREE

Disusun Oleh :

NAMA :

Devi Nurliana

NIM :

103112400144

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. DASAR TEORI

Tree adalah bentuk struktur data yang tidak linear, terdiri dari sekumpulan node yang saling terhubung dan membentuk hubungan berjenjang. Setiap *tree* memiliki satu node utama yang disebut *root*, sedangkan node lainnya bisa memiliki *parent, leaf, child*. Struktur pohon sering digunakan untuk menyajikan data dalam bentuk hierarki, seperti folder dalam komputer, keturunan dalam keluarga, dan ekspresi matematika. *Binary Tree* adalah jenis *tree*di mana setiap node hanya bisa memiliki maksimal dua *child*, yaitu *child* kiri dan *child* kanan. Salah satu contoh *Binary Tree* adalah *Binary Search Tree* (BST), yang memiliki aturan bahwa nilai di *child* kiri lebih kecil dari nilai *parent*, sedangkan nilai di *child* kanan lebih besar. Operasi dasar pada *tree* meliputi menambahkan, menghapus, mengubah data, serta menelusuri seluruh node dengan metode *inorder, preorder, dan postorder.*

B. GUIDED

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 ini berfungsi untuk mengimplementasikan struktur data *Binary Tree* seimbang (*AVL Tree*) menggunakan bahasa C++. Program mendukung operasi penyisipan data, penghapusan data, pembaruan data, serta penelusuran tree menggunakan metode *inorder, preorder, dan postorder.*

1. Pendefinisian struktur dan ADT
   Program mendefinisikan struktur Node yang berisi data, pointer ke child kiri dan kanan, serta tinggi node. Selain itu, dibuat kelas *BinaryTree* yang memiliki *pointer root* dan deklarasi fungsi-fungsi untuk mengelola *tree* seperti *insert, delete, update*, dan traversal.

2. Implementasi mekanisme tree
   Pada proses *insert* dan *delete*, program menghitung tinggi node dan balance factor untuk menjaga keseimbangan tree. Jika tree tidak seimbang, dilakukan rotasi ke kiri atau ke kanan sesuai kondisi sehingga struktur *tree* tetap seimbang.

3. Proses utama program
   Pada fungsi main, program membuat objek *BinaryTree* lalu melakukan penyisipan beberapa data. Program kemudian menampilkan hasil traversal, melakukan *update* data, serta menghapus data tertentu, dan kembali menampilkan hasil traversal untuk melihat perubahan pada *tree.*

4. Selesai
   Setelah seluruh proses *insert, update, delete*, dan traversal selesai dijalankan, program berakhir dan mengembalikan nilai 0.

C. UNGUIDED

1. Unguided 1
   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);
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

```
#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) {
```

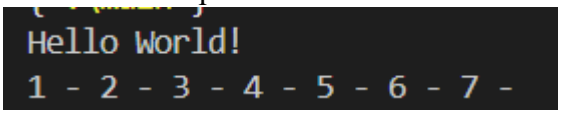
```
            root = alokasi(x);
        } else if (x < root->info) {
            insertNode(root->left, x);
        } else if (x > root->info) {
            insertNode(root->right, x);
        }
    }

    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>
    #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



c. Deskripsi

Program ini berfungsi untuk mengimplementasikan struktur data *Binary Search* Tree (BST) menggunakan bahasa C++. Program menyediakan operasi alokasi node, penyisipan data, pencarian data, serta penelusuran *tree* menggunakan metode *inorder, preorder, dan postorder.*

1. Pendefinisian struktur dan ADT

   Program mendefinisikan struktur Node yang menyimpan data bertipe integer serta pointer ke *child* kiri dan kanan. Selain itu, dideklarasikan fungsi-fungsi dasar untuk mengelola BST seperti alokasi node, penyisipan data, pencarian data, dan traversal *tree.*

2. Implementasi mekanisme BST

   Pada proses penyisipan data, program menerapkan aturan BST, yaitu data yang lebih kecil disimpan di subtree kiri dan data yang lebih besar disimpan di *subtree* kanan. Proses pencarian data dilakukan dengan menelusuri *tree* berdasarkan perbandingan nilai data.

3. Proses utama program

   Pada fungsi main, program membuat tree kosong kemudian menyisipkan beberapa data ke dalam BST. Setelah itu, program menampilkan hasil traversal inorder untuk menampilkan data dalam urutan terurut.

4. Selesai

   Setelah seluruh proses penyisipan dan penelusuran selesai dijalankan, program berakhir dan mengembalikan nilai 0.

2. Unguided 2

   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);
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);
   }
}
```

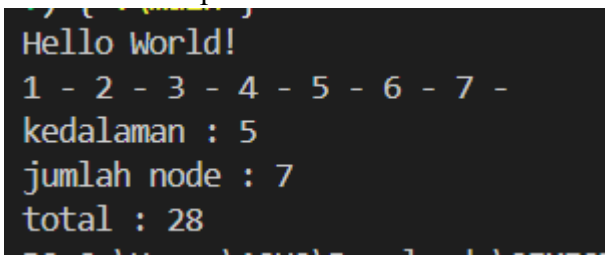
```
void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}

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



c. Deskripsi

Program ini berfungsi untuk mengimplementasikan *Binary Search Tree* (BST) serta menghitung jumlah node, total nilai node, dan kedalaman tree, dengan langkah kerja sebagai berikut:

1. Pendefinisian struktur dan ADT
   Program mendefinisikan struktur Node yang berisi data integer serta pointer ke *child* kiri dan kanan. Selain itu, didefinisikan fungsi untuk alokasi node, penyisipan data ke dalam BST, traversal *inorder*, serta fungsi perhitungan jumlah node, total nilai data, dan kedalaman *tree.*

2. Proses penyisipan data ke BST
   Program menggunakan prosedur insertNode untuk memasukkan data ke dalam *tree.* Data yang lebih kecil disimpan di subtree kiri, sedangkan data yang lebih besar disimpan di subtree kanan sesuai aturan *Binary Search Tree.*

3. Proses traversal dan perhitungan
   Program menampilkan isi *tree* menggunakan traversal *inorder* sehingga data ditampilkan secara terurut. Selanjutnya, program menghitung kedalaman *tree*, jumlah node, dan total nilai seluruh node menggunakan fungsi rekursif.

4. Selesai
   Program menampilkan seluruh hasil ke layar, kemudian program berakhir dan mengembalikan nilai 0.

3. Unguided 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

```
#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 ini berfungsi untuk mengimplementasikan *Binary Search Tree* (BST) serta menampilkan traversal PreOrder dan PostOrder, dengan langkah kerja sebagai berikut:

1. Pendefinisian struktur dan ADT
   Program mendefinisikan struktur Node yang terdiri dari data bertipe integer serta pointer ke child kiri dan kanan. Selain itu, dideklarasikan fungsi alokasi node, penyisipan data ke dalam BST, dan traversal PreOrder serta PostOrder.

2. Proses penyisipan data
   Program menggunakan prosedur insertNode untuk memasukkan data ke dalam BST. Data yang lebih kecil dari node akan disimpan di *subtree* kiri, sedangkan data yang lebih besar disimpan di subtree kanan sesuai aturan *Binary Search Tree.*

3. Traversal *tree*
   Program menampilkan isi BST menggunakan traversal PreOrder dan PostOrder. PreOrder menampilkan data dengan urutan *root*–kiri–kanan, sedangkan PostOrder menampilkan data dengan urutan kiri–kanan–*root.*

4. Selesai
   Setelah hasil traversal ditampilkan ke layar, program berakhir dan mengembalikan nilai 0.

D. KESIMPULAN

Berdasarkan penerapan *Binary Search Tree* pada tiga latihan yang sudah dilakukan, dapat disimpulkan bahwa BST mampu menyimpan dan mengelola data secara teratur berdasarkan aturan nilai di sebelah kiri lebih kecil dan di sebelah kanan lebih besar. Pada BST pertama, operasi dasar seperti menyisipkan data dan traversal InOrder digunakan untuk menampilkan data secara terurut serta menghitung jumlah node, total nilai, dan tinggi pohon.

Pada BST kedua dan ketiga, traversal PreOrder dan PostOrder digunakan untuk melihat urutan tampilan data dari sudut pandang yang berbeda. Traversal PreOrder menampilkan data dengan urutan root terlebih dahulu, sedangkan traversal PostOrder menampilkan data setelah seluruh *subtree* selesai diproses. Perbedaan urutan hasil traversal tersebut menunjukkan bahwa satu struktur BST bisa menghasilkan keluaran yang berbeda tergantung metode yang digunakan, sehingga BST sangat fleksibel untuk berbagai keperluan pengolahan data.

E. REFERNSI

- Modul Praktikum Struktur Data. *Modul 10: Tree(Bagian Kesatu)*. Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Telkom, Tahun 2025.
- Buku “Data Structures and Algorithms in C++” – Michael T. Goodrich
- Buku “Introduction to Algorithms” – Thomas H. Cormen et al.