LAPORAN PRAKTIKUM
Modul 09
“TREE”

Disusun Oleh:
Aji Prasetyo Nugroho - 2211104049
S1SE-07-2

Assisten Praktikum :
Aldi Putra
Andini Nur Hidayah

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

A. Tujuan

1. Memahami konsep penggunaan fungsi rekursif.
2. Mengimplementasikan bentuk-bentuk fungsi rekursif.
3. Mengaplikasikan struktur data *tree* dalam sebuah kasus pemrograman.
4. Mengimplementasikan struktur data *tree*, khususnya *Binary Tree*.
5. Mengimplementasikan struktur data *tree*, khususnya *Binary Tree*.

B. Landasan Teori

1. Definisi Tree

   Tree adalah struktur data hierarkis yang terdiri dari simpul-simpul (nodes), di mana setiap simpul terhubung dengan simpul lainnya melalui hubungan yang disebut edge. Tree digunakan untuk merepresentasikan data dengan hubungan hierarki seperti organisasi, direktori file, atau ekspresi matematika.

2. Komponen Tree

   - Node: Elemen dasar dari tree yang menyimpan data.
   - Root: Node paling atas dari tree, yang tidak memiliki parent.
   - Child: Node yang berada satu tingkat di bawah node tertentu, disebut anak dari node tersebut.
   - Parent: Node yang memiliki child disebut induk dari child tersebut.
   - Leaf: Node yang tidak memiliki child, disebut simpul daun.
   - Edge: Garis penghubung antara dua node.
   - Subtree: Tree kecil yang merupakan bagian dari tree besar.
   - Height: Panjang jalur terpanjang dari root ke leaf.
   - Depth: Jarak dari root ke node tertentu.

3. Jenis-Jenis Tree

   - Binary Tree: Setiap node memiliki maksimal dua child (anak kiri dan anak kanan).
   - Binary Search Tree (BST): Binary tree di mana semua node di subtree kiri lebih kecil dari node induk, dan semua node di subtree kanan lebih besar.
   - Full Binary Tree: Semua node memiliki 0 atau 2 child.
   - Complete Binary Tree: Semua level kecuali level terakhir penuh, dan node di level terakhir terletak sejauh mungkin ke kiri.

- Balanced Binary Tree: Perbedaan tinggi subtree kiri dan kanan tidak lebih dari 1.
- N-ary Tree: Setiap node dapat memiliki maksimal N child.
- Heap Tree: Tree khusus yang memenuhi properti heap, di mana nilai pada node induk selalu lebih besar atau lebih kecil dari nilai child-nya.

4. Operasi pada Tree
   - Traversal: Mengunjungi semua node pada tree sesuai dengan aturan tertentu:
   - Pre-order (Node -> Left -> Right).
   - In-order (Left -> Node -> Right).
   - Post-order (Left -> Right -> Node).
   - Insertion: Menambahkan node ke tree.
   - Deletion: Menghapus node dari tree, termasuk mengatur ulang subtree.
   - Searching: Mencari node dengan nilai tertentu dalam tree.
5. Kelebihan dan Kekurangan Tree
   1) Kelebihan:
      - Struktur data yang efisien untuk pencarian dan penyimpanan data hierarkis.
      - Operasi seperti pencarian, penyisipan, dan penghapusan dapat dilakukan dengan kompleksitas waktu logaritmik dalam kasus terbaik (pada tree seimbang).
   2) Kekurangan:
      - Tidak efisien untuk data yang tidak hierarkis.
      - Tree yang tidak seimbang dapat mengakibatkan kompleksitas waktu yang buruk (O(n)).

C. Guided

Guided

Source Code :

```
#include <iostream>
using namespace std;

// Struktur data Pohon
struct Pohon {
    char data; // data dari node
    Pohon *left, *right; // pointer ke child kiri dan kanan
    Pohon *parent; // pointer ke parent
};

// Pointer ke root pohon dan node sementara untuk operasi insert
Pohon *root, *baru;

// Fungsi untuk inisialisasi pohon
void init() {
    root = NULL;
}

// Fungsi untuk mengecek apakah pohon kosong
bool isEmpty() {
    return root == NULL;
}

// Fungsi untuk membuat root pohon
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL}; // Membuat root baru
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

// Fungsi untuk menambahkan child kiri pada node tertentu
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) { // Jika node sudah punya child kiri
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    }

    // Membuat node baru untuk child kiri
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru; // Menambahkan child kiri
```

```
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data << endl;
    return baru;
}

// Fungsi untuk menambahkan child kanan pada node tertentu
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) { // Jika node sudah punya child kanan
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }

    // Membuat node baru untuk child kanan
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru; // Menambahkan child kanan
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " <<
node->data << endl;
    return baru;
}

// Fungsi untuk mencari node berdasarkan data
void find(char data, Pohon *node) {
    if (!node) return; // Jika node NULL, hentikan pencarian

    // Jika data ditemukan
    if (node->data == data) {
        cout << "\nNode ditemukan: " << data << endl;
    }

    // Rekursif untuk mencari di child kiri dan kanan
    find(data, node->left);
    find(data, node->right);
}

// Traversal pohon secara Pre-order (Root, Left, Right)
void preOrder(Pohon *node) {
    if (!node) return; // Jika node NULL, hentikan traversal
    cout << node->data << " "; // Cetak data node
    preOrder(node->left); // Traversal child kiri
    preOrder(node->right); // Traversal child kanan
}

// Traversal pohon secara In-order (Left, Root, Right)
void inOrder(Pohon *node) {
    if (!node) return; // Jika node NULL, hentikan traversal
    inOrder(node->left); // Traversal child kiri
    cout << node->data << " "; // Cetak data node
    inOrder(node->right); // Traversal child kanan
```

```
}

// Traversal pohon secara Post-order (Left, Right, Root)
void postOrder(Pohon *node) {
    if (!node) return; // Jika node NULL, hentikan traversal
    postOrder(node->left); // Traversal child kiri
    postOrder(node->right); // Traversal child kanan
    cout << node->data << " "; // Cetak data node
}

// Fungsi untuk menghapus node tertentu dalam pohon
Pohon* deleteNode(Pohon *node, char data) {
    if (!node) return NULL; // Jika node NULL, kembalikan NULL

    // Jika data lebih kecil dari data node, cari di subtree kiri
    if (data < node->data) {
        node->left = deleteNode(node->left, data);
    }
    // Jika data lebih besar dari data node, cari di subtree kanan
    else if (data > node->data) {
        node->right = deleteNode(node->right, data);
    }
    // Jika data sama dengan data node, hapus node tersebut
    else {
        if (!node->left) { // Jika tidak ada child kiri
            Pohon *temp = node->right; // Ambil child kanan
            delete node; // Hapus node
            return temp;
        } else if (!node->right) { // Jika tidak ada child kanan
            Pohon *temp = node->left; // Ambil child kiri
            delete node; // Hapus node
            return temp;
        }

        // Jika node memiliki dua child, cari successor
        Pohon *successor = node->right;
        while (successor->left) successor = successor->left; // Cari node
paling kiri di subtree kanan
        node->data = successor->data; // Ganti data node dengan successor
        node->right = deleteNode(node->right, successor->data); // Hapus
successor
    }
    return node;
}

// Fungsi untuk mencari node paling kiri dalam subtree
Pohon* mostLeft(Pohon *node) {
    if (!node) return NULL; // Jika node NULL
```

```
    while (node->left) node = node->left; // Cari node paling kiri
    return node;
}

// Fungsi untuk mencari node paling kanan dalam subtree
Pohon* mostRight(Pohon *node) {
    if (!node) return NULL; // Jika node NULL
    while (node->right) node = node->right; // Cari node paling kanan
    return node;
}

int main() {
    init(); // Inisialisasi pohon

    // Membuat root dan menambah node pada pohon
    buatNode('F');
    insertLeft('B', root);
    insertRight('G', root);
    insertLeft('A', root->left);
    insertRight('D', root->left);
    insertLeft('C', root->left->right);
    insertRight('E', root->left->right);

    // Traversal pohon
    cout << "\nPre-order Traversal: ";
    preOrder(root);
    cout << "\nIn-order Traversal: ";
    inOrder(root);
    cout << "\nPost-order Traversal: ";
    postOrder(root);

    // Menghapus node D dan melakukan traversal kembali
    cout << "\nMenghapus node D.";
    root = deleteNode(root, 'D');
    cout << "\nIn-order Traversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}
```

Output :

```
Node F berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri F

Node G berhasil ditambahkan ke child kanan F

Node A berhasil ditambahkan ke child kiri B

Node D berhasil ditambahkan ke child kanan B

Node C berhasil ditambahkan ke child kiri D

Node E berhasil ditambahkan ke child kanan D

Pre-order Traversal: F B A D C E G
In-order Traversal: A B C D E F G
Post-order Traversal: A C E D B G F
Menghapus node D.
In-order Traversal setelah penghapusan: A B C E F G
PS D:\Praktikum STD_2211104049>
```

D. Unguided

1. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinputkan!
2. Buatlah fungsi rekursif is_valid_bst(node, min_val, max_val) untuk memeriksa apakah suatu pohon memenuhi properti Binary Search Tree. Uji fungsi ini pada berbagai pohon, baik yang valid maupun tidak valid sebagai BST.
3. Buatlah fungsi rekursif cari_simpul_daun(node) untuk menghitung jumlah simpul daun dalam Binary Tree. Simpul daun adalah node yang tidak memiliki anak kiri maupun kanan.

Source Code :

unguidedM10-11.cpp

```
#include <iostream>
#include <climits>
using namespace std;

// Struktur node pohon
struct Pohon {
    char data;
    Pohon *left, *right, *parent;

    // Konstruktor
    Pohon(char x) : data(x), left(NULL), right(NULL), parent(NULL) {}
};

class BinaryTree {
private:
    Pohon *root;

    // Fungsi rekursif untuk mencari node
    Pohon* findNode(Pohon* node, char data) {
        if (!node) return NULL;

        if (node->data == data) return node;

        Pohon* leftResult = findNode(node->left, data);
        if (leftResult) return leftResult;

        Pohon* rightResult = findNode(node->right, data);
        if (rightResult) return rightResult;
```

```
        return NULL;
    }

public:
    BinaryTree() : root(NULL) {}

    // Fungsi untuk memeriksa apakah pohon kosong
    bool isEmpty() {
        return root == NULL;
    }

    // Membuat node root
    void buatNode(char data) {
        if (isEmpty()) {
            root = new Pohon(data);
            cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
        } else {
            cout << "\nPohon sudah dibuat. Gunakan menu insert." << endl;
        }
    }

    // Menambahkan node kiri
    Pohon* insertLeft(char data, Pohon *node) {
        if (!node) {
            cout << "\nNode induk tidak ditemukan!" << endl;
            return NULL;
        }

        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah ada child kiri!" <<
endl;
            return NULL;
        }

        Pohon* baru = new Pohon(data);
        baru->parent = node;
        node->left = baru;
        cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data << endl;
        return baru;
    }

    // Menambahkan node kanan
    Pohon* insertRight(char data, Pohon *node) {
        if (!node) {
            cout << "\nNode induk tidak ditemukan!" << endl;
```

```
            return NULL;
        }

        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah ada child kanan!" <<
endl;
            return NULL;
        }

        Pohon* baru = new Pohon(data);
        baru->parent = node;
        node->right = baru;
        cout << "\nNode " << data << " berhasil ditambahkan ke child kanan "
<< node->data << endl;
        return baru;
    }

    // Menampilkan child dari suatu node
    void tampilkanChild(Pohon* node) {
        if (!node) {
            cout << "Node tidak ditemukan!" << endl;
            return;
        }

        cout << "Node: " << node->data << endl;
        cout << "Child Kiri: " << (node->left ? node->left->data : '-') <<
endl;
        cout << "Child Kanan: " << (node->right ? node->right->data : '-') <<
endl;
    }

    // Menampilkan descendant dari suatu node
    void tampilkanDescendant(Pohon* node) {
        if (!node) {
            cout << "Node tidak ditemukan!" << endl;
            return;
        }

        cout << "Descendant dari " << node->data << ": ";

        if (node->left || node->right) {
            if (node->left) {
                cout << "Kiri: ";
                Pohon* current = node->left;
                while (current) {
                    cout << current->data << " ";
                    current = current->left;
                }
```

```
            }

            if (node->right) {
                cout << "Kanan: ";
                Pohon* current = node->right;
                while (current) {
                    cout << current->data << " ";
                    current = current->right;
                }
            }
            cout << endl;
        } else {
            cout << "Tidak memiliki descendant." << endl;
        }
    }

    // Fungsi untuk mendapatkan root
    Pohon* getRoot() {
        return root;
    }

    // Fungsi untuk mencari node
    Pohon* findNodeWrapper(char data) {
        return findNode(root, data);
    }

    // Fungsi untuk memeriksa validitas BST
    bool is_valid_bst(Pohon* node, long min_val = LONG_MIN, long max_val =
LONG_MAX) {
        if (node == NULL) return true;

        if (node->data <= min_val || node->data >= max_val)
            return false;

        return is_valid_bst(node->left, min_val, node->data) &&
               is_valid_bst(node->right, node->data, max_val);
    }

    // Fungsi untuk menghitung simpul daun
    int cari_simpul_daun(Pohon* node) {
        if (node == NULL) return 0;

        if (node->left == NULL && node->right == NULL)
            return 1;

        return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
    }
};
```

```
// Fungsi menu utama
void menu(BinaryTree& pohon) {
    int pilihan;
    char data, induk;
    Pohon* node;

    do {
        cout << "\n--- MENU POHON BINER ---" << endl;
        cout << "1. Buat Root" << endl;
        cout << "2. Tambah Node Kiri" << endl;
        cout << "3. Tambah Node Kanan" << endl;
        cout << "4. Tampilkan Child" << endl;
        cout << "5. Tampilkan Descendant" << endl;
        cout << "6. Periksa Validitas BST" << endl;
        cout << "7. Hitung Simpul Daun" << endl;
        cout << "8. Keluar" << endl;
        cout << "Pilih menu: ";
        cin >> pilihan;

        switch(pilihan) {
            case 1:
                if (!pohon.isEmpty()) {
                    cout << "Pohon sudah memiliki root!" << endl;
                    break;
                }
                cout << "Masukkan data root: ";
                cin >> data;
                pohon.buatNode(data);
                break;

            case 2:
                if (pohon.isEmpty()) {
                    cout << "Buat root terlebih dahulu!" << endl;
                    break;
                }
                cout << "Masukkan data node induk: ";
                cin >> induk;
                cout << "Masukkan data node kiri: ";
                cin >> data;
                node = pohon.findNodeWrapper(induk);
                pohon.insertLeft(data, node);
                break;

            case 3:
                if (pohon.isEmpty()) {
                    cout << "Buat root terlebih dahulu!" << endl;
                    break;
```

```
                }
                cout << "Masukkan data node induk: ";
                cin >> induk;
                cout << "Masukkan data node kanan: ";
                cin >> data;
                node = pohon.findNodeWrapper(induk);
                pohon.insertRight(data, node);
                break;

            case 4:
                if (pohon.isEmpty()) {
                    cout << "Pohon kosong!" << endl;
                    break;
                }
                cout << "Masukkan data node: ";
                cin >> data;
                node = pohon.findNodeWrapper(data);
                pohon.tampilkanChild(node);
                break;

            case 5:
                if (pohon.isEmpty()) {
                    cout << "Pohon kosong!" << endl;
                    break;
                }
                cout << "Masukkan data node: ";
                cin >> data;
                node = pohon.findNodeWrapper(data);
                pohon.tampilkanDescendant(node);
                break;

            case 6:
                if (pohon.isEmpty()) {
                    cout << "Pohon kosong!" << endl;
                    break;
                }
                cout << "Validitas BST: " <<
(pohon.is_valid_bst(pohon.getRoot()) ? "Ya" : "Tidak") << endl;
                break;

            case 7:
                if (pohon.isEmpty()) {
                    cout << "Pohon kosong!" << endl;
                    break;
                }
                cout << "Jumlah Simpul Daun: " <<
pohon.cari_simpul_daun(pohon.getRoot()) << endl;
                break;
```

```
            case 8:
                cout << "Keluar dari program." << endl;
                break;

            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 8);
}

int main() {
    BinaryTree pohon;
    menu(pohon);
    return 0;
}
```

Output :

1) Buat Root

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 1
Masukkan data root: A

Node A berhasil dibuat menjadi root.
```

2) Tambah Node Kiri

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 2
Masukkan data node induk: A
Masukkan data node kiri: B

Node B berhasil ditambahkan ke child kiri A
```

3) Tambah Node Kanan

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 3
Masukkan data node induk: A
Masukkan data node kanan: C

Node C berhasil ditambahkan ke child kanan A
```

4) Tampilkan Child

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 4
Masukkan data node: A
Node: A
Child Kiri: B
Child Kanan: C
```

5) Periksa Descendant

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 5
Masukkan data node: A
Descendant dari A: Kiri: B Kanan: C
```

6) Periksa Validitas BST

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 6
Validitas BST: Tidak
```

7) Hitung Simpul Daun

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 7
Jumlah Simpul Daun: 2
```

8) Keluar

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa Validitas BST
7. Hitung Simpul Daun
8. Keluar
Pilih menu: 8
Keluar dari program.
PS D:\Praktikum STD_2211104049>
```