Tugas Proceeding

Modul 9 Struktur Data

TREE

**Disusun Oleh:**

**Christian Felix Saliman Sugiono (2311104031)**

**S1SE0701**

**Dosen:**

**Yudha Islami Sulistya**

**Program Studi S1 Software Engineering**

**Fakultas Informatika**

**Telkom University**

**Purwokerto**

**2024**

# Tugas Pendahuluan Modul 8

# STRUKTUR DATA - Ganjil 2024/2025

# "TREE"

## A. Ketentuan Tugas Pendahuluan

1. Tugas Pendahuluan dikerjakan secara Individu.
2. TP ini bersifat WAJIB, tidak mengerjakan = PENGURANGAN POIN JURNAL / TES ASESMEN.
3. Hanya MENGUMPULKAN tetapi TIDAK MENGERJAKAN = PENGURANGAN POIN JURNAL / TES ASESMEN.
4. Deadline pengumpulan TP Modul 4 adalah Senin, 9 Oktober 2023 pukul 06.00 WIB.
5. TIDAK ADA TOLERANSI KETERLAMBATAN, TERLAMBAT ATAU TIDAK MENGUMPULKAN TP MAKA DIANGGAP TIDAK MENGERJAKAN.
6. DILARANG PLAGIAT (PLAGIAT = E).
7. Kerjakan TP dengan jelas agar dapat dimengerti.
8. File diupload di LMS menggunakan format PDF dengan ketentuan: TP_MOD_[XX]_NIM_NAMA.pdf

CP (WA):

- Andini (082243700965)
- Imelda (082135374187)

**SELAMAT MENGERJAKAN^^**

Unguided Modul 9

Implementasi Tree pada C++

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

struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root = NULL;

void init()
{
    root = NULL;
}

bool isEmpty()
{
    return root == NULL;
}
```

Sebagai awalan kita akan membuat struct Pohon yang akan berisikan 3 tipe data yang dimana tipe datanya ada anak kiri dan kanan dari parent, lalu setelah membuat struct tersebut kita akan deklarasikan rootnya dengan nilai nullptr dan inisiate dengan method void, kemudian kita akan membuat method pengecekan apakah tree kosong (belum dibuat) dengan method isEmpty yang dimana jika tree kosong akan mengembalikan nilai NULL untuk rootnya kemudian kita akan membuat method untuk penetapan root yang dimana fungsi dari method ini adalah sebagai titik awal dari tree, lalu kita akan membuat method untuk mencari node tertentu dengan kodingan yang ada seperti pada gambar, jika data tidak ada akan mengembalikan null, jika data ada maka method akan mengembalikan value dari node yang ditemukan.

```cpp
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

Pohon *cariNode(Pohon *node, char data) {
    if (!node) return NULL;
    if (node->data == data) return node;

    Pohon *found = cariNode(node->left, data);
    if (found) return found;

    return cariNode(node->right, data);
}
```

setelah itu kita akan membuat dua metod untuk insert, yakni insert left dan insert right yang dimana kita akan memasukanya kedalam node yang akan ditetapkan sebagai parent, untuk kondisi kedua insert kita akan membuat kondisi untuk tree yang belum dibuat, jika sudah ada anak pada parent node dan untuk data yang berhasil ditambahkan, untuk kodinganya dapat dilihat seperti pada gambar disamping.

```cpp
Pohon *insertLeft(char data, Pohon *parent) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu" << endl;
        return NULL;
    }
    if (!parent) {
        cout << "\nParent tidak ditemukan." << endl;
        return NULL;
    }
    if (parent->left != NULL) {
        cout << "\nNode " << parent->data << " sudah ada child kiri." << endl;
        return NULL;
    }
    Pohon *baru = new Pohon{data, NULL, NULL, parent};
    parent->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << parent->data << endl;
    return baru;
}

Pohon *insertRight(char data, Pohon *parent) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu" << endl;
        return NULL;
    }
    if (!parent) {
        cout << "\nParent tidak ditemukan." << endl;
        return NULL;
    }
    if (parent->right != NULL) {
        cout << "\nNode " << parent->data << " sudah ada child kanan." << endl;
        return NULL;
    }
    Pohon *baru = new Pohon{data, NULL, NULL, parent};
    parent->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << parent->data << endl;
    return baru;
}
```

```
void tampilkanAnak(Pohon *node) {
    if (!node) {
        cout << "\nNode tidak ditemukan" << endl;
        return;
    }
    cout << "\nAnak-anak dari node " << node->data << ": ";
    if (node->left) cout << "Kiri: " << node->left->data << " ";
    else cout << "Kiri: (tidak ada) ";
    if (node->right) cout << "Kanan: " << node->right->data << " ";
    else cout << "Kanan: (tidak ada) ";
    cout << endl;
}

void tampilkanDescendant(Pohon *node) {
    if (!node) return;
    if (node->left) {
        cout << node->left->data << " ";
        tampilkanDescendant(node->left);
    }
    if (node->right) {
        cout << node->right->data << " ";
        tampilkanDescendant(node->right);
    }
}

bool is_valid_bst(Pohon *node, char min_val, char max_val) {
    if (!node) 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);
}

int cari_simpul_daun(Pohon *node) {
    if (!node) return 0;
    if (!node->left && !node->right) return 1;
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}
```

```
void update(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu" << endl;
        return;
    }
    if (!node) {
        cout << "\nNode yang ingin diganti tidak ada." << endl;
        return;
    }
    char temp = node->data;
    node->data = data;
    cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
}

void menu() {
    int pilihan;
    char data, parentData;
    do {
        cout << "\nMenu: \n1. Buat Node Root\n2. Tambah Node Kiri\n3. Tambah Node Kanan";
        cout << "\n4. Tampilkan Anak\n5. Tampilkan Descendant";
        cout << "\n6. Tampilkan Anak (spesifik)\n7. Tampilkan Descendant (spesifik)";
        cout << "\n8. Cek Valid BST\n9. Hitung Jumlah Daun\n0. Keluar\n";
        cout << "Pilih: "; cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan data root: "; cin >> data;
                buatNode(data);
                break;
            case 2:
                cout << "Masukkan data parent: "; cin >> parentData;
                cout << "Masukkan data node kiri: "; cin >> data;
                insertLeft(data, cariNode(root, parentData));
                break;
            case 3:
                cout << "Masukkan data parent: "; cin >> parentData;
                cout << "Masukkan data node kanan: "; cin >> data;
                insertRight(data, cariNode(root, parentData));
                break;
            case 4:
                cout << "Masukkan data node: "; cin >> data;
                tampilkanAnak(cariNode(root, data));
                break;
```

Kemudian kita akan membuat method untuk menampilkan anak dari suatu parent node yang dimana ia akan menampilkan child left dan rightnya jika hanya terdapat satu maka hanya akan menampilkan satu dan untuk child yang kosong akan dikembalikan nilai kosong, lalu kita akan menampilkan descendant (turunan) dari sebuah node, dan untuk ini akan ditampilkan semua anaknya dari node tersebut, setelah selesai dengan method tampil descendant kita akan membuat method untuk mengecek validasi dari tree, yakni apakah tree tersebut masuk kedalam kondisi BST atau tidak untuk methodnya kita akan membuat kodingan yang mengecek satu satu dari node dan menggunakan pencarian dari tiap node, jika BST akan dikembalikan true, jika tidak maka false, kemudian kita akan masuk kedalam penghitungan simpul daun, yang dimana fungsi ini berguna untuk mencari daun (node yang tidak memiliki child dari sebuah tree, dan ia akan menunjukan semua node tanpa child didalam sebuah tree.

Kemudian kita akan masuk kedalam method terahir yakni update nilai dari sebuah node yaitu dengan mengganti value lama dengan value baru yang telah ditentukan oleh user. Lalu untuk bagian terahir adalah pembuatan menu untuk menunya kita akan membuat switch case untuk masing masing keinginan dari soal TP dan kebutuhan untuk pembuatan tree dan juga implementasi masing masing method untuk setiap case yang disesuaikan dengan kebutuhanya, case 1 yakni pembuatan root dengan method buat node, case 2 menambahkan child kiri untuk sebuah node, case 3 menambahkan child kanan dari sebuah node, case 4 menampilkan anak dari node yang di inginkan, case 5 menampilkan descendan (senua turunan) dari node, case 6 dan 7 sama dengan 4 dan 5, lalu untuk case 8 untuk mengecek validasi BST

```
            case 5:
                cout << "Masukkan data node: "; cin >> data;
                cout << "\nDescendant dari node " << data << ": ";
                tampilkanDescendant(cariNode(root, data));
                cout << endl;
                break;
            case 6:
                cout << "Masukkan data node: "; cin >> data;
                tampilkanAnak(cariNode(root, data));
                break;
            case 7:
                cout << "Masukkan data node: "; cin >> data;
                cout << "\nDescendant dari node " << data << ": ";
                tampilkanDescendant(cariNode(root, data));
                cout << endl;
                break;
            case 8:
                cout << "\nApakah pohon ini BST? ";
                cout << (is_valid_bst(root, CHAR_MIN, CHAR_MAX) ? "Ya" : "Tidak") << endl;
                break;
            case 9:
                cout << "\nJumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;
            case 0:
                cout << "Keluar." << endl;
                break;
            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 0);
}

int main() {
    init();
    menu();
    return 0;
}
```

dari tree, case 9 untuk exit dari program. Hasil dari koding yang telah dibuat adalah sebagai berikut ini:

Output di halaman berikut

```
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 1
Masukkan data root: X

Node X berhasil dibuat menjadi root.

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 2
Masukkan data parent: X
Masukkan data node kiri: Y

Node Y berhasil ditambahkan ke child kiri X

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 8

Apakah pohon ini BST? Tidak
```

```
Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 3
Masukkan data parent: X
Masukkan data node kanan: Z

Node Z berhasil ditambahkan ke child kanan X

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 5
Masukkan data node: X

Descendant dari node X: Y Z

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 2
Masukkan data parent: Y
Masukkan data node kiri: A

Node A berhasil ditambahkan ke child kiri Y
```

```
Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 4
Masukkan data node: Y

Anak-anak dari node Y: Kiri: A Kanan: (tidak ada)

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 9

Jumlah simpul daun: 2

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Anak
5. Tampilkan Descendant
6. Tampilkan Anak (spesifik)
7. Tampilkan Descendant (spesifik)
8. Cek Valid BST
9. Hitung Jumlah Daun
0. Keluar
Pilih: 0
Keluar.
```