LAPORAN PRAKTIKUM
Modul IX
“TREE”

Disusun Oleh:
Zivana Afra Yulianto -2211104039
SE-07-02

Dosen:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

- Memahami dan mengimplementasikan struktur data **Pohon Biner** menggunakan bahasa pemrograman C++.
- Mempelajari operasi dasar pada pohon biner seperti:

    - Membuat node root dan menambahkan node baru.
    - Menampilkan child dan descendants dari sebuah node.
    - Menentukan validitas pohon sebagai Binary Search Tree (BST).
    - Menghitung jumlah simpul daun pada pohon.

- Mengembangkan program dengan menu interaktif untuk mempermudah pengujian operasi pada pohon biner.

2. Landasan Teori

- Pohon Biner (Binary Tree)

Pohon biner adalah struktur data hierarkis di mana setiap simpul (node) memiliki paling banyak dua anak, yaitu anak kiri (*left child*) dan anak kanan (*right child*). Pohon ini sering digunakan dalam pemrograman untuk representasi data terstruktur, seperti ekspresi matematika, pengambilan keputusan, dan lainnya.

- Binary Search Tree (BST)

Binary Search Tree adalah jenis pohon biner yang memenuhi properti berikut:

    - Semua nilai pada subtree kiri lebih kecil daripada nilai pada node induk.
    - Semua nilai pada subtree kanan lebih besar daripada nilai pada node induk. BST sering digunakan untuk pengambilan data yang efisien.

- Traversal Pohon

Traversal adalah proses mengunjungi setiap node dalam pohon. Terdapat tiga metode traversal utama:

    - Pre-order: Node dikunjungi sebelum anak-anaknya (Root → Left → Right).
    - In-order: Node dikunjungi setelah anak kiri, tetapi sebelum anak kanan (Left → Root → Right).
    - Post-order: Node dikunjungi setelah anak-anaknya (Left → Right → Root).

- Simpul Daun (Leaf Node)

Simpul daun adalah node pada pohon yang tidak memiliki anak kiri maupun anak kanan. Dalam analisis pohon, simpul daun sering digunakan untuk menghitung tingkat kompleksitas atau mendeteksi pola dalam data.

- Implementasi Pohon

Pohon biner dapat diimplementasikan menggunakan struktur data rekursif di mana setiap node berisi data, pointer ke anak kiri, anak kanan, dan (opsional) induknya. Operasi pada pohon biasanya melibatkan fungsi rekursif untuk navigasi.

3. Guided

GUIDED :

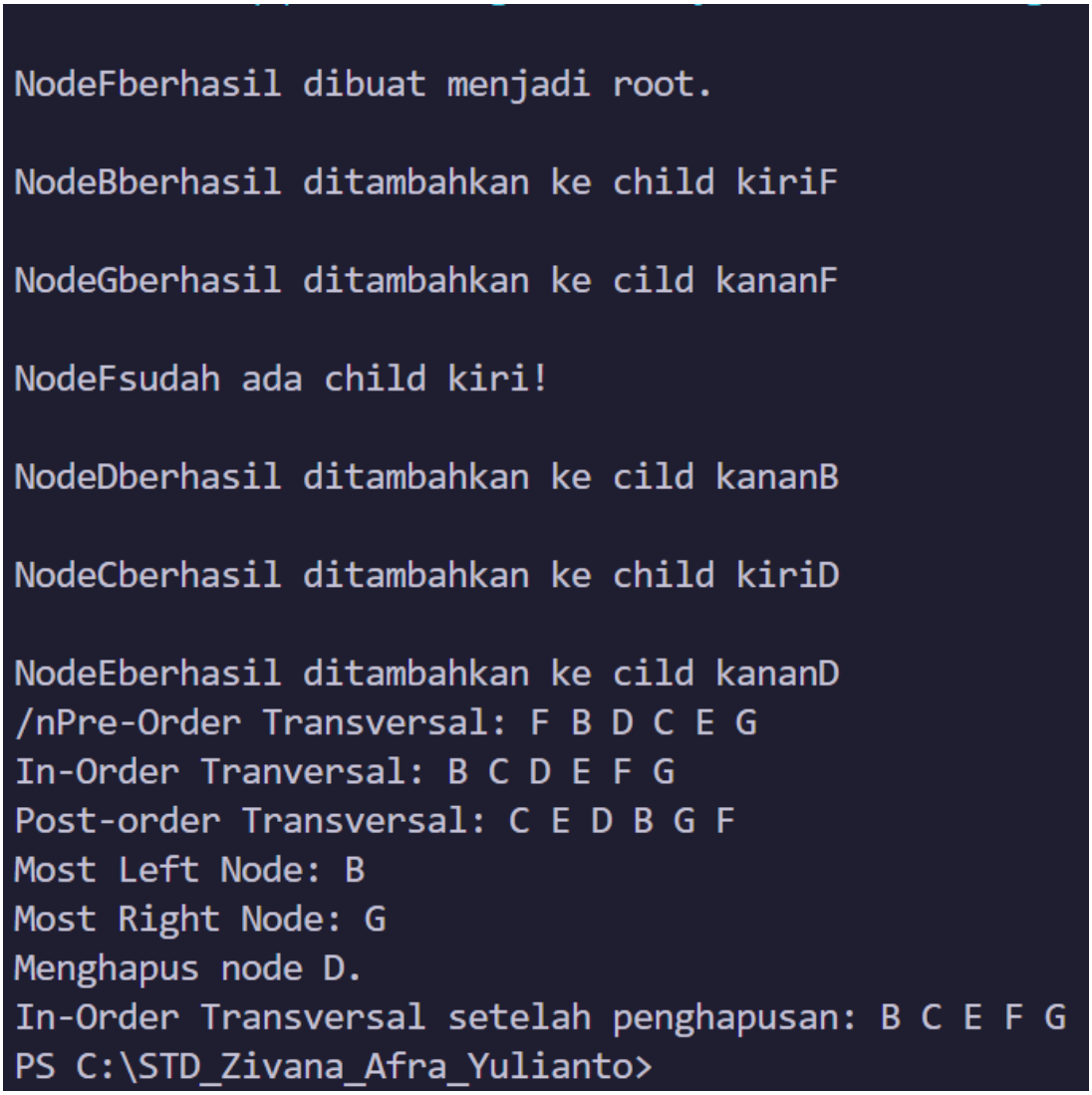
```
#include <iostream>
using namespace std;

/// PROGRAM BINARY TREE

// Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri,
kanan, dan induk
struct Pohon {
    char data;                // Data yang disimpan di node (tipe char)
    Pohon *left, *right;      // Pointer ke anak kiri dan anak kanan
    Pohon *parent;            // Pointer ke node induk
};

// Variabel global untuk menyimpan root (akar) pohon dan node baru
Pohon *root, *baru;

// Inisialisasi pohon agar kosong
void init() {
    root = NULL; // Mengatur root sebagai NULL (pohon kosong)
}

// Mengecek apakah pohon kosong
bool isEmpty() {
    return root == NULL; // Mengembalikan true jika root adalah NULL
}

// Membuat node baru sebagai root pohon
void buatNode(char data) {
    if (isEmpty()) { // Jika pohon kosong
        root = new Pohon{data, NULL, NULL, NULL}; // Membuat node baru sebagai
root
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak membuat
node baru
    }
}

// Menambahkan node baru sebagai anak kiri dari node tertentu
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) { // Jika anak kiri sudah ada
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node-
>data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Menambahkan node baru sebagai anak kanan dari node tertentu
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) { // Jika anak kanan sudah ada
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node-
>data << endl;
    return baru; // Mengembalikan pointer ke node baru
}
```

```
// Mengubah data di dalam sebuah node
void update(char data, Pohon *node)
{
    if (!node)
    { // Jika node tidak ditemukan
        cout << "\nNode yang ingin diubah tidak ditemukan!" << endl;
        return;
    }
    char temp = node->data; // Menyimpan data lama
    node->data = data;      // Mengubah data dengan nilai baru
    cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
}

// Mencari node dengan data tertentu
void find(char data, Pohon *node)
{
    if (!node)
        return; // Jika node tidak ada, hentikan pencarian

    if (node->data == data)
    { // Jika data ditemukan
        cout << "\nNode ditemukan: " << data << endl;
        return;
    }
    // Melakukan pencarian secara rekursif ke anak kiri dan kanan
    find(data, node->left);
    find(data, node->right);
}

// Traversal Pre-order (Node -> Kiri -> Kanan)
void preOrder(Pohon *node)
{
    if (!node)
        return;                // Jika node kosong, hentikan traversal
    cout << node->data << " "; // Cetak data node saat ini
    preOrder(node->left);      // Traversal ke anak kiri
    preOrder(node->right);     // Traversal ke anak kanan
}

// Traversal In-order (Kiri -> Node -> Kanan)
void inOrder(Pohon *node)
{
    if (!node)
        return;                // Jika node kosong, hentikan traversal
    inOrder(node->left);       // Traversal ke anak kiri
    cout << node->data << " "; // Cetak data node saat ini
    inOrder(node->right);      // Traversal ke anak kanan
}

// Traversal Post-order (Kiri -> Kanan -> Node)
void postOrder(Pohon *node)
{
    if (!node)
        return;                // Jika node kosong, hentikan traversal
    postOrder(node->left);     // Traversal ke anak kiri
    postOrder(node->right);    // Traversal ke anak kanan
    cout << node->data << " "; // Cetak data node saat ini
}

// Menghapus node dengan data tertentu
Pohon *deleteNode(Pohon *node, char data)
{
    if (!node)
        return NULL; // Jika node kosong, hentikan

    // Rekursif mencari node yang akan dihapus
    if (data < node->data)
    {
        node->left = deleteNode(node->left, data); // Cari di anak kiri
    }
    else if (data > node->data)
    {
        node->right = deleteNode(node->right, data); // Cari di anak kanan
    }
```

```
    else
    {
        // Jika node ditemukan
        if (!node->left)
        {                                // Jika tidak ada anak kiri
            Pohon *temp = node->right; // Anak kanan menggantikan posisi node
            delete node;
            return temp;
        }
        else if (!node->right)
        {                               // Jika tidak ada anak kanan
            Pohon *temp = node->left; // Anak kiri menggantikan posisi node
            delete node;
            return temp;
        }

        // Jika node memiliki dua anak, cari node pengganti (successor)
        Pohon *successor = node->right;
        while (successor->left)
            successor = successor->left;                          // Cari node
terkecil di anak kanan
        node->data = successor->data;                             // Gantikan data
dengan successor
        node->right = deleteNode(node->right, successor->data); // Hapus
successor
    }
    return node;
}

// Menemukan node paling kiri
Pohon *mostLeft(Pohon *node)
{
    if (!node)
        return NULL; // Jika node kosong, hentikan
    while (node->left)
        node = node->left; // Iterasi ke anak kiri hingga mentok
    return node;
}

// Menemukan node paling kanan
Pohon *mostRight(Pohon *node)
{
    if (!node)
        return NULL; // Jika node kosong, hentikan
    while (node->right)
        node = node->right; // Iterasi ke anak kanan hingga mentok
    return node;
}

// Fungsi utama
int main()
{
    init();                                // Inisialisasi pohon
    buatNode('F');                         // Membuat root dengan data 'F'
    insertLeft('B', root);                 // Menambahkan 'B' ke anak kiri root
    insertRight('G', root);                // Menambahkan 'G' ke anak kanan root
    insertLeft('A', root->left);           // Menambahkan 'A' ke anak kiri dari 'B'
    insertRight('D', root->left);          // Menambahkan 'D' ke anak kanan dari
'B'
    insertLeft('C', root->left->right);  // Menambahkan 'C' ke anak kiri dari 'D'
    insertRight('E', root->left->right); // Menambahkan 'E' ke anak kanan dari
'D'

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

    // Menampilkan node paling kiri dan kanan
    cout << "\nMost Left Node: " << mostLeft(root)->data;
    cout << "\nMost Right Node: " << mostRight(root)->data;
```

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

    return 0;
}
```

Screenshoot output :



Deskripsi :

Program ini adalah implementasi Pohon Biner dalam C++ yang mendukung operasi berikut:

1. Pembuatan Pohon: Membuat root dan menambahkan anak kiri/kanan.
2. Transversal: Menampilkan elemen pohon dalam urutan Pre-order, In-order, dan Post-order.
3. Pencarian Node: Menemukan node berdasarkan data.
4. Penghapusan Node: Menghapus node sesuai aturan Binary Search Tree (BST).
5. Node Ekstrem: Menemukan node dengan nilai terkecil (paling kiri) dan terbesar (paling kanan).

4. Unguided

UNGUIDED :

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

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

Pohon *root, *baru;

// Fungsi inisialisasi dan dasar (sama seperti sebelumnya)
void init() { root = NULL; }
bool isEmpty() { return root == NULL; }

// Fungsi untuk membuat node baru
Pohon* createNode(char data) {
    Pohon* newNode = new Pohon{data, NULL, NULL, NULL};
    return newNode;
}

// Fungsi untuk menambahkan node
Pohon* insertNode(Pohon* node, char data, char parentData, bool isLeft) {
    if (!node) return NULL;

    // Jika node yang dicari ditemukan
    if (node->data == parentData) {
        Pohon* newNode = createNode(data);

        if (isLeft) {
            if (node->left) {
                cout << "Node kiri sudah terisi!" << endl;
                return NULL;
            }
            node->left = newNode;
            newNode->parent = node;
        } else {
            if (node->right) {
                cout << "Node kanan sudah terisi!" << endl;
                return NULL;
            }
            node->right = newNode;
            newNode->parent = node;
        }
        return newNode;
    }

    // Rekursif pencarian
    Pohon* leftResult = insertNode(node->left, data, parentData, isLeft);
    if (leftResult) return leftResult;

    Pohon* rightResult = insertNode(node->right, data, parentData, isLeft);
    return rightResult;
}

// Fungsi 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);
    return rightResult;
}
```

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

    cout << "Node: " << node->data << endl;
    if (node->left)
        cout << "Child Kiri: " << node->left->data << endl;
    else
        cout << "Tidak memiliki child kiri" << endl;

    if (node->right)
        cout << "Child Kanan: " << node->right->data << endl;
    else
        cout << "Tidak memiliki child kanan" << endl;
}

// Fungsi untuk menampilkan descendants
void tampilkanDescendant(Pohon* node) {
    if (!node) {
        cout << "Node tidak ditemukan!" << endl;
        return;
    }

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

    // Fungsi rekursif untuk mencetak descendants
    if (!node->left && !node->right) {
        cout << "Tidak memiliki descendants" << endl;
        return;
    }

    // Pre-order traversal untuk descendants
    if (node->left) {
        cout << node->left->data << " ";
        tampilkanDescendant(node->left);
    }

    if (node->right) {
        cout << node->right->data << " ";
        tampilkanDescendant(node->right);
    }

    cout << endl;
}

// Fungsi untuk memeriksa apakah suatu pohon adalah BST (Rekursif)
bool is_valid_bst(Pohon* node, char min_val = numeric_limits<char>::min(),
                  char max_val = numeric_limits<char>::max()) {
    // Base case: pohon kosong dianggap valid
    if (!node) return true;

    // Cek apakah node saat ini memenuhi batasan
    if (node->data <= min_val || node->data >= max_val)
        return false;

    // Rekursif ke anak kiri dan kanan
    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}

// Fungsi untuk menghitung simpul daun (Rekursif)
int cari_simpul_daun(Pohon* node) {
    // Base case
    if (!node) return 0;

    // Jika node adalah daun (tidak memiliki anak)
    if (!node->left && !node->right) return 1;

    // Rekursif ke anak kiri dan kanan
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}
```

```
// Menu interaktif
void menu() {
    int pilihan;
    char data, parentData;

    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 Descendants" << endl;
        cout << "6. Cek BST" << endl;
        cout << "7. Hitung Simpul Daun" << endl;
        cout << "0. Keluar" << endl;
        cout << "Pilihan: ";
        cin >> pilihan;

        switch(pilihan) {
            case 1:
                if (isEmpty()) {
                    cout << "Masukkan data root: ";
                    cin >> data;
                    root = createNode(data);
                } else {
                    cout << "Root sudah ada!" << endl;
                }
                break;

            case 2:
            case 3:
                if (isEmpty()) {
                    cout << "Buat root terlebih dahulu!" << endl;
                    break;
                }

                cout << "Masukkan data node baru: ";
                cin >> data;
                cout << "Masukkan data parent: ";
                cin >> parentData;

                insertNode(root, data, parentData, pilihan == 2);
                break;

            case 4:
                cout << "Masukkan node yang ingin dilihat child-nya: ";
                cin >> data;
                tampilkanChild(findNode(root, data));
                break;

            case 5:
                cout << "Masukkan node yang ingin dilihat descendants-nya: ";
                cin >> data;
                tampilkanDescendant(findNode(root, data));
                break;

            case 6:
                cout << "Pohon " << (is_valid_bst(root) ? "valid" : "tidak valid")
                     << " sebagai BST" << endl;
                break;

            case 7:
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;

            case 0:
                cout << "Keluar dari program." << endl;
                break;

            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 0);
}
```

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

Screenshoot output :

1. Membuat root

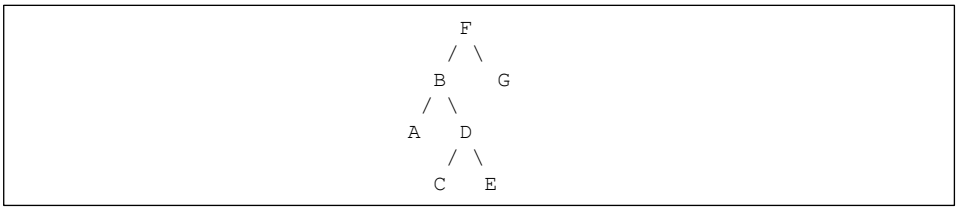
```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 1
Masukkan data root: F
```

2. Menambahkan node

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 2
Masukkan data node baru: B
Masukkan data parent: F

--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 3
Masukkan data node baru: G
Masukkan data parent: F

--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 2
Masukkan data node baru: A
Masukkan data parent: B
```

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 3
Masukkan data node baru: D
Masukkan data parent: B

--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 2
Masukkan data node baru: C
Masukkan data parent: D

--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 3
Masukkan data node baru: E
Masukkan data parent: D
```

Dari data yang dimasukan pohon akan berbentuk seperti dibawah ini:



3. Menampilkan child

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 4
Masukkan node yang ingin dilihat child-nya: B
Node: B
Child Kiri: A
Child Kanan: D
```

4. Menampilkan descendants

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 5
Masukkan node yang ingin dilihat descendants-nya: F
Descendants dari F: B Descendants dari B: A Descendants dari A: Tidak memiliki descendants
```

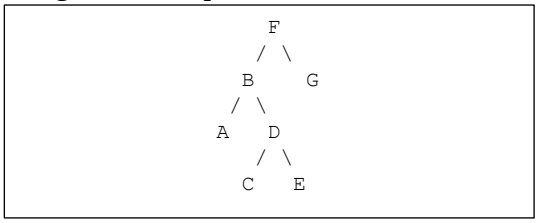
5. Chek apakah BTS

   Agar pohon dinyatakan valid sebagai BST, harus dipenuhi kondisi-kondisi berikut:
   1. Setiap node di subtree kiri HARUS memiliki nilai lebih kecil dari node induknya
   2. Setiap node di subtree kanan HARUS memiliki nilai lebih besar dari node induknya
   3. Baik subtree kiri maupun kanan juga harus merupakan BST

- Valid

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 6
Pohon valid sebagai BST
```

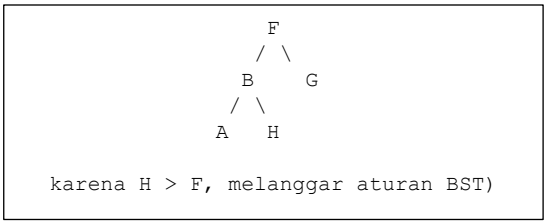
Dengan bentuk pohon :



- Tidak valid

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 6
Pohon tidak valid sebagai BST
```

Dengan bentuk pohon :



6. Hitung simpul daun

```
--- MENU POHON BINER ---
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Cek BST
7. Hitung Simpul Daun
0. Keluar
Pilihan: 7
Jumlah simpul daun: 4
```

Deskripsi :

Program ini adalah implementasi pohon biner dalam C++ yang menyediakan menu interaktif untuk berbagai operasi berikut:

1. Buat Root: Membuat node root untuk pohon.
2. Tambah Node: Menambahkan node baru sebagai anak kiri atau kanan dari node tertentu.
3. Tampilkan Child: Menampilkan anak kiri dan kanan dari sebuah node.
4. Tampilkan Descendants: Menampilkan semua keturunan dari sebuah node.
5. Cek BST: Memeriksa apakah pohon memenuhi sifat Binary Search Tree (BST).
6. Hitung Simpul Daun: Menghitung jumlah simpul daun dalam pohon.
7. Keluar: Mengakhiri program.

5. Kesimpulan

- **Pemahaman Pohon Biner**

Program berhasil mengimplementasikan struktur pohon biner, termasuk operasi dasar seperti menambahkan node, mencari node, menampilkan child dan descendants, serta traversal pohon.

- **Validasi Binary Search Tree**

Program menyediakan fungsi untuk memeriksa validitas pohon sebagai BST dengan membandingkan setiap node terhadap batas maksimum dan minimum yang sesuai.

- **Fungsi Interaktif**

Menu interaktif membantu pengguna untuk memahami dan menguji konsep pohon biner secara langsung, seperti menambahkan node dan menghitung simpul daun.

- **Efisiensi Rekursi**

Fungsi seperti traversal, pencarian node, dan validasi BST menggunakan pendekatan rekursif yang efisien dalam menangani struktur pohon.