LAPORAN PRAKTIKUM
Modul 09
“TREE(BAGIAN KEDUA)”

Disusun Oleh:
Faishal Arif Setiawan -2311104066
Kelas:
SE 07 02

Dosen :
WAHYU ANDI SAPUTRA.S.PD.,M.ENG

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **TUJUAN**

   1. Mengimplementasikan struktur data *tree*, khususnya *Binary Tree*

2. **Landasan Teori**

   Tree adalah struktur data hierarkis yang terdiri dari node-node yang saling terhubung, sering digunakan untuk merepresentasikan hubungan hierarkis seperti silsilah keluarga atau struktur file. Operasi penting pada tree meliputi traversal (pre-order, in-order, post-order), penambahan node, dan penghapusan node. Dalam Binary Search Tree (BST), aturan utamanya adalah semua node di subtree kiri memiliki nilai lebih kecil, dan semua node di subtree kanan memiliki nilai lebih besar dari node induknya. Node yang paling kiri (most-left) dalam BST mewakili nilai terkecil, sementara node yang paling kanan (most-right) adalah nilai terbesar.

   Simpul daun (leaf node) adalah node tanpa anak kiri maupun kanan, dan sering digunakan untuk menghitung elemen terminal dalam tree. Penghapusan node pada tree bergantung pada jumlah anak node yang dihapus: simpul daun dihapus langsung, node dengan satu anak digantikan oleh anaknya, dan node dengan dua anak digantikan oleh nilai pengganti dari subtree (most-right dari kiri atau most-left dari kanan).

3. **Guided**

-Struct pohon

Struktur ini di gunakan untuk mendefinisikan node dari pohon biner.

Data:menyimpan data berupa karakter

Left:pointer ke anak kiri

Right:pointer ke anak kanan

Parent:pointer ke induk node

-Variabel global

Root:menyimpan akar pohon(node pertama)

Baru:Variabel sementara untuk membuat node baru.

Init():
Inisialisasi pohon dengan mengosongkan root
Isempty():
Mengecek apakah pohon kosong
buatNode():
membuat node baru dengan data tertentu.
insertLeft() dan inserRight():
menambahkan node baru sebagai anak kiri atau kanan dari node tertentu
mengecek apakah anak kiri atau kanan sudah ada sebelum menambahkan
updateChar():
mengubah data dalam node tertentu.
-Traversal:
  preOrder():
  traversal dalam urutan Root-> Left -> Right.
  inOrder():
  traversal dalam urutan Left->Root->Right.
  postOrder()
  traversal dalam urutan Left->Right->Root.
-deleteNode()
 Menghapus Node berdasarkan aturan.
-mostLeft() dan mostRight():
  Mencari node paling kiri(nilai terkecil)atau paling kanan(nilai terbesar) dalam pohon.

  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
Most Left Node: A
Most Right Node: G
Menghapus node D.
In-order Traversal setelah penghapusan: A B C E F G
PS D:\struktur data pemograman\GUIDED9\output>
```

4. **Unguided**

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

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

Pohon *root;

void init() {
    root = NULL;
}

Pohon* buatNode(char data, Pohon *parent = NULL) {
    return new Pohon{data, NULL, NULL, parent};
}

Pohon* insertLeft(char data, Pohon *node) {
    if (node->left) {
        cout << "Node " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    }
    node->left = buatNode(data, node);
    return node->left;
}

Pohon* insertRight(char data, Pohon *node) {
    if (node->right) {
        cout << "Node " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }
    node->right = buatNode(data, node);
    return node->right;
}

void simpulAktif(Pohon *node) {
    if (!node) {
        cout << "Node tidak ditemukan!" << endl;
        return;
    }
    cout << "Child dari " << node->data << ": ";
    if (node->left) cout << node->left->data << " (kiri) ";
    if (node->right) cout << node->right->data << " (kanan)";
    if (!node->left && !node->right) cout << "Tidak ada";
    cout << endl;
}

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

int main() {
    init();
    char pilihan, data, parent_data;
    Pohon *node;

    do {
        cout << "\nMENU BST:";
        cout << "\n1. Buat root";
        cout << "\n2. Tambah child kiri";
        cout << "\n3. Tambah child kanan";
        cout << "\n4. Lapirkan child";
        cout << "\n5. Tampilkan descendant";
        cout << "\n6. Cek apakah BST";
        cout << "\n7. Hitung jumlah simpul daun";
        cout << "\n0. Keluar";
        cout << "\nPilih: ";
        cin >> pilihan;

        switch (pilihan) {
            case '1': {
                if (!root) {
                    cout << "Masukkan data root: ";
                    cin >> data;
                    root = buatNode(data);
                    cout << "Root dengan data " << data << " berhasil dibuat." << endl;
                } else {
                    cout << "Root sudah ada!" << endl;
                }
                break;
            }
            case '2': {
                cout << "Masukkan data parent: ";
                cin >> parent_data;
                node = root;
                if (node && node->data == parent_data) {
                    cout << "Masukkan data anak kiri: ";
                    cin >> data;
                    insertLeft(data, node);
                } else {
                    cout << "Parent tidak ditemukan." << endl;
                }
                break;
            }
            case '3': {
                cout << "Masukkan data parent: ";
                cin >> parent_data;
                node = root;
                if (node && node->data == parent_data) {
                    cout << "Masukkan data anak kanan: ";
                    cin >> data;
                    insertRight(data, node);
                } else {
                    cout << "Parent tidak ditemukan." << endl;
                }
                break;
            }
            case '4': {
                cout << "Masukkan data node: ";
                cin >> data;
                node = root;
                if (node && node->data == data) {
                    tampilChild(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            }
            case '5': {
                cout << "Masukkan data node: ";
                cin >> data;
                node = root;
                if (node && node->data == data) {
                    cout << "Descendant dari " << node->data << ": ";
                    tampilDescendant(node);
                    cout << endl;
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            }
            case '6': {
                if (is_valid_bst(root, numeric_limits<char>::min(), numeric_limits<char>::max())) {
                    cout << "Pohon adalah Binary Search Tree (BST)." << endl;
                } else {
                    cout << "Pohon BUKAN Binary Search Tree (BST)." << endl;
                }
                break;
            }
            case '7': {
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;
            }
            case '0': {
                cout << "Keluar program." << endl;
                break;
            }
            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != '0');

    return 0;
}
```

Fungsi tampilChild dan tampilDescendant menampilkan anak langsung dan keturunan dari node tertentu.

is_valid_bst memeriksa apakah pohon memenuhi sifat BST.

cari_simpul_daun menghitung jumlah simpul daun menggunakan rekursi.

Output:

```
MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 1
Masukkan data root: F
Root dengan data F berhasil dibuat.

MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 2
Masukkan data parent: F
Masukkan data anak kiri: B

MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 3
Masukkan data parent: F
Masukkan data anak kanan: G
```

```
MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 4
Masukkan data node: F
Child dari F: B (kiri) G (kanan)

MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 5
Masukkan data node: F
Descendant dari F: B G

MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 6
Pohon adalah Binary Search Tree (BST).
```

```
MENU TREE:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendant
6. Cek apakah BST
7. Hitung jumlah simpul daun
0. Keluar
Pilih: 7
Jumlah simpul daun: 2
```

5. Kesimpulan

Pada praktikum ini, berhasil mengimplementasikan berbagai operasi dasar pada struktur data pohon, khususnya pada Binary Search Tree (BST). Dalam implementasi ini, kami mempelajari cara menambahkan, menghapus, serta melakukan traversal pada pohon dengan metode pre-order, in-order, dan post-order. Selain itu, kami juga mengimplementasikan fungsi untuk mencari node paling kiri dan paling kanan dalam pohon, yang mewakili nilai terkecil dan terbesar pada BST. Dengan memahami operasi dasar seperti ini, kami dapat menerapkan konsep tree pada berbagai masalah pemrograman yang memerlukan penyimpanan data secara hierarkis. Secara keseluruhan, penggunaan struktur data tree, terutama BST, sangat berguna untuk masalah yang melibatkan pencarian data yang cepat dan efisien.