Tugas Pendahuluan Modul 9

STRUKTUR DATA - Ganjil 2024/2025

Tree

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 2 adalah Senin, 30 September 2024 pukul 07.30 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. Codingan diupload di Github dan upload Laporan di Lab menggunakan format PDF dengan ketentuan: TP_MOD_[XX]_NIM_NAMA.pdf

CP (WA):

- Andini (082243700965)
- Imelda (082135374187)

SELAMAT MENGERJAKAN^^

LAPORAN PRAKTIKUM
PERTEMUAN 9
TREE

Nama :

Tsaqif Hisyam Saputra (2311104024)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan
    - Mahasiswa mampu menjelaskan konsep dari struktur data Tree (Binary Tree)
    - Mahasiswa mampu membuat program dengan menerapkan struktur data Tree (Binary Tree)
    - Mahasiswa mampu membuat fungsi tambah simpul dan harus simpul pada struktur data Tree (Binary Tree)

B. Tools
    Visual Studio Code dengan C++ Extensions Pack & Codeblocks

C. Latihan – Unguided

```cpp
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() {
    int choice;
    do {
        cout << "\nMenu Tree:\n";
        cout << "1. Buat Root Node\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Update Node\n";
        cout << "5. Retrieve Node\n";
        cout << "6. Find Node\n";
        cout << "7. Show Descendants\n";
        cout << "8. Check if Tree is Valid BST\n";
        cout << "9. Count Leaf Nodes\n";
        cout << "10. Exit\n";
        cout << "Pilihan: ";
        cin >> choice;

        char data, parent_data;
        Pohon *node = nullptr;

        switch (choice) {
            case 1:
                cout << "Masukkan data root: ";
                cin >> data;
                node = buatNode(data);
                break;
            case 2:
                cout << "Masukkan data parent: ";
                cin >> parent_data;
                node = root;
                while (node && node->data != parent_data) {
                    node = (node->left && node->left->data == parent_data) ? node->left : node->right;
                }
                if (node) {
                    cout << "Masukkan data node kiri: ";
                    cin >> data;
                    insertLeft(data, node);
                } else {
                    cout << "Parent tidak ditemukan.\n";
                }
                break;
            case 3:
                cout << "Masukkan data parent: ";
                cin >> parent_data;
                node = root;
                while (node && node->data != parent_data) {
                    node = (node->left && node->left->data == parent_data) ? node->left : node->right;
                }
                if (node) {
                    cout << "Masukkan data node kanan: ";
                    cin >> data;
                    insertRight(data, node);
                } else {
                    cout << "Parent tidak ditemukan.\n";
                }
                break;
            case 4:
                cout << "Masukkan data node yang ingin diupdate: ";
                cin >> parent_data;
                node = root;
                while (node && node->data != parent_data) {
                    node = (node->left && node->left->data == parent_data) ? node->left : node->right;
                }
                if (node) {
                    cout << "Masukkan data baru: ";
                    cin >> data;
                    update(data, node);
                } else {
                    cout << "Node tidak ditemukan.\n";
                }
                break;
            case 5:
                retrieve(root);
                break;
            case 6:
                cout << "Masukkan data node yang ingin dicari: ";
                cin >> data;
                node = root;
                while (node && node->data != data) {
                    node = (node->left && node->left->data == data) ? node->left : node->right;
                }
                if (node) {
                    find(node);
                } else {
                    cout << "Node tidak ditemukan.\n";
                }
                break;
            case 7:
                cout << "Masukkan data node yang ingin ditampilkan descendantnya: ";
                cin >> data;
                node = root;
                while (node && node->data != data) {
                    node = (node->left && node->left->data == data) ? node->left : node->right;
                }
                if (node) {
                    cout << "Descendants of " << data << ": ";
                    showDescendants(node);
                    cout << endl;
                } else {
                    cout << "Node tidak ditemukan.\n";
                }
                break;
            case 8:
                cout << "Tree is " << (is_valid_bst(root, CHAR_MIN, CHAR_MAX) ? "a valid BST" : "not a valid BST") << endl;
                break;
            case 9:
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;
            case 10:
                cout << "Keluar dari program." << endl;
                break;
            default:
                cout << "Pilihan tidak valid. Coba lagi." << endl;
        }
    } while (choice != 10);

    return 0;
}
```

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

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

Pohon *root = nullptr;

void init() {
    root = nullptr;
}

bool isEmpty() {
    return root == nullptr;
}

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

Pohon* insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return nullptr;
    }

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

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

Pohon* insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return nullptr;
    }

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

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

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 retrieve(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return;
    }

    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
        return;
    }
    cout << "\nData Node: " << node->data << endl;
}

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return;
    }

    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
        return;
    }

    cout << "\nData Node: " << node->data << endl;
    if (node->parent) {
    cout << "Parent: " << node->parent->data << endl;
    } else {
    cout << "Parent: Tidak ada parent" << endl;
    }

    if (node->left) cout << "Left Child: " << node->left->data << endl;
    else cout << "Left Child: (None)" << endl;

    if (node->right) cout << "Right Child: " << node->right->data << endl;
    else cout << "Right Child: (None)" << endl;
}

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

Program diatas adalam implementasi dari pohon biner, program dimulai dengan mendeklarasikan struktur Pohon untuk merepresentasikan simpul pada pohon biner. Struktur ini memiliki empat anggota **data** (nilai simpul), **left** (child kiri), **right** (child kanan), dan **parent** (simpul induk).

Dalam program tersebut memiliki inisialisasi Pohon dengan fungsi **init()** yang menginisialisasi pohon dengan mengatur **root** menjadi **NULL**, menandaakan pohon kosong.

```
void init() {
    root = nullptr;
}
```

Fungsi **buatNode** membuat simpul root jika pohon kosong. Fungsi **insertLeft** dan **insertRight** memungkinkan penambahan child kiri dan kanan pada simpul tertentu, dengan melakukan pengecekan agar simpul tersebut belum memiliki child di posisi yang di maksud.

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

```
Pohon* insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return nullptr;
    }

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

    Pohon* baru = new Pohon{data, nullptr, nullptr, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
    return baru;
}
```

```cpp
Pohon* insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return nullptr;
    }

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

    Pohon* baru = new Pohon{data, nullptr, nullptr, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return baru;
}
```

Untuk memanipulasi data simpul memakai fungsi **update** untuk mengubah nilai data dari simpul yang ditunjuk, fungsi **retrieve** yang menampilkan data simpul trtentu, dan fungsi **find** yang menampilkan data simpul beserta informasi terkait seperti parent dan sibling, serta menangani simpul yang mungkin tidak memiliki parent atau sibling.

```
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 retrieve(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return;
    }

    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
        return;
    }
    cout << "\nData node: " << node->data << endl;
}

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return;
    }

    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
        return;
    }
```

Lalu untuk menmpilkan descendant memakai fungsi **showDescendants** menampilkan semua turunan dari simpul yang dipilih menggunakan rekursi.

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

Untuk mengvalidasi BST memakai **is_valid_bst** yang merupakan fungsi rekursif yang memeriksa apakah pohon memenuhii properti BST, di mana setiap simpul kiri harus memiliki nilai lebih kecil dan setiap simpul kanan harus lebih besar dari simpul induk. Fungsi ini mengembalikan true jika pohon valid sebagai BST, atau false jika tidak.

```
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);
}
```

Menhitung simpul daun memerlukan fungsi **cari_simpul_daun** menghitung jumlah simpul daun dalam pohon. Smpul daun adalah simpul yang tidak memiliki anak kiri maupun kanan.

```
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);
}
```

Output:

```
Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 1
Masukkan data root: A

Node A berhasil dibuat menjadi root.

Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 2
Masukkan data parent: A
Masukkan data node kiri: B

Node B berhasil ditambahkan ke child kiri A
```

```
Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 3
Masukkan data parent: A
Masukkan data node kanan: C

Node C berhasil ditambahkan ke child kanan A
```

```
Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 5

Data node: A

Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 6
Masukkan data node yang ingin dicari: B

Data Node: B
Left Child: D
Right Child: E
```

```
Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 4
Masukkan data node yang ingin diupdate: B
Masukkan data baru: Z

Node B berhasil diubah menjadi Z

Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 8
Tree is not a valid BST

Menu Tree:
1. Buat Root Node
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Show Descendants
8. Check if Tree is Valid BST
9. Count Leaf Nodes
10. Exit
Pilihan: 9
Jumlah simpul daun: 4
```

Semoga Selalu diberi kemudahan^^