LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL IX
GRAPH DAN TREE

Disusun Oleh :

NAMA : WISNU RANANTA RADITYA PUTRA

NIM : 2311102013

Dosen :

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa diharapkan mampu memahami graph dan tree
2. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II

DASAR TEORI

1. Graph

   Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai : G = (V, E) Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

   Jenis-jenis Graph:

   a. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.

   b. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.

   c. Weigth Graph : Graph yang mempunyai nilai pada tiap edgenya.

   Representasi graph:

   a. Matriks: Menggunakan matriks untuk mewakili koneksi antar simpul.

   b. Linked List: Menggunakan daftar berantai untuk mewakili koneksi antar simpul.

2. Tree

   Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi.

   Operasi pada Tree

   a. Create: digunakan untuk membentuk binary tree baru yang masih kosong.

   b. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.

   c. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.

   d. Insert: digunakan untuk memasukkan sebuah node kedalam tree.

   e. Find: digunakan untuk mencari root, parent, left child, atau right child darisuatu node dengan syarat tree tidak boleh kosong.

   f. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

g. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

h. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

i. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.

j. Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

   - Pre-Order: Root, Left, Right.
   - In-Order: Left, Root, Right.
   - Post-Order: Left, Right, Root.

BAB III

GUIDED

Guided 1

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                    "Bandung",
                    "Bekasi",
                    "Tasikmalaya",
                    "Cianjur",
                    "Purwokerto",
                    "Yogjakarta"};
int busur[7][7] =
    {
        {0, 7, 8, 0, 0, 0, 0},
        {0, 0, 5, 0, 0, 15, 0},
        {0, 6, 0, 0, 5, 0, 0},
        {0, 5, 0, 0, 2, 4, 0},
        {23, 0, 0, 10, 0, 0, 8},
        {0, 0, 0, 0, 7, 0, 3},
        {0, 0, 0, 0, 9, 4, 0}};
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
             << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "("<<
busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}
int main()
{
    tampilGraph();
    return 0;
}
```

Screenshots Output

```
 Ciamis          :  Bandung(7) Bekasi(8)
 Bandung         :  Bekasi(5) Purwokerto(15)
 Bekasi          :  Bandung(6) Cianjur(5)
 Tasikmalaya     :  Bandung(5) Cianjur(2) Purwokerto(4)
 Cianjur         :  Ciamis(23) Tasikmalaya(10) Yogjakarta(8)
 Purwokerto      :  Cianjur(7) Yogjakarta(3)
 Yogjakarta      :  Cianjur(9) Purwokerto(4)
PS C:\Semester 2\Praktikum Strutur Data\Modul_9>
```

Deskripsi:

Program di atas mendefinisikan dan menampilkan representasi sebuah graf dengan simpul dan busur. Terdapat array `simpul` yang menyimpan nama-nama simpul, dan matriks `busur` yang menyimpan bobot antara simpul-simpul tersebut. Fungsi `tampilGraph()` digunakan untuk menampilkan graf dengan iterasi melalui setiap simpul dan menampilkan busur yang terhubung beserta bobotnya. Pada fungsi `main()`, `tampilGraph()` dipanggil untuk menampilkan struktur graf ke layar, memperlihatkan koneksi antara kota-kota yang direpresentasikan sebagai simpul dalam graf.

Guided 2

```
#include <iostream>

using namespace std;

/// PROGRAM BINARY TREE

// Deklarasi Pohon

struct Pohon

{

    char data;

    Pohon *left, *right, *parent;

};

Pohon *root, *baru;

// Inisialisasi

void init()

{

    root = NULL;

}

// Cek Node

int isEmpty()

{
```

```
    if (root == NULL)
        return 1;
    else
        return 0;
    // true
    // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
          cout << "\n Node " << data << " berhasil dibuat
menjadi root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
```

```
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
             cout << "\n Node " << node->data << " sudah ada
child kiri!"<< endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
                cout << "\n Node " << data << " berhasil
ditambahkan ke child kiri " << baru->parent->data << endl;
            return baru;
        }
    }
}
// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
```

```
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
             cout << "\n Node " << node->data << " sudah ada
child kanan!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
                cout << "\n Node " << data << " berhasil
ditambahkan ke child kanan " << baru->parent->data << endl;
                return baru;
        }
    }
}
// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
```

```
    else
    {
        if (!node)
             cout << "\n Node yang ingin diganti tidak ada!!"
<< endl;
        else
        {
            char temp = node->data;
            node->data = data;
             cout << "\n Node " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
              cout << "\n Node yang ditunjuk tidak ada!" <<
endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}
// Cari Data Tree
```

```
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
              cout << "\n Node yang ditunjuk tidak ada!" <<
endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;
            if (!node->parent)
                 cout << " Parent : (tidak punya parent)" <<
endl;
            else
                 cout << " Parent : " << node->parent->data <<
endl;
             if (node->parent != NULL && node->parent->left !=
node &&
                node->parent->right == node)
                 cout << " Sibling : " << node->parent->left-
>data << endl;
               else if (node->parent != NULL && node->parent-
>right != node &&
                     node->parent->left == node)
                 cout << " Sibling : " << node->parent->right-
>data << endl;
            else
                 cout << " Sibling : (tidak punya sibling)" <<
endl;
            if (!node->left)
```

```
                    cout << " Child Kiri : (tidak punya Child
kiri)" << endl;
            else
                 cout << " Child Kiri : " << node->left->data
<< endl;
            if (!node->right)
                  cout << " Child Kanan : (tidak punya Child
kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right->data
<< endl;
        }
    }
}
// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}
// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
```

```
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}
// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}
// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
```

```
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}
// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
```

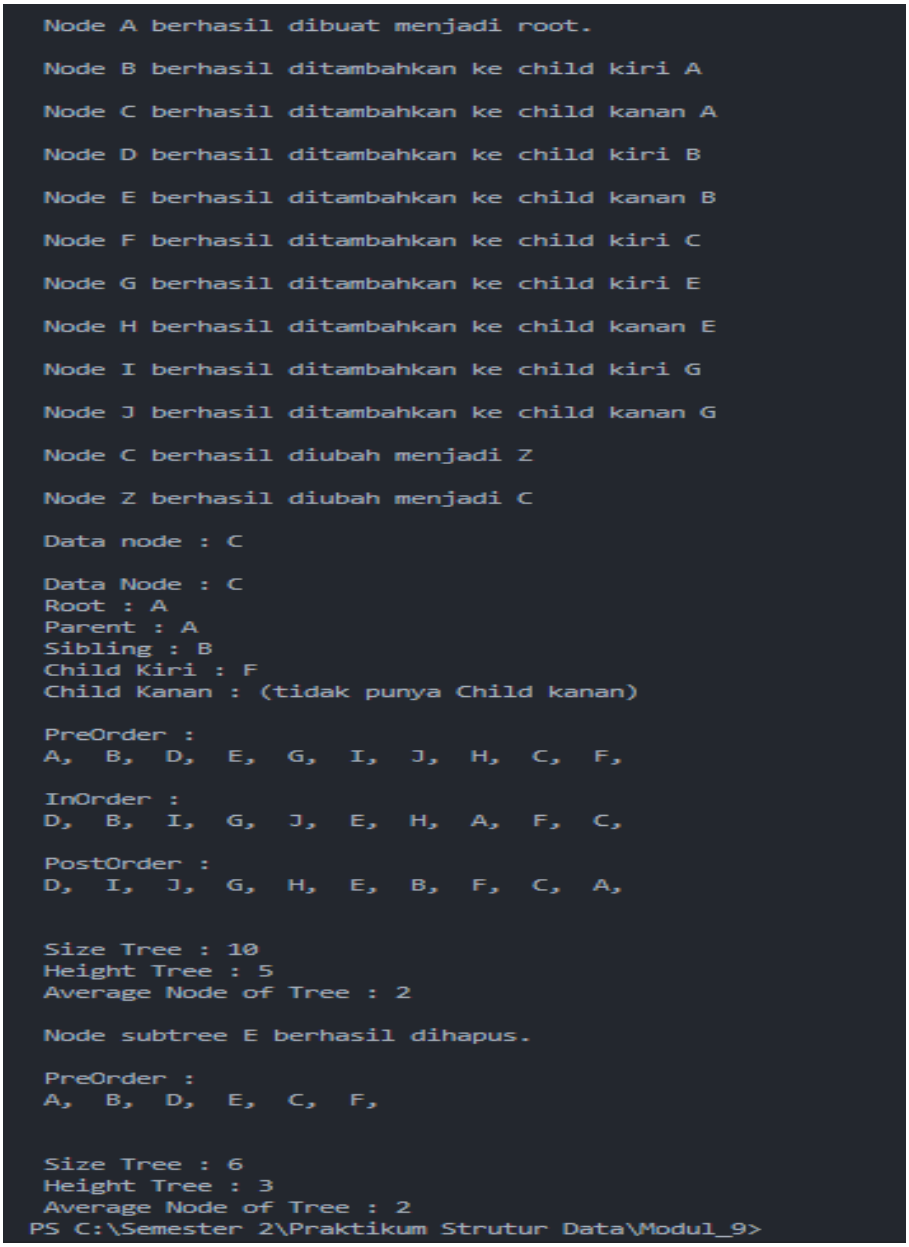
```
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
```

```
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}
// Karakteristik Tree
void charateristic()
```

```
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}
int main()
{
    buatNode('A');
     Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,*nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"<< endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"<< endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"<< endl;
```

```
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"<< endl;
    charateristic();
}
```

Screenshots Output:



Deskripsi :

Program ini mengimplementasikan struktur data pohon biner. Dimulai dengan deklarasi struktur pohon yang memiliki data dan pointer ke anak kiri, anak kanan, serta induk. Fungsi-fungsi utama mencakup inisialisasi pohon (`init`), pengecekan apakah pohon kosong (`isEmpty`), pembuatan node baru (`buatNode`), penambahan node di anak kiri (`insertLeft`) dan kanan (`insertRight`), pengubahan data node (`update`), serta penampilan data node (`retrieve`) dan pencarian node (`find`). Terdapat juga fungsi untuk traversal pohon (pre-order, in-order, post-order), penghapusan pohon atau subtree (`deleteTree`, `deleteSub`), serta pemeriksaan karakteristik pohon (`size`, `height`, `charateristic`). Pada fungsi `main`, pohon biner dibangun dengan root 'A' dan beberapa node tambahan, diikuti oleh berbagai operasi yang hasilnya ditampilkan ke layar.

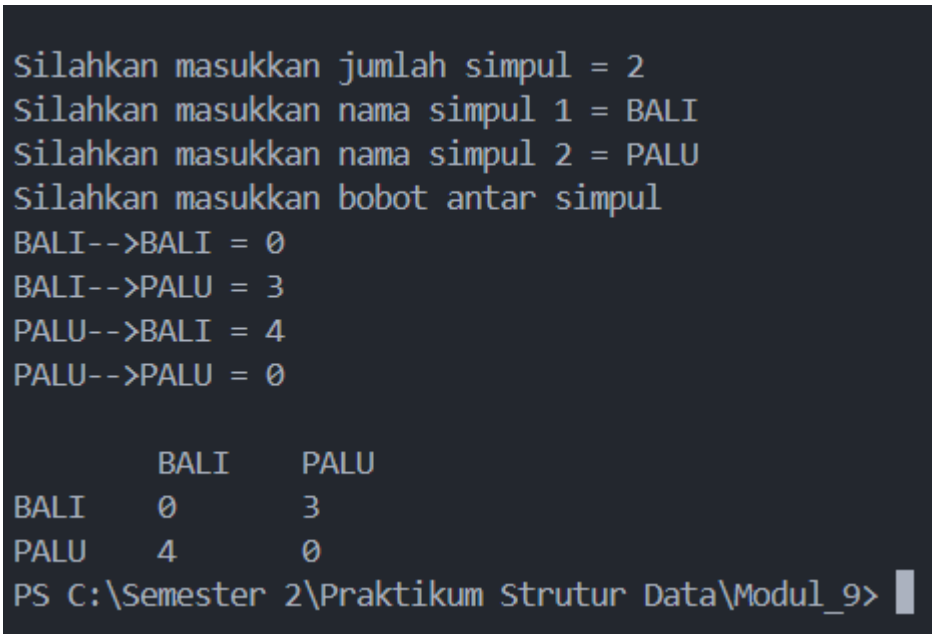
BAB IV
UNGUIDED

Unguided 1

Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

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

int main() {
    int jmlSimpulRadit_2311102013;
    cout << "Silahkan masukkan jumlah simpul = ";
    cin >> jmlSimpulRadit_2311102013;

    vector<string>
namaSimpulRadit_2311102013(jmlSimpulRadit_2311102013);
    vector<vector<int>> bobot(jmlSimpulRadit_2311102013,
vector<int>(jmlSimpulRadit_2311102013));

    for (int i = 0; i < jmlSimpulRadit_2311102013; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << "
= ";
        cin >> namaSimpulRadit_2311102013[i];
    }

    cout << "Silahkan masukkan bobot antar simpul\n";

    for (int i = 0; i < jmlSimpulRadit_2311102013; ++i) {
        for (int j = 0; j < jmlSimpulRadit_2311102013; ++j) {
            cout << namaSimpulRadit_2311102013[i] << "-->" <<
namaSimpulRadit_2311102013[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << "\n\t";
    for (int i = 0; i < jmlSimpulRadit_2311102013; ++i) {
        cout << namaSimpulRadit_2311102013[i] << "\t";
    }
    cout << "\n";

    for (int i = 0; i < jmlSimpulRadit_2311102013; ++i) {
        cout << namaSimpulRadit_2311102013[i] << "\t";
        for (int j = 0; j < jmlSimpulRadit_2311102013; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }

    return 0;
}
```

Screenshots Output:



Deskripsi:

Program ini meminta pengguna untuk memasukkan jumlah simpul pada sebuah graf, kemudian memasukkan nama setiap simpul tersebut. Selanjutnya, pengguna diminta untuk menginput bobot antar simpul yang disimpan dalam matriks bobot. Setelah semua data dimasukkan, program menampilkan tabel yang menunjukkan bobot antar simpul dengan nama simpul sebagai baris dan kolom. Program menggunakan `vector` untuk menyimpan nama simpul dan bobot antar simpul, serta melakukan iterasi melalui input dan output menggunakan loop.

Unguided 2

Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!

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

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

Pohon *root, *baru;

// Inisialisasi
void init()
{
```

```
    root = NULL;
}

// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1;
    else
        return 0;
}

// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat menjadi
root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah
memiliki child kiri!" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
```

```
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data << " berhasil
ditambahkan sebagai child kiri dari " << baru->parent->data <<
endl;
            return baru;
        }
    }
}

// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah
memiliki child kanan!" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil
ditambahkan sebagai child kanan dari " << baru->parent->data <<
endl;
            return baru;
        }
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!" <<
```

```
endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;
            if (!node->parent)
                cout << " Parent : (tidak memiliki parent)" <<
endl;
            else
                cout << " Parent : " << node->parent->data <<
endl;
            if (node->parent != NULL && node->parent->left !=
node &&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left-
>data << endl;
            else if (node->parent != NULL && node->parent-
```

```
>right != node &&
                    node->parent->left == node)
                cout << " Sibling : " << node->parent->right-
>data << endl;
            else
                cout << " Sibling : (tidak memiliki sibling)"
<< endl;
            if (!node->left)
                cout << " Child Kiri : (tidak memiliki child
kiri)" << endl;
            else
                cout << " Child Kiri : " << node->left->data <<
endl;
            if (!node->right)
                cout << " Child Kanan : (tidak memiliki child
kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right->data
<< endl;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}
```

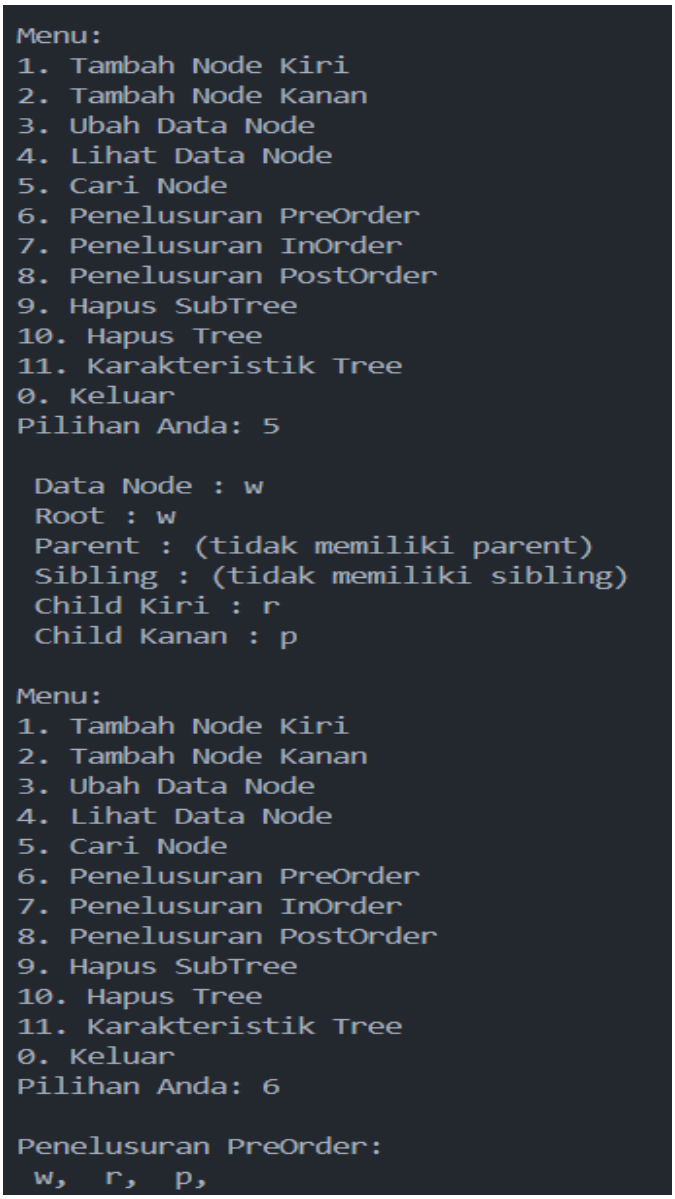
```
// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
```

```
    }
}

// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
```

```
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void charateristic()
{
    cout << "\n Ukuran Tree : " << size() << endl;
    cout << " Tinggi Tree : " << height() << endl;
    cout << " Rata-rata Node pada Tree : " << size() / height()
<< endl;
}

int main()
{
    init();

    char data;
    cout << "Masukkan data root: ";
    cin >> data;
    buatNode(data);

    char pilihan;
    do
    {
        cout << "\nMenu:\n";
        cout << "1. Tambah Node Kiri\n";
        cout << "2. Tambah Node Kanan\n";
        cout << "3. Ubah Data Node\n";
        cout << "4. Lihat Data Node\n";
        cout << "5. Cari Node\n";
        cout << "6. Penelusuran PreOrder\n";
        cout << "7. Penelusuran InOrder\n";
        cout << "8. Penelusuran PostOrder\n";
        cout << "9. Hapus SubTree\n";
        cout << "10. Hapus Tree\n";
        cout << "11. Karakteristik Tree\n";
        cout << "0. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> pilihan;

        switch (pilihan)
        {
        case '1':
            char dataKiriRadit_2311102013;
            cout << "Masukkan data node kiri: ";
            cin >> dataKiriRadit_2311102013;
            insertLeft(dataKiriRadit_2311102013, root);
            break;
        case '2':
```

```
            char dataKananRadit_2311102013;
            cout << "Masukkan data node kanan: ";
            cin >> dataKananRadit_2311102013;
            insertRight(dataKananRadit_2311102013, root);
            break;
        case '3':
            char dataUbahRadit_2311102013;
            cout << "Masukkan data yang ingin diubah: ";
            cin >> dataUbahRadit_2311102013;
            update(dataUbahRadit_2311102013, root);
            break;
        case '4':
            retrieve(root);
            break;
        case '5':
            find(root);
            break;
        case '6':
            cout << "\nPenelusuran PreOrder:\n";
            preOrder(root);
            cout << endl;
            break;
        case '7':
            cout << "\nPenelusuran InOrder:\n";
            inOrder(root);
            cout << endl;
            break;
        case '8':
            cout << "\nPenelusuran PostOrder:\n";
            postOrder(root);
            cout << endl;
            break;
        case '9':
            deleteSub(root);
            break;
        case '10':
            clear();
            break;
        case '11':
            charateristic();
            break;
        case '0':
            cout << "\nTerima kasih telah menggunakan program
ini!\n";
            break;
        default:
            cout << "\nPilihan tidak valid!\n";
            break;
        }
    } while (pilihan != '0');

    return 0;
}
```

Screenshots Output:



Deskripsi:

Program di atas merupakan implementasi dari struktur data pohon biner dengan menu interaktif untuk pengguna. Program ini mendefinisikan struktur `Pohon` untuk merepresentasikan node dalam pohon biner, serta menyediakan berbagai fungsi untuk menginisialisasi pohon (`init`), membuat node baru (`buatNode`), menambah node kiri (`insertLeft`) dan kanan (`insertRight`), mengubah data node (`update`), melihat isi node (`retrieve`), mencari node (`find`), serta traversal pohon dalam urutan pre-order, in-order, dan post-order. Program juga menyediakan fungsi untuk menghapus subtree (`deleteSub`) dan seluruh pohon (`clear`), serta menampilkan karakteristik pohon seperti ukuran dan tinggi. Melalui menu interaktif, pengguna dapat memasukkan data secara dinamis, melakukan berbagai operasi pada pohon, dan melihat hasilnya secara langsung.

BAB V

KESIMPULAN

Graph dan tree adalah dua struktur data fundamental dalam ilmu komputer yang digunakan untuk merepresentasikan hubungan antara objek dan menyimpan data secara hierarki. Graph, terdiri dari node (simpul) dan edge (busur), menggambarkan hubungan antar objek dengan jenis seperti directed graph, undirected graph, dan weight graph, yang direpresentasikan melalui matriks atau linked list. Sementara itu, tree merupakan struktur hierarki dengan node yang terhubung seperti dalam bentuk pohon, dengan operasi seperti create, clear, insert, dan delete sub, serta metode traversal seperti Pre-Order, In-Order, dan Post-Order. Keduanya penting dalam pemodelan sistem kompleks dan representasi jaringan, dengan graph lebih fleksibel untuk hubungan umum antar objek, sementara tree efektif untuk data hierarki dan operasi berbasis struktur.

BAB VI

DAFTAR PUSTAKA

[1] Asisten Praktikum. 2024. Modul 9 “GRAPH DAN TREE”. Diakses 10 Juni 2024, 19:00 WIB. https://lms.ittelkom-pwt.ac.id/