

# **LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA**

## **MODUL IX GRAPH DAN TREE**



Disusun Oleh :

NAUFAL THORIQ MUZHAFAR

2311102078

Dosen

WAHYU ANDI SAPUTRA, S.Pd., M.Eng

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

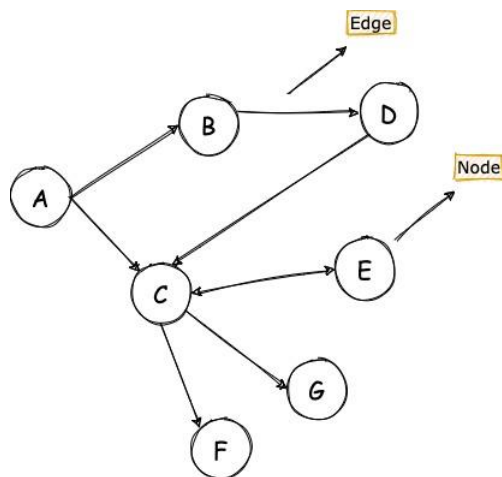
**2024**

## A. Dasar Teori

Data Structure atau Struktur Data adalah sebuah konsep dalam sebuah komputer yang mengacu pada cara pengaturan data, pengolahan dan kemudian menyimpan data tersebut ke dalam memori atau media penyimpanan dengan tujuan agar dapat diakses dan dimodifikasi kembali secara lebih efektif dan efisien.

### GRAPH

Graph terdiri dari beberapa kumpulan titik (node) dan garis (edge).



**Node**, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri.

**Edge**, adalah penghubung antara satu node dengan node yang lain. Sebuah garis harus diawali dan diakhiri titik.

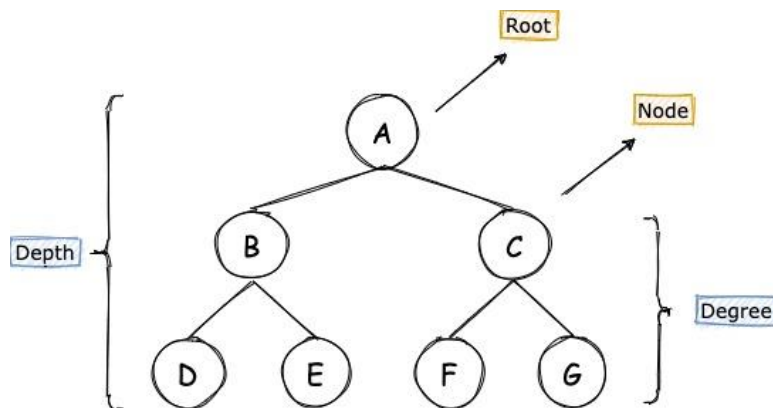
**Path** adalah jalur dari satu titik ke titik lain. Sebuah path yang diawali dan diakhiri dengan titik yang sama disebut juga dengan simpul tertutup.

Berdasarkan orientasi arah sisi nya, graph dapat dibedakan menjadi 2 yaitu :

- Directed graph atau graf berarah adalah graph yang setiap sisi nya memiliki orientasi arah.
- Undirected graph atau graf tak berarah adalah graph yang sisi nya tidak memiliki orientasi arah.

## TREE

Tree adalah struktur data non linier berbentuk hierarki yang terdiri dari sekumpulan node yang berbeda.



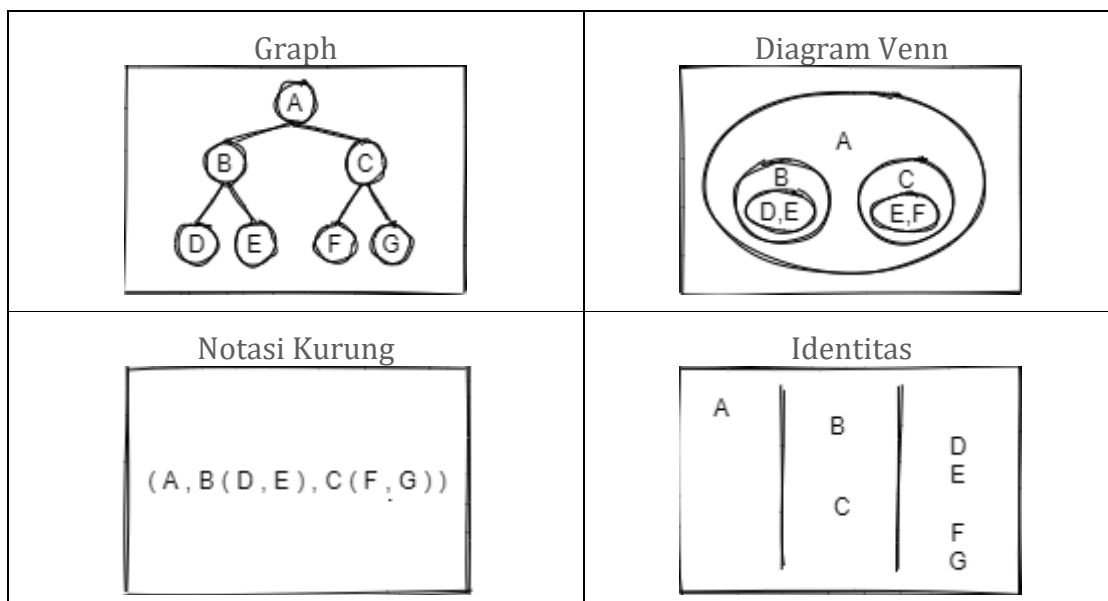
**Node**, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri.

**Root**, adalah sebuah node yang terletak di posisi tertinggi atau urutan pertama dari suatu tree.

**Depth**, adalah jarak atau ketinggian antara root dan node.

**Degree**, adalah banyaknya anak atau turunan dari suatu node.

Ada beberapa cara untuk menggambar sebuah tree, diantaranya dapat dengan :



Tree yang hanya memiliki maksimal dua child (anak) disebut dengan binary tree atau pohon biner.

Operasi yang terdapat pada binary tree berdasarkan gambar diatas umumnya dapat dilakukan dengan urutan – urutan sebagai berikut :

- Pre Order (DFS – Depth First Search) : A, B, D, E, C, F, G
- In Order : D, B, E, A, F, C, G
- Last Order : D, E, B, F, G, C, A
- Level Order (BFS – Bread First Search): A, B, C, D, E, F, G

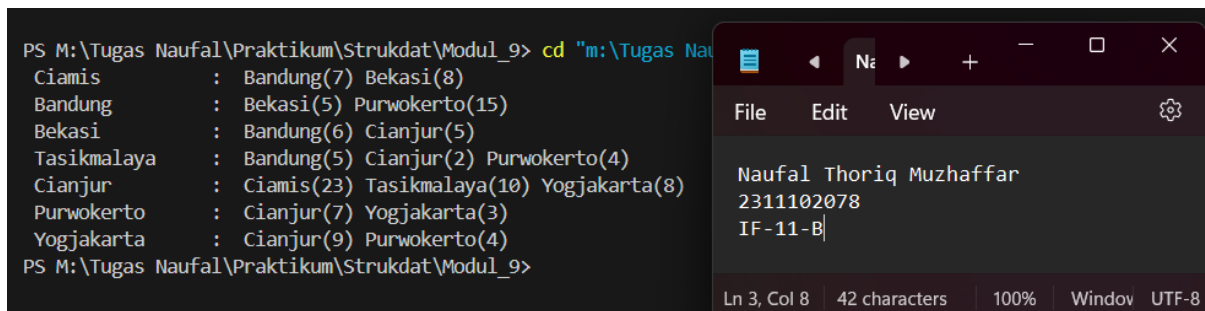
## B. Guided

### Guided 1

#### Source Code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur",
"Purwokerto", "Yogyakarta"};
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



```
PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9> cd "m:\Tugas Naufal\Praktikum\Strukdat\Modul_9"
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) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9>
```

Naufal Thoriq Muzhaffar  
2311102078  
IF-11-B

## Deskripsi Program

Program ini mendefinisikan dan menampilkan graf berbobot sederhana yang merepresentasikan hubungan antar kota dalam array simpul dan busur. Array simpul berisi nama-nama kota, sedangkan array busur berisi bobot (berupa jarak atau biaya) antar kota yang terhubung. Fungsi tampilGraph mencetak setiap simpul dan semua busur yang terhubung dengannya dengan format yang teratur menggunakan manipulasi I/O dari pustaka <iomanip>. Program ini dimulai dengan memanggil fungsi tampilGraph di dalam fungsi main, yang kemudian menampilkan representasi graf di konsol.

## Guided 2

### Source Code

```
#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; // true
    else
```

```

        return 0; // 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

```

PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9> cd "m:\Tugas Naufal\Praktikum\Strukdat\Modul_9\" ; if ($?) { g++ guided_2.cpp
Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kanan A
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kanan B
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kanan E
Node I berhasil ditambahkan ke child kiri G
Node J berhasil ditambahkan ke child kanan G
Node C berhasil diubah menjadi Z
Node Z berhasil diubah menjadi C
Data node : C
Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)
PreOrder :
A, B, D, E, G, I, J, H, C, F,
InOrder :
D, B, I, G, J, E, H, A, F, C,
PostOrder :
D, I, J, G, H, E, B, F, C, A,
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2
Node subtree E berhasil dihapus.
PreOrder :
A, B, D, E, C, F,
Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9>

```

## Deskripsi Program

Program dimulai dengan deklarasi struktur "Pohon" yang memiliki atribut untuk data, anak kiri, anak kanan, dan parent. Fungsi-fungsi yang disediakan dalam program ini meliputi inisialisasi pohon, pengecekan apakah pohon kosong, penambahan node baru, penambahan anak kiri dan kanan, pengubahan data node, pengambilan data node, pencarian data node, berbagai jenis traversal (preOrder, inOrder, postOrder), penghapusan pohon atau subtree, serta penghitungan ukuran dan tinggi pohon. Program juga menyajikan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Dalam fungsi main, contoh implementasi pohon biner dibuat dengan beberapa node, disusul dengan demonstrasi operasi-operasi yang tersedia pada program.

## C. Unguided

### Unguided 1

#### Source Code

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

int main() {
    int jmlhsmpl_2311102078;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jmlhsmpl_2311102078;

    string simpul[jmlhsmpl_2311102078];
    int busur[jmlhsmpl_2311102078][jmlhsmpl_2311102078];

    for (int i = 0; i < jmlhsmpl_2311102078; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jmlhsmpl_2311102078; i++) {
        for (int j = 0; j < jmlhsmpl_2311102078; j++) {
            cout << "Silakan masukkan bobot antara simpul " << simpul[i] <<
" dan " << simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }

    cout << "\nGraf yang dihasilkan:\n";
```

```

    cout << setw(15) << " ";
    for (int i = 0; i < jmlhsmpl_2311102078; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jmlhsmpl_2311102078; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jmlhsmpl_2311102078; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

## Screenshots Output

```

PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9> cd "m:\Tugas Naufal\Praktikum\Strukdat\Modul_9"
Silakan masukkan jumlah simpul: 2
Simpul 1: PEMALANG
Simpul 2: PURWOKERTO
Silakan masukkan bobot antara simpul PEMALANG dan PEMALANG: 0
Silakan masukkan bobot antara simpul PEMALANG dan PURWOKERTO: 8
Silakan masukkan bobot antara simpul PURWOKERTO dan PEMALANG: 7
Silakan masukkan bobot antara simpul PURWOKERTO dan PURWOKERTO: 0

Graf yang dihasilkan:

```

	PEMALANG	PURWOKERTO
PEMALANG	0	8
PURWOKERTO	7	0

```

PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9>

```

## Deskripsi Program

Program ini meminta pengguna untuk memasukkan jumlah simpul pada sebuah graf, kemudian menerima nama-nama simpul tersebut dan bobot busur yang menghubungkan setiap pasangan simpul. Setelah menerima semua input, program menampilkan graf dalam bentuk matriks yang terformat rapi menggunakan fungsi `setw` dari pustaka `iomanip` untuk mengatur lebar setiap kolom.

## Unguided 2

### Source Code

```

#include <iostream>
using namespace std;

// Deklarasi Pohon
struct Pohon {

```



```

    char data;
    Pohon *left, *right, *parent; // Pointer
};

// Pointer global
Pohon *root_2311102078;

// Inisialisasi
void init() {
    root_2311102078 = NULL;
}

bool isEmpty() {
    return root_2311102078 == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root_2311102078 = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah memiliki child kiri!"
<< endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;

```

```

        cout << "\nNode " << data << " berhasil ditambahkan ke child
kiri dari " << node->data << endl;
        return baru;
    }
}

Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah memiliki child
kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child
kanan dari " << node->data << endl;
            return baru;
        }
    }
}

void update(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else {
            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;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
    }
}

```

```

        else {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root_2311102078->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) {
    if (isEmpty())

```

```

        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

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

// postOrder
void postOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat 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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            if (node != root_2311102078) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    node->parent->right = NULL;
            }
        }
    }
}

```

```

        deleteTree(node->left);
        deleteTree(node->right);

        if (node == root_2311102078) {
            delete root_2311102078;
            root_2311102078 = NULL;
        } else {
            delete node;
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil dihapus." <<
endl;
    }
}

// Hapus Tree
void clear() {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(root_2311102078);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat 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) {
    if (isEmpty()) {
        cout << "\nBuat 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 characteristic() {
    int s = size(root_2311102078);
    int h = height(root_2311102078);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

// Menampilkan Child dari Sebuah Node
void displayChild(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nChild dari node " << node->data << " adalah:";
            if (node->left) {
                cout << " " << node->left->data;
            }
            if (node->right) {
                cout << " " << node->right->data;
            }
        }
    }
}

```

```

        cout << endl;
    }
}

// Menampilkan Descendant dari Sebuah Node
void displayDescendant(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nDescendant dari node " << node->data << " adalah:";
            // Gunakan rekursi untuk mencetak descendant
            if (node->left) {
                cout << " " << node->left->data;
                displayDescendant(node->left);
            }
            if (node->right) {
                cout << " " << node->right->data;
                displayDescendant(node->right);
            }
            cout << endl;
        }
    }
}

int main() {
    init();
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI,
    *nodeJ;

    nodeB = insertLeft('B', root_2311102078);
    nodeC = insertRight('C', root_2311102078);
    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 << "\nPreOrder :" << endl;  
    preOrder(root_2311102078);  
    cout << "\n" << endl;  
    cout << "InOrder :" << endl;  
    inOrder(root_2311102078);  
    cout << "\n" << endl;  
    cout << "PostOrder :" << endl;  
    postOrder(root_2311102078);  
    cout << "\n" << endl;  
    characteristic();  
    displayChild(nodeE);  
    displayDescendant(nodeB);  
    deleteSub(nodeE);  
    cout << "\nPreOrder :" << endl;  
    preOrder(root_2311102078);  
    cout << "\n" << endl;  
    characteristic();  
}
```



## Screenshots Output

```
PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9> cd "m:\Tugas Naufal\Praktikum\Strukdat\Modul_9\" ; if ($?) { g++ unguided_2.cpp -o unguided_2 } ;

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri dari A
Node C berhasil ditambahkan ke child kanan dari A
Node D berhasil ditambahkan ke child kiri dari B
Node E berhasil ditambahkan ke child kanan dari B
Node F berhasil ditambahkan ke child kiri dari C
Node G berhasil ditambahkan ke child kiri dari E
Node H berhasil ditambahkan ke child kanan dari E
Node I berhasil ditambahkan ke child kiri dari G
Node J berhasil ditambahkan ke child kanan dari G
Node C berhasil diubah menjadi Z
Node Z berhasil diubah menjadi C

Data node : C
Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak memiliki child kanan)

PreOrder :
A, B, D, E, G, I, J, H, C, F,

InOrder :
D, B, I, G, J, E, H, A, F, C,

PostOrder :
D, I, J, G, H, E, B, F, C, A,

Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Child dari node E adalah: G H
Descendant dari node B adalah: D
Descendant dari node D adalah:
E
Descendant dari node E adalah: G
Descendant dari node G adalah: I
Descendant dari node I adalah:
J
Descendant dari node J adalah:
H
Descendant dari node H adalah:

Node subtree E berhasil dihapus.

PreOrder :
A, B, D, E, C, F,

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS M:\Tugas Naufal\Praktikum\Strukdat\Modul_9>
```

## Deskripsi Program

Pohon diinisialisasi dengan sebuah node root dan mendukung penambahan node anak kiri dan kanan, pembaruan nilai node, serta pengambilan informasi node. Program ini juga memiliki fungsi untuk traversal pohon dalam urutan pre-order, in-order, dan post-order, serta fungsi

untuk menghapus node, subtree, atau seluruh pohon. Selain itu, terdapat fitur untuk menghitung ukuran dan tinggi pohon, serta menampilkan karakteristik pohon dan anak atau keturunan dari sebuah node tertentu. Program ini dimulai dengan membuat root node 'A' dan beberapa node anak, serta menampilkan hasil traversal dan karakteristik pohon sebelum dan setelah beberapa operasi penghapusan node.

#### **D. Kesimpulan**

Graph dan tree adalah dua jenis struktur data non-linear yang digunakan untuk menyimpan dan mengelola data dalam bentuk simpul (nodes) dan tepi (edges). Tree adalah bentuk khusus dari graph yang memiliki struktur hierarkis dengan satu simpul root dan tanpa siklus, di mana setiap simpul hanya dapat memiliki satu parent. Graph, di sisi lain, lebih umum dan dapat memiliki berbagai hubungan antara simpul, termasuk siklus dan multi-relasi, serta bisa tidak berarah atau berarah. Kedua struktur data ini sangat penting dalam berbagai aplikasi komputer, termasuk representasi jaringan, sistem hirarki, dan algoritma searching

#### **E. Referensi**

- [1] Asprak “Modul 9 Graph dan Tree”. Learning Management System 2024.
- [2] Ramdannur – Data Structure : Mengenal Graph & Tree. Diakses pada 10 Juni 2024  
<https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>