

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



Disusun Oleh :

NAMA : Pandia Arya Brata

NIM : 2311102076

Dosen

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA INSTITUT TEKNOLOGI TELKOM
PURWOKERTO 2024**

MODUL 10

GRAPH DAN TREE

A. TUJUAN PRAKTIKUM

- a.** Mahasiswa diharapkan mampu memahami graph dan tree
- b.** Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

B. DASAR TEORI

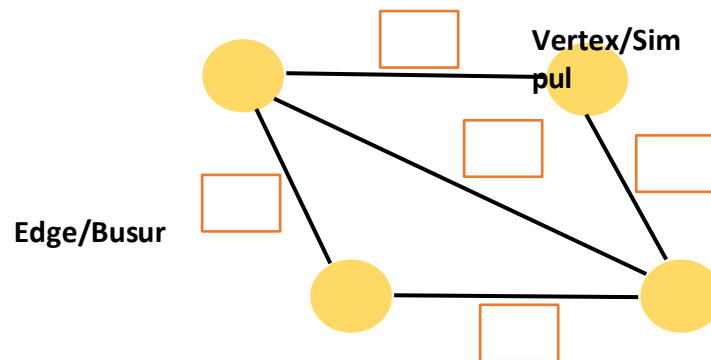
1. Graph

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 sisi 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 E sebagai sisi atau edge.

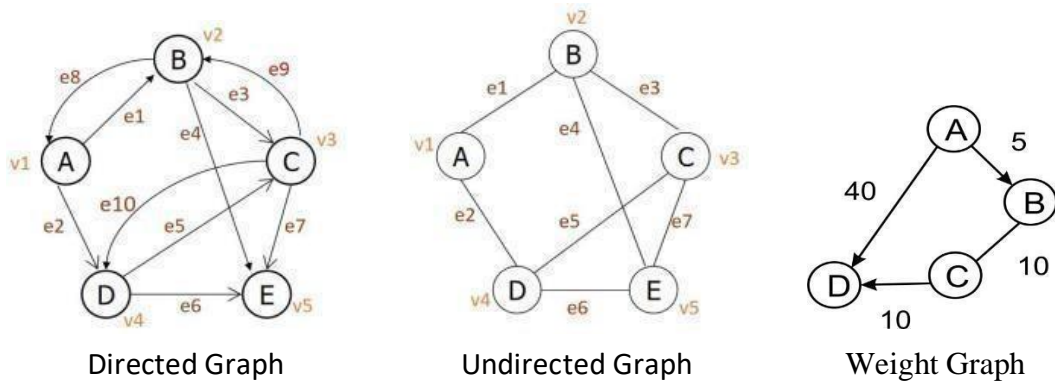
Dapat digambarkan:



Gambar 1 Contoh Graph

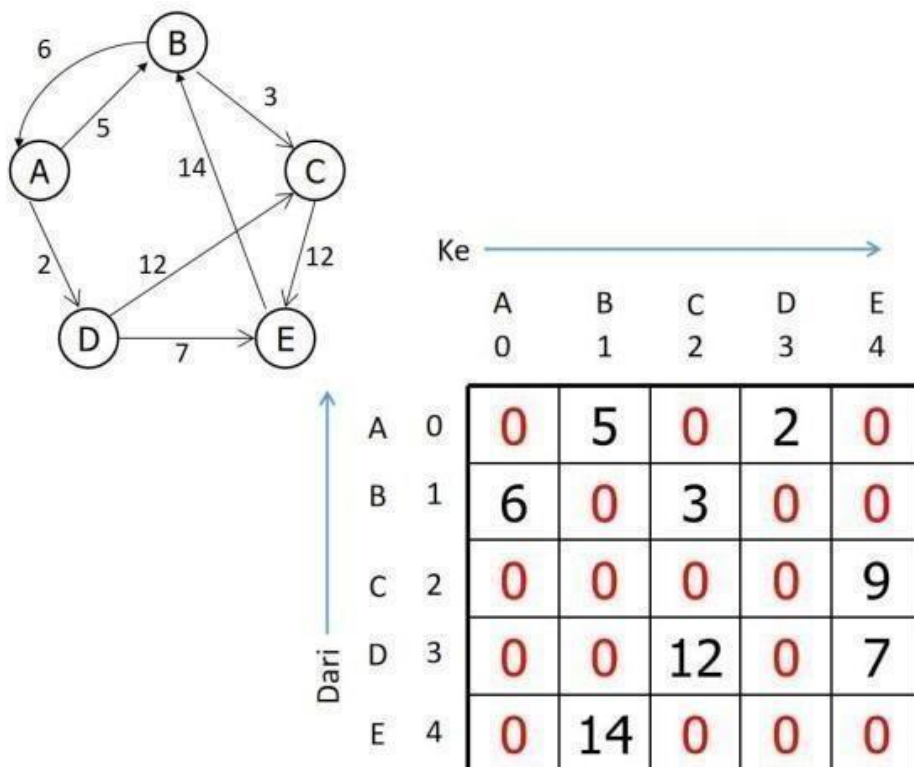
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph



- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
- Weight Graph :** Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph dengan Matriks



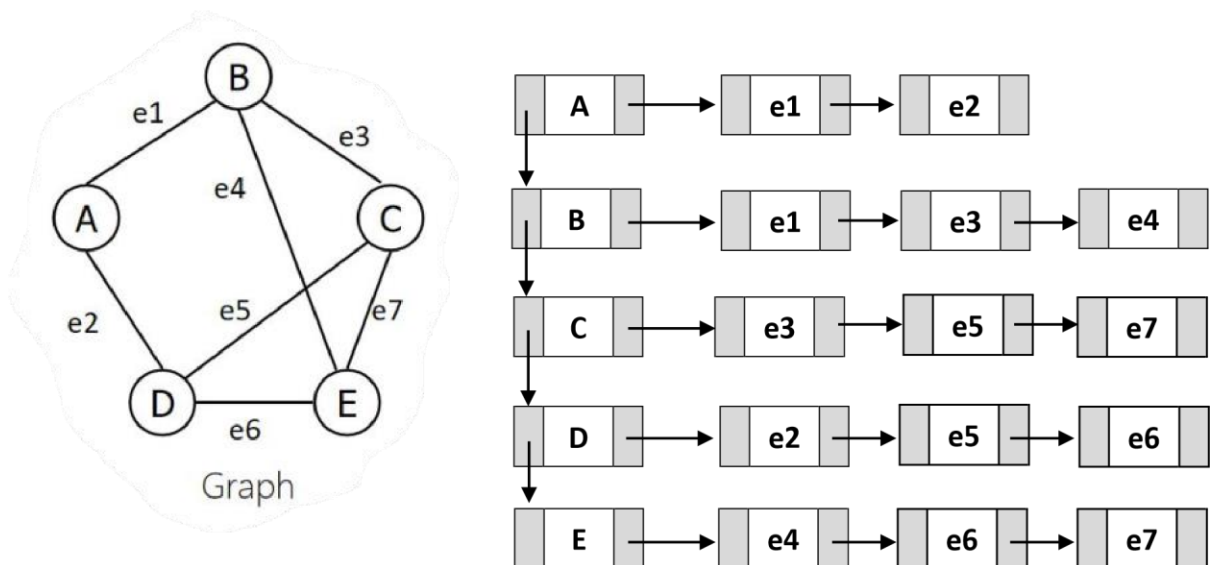
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

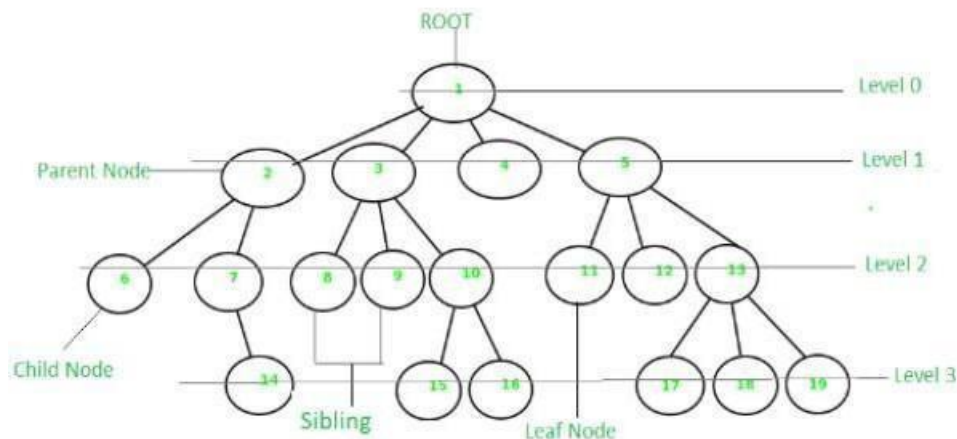
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

Dalam ilmu komputer, pohon/tree 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, dimana 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 hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



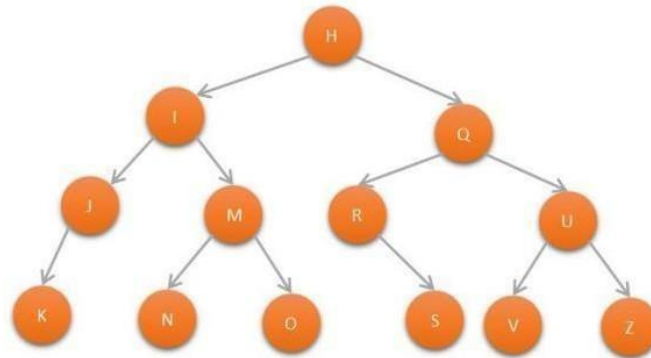
Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyarkan memiliki simpul satu level di bawahnya (child)

tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

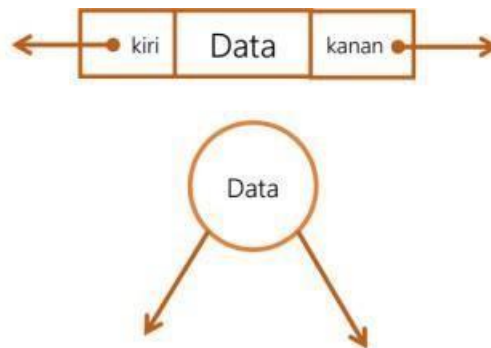
Gambar 1, menunjukkan contoh dari struktur data binary tree.



Gambar 1 Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{  
    char data;  
    pohon *kanan;  
    pohon *kiri;  
};  
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

- Create:** digunakan untuk membentuk binary tree baru yang masih kosong.
- Clear:** digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- isEmpty:** digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- Insert:** digunakan untuk memasukkan sebuah node kedalam tree.
- Find:** digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- 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.

1. Pre-Order

Penelusuran secara pre-order memiliki alur:

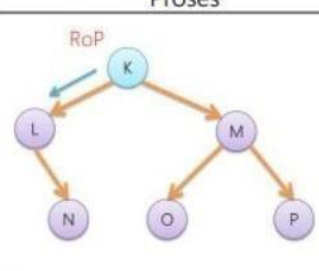
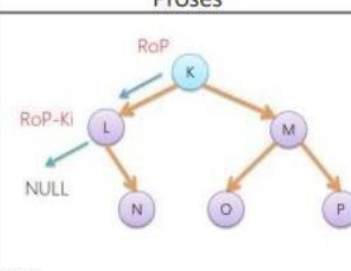
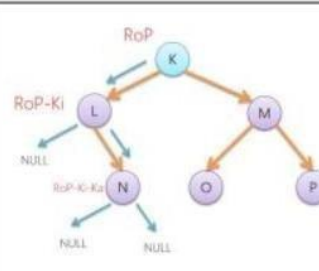
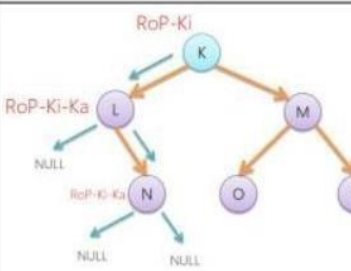
- a. Cetak data pada simpul root
- b. Secara rekursif mencetak seluruh data pada subpohon kiri
- c. Secara rekursif mencetak seluruh data pada subpohon kanan

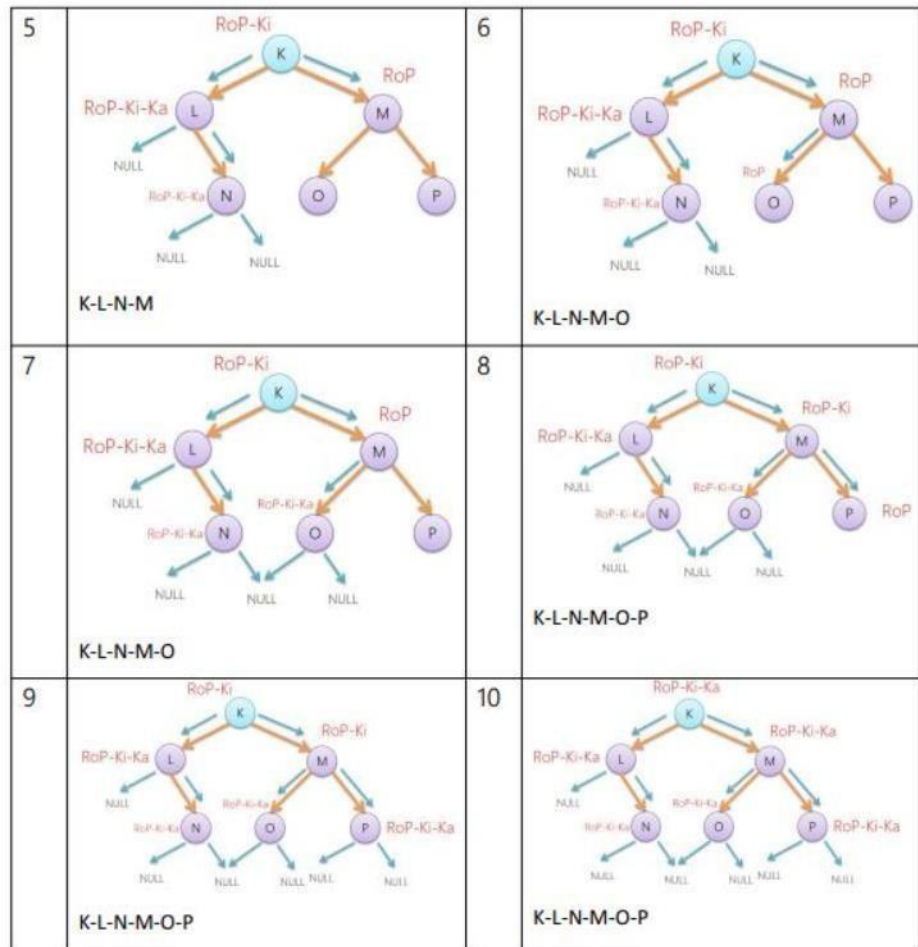
Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Alur pre-order

No	Proses	No	Proses
1	 <p>K</p>	2	 <p>K - L</p>
3	 <p>K-L-N</p>	4	 <p>K-L-N</p>



2. In-Order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Cetak data pada root
- Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

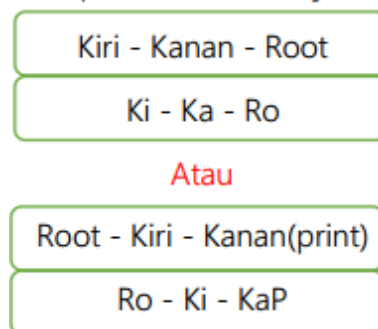
Ro - KiP - Ka

3. Post Order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan
- Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



C. GUIDED

Program Graph

```
#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 << " " << setw(15) << left
            << 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;}

```

Screenshot program :

Ciamis	: Bandung(7) Bekasi(8)	Pandia Arya Brata 2311102076
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)	Ln 2, Col 11 28 characters

Deskripsi program :

Berikut adalah penjelasan tentang program tersebut:

1. Program ini menggunakan library iostream untuk input/output stream dan library iomanip untuk mengatur tampilan output.
2. Program mendeklarasikan array simpul yang berisi nama-nama simpul yang ada dalam grafik. Terdapat 7 simpul yang didefinisikan dalam array ini. Nama-nama simpul tersebut adalah "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogyakarta".
3. Selanjutnya, program mendeklarasikan array busur yang merupakan matriks adjacency yang merepresentasikan hubungan antara simpul-simpul dalam grafik. Setiap elemen dari matriks ini menunjukkan bobot atau jarak antara dua simpul. Matriks busur ini memiliki ukuran 7x7 dan elemen-elemennya diinisialisasi dengan angka-angka tertentu yang menunjukkan jarak antara simpul-simpul tersebut. Jika nilai elemen adalah 0, itu berarti tidak ada busur yang menghubungkan dua simpul tersebut.

Program Tree

```
#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();
}

```

Screenshot program :

```
Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kananA
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kananB
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kananE
Node I berhasil ditambahkan ke child kiri G
Node J berhasil ditambahkan ke child kananG
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,

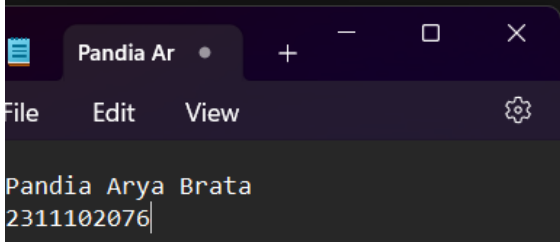
PostOrder :
D, I, J, G, H, E, B, F, C, A,

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
```



Deskripsi program :

Program di atas mengimplementasikan binary tree dengan struktur Pohon yang menyimpan karakter pada setiap node. Program ini memiliki fungsi untuk membuat node baru, menambahkan node anak kiri dan anak kanan, mengubah data pada node, melihat isi data pada node, mencari informasi tentang suatu node, dan melakukan penelusuran pohon dalam bentuk pre-order, in-order, dan post-order. Dalam contoh penggunaan program, binary tree dibangun dengan root 'A' dan beberapa node lainnya ditambahkan. Operasi seperti mengubah data node, menampilkan isi dan informasi node, serta penelusuran pohon dilakukan. Pada akhir program, subpohon yang dimulai dari node 'E' dihapus, dan karakteristik pohon ditampilkan kembali setelah penghapusan.

D. UNGUIDED

*Cantumkan NIM pada salah satu variabel di dalam program.

Contoh : int nama_22102003;

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

Source Code :

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

const int simpulMax = 10;
string simpul[simpulMax];
int busur[simpulMax][simpulMax];
int Pandia_2311102076;
void tampilGraph()
{
    cout << setw(10) << " ";
    for (int kolom = 0; kolom < Pandia_2311102076;
        kolom++)
    {
        cout << setw(10) << simpul[kolom];
    }
    cout << endl;
    for (int baris = 0; baris < Pandia_2311102076;
        baris++)
    {
        cout << setw(10) << simpul[baris];
        for (int kolom = 0; kolom < Pandia_2311102076;
            kolom++)
        {
            cout << setw(10) << busur[baris][kolom];
        }
        cout << endl;
    }
}

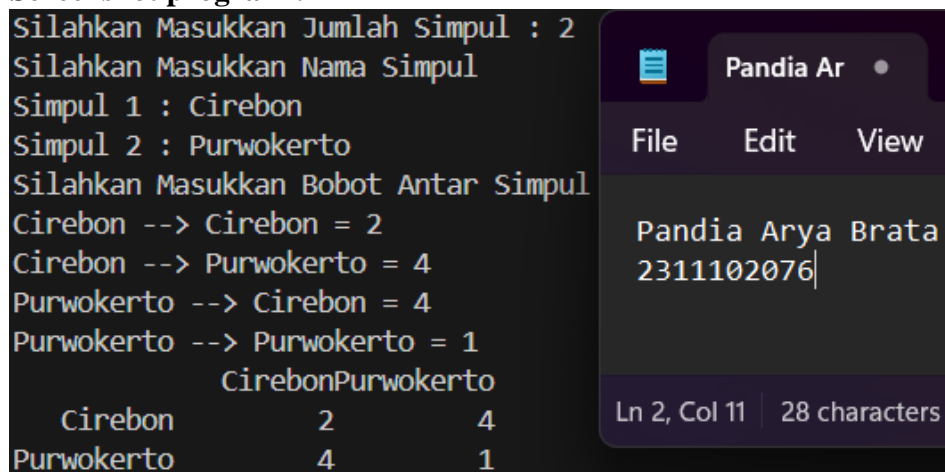
int main()
{
    cout << "Silahkan Masukkan Jumlah Simpul : ";
    cin >> Pandia_2311102076;
    if (Pandia_2311102076 > simpulMax || Pandia_2311102076 < 2)
    {
        cout << "Jumlah simpul tidak valid!" << endl;
        return 0;
    }
    cout << "Silahkan Masukkan Nama Simpul" << endl;
    for (int i = 0; i < Pandia_2311102076; i++)
    {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }
    cout << "Silahkan Masukkan Bobot Antar Simpul" << endl;
```

```

for (int baris = 0; baris < Pandia_2311102076;
    baris++)
{
    for (int kolom = 0; kolom < Pandia_2311102076;
        kolom++)
    {
        cout << simpul[baris] << " --> " << simpul[kolom]
            << " = ";
        cin >> busur[baris][kolom];
    }
}
tampilGraph();
return 0;
}

```

Screenshot program :



Deskripsi program :

1. Program ini menggunakan library iostream dan iomanip untuk input/output stream dan pengaturan tampilan output.
2. Terdapat deklarasi konstanta simpulMax yang menentukan ukuran maksimum array simpul dan array busur.
3. Program mendeklarasikan array simpul yang digunakan untuk menyimpan nama nama simpul dalam grafik. Ukuran array ini ditentukan oleh nilai variabel Pandia_2311102076 yang diinput oleh pengguna.
4. Selanjutnya, program mendeklarasikan array busur yang merupakan matriks adjacency yang merepresentasikan hubungan antara simpul-simpul dalam grafik. Elemen-elemen matriks ini akan diisi dengan bobot antar simpul yang diinput oleh pengguna.
5. Program selanjutnya mendefinisikan fungsi tampilGraph() yang bertujuan untuk menampilkan grafik ke layar. Fungsi ini menggunakan nested loop untuk mengakses setiap elemen matriks busur dan menampilkan simpul-simpul dan bobot antar simpul dalam format yang teratur.
6. Pada fungsi main(), program melakukan langkah-langkah berikut:
 - a. Meminta pengguna untuk memasukkan jumlah simpul yang diinginkan.
 - b. Memeriksa apakah jumlah simpul valid (antara 2 dan simpulMax). Jika tidak valid, program akan menampilkan pesan kesalahan dan berakhir.
 - c. Meminta pengguna untuk memasukkan nama-nama simpul.

- d. Meminta pengguna untuk memasukkan bobot antar simpul.
 - e. Memanggil fungsi tampilGraph() untuk menampilkan grafik yang telah dibuat berdasarkan input pengguna.
 - 7. Terakhir, program mengembalikan nilai 0 sebagai tanda bahwa program berakhir dengan sukses.
2. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

Source code :

```
#include <iostream>
#include <queue> // Untuk implementasi level-order traversal
using namespace std;

// Deklarasi Tree
struct Tree
{
    char pandia_2311102076;
    Tree *left, *right, *parent;
};

Tree *root, *baru;

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

// Cek Node
bool isEmpty()
{
    return root == NULL;
}

// Buat Node Baru
void buatNode(char pandia_2311102076)
{
    if (isEmpty())
    {
        root = new Tree();
        root->pandia_2311102076 = pandia_2311102076;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\nNode " << pandia_2311102076 << " berhasil dibuat
menjadi root." << endl;
    }
    else
    {

```

```

        cout << "\nTree sudah dibuat" << endl;
    }
}
// Tambah Kiri
Tree *insertLeft(char pandia_2311102076, Tree *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->pandia_2311102076 << "sudah
ada child kiri !" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Tree();
            baru->pandia_2311102076 = pandia_2311102076;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << pandia_2311102076 << " berhasil
ditambahkan ke child kiri " << baru->parent->pandia_2311102076
<< endl;
            return baru;
        }
    }
}
// Tambah Kanan
Tree *insertRight(char pandia_2311102076, Tree
*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->pandia_2311102076 << "sudah

```

```

ada child kanan !" << endl;
    return NULL;
}
else
{
    // kalau tidak ada
    baru = new Tree();
    baru->pandia_2311102076 = pandia_2311102076;
    baru->left = NULL;
    baru->right = NULL;
    baru->parent = node;
    node->right = baru;
    cout << "\n Node " << pandia_2311102076 << "berhasil
ditambahkan ke child kanan " << baru->parent->pandia_2311102076
    << endl;
    return baru;
}
}
}
// Ubah pandia_2311102076 Tree
void update(char pandia_2311102076, Tree *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->pandia_2311102076;
            node->pandia_2311102076 = pandia_2311102076;
            cout << "\n Node " << temp << " berhasil diubah menjadi
"
            << pandia_2311102076 << endl;
        }
    }
}
// Lihat Isi pandia_2311102076Tree
void retrieve(Tree *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else

```



```

        {
            cout << "\n pandia_2311102076 node : " << node->
pandia_2311102076 << endl;
        }
    }
}
// Cari pandia_2311102076 Tree
void find(Tree *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n pandia_2311102076 Node : " << node->
pandia_2311102076 << endl;
            cout << " Root : " << root->pandia_2311102076 << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->
pandia_2311102076
                << endl;
            if (node->parent != NULL && node->parent->left != node
&& node->parent->right == node)
                cout << " Sibling : " << node->parent->left->
pandia_2311102076 << endl;
            else if (node->parent != NULL && node->parent->right !=
node && node->parent->left == node)
                cout << " Sibling : " << node->parent->right->
pandia_2311102076 << 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->
pandia_2311102076 << endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child kanan)"
                << endl;
            else
                cout << " Child Kanan : " << node->right->
pandia_2311102076 << endl;
        }
    }
}
}

```

```

// Tampilkan Child Node
void showChildren(Tree *node)
{
    if (!node)
    {
        cout << "\nNode tidak ada!" << endl;
        return;
    }

    cout << "\nChild dari node " << node->pandia_2311102076 << ":
";
    if (node->left)
        cout << node->left->pandia_2311102076 << " ";
    if (node->right)
        cout << node->right->pandia_2311102076 << " ";
    cout << endl;
}

// Tampilkan Descendant Node (menggunakan level-order traversal)
void showDescendants(Tree *node)
{
    if (!node)
    {
        cout << "\nNode tidak ada!" << endl;
        return;
    }

    queue<Tree *> q;
    q.push(node);

    cout << "\nDescendant dari node " << node->pandia_2311102076 <<
": ";
    while (!q.empty())
    {
        Tree *current = q.front();
        q.pop();

        if (current != node)
        { // Jangan tampilkan node awal (node yang dicari
descendant-nya)
            cout << current->pandia_2311102076 << " ";
        }

        if (current->left)
            q.push(current->left);
        if (current->right)
            q.push(current->right);
    }
    cout << endl;
}

// preOrder
void preOrder(Tree *node)
{

```

```

    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->pandia_2311102076 << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Tree *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->pandia_2311102076 << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Tree *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->pandia_2311102076 << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Tree *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(Tree *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->pandia_2311102076 << "
berhasil dihapus." << endl;
    }
}

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

// Cek Size Tree
int size(Tree *node)
{
    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(Tree *node)
{
    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(root) << endl;
    cout << " Height Tree : " << height(root) << endl;
    cout << " Average Node of Tree : " << size(root) / height(root)
<< endl;
}

int main()
{
    init();
    char pandia_2311102076;
    cout << "Masukkan pandia_2311102076 untuk root: ";
    cin >> pandia_2311102076;
    buatNode(pandia_2311102076);
}

```

```

Tree *nodeB, *nodeC, *nodeD, *nodeE, *nodeF,
    *nodeG, *nodeH, *nodeI, *nodeJ;
cout << "Masukkan pandia_2311102076 untuk node B: ";
cin >> pandia_2311102076;
nodeB = insertLeft(pandia_2311102076, root);
cout << "Masukkan pandia_2311102076 untuk node C: ";
cin >> pandia_2311102076;
nodeC = insertRight(pandia_2311102076, root);
cout << "Masukkan pandia_2311102076 untuk node D: ";
cin >> pandia_2311102076;
nodeD = insertLeft(pandia_2311102076, nodeB);
cout << "Masukkan pandia_2311102076 untuk node E: ";
cin >> pandia_2311102076;
nodeE = insertRight(pandia_2311102076, nodeB);
cout << "Masukkan pandia_2311102076 untuk node F: ";
cin >> pandia_2311102076;
nodeF = insertLeft(pandia_2311102076, nodeC);
cout << "Masukkan pandia_2311102076 untuk node G: ";
cin >> pandia_2311102076;
nodeG = insertLeft(pandia_2311102076, nodeE);
cout << "Masukkan pandia_2311102076 untuk node H: ";
cin >> pandia_2311102076;
nodeH = insertRight(pandia_2311102076, nodeE);
cout << "Masukkan pandia_2311102076 untuk node I: ";
cin >> pandia_2311102076;
nodeI = insertLeft(pandia_2311102076, nodeG);
cout << "Masukkan pandia_2311102076 untuk node J: ";
cin >> pandia_2311102076;
nodeJ = insertRight(pandia_2311102076, nodeG);
cout << "Traversal PreOrder: ";
preOrder(root);
cout << endl;
cout << "Traversal InOrder: ";
inOrder(root);
cout << endl;
cout << "Traversal PostOrder: ";
postOrder(root);
cout << endl;
cout << "Karakteristik Tree:" << endl;
charateristic();
cout << endl;
cout << "Update node C: ";
update('Z', nodeC);
update('C', nodeC);
cout << endl;
cout << "Cari node C ";
find(nodeC);
cout << endl;
cout << "Hapus subtree G:" << endl;
deleteSub(nodeG);
cout << endl;
cout << "Traversal InOrder setelah menghapus subtree :";
inOrder(root);

```

```

    cout << endl;
    cout << "Hapus Tree" << endl;
    clear();
    return 0;
}

```

Screenshot program :

```

Node I berhasil dibuat menjadi root.
Masukkan pandia_2311102076 untuk node B:
Node N berhasil ditambahkan ke child kiri I
Masukkan pandia_2311102076 untuk node C:
Node F berhasil ditambahkan ke child kanan I
Masukkan pandia_2311102076 untuk node D:
Node O berhasil ditambahkan ke child kiri N
Masukkan pandia_2311102076 untuk node E:
Node R berhasil ditambahkan ke child kanan N
Masukkan pandia_2311102076 untuk node F:
Node M berhasil ditambahkan ke child kiri F
Masukkan pandia_2311102076 untuk node G:
Node A berhasil ditambahkan ke child kiri R
Masukkan pandia_2311102076 untuk node H:
Node T berhasil ditambahkan ke child kanan R
Masukkan pandia_2311102076 untuk node I:
Node I berhasil ditambahkan ke child kiri A
Masukkan pandia_2311102076 untuk node J:
Node K berhasil ditambahkan ke child kanan A
Traversal PreOrder: I, N, O, R, A, I, K, T, F, M,
Traversal InOrder: O, N, I, A, K, R, T, I, M, F,
Traversal PostOrder: O, I, K, A, T, R, N, M, F, I,
Karakteristik Tree:

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

Update node C:
Node F berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Cari node C
pandia_2311102076 Node : C
Root : I
Parent : I
Sibling : N
Child Kiri : M
Child Kanan : (tidak punya Child kanan)

Hapus subtree G:

Node subtree A berhasil dihapus.

Traversal InOrder setelah menghapus subtree : O, N, A, R, T, I, M, C,
Hapus Tree

Tree berhasil dihapus.

```

Deskripsi program :

Kode di atas mengimplementasikan struktur data pohon yang memungkinkan penyimpanan dan manipulasi data hierarkis. Kode ini membuat pohon, menambahkan node baru sebagai anak kiri atau kanan, memodifikasi data dalam node, menemukan informasi tentang node, dan menelusuri pohon (pre-order, in-order, post-order , dan tidak hanya tampilan). node atau keseluruhan pohon, tetapi juga menampilkan karakteristik pohon seperti ukuran dan tinggi. Khususnya, kode ini juga memiliki fungsi yang menampilkan anak langsung dari sebuah simpul (showChildren) dan semua keturunan simpul tersebut (showDescendants). Fungsi showChildren langsung mencetak anak kiri dan kanan, jika ada. Fungsi showDescendants menggunakan antrian untuk melakukan traversal tingkat-urutan sehingga semua node anak ditampilkan dalam urutan tingkat. Kode ini mendemonstrasikan penggunaan semua fungsi yang ditentukan dalam fungsi utama dengan membuat pohon, melakukan operasi seperti memperbarui dan menghapus, dan mencetak informasi tentang pohon.

KESIMPULAN

Dalam Praktikum Modul 9 ini kita mempelajari materi Graf & Tree. 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 $G = (V, E)$. Sedangkan pohon atau tree adalah struktur data yang menyerupai pohon nyata dan terdiri dari satu set node yang terhubung dan terurut. Setiap node memiliki paling banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu. Baik graf maupun pohon memiliki berbagai aplikasi yang luas, seperti jaringan sosial, pemetaan jalan, pemodelan data, dan penyimpanan data hierarki seperti pohon keluarga atau struktur organisasi. Program C++ untuk grafik (graph) memungkinkan pengguna untuk membuat, mengatur, dan menampilkan hubungan antara simpul-simpul dalam grafik berdasarkan input yang diberikan.

Program C++ untuk pohon (tree) memungkinkan pengguna untuk membangun, mengatur, dan menampilkan struktur hierarkis dari elemen-elemen dalam pohon berdasarkan input yang diberikan.

Kedua program ini menggunakan array, loop, dan kondisi untuk mengelola dan memvisualisasikan hubungan antara elemen-elemen dalam grafik dan pohon, sehingga mempermudah pemahaman dan manipulasi data dalam konteks struktur tersebut.

E. DAFTAR PUSTAKA

Karumanchi, N. (2011). Data Structures and Algorithms Made Easy: 700 Data Structure and Algorithmic Puzzles. CreateSpace.