

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 9
GRAPH DAN TREE**



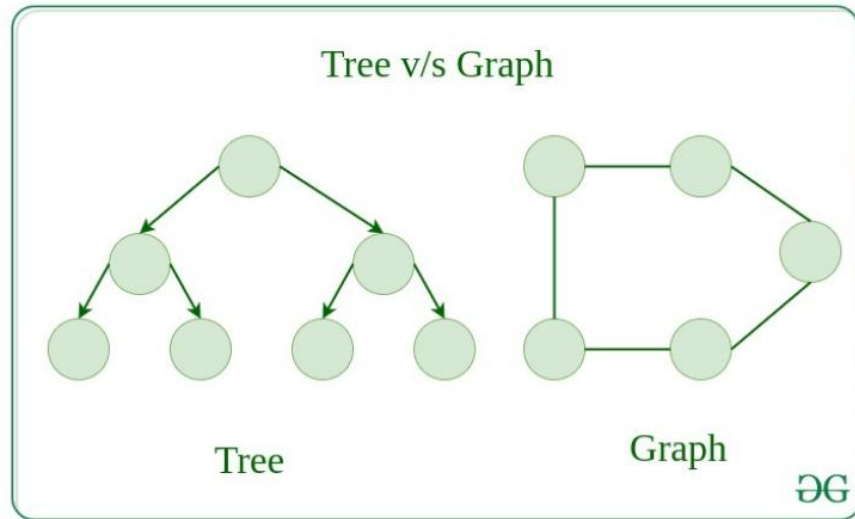
Disusun Oleh :
MARSHELY AYU ISWANTO
2311102073

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

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

A. Dasar Teori

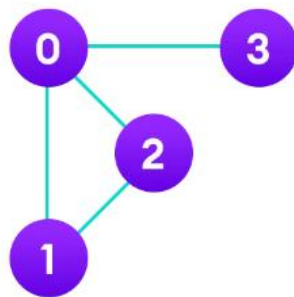
Grafik dan **pohon** adalah dua struktur data mendasar yang digunakan dalam ilmu komputer untuk merepresentasikan hubungan antar objek. Meskipun keduanya memiliki kesamaan, keduanya juga memiliki perbedaan yang membuatnya cocok untuk aplikasi yang berbeda.



Perbedaan Antara Grafik dan Pohon

1. Graph

Graph adalah jenis **struktur data** umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan **verteks (V)**, sedangkan sisi yang menghubungkan antar verteks disebut **edge (E)**. Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y. Sebagai contoh, terdapat graph seperti berikut:



Sumber: programiz.com

Graph di atas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan sebagai V, edge dilambangkan E, dan graph disimbolkan G, ilustrasi di atas dapat ditulis dalam notasi berikut:

$$V = \{0, 1, 2, 3\}$$

$$E = \{(0,1), (0,2), (0,3), (1,2)\}$$

$$G = \{V, E\}$$

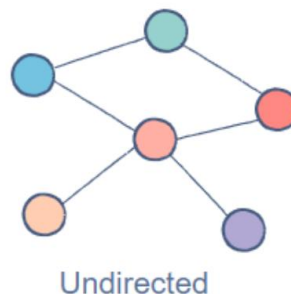
Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk)

Graph dapat dibedakan berdasarkan arah jelajahnya dan ada tidaknya label bobot pada relasinya.

Berdasarkan arah jelajahnya graph dibagi menjadi **Undirected graph** dan **Directed Graph**

a. Undirected Graph

Pada undirected graph, simpul-simpulnya terhubung dengan edge yang sifatnya dua arah. Misalnya kita punya simpul 1 dan 2 yang saling terhubung, kita bisa menjelajah dari simpul 1 ke simpul 2, begitu juga sebaliknya.

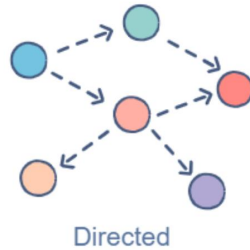


Undirected

Sumber: educative.io

b. Directed Graph

Kebalikan dari undirected graph, pada graph jenis ini simpul-simpulnya terhubung oleh edge yang hanya bisa melakukan jelajah satu arah pada simpul yang ditunjuk. Sebagai contoh jika ada simpul A yang terhubung ke simpul B, namun arah panahnya menuju simpul B, maka kita hanya bisa melakukan jelajah (traversing) dari simpul A ke simpul B, dan tidak berlaku sebaliknya.

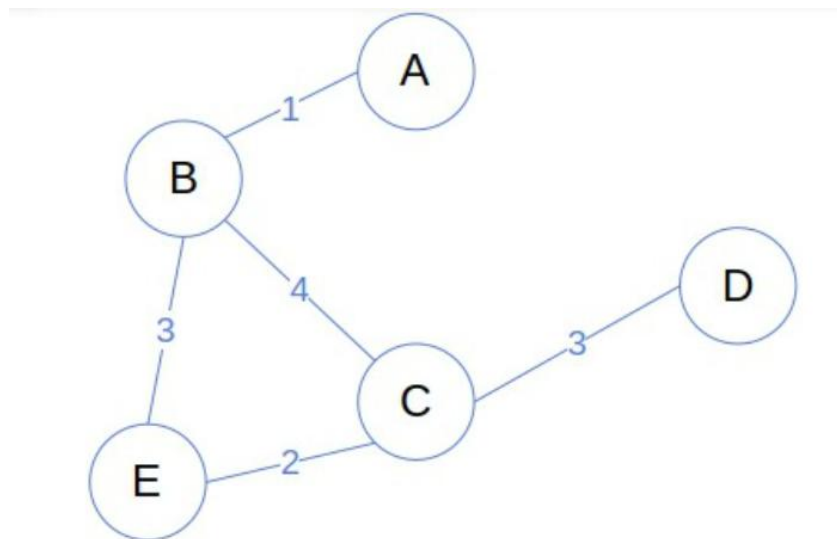


Sumber: educative.io

c. Weighted Graph

Weighted graph adalah jenis graph yang cabangnya diberi label bobot berupa bilangan numerik. Pemberian label bobot pada edge biasanya digunakan untuk memudahkan algoritma dalam menyelesaikan masalah.

Contoh implementasinya misalkan kita ingin menyelesaikan masalah dalam mencari rute terpendek dari lokasi A ke lokasi D, namun kita juga dituntut untuk mempertimbangkan kepadatan lalu lintas, panjang jalan dll. Untuk masalah seperti ini, kita bisa mengasosiasikan sebuah edge e dengan bobot $w(e)$ berupa bilangan ril.



Sumber: baeldung.com

Nilai bobot ini bisa apa saja yang relevan untuk masalah yang dihadapi: misalnya jarak, kepadatan, durasi, biaya, probabilitas, dan sebagainya.

d. Unweighted Graph

Berbeda dengan jenis sebelumnya, unweighted graph tidak memiliki properti bobot pada koneksinya. Graph ini hanya mempertimbangkan apakah dua node saling terhubung atau tidak.

2. Tree

Tree adalah tipe **struktur data** yang sifatnya **non-linier** dan berbentuk **hierarki**. Struktur data tree terdiri atas kumpulan simpul atau node dimana tiap-tiap simpul dari tree digunakan untuk menyimpan nilai dan sebuah list rujukan ke simpul lain yang disebut simpul anak atau child node.

Tiap-tiap simpul dari tree akan dihubungkan oleh sebuah garis hubung yang dalam istilah teknis disebut edge. Biasanya diimplementasikan menggunakan pointer.

Simpul pada tree bisa memiliki beberapa simpul anak (child node). Namun, jalan menuju sebuah child node hanya bisa dicapai melalui maksimal 1 node. Apabila sebuah node atau simpul tidak memiliki child node sama sekali maka dinamakan leaf node.

a. Istilah-istilah pada tree

1. Node

Node atau simpul adalah entitas pada struktur data tree yang mengandung sebuah nilai dan pointer yang menunjuk simpul di bawahnya (child node).

2. Child node

Child node atau simpul anak adalah simpul turunan dari simpul di atasnya.

3. Leaf Node

Leaf node atau simpul daun adalah simpul yang tidak memiliki child node dan merupakan node yang paling bawah dalam struktur data tree. Simpul ini biasa disebut juga sebagai external node

3. Root

Root atau akar adalah simpul teratas dari sebuah tree.

4. Internal node

Internal node adalah istilah untuk menyebut simpul yang memiliki minimal satu child node.

5. Edge

Edge merujuk pada garis yang menghubungkan antara dua buah simpul dalam tree. Jika sebuah tree memiliki N node maka tree tersebut akan memiliki (N-1) edge. Hanya ada satu jalur dari setiap simpul ke simpul lainnya.

6. Height of node

Height of node adalah jumlah edge dari sebuah node ke leaf node yang paling dalam.

7. Depth of node

Depth of node adalah banyaknya edge dari root ke sebuah node.

8. Height of tree

Height of tree dapat diartikan sebagai panjang jalur terpanjang dari simpul akar ke simpul daun dari sebuah tree.

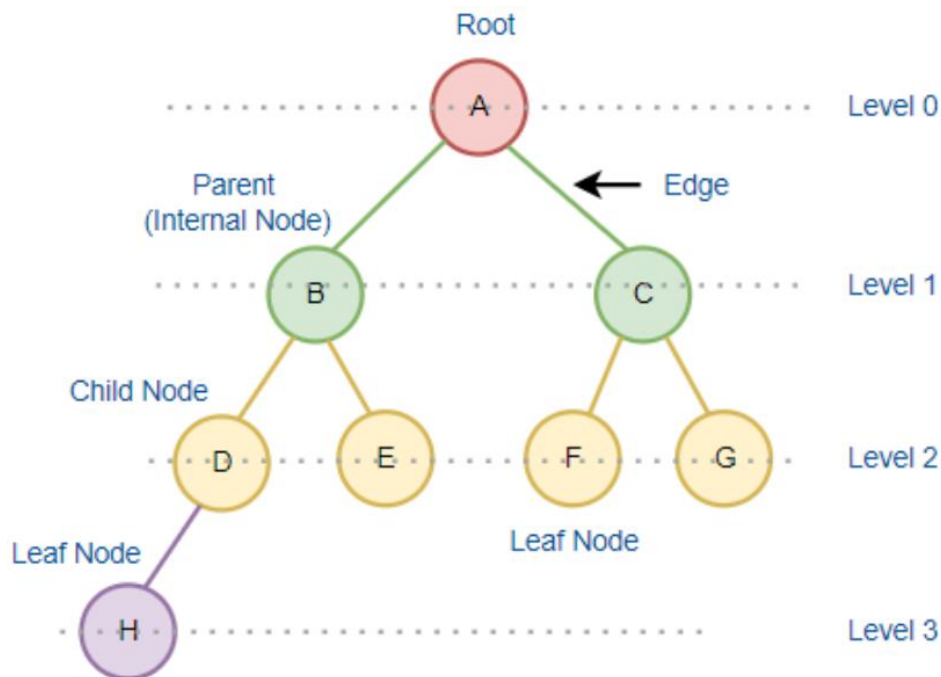
9. Degree of node

Jumlah cabang yang melekat pada simpul disebut Degree of node atau derajat simpul. Derajat simpul pada sebuah leaf node adalah 0.

Selain Degree of node, terdapat juga Degree of tree yaitu derajat maksimum simpul di antara semua simpul pada tree.

10. Subtree

Subtree adalah setiap simpul dari tree beserta turunannya



b. Jenis-jenis Tree

Struktur data tree dapat diklasifikasikan ke dalam 4 jenis, yaitu: **General tree**, **Binary tree**, **Balanced tree**, dan **Binary search tree**.

1. General tree

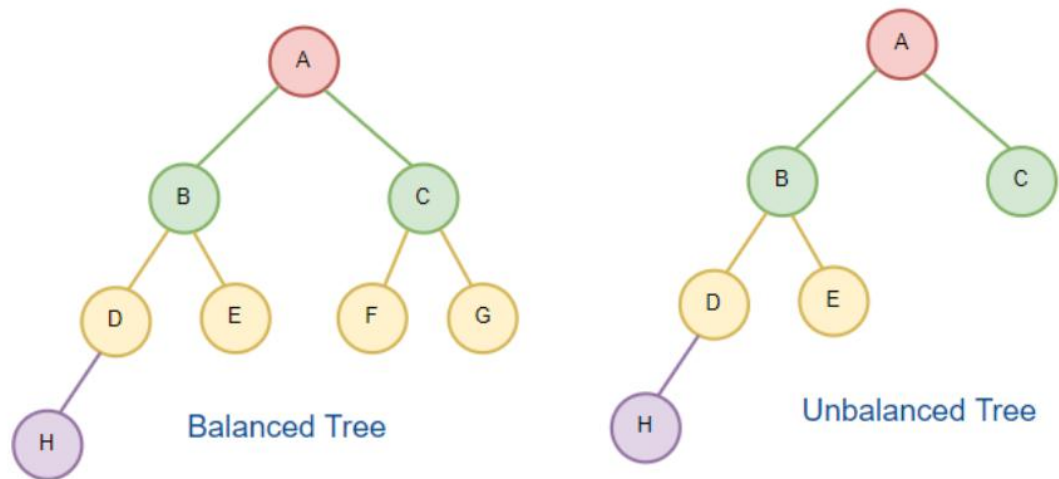
Struktur data tree yang tidak memiliki batasan jumlah node pada hierarki tree disebut General tree. Setiap simpul atau node bebas memiliki berapapun child node. Tree jenis adalah superset dari semua jenis tree.

2. Binary tree

Binary tree adalah jenis tree yang simpulnya hanya dapat memiliki paling banyak 2 simpul anak (child node). Kedua simpul tersebut biasa disebut simpul kiri (left node) dan simpul kanan (right node). Tree tipe ini lebih populer daripada jenis lainnya.

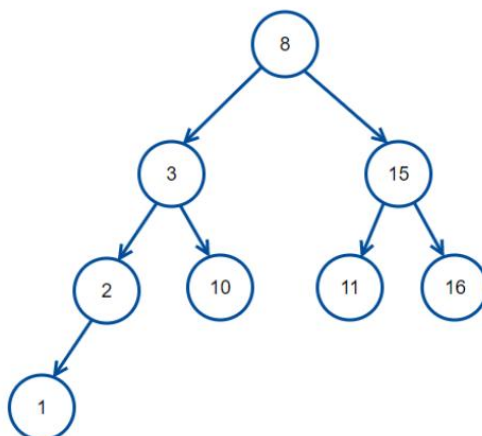
3. Balanced tree

Apabila tinggi dari subtree sebelah kiri dan subtree sebelah kanan sama atau kalaupun berbeda hanya berbeda 1, maka disebut sebagai balanced tree.



4. Binary search tree

Sesuai dengan namanya, Binary search tree digunakan untuk berbagai algoritma pencarian dan pengurutan. Contohnya seperti AVL tree dan Red-black tree. Struktur data tree jenis ini memiliki nilai pada simpul sebelah kiri lebih kecil daripada induknya. Sedangkan nilai simpul sebelah kanan lebih besar dari induknya.



B. Guided

Guided 1

Source Code

```
#include <iostream>
#include <iomanip>

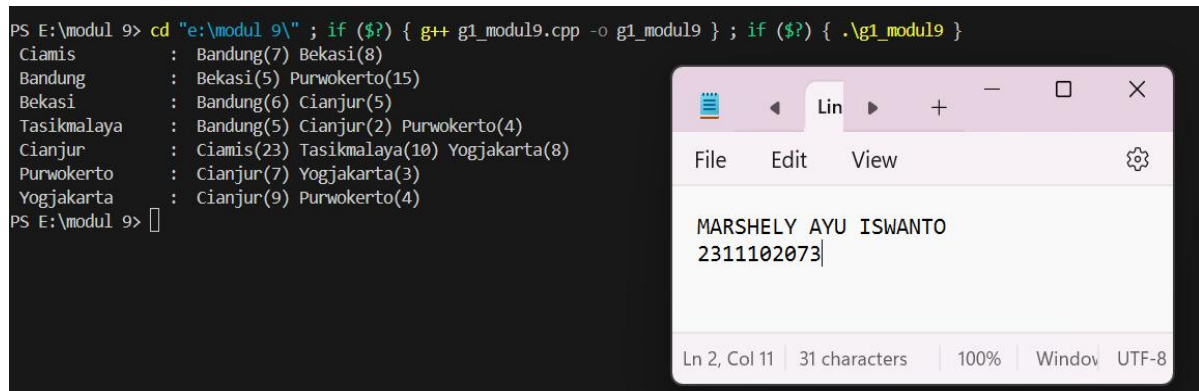
using namespace std;

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



The screenshot shows two windows. The terminal window on the left displays the output of a C++ program, listing cities and their connections with weights. The text editor window on the right shows the source code of the program, which is a binary tree structure.

```
PS E:\modul 9> cd "e:\modul 9\" ; if ($?) { g++ g1_modul9.cpp -o g1_modul9 } ; if ($?) { .\g1_modul9 }
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 E:\modul 9> 
```

```
File Edit View
MARSHELY AYU ISWANTO
2311102073
Ln 2, Col 11 | 31 characters | 100% | Window UTF-8
```

Deskripsi Program:

Kode program di atas ditulis dalam bahasa C++ dan bertujuan untuk menampilkan representasi graf berbobot dari tujuh kota di Jawa Barat dan sekitarnya. Terdapat dua array: `simpul`, yang berisi nama-nama kota, dan `busur`, sebuah matriks yang merepresentasikan bobot (misalnya jarak atau biaya) antar kota-kota tersebut. Fungsi `tampilGraph` digunakan untuk mencetak graf ke layar dengan format yang mudah dibaca. Dalam fungsi ini, setiap baris merepresentasikan sebuah kota dan diikuti oleh kota-kota tujuan yang terhubung dengannya beserta bobot busurnya. Program utama (`main`) hanya memanggil fungsi `tampilGraph` untuk menampilkan informasi graf tersebut.

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();
}

```

Screenshoots Output

```
PS E:\modul 9> cd "e:\modul 9\" ; if ($?) { g++ g2_modul9.cpp -o g2_modul9 } ; if ($?) { .\g2_modul9 }
```

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

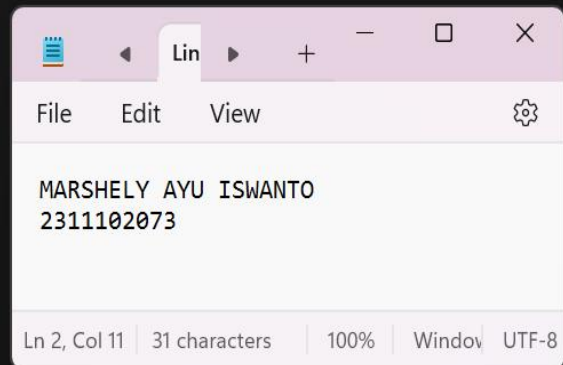
Root : A

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,

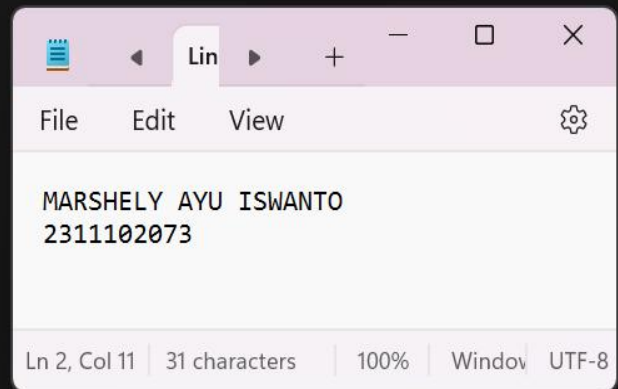
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS E:\modul 9>

```



Deskripsi Program

Kode program tersebut merupakan implementasi pohon biner dalam bahasa C++. Program ini mencakup beberapa fungsi untuk manipulasi pohon biner, seperti inisialisasi, pengecekan apakah pohon kosong, pembuatan node baru, penambahan anak kiri dan kanan, pengubahan data node, pengambilan data node, pencarian node, penelusuran (preorder, inorder, postorder), serta penghapusan node atau subtree. Program juga menghitung karakteristik pohon seperti ukuran dan tinggi. Dalam `main`, program membuat pohon biner dengan root 'A', menambahkan beberapa node, mengubah data pada node, menampilkan data node, melakukan penelusuran preorder, inorder, dan postorder, serta menampilkan karakteristik pohon. Terakhir, program menghapus subtree dari node 'E' dan menampilkan kembali pohon dalam preorder serta karakteristik pohon setelah penghapusan subtree.

C. Unguided

Unguided 1

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

Output Program Source Code

```

#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> jumlah_simpul;
}

```

```

        vector<string>
marshelyayuiswanto_2311102073(jumlah_simpul);
        vector<vector<int>> bobot(jumlah_simpul,
                                   vector<int>(jumlah_simpul));
        for (int i = 0; i < jumlah_simpul; ++i)
        {
            cout << "Silahkan masukkan nama simpul " << i + 1
<< " : ";
                cin >>
                marshelyayuiswanto_2311102073[i];
        }
        cout << "Silahkan masukkan bobot antar simpul\n";
        for (int i = 0; i < jumlah_simpul; ++i)
        {
            for (int j = 0; j < jumlah_simpul; ++j)
            {
                cout << marshelyayuiswanto_2311102073[i] <<
"-->" << marshelyayuiswanto_2311102073[j] << " : ";
                cin >> bobot[i][j];
            }
        }
        cout << "\n\t";
        for (int i = 0; i < jumlah_simpul; ++i)
        {
            cout << marshelyayuiswanto_2311102073[i] << "\t";
        }
        cout << "\n";
        for (int i = 0; i < jumlah_simpul; ++i)
        {
            cout << marshelyayuiswanto_2311102073[i] << "\t";
            for (int j = 0; j < jumlah_simpul; ++j)
            {
                cout << bobot[i][j] << "\t";
            }
            cout << "\n";
        }
        return 0;
}

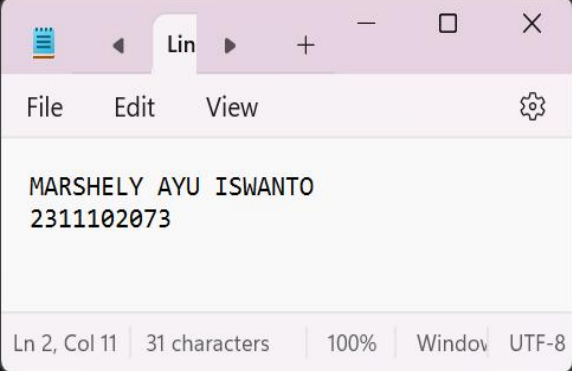
```

Screenshots Output

```
PS E:\modul 9> cd "e:\modul 9\" ; if ($?) { g++ ug1_modul9.cpp -o ug1_modul9 } ; if ($?) { .\ug1_modul9 }
Silahkan masukkan jumlah simpul : 3
Silahkan masukkan nama simpul 1 : Surabaya
Silahkan masukkan nama simpul 2 : Wonokromo
Silahkan masukkan nama simpul 3 : Sidoarjo
Silahkan masukkan bobot antar simpul
Surabaya-->Surabaya : 0
Surabaya-->Wonokromo : 1
Surabaya-->Sidoarjo : 2
Wonokromo-->Surabaya : 1
Wonokromo-->Wonokromo : 0
Wonokromo-->Sidoarjo : 1
Sidoarjo-->Surabaya : 2
Sidoarjo-->Wonokromo : 1
Sidoarjo-->Sidoarjo : 0
```

	Surabaya	Wonokromo	Sidoarjo
Surabaya	0	1	2
Wonokromo	1	0	1
Sidoarjo	2	1	0

```
PS E:\modul 9> 
```



Deskripsi Program :

Kode program tersebut adalah implementasi sederhana dari graf berbobot dalam bahasa C++ yang menggunakan matriks ketetanggaan untuk menyimpan bobot antar simpul. Program meminta pengguna untuk memasukkan jumlah simpul dalam graf, kemudian mengumpulkan nama-nama simpul tersebut. Setelah itu, program meminta pengguna untuk memasukkan bobot antar setiap pasangan simpul. Bobot ini disimpan dalam matriks dua dimensi. Akhirnya, program mencetak matriks ketetanggaan yang menampilkan bobot antar simpul, di mana baris dan kolom matriks di-label dengan nama-nama simpul yang telah dimasukkan pengguna.

Unguided 2

Buatlah sebuah program yang dapat menghitung banyaknya huruf vocal dalam sebuah kalimat!

```
#include <iostream>
#include <vector>
using namespace std;
// Declaring the Tree structure
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root = nullptr;
// Initialize the tree
void init()
{
    root = NULL;
}
```

```

// Check if the tree is empty
int isEmpty()
{
    return (root == NULL) ? 1 : 0;
}
// Create a new node
Pohon *buatNode(char data)
{
    Pohon *newNode = new Pohon();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->parent = NULL;
    // cout << "\nNode " << data << " berhasil dibuat." <<
endl;
    return newNode;
}
// Insert a node to the left
Pohon *insertLeft(Pohon *parent, Pohon *child)
{
    if (isEmpty() == 1)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (parent->left != NULL)
        {
            cout << "\nNode " << parent->left->data << "
sudah ada child kiri !" << endl;
            return NULL;
        }
        else
        {
            child->parent = parent;
            parent->left = child;
            // cout << "\nNode " << child->data << "
berhasil ditambahkan ke child kiri " << child->parent-
>data << endl;
            return child;
        }
    }
}
// Insert a node to the right
Pohon *insertRight(Pohon *parent, Pohon *child)
{
    if (root == NULL)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
}

```

```

    }
    else
    {
        if (parent->right != NULL)
        {
            cout << "\nNode " << parent->right->data << "
sudah ada child kanan !" << endl;
            return NULL;
        }
        else
        {
            child->parent = parent;
            parent->right = child;
            // cout << "\nNode " << child->data << "
berhasil ditambahkan ke child kanan " << child->parent-
>data << endl;
            return child;
        }
    }
}
// Update node data
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        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;
        }
    }
}
// Retrieve node data
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)

```

```

        cout << "\nNode yang ditunjuk tidak ada!" <<
endl;
    else
    {
        cout << "\nData node : " << node->data <<
endl;
    }
}
// Find node and display its properties
void find(Pohon *node)
{
    if (!root)
    {
        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->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;
    }
}
// Pre-order traversal
void preOrder(Pohon *node)
{
    if (node != NULL)
    {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}
// In-order traversal
void inOrder(Pohon *node)
{
    if (node != NULL)
    {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}
// Post-order traversal
void postOrder(Pohon *node)
{
    if (node != NULL)
    {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}
// Delete the entire tree
void deleteTree(Pohon *node)
{
    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;
        }
    }
}
// Delete a subtree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << "
berhasil dihapus." << endl;
    }
}
// Clear the entire tree
void clear()
{
    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}
// Get the size of the tree
int size(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        return 1 + size(node->left) + size(node->right);
    }
}
// Get the height of the tree
int height(Pohon *node)
{

```

```

        if (node == NULL)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            return (heightKiri >= heightKanan) ? heightKiri +
1 : heightKanan + 1;
        }
    }
    // Display tree characteristics
    void charateristic()
    {
        cout << "\nSize Tree : " << size(root) << endl;
        cout << "Height Tree : " << height(root) << endl;
        cout << "Average Node of Tree : " << (size(root) /
(float)height(root)) << endl;
    }
    int main()
    {
        root = buatNode('A');
        int menu, part, part2;
        char marshelyayuiswanto_2311102073;
        vector<Pohon *> nodes;
        nodes.push_back(buatNode('B'));
        nodes.push_back(buatNode('C'));
        nodes.push_back(buatNode('D'));
        nodes.push_back(buatNode('E'));
        nodes.push_back(buatNode('F'));
        nodes.push_back(buatNode('G'));
        nodes.push_back(buatNode('H'));
        nodes.push_back(buatNode('I'));
        nodes.push_back(buatNode('J'));
        insertLeft(root, nodes[0]);
        insertRight(root, nodes[1]);
        insertLeft(nodes[0], nodes[2]);
        insertRight(nodes[0], nodes[3]);
        insertLeft(nodes[1], nodes[4]);
        insertLeft(nodes[3], nodes[5]);
        insertRight(nodes[3], nodes[6]);
        insertLeft(nodes[5], nodes[7]);
        insertRight(nodes[5], nodes[8]);
        do
        {
            cout << "\n----- PROGHRAM GRAPH ----- \n"
                << "1. Tambah node\n"
                << "2. Tambah di kiri\n"
                << "3. Tambah di kanan\n"
                << "4. Lihat karakteristik tree\n"

```

```

        "5. Lihat isi data tree\n"
        "6. Cari data tree\n"
        "7. Penelurusan (Traversal) preOrder\n"
        "8. Penelurusan (Traversal) inOrder\n"
        "9. Penelurusan (Traversal) postOrder\n"
        "10. Hapus subTree\n"
        "0. KELUAR\n"
        "\nPilih : ";
    cin >> menu;
    cout << "-----Running Command...\n";
    switch (menu)
    {
    case 1:
        cout << "\n Nama Node (Character) : ";
        cin >> marshelyayuiswanto_2311102073;

        nodes.push_back(buatNode( marshelyayuiswanto_2
311102073));
        break;
    case 2:
        cout << "\nMasukkan nomor untuk node parent :
";

        cin >> part;
        cout << "\nMasukkan nomor untuk node child :
";

        cin >> part2;
        insertLeft(nodes[part], nodes[part2]);
        break;
    case 3:
        cout << "\nMasukkan nomor untuk node parent :
";

        cin >> part;
        cout << "\nMasukkan nomor untuk node child :
";

        cin >> part2;
        insertRight(nodes[part], nodes[part2]);
        break;
    case 4:
        charateristic();
        break;
    case 5:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        retrieve(nodes[part]);
        break;
    case 6:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        find(nodes[part]);
        break;
    case 7:

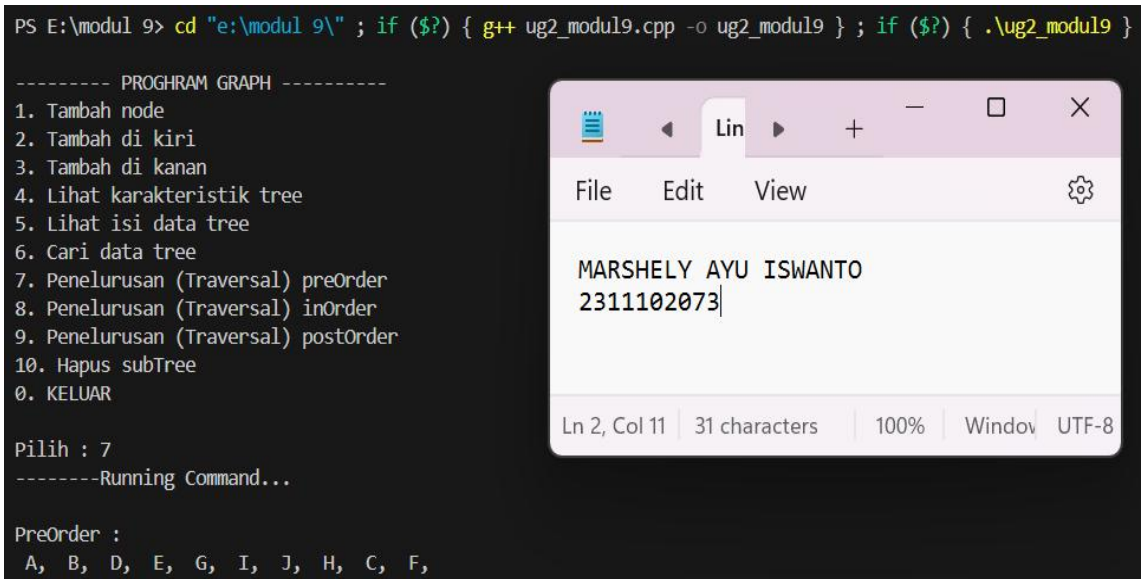
```

```

        cout << "\nPreOrder :" << endl;
        preOrder(root);
        cout << "\n"
             << endl;
        break;
    case 8:
        cout << "\nInOrder :" << endl;
        inOrder(root);
        cout << "\n"
             << endl;
        break;
    case 9:
        cout << "\nPostOrder :" << endl;
        postOrder(root);
        cout << "\n"
             << endl;
        break;
    case 10:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        deleteSub(nodes[part]);
        break;
    default:
        break;
    }
} while (menu != 0);
}

```

Screenshoots Output



The screenshot shows a terminal window with the following content:

```

PS E:\modul 9> cd "e:\modul 9\" ; if ($?) { g++ ug2_modul9.cpp -o ug2_modul9 } ; if ($?) { .\ug2_modul9 }

----- PROGRAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelusuran (Traversal) preOrder
8. Penelusuran (Traversal) inOrder
9. Penelusuran (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 7
-----Running Command...

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

```

Overlaid on the terminal is a text editor window titled "Lin" with a menu bar (File, Edit, View) and a settings icon. The text in the editor reads:

```

MARSHELY AYU ISWANTO
2311102073|

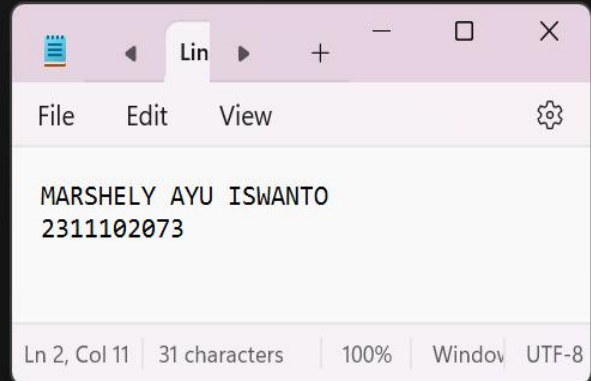
```

The status bar at the bottom of the text editor shows "Ln 2, Col 11 | 31 characters | 100% | Window | UTF-8".

```
----- PROGHAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 8
-----Running Command...

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



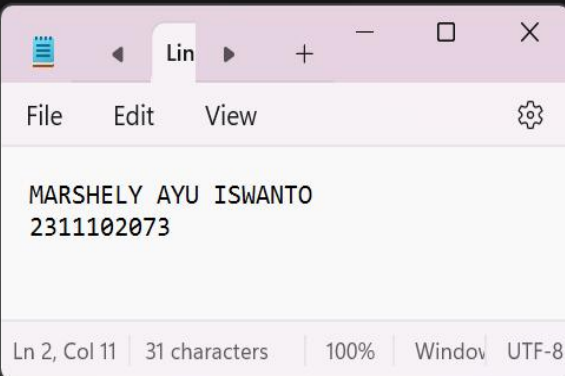
```
----- PROGHAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 9
-----Running Command...

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

----- PROGHAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 0
-----Running Command...
PS E:\modul 9> 
```



Deskripsi Program

Kode program tersebut adalah implementasi dari struktur data pohon biner dalam bahasa C++. Program ini memungkinkan pengguna untuk membuat dan mengelola pohon biner dengan berbagai operasi seperti menambah node, mengupdate node, melakukan traversal (preorder, inorder, postorder), mencari node, melihat karakteristik

pohon (ukuran, tinggi, rata-rata node), dan menghapus subtree. Program ini menggunakan struktur data 'Pohon' untuk merepresentasikan node dengan atribut data, pointer ke child kiri, child kanan, dan parent. Node dapat ditambahkan ke pohon menggunakan fungsi 'insertLeft' dan 'insertRight', sementara fungsi lain seperti 'update', 'retrieve', dan 'find' memberikan kemampuan untuk mengubah dan memeriksa nilai node. Program ini menyediakan antarmuka menu untuk pengguna sehingga pengguna dapat berinteraksi dengan pohon secara mudah melalui input di terminal.

Kesimpulan

Pohon dan graf adalah struktur data esensial yang digunakan untuk merepresentasikan dan memanipulasi hubungan antar data. Pohon adalah struktur hierarkis dengan satu root dan child nodes yang dibatasi menjadi maksimal dua untuk pohon biner. Implementasi pohon dalam C++ menggunakan struktur dengan pointer untuk child kiri, kanan, dan parent, serta fungsi-fungsi dasar seperti inisialisasi, penambahan, pengubahan, penghapusan node, dan traversal (preorder, inorder, postorder). Pohon digunakan dalam sistem file, struktur data dinamis, parsing ekspresi, dan routing jaringan.

Graf adalah struktur data yang lebih umum yang terdiri dari simpul (nodes) dan sisi (edges) yang dapat berbentuk tidak terarah atau terarah. Implementasi graf dalam C++ menggunakan adjacency matrix, adjacency list, atau edge list. Operasi pada graf termasuk traversal (DFS, BFS), pencarian jalur terpendek (Dijkstra), dan deteksi siklus. Graf digunakan untuk berbagai aplikasi seperti jaringan komputer, peta jalan, analisis jaringan sosial, dan optimasi rute. Kedua struktur ini memungkinkan representasi yang efisien dan manipulasi hubungan kompleks antar data.

D. Referensi

Graf (Graph) dan Pohon (Tree) pada C++. (2019, Mei 31). From <https://ahmadhadari77.blogspot.com/>
<https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html>

Periksa apakah grafik yang diberikan adalah pohon atau bukan. (2023, Februari 06). From <https://www.geeksforgeeks.org/>: <https://www.geeksforgeeks.org/check-given-graph-tree/>

SHARMA, A. (2022, September 14). *Data Structures in C++ - Trees & Graph*. From <https://www.prepbytes.com/>: <https://www.prepbytes.com/blog/tree/data-structures-in-c-trees-graph-3/>

<https://www.trivusi.web.id/2022/07/struktur-data-graph.html>

<https://www.trivusi.web.id/2022/07/struktur-data-tree.html>

