

**LAPORAN PRAKTIKUM STRUKTUR  
DATA DAN ALGORITMA**

**MODUL IX  
GRAPH DAN TREE**



**Disusun Oleh :**

NAMA : Ahmad Titana Nanda Pramudya  
NIM : 2311102042

**Dosen**

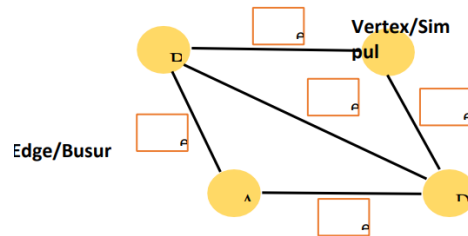
Wahyu Andi Saputra, S.pd., M,Eng

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

## A. DASAR TEORI

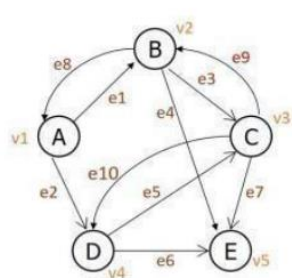
### 1. Graph

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:

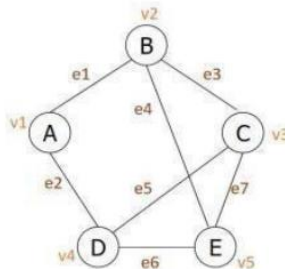


Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

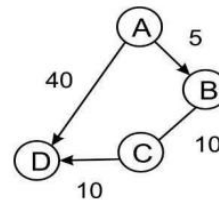
### Jenis-jenis Graph



Directed Graph



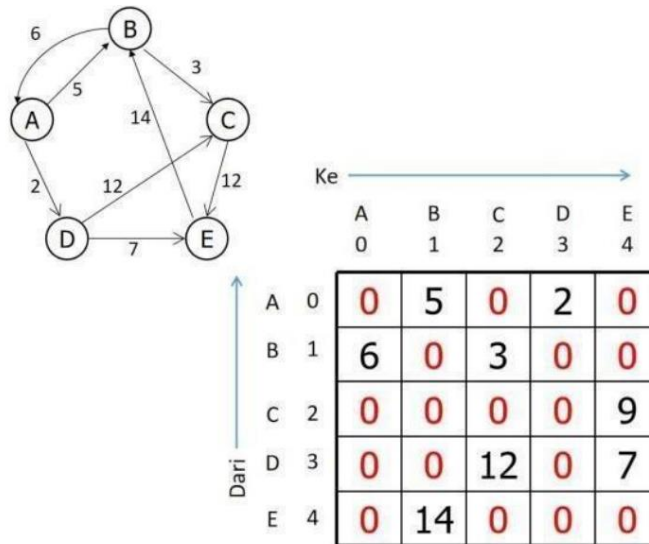
Undirected Graph



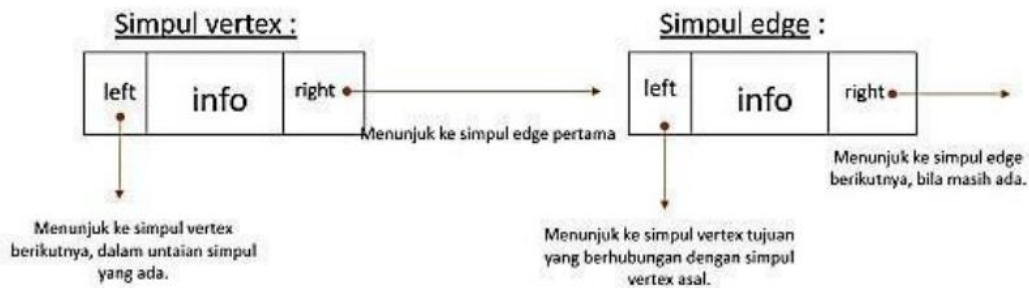
Weight Graph

- Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah  $e_1$  sedangkan busur BA adalah  $e_8$ .
- Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur  $e_1$  dapat disebut busur AB atau BA.
- Weight Graph : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph dengan Matriks

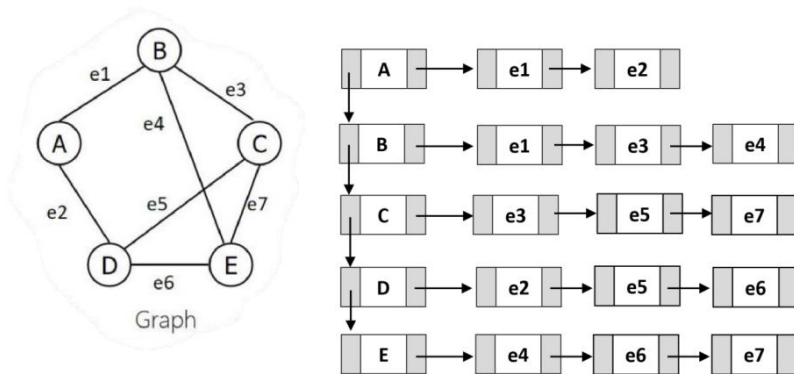


## Representasi dengan Linked List



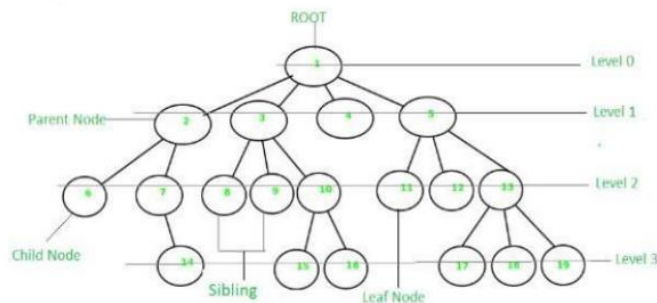
Gambar 5 Representasi Graph dengan Linked List

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.



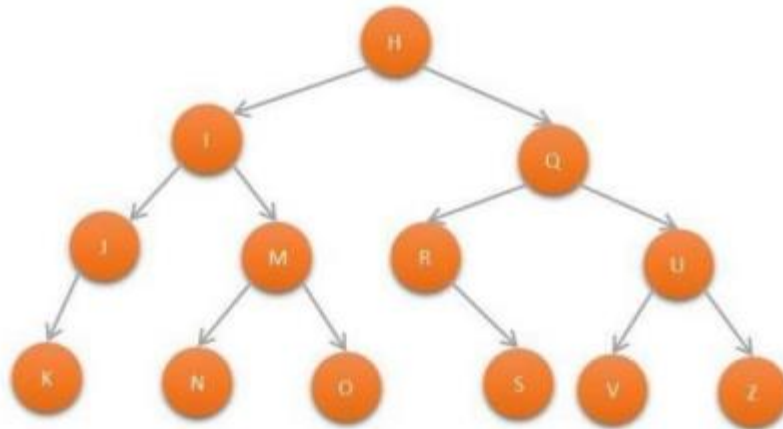
## 2. Tree atau Pohon

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 :



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

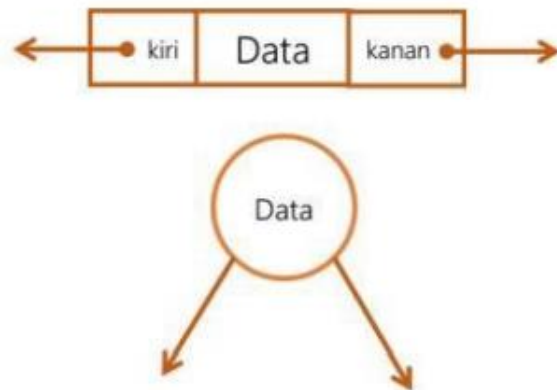
Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



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;
  
```



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

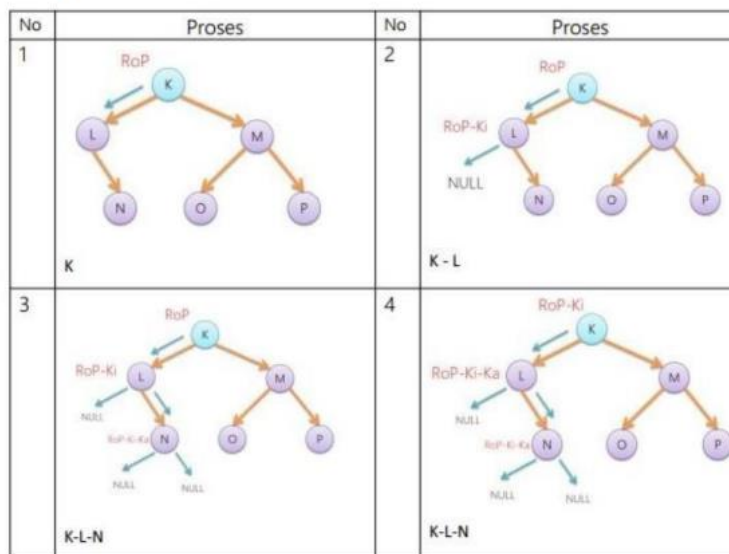
Penelusuran secara pre-order memiliki alur:

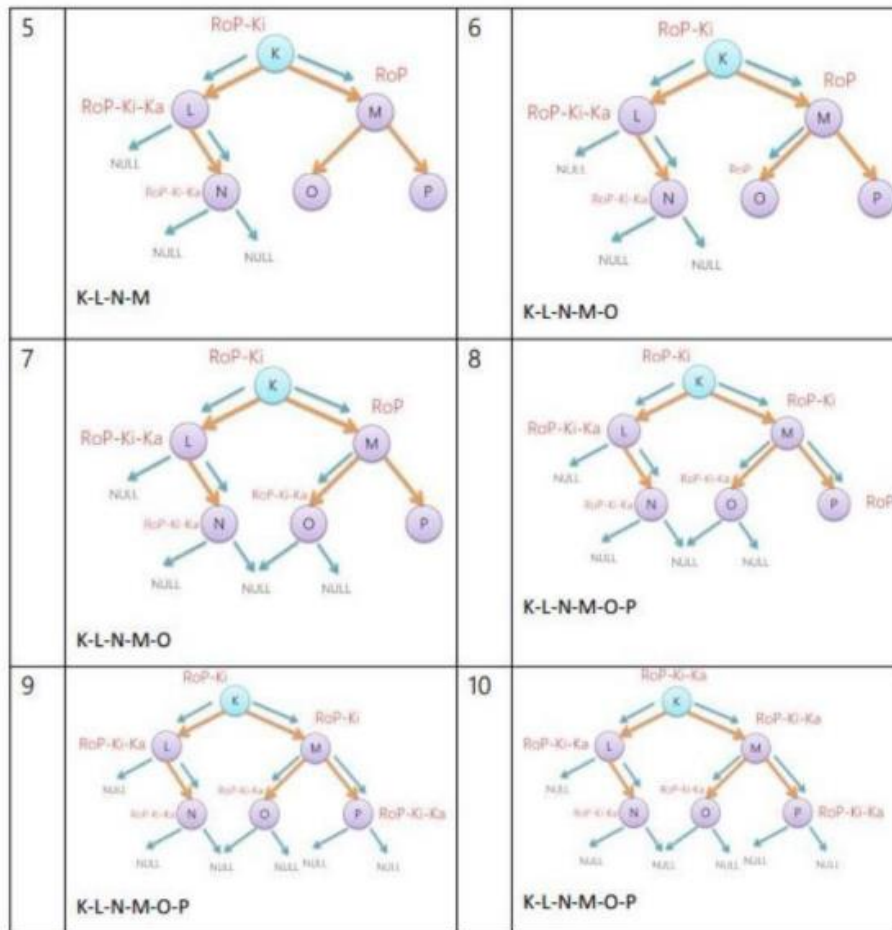
- a. Cetak data pada simpul root
- b. Secara rekursif mencetak seluruh data pada subpohon kiri
- c. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan
RoP - Ki - Ka

#### Alur pre-order





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:

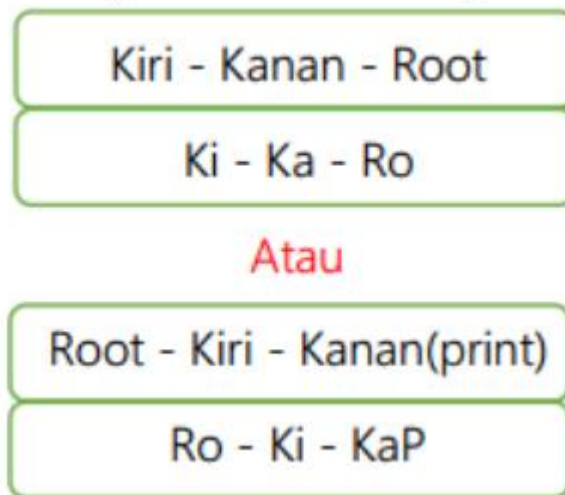


### 3. Post Order

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:





## B. Guided

### Guided 1

Sourcode :

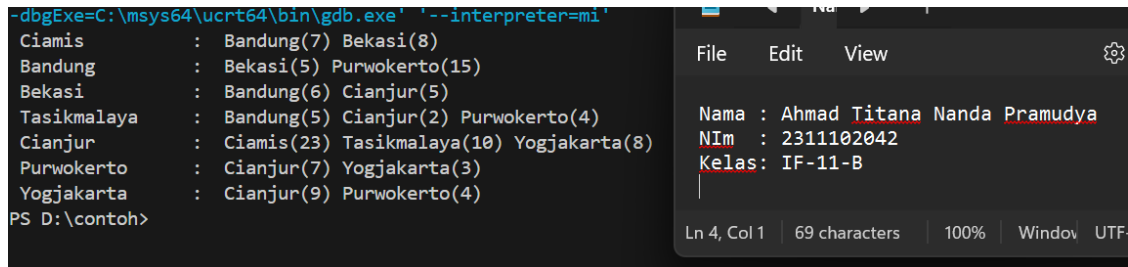
```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                  "Bandung",
                  "Bekasi",
                  "Tasikmalaya",
                  "Cianjur",
                  "Purwokerto",
                  "Yogjakarta"};

int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15) <<
simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}

int main()
{
    tampilGraph();
    return 0;
}
```

Screenshots Output :



The screenshot shows two windows. The left window is a debugger (gdb) displaying a list of cities and their connections in a binary tree structure. The right window is a text editor showing student information.

```
-dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
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 D:\contoh>
```

File Edit View

Nama : Ahmad Titana Nanda Pramudya  
NIm : 2311102042  
Kelas: IF-11-B

Ln 4, Col 1 | 69 characters | 100% | Window UTF

Deskripsi:

program ini menampilkan daftar simpul (kota) dan busur (jalur dengan berat tertentu) yang menghubungkan simpul-simpul tersebut, mencetak informasi ini dalam format yang mudah dibaca.

Guided 2

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

```

```

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

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

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

```

```

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

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

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

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

```



```

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

```
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

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,

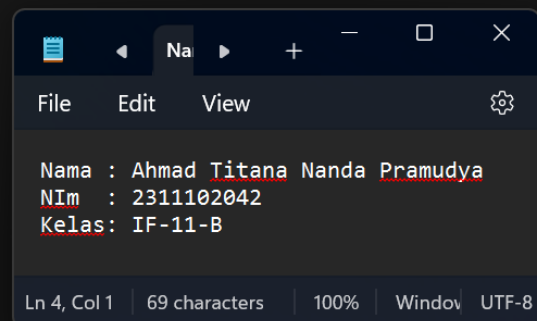
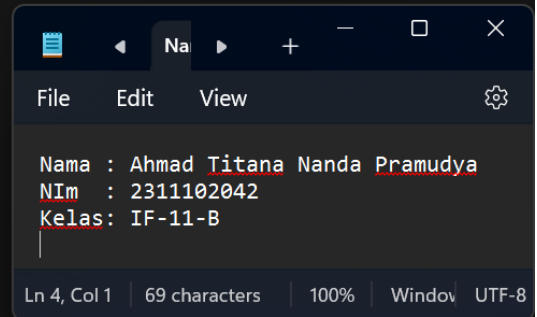
PostOrder :
D, I, J, G, H, E, B, F, C, A,

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 D:\contoh> implementasi struktur data Binary Tree (pohon biner) dalam bahasa C++. Program i
erbagai fungsi untuk membuat, memodifikasi, dan mengelola pohon biner, termasuk penelusuran da
```



Deskripsi:

implementasi struktur data Binary Tree (pohon biner) dalam bahasa C++. Program ini menyediakan berbagai fungsi untuk membuat, memodifikasi, dan mengelola pohon biner, termasuk penelusuran dan penghapusan.

### C. Unguided

Unguided 1

Sourcode :

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

int main() {
    int jumlahSimpul2311102042;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul2311102042;

    string simpul[jumlahSimpul2311102042];
    int busur[jumlahSimpul2311102042][jumlahSimpul2311102042];

    for (int i = 0; i < jumlahSimpul2311102180; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jumlahSimpul2311102042; i++) {
        for (int j = 0; j < jumlahSimpul2311102042; 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 < jumlahSimpul2311102042; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul2311102042; i++) {
        cout << setw(15) << simpul[i];
```

```

        for (int j = 0; j < jumlahSimpul2311102042; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Screenshot Output:

The screenshot shows the output of a C++ program in a terminal window. The program prompts the user to enter the number of nodes (2), the names of the nodes (Bali and Palu), and the weights of the edges between them (0, 3, 4, 0). The output then displays a graph in the form of a matrix table.

	Bali	Palu
Bali	0	3
Palu	4	0

Below the table, the terminal shows the command prompt: PS D:\contoh> ^C

Overlaid on the right is a code editor window showing a menu (File, Edit, View) and a text area with the following text:

```

Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B

```

The editor status bar at the bottom indicates: Ln 4, Col 1 | 69 characters | 100% | Window | UTF

Deskripsi:

implementasi untuk mengelola dan menampilkan representasi matriks ketetanggaan (adjacency matrix) dari sebuah graf berbobot dalam bahasa C++. Program ini meminta pengguna untuk memasukkan jumlah simpul (nodes), nama-nama simpul, dan bobot dari busur (edges) yang menghubungkan simpul-simpul tersebut. Kemudian, program menampilkan graf yang dihasilkan dalam bentuk tabel matriks.

Unguded 2

Sourcode :

```

#include <iostream>
using namespace std;

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

// Pointer global

```

```

Pohon *root2311102042;

// Inisialisasi
void init() {
    root2311102042 = NULL;
}

bool isEmpty() {
    return root2311102042 == 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()) {
        root2311102042 = 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 : " << root2311102042->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 != root2311102042) {
                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 == root2311102042) {
                delete root2311102042;
                root2311102042 = 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(root2311102042);
        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(root2311102042);
    int h = height(root2311102042);
    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', root2311102042);  
nodeC = insertRight('C', root2311102042);  
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(root2311102042);  
cout << "\n" << endl;  
cout << "InOrder :" << endl;  
inOrder(root2311102042);  
cout << "\n" << endl;  
cout << "PostOrder :" << endl;  
postOrder(root2311102042);  
cout << "\n" << endl;  
characteristic();  
displayChild(nodeE);  
displayDescendant(nodeB);  
deleteSub(nodeE);  
cout << "\nPreOrder :" << endl;  
preOrder(root2311102042);  
cout << "\n" << endl;  
characteristic();
```

```
}
```

## Screenshot Output :

```
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
```

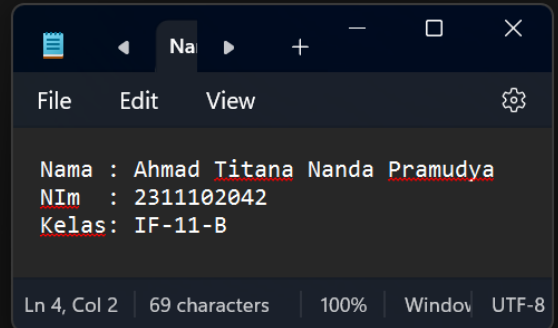
```
Descendant dari node G adalah: I
```

```
Descendant dari node I adalah:
J
```

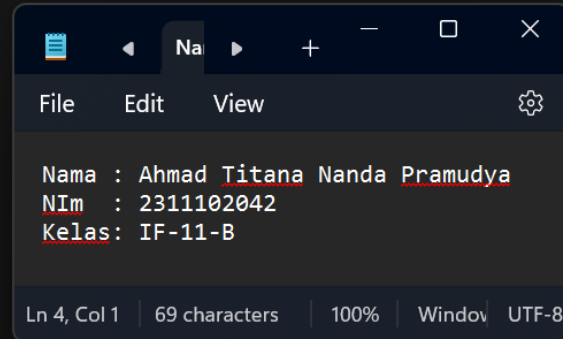
```
Descendant dari node J adalah:
```

```
H
```

```
Descendant dari node H adalah:
```



```
Node 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
```

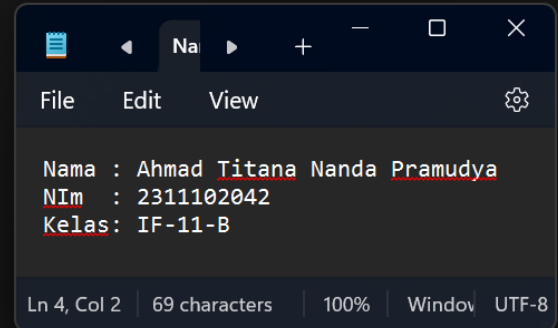
```
Descendant dari node G adalah: I
```

```
Descendant dari node I adalah:
J
```

```
Descendant dari node J adalah:
```

```
H
```

```
Descendant dari node H adalah:
```

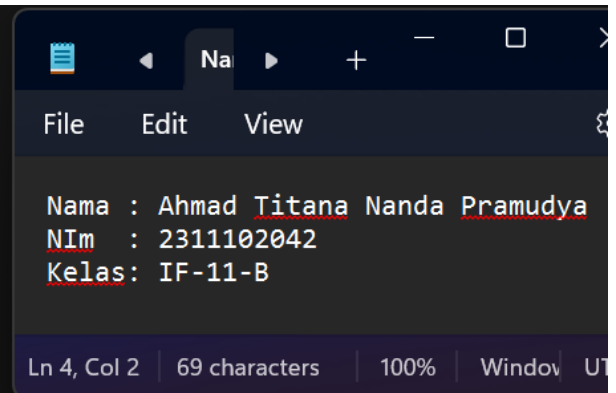


```
H
Descendant dari node H adalah:
```

```
Node subtree E berhasil dihapus.
```

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

```
Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
```



Deskripsi :

implementasi pohon biner dalam bahasa C++. Pohon biner adalah struktur data hierarkis di mana setiap node memiliki paling banyak dua anak, yaitu anak kiri dan anak kanan. Program ini mencakup berbagai fungsi untuk membuat dan mengelola pohon biner.



#### **D. Kesimpulan**

Graf dan pohon adalah dua struktur data fundamental dalam pemrograman C++. Graf merepresentasikan hubungan umum antar objek, sedangkan pohon merepresentasikan hubungan hierarki (tingkatan atau lapisan). Implementasi dan manipulasi graf dan pohon dalam C++ sangat penting untuk berbagai aplikasi dalam komputasi, dari algoritma graf hingga manajemen data yang efisien.

#### **E. Referensi**

[1] Academia.edu

[https://www.academia.edu/43671744/Graph\\_C](https://www.academia.edu/43671744/Graph_C)

[2] Trivusi

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