

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

MODUL IX

GRAPH



Disusun Oleh :

Muhammad Arsyad Zaidan (2311102058)

Dosen

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

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

B. Dasar Teori

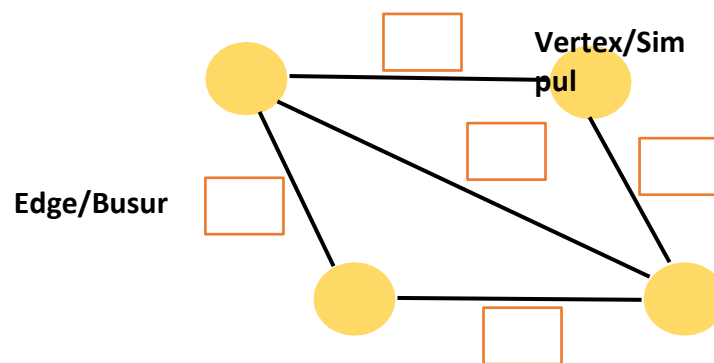
1. Graph

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.

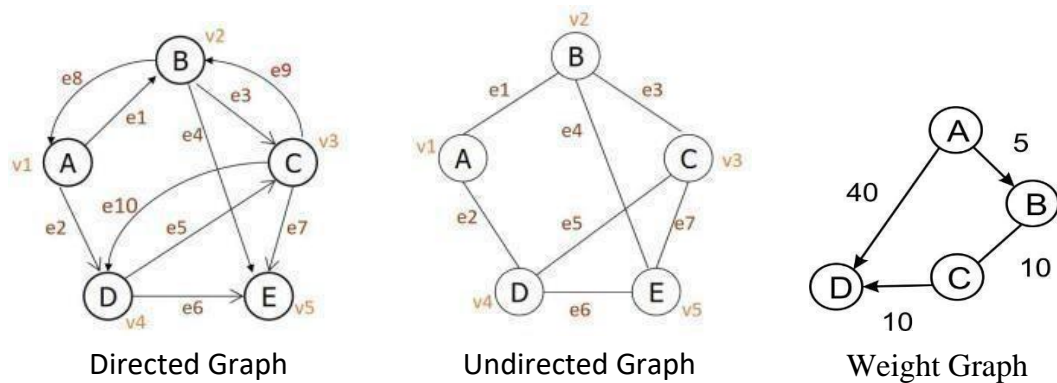
Dapat digambarkan:



Gambar 1 Contoh Graph

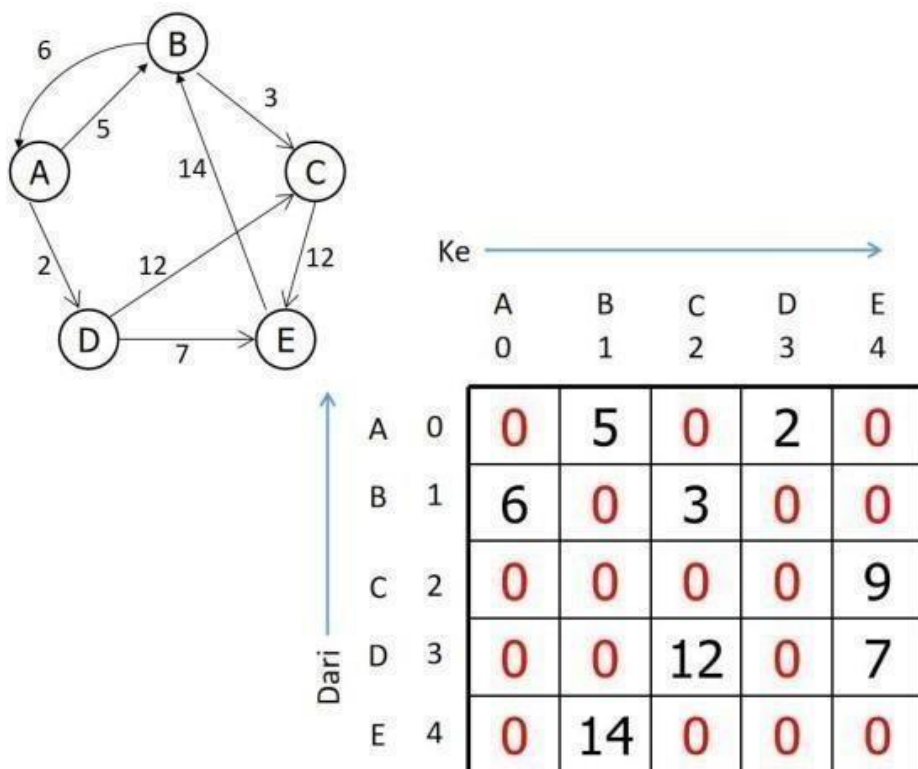
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph



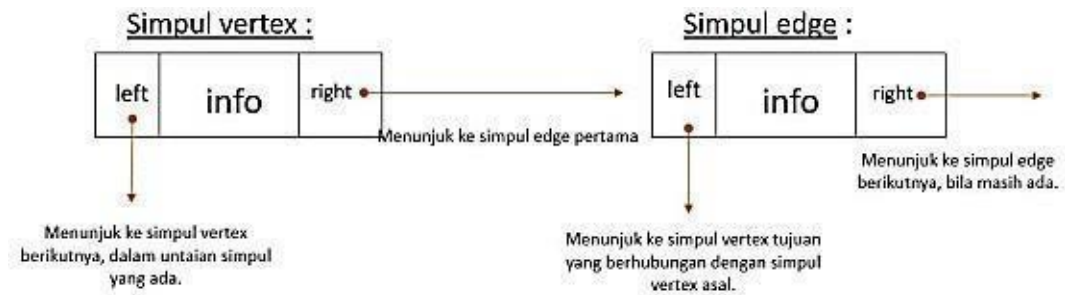
- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
- Weight Graph :** Graph yang mempunyai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

Representasi Graph dengan Matriks



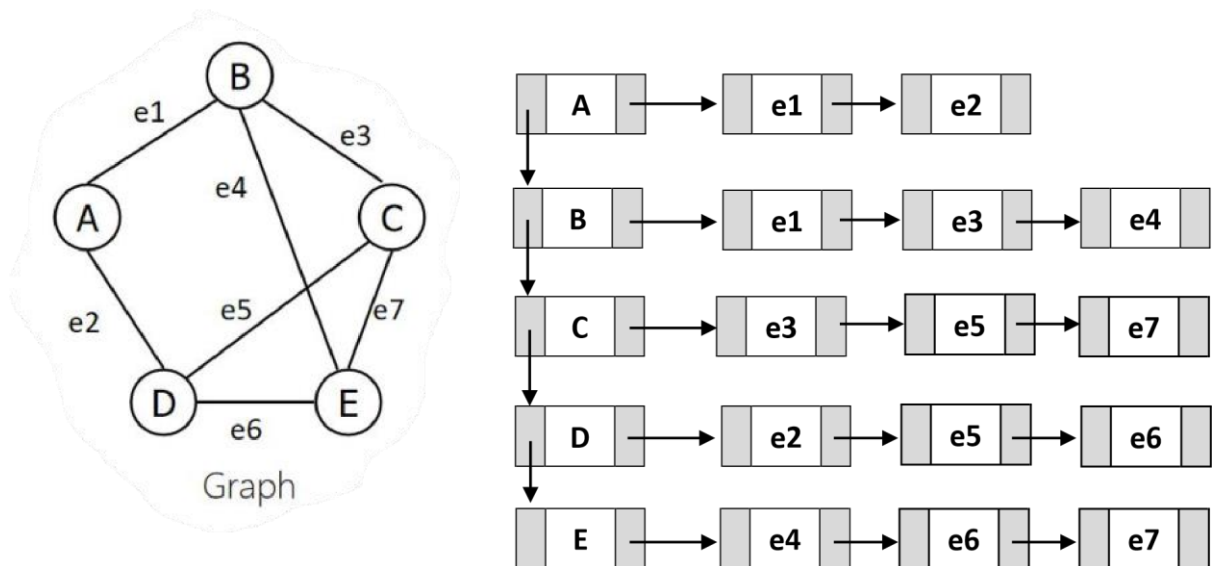
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

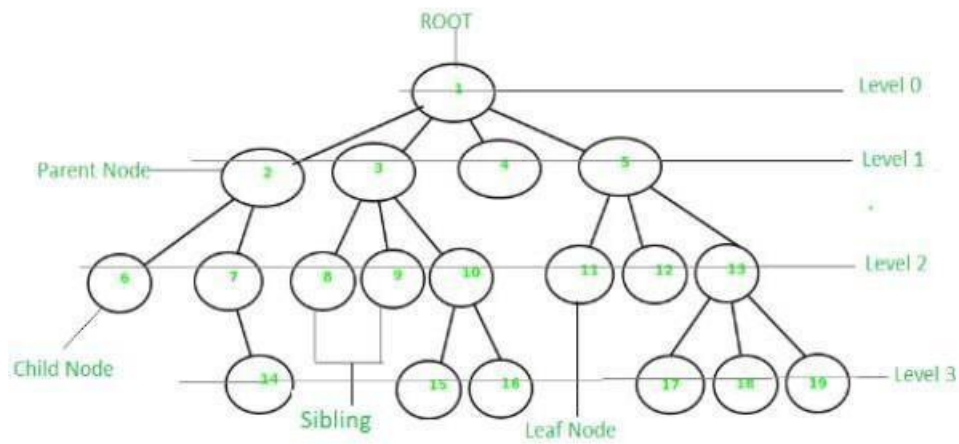
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung 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 :

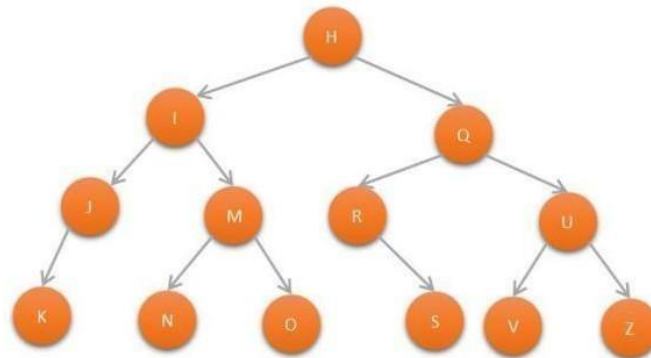


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor 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
Root	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.

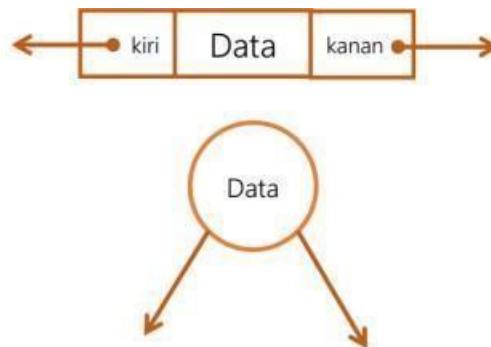


Gambar 1 Struktur Data Binary Tree

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

- a. **Create:** digunakan untuk membentuk binary tree baru yang masih kosong.
- b. **Clear:** digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- c. **isEmpty:** digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- d. **Insert:** digunakan untuk memasukkan sebuah node kedalam tree.
- e. **Find:** digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- f. **Update:** digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

- g. **Retrive:** digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- h. **Delete Sub:** digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- i. **Characteristic:** digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
- j. **Traverse:** digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

1. Pre-Order

Penelusuran secara pre-order memiliki alur:

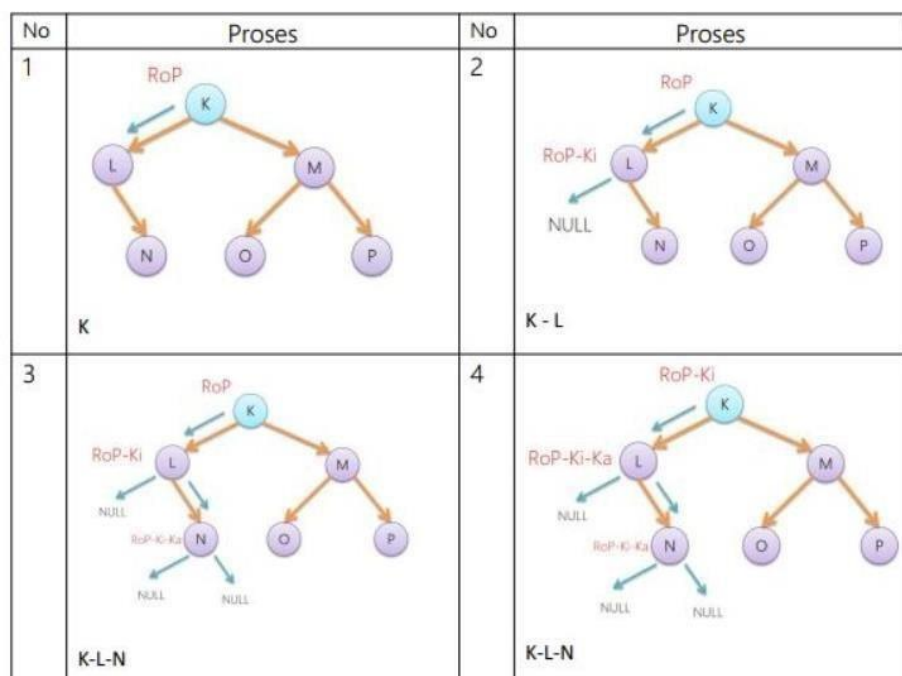
- a. Cetak data pada simpul root
- b. Secara rekursif mencetak seluruh data pada subpohon kiri
- c. Secara rekursif mencetak seluruh data pada subpohon kanan

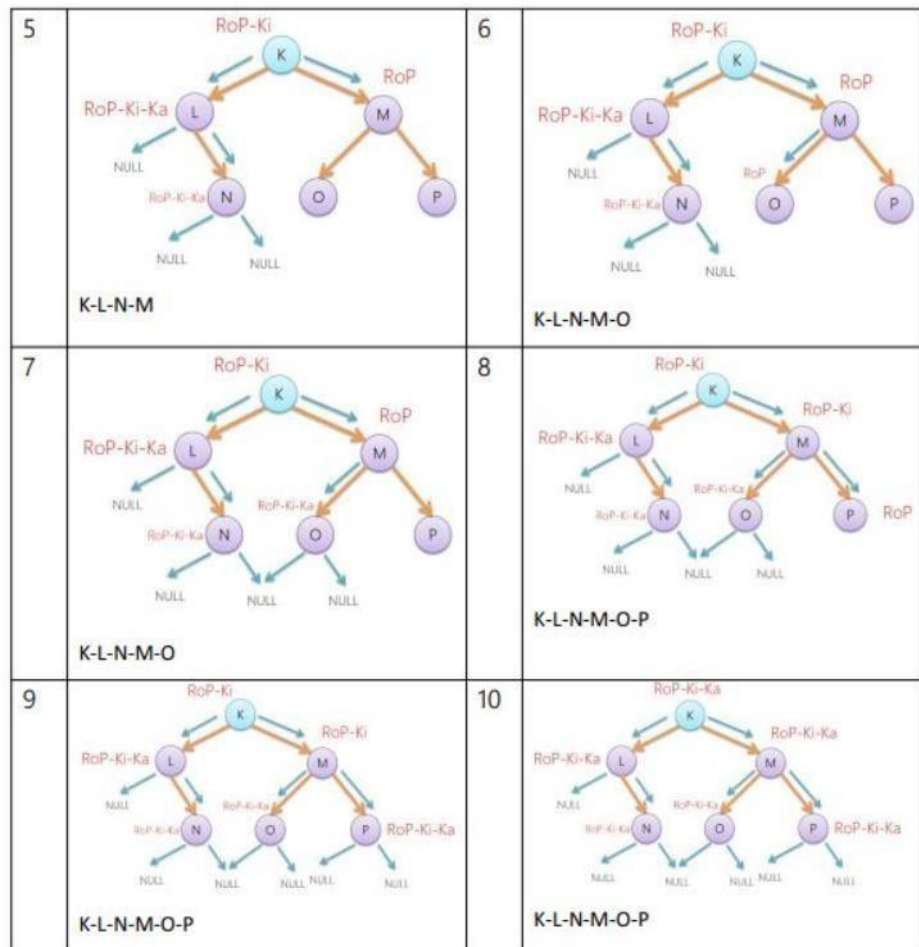
Dapat kita turunkan rumus penelusuran menjadi:

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:

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:

Kiri - Kanan - Root

Ki - Ka - Ro

Atau

Root - Kiri - Kanan(print)

Ro - Ki - KaP

1. Latihan Guided dan Unguided

1) Guided

Guided 1

Source Code

```
#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;
}

```

Output Code

```

PS C:\Learning Code\STRUKDAT (PRAKTEK)\Laprak 9_2311102058_Muhammad Arsyad Zaidan>
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 C:\Learning Code\STRUKDAT (PRAKTEK)\Laprak 9_2311102058_Muhammad Arsyad Zaidan>

```

Deskripsi Code

Representasi graf berbobot dalam program ini menampilkan kumpulan tujuh simpul dan beberapa sisi, masing-masing memiliki bobotnya sendiri. Titik-titik simpul yaitu "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogyakarta" dihubungkan oleh busur yang menandakan

jarak atau biaya.

Representasi graf ini menggunakan matriks ketetanggaan 7x7 yang menampilkan bobot yang diberikan pada hubungan antar simpul. Di dalam matriks, elemen `arc[i][j]` menampung nilai bobot busur yang menghubungkan node `i` ke node `j`. Jika tidak ada busur lurus antara dua simpul, nilai yang disimpan adalah 0. Untuk menyajikan grafik dengan jelas dan mudah dipahami, fungsi `displayGraph()` bertanggung jawab untuk menampilkan nama node, serta daftar lengkap node tetangganya dan bobot busur yang sesuai.

Guided2

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" << endl;
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);

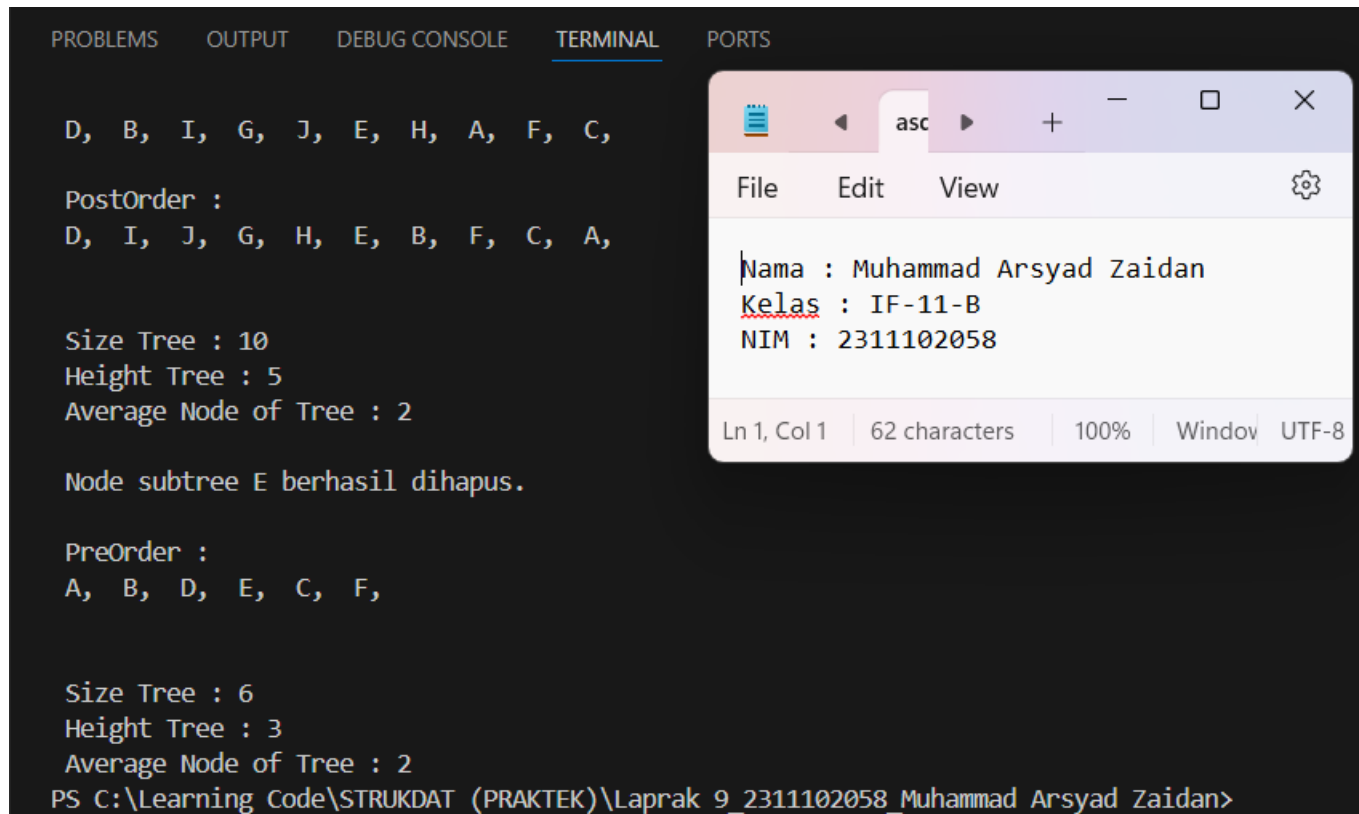
```

```

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();
}

```

Output Code



The screenshot shows a code editor with a dark theme. The 'TERMINAL' tab is active, displaying the following output:

```
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 C:\Learning Code\STRUKDAT (PRAKTEK)\Laprak 9_2311102058_Muhammad Arsyad Zaidan>
```

Overlaid on the right is a file window titled 'asc' with a menu bar (File, Edit, View) and a settings gear icon. The text content of the file is:

```
Nama : Muhammad Arsyad Zaidan  
Kelas : IF-11-B  
NIM : 2311102058
```

The status bar at the bottom of the file window shows 'Ln 1, Col 1', '62 characters', '100%', 'Window', and 'UTF-8'.

Deskripsi Code

Pengimplementasi pohon tree biner dalam program ini mencakup serangkaian operasi mendasar. Data dan pointer ke node kiri, kanan, dan induk disimpan dalam Struktur dan Struktur Pohon Inisialisasi. Selain itu, ada penunjuk akar global, yang dikenal sebagai Root, yang menunjuk ke akar pohon. Akar disetel ke NULL oleh fungsi `init()` selama inisialisasi. Untuk menentukan apakah pohon tersebut tidak memiliki elemen apa pun, fungsi `isEmpty()` dapat digunakan. Jika pohon tersebut benar-benar kosong, fungsi `createNode(char data)` akan membuat node baru dan menentukannya sebagai root. Untuk memasukkan node tambahan ke dalam struktur pohon, fungsi `insertLeft(char data, Tree *node)` dapat digunakan untuk menambahkan node baru di sisi kiri, sedangkan fungsi `insertRight(char data, Tree *node)` dapat digunakan untuk menambahkan simpul baru di sisi kanan.

2) Unguided

Unguided1

Source Code

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

int main() {
    int jmlhsmpl_2311102058;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jmlhsmpl_2311102058;

    string simpul[jmlhsmpl_2311102058];
    int busur[jmlhsmpl_2311102058][jmlhsmpl_2311102058];

    for (int i = 0; i < jmlhsmpl_2311102058; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jmlhsmpl_2311102058; i++) {
        for (int j = 0; j < jmlhsmpl_2311102058; j++) {
            cout << "Silakan masukkan bobot antara simpul " << simpul[i] << " dan " <<
simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }

    cout << "\nGraf yang dihasilkan:\n";
    cout << setw(15) << " ";
    for (int i = 0; i < jmlhsmpl_2311102058; i++) {
        cout << setw(15) << simpul[i];
    }
}
```



```

    cout << endl;

    for (int i = 0; i < jmlhsmpl_2311102058; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jmlhsmpl_2311102058; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Output Code

The screenshot shows a Windows command prompt window with the following text:

```

PS C:\Learning Code\STRUKDAT (PRAKTEK)\Laprak 9_2311102058_Muhammad Arsyad Zaidan> & 'c:\Users\ASUS\
n32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1akfwn4d.cpe' '--s
tderr=Microsoft-MIEngine-Error-gou3tesd.v4w' '--pid=Microsoft-MIEngine-Pid-1kdiyvko.xwm' '--dbgExe=C

```

Below the command prompt, the program's output is displayed:

```

Silakan masukkan jumlah simpul: 2
Simpul 1: 23
Simpul 2: 1
Silakan masukkan bobot antara simpul 23 dan 23: 21
Silakan masukkan bobot antara simpul 23 dan 1: 3
Silakan masukkan bobot antara simpul 1 dan 23: 4
Silakan masukkan bobot antara simpul 1 dan 1: 1

Graf yang dihasilkan:

```

	23	1
23	21	3
1	4	1

At the bottom of the command prompt, the following text is visible:

```

PS C:\Learning Code\STRUKDAT (PRAKTEK)\Laprak 9_2311102058_Muhammad Arsyad Zaidan> 

```

Overlaid on the right side of the command prompt is a Notepad window titled "asc" with the following text:

```

Nama : Muhammad Arsyad Zaidan
Kelas : IF-11-B
NIM : 2311102058

```

The Notepad window also shows a status bar at the bottom indicating "Ln 1, Col 30 | 62 characters | 100% | Window | UTF-8".

Deskripsi Code

Program tersebut mempresentasikan graf yang berbobot dan bermatriks ketetanggaan. Yang mana meminta para user sejumlah simpul dan nama. Bobot yang dimasukan oleh bersifat bobot busur. Agar program dapat mencetak tabel yang menampilkan bobot dari setiap busur antar simpul dan matriksnya menampilkan hubungan dan bobot antar simpul dalam bentuk tabel.

Unguided2

Source Code

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

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

// Pointer global
Pohon *root_2311102058;

// Inisialisasi
void init() {
    root_2311102058 = NULL;
}

bool isEmpty() {
    return root_2311102058 == NULL;
}

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

void buatNode(char data) {
```

```

    if (isEmpty()) {
        root_2311102058 = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

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

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

```

```

        Pohon *baru = newPohon(data);
        baru->parent = node;
        node->right = baru;
        cout << "\nNode " << data << " berhasil ditambahkan ke child kanan dari "
<< node->data << endl;
        return baru;
    }
}

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

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

```

```

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root_2311102058->data << endl;

            if (!node->parent)
                cout << "Parent : (tidak memiliki parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;

            if (node->parent != NULL && node->parent->left != node && node->parent->right == node)
                cout << "Sibling : " << node->parent->left->data << endl;
            else if (node->parent != NULL && node->parent->right != node && node->parent->left == node)
                cout << "Sibling : " << node->parent->right->data << endl;
            else
                cout << "Sibling : (tidak memiliki sibling)" << endl;

            if (!node->left)
                cout << "Child Kiri : (tidak memiliki child kiri)" << endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;

            if (!node->right)
                cout << "Child Kanan : (tidak memiliki child kanan)" << endl;
            else
                cout << "Child Kanan : " << node->right->data << endl;
        }
    }
}

```

```

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

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

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

```

```

        cout << " " << node->data << ", ";

    }

}

}

// Hapus Node Tree
void deleteTree(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            if (node != root_2311102058) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);

            if (node == root_2311102058) {
                delete root_2311102058;
                root_2311102058 = NULL;
            } else {
                delete node;
            }
        }
    }
}

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

```

```

        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil dihapus." << endl;
    }
}

// Hapus Tree
void clear() {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(root_2311102058);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

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

// Cek Height Level Tree
int height(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
    } else {
        if (!node) {

```



```

        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);

        if (heightKiri >= heightKanan) {
            return heightKiri + 1;
        } else {
            return heightKanan + 1;
        }
    }
}

// Karakteristik Tree
void characteristic() {
    int s = size(root_2311102058);
    int h = height(root_2311102058);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

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

```

```

        cout << " " << node->left->data;
    }
    if (node->right) {
        cout << " " << node->right->data;
    }
    cout << endl;
}
}

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

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

```

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

```
nodeB = insertLeft('B', root_2311102058);  
nodeC = insertRight('C', root_2311102058);  
nodeD = insertLeft('D', nodeB);  
nodeE = insertRight('E', nodeB);  
nodeF = insertLeft('F', nodeC);  
nodeG = insertLeft('G', nodeE);  
nodeH = insertRight('H', nodeE);  
nodeI = insertLeft('I', nodeG);  
nodeJ = insertRight('J', nodeG);
```

```
update('Z', nodeC);  
update('C', nodeC);  
retrieve(nodeC);  
find(nodeC);  
cout << "\nPreOrder :" << endl;  
preOrder(root_2311102058);  
cout << "\n" << endl;  
cout << "InOrder :" << endl;  
inOrder(root_2311102058);  
cout << "\n" << endl;  
cout << "PostOrder :" << endl;  
postOrder(root_2311102058);  
cout << "\n" << endl;  
characteristic();  
displayChild(nodeE);  
displayDescendant(nodeB);  
deleteSub(nodeE);  
cout << "\nPreOrder :" << endl;  
preOrder(root_2311102058);  
cout << "\n" << endl;  
characteristic();
```

```
}
```

Output Code

```
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

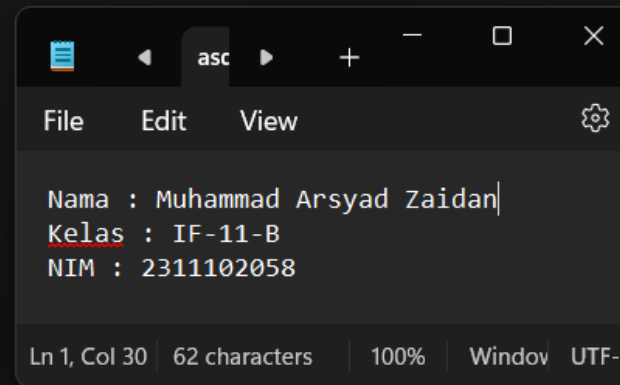
Data node : C

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,
```



Deskripsi Code

Kode ini mengeksekusi operasi-operasi pada tree biner, yang bertujuan untuk membuat node baru dan mengatur node root. Kemudian, dalam kode tersebut akan dilakukan proses insert sebagai child kiri atau

child kanan dari node yang telah diberikan asalkan belum ada child pada posisi tersebut. Selain fungsi update, retrieve, dan find, kode ini juga melibatkan proses penjelajahan dengan tiga jenis yaitu preorder, inorder, dan postorder.

2. Kesimpulan

Dari pratikum ini, mahasiswa dapat mempelajari mengimplementasikan 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. Jenis-jenis graph terdapat graph berarah (directed graph), graph tak berarah (undirected graph), dan weight graph. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.

3. Referensi

- [1] Tim Asisten Asprak, "Graph", Learning Management System, 2024.
- [2] GeeksforGeeks. (2024a, March 20). Difference Between Graph and Tree. GeeksforGeeks. [https://www-geeksforgeeks-org.translate.goog/difference-between-graph-and-tree/? x tr sl=en& x tr tl=id& x tr hl=id& x tr pto=tc](https://www.geeksforgeeks-org.translate.goog/difference-between-graph-and-tree/? x tr sl=en& x tr tl=id& x tr hl=id& x tr pto=tc). Terakhir kali diakses 13 Juni 2024.