

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



**Disusun oleh:
Alfin Ilham Berlianto
2311102047**

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

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

MODUL 9

GRAPH DAN TREE

A. Tujuan

1. Mampu memahami konsep graph dan tree pada struktur data dan algoritma.
2. Mampu mengimplementasikan graph dan tree pada pemrograman.

B. Dasar Teori

Graph adalah kumpulan node (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi).

istilah yang berkaitan dengan graph yaitu:

a. Vertex

Adalah himpunan node / titik pada sebuah graph.

b. Edge

Adalah himpunan garis yang menghubungkan tiap node / vertex.

c. Adjacent

Adalah dua buah titik dikatakan berdekatan (adjacent) jika dua buah titik tersebut terhubung dengan sebuah sisi.

d. Weight

Adalah Sebuah graph $G = (V, E)$ disebut sebuah graph berbobot (weight graph).

e. Path

Adalah jalur dengan setiap vertex berbeda

f. Cycle

Adalah Siklus (Cycle) atau Sirkuit (Circuit) yaitu lintasan yang berawal dan berakhir pada simpul yang sama .

Graph terdapat dua jenis traversal, yaitu:

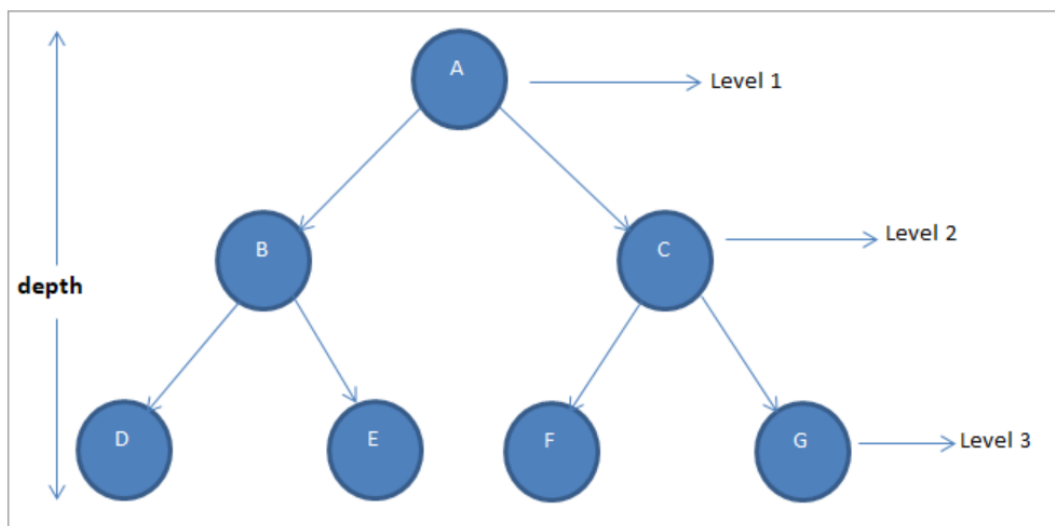
- DFS (Depth First Search), melakukan kunjungan ke node-node dengan cara mengunjungi node terdalam/kebawah ,setelah itu mencari tempat yang lainnya , dan sistemnya menggunakan stack
- BFS (Breadth First Search), melakukan visit ke node- node dengan cara melebar kesamping, dan sistemnya menggunakan queue

Tree

Tree adalah struktur data pohon yang banyak digunakan. Jika setiap node dari sebuah pohon mempunyai paling banyak dua node anak, maka pohon tersebut disebut Tree.

Jadi pohon biner tipikal akan memiliki komponen-komponen berikut:

- Subpohon kiri
- Sebuah simpul akar
- Subpohon kanan



Dalam pohon biner tertentu, jumlah maksimum node pada level mana pun adalah 2^{l-1} dengan 'l' adalah nomor level.

Jadi jika node akar berada pada level 1, jumlah node maksimal $= 2^{1-1} = 2^0 = 1$

Karena setiap node dalam pohon biner memiliki paling banyak dua node, node maksimum pada level berikutnya adalah, $2 * 2^{l-1}$.

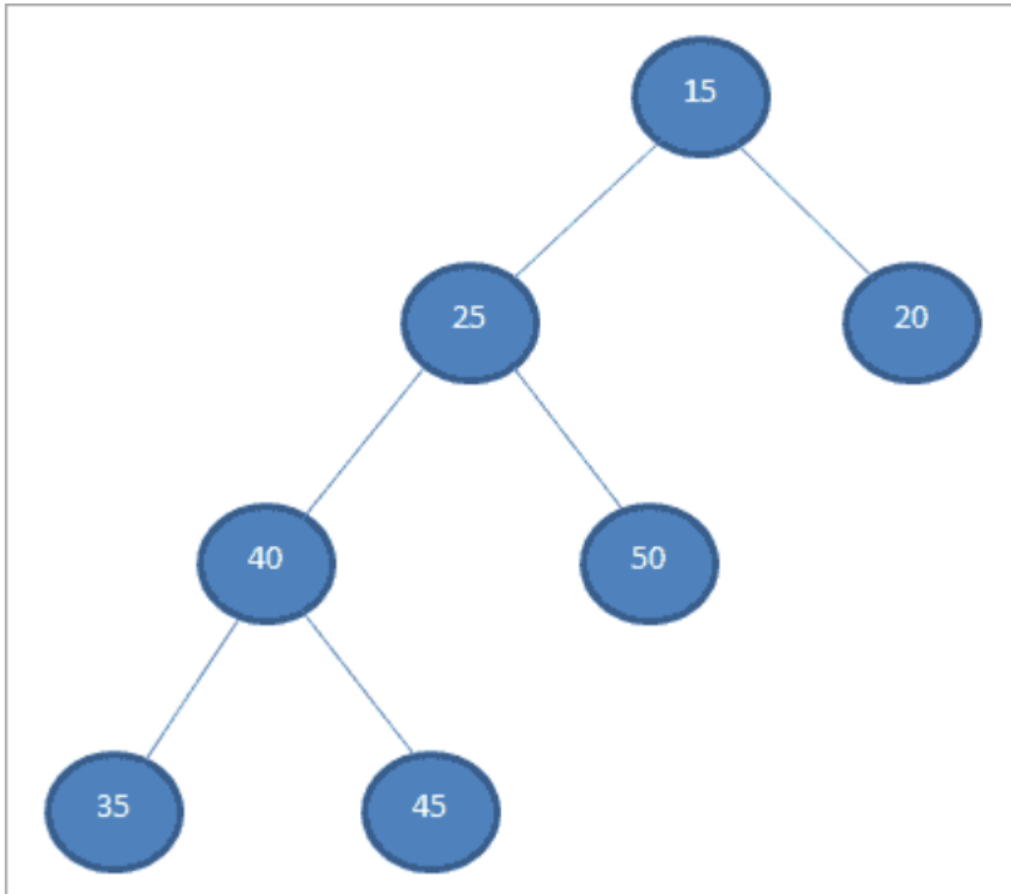
Diketahui pohon biner dengan kedalaman atau tinggi h, jumlah maksimum node dalam pohon biner dengan tinggi $h = 2^h - 1$.

Oleh karena itu, dalam pohon biner dengan tinggi 3 (ditunjukkan di atas), jumlah maksimum node $= 2^3 - 1 = 7$.

Berikut ini adalah jenis pohon biner yang paling umum.

#1) Pohon Biner Penuh

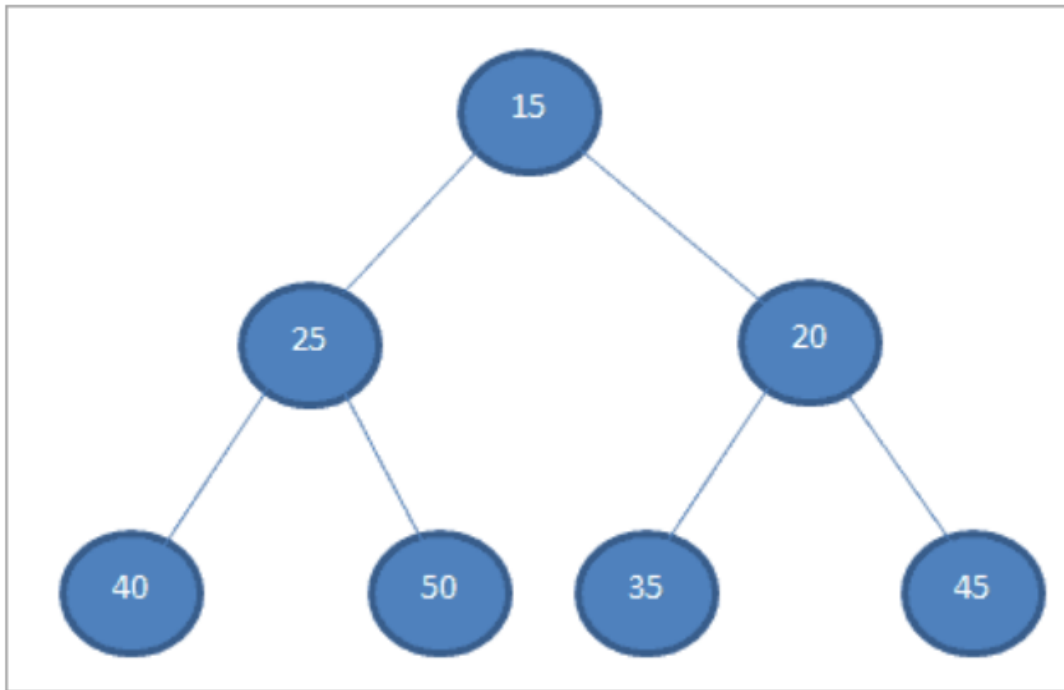
Pohon biner yang setiap simpulnya mempunyai 0 atau 2 anak disebut pohon biner penuh.



Gambar di atas adalah pohon biner penuh dimana kita dapat melihat bahwa semua nodenya kecuali node daun memiliki dua anak. Jika L adalah jumlah simpul daun dan I adalah jumlah simpul internal atau non-daun, maka untuk pohon biner penuh, $L = I + 1$.

#2) Pohon Biner Lengkap

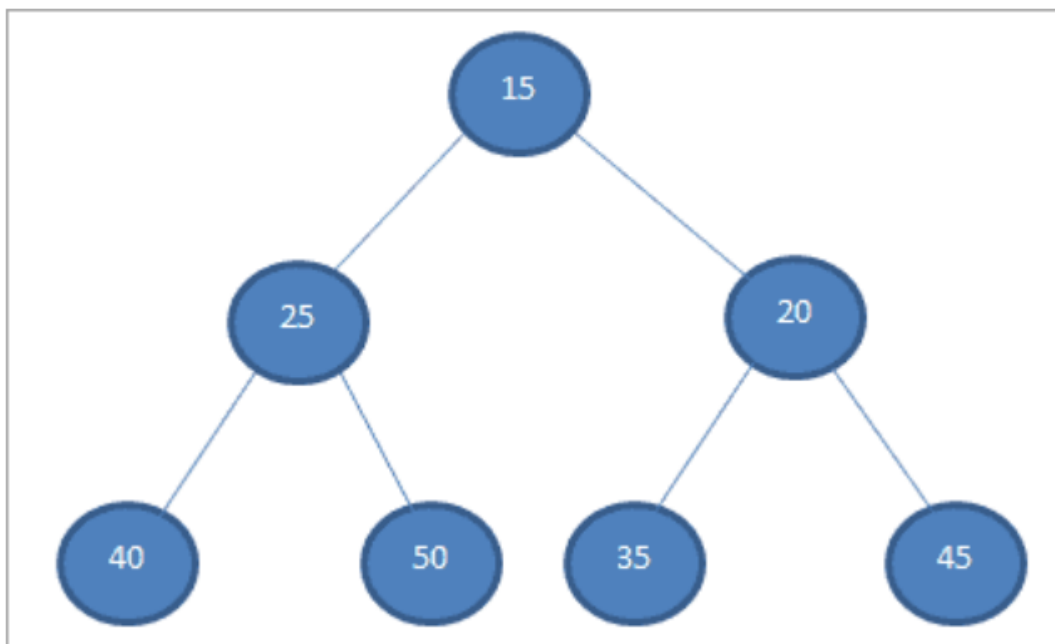
Pohon biner lengkap memiliki semua level yang terisi kecuali level terakhir dan level terakhir memiliki semua node di sebelah kirinya.



Pohon yang ditunjukkan di atas adalah pohon biner lengkap.

#3) Pohon Biner Sempurna

Pohon biner disebut sempurna jika semua simpul internalnya mempunyai dua anak dan semua simpul daun berada pada level yang sama.



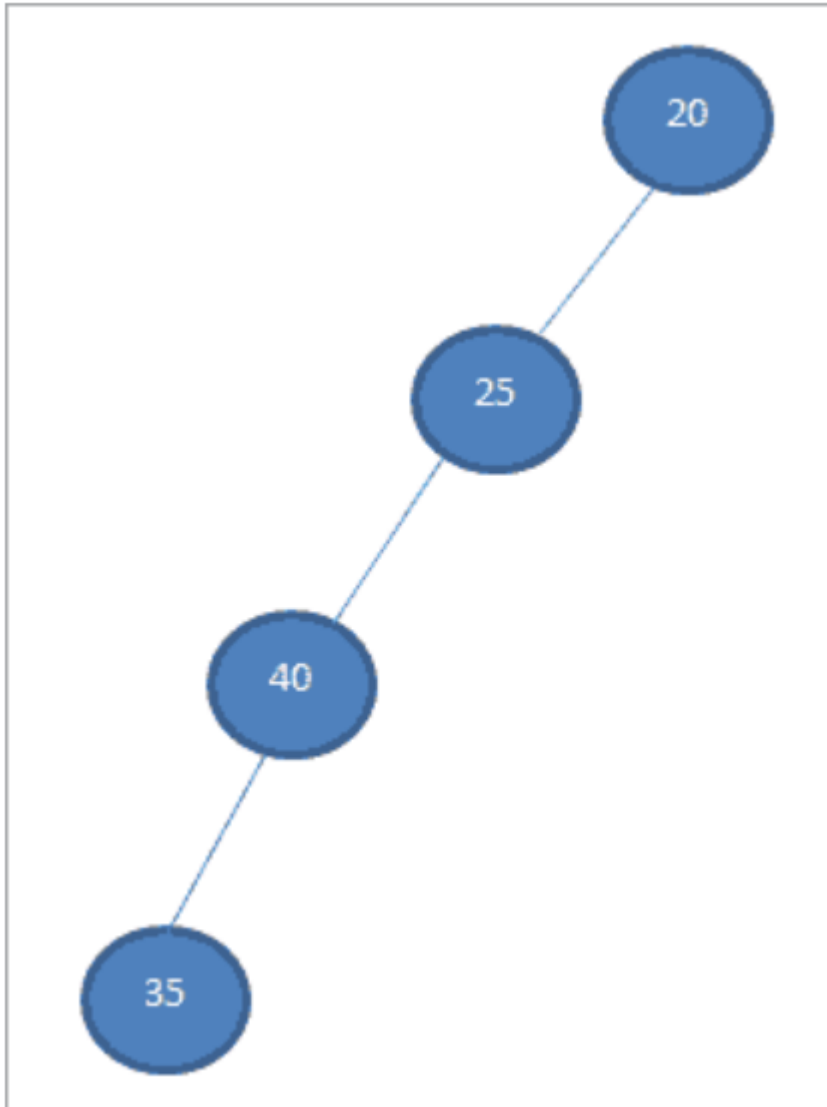
Contoh pohon biner yang ditunjukkan di atas adalah pohon biner sempurna karena setiap

simpulnya memiliki dua anak dan semua simpul daun berada pada level yang sama.

Pohon biner sempurna dengan tinggi h memiliki $2^h - 1$ jumlah node.

#4) Pohon yang Merosot

Pohon biner yang setiap simpul internalnya hanya mempunyai satu anak disebut pohon degenerasi.



Pohon yang ditunjukkan di atas adalah pohon yang merosot. Se jauh menyangkut kinerja pohon ini, pohon yang mengalami degenerasi sama dengan daftar tertaut.

C. Guided

Guided 1

SOURCE CODE

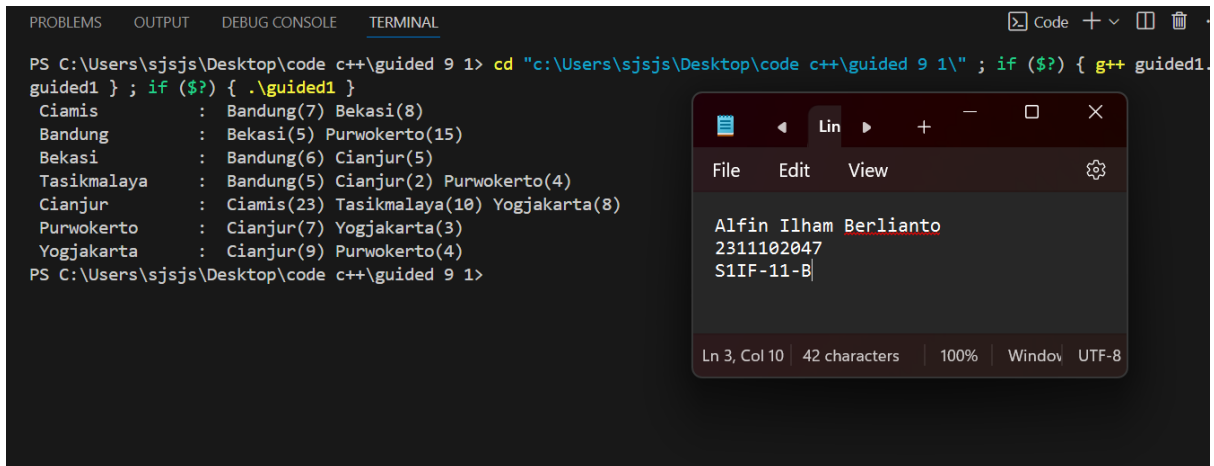
```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};

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;
}
```

SCREENSHOT OUTPUT



The screenshot shows a Windows terminal window with a dark background. The terminal output displays a C++ program's execution, showing a graph structure with nodes and edges. The nodes are listed with their connections and weights: Cianjis (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)), and Yogyakarta (Cianjur(9), Purwokerto(4)).

```
PS C:\Users\sjsjs\Desktop\code c++\guided 9 1> cd "c:\Users\sjsjs\Desktop\code c++\guided 9 1\" ; if ($?) { g++ guided1.  
guided1 } ; if ($?) { .\guided1 }  
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:\Users\sjsjs\Desktop\code c++\guided 9 1>
```

Overlaid on the terminal is a code editor window titled 'Lin'. It contains the following text:

```
Alfin Ilham Berlianto  
2311102047  
S1IF-11-B
```

The code editor's status bar at the bottom indicates 'Ln 3, Col 10', '42 characters', '100%', 'Window', and 'UTF-8'.

DESKRIPSI PROGRAM

Program di atas adalah program yang mendemonstrasikan penggunaan graf. Dalam program ini, keterhubungan antar node (verteks) dalam graf digambarkan melalui matriks (array 2 dimensi). Setiap baris di dalam array memberikan informasi terkait keterhubungan node tersebut dengan node lain. Ketika di eksekusi, program akan looping ke setiap elemen dalam matriks. Untuk setiap iterasi baris matriks akan diawali oleh nama kota. Untuk iterasi kolom, jika nilai elemen tidak sama dengan 0, maka program akan menampilkan simpul yang terhubung dan bobot dari edge. Looping akan dijalankan sampai elemen terakhir dalam matriks.

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;
        }
    }
}
// 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


```

PS C:\Users\sjsjs\Desktop\code c++\guided 9 2> cd "c:\Users\sjsjs\Desktop\code c++\guided 9 2\" ; if ($?) { g++ guided2.cpp
guided2 } ; if ($?) { .\guided2 }

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

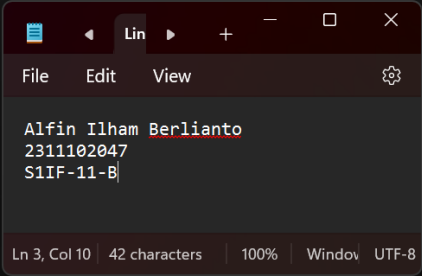
Node C berhasil diubah menjadi Z

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,

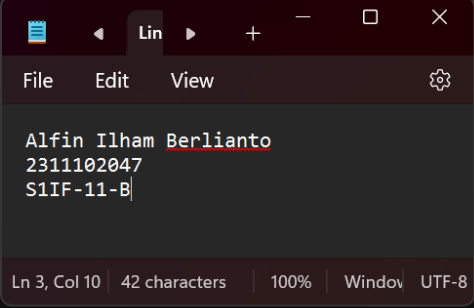
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\Users\sjsjs\Desktop\code c++\guided 9 2>

```



DESKRIPSI PROGRAM

Program di atas adalah program yang mendemonstrasikan penggunaan tree. Dalam program ini, data tree disimpan dalam bentuk node yang menyimpan nilai (value) dan pointer yang mengarah ke parent serta child (kanan & kiri) dari node tersebut. Ketika running, program akan menginisialisasi tree dengan membuat root dari tree terlebih dahulu dengan fungsi buatNode(). Setelah itu, program akan membuat beberapa child

(kanan & kiri) dengan menggunakan insertLeft() untuk child kiri dan insertRight() untuk child kanan. Data node dalam program dapat diupdate melalui fungsi update(). Adapun fungsi retrieve untuk menampilkan nilai dari node dan find() untuk menampilkan data dari node dengan lebih rinci (informasi tentang parent, child,

sibling). Dalam menampilkan semua node dalam tree dapat dilakukan dengan menggunakan fungsi preOrder(), inOrder(), dan postOrder(). Perbedaan dari ketiga fungsi tersebut adalah urutan data yang ditampilkan. Adapula fungsi characteristic() yang digunakan untuk memberikan status terkait ukuran, tinggi, dan rata - rata node dalam tree. Fungsi tersebut memanggil 2 fungsi lain yaitu size() untuk mendapatkan ukuran tree dan height() untuk mendapatkan tinggi tree. Lalu untuk fungsi penghapusan dalam program ini terdiri dari 3, yaitu deleteTree() untuk menghapus node, deleteSub() untuk menghapus upapohon (descendant dari node), dan clear() untuk menghapus tree secara keseluruhan.

E. Unguided

Unguided 1

SOURCE CODE

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

using namespace std;

int main()
{
    int countVertex_2311102047;

    cout << "Masukkan jumlah simpul : ";
    cin >> countVertex_2311102047;

    string vertecies[countVertex_2311102047];
    int edgeValues[countVertex_2311102047][countVertex_2311102047];
    cout << "Masukkan nama simpul,\n";
    for (int i = 0; i < countVertex_2311102047; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> vertecies[i];
    }

    cout << "Masukkan bobot antar simpul,\n";
    for (int i = 0; i < countVertex_2311102047; i++) {
        for (int j = 0; j < countVertex_2311102047; j++) {
```

```

        cout << vertecies[i] << "->" << vertecies[j] << " : ";
        cin >> edgeValues[i][j];
    }
}

cout << endl << setw(10) << " ";
for (int i = 0; i < countVertex_2311102047; i++) {
    cout << setw(10) << vertecies[i];
}
cout << endl;

for (int i = 0; i < countVertex_2311102047; i++) {
    cout << setw(10) << vertecies[i];
    for (int j = 0; j < countVertex_2311102047; j++) {
        cout << setw(10) << edgeValues[i][j];
    }
    cout << endl;
}

return 0;
}

```

SCREENSHOT OUTPUT

```

PS C:\Users\sjsjs\Desktop\code c++\guided 9 2> cd "c:\Users\sjsjs\Desktop\code c++\guided 9 2\" ; if ($?) { g++ guide
guided2 } ; if ($?) { .\guided2 }
Masukkan jumlah simpul : 2
Masukkan nama simpul,
Simpul 1 : BALI
Simpul 2 : PALU
Masukkan bobot antar simpul,
BALI->BALI : 0
BALI->PALU : 3
PALU->BALI : 4
PALU->PALU : 0

      BALI      PALU
BALI      0      3
PALU      4      0
PS C:\Users\sjsjs\Desktop\code c++\guided 9 2>

```

Alfin Ilham Berlianto
2311102047
S1IF-11-B

Ln 3, Col 10 | 42 characters | 100% | Window | UTF-8

DESKRIPSI PROGRAM

Program di atas adalah sebuah program graf yang dapat menerima input dari pengguna. Pertama - tama program akan meminta input berupa jumlah simpul dari graf. Setelah itu, program melakukan looping untuk mendapatkan nama dari simpul. Banyaknya iterasi looping tadi akan menyesuaikan dengan jumlah simpul yang diinputkan. Setelah itu, program akan melakukan nested looping untuk memberikan nilai ke setiap edge dari graf. Banyaknya iterasi looping dapat dirumuskan sebagai $n \times n$ atau n^2 . Setelah semua edge diberikan nilai, maka program akan menampilkan hasil inputan tadi. Setelah itu, program selesai.

Unguided 2

SOURCE CODE

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru, *current;
// 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;
        current = root;
    }
    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;
}

void showChild(Pohon *node) {
    if (!root) {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else {
        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;
    }
}

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

```

```

char inData() {
    char t;
    cout << "Masukkan data : ";
    cin >> t;
    return t;
}

void changeCurrent(char dest, Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else if (current->data == dest) return;
    else
    {
        if (node->data == dest) {
            Pohon *temp = current;
            current = node;
            cout << "\n Current node : " << temp->data << " -> " << current-
>data << endl;
        }
        else
        {
            if (node->left != nullptr) changeCurrent(dest, node->left);
            if (node->right != nullptr) changeCurrent(dest, node->right);
        }
    }
}

int main()
{
    string line(15, '=');
    int choice;

    while(true) {
        string options[] = {
            "Tampilkan node", "Tampilkan node (detail)", "Tampilkan child",
            "Tampilkan descendant", "Ukuran tree", "Tinggi tree", "Karakteristik tree",
            "Buat root", "Tambah child (kiri)", "Tambah child (kanan)",
            "Ganti `current node`", "Update data node",
            "Urutkan node (Pre-Order)", "Urutkan node (In-Order)", "Urutkan node
(Post-Order)",
            "Hapus tree", "Hapus sub-tree", "Hapus node",
        };
        int optSize = sizeof(options)/sizeof(options[0]);

        cout << line << " Program Tree " << line << endl;
        for (int i = 0; i < optSize; i++) {
            cout << i + 1 << ". " << options[i] << endl;
        }
        cout << "0. Keluar\n";
    }
}

```

```

        cout << "\nNode saat ini : ";
        if (isEmpty()) {
            cout << "-\n";
        }
        else {
            cout << current->data << endl;
        }
        cout << "Pilih menu [0 - " << optSize <<"] : ";
        cin >> choice;

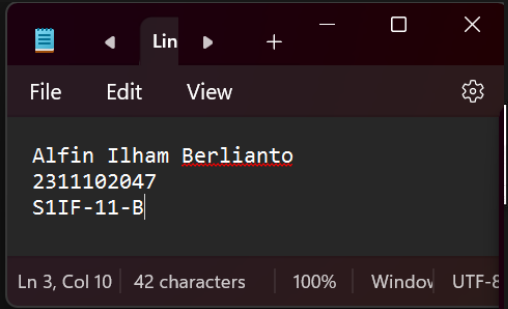
        switch(choice) {
            case 0: {
                cout << "\n\nKeluar dari aplikasi.\n";
                return 0;
                break;
            }
            case 1: retrieve(current); break;
            case 2: find(current); break;
            case 3: showChild(current); break;
            case 4: showDescendant(); cout << endl; break;
            case 5: cout << "Ukuran tree : " << size() << endl; break;
            case 6: cout << "Tinggi tree : " << height() << endl; break;
            case 7: charateristic(); break;
            case 8: buatNode(inData()); break;
            case 9: insertLeft(inData(), current); break;
            case 10: insertRight(inData(), current); break;
            case 11: changeCurrent(inData()); break;
            case 12: update(inData(), current); break;
            case 13: preOrder(); break;
            case 14: inOrder(); break;
            case 15: postOrder(); break;
            case 16: clear(); break;
            case 17: deleteSub(current); break;
            case 18: deleteTree(current); break;
            default: cout << "Tolong masukkan input yang sesuai!\n";
        }
    }
}

```

SCREENSHOT OUTPUT

```
===== Program Tree =====
1. Tampilkan node
2. Tampilkan node (detail)
3. Tampilkan child
4. Tampilkan descendant
5. Ukuran tree
6. Tinggi tree
7. Karakteristik tree
8. Buat root
9. Tambah child (kiri)
10. Tambah child (kanan)
11. Ganti `current node`
12. Update data node
13. Urutkan node (Pre-Order)
14. Urutkan node (In-Order)
15. Urutkan node (Post-Order)
16. Hapus tree
17. Hapus sub-tree
18. Hapus node
0. Keluar

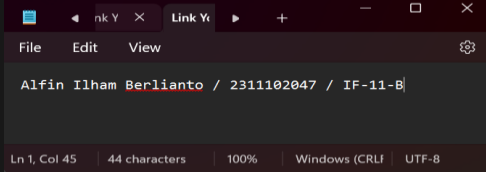
Node saat ini : -
Pilih menu [0 - 18] : 
```



```
0. Keluar

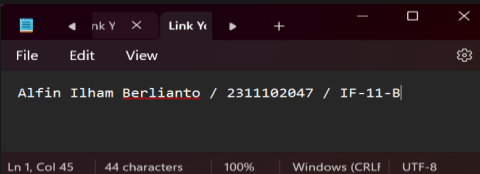
Node saat ini : b
Pilih menu [0 - 18] : 1

Data node : b
```

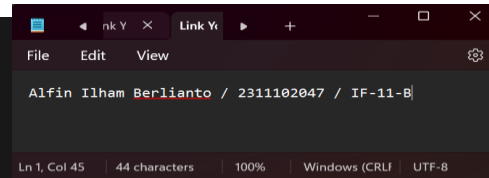


```
Node saat ini : b
Pilih menu [0 - 18] : 2

Data Node : b
Root : a
Parent : a
Sibling : c
Child Kiri : d
Child Kanan : e
```



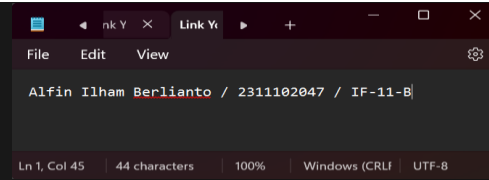
```
Node saat ini : b
Pilih menu [0 - 18] : 3
Child Kiri : d
Child Kanan : e
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
18. Hapus node
0. Keluar

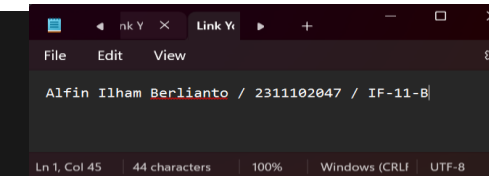
Node saat ini : a
Pilih menu [0 - 18] : 4
b, d, e, c,
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
0. Keluar

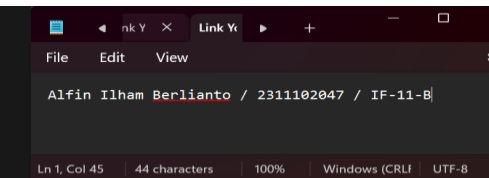
Node saat ini : b
Pilih menu [0 - 18] : 5
Ukuran tree : 5
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
0. Keluar

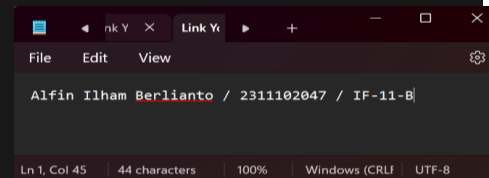
Node saat ini : b
Pilih menu [0 - 18] : 6
Tinggi tree : 3
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
Node saat ini : b
Pilih menu [0 - 18] : 7

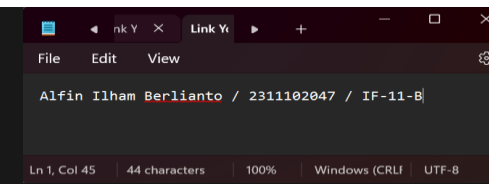
Size Tree : 5
Height Tree : 3
Average Node of Tree : 1
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
Node saat ini : -
Pilih menu [0 - 18] : 8
Masukkan data : a

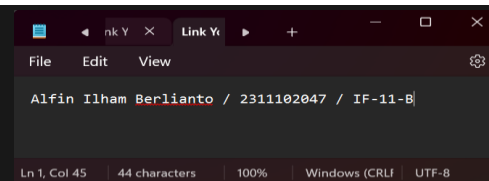
Node a berhasil dibuat menjadi root.
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
Node saat ini : a
Pilih menu [0 - 18] : 9
Masukkan data : b

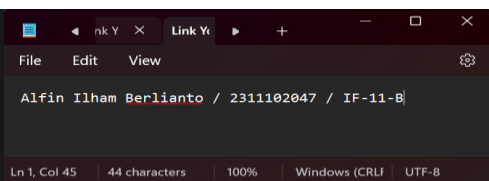
Node b berhasil ditambahkan ke child kiri a
```



```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```

```
Node saat ini : a
Pilih menu [0 - 18] : 10
Masukkan data : c

Node c berhasil ditambahkan ke child kanana
```

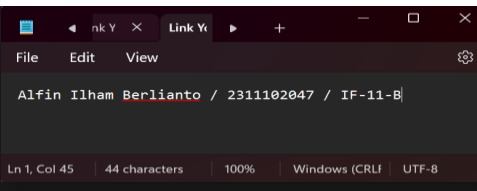


```
Alfin Ilham Berlianto / 2311102047 / IF-11-B
```



```
Node saat ini : a
Pilih menu [0 - 18] : 11
Masukkan data : b

Current node : a -> b
```



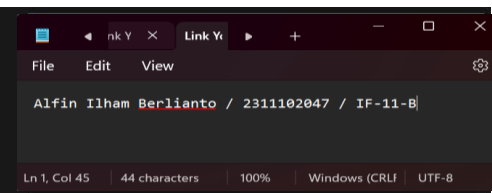
```
Node saat ini : b
Pilih menu [0 - 18] : 12
Masukkan data : L

Node b berhasil diubah menjadi L
```



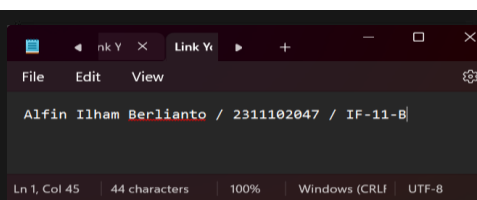
```
0. Keluar

Node saat ini : L
Pilih menu [0 - 18] : 13
a, L, d, e, c, ===== Program Tree =====
```

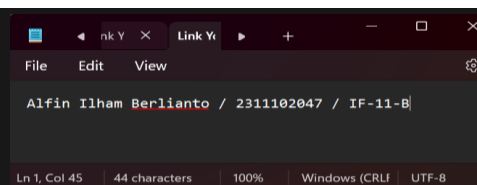


```
0. Keluar

Node saat ini : L
Pilih menu [0 - 18] : 14
d, L, e, a, c, ===== Program Tree =====
```

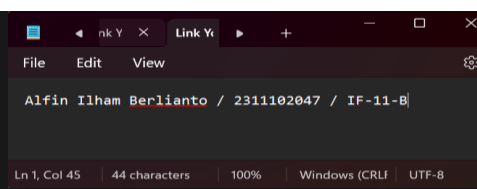


```
Node saat ini : L
Pilih menu [0 - 18] : 15
d, e, L, c, a, ===== Program Tree =====
```



```
Node saat ini : L
Pilih menu [0 - 18] : 16

Pohon berhasil dihapus.
```



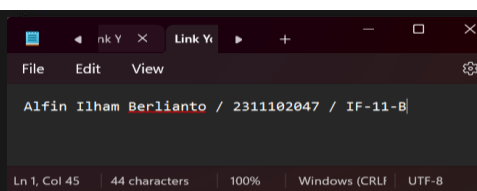
```
Node saat ini : L
Pilih menu [0 - 18] : 17

Node subtree L berhasil dihapus.
===== Program Tree =====
```



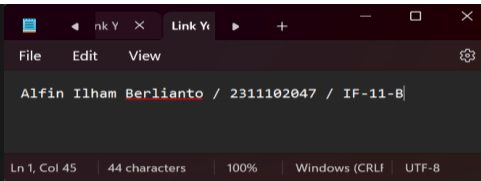
```
18. Hapus node
0. Keluar

Node saat ini : e
Pilih menu [0 - 18] : 18
```



```
Node saat ini : -
Pilih menu [0 - 18] : 0

Keluar dari aplikasi.
```

A screenshot of a code editor window. The window has a title bar with a blue icon, a back arrow, a close button, and a tab labeled 'Link Y'. Below the title bar is a menu bar with 'File', 'Edit', and 'View'. The main text area contains the text 'Alfin Ilham Berlianto / 2311102047 / IF-11-B'. At the bottom of the window, there is a status bar showing 'Ln 1, Col 45', '44 characters', '100%', 'Windows (CRLF)', and 'UTF-8'.

DESKRIPSI PROGRAM

Program tersebut merupakan sebuah program untuk membuat dan menampilkan tree yang merupakan modifikasi dari program unguided 2. Modifikasi yang dilakukan

salah satunya adalah menambahkan menu. Untuk jumlah opsi dalam menu sendiri terdapat 18, sebagian besar dibuat untuk menghubungkan fungsi lama dengan menu. Dalam penghubungan ini, dibuat juga variabel baru yaitu `current` untuk menyimpan data node saat ini (semacam pointer yang menentukan data mana yang akan diubah saat itu). Disamping itu, ada juga beberapa fungsi baru seperti `showChild()`, `showDescendant()`, `inData()`, dan `changeCurrent()`. Berikut adalah uraian terkait rincian fungsi tersebut :

1. `showChild()` : digunakan untuk menampilkan data child dari node.
2. `showDescendant()` : fungsi untuk menampilkan data di bawah node saat ini (`current node`).
3. `inData()` : untuk menginputkan data char.
4. `changeCurrent()` : digunakan untuk mengganti pointer yang menunjuk node saat ini (node tersebut nantinya bisa ditampilkan detailnya, diubah, dihapus, dll.).

KESIMPULAN

Graf dan tree adalah struktur data yang esensial dalam pemrograman, masing-masing digunakan untuk merepresentasikan dan mengelola hubungan antar elemen. Graf terdiri dari simpul (nodes) dan sisi (edges) yang menghubungkan pasangan simpul, digunakan dalam berbagai aplikasi seperti jaringan komputer, media sosial, dan peta. Tree, sejenis graf berarah yang tidak mengandung siklus, digunakan untuk struktur data hirarkis (yang memiliki level / tingkatan) seperti file system dan binary search tree (BST). Keduanya mendukung operasi seperti traversal, pencarian, penyisipan, dan penghapusan, namun tree memiliki struktur yang lebih teratur dengan satu simpul akar dan hierarki induk-anak yang jelas, sedangkan graf lebih fleksibel tanpa hirarki yang kaku. Memahami dan menerapkan graf dan tree dengan tepat dapat meningkatkan efisiensi dalam menyelesaikan berbagai masalah komputasi.

REFERENSI

1. Software Testing Help (2023, 07 Agustus) Implementasi Graf di C++ Menggunakan Adjacency List. Diakses pada 11 Juni 2024, dari <https://g.co/kgs/TVgk199>
- 2, PrepBytes, (2022,20 September) Struktur Data di C++ - Pohon dan Graf. Diakses pada 11 Juni 2024, dari <https://g.co/kgs/5aUwbcd>