

LAPORAN PRAKTIKUM

MODUL 9 GRAPH & TREE



Disusun oleh:
Nabila Shasya Sabrina
NIM: 2311102039

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

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

BAB I

TUJUAN PRAKTIKUM

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

BAB II

DASAR TEORI

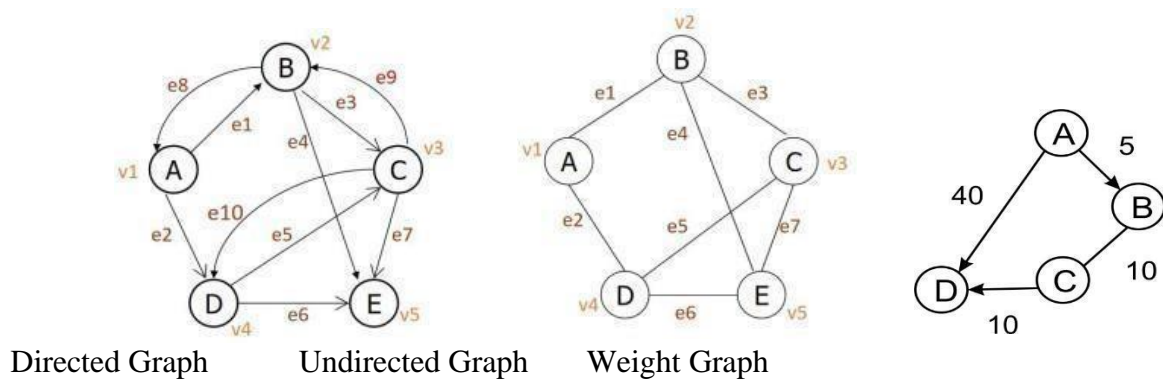
Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

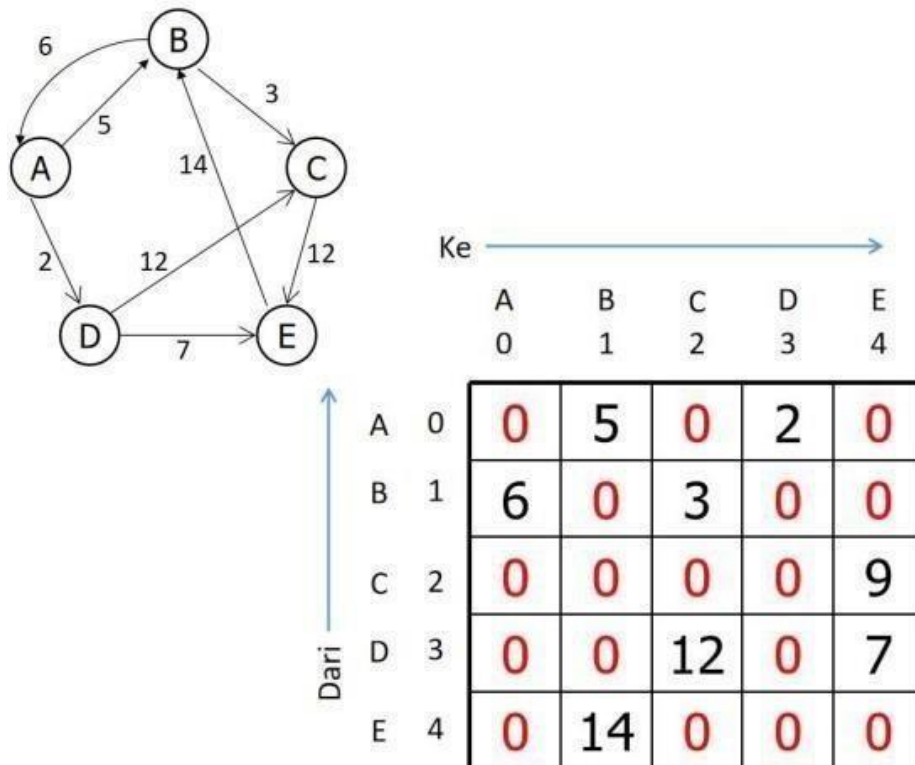
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph



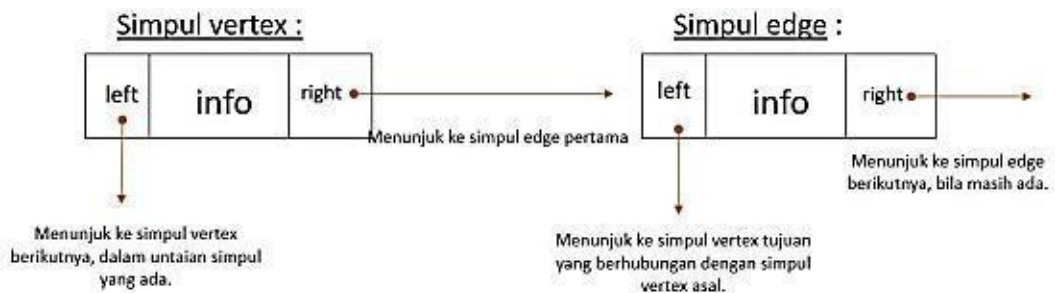
- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
- Weight Graph :** Graph yang mempunyai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

Representasi Graph Representasi dengan Matriks



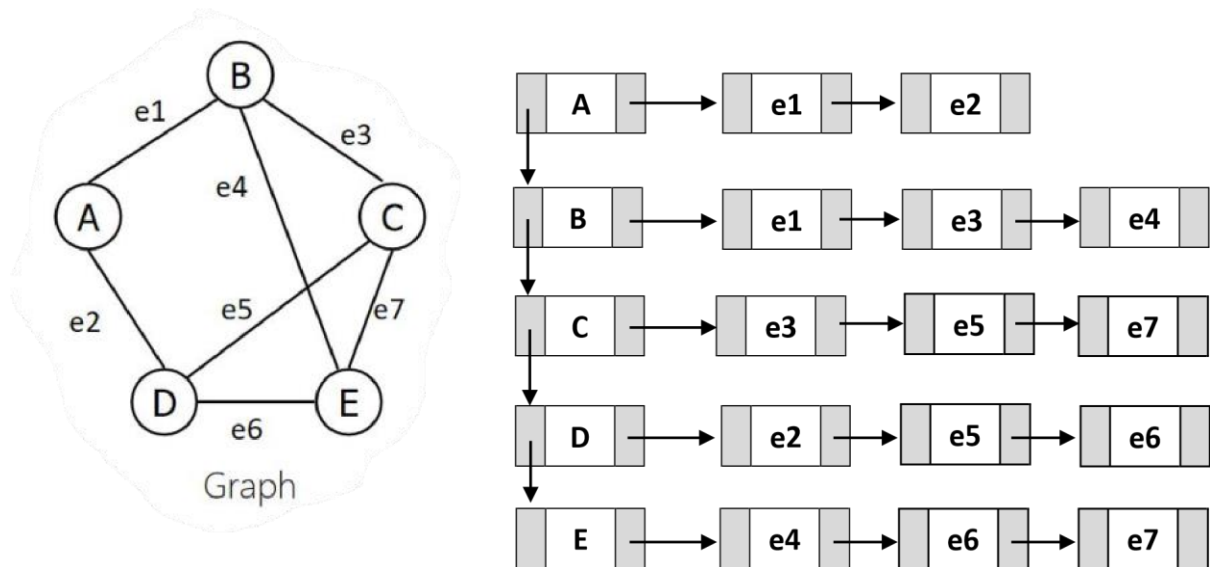
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

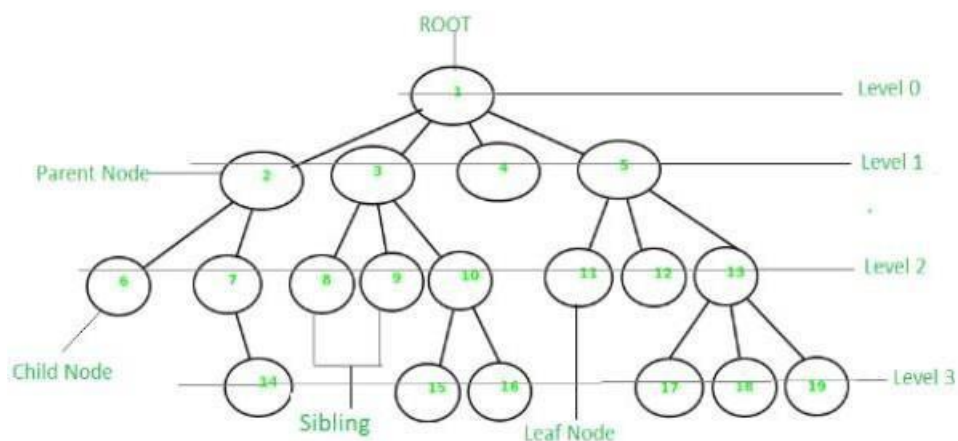
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

1. Tree atau Pohon

Dalam ilmu komputer, pohon 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, di mana 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 hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

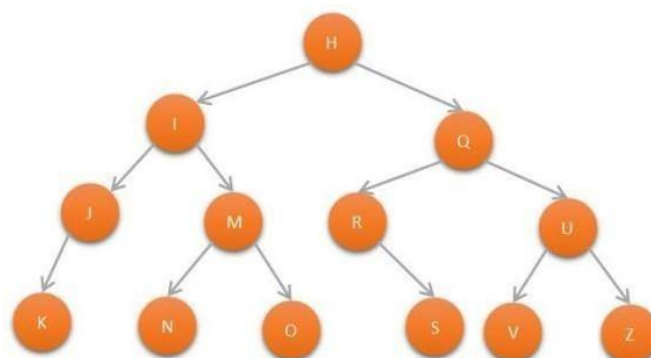


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor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



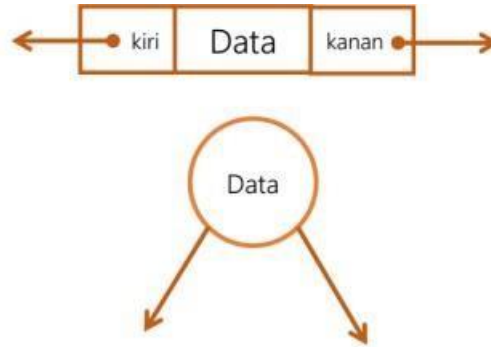
Gambar 1 Struktur Data Binary Tree

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 pointercurrent 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

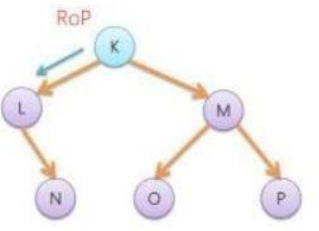
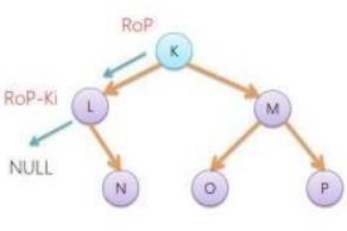
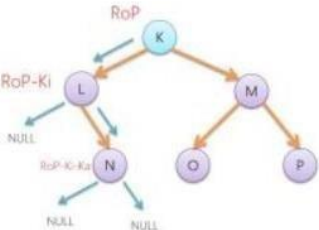
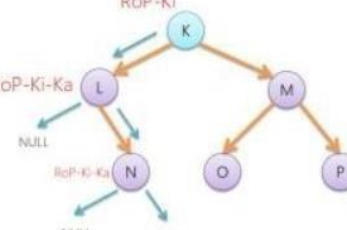
Penelusuran secara pre-order memiliki alur:

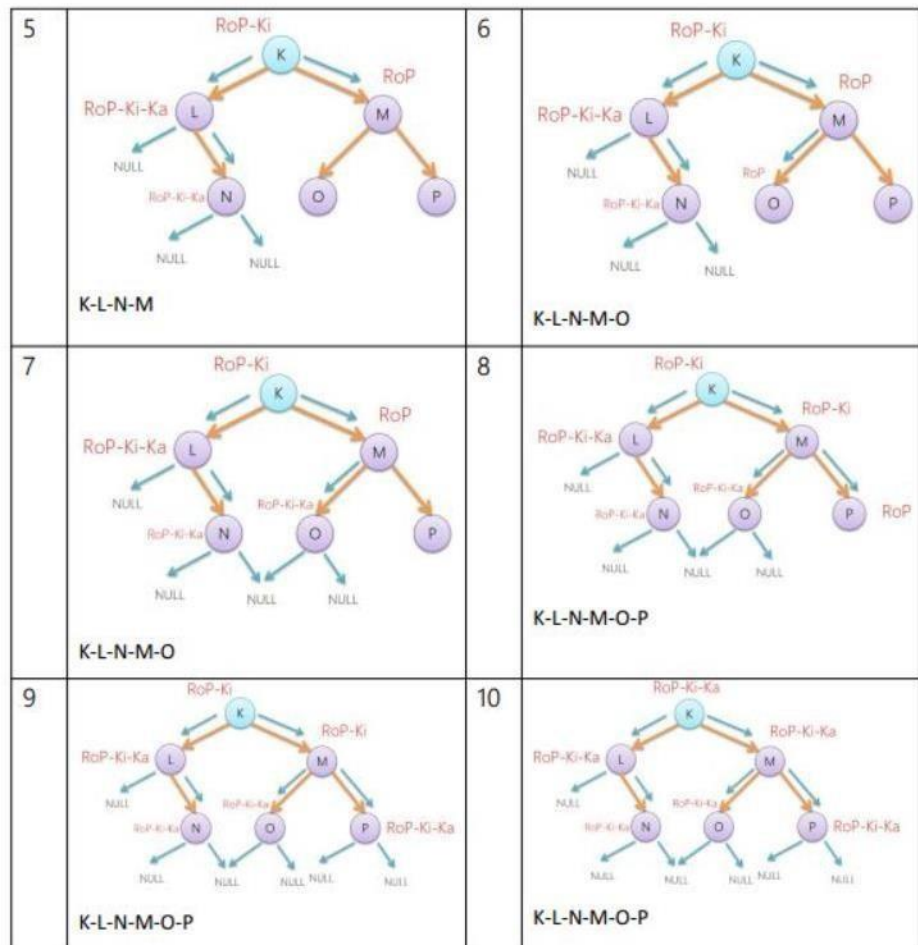
- Cetak data pada simpul root
- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

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

Alur pre-order

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



2. In-Order

Penelusuran secara in-order memiliki alur:

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

kananDapat 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

BAB III

GUIDED

1. 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;
}
```

Screenshoot Program:

```
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)
```

Deskripsi program:

Program ini bertujuan untuk menampilkan graf berbentuk tabel yang merepresentasikan konektivitas antar kota di Jawa Barat dan sekitarnya. Graf tersebut diwakili oleh dua array: array simpul yang berisi nama-nama kota, dan array dua dimensi busur yang berisi bobot atau jarak antar kota yang saling terhubung.

Array simpul terdiri dari tujuh kota yaitu Ciamis, Bandung, Bekasi, Tasikmalaya, Cianjur, Purwokerto, dan Yogyakarta. Sedangkan array busur berukuran 7x7 menyimpan nilai integer yang menunjukkan bobot hubungan antara dua kota, dengan nilai 0 menandakan tidak adanya hubungan langsung antara kota-kota tersebut.

Fungsi `tampilGraph()` bertanggung jawab untuk mencetak graf tersebut. Dengan menggunakan dua loop bersarang, fungsi ini menelusuri setiap kota (baris) dan mencetak kota-kota yang terhubung (kolom) beserta bobotnya. Hasilnya adalah daftar koneksi setiap kota dengan kota lain yang terhubung, yang ditampilkan dalam format yang mudah dibaca. Program ini kemudian memanggil fungsi `tampilGraph()` dalam fungsi `main()` untuk menghasilkan output graf di layar.

2. 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;
        else
            return 0;
        // true
        // false
    }
    // Buat Node Baru
    void buatNode(char data)
    {
        if (isEmpty() == 1)
        {
            root = new Pohon();
            root->data = data;
            root->left = NULL;
            root->right = NULL;
            root->parent = NULL;
            cout << "\n Node " << data << " berhasil dibuat menjadi
root."
                << endl;
        }
        else
        {
            cout << "\n Pohon sudah dibuat" << endl;
        }
    }
    // Tambah Kiri
    Pohon *insertLeft(char data, Pohon *node)
    {
        if (isEmpty() == 1)
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
            return NULL;
        }
        else
        {
            // cek apakah child kiri ada atau tidak
            if (node->left != NULL)
            {
                // kalau ada
                cout << "\n Node " << node->data << " sudah ada child
kiri!"
                    << endl;
                return NULL;
            }
            else
            {
                // kalau tidak ada
                baru = new Pohon();
                baru->data = data;
                baru->left = NULL;
                baru->right = NULL;
                baru->parent = node;
                node->left = baru;
                cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri " << baru->parent->data << endl;
                return baru;
            }
        }
    }
    // Tambah Kanan

```

```

Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan!"
                << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kanan " << baru->parent->data << endl;
            return baru;
        }
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah menjadi
" << data << endl;
        }
    }
}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else

```

```

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

```

```

    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

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

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

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

```



```

        {
            delete node;
        }
    }
}

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

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

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

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

```

```

        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
        << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
        << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
        << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
        << endl;
}

```

```
charateristic();  
}
```

Screenshoot Program:

```
Node A berhasil dibuat menjadi root.  
Node B berhasil ditambahkan ke child kiri A  
Node C berhasil ditambahkan ke child kanan A  
Node D berhasil ditambahkan ke child kiri B  
Node E berhasil ditambahkan ke child kanan B  
Node F berhasil ditambahkan ke child kiri C  
Node G berhasil ditambahkan ke child kiri E  
Node H berhasil ditambahkan ke child kanan E  
Node I berhasil ditambahkan ke child kiri G  
Node J berhasil ditambahkan ke child kanan 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 punya Child kanan)  
  
PreOrder :  
A, B, D, E, G, I, J, H, C, F,  
  
InOrder :  
D, B, I, G, J, E, H, A, F, C,  
  
PostOrder :  
D, I, J, G, H, E, B, F, C, A,  
  
Size Tree : 10  
Height Tree : 5  
Average Node of Tree : 2  
  
Node subtree E berhasil dihapus.  
  
PreOrder :  
A, B, D, E, C, F,
```

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

Deskripsi program:

Program ini adalah implementasi struktur data pohon biner (binary tree). Program ini mencakup berbagai fungsi untuk menginisialisasi pohon, menambahkan node, mengubah data node, menampilkan informasi node, dan melakukan penelusuran pohon dalam berbagai urutan (preorder, inorder, dan postorder). Selain itu, program ini juga menyediakan fungsi untuk menghapus pohon, menghitung ukuran dan tinggi pohon, serta menampilkan karakteristik pohon.

Struktur data Pohon digunakan untuk merepresentasikan setiap node dalam pohon, yang memiliki atribut data, serta pointer ke node anak kiri, anak kanan, dan parent. Fungsi init digunakan untuk menginisialisasi pohon dengan mengatur root menjadi NULL. Fungsi isEmpty memeriksa apakah pohon kosong.

Fungsi buatNode membuat node baru sebagai root jika pohon masih kosong, sedangkan fungsi insertLeft dan insertRight menambahkan node baru sebagai anak kiri atau kanan dari node yang ditentukan. Fungsi update mengubah data dari node tertentu, dan retrieve serta find digunakan untuk menampilkan data node serta informasi terkait seperti parent, sibling, dan children.

Program ini juga menyediakan fungsi untuk menelusuri pohon dalam urutan preorder, inorder, dan postorder, serta menghitung ukuran (size) dan tinggi (height) pohon. Fungsi characteristic menampilkan karakteristik pohon, termasuk ukuran, tinggi, dan rata-rata jumlah node. Fungsi deleteTree dan deleteSub digunakan untuk menghapus seluruh pohon atau subpohon tertentu, sedangkan clear menghapus seluruh pohon dan mengosongkan root.

Dalam fungsi main, program ini membuat root node dengan data 'A' dan menambahkan beberapa node anak dengan data tertentu, kemudian melakukan berbagai operasi seperti mengubah data node, menampilkan informasi node, melakukan penelusuran pohon, dan menampilkan karakteristik pohon sebelum dan sesudah penghapusan subpohon tertentu.

BAB IV

UNGUIDED

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

Output Program

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

      BALI      PALU
BALI      0      3
PALU      4      0

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.
```

Source code:

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

int main() {
    int jumlahSimpul_2311102039;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> jumlahSimpul_2311102039;

    vector<string> simpul(jumlahSimpul_2311102039);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < jumlahSimpul_2311102039; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    vector<vector<int>>> bobot(jumlahSimpul_2311102039,
        vector<int>(jumlahSimpul_2311102039, 0));
    cout << "Silakan masukan bobot antar simpul" << endl;
    for (int i = 0; i < jumlahSimpul_2311102039; i++) {
        for (int j = 0; j < jumlahSimpul_2311102039; j++) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << setw(15) << " ";
    for (int i = 0; i < jumlahSimpul_2311102039; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul_2311102039; i++) {
        cout << setw(15) << simpul[i];
```

```

        for (int j = 0; j < jumlahSimpul_2311102039; j++) {
            cout << setw(15) << bobot[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Screenshot Output:

```

Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul
Simpul 1: Purwokerto
Simpul 2: Bekasi
Silakan masukkan bobot antar simpul
Purwokerto--> Purwokerto = 0
Purwokerto--> Bekasi = 6
Bekasi--> Purwokerto = 5
Bekasi--> Bekasi = 0

```

	Purwokerto	Bekasi
Purwokerto	0	6
Bekasi	5	0

Deskripsi Program:

Program ini bertujuan untuk membuat dan menampilkan graf berbasis input pengguna yang merepresentasikan jarak antar kota. Program ini meminta pengguna untuk memasukkan jumlah simpul (kota) dan nama-nama simpul tersebut, serta bobot atau jarak antar simpul.

Pertama, pengguna diminta untuk memasukkan jumlah simpul yang diinginkan, kemudian nama-nama simpul tersebut dimasukkan satu per satu. Setelah itu, pengguna memasukkan bobot atau jarak antara setiap pasangan simpul. Semua input ini disimpan dalam struktur data vektor untuk nama simpul dan matriks vektor dua dimensi untuk bobot.

Setelah semua data dimasukkan, program menampilkan matriks bobot dalam format tabel yang mudah dibaca. Tabel ini menunjukkan jarak antar setiap pasangan simpul, dengan baris dan kolom yang diberi label sesuai dengan nama simpul yang telah dimasukkan.

2. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user!

```

#include <iostream>
using namespace std;

/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data_2311102039;
    Pohon *left, *right, *parent;
};

```

```

Pohon *root, *baru;

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

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

// Buat Node Baru
void buatNode(char data_2311102039)
{
    if (isEmpty())
    {
        root = new Pohon();
        root->data_2311102039 = data_2311102039;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data_2311102039 << " berhasil
dibuat menjadi root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data_2311102039, Pohon *node)
{
    if (isEmpty())
    {
        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_2311102039 << "
sudah ada child kiri!" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data_2311102039 = data_2311102039;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
        }
    }
}

```

```

        cout << "\n Node " << data_2311102039 << " berhasil
        ditambahkan ke child kiri " << baru->parent-
        >data_2311102039 << endl;
        return baru;
    }
}

// Tambah Kanan
Pohon *insertRight(char data_2311102039, Pohon *node)
{
    if (isEmpty())
    {
        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_2311102039 << "
            sudah ada child kanan!" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data_2311102039 = data_2311102039;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data_2311102039 << " berhasil
            ditambahkan ke child kanan " << baru->parent-
            >data_2311102039 << endl;
            return baru;
        }
    }
}

// Ubah Data Tree
void update(char data_2311102039, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" <<
            endl;
        else
        {
            char temp = node->data_2311102039;
            node->data_2311102039 = data_2311102039;
            cout << "\n Node " << temp << " berhasil diubah
            menjadi " << data_2311102039 << endl;
        }
    }
}

```



```

    }
}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        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_2311102039
            << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (isEmpty())
    {
        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_2311102039
            << endl;
            cout << " Root : " << root->data_2311102039 << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" <<
            endl;
            else
                cout << " Parent : " << node->parent-
            >data_2311102039 << endl;
            if (node->parent != NULL && node->parent->left !=
            node &&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left-
            >data_2311102039 << endl;
            else if (node->parent != NULL && node->parent->right
            != node &&
                node->parent->left == node)
                cout << " Sibling : " << node->parent->right-
            >data_2311102039 << 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_2311102039 << endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child
kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right-
>data_2311102039 << endl;
        }
    }
}

// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (isEmpty())
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data_2311102039 << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node = root)
{
    if (isEmpty())
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data_2311102039 << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node = root)
{
    if (isEmpty())
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data_2311102039 << ", ";
        }
    }
}

```

```

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (isEmpty())
        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 (isEmpty())
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data_2311102039 <<
            " berhasil dihapus." << endl;
    }
}

// Hapus Tree
void clear()
{
    if (isEmpty())
        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 (isEmpty())
    {

```

```

        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 (isEmpty())
    {
        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 characteristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

void menu()
{
    int pilihan;
    char data;
    Pohon *node = NULL;

```

```

do
{
    cout << "\n\nMenu:\n";
    cout << "1. Buat Node Root\n";
    cout << "2. Tambah Node Kiri\n";
    cout << "3. Tambah Node Kanan\n";
    cout << "4. Ubah Data Node\n";
    cout << "5. Lihat Isi Data Node\n";
    cout << "6. Cari Data Node\n";
    cout << "7. Traversal Pre-Order\n";
    cout << "8. Traversal In-Order\n";
    cout << "9. Traversal Post-Order\n";
    cout << "10. Hapus SubTree\n";
    cout << "11. Hapus Seluruh Tree\n";
    cout << "12. Lihat Karakteristik Tree\n";
    cout << "13. Keluar\n";
    cout << "Pilih opsi: ";
    cin >> pilihan;

    switch (pilihan)
    {
    case 1:
        cout << "Masukkan data root: ";
        cin >> data;
        buatNode(data);
        break;
    case 2:
        cout << "Masukkan data node kiri: ";
        cin >> data;
        if (root)
        {
            node = insertLeft(data, root);
        }
        else
        {
            cout << "Root tidak ada, buat node root terlebih
dahulu.\n";
        }
        break;
    case 3:
        cout << "Masukkan data node kanan: ";
        cin >> data;
        if (root)
        {
            node = insertRight(data, root);
        }
        else
        {
            cout << "Root tidak ada, buat node root terlebih
dahulu.\n";
        }
        break;
    case 4:
        cout << "Masukkan data node yang ingin diubah: ";
        cin >> data;
        update(data, root);
        break;
    case 5:
        cout << "Masukkan data node yang ingin dilihat: ";
        cin >> data;
        retrieve(root);
    }
}

```

```

        break;
    case 6:
        cout << "Masukkan data node yang ingin dicari: ";
        cin >> data;
        find(root);
        break;
    case 7:
        cout << "\nPreOrder Traversal: ";
        preOrder(root);
        cout << endl;
        break;
    case 8:
        cout << "\nInOrder Traversal: ";
        inOrder(root);
        cout << endl;
        break;
    case 9:
        cout << "\nPostOrder Traversal: ";
        postOrder(root);
        cout << endl;
        break;
    case 10:
        cout << "Masukkan data node subtree yang ingin
dihapus: ";
        cin >> data;
        deleteSub(root);
        break;
    case 11:
        clear();
        break;
    case 12:
        charateristic();
        break;
    case 13:
        cout << "Keluar program.\n";
        break;
    default:
        cout << "Pilihan tidak valid.\n";
    }
} while (pilihan != 13);
}

int main()
{
    init();
    menu();
    return 0;
}

```

Screenshot Output:

```
Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Ubah Data Node
5. Lihat Isi Data Node
6. Cari Data Node
7. Traversal Pre-Order
8. Traversal In-Order
9. Traversal Post-Order
10. Hapus SubTree
11. Hapus Seluruh Tree
12. Lihat Karakteristik Tree
13. Keluar
Pilih opsi: 1
Masukkan data root: A

Node A berhasil dibuat menjadi root.
```

```
Pilih opsi: 2
Masukkan data node kiri: B

Node B berhasil ditambahkan ke child kiri A
```

```
Pilih opsi: 3
Masukkan data node kanan: C

Node C berhasil ditambahkan ke child kanan A
```

```
Pilih opsi: 7

PreOrder Traversal: A, B, C,
```

```
Pilih opsi: 12

Size Tree : 3
Height Tree : 2
Average Node of Tree : 1
```

```
Pilih opsi: 13
Keluar program.
```

Deskripsi Program:

Program ini merupakan implementasi dari struktur data pohon biner. Dalam program ini, terdapat berbagai fungsi untuk mengelola dan memanipulasi pohon biner. Program dimulai dengan mendeklarasikan struktur pohon biner yang terdiri dari data (tipe char), serta pointer ke anak kiri, anak kanan, dan orang tua dari node tersebut. Fungsi-fungsi yang diimplementasikan meliputi:

1. Inisialisasi: Menginisialisasi pohon dengan root yang diset NULL.
2. Pengecekan Node Kosong: Memeriksa apakah pohon biner masih kosong.
3. Pembuatan Node Baru: Membuat node baru dan menjadikannya root jika

pohon masih kosong.

4. Penambahan Node Kiri dan Kanan: Menambahkan node baru sebagai anak kiri atau kanan dari node tertentu.
5. Pengubahan Data Node: Mengubah data pada node tertentu.
6. Penelusuran Data Node: Menampilkan data dari node tertentu.
7. Pencarian Data Node: Mencari dan menampilkan informasi lengkap dari node tertentu, termasuk data, root, parent, sibling, dan child.
8. Traversal Pohon: Menelusuri pohon dalam tiga cara: pre-order, in-order, dan post-order.
9. Penghapusan Node dan SubTree: Menghapus node tertentu atau subtree.
10. Penghapusan Seluruh Pohon: Menghapus seluruh pohon.
11. Karakteristik Pohon: Menampilkan ukuran pohon, tinggi pohon, dan rata-rata node per tingkat.

BAB V

KESIMPULAN

Graph dan Tree merupakan dua struktur data yang sangat penting dalam dunia pemrograman dan ilmu komputer. Graph adalah representasi visual dari kumpulan objek yang terhubung satu sama lain melalui serangkaian edge atau sisi. Graph digunakan untuk merepresentasikan hubungan antara entitas dalam berbagai konteks, seperti jaringan komputer, media sosial, rute perjalanan, dan sebagainya. Graph memiliki beragam jenis, termasuk directed dan undirected, weighted dan unweighted, serta cyclic dan acyclic.

Sementara itu, Tree adalah salah satu jenis khusus dari graph yang memiliki struktur hirarkis, di mana setiap node memiliki tepat satu parent kecuali root node, dan tidak ada siklus (acyclic). Tree sering digunakan untuk merepresentasikan hubungan hierarkis, seperti struktur file pada sistem operasi, organisasi perusahaan, dan struktur data seperti Binary Search Tree (BST), AVL Tree, dan Red-Black Tree.

Keduanya memiliki berbagai operasi dan algoritma yang digunakan untuk manipulasi dan analisis data, seperti traversal, pencarian, penyisipan, penghapusan, dan banyak lagi. Graph dan Tree memiliki aplikasi luas dalam berbagai bidang, termasuk pemodelan jaringan, algoritma pencarian, optimisasi rute, pengelolaan basis data, kecerdasan buatan, dan masih banyak lagi. Pemahaman yang kuat tentang kedua struktur data ini sangat penting bagi para pengembang perangkat lunak untuk merancang solusi yang efisien dan efektif dalam menangani berbagai tantangan komputasi.

DAFTAR PUSTAKA

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.