

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
ALGORITMA SEARCHING**



Disusun Oleh :

NAMA : Aulia Radix Putra Winarko

NIM : 2311102056

Dosen

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

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

A. Dasar Teori

Graph dan tree merupakan dua struktur data penting yang sering digunakan dalam berbagai aplikasi pemrograman. Graph merepresentasikan hubungan antar entitas, sedangkan tree merepresentasikan hierarki. Dalam C++, implementasi graph dan tree dapat dilakukan dengan berbagai cara, dan pilihan cara yang tepat tergantung pada kebutuhan aplikasi.

1. Graph

Graph terdiri dari dua elemen utama: vertex (simpul) dan edge (sisi). Vertex merepresentasikan entitas, sedangkan edge merepresentasikan hubungan antar vertex. Edge dapat berarah (directed graph) atau tidak berarah (undirected graph).

Ada beberapa cara untuk mengimplementasikan graph dalam C++. Cara yang umum digunakan adalah dengan menggunakan struktur data array atau list untuk menyimpan vertex, dan struktur data adjacency list atau adjacency matrix untuk menyimpan edge.

2. Tree

Tree terdiri dari node (simpul) dan edge (sisi). Node merepresentasikan entitas, sedangkan edge merepresentasikan hubungan antar node. Tree memiliki satu node khusus yang disebut root (akar). Setiap node dalam tree, kecuali root, memiliki satu parent (induk). Node-node yang terhubung ke node yang sama disebut siblings (saudara).

Tree dapat diimplementasikan dalam C++ dengan menggunakan struktur data array atau list

B. Guided

Guided 1

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

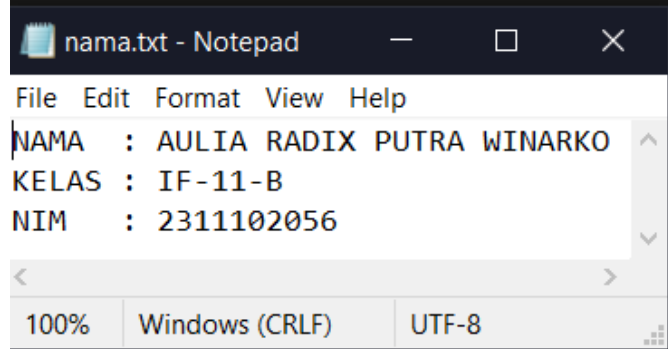
int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

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

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

Screenshots Output

```
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS D:\C++\Praktikum struktur data\Modul9> 
```



The screenshot shows a Notepad window titled 'nama.txt - Notepad'. The text inside the window is as follows:

```
File Edit Format View Help
NAMA : AULIA RADIX PUTRA WINARKO
KELAS : IF-11-B
NIM   : 2311102056
```

At the bottom of the Notepad window, there is a status bar showing '100%', 'Windows (CRLF)', and 'UTF-8'.

Deskripsi:

Program tersebut merupakan implementasi sederhana dari representasi graf menggunakan matriks ketetanggaan. Berikut adalah penjelasan singkat mengenai program tersebut :

1. Program tersebut menggunakan beberapa header file, yaitu `iostream` dan `iomanip`, yang digunakan untuk input/output dan manipulasi tampilan output.
2. Program ini menggunakan `namespace std` agar tidak perlu menuliskan `std::` di depan objek-objek standar dari C++.
3. Terdapat sebuah array `simpul` yang berisi daftar simpul/simpul pada graf yang akan ditampilkan.
4. Terdapat matriks `busur` berukuran 7x7 yang merepresentasikan hubungan antara simpul-simpul pada graf. Angka-angka pada matriks tersebut menunjukkan bobot atau jarak antara simpul-simpul tersebut. Jika nilai elemen adalah 0, itu berarti

tidak ada hubungan antara simpul-simpul tersebut.

5. Terdapat fungsi `tampilGraph()` yang digunakan untuk menampilkan graf. Fungsi ini menggunakan perulangan untuk mencetak nama simpul dan busur yang terhubung ke simpul tersebut. Hanya busur dengan bobot yang lebih besar dari 0 yang akan ditampilkan.
6. Fungsi `main()` memanggil `tampilGraph()` untuk menampilkan graf yang direpresentasikan oleh matriks `busur`.
7. Program ini mengembalikan nilai 0 sebagai indikasi bahwa program berjalan dengan sukses.

Program ini menggambarkan graf dengan 7 simpul dan hubungan antara simpul-simpul tersebut ditentukan oleh matriks `busur`. Output program akan mencetak nama simpul dan hubungan antara simpul-simpul tersebut beserta bobotnya.

Guided 2

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // false
}
// Buat Node Baru
```

```

void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat menjadi root."
<< endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child kiri!"
<< endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
        }
    }
}

```

```

        cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri " << baru->parent->data << endl;
        return baru;
    }
}
// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan!" << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kanan" << baru->parent->data << endl;
            return baru;
        }
    }
}
// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;

```

```

    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah menjadi " <<
data << endl;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
        }
    }
}

```



```

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

```

```

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

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

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

```

```

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

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

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

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

```

```

    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

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

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

```

```

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

```

Screenshots Output

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kananA

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kananB

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kananE

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kananG

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C

Data Node : C

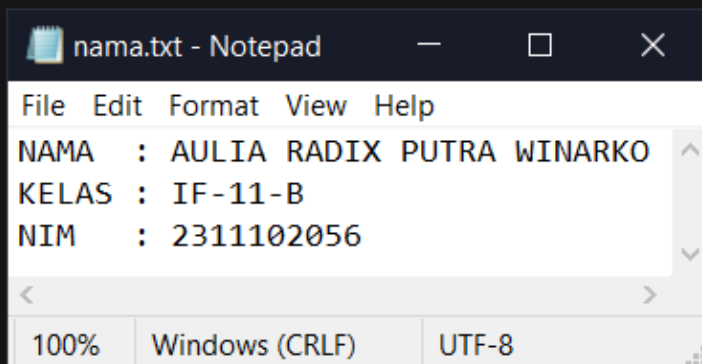
Root : A

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.

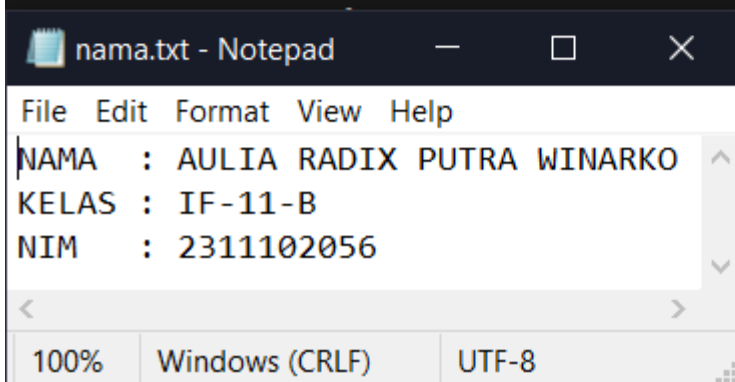
PreOrder :

A, B, D, E, C, F,

Size Tree : 6

Height Tree : 3

Average Node of Tree : 2



Modul19>

Deskripsi:

Program di atas merupakan implementasi binary tree dalam bahasa C++. Berikut adalah penjelasan singkat mengenai program tersebut :

1. Program ini menggunakan header file `` yang digunakan untuk input/output dalam C++.
2. Program ini mendefinisikan struktur `Pohon` yang merupakan simpul dalam binary tree. Setiap simpul memiliki data bertipe `char` dan memiliki pointer ke simpul kiri, simpul kanan, dan simpul parent.
3. Program ini mendeklarasikan variabel `root` dan `baru` yang merupakan pointer ke simpul root dan simpul yang akan ditambahkan.
4. Terdapat fungsi `init()` yang digunakan untuk menginisialisasi binary tree dengan mengatur `root` menjadi NULL.
5. Fungsi `isEmpty()` digunakan untuk memeriksa apakah binary tree kosong atau tidak. Jika `root` adalah NULL, maka dikembalikan nilai 1 (true), jika tidak, dikembalikan nilai 0 (false).
6. Fungsi `buatNode()` digunakan untuk membuat simpul baru dan menjadikannya sebagai root jika binary tree masih kosong.
7. Fungsi `insertLeft()` dan `insertRight()` digunakan untuk menambahkan simpul anak kiri dan anak kanan dari sebuah simpul yang sudah ada dalam binary tree.
8. Fungsi `update()` digunakan untuk mengubah data pada sebuah simpul dalam binary tree.
9. Fungsi `retrieve()` digunakan untuk melihat data pada sebuah simpul dalam binary tree
10. Fungsi `find()` digunakan untuk mencari informasi tentang sebuah simpul dalam binary tree, seperti data, root, parent, sibling, dan anak kiri dan kanan.
11. Fungsi-fungsi `preOrder()`, `inOrder()`, dan `postOrder()` digunakan untuk melakukan penelusuran (traversal) pada binary tree dengan metode pre-order, in-order, dan post-order.
12. Fungsi `deleteTree()` digunakan untuk menghapus seluruh simpul dalam binary tree.
13. Fungsi `deleteSub()` digunakan untuk menghapus sebuah subtree dalam binary tree.

14. Fungsi `clear()` digunakan untuk menghapus seluruh binary tree.
15. Fungsi `size()` digunakan untuk menghitung jumlah simpul dalam binary tree.
16. Fungsi `height()` digunakan untuk menghitung tinggi (level) binary tree.
17. Fungsi `characteristic()` digunakan untuk menampilkan karakteristik dari binary tree, seperti ukuran, tinggi, dan rata-rata simpul.
18. Fungsi `main()` merupakan fungsi utama yang melakukan sejumlah operasi pada binary tree, seperti membuat simpul, menambahkan anak kiri dan kanan, mengubah data simpul, melakukan penelusuran, dan menghapus subtree.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[10];
int busur[10][10];
void inputSimpul(int length)
{
    cout << "Silahkan masukkan nama simpul : " << endl;
    for (int i = 0; i < length; i++)
    {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }
}
void inputBusur(int length)
{
    cout << "Silahkan masukan bobot antara simpul: " << endl;
    for (int i = 0; i < length; i++)
    {
        for (int j = 0; j < length; j++)
        {
            cout << simpul[i] << " ==> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }
}
void tampilGrph(int length)
```

```

{
    cout << endl;
    for (int i = 0; i < length; i++)
    {
        cout << " \t" << simpul[i];
    }
    cout << endl;
    for (int baris = 0; baris < length; baris++)
    {
        cout << simpul[baris] << "\t";
        for (int kolom = 0; kolom < length; kolom++)
        {
            cout << busur[baris][kolom] << "\t";
        }
        cout << endl;
    }
}
int main()
{
    int length;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> length;
    inputSimpul(length);
    inputBusur(length);
    tampilGrph(length);
    return 0;
}

```

Screenshots Output

```

Silahkan masukkan jumlah simpul : 4
Silahkan masukkan nama simpul :
Simpul 1 : ZEE
Simpul 2 : FREYA
Simpul 3 : FENI
Simpul 4 : ELA
Silahkan masukan bobot antara simpul:
ZEE ==> ZEE = 2
ZEE ==> FREYA = 2
ZEE ==> FENI = 1
ZEE ==> ELA = 2
FREYA ==> ZEE = 2
FREYA ==> FREYA = 2
FREYA ==> FENI = 1
FREYA ==> ELA = 2
FENI ==> ZEE = 2
FENI ==> FREYA = 1
FENI ==> FENI = 2
FENI ==> ELA = 2
ELA ==> ZEE = 1
ELA ==> FREYA = 2
ELA ==> FENI = 2
ELA ==> ELA = 1

```

```

nama.txt - Notepad
File Edit Format View Help
NAMA : AULIA RADIX PUTRA WINARKO
KELAS : IF-11-B
NIM : 2311102056
100% Windows (CRLF) UTF-8

```

	ZEE	FREYA	FENI	ELA
ZEE	2	2	1	2
FREYA	2	2	1	2
FENI	2	1	2	2
ELA	1	2	2	1

```
PS D:\C++\Praktikum struktur data\Modul9> 
```

```

nama.txt - Notepad
File Edit Format View Help
NAMA : AULIA RADIX PUTRA WINARKO
KELAS : IF-11-B
NIM : 2311102056
100% Windows (CRLF) UTF-8

```

Deskripsi:

Kode diatas adalah alat dasar yang efektif untuk menginput dan menampilkan graf berbobot, memberikan dasar yang kuat untuk pengembangan lebih lanjut atau penggunaan dalam konteks pendidikan dan analisis graf.

Unguided 2

```
#include <iostream>
using namespace std;
int AuliaRadixPutraWinarko_2311102056;
/// 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;
        }
    }
}

Pohon *findParent(char data, Pohon *node = root)
{
    if (!root)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (!node)
        {
            return NULL;
        }
        else
        {
            if (node->data == data)
            {
                return node;
            }
        }
    }
}

```

```

        else
        {
            Pohon *foundNode = findParent(data, node->left);
            if (foundNode == NULL)
            {
                foundNode = findParent(data, node->right);
            }
            return foundNode;
        }
    }
}

// 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()
{
    init();
    int menu;
    char data, data2;
    Pohon *selectedNode;
    while (true)
    {
        cout << "\n=== MENU ===" << endl;
        cout << "1. Buat Node Baru" << endl;
        cout << "2. Insert Left" << endl;
        cout << "3. Insert Right" << endl;
        cout << "4. Update Node" << endl;
        cout << "5. Lihat Data Node" << endl;
        cout << "6. Find Node" << endl;
        cout << "7. PreOrder" << endl;
        cout << "8. InOrder" << endl;
        cout << "9. PostOrder" << endl;
        cout << "10. Hapus SubTree" << endl;
        cout << "11. Clear Tree" << endl;
        cout << "12. Karakteristik Tree" << endl;
        cout << "13. Keluar" << endl;
        cout << "Masukkan pilihan: ";
        cin >> menu;
    }
}

```

```

switch (menu)
{
case 1:
    cout << "\nMasukkan data node: ";
    cin >> data;
    buatNode(data);
    break;
case 2:
    cout << "\nMasukkan data node: ";
    cin >> data;
    cout << "Pilih parent node: ";
    cin >> data2;
    selectedNode = findParent(data2, root);
    if (selectedNode != NULL)
        insertLeft(data, selectedNode);
    break;
case 3:
    cout << "\nMasukkan data node: ";
    cin >> data;
    cout << "Pilih parent node: ";
    cin >> data2;
    selectedNode = findParent(data2, root);
    if (selectedNode != NULL)
        insertRight(data, selectedNode);
    break;
case 4:
    cout << "\nMasukkan data node yang ingin diubah: ";
    cin >> data;
    selectedNode = findParent(data, root);
    if (selectedNode != NULL)
    {
        cout << "Masukkan data baru: ";
        cin >> data2;
        update(data2, selectedNode);
    }
    break;
case 5:
    cout << "\nMasukkan data node yang ingin dilihat: ";
    cin >> data;
    selectedNode = findParent(data, root);
    if (selectedNode != NULL)
        retrieve(selectedNode);
    break;
case 6:
    cout << "\nMasukkan data node yang ingin dicari: ";

```

```

        cin >> data;
        selectedNode = findParent(data, root);
        find(selectedNode);
        break;
    case 7:
        cout << "\nPreOrder : ";
        preOrder();
        cout << endl;
        break;
    case 8:
        cout << "\nInOrder : ";
        inOrder();
        cout << endl;
        break;
    case 9:
        cout << "\nPostOrder : ";
        postOrder();
        cout << endl;
        break;
    case 10:
        cout << "\nMasukkan data node yang ingin dihapus subtreenya:
";

        cin >> data;
        selectedNode = findParent(data, root);
        if (selectedNode != NULL)
            deleteSub(selectedNode);
        break;
    case 11:
        clear();
        break;
    case 12:
        charateristic();
        break;
    case 13:
        return 0;
    default:
        cout << "\nPilihan tidak valid!" << endl;
        break;
    }
}
return 0;
}

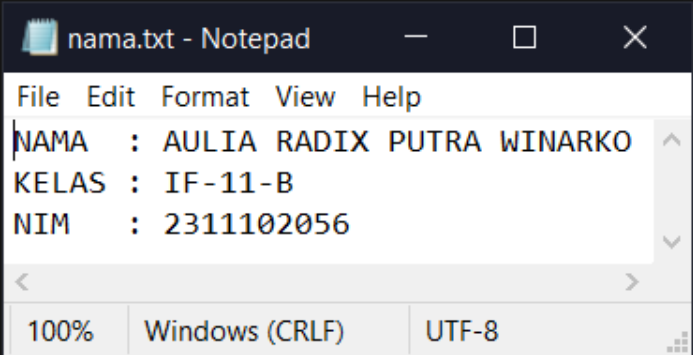
```

Screenshots Output

```
=== MENU ===
1. Buat Node Baru
2. Insert Left
3. Insert Right
4. Update Node
5. Lihat Data Node
6. Find Node
7. PreOrder
8. InOrder
9. PostOrder
10. Hapus SubTree
11. Clear Tree
12. Karakteristik Tree
13. Keluar
Masukkan pilihan: 1

Masukkan data node: 3

Node 3 berhasil dibuat menjadi root.
```



Deskripsi:

Kode di atas adalah program untuk mengelola dan memanipulasi pohon biner (binary tree) dengan berbagai operasi dasar seperti penambahan node, penghapusan node, pembaruan node, penelusuran pohon, dan pencarian node.

D. Kesimpulan

Kesimpulan yang dapat diambil dari laporan praktikum dengan “Modul 9 : Graph dan Tree” adalah bahwa praktikum ini bertujuan untuk mempelajari fungsi dari penggunaan Graph dan Tree dalam pemrograman. Praktisi jadi bisa memahami penggunaan dari konsep tersebut dan bagaimana kegunaan dari konsep tersebut dalam program yang dibuat, serta setelah melakukan pratikum, mahasiswa mampu menerapkan konsep tersebut didalam program yang mereka buat.