

**LAPORAN PRAKTIKUM**  
**MODUL 9**  
**GRAPH DAN TREE**



**Disusun oleh:**

**Aldi Ransi**

**NIM :**

**2311102328**

**Dosen Pengampu:**

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

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

# **BAB I**

## **TUJUAN PRAKTIKUM**

Pada program kali ini di harapkan agar mahasiswa dapat:

1. Mahasiswa diharapkan mampu memahami graph dan tree
2. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

## **BAB II**

### **DASAR TEORI**

#### **Graph**

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk 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  $E$  sebagai titik atau edge

#### **Jenis-jenis Graph**

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

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.

#### **Tree atau Pohon**

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. Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child). tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree. Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

### **Operasi pada Tree**

Pada operasi creat di bagi menjadi beberapa bagian yaitu:

- a. Create: digunakan untuk membentuk binary tree baru yang masih kosong.
- b. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- c. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- d. Insert: digunakan untuk memasukkan sebuah node kedalam tree.
- e. Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- f. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- g. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- h. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- i. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
- j. Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

## **BAB III**

### **GUIDED**

#### **1. GUIDED 1**

##### **SOURCE CODE**

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

string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogyakarta",};

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)

:\\Master\\Institut Teknologi Telkom Purwokerto\\Semester 2\\P

```

## DESKRIPSI PROGRAM

Kode di atas mengimplementasikan graf menggunakan representasi matriks adjacency. Fungsi tampilGraph() digunakan untuk menampilkan hubungan antara simpul-simpul dalam graf berdasarkan matriks busur yang diberikan. Dengan

demikian, kita dapat dengan jelas melihat keterhubungan antara simpul-simpul dalam graf menggunakan representasi m.

## 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;
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }

```

## SCREENSHOOT PROGRAM

```

Node A berhasil dibuat menjadi root.
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
Node B berhasil ditambahkan ke child kiri A
        if( !node ){
            return 0;
        }
Node C berhasil ditambahkan ke child kanan A
        int heightKiri = height( node->left );
        int heightKanan = height( node->right );
Node D berhasil ditambahkan ke child kiri B
        if( heightKiri >= heightKanan ){
            return heightKiri + 1;
        }
Node E berhasil ditambahkan ke child kanan B
        return heightKanan + 1;
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kanan E
// karakteristik tree
void characteristic()
Node I berhasil ditambahkan ke child kiri G
        cout << "\n Size Tree : " << size() << endl;
        cout << " Height Tree : " << height() << endl;
Node J berhasil ditambahkan ke child kanan G << size() / height() << endl;
Node C berhasil diubah menjadi Z
        int main()
Node Z berhasil diubah menjadi 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,

Buat tree terlebih dahulu!

## SCREENSHOOT PROGRAM

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child ki

Node C berhasil ditambahkan ke child ka

Node D berhasil ditambahkan ke child ki

Node E berhasil ditambahkan ke child ka

Node F berhasil ditambahkan ke child ki

Node G berhasil ditambahkan ke child ki

Node H berhasil ditambahkan ke child ka

Node I berhasil ditambahkan ke child ki

## DESKRIPSI PROGRAM

Program Binary Tree ini memungkinkan kita untuk membuat, mengubah, dan menghapus node dalam sebuah pohon biner. Selain itu, program ini juga menyediakan fungsi-fungsi untuk melakukan penelusuran (traversal) pada pohon biner. Dengan menggunakan program ini, kita dapat dengan mudah memanipulasi dan menganalisis struktur data pohon biner.

## UNGUIDED

### 1. 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 <string>
#include <vector>
using namespace std;

int main() {
    int aldi_2311102328jumlahSimpul;
    cout << "Silahkan masukan jumlah simpul = ";
    cin >> aldi_2311102328jumlahSimpul;

    vector<string> nama_simpul(aldi_2311102328jumlahSimpul);
```

```

        vector<vector<int>>> bobot(aldi_2311102328jumlahSimpul,
vector<int>(aldi_2311102328jumlahSimpul));

        for (int i = 0; i < aldi_2311102328jumlahSimpul; ++i) {
            cout << "Silahkan masukkan nama simpul " << i + 1 << " =
";

            cin >> nama_simpul[i];
        }

        cout << "Silahkan masukkan bobot antar simpul\n";

        for (int i = 0; i < aldi_2311102328jumlahSimpul; ++i) {
            for (int j = 0; j < aldi_2311102328jumlahSimpul; ++j)
            {
                cout << nama_simpul[i] << "-->" << nama_simpul[j] <<
" = ";

                cin >> bobot[i][j];
            }
        }

        // Output matriks jarak antar
        kotacout << "\n\t";
        for (int i = 0; i < aldi_2311102328jumlahSimpul; ++i) {
            cout << nama_simpul[i] << "\t";
        }
        cout << "\n";

        for (int i = 0; i < aldi_2311102328jumlahSimpul; ++i) {cout
<< nama_simpul[i] << "\t";
            for (int j = 0; j < aldi_2311102328jumlahSimpul; ++j)
            {
                cout << bobot[i][j] << "\t";
            }
            cout << "\n";
        }
    }
}

```

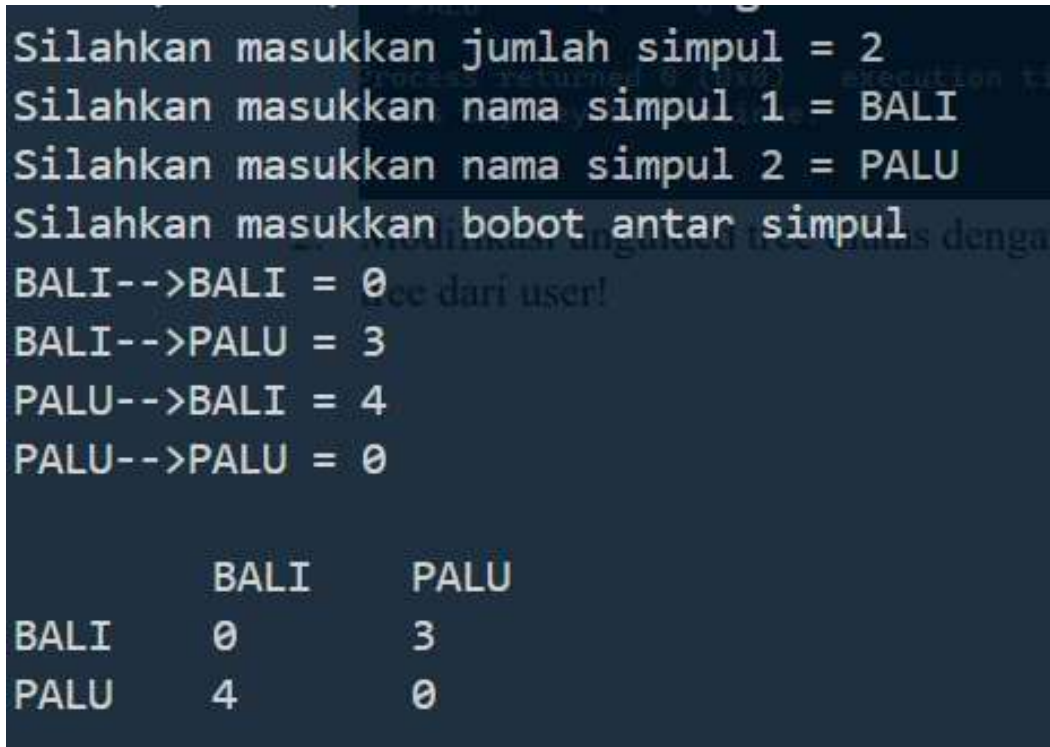
```

    }

    return 0;
}

```

## SCREENSHOOT PROGRAM



```

Silahkan masukkan jumlah simpul = 2
Silahkan masukkan nama simpul 1 = BALI
Silahkan masukkan nama simpul 2 = PALU
Silahkan masukkan bobot antar simpul
BALI-->BALI = 0
BALI-->PALU = 3
PALU-->BALI = 4
PALU-->PALU = 0

      BALI    PALU
BALI   0      3
PALU   4      0

```

## DESKRIPSI PROGRAM

pada Program ini memungkinkan user memasukkan informasi jumlah simpul, nama simpul, dan bobot antar simpul untuk menghasilkan matriks jarak antar simpul. Dengan menggunakan vektor dan perulangan, program dapat menampilkan matriks jarak dengan tepat berdasarkan input pengguna.

## 2. UNGUIDED 2

Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!

## SOURCE CODE

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

// 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;
}

// 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 memiliki
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
sebagai child kiri dari " << 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 memiliki
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
sebagai child kanan dari " << 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 memiliki parent)" <<
endl;
        else
            cout << " Parent : " << node->parent->data <<
endl;
        if (node->parent != NULL && node->parent->left !=
node &&
            node->parent->right == node)
            cout << " Sibling : " << node->parent->left->data
<< endl;
        else if (node->parent != NULL && node->parent->right
!= node &&
            node->parent->left == node)
            cout << " Sibling : " << node->parent->right-
>data << endl;
        else
            cout << " Sibling : (tidak memiliki sibling)" <<
endl;
        if (!node->left)
            cout << " Child Kiri : (tidak memiliki child
kiri)" << endl;
        else
            cout << " Child Kiri : " << node->left->data <<
endl;
        if (!node->right)
            cout << " Child Kanan : (tidak memiliki child
kanan)" << endl;
        else
            cout << " Child Kanan : " << node->right->data
<< endl;
    }
}
}

```

```

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = 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 Ukuran Tree : " << size() << endl;
    cout << " Tinggi Tree : " << height() << endl;
    cout << " Rata-rata Node pada Tree : " << size() / height()
<< endl;
}

int main()
{
    init();

    char data;

```

```

cout << "Masukkan data root: ";
cin >> data;
buatNode(data);

char pilihan;
do
{
    cout << "\nMenu:\n";
    cout << "1. Tambah Node Kiri\n";
    cout << "2. Tambah Node Kanan\n";
    cout << "3. Ubah Data Node\n";
    cout << "4. Lihat Data Node\n";
    cout << "5. Cari Node\n";
    cout << "6. Penelusuran PreOrder\n";
    cout << "7. Penelusuran InOrder\n";
    cout << "8. Penelusuran PostOrder\n";
    cout << "9. Hapus SubTree\n";
    cout << "10. Hapus Tree\n";
    cout << "11. Karakteristik Tree\n";
    cout << "0. Keluar\n";
    cout << "Pilihan Anda: ";
    cin >> pilihan;

    switch (pilihan)
    {
    case '1':
        char dataKiri;
        cout << "Masukkan data node kiri: ";
        cin >> dataKiri;
        insertLeft(dataKiri, root);
        break;
    case '2':
        char dataKanan;
        cout << "Masukkan data node kanan: ";

```

```
        cin >> dataKanan;
        insertRight(dataKanan, root);
        break;
    case '3':
        char dataUbah;
        cout << "Masukkan data yang ingin diubah: ";
        cin >> dataUbah;
        update(dataUbah, root);
        break;
    case '4':
        retrieve(root);
        break;
    case '5':
        find(root);
        break;
    case '6':
        cout << "\nPenelusuran PreOrder:\n";
        preOrder(root);
        cout << endl;
        break;
    case '7':
        cout << "\nPenelusuran InOrder:\n";
        inOrder(root);
        cout << endl;
        break;
    case '8':
        cout << "\nPenelusuran PostOrder:\n";
        postOrder(root);
        cout << endl;
        break;
    case '9':
        deleteSub(root);
        break;
    case '10':
```

```
        clear();
        break;
    case '11':
        charateristic();
        break;
    case '0':
        cout << "\nTerima kasih telah menggunakan program
ini!\n";
        break;
    default:
        cout << "\nPilihan tidak valid!\n";
        break;
    }
} while (pilihan != '0');

return 0;
}
```

## SCREENSHOOT PROGRAM

```
Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Ubah Data Node
4. Lihat Data Node
5. Cari Node
6. Penelusuran PreOrder
7. Penelusuran InOrder
8. Penelusuran PostOrder
9. Hapus SubTree
10. Hapus Tree
11. Karakteristik Tree
0. Keluar
Pilihan Anda: 5

Data Node : r
Root : r
Parent : (tidak memiliki parent)
Sibling : (tidak memiliki sibling)
Child Kiri : (tidak memiliki child kiri)
Child Kanan : (tidak memiliki child kanan)
```

## DESKRIPSI PROGRAM

Program berfungsi untuk pengguna untuk menghitung jumlah huruf vokal dalam sebuah kalimat yang dimasukkan oleh pengguna itu sendiri. Dengan menggunakan loop dan kondisi if, program dapat mengidentifikasi huruf vokal dan menghitungnya secara akurat.

Hal ini memperlihatkan bagaimana C++ dapat digunakan untuk membuat program sederhana yang dapat memproses input pengguna dengan efisien.

## **BAB IV**

### **KESIMPULAN**

Setelah melakukan pembelajaran mengenai Queue di cpp dapat di ambil beberapa kesimpulan yaitu :

- a. Pada Graph dan Tree adalah dua struktur data fundamental dalam pemrograman yang digunakan untuk memodelkan hubungan dan hirarki.
- b. Pemahaman dan implementasi keduanya dalam C++ memerlukan pengetahuan tentang representasi, algoritma traversal, dan operasi dasar untuk manipulasi data.

## **DAFTAR PUSTAKA**

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