LAPORAN PRAKTIKUM
MODUL 9
GRAPH DAN TREE

Disusun oleh:

Reza Alvonzo

NIM : 2311102026

Dosen Pengampu:

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

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

BAB II
DASAR TEORI

A. Graph

Graf adalah suatu struktur data yang terdiri dari beberapa titik (node) yang dihubungkan oleh beberapa garis (sisi). Graf dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Graf dapat didefinisikan sebagai G = (V, E), di mana G adalah graf, V adalah himpunan node, dan E adalah himpunan sisi. Graf dapat dibedakan menjadi beberapa tipe, seperti graf tak berarah (undirected graph), graf berarah (directed graph), dan graf berbobot (weighted graph). Beberapa istilah yang umum digunakan dalam graf adalah vertex (node), edge (sisi), adjacent (berdekatan), weight (bobot), path (jalan), dan cycle (siklus). Program graf dapat dibuat menggunakan struktur data seperti adjacency matrix dan linked list untuk merepresentasikan graf. Contoh program graf yang dapat dilakukan adalah pewarnaan graf dengan menggunakan algoritma greedy.

B. Tree

Pohon adalah suatu struktur data yang memiliki beberapa node yang dihubungkan oleh beberapa sisi. Pohon dapat digunakan untuk merepresentasikan struktur data yang memiliki hubungan antara node. Pohon dapat didefinisikan sebagai kumpulan node yang memiliki satu node root dan node lainnya. Pohon juga dapat didefinisikan sebagai kumpulan node yang tidak memiliki loop dan tidak terhubung dengan diri sendiri. Pohon dapat dibedakan menjadi beberapa tipe, seperti pohon biner (binary tree), pohon berarah (directed tree), dan pohon berbobot (weighted tree). Pohon dapat direpresentasikan dalam bentuk array, linked list, atau matrix. Program pohon dapat dibuat menggunakan struktur data seperti binary tree untuk merepresentasikan pohon. Contoh program pohon yang dapat dilakukan adalah traversal pada pohon dengan menggunakan algoritma preorder, inorder, dan postorder.

BAB III

GUIDED

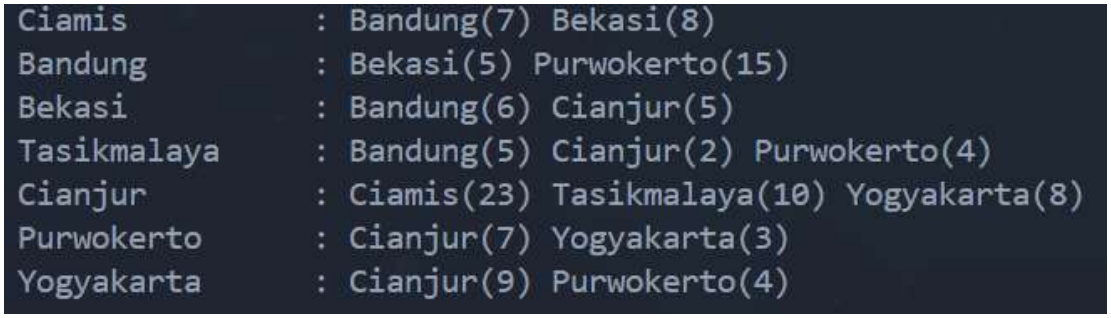
1. GUIDED 1

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



DESKRIPSI PROGRAM

Program ini memiliki fungsi tampilGraph() yang bertugas menampilkan graf. Fungsi ini menggunakan perulangan untuk mengakses setiap simpul dan busur dalam graf, serta menggunakan fungsi setiosflags() dan setw() untuk memformat output agar lebih rapih.

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;
```

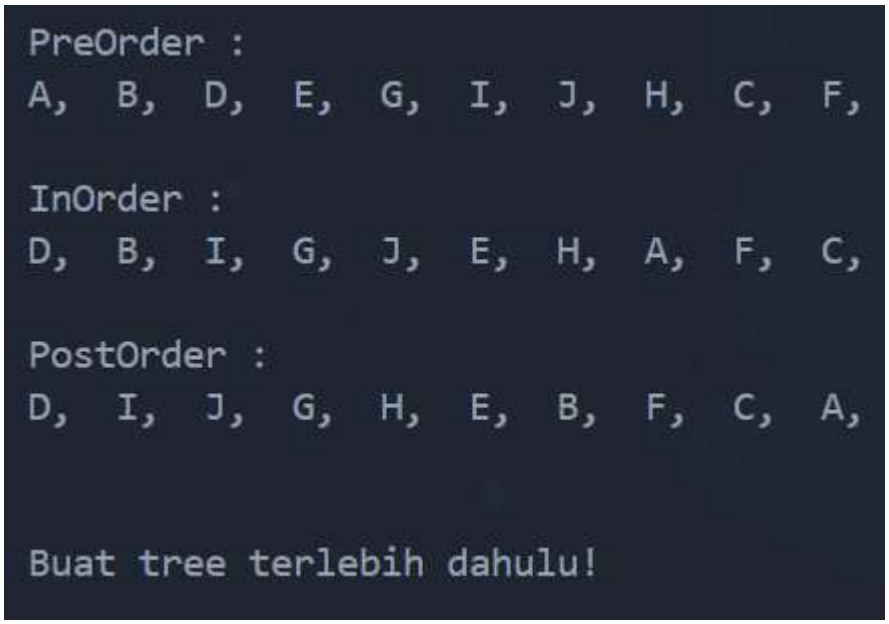
```
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if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

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 :
```



DESKRIPSI PROGRAM

Binary tree didefinisikan menggunakan struktur data yang terdiri dari node yang memiliki atribut data, left child, right child, dan parent. Setiap node dalam binary tree dapat memiliki maksimal dua child, yaitu child kiri dan child kan.

UNGUIDED

1. UNGUIDED 1

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

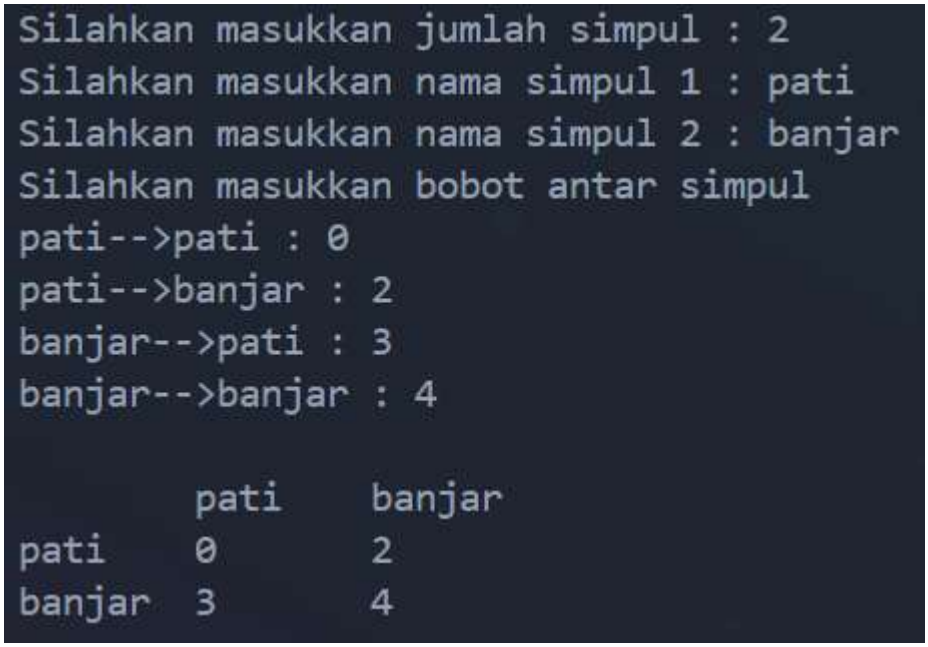
SOURCE CODE

```
#include <iostream>
#include <string>
```

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

int main() {
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> jumlah_simpul;

    vector<string> arjunwerdhokumoro_2311102009(jumlah_simpul);
    vector<vector<int>>                    bobot(jumlah_simpul,
vector<int>(jumlah_simpul));

    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << " :
";
        cin >> arjunwerdhokumoro_2311102009[i];
    }

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

    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << arjunwerdhokumoro_2311102009[i] << "-->" <<
arjunwerdhokumoro_2311102009[j] << " : ";
            cin >> bobot[i][j];
        }
    }
    cout << "\n\t";
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << arjunwerdhokumoro_2311102009[i] << "\t";
    }
    cout << "\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
```

```
        cout << arjunwerdhokumoro_2311102009[i] << "\t";
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }

    return 0;
}
```

SCREENSHOOT PROGRAM



DESKRIPSI PROGRAM

Graf ini didefinisikan menggunakan struktur data yang terdiri dari dua bagian: simpul dan sisi. Simpul adalah node yang memiliki nama dan sisi adalah hubungan antara dua simpul yang memiliki bob.

2. UNGUIDED 2

Buatlah sebuah program yang dapat menghitung banyaknya huruf vocal dalam sebuah kalimat!

SOURCE CODE

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

// Declaring the Tree structure
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root = nullptr;

// Initialize the tree
void init() {
    root = NULL;
}

// Check if the tree is empty
int isEmpty() {
    return (root == NULL) ? 1 : 0;
}

// Create a new node
Pohon* buatNode(char data) {
    Pohon* newNode = new Pohon();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->parent = NULL;
```

```
    // cout << "\nNode " << data << " berhasil dibuat." << endl;
    return newNode;
}

// Insert a node to the left
Pohon* insertLeft(Pohon* parent, Pohon* child) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->left != NULL) {
            cout << "\nNode " << parent->left->data << " sudah
ada child kiri!" << endl;
            return NULL;
        } else {
            child->parent = parent;
            parent->left = child;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kiri " << child->parent->data << endl;
            return child;
        }
    }
}

// Insert a node to the right
Pohon* insertRight(Pohon* parent, Pohon* child) {
    if (root == NULL) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->right != NULL) {
            cout << "\nNode " << parent->right->data << " sudah
ada child kanan!" << endl;
            return NULL;
```

```
        } else {
            child->parent = parent;
            parent->right = child;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kanan " << child->parent->data << endl;
            return child;
        }
    }
}

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

// Retrieve node data
void retrieve(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
```

```
        else {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

// Find node and display its properties
void find(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->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;
        }
    }
}

// Pre-order traversal
void preOrder(Pohon *node) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// In-order traversal
void inOrder(Pohon *node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

// Post-order traversal
```

```
void postOrder(Pohon *node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Delete the entire tree
void deleteTree(Pohon *node) {
    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;
        }
    }
}

// Delete a subtree
void deleteSub(Pohon *node) {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
```

```
        cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Clear the entire tree
void clear() {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Get the size of the tree
int size(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

// Get the height of the tree
int height(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return (heightKiri >= heightKanan) ? heightKiri + 1 :
heightKanan + 1;
    }
```

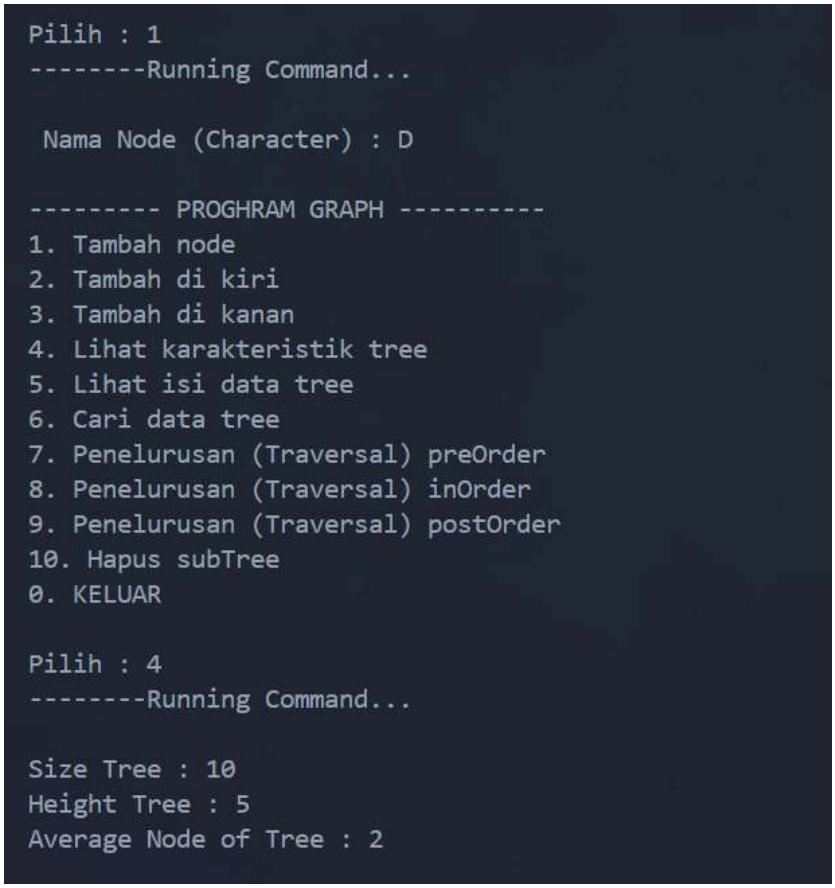
```
}

// Display tree characteristics
void charateristic() {
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
    cout << "Average Node of Tree : " << (size(root) /
(float)height(root)) << endl;
}

int main() {
    root = buatNode('A');
    int menu, part, part2;
    char RezaAlvonzo_2311102026;

    vector<Pohon*> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));

    insertLeft(root, nodes[0]);
    insertRight(root, nodes[1]);
    insertLeft(nodes[0], nodes[2]);
    insertRight(nodes[0], nodes[3]);
    insertLeft(nodes[1], nodes[4]);
    insertLeft(nodes[3], nodes[5]);
    insertRight(nodes[3], nodes[6]);
    insertLeft(nodes[5], nodes[7]);
```

```
    insertRight(nodes[5], nodes[8]);

    do
    {
        cout << "\n--------- PROGHRAM GRAPH ----------\n"
        "1. Tambah node\n"
        "2. Tambah di kiri\n"
        "3. Tambah di kanan\n"
        "4. Lihat karakteristik tree\n"
        "5. Lihat isi data tree\n"
        "6. Cari data tree\n"
        "7. Penelurusan (Traversal) preOrder\n"
        "8. Penelurusan (Traversal) inOrder\n"
        "9. Penelurusan (Traversal) postOrder\n"
        "10. Hapus subTree\n"
        "0. KELUAR\n"
        "\nPilih : ";
        cin >> menu;
        cout << "--------Running Command...\n";
        switch (menu) {
            case 1:
                cout << "\n Nama Node (Character) : ";
                cin >> RezaAlvonzo_2311102026;

nodes.push_back(buatNode(RezaAlvonzo_2311102026));
                break;
            case 2:
                cout << "\nMasukkan nomor untuk node parent : ";
                cin >> part;
                cout << "\nMasukkan nomor untuk node child : ";
                cin >> part2;
                insertLeft(nodes[part], nodes[part2]);
                break;
            case 3:
```

```
                cout << "\nMasukkan nomor untuk node parent : ";
                cin >> part;
                cout << "\nMasukkan nomor untuk node child : ";
                cin >> part2;
                insertRight(nodes[part], nodes[part2]);
                break;
            case 4:
                charateristic();
                break;
            case 5:
                cout << "\nMasukkan nomor node : ";
                cin >> part;
                retrieve(nodes[part]);
                break;
            case 6:
                cout << "\nMasukkan nomor node : ";
                cin >> part;
                find(nodes[part]);
                break;
            case 7:
                cout << "\nPreOrder :" << endl;
                preOrder(root);
                cout << "\n" << endl;
                break;
            case 8:
                cout << "\nInOrder :" << endl;
                inOrder(root);
                cout << "\n" << endl;
                break;
            case 9:
                cout << "\nPostOrder :" << endl;
                postOrder(root);
                cout << "\n" << endl;
                break;
```

```
            case 10:
                cout << "\nMasukkan nomor node : ";
                cin >> part;
                deleteSub(nodes[part]);
                break;
            default:
                break;
        }
    } while (menu != 0);
}
```

SCREENSHOOT PROGRAM



DESKRIPSI PROGRAM

Program ini dapat menambahkan node, menambahkan node sebagai anak kiri atau kanan dari node lain, memperbarui data node, menampilkan data node, mencari node, melakukan traversal pohon, dan menghapus subtree tertentu. Antarmuka ini ditampilkan dalam loop do-while yang berlanjut sampai pengguna memilih untuk keluar (dengan memilih opsi menu 0).

BAB IV
KESIMPULAN

Setelah melakukan pembelajaran mengenai Queue di Bahasa Pemrograman C++ berikut poin utama yang telah dipelajari :

1. Definisi: Graf adalah struktur data non-linear yang terdiri dari node atau vertex yang terhubung oleh edge.
2. Definisi: Pohon adalah graf terhubung yang tidak memiliki siklus.

DAFTAR PUSTAKA

Ripaldi. (2020) Graph C++. diakses pada 10 Juni 2024 dari https://www.academia.edu/43671744/Graph_C_