

LAPORAN PRAKTIKUM
MODUL 9
GRAPH DAN TREE



Disusun oleh:
Aji Tri Prasetyo
NIM : 2311102064

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. Apa itu graph

Graph adalah struktur data yang terdiri dari himpunan simpul (vertices atau nodes) dan himpunan tepi (edges) yang menghubungkan pasangan simpul. Graph digunakan untuk memodelkan hubungan antara objek, seperti jaringan komputer, jejaring sosial, atau peta jalan. Dalam graph, simpul mewakili entitas atau titik, sementara tepi mewakili koneksi atau hubungan antara dua simpul. Graph dapat dikategorikan berdasarkan sifat-sifatnya, seperti arah dan bobot tepi. Terdapat beberapa jenis graph, antara lain graf berarah (directed graph) di mana setiap tepi memiliki arah yang menunjukkan hubungan satu arah dari satu simpul ke simpul lainnya, dan graf tak berarah (undirected graph) di mana tepi menunjukkan hubungan dua arah. Selain itu, ada graf tertimbang (weighted graph) yang setiap tepinya memiliki bobot atau nilai tertentu yang mewakili biaya, jarak, atau kapasitas, dan graf tidak tertimbang (unweighted graph) di mana semua tepi dianggap memiliki bobot yang sama. Untuk merepresentasikan graph, terdapat dua metode yang umum digunakan, yaitu matriks ketetanggaan (adjacency matrix) dan list ketetanggaan (adjacency list). Matriks ketetanggaan adalah matriks dua dimensi di mana elemen $[i][j]$ menunjukkan ada atau tidaknya tepi antara simpul i dan j . Dalam graph berarah, elemen $[i][j]$ menunjukkan arah dari i ke j , dan dalam graph tertimbang, elemen $[i][j]$ menyimpan bobot tepi. List ketetanggaan, di sisi lain, adalah array dari list di mana setiap list berisi simpul-simpul yang terhubung dengan simpul tertentu. List ketetanggaan lebih efisien dalam hal penggunaan ruang untuk graph yang jarang (sparse graph). Algoritma dasar dalam graph meliputi traversal yang mencakup Breadth-First Search (BFS) dan Depth-First Search (DFS), yang digunakan untuk menjelajah seluruh simpul dalam graph. Algoritma lain yang penting adalah algoritma untuk mencari jalur terpendek, seperti algoritma Dijkstra, Bellman-Ford, dan Floyd-

Warshall, serta algoritma untuk menemukan minimum spanning tree, seperti algoritma Kruskal dan Prim.

B. Apa itu Tree

Tree adalah graph tak berarah yang terhubung dan tidak memiliki siklus. Tree adalah bentuk khusus dari graph yang sering digunakan untuk merepresentasikan struktur hirarkis. Dalam tree, setiap simpul memiliki satu simpul induk (parent node) kecuali simpul akar (root node) yang tidak memiliki induk, dan setiap simpul dapat memiliki satu atau lebih simpul anak (child nodes). Terdapat berbagai jenis tree, salah satunya adalah binary tree, di mana setiap simpul memiliki paling banyak dua anak. Binary Search Tree (BST) adalah jenis binary tree yang memiliki properti bahwa setiap simpul kiri lebih kecil dan setiap simpul kanan lebih besar dari simpul induknya. AVL tree adalah self-balancing binary search tree, yang secara otomatis menjaga keseimbangan tinggi untuk memastikan operasi pencarian, penyisipan, dan penghapusan dilakukan dalam waktu logaritmik. Red-Black tree adalah jenis lain dari self-balancing binary search tree yang menjaga keseimbangan tinggi melalui aturan pewarnaan simpul. Segment tree digunakan untuk interval queries, seperti mencari nilai minimum atau maksimum dalam rentang tertentu dari array. Operasi dasar pada tree meliputi penyisipan (insertion) untuk menambahkan simpul baru, penghapusan (deletion) untuk menghapus simpul, penelusuran (traversal) untuk mengunjungi semua simpul dalam tree, dan pencarian (search) untuk mencari nilai tertentu dalam tree, terutama dalam BST. Traversal pada tree dapat dilakukan dalam tiga cara utama: pre-order traversal, in-order traversal, dan post-order traversal, yang masing-masing mengunjungi simpul dalam urutan yang berbeda. Implementasi tree dan graph dalam C++ melibatkan penggunaan struktur data dasar seperti array, list, dan struktur pointer untuk merepresentasikan hubungan antar simpul dan tepi. Pengetahuan tentang dasar teori graph dan tree sangat penting dalam pemrograman karena struktur data ini digunakan secara luas dalam berbagai aplikasi komputer dan algoritma.

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)

```

DESKRIPSI PROGRAM

Program ini menunjukkan cara dasar untuk merepresentasikan dan menampilkan sebuah graph menggunakan matriks ketetanggaan dalam C++. Ini sangat berguna untuk memodelkan dan mengelola data yang berbentuk jaringan atau hubungan antar entitas dalam berbagai aplikasi.

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 :
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!
```

DESKRIPSI PROGRAM

kode ini memberikan gambaran yang komprehensif tentang bagaimana pohon biner dapat dibangun, dimanipulasi, dan dihapus dalam bahasa pemrograman C++. Dengan berbagai fungsi yang diimplementasikan, kode ini memungkinkan pengguna untuk melakukan berbagai operasi dasar dan lanjutan pada struktur pohon biner.

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> ajitriprasetyo_2311102064(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 >> ajitriprasetyo_2311102064[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 << ajitriprasetyo_2311102064[i] << "-->" <<
ajitriprasetyo_2311102064[j] << " : ";
```

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

    }

}

cout << "\n\t";
for (int i = 0; i < jumlah_simpul; ++i) {
    cout << ajitriprasetyo_2311102064[i] << "\t";
}

cout << "\n";

for (int i = 0; i < jumlah_simpul; ++i) {
    cout << ajitriprasetyo_2311102064[i] << "\t";
    for (int j = 0; j < jumlah_simpul; ++j) {
        cout << bobot[i][j] << "\t";
    }
    cout << "\n";
}

return 0;
}
```

SCREENSHOOT PROGRAM

```
Silahkan masukkan jumlah simpul : 2
Silahkan masukkan nama simpul 1 : SOLO
Silahkan masukkan nama simpul 2 : JOGJA
Silahkan masukkan bobot antar simpul
SOLO-->SOLO : 0
SOLO-->JOGJA : 3
JOGJA-->SOLO : 2
JOGJA-->JOGJA : 0

      SOLO    JOGJA
SOLO   0      3
JOGJA  2      0
```

DESKRIPSI PROGRAM

Fungsi `binarySearch` Fungsi ini melakukan pencarian biner (binary search) pada string yang sudah diurutkan. Parameter `const string& sentence` merujuk pada kalimat yang akan dicari. Parameter `char target` adalah karakter yang akan dicari dalam kalimat. Menggunakan dua indeks, `left` dan `right`, untuk menandai batas pencarian.

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 Ajitriprasetyo_2311102064;

    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 >> Ajitriprasetyo_2311102064;

nodes.push_back(buatNode(Ajitriprasetyo_2311102064));
        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

```
----- PROGHRAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 4
-----Running Command...

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

DESKRIPSI PROGRAM

Pohon adalah struktur yang merepresentasikan sebuah simpul dalam pohon biner. Setiap simpul memiliki data (karakter), pointer ke anak kiri (left), anak kanan (right), dan parent (parent). root adalah pointer global yang menunjuk ke akar pohon.

BAB IV

KESIMPULAN

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

1. **Graf Berarah (Directed Graph):** Setiap tepi memiliki arah, menunjukkan dari simpul mana ke simpul mana.
2. **Traversal:** BFS (Breadth-First Search) dan DFS (Depth-First Search) digunakan untuk menjelajah graph.
3. **Binary Tree:** Setiap simpul memiliki paling banyak dua anak.

DAFTAR PUSTAKA

Ramdan, Nurul. (10 November, 2020) Data Structure : Mengenal Graph & Tree. diakses pada 09 Juni 2024 dari <https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/>