LAPORAN PRAKTIKUM

MODUL IX
GRAPH DAN TREE

Disusun oleh:
Nadhif Atha Zaki
NIM: 2311102007

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

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

BAB I
TUJUAN PRAKTIKUM

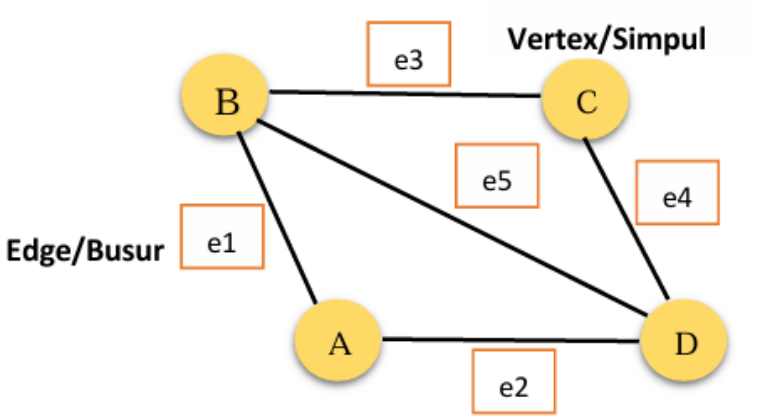
a. Memahami mampu memahami graph dan tree
b. Mengimplementasikan graph dan tree pemrograman

BAB II

DASAR TEORI

1. 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 node sebagai titik atau egde. Dapat digambarkan:



Gambar 1 Contoh Graph

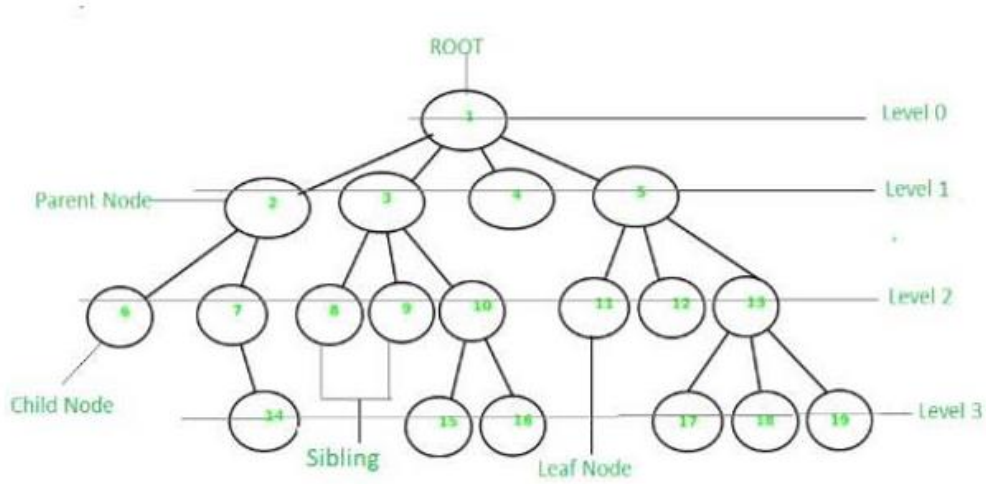
   Jenis – Jenis Graph

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

2. Tree atau Pohon

   Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk

menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



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.

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

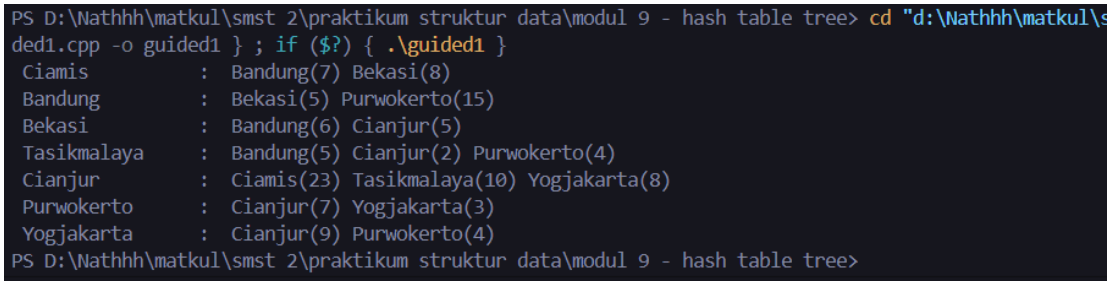
1. Guided 1

Source code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                    "Bandung",
                    "Bekasi",
                    "Tasikmalaya",
                    "Cianjur",
                    "Purwokerto",
                    "Yogjakarta"};
int busur[7][7] =
    {
        {0, 7, 8, 0, 0, 0, 0},
        {0, 0, 5, 0, 0, 15, 0},
        {0, 6, 0, 0, 5, 0, 0},
        {0, 5, 0, 0, 2, 4, 0},
        {23, 0, 0, 10, 0, 0, 8},
        {0, 0, 0, 0, 7, 0, 3},
        {0, 0, 0, 0, 9, 4, 0}};
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
             << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "("
                     << busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}
int main()
```

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

Screenshoot program:



Deskripsi program:

Program di atas merupakan program dalam bahasa C++ yang mendefinisikan sebuah graf dan menampilkan representasi graf tersebut. Graf tersebut terdiri dari tujuh simpul yang diwakili oleh nama kota-kota, dan terdapat busur (atau tepi) yang menghubungkan simpul-simpul tersebut dengan bobot tertentu.

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()
{
    return root == NULL;
}

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

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
```

```
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\nNode " << 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 << "\nNode " << data << " berhasil ditambahkan
ke child kiri " << baru->parent->data << endl;
            return baru;
        }
    }
}

// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\nNode " << 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 << "\nNode " << data << " berhasil ditambahkan
ke child kanan " << baru->parent->data << endl;
            return baru;
        }
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
```

```
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData node: " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

// Penelusuran (Traversal)

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

// inOrder
void inOrder(Pohon *node = root)
{
```

```
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node = root)
{
    if (isEmpty())
        cout << "\nBuat 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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
```

```
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

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

// Hapus Tree
void clear()
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
```

```
}

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

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (isEmpty())
    {
        cout << "\nBuat 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 characteristics()
{
    cout << "\nSize 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 << "\nPreOrder:" << endl;
    preOrder(root);
    cout << "\n"
         << endl;

    cout << "InOrder:" << endl;
    inOrder(root);
    cout << "\n"
         << endl;

    cout << "PostOrder:" << endl;
    postOrder(root);
    cout << "\n"
         << endl;

    characteristics();

    deleteSub(nodeE);

    cout << "\nPreOrder:" << endl;
    preOrder();
    cout << "\n"
         << endl;

    characteristics();

    return 0;
}
```

Screenshoot program:

Deskripsi program:

Program di atas mengimplementasikan struktur data pohon biner dalam C++, termasuk operasi dasar seperti inisialisasi, pengecekan kosong, penambahan node (sebagai anak kiri atau kanan), pengubahan data node, penelusuran pohon (pre-order, in-order, post-order), serta penghapusan node atau subtree. Program ini juga menyediakan fungsi untuk menghitung ukuran dan tinggi pohon, serta menampilkan karakteristik pohon. Pada fungsi utama, program membuat pohon dengan beberapa node dan melakukan berbagai operasi pada pohon tersebut, seperti menambahkan, mengubah, menampilkan data, dan menghapus

LATIHAN KELAS - 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 masukkan 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

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
using namespace std;
// NADHIF ATHA ZAKI // 2311102007
int main()
{
    int jmlhssimpul153;
    cout << "IPUT JUMLAH SIMPUL: ";
    cin >> jmlhssimpul153;
    vector<string> simpul(jmlhssimpul153);    // UNTUK MENYIMPAN NAMA NAMA SIMPUL
    vector<vector<int>> busur(jmlhssimpul153, // MATRIX 2D UNTUK MENYIMPAN BOBOT BUSUR ANTAR SIMPUL
                              vector<int>(jmlhssimpul153, 0));

    cout << "INPUT NAMA SIMPUL " << endl;
```

```
    // LOOP FOR UNTUK MENGITERASI PASANGAN SIMPUL
    for (int i = 0; i < jmlhssimpul153; i++)
    {
        cout << "SIMPUL KE-" << (i + 1) << ": ";
        cin >> simpul[i];
    }
    cout << "INPUT BOBOT SIMPUL" << endl;

    for (int i = 0; i < jmlhssimpul153; i++)
    {
        for (int j = 0; j < jmlhssimpul153; j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }
    cout << endl;
    cout << setw(7) << " ";
    for (int i = 0; i < jmlhssimpul153; i++)
    {
        cout << setw(8) << simpul[i];
    }
    cout << endl;
    for (int i = 0; i < jmlhssimpul153; i++)
    {
        cout << setw(7) << simpul[i];
        for (int j = 0; j < jmlhssimpul153; j++)
        {
            cout << setw(8) << busur[i][j];
        }
        cout << endl;
    }
}
```

Screenshoot program

```
IPUT JUMLAH SIMPUL: 2
INPUT NAMA SIMPUL
SIMPUL KE-1: purwokerto
SIMPUL KE-2: cilacap
INPUT BOBOT SIMPUL
purwokerto --> purwokerto = 1
purwokerto --> cilacap = 2
cilacap --> purwokerto = 3
cilacap --> cilacap = 4

       purwokerto cilacap
purwokerto       1       2
cilacap       3       4
PS D:\Nathhh\matkul\smst 2\praktikum struktur data\modul 9 - hash table tree>
```

Deskripsi dan Penjelasan Program:

Program di atas meminta pengguna untuk memasukkan jumlah simpul dalam sebuah graf, kemudian memasukkan nama-nama simpul tersebut serta bobot busur yang menghubungkan setiap pasangan simpul. Nama-nama simpul disimpan dalam vektor simpul, sedangkan bobot busur disimpan dalam matriks 2D busur. Setelah semua data dimasukkan, program menampilkan matriks bobot busur dalam format tabel, dengan baris dan kolom yang diberi label sesuai dengan nama simpul. Program menggunakan manipulasi lebar (setw) untuk merapikan tampilan tabel di konsol, sehingga memudahkan pengguna untuk melihat hubungan antar simpul beserta bobotnya.

2. .

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

Souce Code:

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

```
// STRUCT POHON
struct tree153
{
    char char153;
    tree153 *left, *right, *parent;
};

tree153 *root, *baru;

// MENGINISIALISASI POHON
void init()
{
    root = NULL;
}

// MENGECEK POHON APAKAH KOSONG
bool isEmpty()
{
    return root == NULL;
}

// MEMBUAT NODE BARU
void buatNode(char char153)
{
    if (isEmpty())
    {
        root = new tree153();
        root->char153 = char153;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << char153 << " berhasil dibuat
sebagai root." << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

// MENAMBAH NODE KIRI
```

```
tree153 *insertLeft(char char153, tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->char153 << " sudah ada
child kiri !" << endl;
            return NULL;
        }
        else
        {
            tree153 *baru = new tree153();
            baru->char153 = char153;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << char153 << " berhasil
ditambahkan ke child kiri " << baru->parent->char153 << endl;
// Pesan sukses
            return baru;
        }
    }
}

// MENAMBAH NODE KANAN
tree153 *insertRight(char char153, tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
```

```
        if (node->right != NULL)
        {
            cout << "\n Node " << node->char153 << " sudah ada
child kanan !" << endl;
            return NULL;
        }
        else
        {
            tree153 *baru = new tree153();
            baru->char153 = char153;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << char153 << " berhasil
ditambahkan ke child kanan " << baru->parent->char153 << endl;
            return baru;
        }
    }
}

// MENGUBAH DATA NODE
void update(char char153, tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\n Node yang ingin diganti tidak ada!!" <<
endl;
        }
        else
        {
            char temp = node->char153;
            node->char153 = char153;
            cout << "\n Node " << temp << " berhasil diubah
menjadi " << char153 << endl;
```

```
        }
    }
}

// MENAMPILKAN DATA NODE
void retrieve(tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\n Data node : " << node->char153 << endl;
        }
    }
}

// MECARI DAN MENAMILKAN DATA NODE
void find(tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\n Data Node : " << node->char153 << endl;
            cout << " Root : " << root->char153 << endl;
```

```
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" <<
endl;
            else
                cout << " Parent : " << node->parent->char153
<< endl;
            if (node->parent != NULL && node->parent->left !=
node && node->parent->right == node)
                cout << " Sibling : " << node->parent->left-
>char153 << endl;
            else if (node->parent != NULL && node->parent-
>right != node && node->parent->left == node)
                cout << " Sibling : " << node->parent->right-
>char153 << 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->char153
<< endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child
kanan)" << endl;
            else
                cout << " Child Kanan : " << node->right-
>char153 << endl;
        }
    }
}

// TRAVERSAL PREORDER
void preOrder(tree153 *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
```

```
    {
        if (node != NULL)
        {
            cout << " " << node->char153 << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// TRAVERSAL INORDER
void inOrder(tree153 *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->char153 << ", ";
            inOrder(node->right);
        }
    }
}

// TRAVERSAL POSTORDER
void postOrder(tree153 *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
```

```
            cout << " " << node->char153 << ", ";
        }
    }
}

// MENGHAPUS POHON
void deleteTree(tree153 *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

// MEGHAPUS SUBPOHON
void deleteSub(tree153 *node)
{
    if (isEmpty())
    {
```

```
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->char153 << "
berhasil dihapus." << endl;
    }
}

// MENGHAPUS SEMUA NODE POHON
void clear()
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    }
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// MENGHITUNG UKURAN POHON
int size(tree153 *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
```

```
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// MEGHITUNG TINGGI POHON
int height(tree153 *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// MENAMPILKAN KARAKTERISTIK POHON
void characteristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
```

```
    cout << " Average Node of Tree : " << size() / height() <<
endl;
}

// MENAMPILKAN CHILDREN DARI NODE
void displayChildren(tree153 *node)
{
    if (node)
    {
        cout << "\nChildren of Node " << node->char153 << ": ";
        if (node->left)
            cout << "Left: " << node->left->char153 << " ";
        else
            cout << "Left: NULL ";
        if (node->right)
            cout << "Right: " << node->right->char153 << " ";
        else
            cout << "Right: NULL ";
        cout << endl;
    }
    else
    {
        cout << "\nNode tidak ditemukan!" << endl;
    }
}

// MENAMPILKAN KETURUNAN DARI NODE
void displayDescendants(tree153 *node)
{
    if (node)
    {
        cout << "\nDescendants of Node " << node->char153 << ":
";
        preOrder(node->left);
        preOrder(node->right);
        cout << endl;
    }
    else
    {
        cout << "\nNode tidak ditemukan!" << endl;
    }
```

```
}

// MENEMUKAN NODE BERDASAR DATA
tree153 *findNode(char char153, tree153 *node = root)
{
    if (!node)
        return NULL;
    if (node->char153 == char153)
        return node;
    tree153 *left = findNode(char153, node->left);
    if (left)
        return left;
    return findNode(char153, node->right);
}

int main()
{
    init();
    int pilih153;
    char char153, parentData;
    tree153 *parentNode = NULL;

    do
    {
        // MENU
        cout << "\n_+_+_+_+_+_+_+_+_+_+_+__+_\n";
        cout << "1. BUAT NODE ROOT\n";
        cout << "2. INSERT NODE KIRI\n";
        cout << "3. INSERT NODE KANAN\n";
        cout << "4. UPDATE NODE\n";
        cout << "5. RETRIEVE NODE\n";
        cout << "6. CARI NODE\n";
        cout << "7. DISPLAY CHILDREN\n";
        cout << "8. DISPLAY DESCENDANT\n";
        cout << "9. TRAVERSAL PRE-ORDER\n";
        cout << "10. TRAVERSAL IN-ORDER\n";
        cout << "11. TRAVERSAL POST-ORDER\n";
        cout << "12. DELETE TREE\n";
        cout << "13. KARAKTERISTIKK TREE\n";
        cout << "0. EXIT\n";
        cout << "PILIH CUUY : ";
```

```
        cin >> pilih153;

        // PILIHAN DARI MENU
        switch (pilih153)
        {
        case 1:
            cout << "Masukkan data untuk root: ";
            cin >> char153;
            buatNode(char153);
            break;
        case 2:
            cout << "Masukkan data untuk node kiri: ";
            cin >> char153;
            cout << "Masukkan data parent: ";
            cin >> parentData;
            parentNode = findNode(parentData);
            insertLeft(char153, parentNode);
            break;
        case 3:
            cout << "Masukkan data untuk node kanan: ";
            cin >> char153;
            cout << "Masukkan data parent: ";
            cin >> parentData;
            parentNode = findNode(parentData);
            insertRight(char153, parentNode);
            break;
        case 4:
            cout << "Masukkan data baru: ";
            cin >> char153;
            cout << "Masukkan data node yang ingin diupdate: ";
            cin >> parentData;
            parentNode = findNode(parentData);
            update(char153, parentNode);
            break;
        case 5:
            cout << "Masukkan data node yang ingin di-retrieve:
";
            cin >> char153;
            parentNode = findNode(char153);
            retrieve(parentNode);
            break;
```

```
        case 6:
            cout << "Masukkan data node yang ingin di-find: ";
            cin >> char153;
            parentNode = findNode(char153);
            find(parentNode);
            break;
        case 7:
            cout << "Masukkan data node untuk menampilkan
children: ";
            cin >> char153;
            parentNode = findNode(char153);
            displayChildren(parentNode);
            break;
        case 8:
            cout << "Masukkan data node untuk menampilkan
descendants: ";
            cin >> char153;
            parentNode = findNode(char153);
            displayDescendants(parentNode);
            break;
        case 9:
            cout << "\nPreOrder :\n ";
            preOrder(root);
            cout << endl;
            break;
        case 10:
            cout << "\nInOrder :\n ";
            inOrder(root);
            cout << endl;
            break;
        case 11:
            cout << "\nPostOrder :\n ";
            postOrder(root);
            cout << endl;
            break;
        case 12:
            clear();
            break;
        case 13:
            characteristic();
            break;
```

```
        case 0:
            cout << "Terima kasih!\n";
            return 0;
            break;
        default:
            cout << "Pilihan tidak valid!\n";
        }
    } while (pilih153 != 0);
}
```

Screenshot Program:

Deskripsi dan Penjelasan Program:

Program di atas merupakan implementasi dari operasi dasar pada pohon biner, mencakup pembuatan, penambahan, penghapusan, serta

traversal node dalam pohon. Program ini menginisialisasi pohon dengan root yang kosong, memungkinkan penambahan node baru di posisi kiri atau kanan dari node yang sudah ada, serta memperbarui data node yang ditentukan. Pengguna dapat mencari node tertentu dan menampilkan data node tersebut beserta parent, sibling, dan children-nya. Selain itu, program menyediakan fungsi traversal pre-order, in-order, dan post-order untuk mengunjungi node dalam urutan tertentu. Program juga dapat menampilkan ukuran dan tinggi pohon, serta menghapus seluruh pohon atau subpohon yang ditentukan. Menu interaktif memungkinkan pengguna untuk memilih operasi yang diinginkan melalui input dari konsol.

BAB IV

KESIMPULAN

Graph dan tree adalah dua struktur data fundamental dalam ilmu komputer yang digunakan untuk memodelkan hubungan antar objek. Graph terdiri dari simpul (nodes atau vertices) yang terhubung oleh busur (edges). Graph dapat dibagi menjadi beberapa jenis, seperti graph berarah (directed graph) dan graph tak berarah (undirected graph), serta graph berimbang (weighted graph) dan graph tak berimbang (unweighted graph). Graph sering digunakan dalam berbagai aplikasi seperti jaringan komputer, sistem rekomendasi, dan analisis sosial.

Tree adalah jenis graph khusus yang tidak memiliki siklus dan biasanya digunakan untuk merepresentasikan struktur hierarkis. Pohon biner, sebagai contoh umum, memiliki node yang masing-masing bisa memiliki paling banyak dua anak (left child dan right child). Pohon memiliki satu node utama yang disebut root, dan setiap node di tree hanya memiliki satu parent. Tree digunakan dalam banyak aplikasi, termasuk struktur data seperti binary search tree (BST) untuk pencarian dan pengurutan, heap dalam algoritma pengurutan, serta pohon ekspresi dalam kompilator.

Keduanya, graph dan tree, menyediakan cara yang efisien untuk menyimpan dan mengelola data yang memiliki hubungan kompleks, memungkinkan operasi seperti pencarian, penyortiran, dan navigasi dilakukan dengan lebih efektif. Memahami kedua struktur data ini adalah kunci untuk menguasai banyak aspek dari algoritma dan pemrograman komputer.