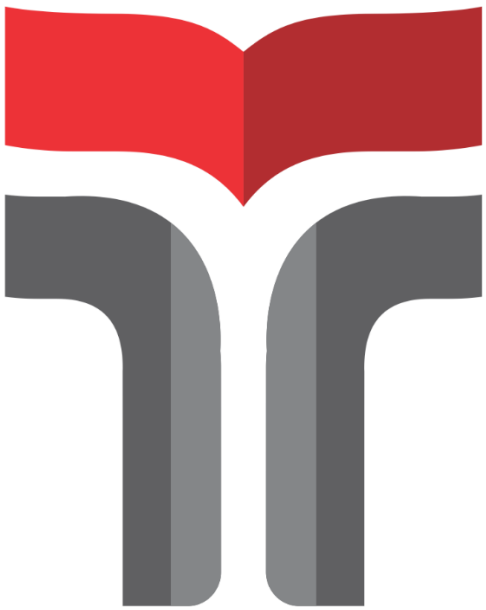
LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA

MODUL IX
GRAPH & TREE



DISUSUN OLEH :

MUHAMAD NASRULLOH

2311102044

DOSEN :

WAHYU ANDI SAPUTRA, S.PD., M.ENG.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

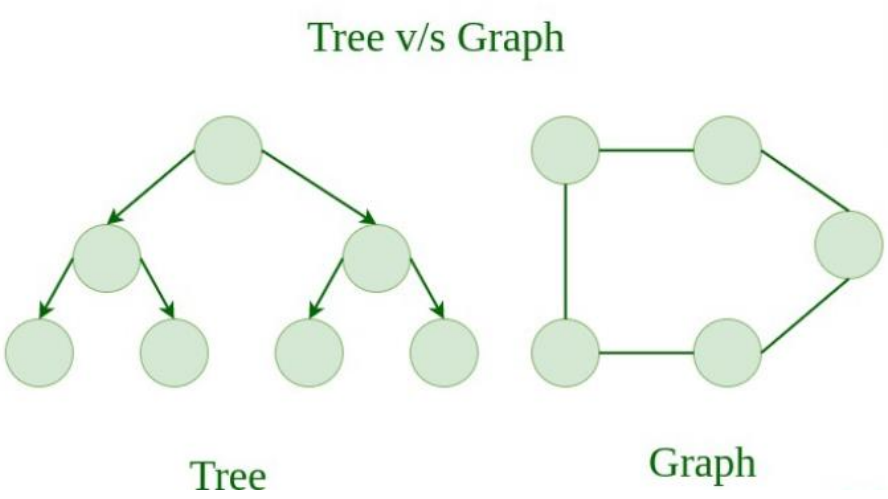
FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. DASAR TEORI

Graph dan tree adalah dua struktur data yang digunakan dalam ilmu komputer untuk mewakili hubungan antarobjek. Meskipun keduanya memiliki kesamaan, namun juga memiliki perbedaan yang membuatnya cocok untuk aplikasi yang berbeda.

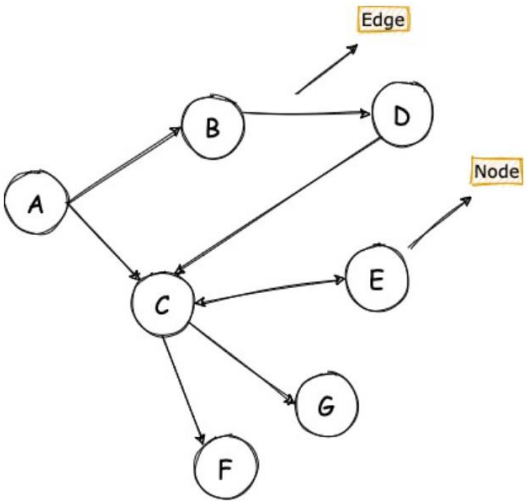


Gb. Ilustrasi perbedaan graph dan tree

1. Graph

Adalah kumpulan node (disebut juga simpul) dengan garis yang menghubungkannya. Node dapat mewakili entitas, seperti orang, tempat atau benda. Sementara garis mewakili hubungan antara entitas tersebut. Graph digunakan untuk memodelkan berbagai sistem dunia nyata, seperti jejaring sosial, jaringan transportasi dan jaringan komputer.

Graph terdiri dari beberapa kumpulan titik (node) dan garis (edge).



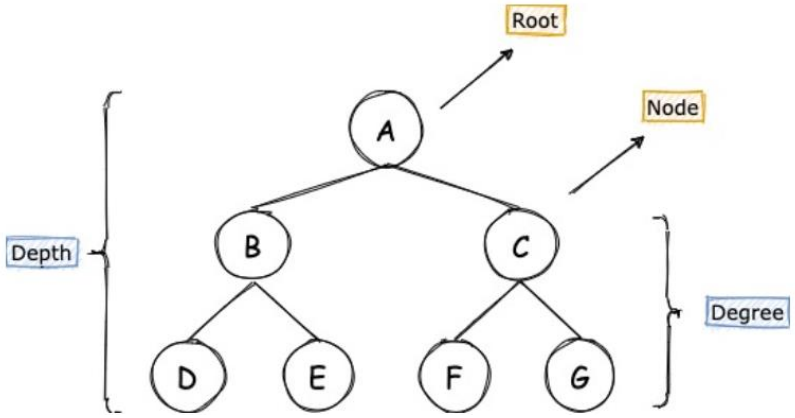
Gb 1.0 Ilustrasi graph

Berdasar orientasi arah sisinya, graph dapat dibedakan menjadi 2:
- Directed graph atau graf berarah adalah graf yang setiap sisinya memiliki orientasi arah.

- Undirected graph atau graf tak berarah adalah graf yang sisinya tidak memiliki orientasi arah.

2. Tree

Adalah struktur data hierarkis yang terdiri dari node yang dihubungkan oleh garis. Setiap node dapat memiliki beberapa node anak, tapi hanya ada 1 node induk. Node atau simpul paling atas dalam tree disebut simpul akar.



Gb 2.0 Ilustrasi tree

Keterangan :
- Node, struktur yang berisi sebuah nilai atau suatu kondisi yang menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri
- Root, sebuah node yang terletak di posisi tertinggi atau urutan pertama dari suatu tree
- Depth, jarak atau ketinggian antara root dan node
- Degree, banyaknya anak atau turunan dari suatu node

B. GUIDED

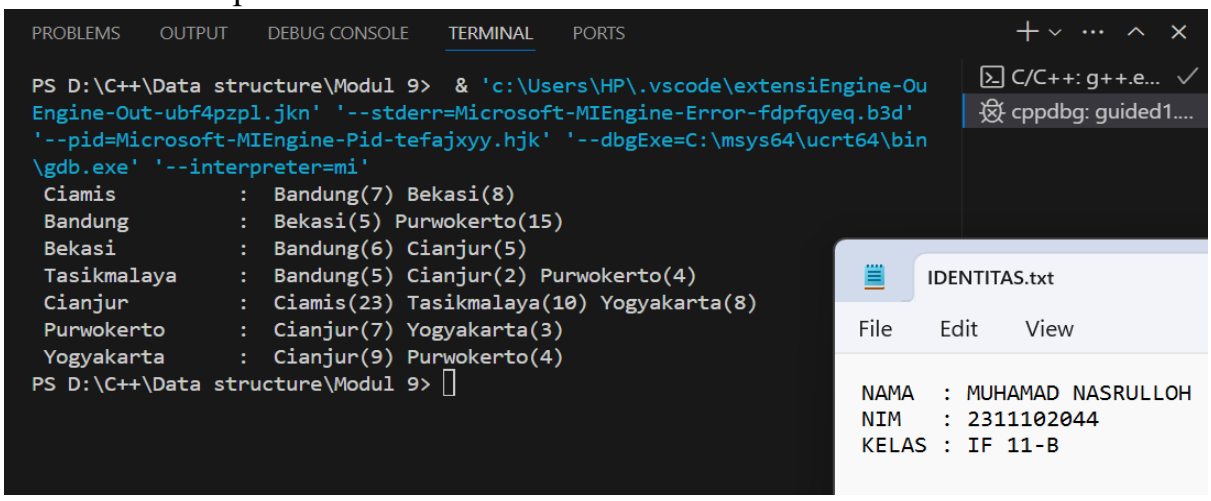
- Guided I

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;
}
```

Screenshot Output



Deskripsi
Program diatas merupakan implementasi graf dalam c++ yang berbentuk matriks ketetanggaan (adjacency matrix) dari beberapa kota di Jawa Barat dan Jawa Tengah. Program ini terdiri dari 2 array utama : simpul (string berisi nama-nama kota) dan busur (matrix 2D yang merepresentasikan bobot atau jarak antara dua kota tertentu). Selain itu juga terdiri dari 2 fungsi : tampilGraph (untuk menampilkan graf ke layar) dan main (untuk memanggil fungsi 'tampilGraph' dan menampilkan graf pada konsole').

- **Guided II**

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; // true
    else
        return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
```

```
        cout << "\n Node " << " 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;
            baru->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 terlebih dahulu!" << endl;
                else
            {
                char temp = node->data;
                node->data = data;
                cout << "\n Node" << temp << " berhasil diubah menjadi " <<
data << endl;
            }
    }
}
// Lihat Isi Data Tree
```

```
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left->data << endl;
            else if (node->parent != NULL && node->parent->right != node &&
                     node->parent->left == node)
                cout << " Sibling : " << node->parent->right->data << endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << " Child Kiri : (tidak punya Child kiri)" << endl;
            else
                cout << " Child Kiri : " << node->left->data << endl;
            if (!node->right)
```

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

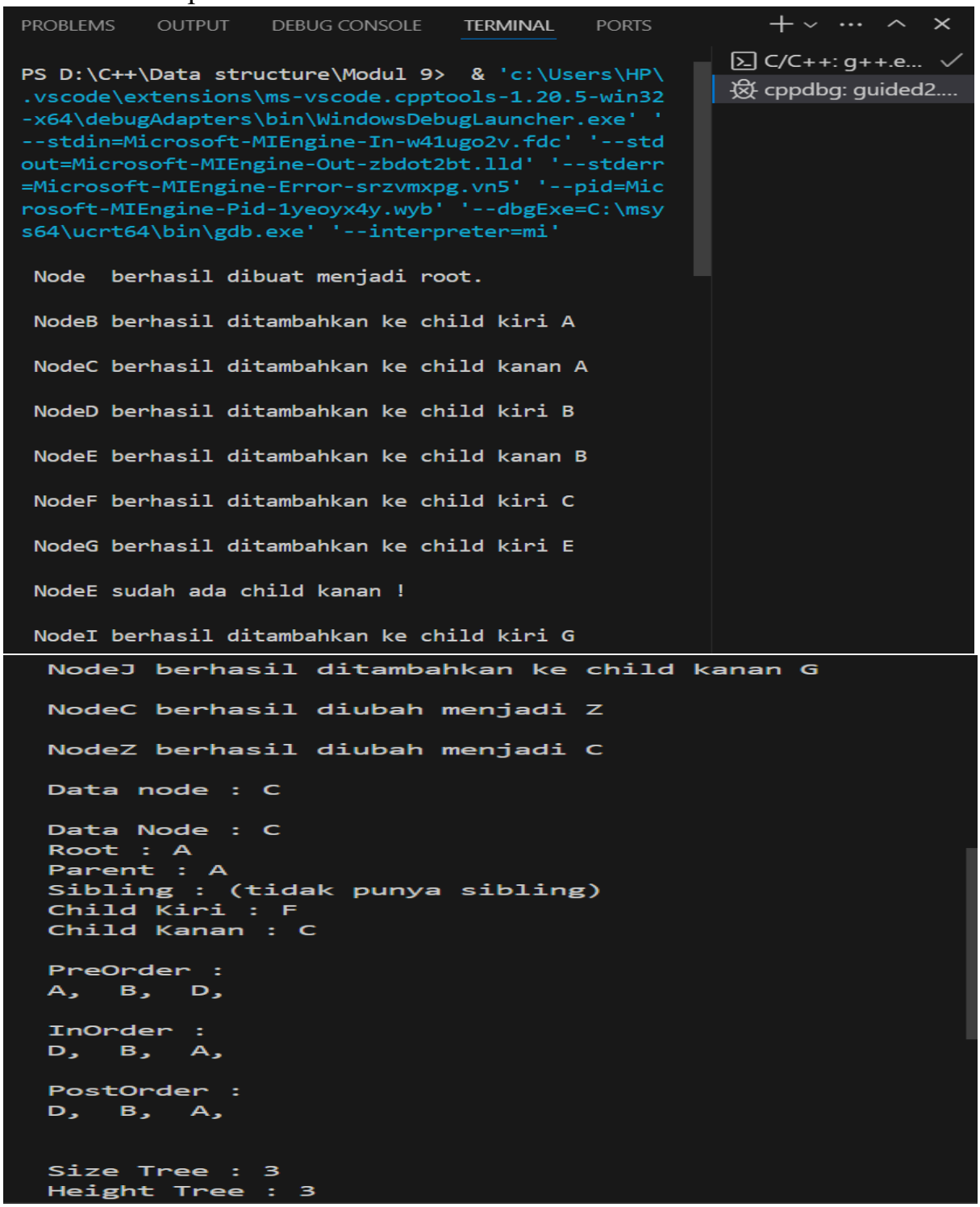
```
            cout << " " << node->data << ", ";
        }
    }
}
// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
              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

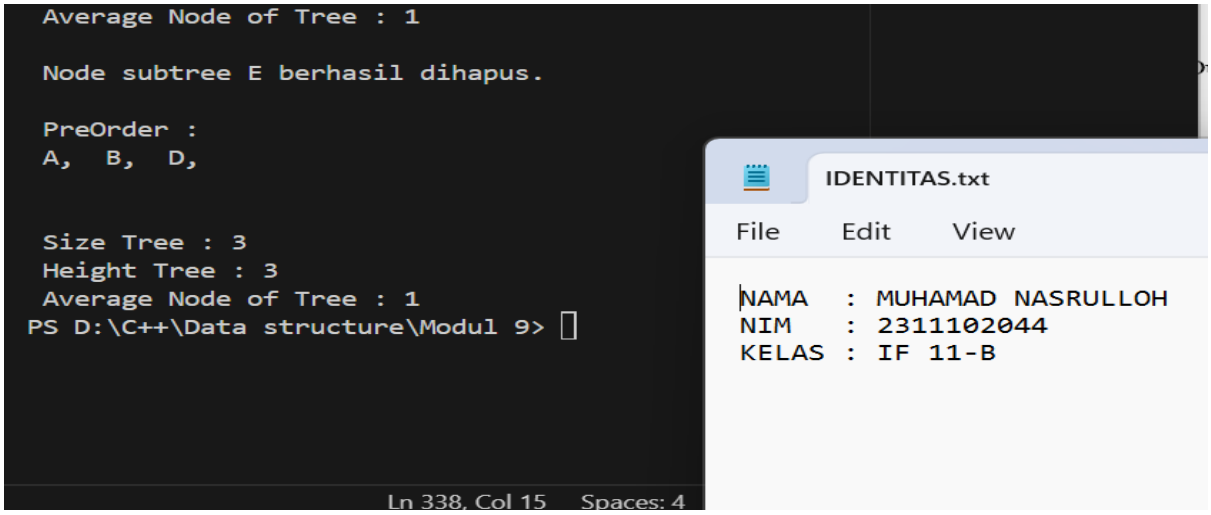
```
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" <<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

```
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}
// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}
int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
         << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
         << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
         << endl;
    charateristic();
    deleteSub(nodeE);
```

```
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
         << endl;
    charateristic();
}
```

Screenshot Output





Deskripsi

Program diatas implementasi pohon biner (binary tree) dalam c++ dengan berbagai operasi seperti penambahan node, penghapusan subtree, dan traversal. Program ini menggunakan struct 'pohon' untuk merepresentasikan setiap node dalam pohon, yang memiliki atribut data, pointer ke anak kiri dan kanan, serta parent. Fungsi yang terdapat dalam program ini seperti : init() dan isEmpty() untuk inisialisasi dan pengecekan, buatNode insertLeft dan insertRight(char data, Pohon *node) untuk menambahkan node, update dan retrieve(Pohon *node) untuk mengubah dan mengambil data dari node, find(Pohon *node) untuk mencari informasi node tertentu, preOrder inOrder dan postOrder(Pohon *node) untuk melakukan traversal, deleteTree deleteSub clear() untuk menghapus seluruh atau sebagian pohon, size height dan characteristic() untuk menghitung dan menampilkan karakteristik pohon.

C. UNGUIDED

- **Unguided I**

Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya. Cantumkan NIM pada salah satu variabel dalam program (contoh int nama_2311102044).

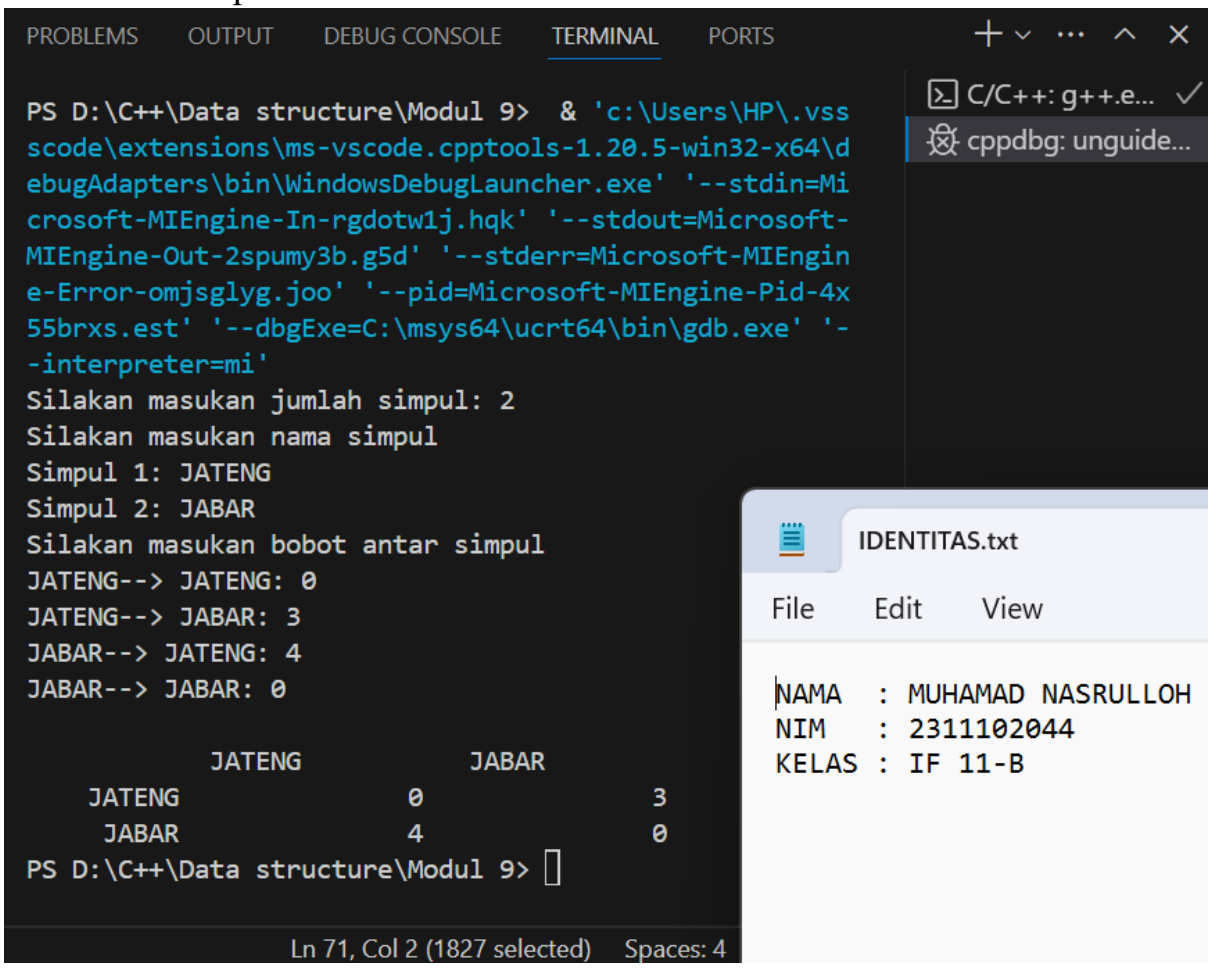
Source Code

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

// NIM: 2311102044
int nim_2311102044 = 2311102044;

// Maksimal jumlah simpul (kota)
```

```cpp
const int MAX = 10;

void displayGraph(int graph[MAX][MAX], string cities[MAX], int n) {
    cout << "\n\t";
    for (int i = 0; i < n; ++i) {
        cout << setw(10) << cities[i] << "\t";
    }
    cout << endl;

    for (int i = 0; i < n; ++i) {
        cout << setw(10) << cities[i] << "\t";
        for (int j = 0; j < n; ++j) {
            if (graph[i][j] == INT_MAX) {
                cout << setw(10) << "INF" << "\t";
            } else {
                cout << setw(10) << graph[i][j] << "\t";
            }
        }
        cout << endl;
    }
}

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    // Array untuk menyimpan nama-nama simpul (kota)
    string cities[MAX];
    cout << "Silakan masukan nama simpul\n";
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> cities[i];
    }

    // Array untuk menyimpan graf dalam bentuk matriks ketetanggaan
    int graph[MAX][MAX];

    // Inisialisasi matriks ketetanggaan dengan nilai tak hingga (INT_MAX)
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) {
                graph[i][j] = 0;
            } else {
                graph[i][j] = INT_MAX;
            }
        }
    }
```

```
    cout << "Silakan masukan bobot antar simpul\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << cities[i] << "--> " << cities[j] << ": ";
            int weight;
            cin >> weight;
            graph[i][j] = weight;
        }
    }

    displayGraph(graph, cities, n);

    return 0;
}
```

Screenshot Output



Deskripsi

Program diatas dibuat dengan array 2D statis untuk merepresentasikan graf dalam bentuk matriks ketetanggaan. User diminta untuk memasukkan jumlah simpul (kota) dan nama-nama simpul tersebut, serta bobot (jarak) antar simpul

termasuk simpul ke dirinya sendiri. Matriks ketetanggaan diinisialisasi dengan nilai tak hingga (INT_MAX) untuk menunjukkan bahwa tidak ada jalur antara dua kota, kecuali jika diatur user. Setelah semua data dimasukkan, program akan menampilkan graf dalam bentuk tabel yang menunjukkan bobot antar simpul. Angka NIM ‘2311102044’ disertakan dalam variabel sebagai bagian dari kode.

- **Unguided II**

Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang di inputkan!

Source Code

```
#include <iostream>
#include <cstddef>
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;
    }
    if (!node) {
        cout << "\nNode parent tidak ditemukan!" << endl;
        return NULL;
    }
    if (node->left != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    } else {
        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 "
<< node->data << endl;
        return baru;
    }
}

// Tambah Kanan
Pohon* insertRight(char data, Pohon* node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    if (!node) {
        cout << "\nNode parent tidak ditemukan!" << endl;
        return NULL;
    }
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kanan!" <<
endl;
        return NULL;
    } else {
```

```
        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 "
<< node->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 ditemukan!" << 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 (!root) {
        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 (!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;
            }
        }
    }
}

// Tampilkan Child dari Node
void showChild(Pohon* node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "Child Kiri: " << (node->left ? node->left->data : ' ')
<< endl;
            cout << "Child Kanan: " << (node->right ? node->right->data : '
') << endl;
        }
    }
}

// Tampilkan Descendants dari Node
```

```
void showDescendants(Pohon* node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            if (node->left) {
                cout << node->left->data << " ";
                showDescendants(node->left);
            }
            if (node->right) {
                cout << node->right->data << " ";
                showDescendants(node->right);
            }
        }
    }
}

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

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

// postOrder
void postOrder(Pohon* node) {
```

```
    if (!root) {
        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 (!root) {
        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 (!root) {
        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 (!root) {
```

```
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    } else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

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

// Cek Height Tree
int height(Pohon* node = root) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!!" << endl;
        return 0;
    } else {
        if (node == NULL) {
            return -1;
        } else {
            int kiri, kanan;
            kiri = height(node->left);
            kanan = height(node->right);
            return max(kiri, kanan) + 1;
        }
    }
}

// Cek rata-rata Node Tree
int average() {
    return size() / height();
}

int main() {
    init();
    int pil;
    do {
        cout << "\nMenu:";
```

```
        cout << "\n1. Buat Node Root";
        cout << "\n2. Tambah Node Kiri";
        cout << "\n3. Tambah Node Kanan";
        cout << "\n4. Ubah Data Node";
        cout << "\n5. Lihat Isi Data Node";
        cout << "\n6. Cari Data Node";
        cout << "\n7. Tampilkan Child dari Node";
        cout << "\n8. Tampilkan Descendants dari Node";
        cout << "\n9. PreOrder Traversal";
        cout << "\n10. InOrder Traversal";
        cout << "\n11. PostOrder Traversal";
        cout << "\n12. Hapus SubTree";
        cout << "\n13. Hapus Tree";
        cout << "\n14. Karakteristik Tree";
        cout << "\n0. Keluar";
        cout << "\nPilih menu: ";
        cin >> pil;

        switch (pil) {
            case 1:
                char data;
                cout << "\nMasukkan data untuk root: ";
                cin >> data;
                buatNode(data);
                break;
            case 2:
                char dataKiri, parentKiri;
                cout << "\nMasukkan data untuk node kiri: ";
                cin >> dataKiri;
                cout << "Masukkan data parent: ";
                cin >> parentKiri;
                insertLeft(dataKiri, root);  // Asumsi parent adalah root.
Perlu disesuaikan sesuai kebutuhan.
                break;
            case 3:
                char dataKanan, parentKanan;
                cout << "\nMasukkan data untuk node kanan: ";
                cin >> dataKanan;
                cout << "Masukkan data parent: ";
                cin >> parentKanan;
                insertRight(dataKanan, root);  // Asumsi parent adalah root.
Perlu disesuaikan sesuai kebutuhan.
                break;
            case 4:
                char dataBaru, nodeUbah;
                cout << "\nMasukkan data baru: ";
                cin >> dataBaru;
                cout << "Masukkan data node yang ingin diubah: ";
```

```
                cin >> nodeUbah;
                update(dataBaru, root);  // Asumsi node adalah root. Perlu
disesuaikan sesuai kebutuhan.
                break;
            case 5:
                char nodeLihat;
                cout << "\nMasukkan data node yang ingin dilihat: ";
                cin >> nodeLihat;
                retrieve(root);  // Asumsi node adalah root. Perlu
disesuaikan sesuai kebutuhan.
                break;
            case 6:
                char nodeCari;
                cout << "\nMasukkan data node yang ingin dicari: ";
                cin >> nodeCari;
                find(root);  // Asumsi node adalah root. Perlu disesuaikan
sesuai kebutuhan.
                break;
            case 7:
                char nodeChild;
                cout << "\nMasukkan data node yang ingin dilihat child-nya:
";
                cin >> nodeChild;
                showChild(root);  // Asumsi node adalah root. Perlu
disesuaikan sesuai kebutuhan.
                break;
            case 8:
                char nodeDescendant;
                cout << "\nMasukkan data node yang ingin dilihat descendant-
nya: ";
                cin >> nodeDescendant;
                showDescendants(root);  // Asumsi node adalah root. Perlu
disesuaikan sesuai kebutuhan.
                break;
            case 9:
                cout << "\nPreOrder Traversal:" << endl;
                preOrder(root);
                break;
            case 10:
                cout << "\nInOrder Traversal:" << endl;
                inOrder(root);
                break;
            case 11:
                cout << "\nPostOrder Traversal:" << endl;
                postOrder(root);
                break;
            case 12:
                char nodeSub;
```

```
                cout << "\nMasukkan data node yang ingin dihapus subtree-
nya: ";
                cin >> nodeSub;
                deleteSub(root);  // Asumsi node adalah root. Perlu
disesuaikan sesuai kebutuhan.
                break;
            case 13:
                clear();
                break;
            case 14:
                cout << "\nSize Tree: " << size() << endl;
                cout << "Height Tree: " << height() << endl;
                cout << "Average Node of Tree: " << average() << endl;
                break;
            case 0:
                cout << "\nKeluar dari program." << endl;
                break;
            default:
                cout << "\nPilihan tidak valid!" << endl;
        }
    } while (pil != 0);

    return 0;
}
```

Screenshot Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\C++\Data structure\Modul 9> & 'c:\Users\HP\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters
\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-3jch0bpt.gg3' '--stdout=Microsoft-MIEngine-Out-yjlsiir1.
vzj' '--stderr=Microsoft-MIEngine-Error-xwq102al.0ya' '--pid=Microsoft-MIEngine-Pid-xeecipja.uui' '--dbgExe=C:\msys64\
ucrt64\bin\gdb.exe' '--interpreter=mi'

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Ubah Data Node
5. Lihat Isi Data Node
6. Cari Data Node
7. Tampilkan Child dari Node
8. Tampilkan Descendants dari Node
9. PreOrder Traversal
10. InOrder Traversal
11. PostOrder Traversal
12. Hapus SubTree
13. Hapus Tree
14. Karakteristik Tree
0. Keluar
```

#1 Membuat root node

```
Pilih menu: 1

Masukkan data untuk root: A

Node A berhasil dibuat menjadi root.
```

NAMA : MUHAMAD NASRULLOH
NIM : 2311102044

#2 Tambah node kanan pada root

```
Pilih menu: 3

Masukkan data untuk node kanan: B
Masukkan data parent: A

Node B berhasil ditambahkan ke child kanan A
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#3 Tambah node kiri pada root

```
Pilih menu: 2

Masukkan data untuk node kiri: C
Masukkan data parent: A

Node C berhasil ditambahkan ke child kiri A
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#4 Menampilkan child dari root

```
Pilih menu: 7

Masukkan data node yang ingin dilihat child-nya: A
Child Kiri: C
Child Kanan: B
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#5 Menampilkan descendants dari root

```
Pilih menu: 8

Masukkan data node yang ingin dilihat descendant-nya: A
C B
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#6 Menampilkan preorder traversal

```
Pilih menu: 9

PreOrder Traversal:
 A,  C,  B,
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#7 Menampilkan karakteristik tree

```
Pilih menu: 14

Size Tree: 3
Height Tree: 1
Average Node of Tree: 3
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#8 Menghapus tree keseluruhan

```
Pilih menu: 13

Pohon berhasil dihapus.
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

#9 Keluar program

```
Pilih menu: 0

Keluar dari program.
PS D:\C++\Data structure\Modul 9>
```

```
NAMA  : MUHAMAD NASRULLOH
NIM   : 2311102044
```

Deskripsi

Program diatas dibuat dengan binary tree dalam c++ yang menyediakan berbagai fungsi untuk membuat, menambah, mengubah, menampilkan dan

menghapus node dalam tree. Struct ‘Pohon’ yang dideklarasikan di awal mewakili node dalam tree dimana setiap nodenya memiliki data, serta pointer ke anak kiri, anak kanan dan parentnya. Dengan sedikit modifikasi dari versi sebelumnya (pada guide) diantaranya adanya menu opsi untuk user bebas menginputkan pilihan dan melakukan operasi yang diinginkan, termasuk menampilkan node child dan descendant.

D. KESIMPULAN

Aplikasi graph dan tree dalam c++ memainkan peran penting dalam pemecahan masalah yang melibatkan struktur data non linier. Dengan menggunakan graph, kita dapat memodelkan berbagai sistem kompleks seperti jaringan komputer, transportasi, dan hubungan antar objek. Disisi lain, pohon (tree) adalah struktur hierarkis yang sering digunakan dalam pengolahan data seperti file sistem, database, dan ekspresi matematika. Khususnya pohon biner digunakan dalam pencarian dan penyortiran data yang cepat melalui algoritma seperti binary search tree (BST).

E. REFERENSI

[1] Ramdan Nurul (2020). Data Structure : Mengenal Graph & Tree

https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/.

[2] Romin Vaghani (2024). Difference Between Graph and Tree

https://www.geeksforgeeks.org/difference-between-graph-and-tree/.

[3] Asisten Praktikum (2024). Modul 9 Graph dan Tree