

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 9
GRAPH DAN TREE**



Disusun Oleh:

Muhamad ihsan

2311102077

Dosen

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMARIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

Modul 8

GRAPH DAN TREE

A. Tujuan

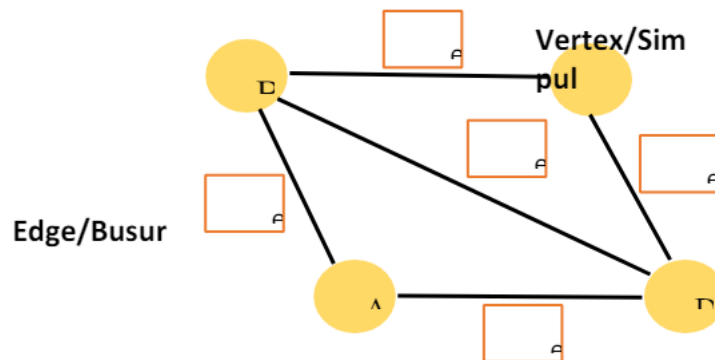
1. Mahasiswa diharapkan mampu memahami graph dan tree.
2. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

B. Dasar Teori

1. Graph

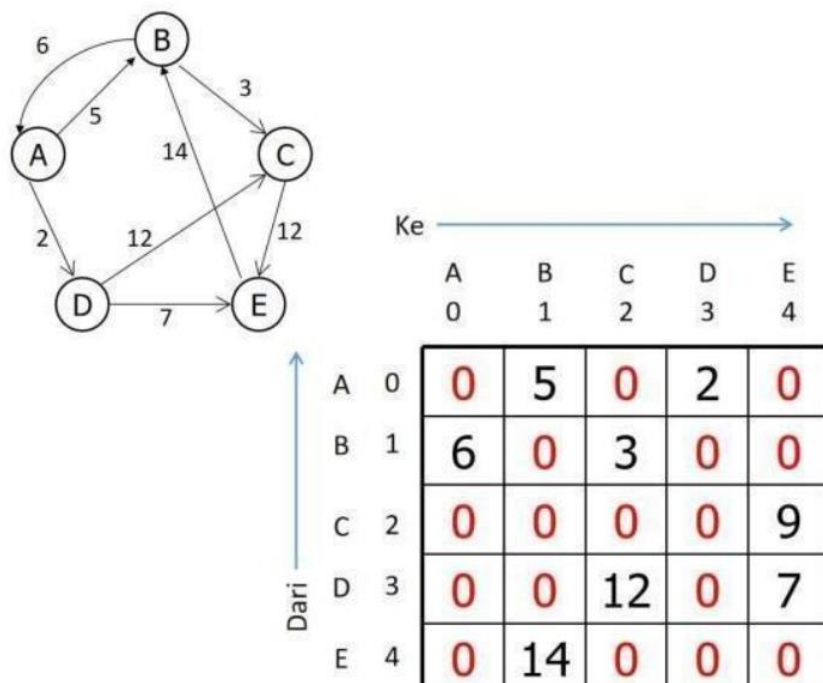
Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai : $G = (V, E)$

Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge. Dapat digambarkan:



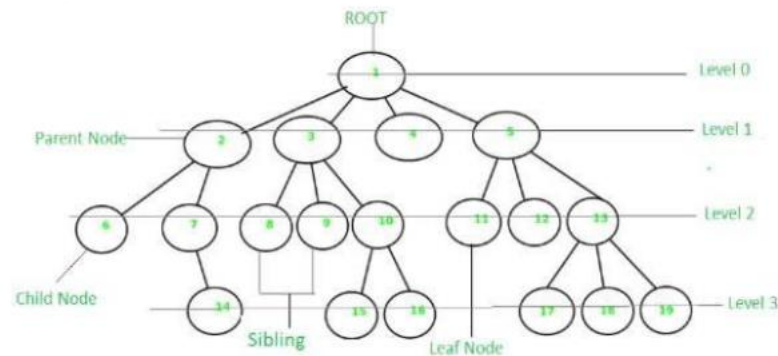
Gambar 1 Contoh Graph.

Representasi Graph dengan Matriks



2. Tree atau Pohon

Dalam ilmu komputer, pohon/tree 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, dimana 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 hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut:



Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Root	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

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.

GUIDED 1

```
#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;
}
```

Output:

```

cpp -o guided1per9 } ; if ($?) { .\guided1per9 }
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)
PS C:\Users\ASUS\OneDrive\Desktop\praktikum struktur data\bui

```

File	Edit	Lihat
NAMA	:	Muhamad ihsan
NIM	:	2311102077
KELAS	:	IF-11-B

Ln 4, Col 1 | 1 dari 54 karakter | 100%

Deskripsi:

Program di atas adalah mengimplementasikan graf berbobot (weighted graph) dengan beberapa simpul dan busur (edge) yang menghubungkannya. Kode ini mendefinisikan 7 simpul dengan nama seperti “Ciamis”, “Bandung”, “Bekasi”, dan lainnya. Matriks busur merepresentasikan hubungan antara simpul-simpul, dan fungsi tampilGraph() menampilkan informasi tentang hubungan tersebut.

GUIDED 2

```

#include <iostream>
using namespace std;

// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon {
    char data;
    Pohon *left, *right, *parent; //pointer
};

//pointer global
Pohon *root;

// Inisialisasi
void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {

```

```

        root = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

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

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

            return baru;
        }
    }
}

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;
        }
    }
}

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;
        }
    }
}

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) {
    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) {
    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) {

    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) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    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) {
    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) {
    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 characteristic() {
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)

```

```

        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

int main() {
    init();
    buatNode('A');

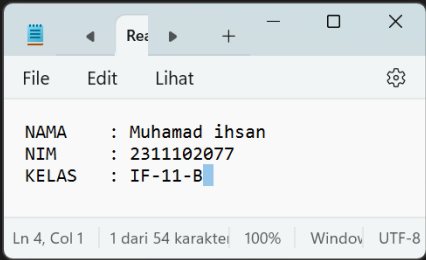
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI,
    *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    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;
    characteristic();
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    characteristic();
}

```

Output:

```
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,
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2
Node subtree E berhasil dihapus.
PreOrder :
A, B, D, E, C, F,
Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
```



Deskripsi:

Program di atas membuat sebuah binary tree dengan node A sebagai root, kemudian menambahkan node-node lainnya sebagai child kiri dan kanan. Kemudian, program ini melakukan update pada node C, retrieve data node C, dan menampilkan informasi tentang node C. Selanjutnya, program ini melakukan traversal pada tree menggunakan preOrder, inOrder, dan postOrder. Kemudian, program ini menampilkan karakteristik tree seperti size, height, dan average node. Terakhir, program ini menghapus subtree E dan menampilkan kembali karakteristik tree.

C. 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 : UNGUIDED 1

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

int main() {
    int jumlahSimpul2311102077;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul2311102077;

    string simpul[jumlahSimpul2311102077];
    int busur[jumlahSimpul2311102077][jumlahSimpul2311102077];

    for (int i = 0; i < jumlahSimpul2311102077; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jumlahSimpul2311102077; i++) {
        for (int j = 0; j < jumlahSimpul2311102077; j++) {
            cout << "Silakan masukkan bobot antara simpul " << simpul[i] << " dan " <<
simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }

    cout << "\nGraf yang dihasilkan:\n";
    cout << setw(15) << " ";
    for (int i = 0; i < jumlahSimpul2311102077; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul2311102077; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jumlahSimpul2311102077; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

SCREENSHOT OUTPUT

```
Silakan masukkan jumlah simpul: 2
Simpul 1: depaul
Simpul 2: rodri
Silakan masukkan bobot antara simpul depaul dan depaul: 2
Silakan masukkan bobot antara simpul depaul dan rodri: 4
Silakan masukkan bobot antara simpul rodri dan depaul: 2
Silakan masukkan bobot antara simpul rodri dan rodri: 0

Graf yang dihasilkan:
      depaul      rodri
depaul      2      4
rodri       2      0
```

	File	Edit	Lihat
NAMA	:	Muhamad	ihsan
NIM	:	2311102077	
KELAS	:	IF-11-B	

Ln 3, Col 16 | 54 karakter | 100%

PS C:\Users\ASUS\OneDrive\Desktop\praktikum struktur data\build\modul9.cpp> |

DESKRIPSI:

Program di atas adalah sebuah program yang digunakan untuk membuat sebuah graf yang diwakili oleh sebuah matriks bobot. Graf ini terdiri dari beberapa simpul yang dihubungkan oleh busur dengan bobot tertentu. Program meminta user untuk memasukkan jumlah simpul, nama simpul, dan bobot antara simpul, kemudian menampilkan graf yang dihasilkan dalam bentuk matriks.

2. 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 diinput kan!

SUORCE CODE : UNGUIDED 2

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

// Deklarasi Pohon
struct Pohon {
    char data;
    Pohon *left, *right, *parent; // Pointer
};

// Pointer global
Pohon *root2311102077;

// Inisialisasi
void init() {
    root2311102077 = NULL;
}

bool isEmpty() {
    return root2311102077 == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
```

```

node->left = NULL;
node->right = NULL;
node->parent = NULL;
return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root2311102077 = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah memiliki child kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child kiri dari " << node->data
<< endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah memiliki child kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child kanan dari " << node-
>data << endl;

```

```

        return baru;
    }
}

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;
        }
    }
}

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;
        }
    }
}

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 : " << root2311102077->data << endl;

            if (!node->parent)
                cout << "Parent : (tidak memiliki parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;

            if (node->parent != NULL && node->parent->left != node && node->parent->right ==
node)

```



```

        cout << "Sibling : " << node->parent->left->data << endl;
        else if (node->parent != NULL && node->parent->right != node && node->parent->left == node)
            cout << "Sibling : " << node->parent->right->data << endl;
        else
            cout << "Sibling : (tidak memiliki sibling)" << endl;

        if (!node->left)
            cout << "Child Kiri : (tidak memiliki child kiri)" << endl;
        else
            cout << "Child Kiri : " << node->left->data << endl;

        if (!node->right)
            cout << "Child Kanan : (tidak memiliki child kanan)" << endl;
        else
            cout << "Child Kanan : " << node->right->data << endl;
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node) {
    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) {
    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) {
    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 != root2311102077) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);

            if (node == root2311102077) {
                delete root2311102077;
                root2311102077 = 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(root2311102077);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node) {
    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) {
    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 characteristic() {
    int s = size(root2311102077);
    int h = height(root2311102077);
}

```

```

cout << "\nSize Tree : " << s << endl;
cout << "Height Tree : " << h << endl;
if (h != 0)
    cout << "Average Node of Tree : " << s / h << endl;
else
    cout << "Average Node of Tree : 0" << endl;
}

// Menampilkan Child dari Sebuah Node
void displayChild(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nChild dari node " << node->data << " adalah:";
            if (node->left) {
                cout << " " << node->left->data;
            }
            if (node->right) {
                cout << " " << node->right->data;
            }
            cout << endl;
        }
    }
}

// Menampilkan Descendant dari Sebuah Node
void displayDescendant(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nDescendant dari node " << node->data << " adalah:";
            // Gunakan rekursi untuk mencetak descendant
            if (node->left) {
                cout << " " << node->left->data;
                displayDescendant(node->left);
            }
            if (node->right) {
                cout << " " << node->right->data;
                displayDescendant(node->right);
            }
            cout << endl;
        }
    }
}

```

```

    }
}

int main() {
    init();
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI, *nodeJ;

    nodeB = insertLeft('B', root2311102077);
    nodeC = insertRight('C', root2311102077);
    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(root2311102077);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root2311102077);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root2311102077);
    cout << "\n" << endl;
    characteristic();
    displayChild(nodeE);
    displayDescendant(nodeB);
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root2311102077);
    cout << "\n" << endl;
    characteristic();
}

```

SRENSHOOT OUTPUT:

```
PostOrder :
D, I, J, G, H, E, B, F, C, A,

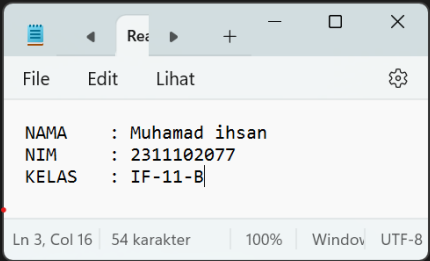
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Child dari node E adalah: G H
Descendant dari node B adalah: D
Descendant dari node D adalah:
E
Descendant dari node E adalah: G
Descendant dari node G adalah: I
Descendant dari node I adalah:
J
Descendant dari node J adalah:
H
Descendant dari node H adalah:

Node subtree E berhasil dihapus.

PreOrder :
A, B, D, E, C, F,

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\Users\ASUS\OneDrive\Desktop\praktikum struktur data\build\model9.cpp>
```



DESKRIPSI:

Program di atas adalah sebuah implementasi struktur data pohon (tree) yang memungkinkan pengguna untuk membuat, mengupdate, dan menghapus node pada pohon. Program ini juga menyediakan fungsi untuk menampilkan karakteristik pohon, seperti ukuran dan tinggi pohon, serta menampilkan child dan descendant dari sebuah node.

D. KESIMPULAN

Pada praktikum ini, mempraktikkan konsep-konsep dasar dari struktur data Graph dan Tree. Kita telah memahami cara membuat graph dan tree, menambahkan node dan edge, serta menampilkan graph dan tree dalam bentuk yang sesuai. Kita juga telah mempelajari cara mengupdate dan menghapus node pada graph dan tree, serta menampilkan karakteristiknya seperti ukuran dan tinggi tree.

Dengan demikian, praktikum ini telah memberikan kita pemahaman yang lebih baik tentang konsep-konsep dasar dari Graph dan Tree, serta kemampuan untuk menerapkan konsep-konsep tersebut dalam berbagai aplikasi. Kita dapat menggunakan graph dan tree untuk merepresentasikan relasi antar objek dalam berbagai bidang, seperti jaringan sosial, jaringan komputer, struktur organisasi, dan lain-lain. Oleh karena itu, praktikum ini sangat berguna untuk meningkatkan kemampuan kita dalam menganalisis dan menyelesaikan masalah yang terkait dengan struktur data.

E. REFERENSI

[1] Asprak “Modul 9 **GRAPH DAN TREE** Learning meaning system

[2] blogspot

<https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html>

[3] Wordpress

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