

**LAPORAN PRAKTIKUM STRUKTUR DATA  
DAN ALGORITMA MODUL 9  
“Graph And Tree”**



**DISUSUN OLEH:**  
**Farrell Edric Kelvianto**  
**2311102079**  
**S1 IF-11-B**

**DOSEN:**

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

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

## **A. Dasar Teori**

### **a. Pengertian Graph dan Tree**

Graf adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dengan bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi. Graf ini terdiri dari simpul dan busur  $G = (V, E)$ ,  $V$  sebagai vertex atau simpul dan  $E$  sebagai Edge atau garis. Graf ini biasa digunakan untuk jaringan, permetaan jalan dan permodelan data. Tree adalah struktur data yang umum dan menyerupai pohon yang nyata. Tree ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terjinimh, dimana setiap node yang memiliki banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu. Tree ini digunakan untuk menyimpan data – data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Graf adalah seperti peta yang digunakan untuk menunjukkan bagaimana objek terhubung satu sama lain. Objek ini direpresentasikan sebagai titik (atau "simpul"), dan hubungannya direpresentasikan oleh garis yang menghubungkan titik-titik tersebut. Garis ini bisa memiliki arah (seperti jalan satu arah) atau tidak (seperti jalan dua arah). Graf ini digunakan dalam berbagai konteks, mulai dari jaringan komputer hingga pemodelan hubungan sosial. Tree adalah seperti pohon kehidupan digital. Mirip dengan pohon di alam, tree ini memiliki satu titik yang berfungsi sebagai "akar," dan dari titik tersebut, terdapat cabang-cabang yang menjalar ke bawah. Setiap cabang mungkin memiliki cabang lain di bawahnya, membentuk struktur yang terorganisir secara hierarkis. Tree ini umumnya digunakan untuk menyimpan data hierarkis, seperti struktur organisasi perusahaan atau direktori file pada komputer. Graf dan tree merupakan dua konsep yang sangat penting dalam pemrograman dan pemodelan data, membantu kita memahami hubungan antar objek dan hierarki dalam suatu sistem.

### **b. Jenis Graph (Pembahasan Dimodul)**

#### **a. Directed Graph**

Graf berarah itu seperti jalur satu arah di jalan raya. Setiap jalur atau sisi dalam graf ini memiliki arah yang spesifik. Misalnya, bayangkan situasi di mana kamu bisa mengikuti seseorang di media sosial, tapi mereka tidak bisa mengikuti balik kamu, seperti di Twitter. Jadi pada intinya directed graf ini graf yang memiliki arah setiap sisinya.

#### **b. Undirected Graph**

Ini mirip dengan jalan dua arah di dunia nyata. Disini, hubungan antara dua titik atau simpul tidak memiliki arah tertentu. Jadi, semisalnya berteman dengan seseorang, maka secara otomatis kamu juga dianggap sebagai teman oleh mereka, begitu pula sebaliknya. Untuk undirect graph ini intinya hanya graf yang setiap simpulnya saling terhubung antar simpul sehingga membentuk edges dan tidak memiliki arah.

#### **c. Weighted Graph**

Graf berbobot ini seperti jalur-jalur di peta dengan informasi tambahan seperti jarak, biaya, atau waktu tempuh di setiap jalurnya. Misalnya, ketika kamu merencanakan perjalanan dan ingin mengetahui jarak dan biaya perjalanan dari satu tempat ke tempat lain, graf berbobot ini memberikan informasi tersebut di setiap jalurnya. Weighted graph ini graf yang memiliki bobot setiap sisinya, dan weighted graf ini bisa berupa directed graf maupun undirected graf.

## Guided

### Guided 1

```
#include <iostream>
#include <iomanip>

using namespace std;
string simpul[7]={"Ciamis ", "Bandung ", "Bekasi ", "Tasikmalaya ", "Cianjur ", "Purwokerto ", "Yogyakarta "};

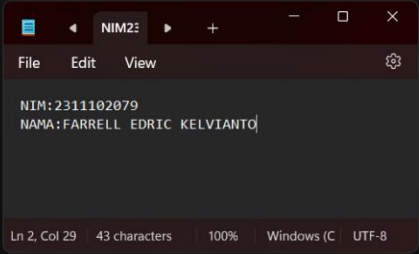
int busur[7][7]={
    {0,7,8,0,0,0,0},
    {0,0,5,0,0,15,0},
    {0,6,0,0,5,0,0},
    {0,5,0,0,2,4,0},
    {23,0,0,10,0,0,8},
    {0,0,0,0,9,4,0},
    {0,0,0,0,9,4,0}
};

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

```
PS C:\Users\VICTUS\Documents\Modul IX Graph> cd "c:\Users\VICTUS\Documents\Modul IX Graph\" ; if ($?) { g++ Guided1.cpp -o Guided1 } ; if ($?) { .\Guided1 }
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 (9) Purwokerto (4)
Yogyakarta  : Cianjur (9) Purwokerto (4)
PS C:\Users\VICTUS\Documents\Modul IX Graph>
```



## Deskripsi Program

Program ini mendeklarasikan array “simpul” dengan tipe data string yang berisikan nama – nama kota. Dan dideklarasikan sebuah array 2x2 sebagai busur/edges dan berisikan bobotnya. Lalu untuk fungsinya terdapat tampilGraph yang berfungsi untuk menampilkan dari fungsi – fungsi yang telah dideklarasikan yaitu “Simpul” dan “Busur”. Lalu untuk cara kerjanya semisal vertex berada di Ciamis dan Ciamis ini saling berhubungan dengan Bandung dan Bekasi karena memiliki bobot dan saling terhubung. Dan untuk setiap indexnya yang berada didalam busur ini berurutan dengan yang dideklarasikan simpul tadi yang berisikan nama – nama kota.

## Guided 2

```
#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 " << data << " berhasil dibuat menjadi root."
              << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child kiri!"
                  << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;

```

```

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

```

```

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

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else

```

```

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

```



```

        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

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

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

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {

```

```

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

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

// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node = root)

```

```

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

```

```

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

```

```

    characteristic();
}

```

## Screenshot Output

```

Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kananA
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kananB
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kananE
Node I berhasil ditambahkan ke child kiri G
Node J berhasil ditambahkan ke child kananG
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
S C:\Users\VICTUS\Documents\Modul IX Graph>

```

## Deskripsi Program

Program ini mendeklarasikan sebuah pohon yang untuk sebagai object dan untuk mengindikasikan kiri,kanan dan parrent. Sebagai berikut untuk penjelasan setiap fungsi – fungsinya dari program ini:

- Init(): ini berfungsi untuk menginisialisasi root/akar adalah kosong/null.
- isEmpty():mendeklarasikan untuk menyatakan root itu adalah kosong atau tidak.

- insertLeft(): ini berfungsi untuk menambahkan node disebelah kiri, dicek dahulu apakah disebelah kiri sudah memiliki child atau belum.
- insertRight(): ini berfungsi untuk menambahkan node disebelah kanan, dicek dahulu apakah disebelah kanan sudah memiliki child atau belum.
- Update() : ini berfungsi untuk mengganti char yang lama menjadi char yang baru.
- Retrieve(): ini berfungsi untuk melihat dari induk pohon dan beserta childnya.
- Find(): ini untuk mencari simpul dan beserta induk dan childnya.
- Preorder(): mencetak seluruh data pada subpohon dapat dibuat rumus menjadi Root – kiri – kanan.
- inOrder(): mencetak seluruh data pada subpohon dapat dibuat rumus menjadi Root – Kiri – Kanan.
- PostOrder(): Untuk post order ini dapat dibuat rumus menjadi Root – kiri – kanan.
- DeleteTree(): berfungsi untuk menghapus akar.
- deleteSub(): berfungsi untuk menghapus subtree.
- Clear () : menghapus semua akar.
- Size () : berfungsi untuk menghitung banyaknya node.
- Height(): berfungsi menghitung tinggi pohon.
- Main(): ini berfungsi untuk mengoperasikan tiap fungsinya.

## B. Tugas

### Unguided 1

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>

using namespace std;

vector<string> simpul;
```

```

vector<vector<int>> busur;

void tampilGraph() {
    cout << endl;
    cout << "      ";
    for (const auto& s : simpul) {
        cout << setw(10) << s;
    }
    cout << endl;

    for (size_t baris = 0; baris < simpul.size(); baris++) {
        cout << setw(10) << simpul[baris];
        for (size_t kolom = 0; kolom < simpul.size(); kolom++) {
            cout << setw(10) << busur[baris][kolom];
        }
        cout << endl;
    }
}

int main() {
    int FarrellEdricKelvianto_2311102079;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> FarrellEdricKelvianto_2311102079;

    simpul.resize(FarrellEdricKelvianto_2311102079);
    busur.resize(FarrellEdricKelvianto_2311102079,
vector<int>(FarrellEdricKelvianto_2311102079, 0));

    cin.ignore();

    for (int i = 0; i < FarrellEdricKelvianto_2311102079; i++) {
        cout << "Simpul " << i + 1 << ": ";
        getline(cin, simpul[i]);
    }

    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < FarrellEdricKelvianto_2311102079; i++) {
        for (int j = 0; j < FarrellEdricKelvianto_2311102079; j++) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }

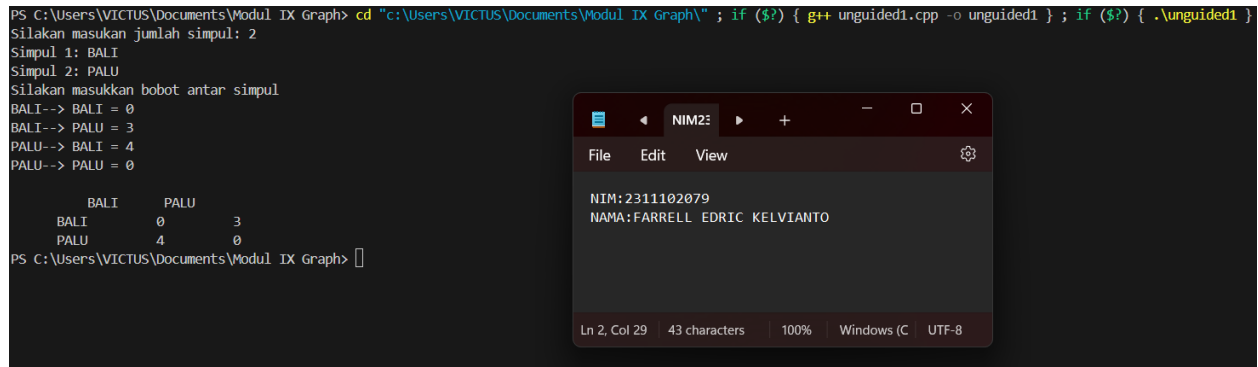
    tampilGraph();
}

```

```
    return 0;
}
```

## Screenshot Output

**Soal no 1.** Membuat program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.



The screenshot shows a Windows command prompt window with the following text:

```
PS C:\Users\VICTUS\Documents\Modul IX Graph> cd "C:\Users\VICTUS\Documents\Modul IX Graph\" ; if ($?) { g++ unguided1.cpp -o unguided1 } ; if ($?) { .\unguided1 }
Silakan masukan jumlah simpul: 2
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
PS C:\Users\VICTUS\Documents\Modul IX Graph>
```

Overlaid on the command prompt is a Notepad++ window titled "NIM2311102079". The text in the Notepad++ window is:

```
NIM:2311102079
NAMA:FARRELL EDRIC KELVIANTO
```

The status bar at the bottom of the Notepad++ window shows "Ln 2, Col 29 | 43 characters | 100% | Windows (C | UTF-8".

## Deskripsi Program

Cara kerja program ini adalah Program ini mendeklarasikan array “simpul” dengan tipe data string yang berisikan nama – nama kota. Dan dideklarasikan sebuah array 2x2 sebagai busur/edges dan berisikan bobotnya. Lalu untuk fungsinya terdapat tampilGraph yang berfungsi untuk menampilkan dari fungsi – fungsi yang telah dideklarasikan yaitu “Simpul” dan “Busur”. Disini mirip dengan guided 1 hanya saja diubah menjadi inputan saja dan untuk inputannya ini di fungsi utamanya itu sendiri. Sebenarnya cara kerja program ini adalah hanya mencetak dari Array saja dan mengikuti index dimensi pada array itu sendiri.

## Unguided 2

```
#include <iostream>
#include <queue>
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 {
        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 " << baru->parent->data << endl;
        return baru;
    }
}

// Tambah Kanan
Pohon* insertRight(char data, Pohon *node) {
    if (root == NULL) {
        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 {
            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 (!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;
    }
}

// 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) {
    if (!root) {
        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 (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
    } else {
        if (!node) {
            return 0;
        } else {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            return max(heightKiri, heightKanan) + 1;
        }
    }
}

// Karakteristik Tree
void characteristic() {
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
    cout << "Average Node of Tree : " << (float)size(root) /
height(root) << endl;
}

// Display Child and Descendants
void displayChildAndDescendants(Pohon *node) {

```

```

    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
        return;
    }

    cout << "\nData Node : " << node->data << endl;

    if (node->left) {
        cout << "Child Kiri : " << node->left->data << endl;
    } else {
        cout << "Child Kiri : (tidak punya Child kiri)" << endl;
    }

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

    cout << "Descendants : ";
    queue<Pohon*> q;
    if (node->left) q.push(node->left);
    if (node->right) q.push(node->right);
    while (!q.empty()) {
        Pohon* current = q.front();
        q.pop();
        cout << current->data << " ";
        if (current->left) q.push(current->left);
        if (current->right) q.push(current->right);
    }
    cout << endl;
}

// Menu
void menu() {
    int pilihan;
    char data;
    Pohon *node = nullptr;

    do {
        cout << "\nMenu:\n";
        cout << "1. Buat Node\n";
        cout << "2. Tambah Kiri\n";
        cout << "3. Tambah Kanan\n";
        cout << "4. Ubah Data\n";
    }

```

```

cout << "5. Lihat Isi Data\n";
cout << "6. Cari Data\n";
cout << "7. PreOrder\n";
cout << "8. InOrder\n";
cout << "9. PostOrder\n";
cout << "10. Hapus SubTree\n";
cout << "11. Hapus Tree\n";
cout << "12. Karakteristik Tree\n";
cout << "13. Tampilkan Child dan Descendants\n";
cout << "0. Keluar\n";
cout << "Pilihan: ";
cin >> pilihan;

switch (pilihan) {
    case 1:
        cout << "Masukkan data node root: ";
        cin >> data;
        buatNode(data);
        break;
    case 2:
if (!isEmpty()) {
        cout << "Masukkan data node: ";
        cin >> data;
        char parentData;
        cout << "Masukkan data node parent: ";
        cin >> parentData;

        // Mencari node parent
        Pohon* parentNode = nullptr;
        queue<Pohon*> q;
        q.push(root);
        while (!q.empty()) {
            Pohon* current = q.front();
            q.pop();
            if (current->data == parentData) {
                parentNode = current;
                break;
            }
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }

        // Menambahkan node kiri jika ditemukan parent
        if (parentNode) {
            insertLeft(data, parentNode);
        }
    }
}

```



```

    } else {
        cout << "Node parent tidak ditemukan!" << endl;
    }
}
else {
    cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree
terlebih dahulu.\n";
}
break;

case 3:
    if (!isEmpty()) {
        cout << "Masukkan data node: ";
        cin >> data;
        cout << "Masukkan data node parent: ";
        cin >> node->data;
        node = root;
        if (node->data == root->data) {
            insertRight(data, node);
        }
        else {
            queue<Pohon*> q;
            q.push(root);
            while (!q.empty()) {
                Pohon* current = q.front();
                q.pop();
                if (current->data == node->data) {
                    insertRight(data, current);
                    break;
                }
                if (current->left) q.push(current->left);
                if (current->right) q.push(current->right);
            }
        }
    }
    else {
        cout << "Tree belum dibuat. Pilih menu 1 untuk
membuat tree terlebih dahulu.\n";
    }
    break;

case 4:
    if (!isEmpty()) {
        cout << "Masukkan data baru: ";
        cin >> data;
        cout << "Masukkan data node yang ingin diubah: ";
        cin >> node->data;
    }

```

```

        node = root;
        if (node->data == root->data) {
            update(data, node);
        }
        else {
            queue<Pohon*> q;
            q.push(root);
            while (!q.empty()) {
                Pohon* current = q.front();
                q.pop();
                if (current->data == node->data) {
                    update(data, current);
                    break;
                }
                if (current->left) q.push(current->left);
                if (current->right) q.push(current->right);
            }
        }
    }
    else {
        cout << "Tree belum dibuat. Pilih menu 1 untuk
membuat tree terlebih dahulu.\n";
    }
    break;
case 5:
    if (!isEmpty()) {
        cout << "Masukkan data node yang ingin dilihat: ";
        cin >> node->data;
        node = root;
        if (node->data == root->data) {
            retrieve(node);
        }
        else {
            queue<Pohon*> q;
            q.push(root);
            while (!q.empty()) {
                Pohon* current = q.front();
                q.pop();
                if (current->data == node->data) {
                    retrieve(current);
                    break;
                }
                if (current->left) q.push(current->left);
                if (current->right) q.push(current->right);
            }
        }
    }
}

```

```

    }
}
else {
    cout << "Tree belum dibuat. Pilih menu 1 untuk
membuat tree terlebih dahulu.\n";
}
break;
case 6:
    if (!isEmpty()) {
        cout << "Masukkan data node yang ingin dicari: ";
        cin >> node->data;
        node = root;
        if (node->data == root->data) {
            find(node);
        }
        else {
            queue<Pohon*> q;
            q.push(root);
            while (!q.empty()) {
                Pohon* current = q.front();
                q.pop();
                if (current->data == node->data) {
                    find(current);
                    break;
                }
                if (current->left) q.push(current->left);
                if (current->right) q.push(current->right);
            }
        }
    }
    else {
        cout << "Tree belum dibuat. Pilih menu 1 untuk
membuat tree terlebih dahulu.\n";
    }
    break;
case 7:
    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    break;
case 8:
    cout << "\nInOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;
    break;

```

```

        case 9:
            cout << "\nPostOrder :" << endl;
            postOrder(root);
            cout << "\n" << endl;
            break;
        case 10:
            if (!isEmpty()) {
                cout << "Masukkan data node subtree yang ingin
dihapus: ";

                cin >> node->data;
                node = root;
                if (node->data == root->data) {
                    deleteSub(node);
                }
                else {
                    queue<Pohon*> q;
                    q.push(root);
                    while (!q.empty()) {
                        Pohon* current = q.front();
                        q.pop();
                        if (current->data == node->data) {
                            deleteSub(current);
                            break;
                        }
                        if (current->left) q.push(current->left);
                        if (current->right) q.push(current->right);
                    }
                }
            }
            else {
                cout << "Tree belum dibuat. Pilih menu 1 untuk
membuat tree terlebih dahulu.\n";
            }
            break;
        case 11:
            clear();
            break;
        case 12:
            characteristic();
            break;
        case 13:
            if (!isEmpty()) {
                cout << "Masukkan data node yang ingin dilihat child dan
descendants-nya: ";
                cin >> data;
            }
    }
}

```

```

        // Mencari node yang sesuai
        Pohon* targetNode = nullptr;
        queue<Pohon*> q;
        q.push(root);
        while (!q.empty()) {
            Pohon* current = q.front();
            q.pop();
            if (current->data == data) {
                targetNode = current;
                break;
            }
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }
        // Jika node ditemukan, tampilkan child dan descendants-nya
        if (targetNode) {
            displayChildAndDescendants(targetNode);
        } else {
            cout << "Node tidak ditemukan!" << endl;
        }
    }
    else {
        cout << "Tree belum dibuat. Pilih menu 1 untuk membuat tree
        terlebih dahulu.\n";
    }
    break;

    case 0:
        cout << "Keluar dari program.\n";
        break;
    default:
        cout << "Pilihan tidak valid! Silakan coba lagi.\n";
    }
} while (pilihan != 0);
}

int main() {
    menu();
    return 0;
}

```

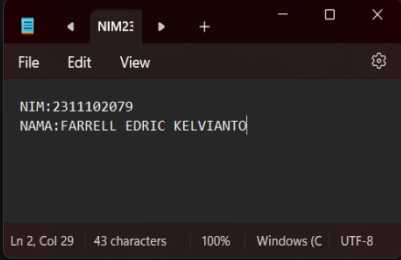
### Screenshot Output

**Soal No.2** tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

```
4. Ubah Data
5. Lihat Isi Data
6. Cari Data
7. PreOrder
8. InOrder
9. PostOrder
10. Hapus SubTree
11. Hapus Tree
12. Karakteristik Tree
13. Tampilkan Child dan Descendants
0. Keluar
Pilihan: 13
Masukkan data node yang ingin dilihat child dan descendants-nya: A

Data Node : A
Child Kiri : S
Child Kanan : (tidak punya Child kanan)
Descendants : S

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



## Deskripsi Program

Program ini hanya menambahkan descendendant dari node yang telah dibuat lalu menampilkan Node child dan descendant dari node yang telah diinputkan. Cara kerja program ini adalah sesuai dengan implementasi nyatanya yang terdiri dari root dan child. Program ini mendeklarasikan sebuah pohon yang untuk sebagai object dan untuk mengindikasikan kiri,kanan dan parrent. Sebagai berikut untuk penjelasan setiap fungsi – fungsinya dari program ini:

- Init(): ini berfungsi untuk menginisialisasi root/akar adalah kosong/null.
- isEmpty():mendeklarasikan untuk menyatakan root itu adalah kosong atau tidak.
- insertLeft(): ini berfungsi untuk menambahkan node disebelah kiri, dicek dahulu apakah disebelah kiri sudah memiliki child atau belum.
- insertRight(): ini berfungsi untuk menambahkan node disebelah kanan, dicek dahulu apakah disebelah kanan sudah memiliki child atau belum.
- Update() : ini berfungsi untuk mengganti char yang lama menjadi char yang baru.
- Retrieve(): ini berfungsi untuk melihat dari induk pohon dan beserta childnya.
- Find(): ini untuk mencari simpul dan beserta induk dan childnya.
- Preorder(): mencetak seluruh data pada subpohon dapat dibuat rumus menjadi Root – kiri – kanan.
- inOrder(): mencetak seluruh data pada subpohon dapat dibuat rumus menjadi Root – Kiri – Kanan.
- PostOrder():Untuk post order ini dapat dibuat rumus menjadi Root – kiri – kanan.
- DeleteTree(): berfungsi untuk menghapus akar.
- deleteSub(): berfungsi untuk menghapus subtree.
- Clear () : menghapus semua akar.
- Size () : berfungsi untuk menghitung banyaknya node.
- Height(): berfungsi menghitung tinggi pohon.
- Main(): ini berfungsi untuk mengoperasikan tiap fungsinya.
- DisplayChildsandDescendant(): ini berfungsi untuk menampilkan dari seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama.

**Kesimpulan**

Graph adalah struktur data yang terdiri dari simpul (node) dan sisi (edge) yang menghubungkan simpul-simpul tersebut. Graph bisa berarah (directed) atau tak berarah (undirected), serta bisa berbobot (weighted) atau tidak berbobot (unweighted). Contohnya, graph digunakan untuk merepresentasikan jaringan sosial, rute transportasi, dan jaringan komputer. Tree adalah jenis khusus dari graph yang tidak memiliki siklus dan memiliki struktur hierarki. Tree memiliki satu simpul utama yang disebut root, dengan simpul lainnya sebagai anak (child) dan daun (leaf). Setiap node dalam tree hanya memiliki satu parent kecuali root. Contoh aplikasi tree adalah dalam struktur file sistem, ekspresi aritmatika, dan berbagai algoritma pencarian. Graph ini lebih fleksibel dan bisa memiliki siklus, sedangkan tree terstruktur hirarki tanpa siklus dan memastikan setiap node hanya memiliki satu parent. Graph cocok untuk hubungan yang kompleks tanpa hirarki, sementara tree ideal untuk data yang membutuhkan struktur hierarki yang jelas.

**Referensi**

Asisten Praktikum. "Modul 9 Graph ". Learning Management System. 2024