

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN  
ALGORITMA**

**MODUL IX  
GRAPH DAN TREE**



**Disusun Oleh :**

Raka Andriy Shevchenko  
2311102054

**Dosen**

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

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

## A. Dasar Teori

### ➤ Struktur Data Graph

Graph adalah struktur data nonlinier yang terdiri dari kumpulan simpul (vertex) yang saling terhubung dengan sisi (edge). Simpul dapat berupa node yang menyimpan data, sedangkan sisi menghubungkan antar simpul. Graph dapat digunakan untuk merepresentasikan aliran komputasi, pemodelan grafik, sistem operasi untuk alokasi sumber daya, dan aplikasi lainnya seperti Google Maps.

### ➤ Kekurangan Graph

Namun, Graph memiliki beberapa kekurangan, seperti:

1. Menggunakan banyak pointer yang dapat rumit untuk ditangani.
2. Memiliki kompleksitas memori yang besar.
3. Jika Graph direpresentasikan dengan adjacency matrix, edge tidak memungkinkan untuk sejajar dan operasi perkalian Graph juga sulit dilakukan.

### ➤ Struktur Data Tree

Tree adalah struktur data yang terdiri dari kumpulan node yang saling terhubung dengan struktur data hirarki (one-to-many). Tree dapat digunakan untuk mengorganisasi informasi berdasarkan struktur logik, memungkinkan cara akses yang khusus terhadap suatu elemen, dan aplikasi lainnya seperti pengembangan game, pengindeksan pada database, dan analisis keputusan.

### ➤ Jenis-jenis Tree

Tree dapat diklasifikasikan ke dalam beberapa jenis, seperti:

1. General Tree: tidak memiliki batasan jumlah node pada hierarki.
2. Binary Tree: setiap node memiliki maksimal dua anak.
3. Balanced Tree: setiap node memiliki keseimbangan antara anak kiri dan kanan.
4. Binary Search Tree: setiap node memiliki nilai yang lebih besar dari anak kiri dan lebih kecil dari anak kanan.

➤ Operasi pada Tree

Operasi-operasi yang dapat dilakukan pada Tree antara lain:

1. Insert: menambahkan node baru ke dalam Tree.
2. Search: mencari node tertentu dalam Tree.
3. Traverse: mengunjungi node-node dalam Tree secara berurutan.
4. Delete: menghapus node dari Tree.

➤ Implementasi Tree

Tree dapat diimplementasikan dalam berbagai bahasa pemrograman, seperti C, Java, dan Python. Implementasi Tree biasanya menggunakan struktur data seperti linked list dan array untuk menyimpan node-node Tree.

## B. Guided

### a. Guided 1

Source Code:

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

Output:

```
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)
```

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Ln 1, Col 1 | 63 characters | 100% | Wind

Deskripsi:

Terdapat tujuh simpul yang disimpan dalam array simpul, yaitu "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogyakarta". Hubungan antar simpul direpresentasikan dalam matriks busur yang menyimpan bobot dari setiap busur yang menghubungkan dua simpul. Jika nilai dalam matriks adalah nol, berarti tidak ada busur antara dua simpul tersebut.

Fungsi tampilGraph() digunakan untuk menampilkan graf tersebut dalam bentuk yang mudah dibaca. Fungsi ini berjalan melalui setiap simpul, dan untuk setiap simpul, ia mencetak nama simpul tersebut diikuti dengan daftar simpul-simpul tujuan beserta bobot busur yang menghubungkannya. Fungsi main() memanggil fungsi tampilGraph() untuk menampilkan graf ke layar ketika program dijalankan.

b. Guided 2

Source Code:

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;

// Inisialisasi
void init()
{
```

```

    root = NULL;
}

// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // 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;
    }
}

// Penelusuran (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();  
}
```

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,

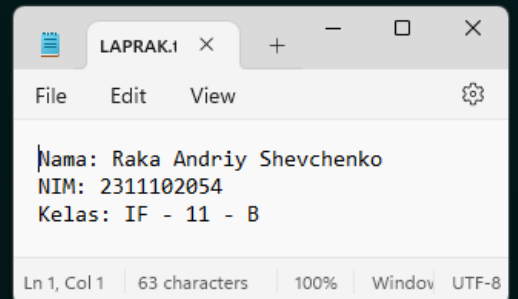
PostOrder :
D, I, J, G, H, E, B, F, C, A,

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
```





### Deskripsi:

Pohon biner direpresentasikan menggunakan struktur Pohon, di mana setiap node memiliki atribut data, left, right, dan parent. Program dimulai dengan inisialisasi pohon menggunakan fungsi `init()` yang mengatur akar (root) pohon menjadi NULL. Fungsi `isEmpty()` digunakan untuk memeriksa apakah pohon kosong, sedangkan fungsi `buatNode(char data)` membuat node baru yang menjadi akar jika pohon masih kosong. Node baru dapat ditambahkan sebagai anak kiri atau kanan menggunakan fungsi `insertLeft(char data, Pohon *node)` dan `insertRight(char data, Pohon *node)`. Fungsi `update(char data, Pohon *node)` digunakan untuk mengubah data pada node tertentu, sementara `retrieve(Pohon *node)` menampilkan data dari node yang ditunjuk. Untuk mencari dan menampilkan informasi lengkap mengenai node, termasuk data root, parent, sibling, dan anak-anaknya, digunakan fungsi `find(Pohon *node)`. Program ini juga menyediakan tiga metode traversal pohon, yaitu `preOrder(Pohon *node = root)`, `inOrder(Pohon *node = root)`, dan `postOrder(Pohon *node = root)`, yang memungkinkan penelusuran pohon dalam urutan pre-order, in-order, dan post-order.

Penghapusan node atau subtree dilakukan melalui fungsi `deleteTree(Pohon *node)` dan `deleteSub(Pohon *node)`, sedangkan seluruh pohon dapat dihapus dengan fungsi `clear()`. Program juga dapat menghitung ukuran dan tinggi pohon menggunakan fungsi `size(Pohon *node = root)` dan `height(Pohon *node = root)`, yang kemudian ditampilkan bersama karakteristik pohon lainnya oleh fungsi `charateristic()`. Fungsi utama `main()` mengilustrasikan penggunaan berbagai fungsi ini dengan membuat node root, menambahkan beberapa node anak, mengubah data node, dan menampilkan data serta informasi detail node. Selain itu, fungsi `main()` juga melakukan berbagai traversal dan menampilkan karakteristik pohon sebelum dan sesudah penghapusan subtree tertentu. Dengan berbagai operasi dasar ini, program menyediakan alat yang komprehensif untuk manipulasi dan analisis pohon biner.

### C. Unguided

#### a. Unguided 1

Source code:

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

using namespace std;

int main()
{
    int raka_2311102054;
    cout << "Silakan masukkan jumlah simpul : ";
    cin >> raka_2311102054;

    string simpul[raka_2311102054];
    int bobot[raka_2311102054][raka_2311102054];
    cout << "Silakan masukkan nama simpul\n";
    for (int i = 0; i < raka_2311102054; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    cout << "Silakan masukkan bobot antar simpul\n";
    for (int i = 0; i < raka_2311102054; i++) {
        for (int j = 0; j < raka_2311102054; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " : ";
            cin >> bobot[i][j];
        }
    }

    cout << endl << setw(10) << " ";
    for (int i = 0; i < raka_2311102054; i++) {
        cout << setw(10) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < raka_2311102054; i++) {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < raka_2311102054; j++) {
            cout << setw(10) << bobot[i][j];
        }
    }
}
```

```

        cout << endl;
    }

    return 0;
}

```

Output:

```

PS C:\kuliah\Struktur Data\TUGAS 9> cd "c:\kuliah\Struktur Data\TUGAS 9\" ;
Silakan masukkan jumlah simpul : 2
Silakan masukkan nama simpul
Simpul 1 : Cirebon
Simpul 2 : Tangerang
Silakan masukkan bobot antar simpul
Cirebon --> Cirebon : 3
Cirebon --> Tangerang : 5
Tangerang --> Cirebon : 4
Tangerang --> Tangerang : 2

          Cirebon Tangerang
Cirebon      3         5
Tangerang    4         2

```

Deskripsi:

Program ini bertujuan untuk membuat dan menampilkan representasi graf berbobot menggunakan matriks ketetanggaan (adjacency matrix) dalam bahasa C++. Awalnya, program meminta pengguna untuk memasukkan jumlah simpul (node) dalam graf dan menyimpannya dalam variabel `raka_2311102054`. Kemudian, program meminta pengguna untuk memasukkan nama-nama setiap simpul yang disimpan dalam array `simpul`. Setelah itu, pengguna diminta untuk memasukkan bobot antar simpul yang disimpan dalam matriks bobot. Setiap bobot menunjukkan hubungan dan berat antara dua simpul tertentu. Program ini menampilkan matriks ketetanggaan dengan nama-nama simpul pada baris dan kolom untuk memudahkan visualisasi hubungan antar simpul. Pada akhir eksekusi, program mencetak tabel yang memperlihatkan nama simpul pada sumbu horizontal dan vertikal, serta bobot hubungan antar simpul di dalam tabel tersebut.

b. Unguided 2

Source Code:

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

// 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 << "\n Node " << data << " berhasil dibuat menjadi
root." << endl;
    } else {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Cari Node Berdasarkan Data
Pohon* findNode(Pohon* node, char data) {
    if (node == NULL) return NULL;
```

```

        if (node->data == data) return node;
        Pohon* foundNode = findNode(node->left, data);
        if (foundNode == NULL) foundNode = findNode(node->right,
data);
        return foundNode;
    }

// Tambah Kiri
Pohon* insertLeft(char data, Pohon* node) {
    if (isEmpty()) {
        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 (isEmpty()) {
        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()) {
        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) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// inOrder
void inOrder(Pohon* node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

// postOrder
void postOrder(Pohon* node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Hapus Node Tree
void deleteTree(Pohon* node) {
    if (node != NULL) {
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root) {
            delete root;
            root = NULL;
        } else {
            delete node;
        }
    }
}

```



```

    }
}

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

// Hapus Tree
void clear() {
    deleteTree(root);
    cout << "\n Pohon berhasil dihapus." << endl;
}

// Cek Size Tree
int size(Pohon* node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

// Cek Height Level Tree
int height(Pohon* node) {
    if (node == NULL) {
        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return max(heightKiri, heightKanan) + 1;
    }
}

// Karakteristik Tree

```

```

void characteristic() {
    cout << "\n Size Tree : " << size(root) << endl;
    cout << " Height Tree : " << height(root) << endl;
    cout << " Average Node of Tree : " << (height(root) == 0 ? 0
: size(root) / height(root)) << endl;
}

// Menampilkan Child Node
void showChildren(Pohon* node) {
    if (node) {
        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;
    }
}

// Menampilkan Descendants Node
void showDescendants(Pohon* node) {
    if (node) {
        cout << " Descendants of Node " << node->data << ": ";
        preOrder(node);
        cout << endl;
    }
}

void menu() {
    int pilihan;
    char data;
    char raka_2311102054;
    Pohon* temp = nullptr;
    do {
        cout << "\nMENU:\n";
        cout << "1. Buat Node Root\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
    }
}

```

```

cout << "4. Update Node\n";
cout << "5. Retrieve Node\n";
cout << "6. Find Node\n";
cout << "7. Tampilkan PreOrder\n";
cout << "8. Tampilkan InOrder\n";
cout << "9. Tampilkan PostOrder\n";
cout << "10. Tampilkan Characteristic\n";
cout << "11. Hapus SubTree\n";
cout << "12. Hapus Tree\n";
cout << "13. Tampilkan Children\n";
cout << "14. Tampilkan Descendants\n";
cout << "0. Keluar\n";
cout << "Masukkan pilihan: ";
cin >> pilihan;
switch (pilihan) {
case 1:
    if (isEmpty()) {
        cout << "Masukkan data root: ";
        cin >> data;
        buatNode(data);
    } else {
        cout << "\n Root sudah ada!" << endl;
    }
    break;
case 2:
    if (!isEmpty()) {
        cout << "Masukkan data node kiri: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);
        insertLeft(data, temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 3:
    if (!isEmpty()) {
        cout << "Masukkan data node kanan: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);

```

```

        insertRight(data, temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 4:
    if (!isEmpty()) {
        cout << "Masukkan data baru: ";
        cin >> data;
        cout << "Masukkan data node yang akan diupdate: ";

        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);
        update(data, temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 5:
    if (!isEmpty()) {
        cout << "Masukkan data node yang akan dilihat: ";

        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);
        retrieve(temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 6:
    if (!isEmpty()) {
        cout << "Masukkan data node yang akan dicari: ";
        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);
        find(temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 7:
    if (!isEmpty()) {
        cout << "\n PreOrder :" << endl;
        preOrder(root);
    }

```

```

        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 8:
    if (!isEmpty()) {
        cout << "\n InOrder :" << endl;
        inOrder(root);
        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 9:
    if (!isEmpty()) {
        cout << "\n PostOrder :" << endl;
        postOrder(root);
        cout << "\n" << endl;
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 10:
    if (!isEmpty()) {
        characteristic();
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 11:
    if (!isEmpty()) {
        cout << "Masukkan data node yang subtreenya akan
dihapus: ";

        cin >> raka_2311102054;
        temp = findNode(root, raka_2311102054);
        deleteSub(temp);
    } else {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 12:
    clear();

```

```

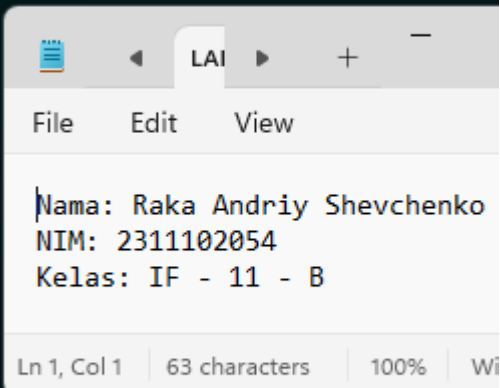
        break;
    case 13:
        if (!isEmpty()) {
            cout << "Masukkan data node yang akan
ditampilkan childnya: ";
            cin >> raka_2311102054;
            temp = findNode(root, raka_2311102054);
            showChildren(temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 14:
        if (!isEmpty()) {
            cout << "Masukkan data node yang akan
ditampilkan descendantnya: ";
            cin >> raka_2311102054;
            temp = findNode(root, raka_2311102054);
            showDescendants(temp);
        } else {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 0:
        cout << "\n Keluar dari program..." << endl;
        break;
    default:
        cout << "\n Pilihan tidak valid!" << endl;
    }
} while (pilihan != 0);
}

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

```

Output:

```
MENU:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Tampilkan PreOrder
8. Tampilkan InOrder
9. Tampilkan PostOrder
10. Tampilkan Characteristic
11. Hapus SubTree
12. Hapus Tree
13. Tampilkan Children
14. Tampilkan Descendants
0. Keluar
Masukkan pilihan: 
```



Masukkan pilihan: 1  
Masukkan data root: 7

Node 7 berhasil dibuat menjadi root.

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 2  
Masukkan data node kiri: 5  
Masukkan data parent: 7

Node 5 berhasil ditambahkan ke child kiri 7

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 3  
Masukkan data node kanan: 9  
Masukkan data parent: 7

Node 9 berhasil ditambahkan ke child kanan 7

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 4  
Masukkan data baru: 6  
Masukkan data node yang akan diupdate: 7

Node 7 berhasil diubah menjadi 6

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 5  
Masukkan data node yang akan dilihat: 6  
  
Data node : 6

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 6  
Masukkan data node yang akan dicari: 5

Data Node : 5  
Root : 6  
Parent : 6  
Sibling : 9  
Child Kiri : (tidak punya Child kiri)  
Child Kanan : (tidak punya Child kanan)

LAIR

File Edit View

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 7

PreOrder :  
6, 5, 9,

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 8

InOrder :  
5, 6, 9,

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 9

PostOrder :  
5, 9, 6,

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 10

Size Tree : 3  
Height Tree : 2  
Average Node of Tree : 1

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

Masukkan pilihan: 13  
Masukkan data node yang akan ditampilkan childnya: 6  
Child Kiri: 5  
Child Kanan: 9

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B



```
Masukkan pilihan: 14
Masukkan data node yang akan ditampilkan descendantnya: 6
Descendants of Node 6: 6, 5, 9,
```

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

```
Masukkan pilihan: 11
Masukkan data node yang subtreenya akan dihapus: 5

Node subtree 5 berhasil dihapus.
```

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

```
Masukkan pilihan: 12

Pohon berhasil dihapus.
```

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

#### Deskripsi:

Pertama-tama, program mendeklarasikan struktur pohon biner yang memiliki data, pointer ke anak kiri, anak kanan, dan parent. Fungsi `init()` menginisialisasi root sebagai NULL, dan `isEmpty()` mengecek apakah pohon kosong atau tidak. Fungsi `buatNode()` membuat node baru sebagai root jika pohon masih kosong. Fungsi `findNode()` mencari node berdasarkan data yang diberikan. Fungsi `insertLeft()` dan `insertRight()` menambahkan node baru sebagai anak kiri atau kanan dari node yang ditentukan, dengan memastikan bahwa posisi yang diinginkan belum terisi.

Program juga menyediakan berbagai fungsi untuk mengelola dan memanipulasi pohon, seperti `update()` untuk mengubah data node, `retrieve()` untuk melihat isi data node tertentu, `find()` untuk mencari dan menampilkan informasi detail tentang node, serta fungsi traversal `preOrder()`, `inOrder()`, dan `postOrder()` untuk menelusuri pohon. Selain itu, ada fungsi `deleteTree()` dan `deleteSub()` untuk menghapus pohon atau subtree, dan `clear()` untuk menghapus seluruh pohon. Fungsi `size()` dan `height()` digunakan untuk menghitung ukuran dan tinggi pohon, sedangkan `characteristic()` menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node.

Program ini juga memiliki fungsi `showChildren()` untuk menampilkan anak-anak dari node tertentu, dan `showDescendants()` untuk menampilkan semua keturunan dari node. Semua fitur ini diakses melalui menu interaktif yang

memungkinkan pengguna untuk membuat, mengubah, menelusuri, dan menghapus pohon biner serta mendapatkan informasi lengkap tentang setiap node. Dengan cara ini, program memberikan cara yang komprehensif untuk memanipulasi dan mengelola struktur data pohon biner.

#### D. Kesimpulan

Graph dan Tree adalah struktur data yang sangat penting dan umum digunakan dalam berbagai aplikasi. Graph digunakan dalam aplikasi seperti Google Maps untuk merepresentasikan jaringan jalan dan memungkinkan pengguna untuk mencari rute terbaik. Graph juga digunakan dalam aplikasi lainnya seperti pemodelan grafik, sistem operasi untuk alokasi sumber daya, dan analisis keputusan.

Tree digunakan dalam aplikasi seperti pengembangan game untuk mengorganisasi informasi berdasarkan struktur logik, memungkinkan cara akses yang khusus terhadap suatu elemen. Tree juga digunakan dalam aplikasi lainnya seperti pengindeksan pada database, analisis keputusan, dan sistem operasi untuk mengorganisasi file dan direktori.

Dalam sintesis, Graph dan Tree adalah struktur data yang sangat penting dan umum digunakan dalam berbagai aplikasi. Graph digunakan untuk merepresentasikan aliran komputasi dan pemodelan grafik, sedangkan Tree digunakan untuk mengorganisasi informasi berdasarkan struktur logik dan aplikasi lainnya.

#### E. Referensi

- [1] Trivusi, Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya. Diakses dari  
[Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya - Trivusi](#)
- [2] Trivusi, Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya. Diakses dari  
[Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya - Trivusi](#)
- [3] Rahma Atillah, Pengertian Struktur Data Graph dan Kegunaannya. Diakses dari  
[Pengertian Struktur Data Graph dan Kegunaannya \(kompas.com\)](#)