

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA  
MODUL 9  
“GRAPH DAN TREE”**



**DISUSUN OLEH:  
TSAQIF KANZ AHMAD  
2311102075  
IF-11-B**

**DOSEN:**

**WAHYU ANDI SAPUTRA S.Pd., M.Eng.**

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

## **TUJUAN PRAKTIKUM**

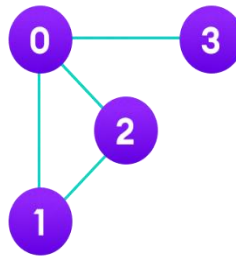
1. Mampu memahami graph dan tree.
2. Mampu mengimplementasikan graph dan tree pada pemrograman.

## DASAR TEORI

### A). GRAPH

#### - Pengertian

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan verteks (V), sedangkan sisi yang menghubungkan antar verteks disebut edge (E). Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y. Sebagai contoh, terdapat graph seperti berikut



**Gambar A.1** Simpul dan tepi pada graph

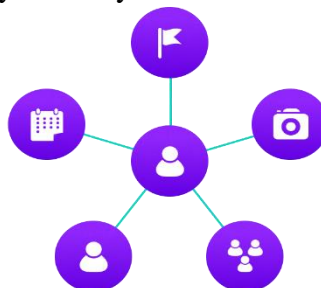
Graph di atas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan sebagai V, edge dilambangkan E, dan graph disimbolkan G, ilustrasi di atas dapat ditulis dalam notasi berikut:

$$V = \{0, 1, 2, 3\}$$

$$E = \{(0,1), (0,2), (0,3), (1,2)\}$$

$$G = \{V, E\}$$

Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk) Pengguna di Facebook dapat dimisalkan sebagai sebuah simpul atau verteks, sementara hubungan pertemanan antara pengguna tersebut dengan pengguna lain direpresentasikan sebagai edge. Tiap tiap verteks dapat berupa struktur yang mengandung informasi seperti id user, nama, gender, dll. Tidak hanya data pengguna, data apapun yang ada di Facebook adalah sebuah simpul atau verteks. Termasuk foto, album, komentar, event, group, story, dll. Pengguna dapat mengunggah foto. Ketika telah diunggah, foto akan menjadi bagian dari album. Foto juga dapat dikomentari oleh pengguna lain dan mereka dapat saling berbalas komentar. Semuanya terhubung satu sama lain, baik dalam bentuk relasi one-to-many, many-to-one, atau many-to-many.



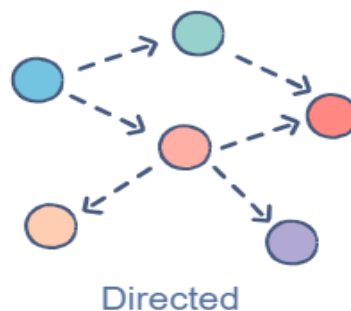
**Gambar A.2** Contoh struktur data graph

## - Jenis-Jenis Graph

Graph dapat dibedakan berdasarkan arah jelajahnya dan ada tidaknya label bobot pada relasinya. Berdasarkan arah jelajahnya graph dibagi menjadi **Undirected graph** (tak berarah), **Directed graph** (berarah) dan **Weighted graph** (berbobot).

### a. Directed Graph (Graph berarah)

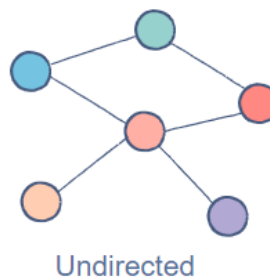
Directed graph adalah graph berarah yang urutan simpulnya mempunyai arti. pada graph jenis ini simpul-simpulnya terhubung oleh edge yang hanya bisa melakukan jelajah satu arah pada simpul yang ditunjuk. Sebagai contoh jika ada simpul A yang terhubung ke simpul B, namun arah panahnya menuju simpul B, maka kita hanya bisa melakukan jelajah (traversing) dari simpul A ke simpul B, dan tidak berlaku sebaliknya.



**Gambar A.3** graph Directed (graph berarah)

### b. Undirected Graph (Graph tak berarah)

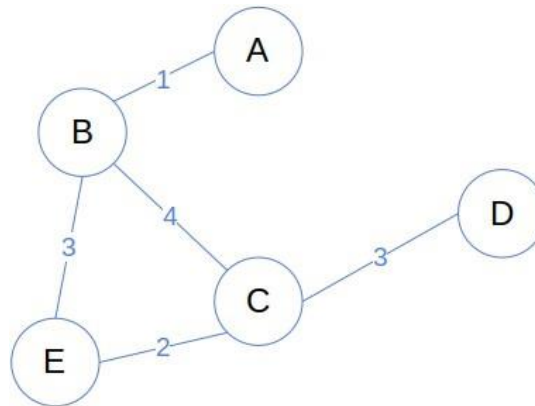
Undirected graph adalah graph tak berarah yang urutan simpulnya dalam sebuah busur tidak diperhatikan. Pada graph ini simpul-simpulnya terhubung dengan edge yang sifatnya dua arah. Misalnya kita punya simpul 1 dan 2 yang saling terhubung, kita bisa menjelajah dari simpul 1 ke simpul 2, begitu juga sebaliknya.



**Gambar A.4** graph Undirected (graph tak berarah)

### c. Weighted graph (Graph berbobot)

Weighted graph adalah jenis graph yang cabangnya diberi label bobot berupa bilangan numerik. Pemberian label bobot pada edge biasanya digunakan untuk memudahkan algoritma dalam menyelesaikan masalah. Contoh implementasinya misalkan kita ingin menyelesaikan masalah dalam mencari rute terpendek dari lokasi A ke lokasi D, namun kita juga dituntut untuk mempertimbangkan kepadatan lalu lintas, panjang jalan dll. Untuk masalah seperti ini, kita bisa mengasosiasikan sebuah edge  $e$  dengan bobot  $w(e)$  berupa bilangan ril.



**Gambar A.5** graph Weighted (graph berbobot)

### - Karakteristik Graph

Graph memiliki beberapa karakteristik sebagai berikut:

- \* Jarak maksimum dari sebuah simpul ke semua simpul lainnya dianggap sebagai eksentrisitas dari simpul tersebut.
- \* Titik yang memiliki eksentrisitas minimum dianggap sebagai titik pusat dari graph.
- \* Nilai eksentrisitas minimum dari semua simpul dianggap sebagai jari-jari dari graph terhubung.

### - Fungsi dan kegunaan graph

- Graph digunakan untuk merepresentasikan aliran komputasi.
- Digunakan dalam pemodelan grafik.
- Graph dipakai pada sistem operasi untuk alokasi sumber daya.
- Google maps menggunakan graph untuk menemukan rute terpendek.
- Graph digunakan dalam sistem penerbangan untuk optimasi rute yang efektif.
- Pada state-transition diagram, graph digunakan untuk mewakili state dan transisinya.
- Di sirkuit, graph dapat digunakan untuk mewakili titik sirkuit sebagai node dan kabel sebagai edge.
- Graph digunakan dalam memecahkan teka-teki dengan hanya satu solusi, seperti labirin.
- Graph digunakan dalam jaringan komputer untuk aplikasi Peer to peer (P2P).
- Umumnya graph dalam bentuk DAG (Directed acyclic graph) digunakan sebagai alternatif blockchain untuk cryptocurrency. Misalnya crypto seperti IOTA

### - Kelebihan dan kekurangan Graph

- **Keunggulan** dari struktur data graph adalah :

- Graph dapat digunakan dengan mudah untuk menemukan jalur terpendek dan tetangga dari node
- Graph digunakan untuk mengimplementasikan algoritma seperti [DFS](#) dan [BFS](#).
- Graph membantu dalam mengatur data.
- Karena strukturnya yang non-linier, membantu dalam memahami masalah yang kompleks dan visualisasinya.

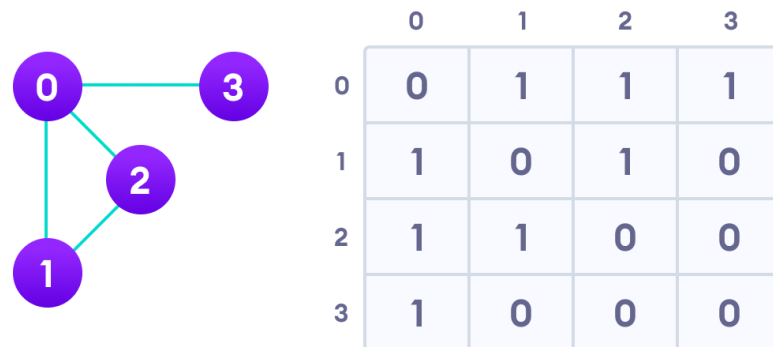
- **Kekurangan** dari struktur data graph adalah :

- Graph menggunakan banyak pointer yang bisa rumit untuk ditangani.
- Memiliki kompleksitas memori yang besar.
- Jika graph direpresentasikan dengan adjacency matrix maka edge tidak memungkinkan untuk sejajar dan operasi perkalian graph juga sulit dilakukan.

- Representasi graph

Graph biasanya direpresentasikan dalam dua cara :

- **Matriks kedekatan (Adjacency Matrix)** adalah larik 2D dengan simpul  $V \times V$ . Setiap baris dan kolom mewakili sebuah titik. Jika suatu elemen bernilai  $a[i][j]1$ , berarti terdapat sebuah sisi yang menghubungkan titik  $i$  dan titik  $j$ . Matriks ketetanggaan untuk graph dapat dibuat sebagai berikut :

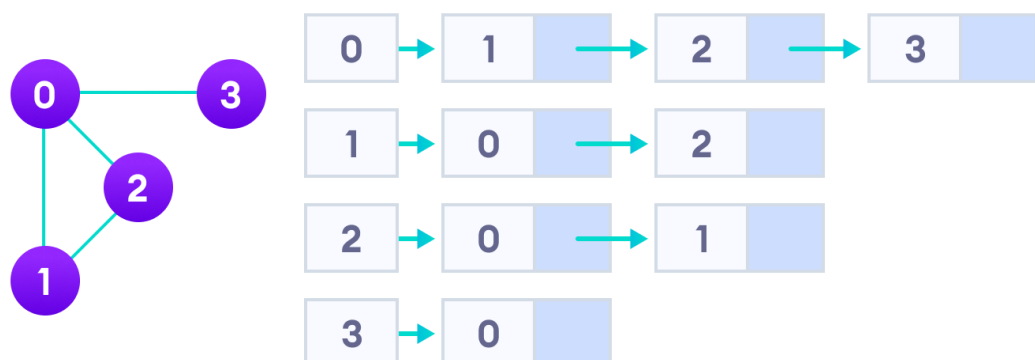


**Gambar A.6** Matriks ketetanggaan grafik

Graph diatas adalah graph yang tidak berarah, maka untuk sisi (0,2) kita juga perlu menandai sisi (2,0); membuat matriks ketetanggaan simetris terhadap diagonal.

Pencarian tepi (memeriksa apakah ada tepi antara simpul A dan simpul B) sangat cepat dalam representasi matriks ketetanggaan tetapi kita harus menyediakan ruang untuk setiap kemungkinan hubungan antara semua simpul ( $V \times V$ ), sehingga memerlukan lebih banyak ruang.

- **Daftar kedekatan (Adjacency List)** adalah Daftar kedekatan mewakili grafik sebagai larik daftar tertaut. Indeks array mewakili sebuah simpul dan setiap elemen dalam daftar tertautnya mewakili simpul-simpul lain yang membentuk suatu sisi dengan simpul tersebut. Daftar ketetanggaan untuk grafik dapat dibuat sebagai berikut:



**Gambar A.7** Representasi daftar kedekatan

Daftar kedekatan efisien dalam hal penyimpanan karena kita hanya perlu menyimpan nilai tepinya. Untuk graf dengan jutaan simpul, hal ini dapat menghemat banyak ruang.

## B. TREE

### - Pengertian

Tree adalah struktur data non linier berbentuk hierarki yang terdiri dari sekelompok node yang berbeda. Struktur data non-linier artinya data pada tree tidak disimpan secara berurutan. Sedangkan hierarki diibaratkan seperti sebuah pohon keluarga di mana terdapat hubungan antara orang tua dan anak. Titik yang lebih atas disebut simpul induk sedangkan simpul di bawahnya adalah simpul anak. Struktur data tree terdiri atas kumpulan simpul atau node dimana tiap-tiap simpul dari tree digunakan untuk menyimpan nilai dan sebuah list rujukan ke simpul lain yang disebut simpul anak atau child node. Tiap-tiap simpul dari tree akan dihubungkan oleh sebuah garis hubung yang dalam istilah teknis disebut edge. Biasanya diimplementasikan menggunakan pointer.

Simpul pada tree bisa memiliki beberapa simpul anak (child node). Namun, jalan menuju sebuah child node hanya bisa dicapai melalui maksimal 1 node. Apabila sebuah node atau simpul tidak memiliki child node sama sekali maka dinamakan leaf node. Struktur data ini adalah metode khusus untuk mengatur dan menyimpan data di komputer agar dapat digunakan secara lebih efektif. Jenis tree yang paling umum digunakan adalah Binary Tree, dimana sebuah tree memiliki maksimal 2 child node.

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

**Gambar B.1** Tabel terminologi dalam Struktur data Tree

### - Istilah-istilah Pada Tree

Layaknya sebuah pohon yang memiliki akar, cabang, dan daun yang terhubung satu sama lain, pada struktur data tree terdapat beberapa istilah penting yang mirip seperti istilah di dunia nyata, antara lain:

#### 1. Node

Node atau simpul adalah entitas pada struktur data tree yang mengandung sebuah nilai dan pointer yang menunjuk simpul di bawahnya (child node).

#### 2. Child node

Child node atau simpul anak adalah simpul turunan dari simpul di atasnya.

#### 3. Leaf Node

Leaf node atau simpul daun adalah simpul yang tidak memiliki child node

dan merupakan node yang paling bawah dalam struktur data tree. Simpul ini biasa disebut juga sebagai external node

### 3. Root

Root atau akar adalah simpul teratas dari sebuah tree.

### 4. Internal node

Internal node adalah istilah untuk menyebut simpul yang memiliki minimal satu child node.

### 5. Edge

Edge merujuk pada garis yang menghubungkan antara dua buah simpul dalam tree. Jika sebuah tree memiliki  $N$  node maka tree tersebut akan memiliki  $(N-1)$  edge. Hanya ada satu jalur dari setiap simpul ke simpul lainnya.

### 6. Height of node

Height of node adalah jumlah edge dari sebuah node ke leaf node yang paling dalam.

### 7. Depth of node

Depth of node adalah banyaknya edge dari root ke sebuah node.

### 8. Height of tree

Height of tree dapat diartikan sebagai panjang jalur terpanjang dari simpul akar ke simpul daun dari sebuah tree.

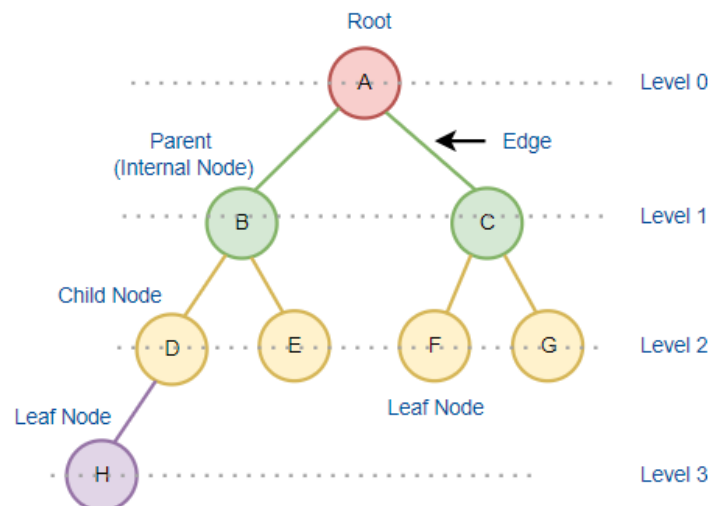
### 9. Degree of node

Jumlah cabang yang melekat pada simpul disebut Degree of node atau derajat simpul. Derajat simpul pada sebuah leaf node adalah 0.

Selain Degree of node, terdapat juga Degree of tree yaitu derajat maksimum simpul di antara semua simpul pada tree.

### 10. Subtree

Subtree adalah setiap simpul dari tree beserta turunannya.



**Gambar B.2** Struktur data pada Tree

#### - Karakteristik Tree

Adapun karakteristik dari struktur data tree adalah sebagai berikut:

- Penjelahan data (traversing) pada tree dilakukan oleh algoritma [Depth First Search](#) dan [Breadth First Search](#)
- Tidak ada loop dan circuit
- Tidak memiliki self-loop
- Disusun dalam model hierarki



## - Jenis-jenis Tree

Struktur data tree dapat diklasifikasikan ke dalam 4 jenis, yaitu: General tree, Binary tree, Balanced tree, dan Binary search tree.

### 1. General tree

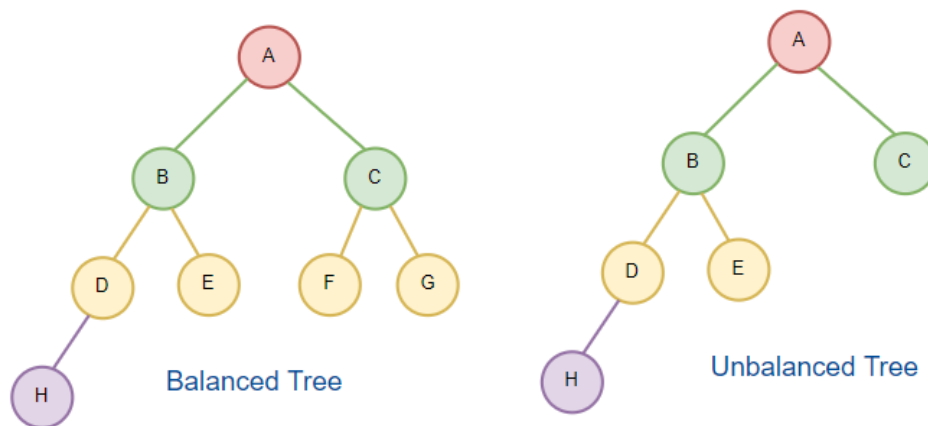
Struktur data tree yang tidak memiliki batasan jumlah node pada hierarki tree disebut General tree. Setiap simpul atau node bebas memiliki berapapun child node. Tree jenis adalah superset dari semua jenis tree.

### 2. Binary tree

Binary tree adalah jenis tree yang simpulnya hanya dapat memiliki paling banyak 2 simpul anak (child node). Kedua simpul tersebut biasa disebut simpul kiri (left node) dan simpul kanan (right node). Tree tipe ini lebih populer daripada jenis lainnya.

### 3. Balanced tree

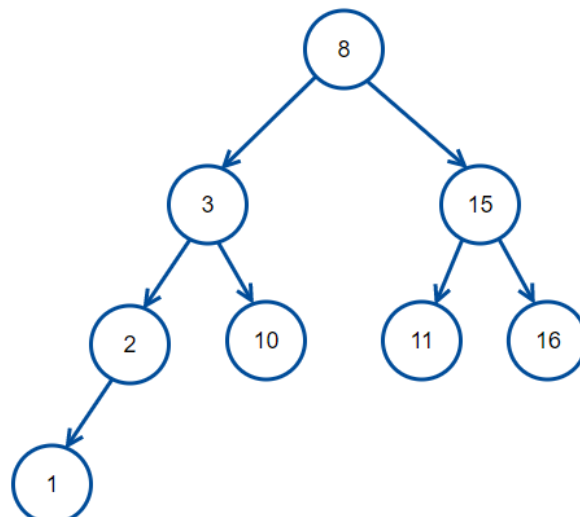
Apabila tinggi dari subtree sebelah kiri dan subtree sebelah kanan sama atau kalaupun berbeda hanya berbeda 1, maka disebut sebagai balanced tree.



**Gambar B.3** Balanced Tree dan Unbalanced Tree

### 4. Binary search tree

Sesuai dengan namanya, Binary search tree digunakan untuk berbagai algoritma pencarian dan pengurutan. Contohnya seperti AVL tree dan Red-black tree. Struktur data tree jenis ini memiliki nilai pada simpul sebelah kiri lebih kecil daripada induknya. Sedangkan nilai simpul sebelah kanan lebih besar dari induknya.



**Gambar B.4** Binary Search Tree

### - Fungsi dan kegunaan Tree

Berikut adalah fungsi dan kegunaan dari struktur data tree

- Dalam kehidupan nyata, struktur data tree membantu dalam pengembangan game.
- Membantu pengindeksan pada database.
- [Decision Tree](#) adalah tools yang biasanya digunakan dalam analisis keputusan. Metode ini memiliki struktur seperti diagram alur yang membantu untuk memahami data.
- Domain Name Server juga menggunakan struktur data tree.
- Kasus penggunaan tree yang paling umum adalah situs jejaring sosial, seperti Facebook, Instagram, Twitter, dll.

### - Keunggulan Struktur Data Tree

Berikut adalah beberapa keunggulan atau kelebihan dari tree:

- Memungkinkan subtree untuk dipindahkan dengan usaha yang minim.
- Mencerminkan hubungan data secara struktural.
- Menawarkan operasi pencarian dan penyisipan yang efisien.
- Tree sangat baik digunakan untuk membuat hierarki data.

### - Operasi pada Tree

- a. **Create**: digunakan untuk membentuk binary tree baru yang masih kosong.
- b. **Clear**: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- c. **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- d. **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.
- e. **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- f. **Update**: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- g. **Retrieve**: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- h. **Delete Sub**: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- i. **Characteristic**: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.
- j. **Traverse**: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

#### 1. Pre-order

Penelusuran secara pre-order memiliki alur:

- a. Cetak data pada simpul root
  - b. Secara rekursif mencetak seluruh data pada subpohon kiri
  - c. Secara rekursif mencetak seluruh data pada subpohon kanan
- Dapat diturunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

Ro - KiP - Ka

#### 2. In-Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
  - b. Cetak data pada root
  - c. Secara rekursif mencetak seluruh data pada subpohon kanan
- dapat diturunkan rumus penelusuran menjadi

### 3. Post Order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
  - Secara rekursif mencetak seluruh data pada subpohon kanan
  - Cetak data pada root
- Dapat diturunkan rumus penelusuran menjadi:

Kiri - Kanan - Root

Ki - Ka - Ro

Atau

Root - Kiri - Kanan(print)

Ro - Ki - KaP

## 1). Program Graph

### SOURCE CODE

## SCREENSHOT OUTPUT

```
PS C:\Users\ACER> cd "c:\Users\ACER\Documents\Struktur data d
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)
```

## **DESKRIPSI PROGRAM**

*Program di atas adalah program implementasi yang menampilkan representasi graf yang berbentuk matriks ketetanggaan (adjacency matrix). Program ini mendefinisikan simpul (nodes) yang berperan array string untuk mengisi nama-nama kota sebagai simpul dari graf dan busur (edges) matriks dua dimensi yang mempresentasikan bobot (weight) dari setiap busur antara simpul. Jika 'busur[i][j]' bernilai 0, maka tidak ada busur langsung dari simpul i ke simpul j. Ketika program dijalankan, program akan menghasilkan output yang menampilkan setiap simpul dan busur yang terhubung dengannya beserta bobot unsur tersebut.*

## 2). Program Tree

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

```

```

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

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

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

```

```

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

```

```

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

// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
}

```

```

    cout << "\n" << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n" << endl;
    charateristic();
}

```

## SCREENSHOT OUTPUT

```

PS C:\Users\ACER> cd "c:\Users\ACER\Documents\Struktur data dan Algoritma\Laprak Strukdat\LaprakModul1"

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)

```

am

File Edit View

Nama : TSAQIF KANZ AHMAD  
 Nim : 2311102075  
 Kelas: IF-11-B

Ln 3, Col 15 | 57 characters | 100% | Window UTF-8

```
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
```

```
PS C:\Users\ACER\Documents\Struktur data dan Algoritma\Laprak Strukdat\LaprakModul 9>
```

 am  + -  X

File Edit View 

Nama : TSAQIF KANZ AHMAD  
Nim : 2311102075  
Kelas: IF-11-B

Ln 3, Col 15 | 57 characters | 100% | Window UTF-8

## DESKRIPSI PROGRAM

Program di atas adalah program implementasi struktur data pohon biner yang setiap nodenya memiliki maksimal dua anak (left dan right). Program ini menyediakan berbagai fungsi untuk mengelola dan memanipulasi pohon biner, termasuk pembuatan node, penambahan anak kiri dan kanan, pengubahan data node, penghapusan subtree dan penelusuran pohon dalam berbagai urutan (preorder, inorder dan postorder).

## PENJELASAN UNGUIDED

Cantumkan NIM pada salah satu variabel didalam program

**Contoh :int nama\_22102003;**

1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

```
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

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

int main() {
    int jmlhsmp1_2311102075;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jmlhsmp1_2311102075;

    string simpul[jmlhsmp1_2311102075];
    int busur[jmlhsmp1_2311102075][jmlhsmp1_2311102075];

    for (int i = 0; i < jmlhsmp1_2311102075; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jmlhsmp1_2311102075; i++) {
        for (int j = 0; j < jmlhsmp1_2311102075; 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 < jmlhsmp1_2311102075; i++) {
        cout << setw(15) << simpul[i];
```

```

    }
    cout << endl;

    for (int i = 0; i < jmlhsmp1_2311102075; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jmlhsmp1_2311102075; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

## SCREENSHOT OUTPUT

```

PS C:\Users\ACER> cd "C:\Users\ACER\AppData\Local\Temp\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeR
Silakan masukkan jumlah simpul: 2
Simpul 1: BALI
Simpul 2: PALU
Silakan masukkan bobot antara simpul BALI dan BALI: 0
Silakan masukkan bobot antara simpul BALI dan PALU: 3
Silakan masukkan bobot antara simpul PALU dan BALI: 4
Silakan masukkan bobot antara simpul PALU dan PALU: 0

Graf yang dihasilkan:
      BALI      PALU
BALI      0      3
PALU      4      0
PS C:\Users\ACER\AppData\Local\Temp>

```

## DESKRIPSI PROGRAM

Program diatas adalah program implementasi graph berbobot menggunakan matriks ketetanggaan (adjacency matrix) untuk menghitung jarak. Pada bagian pertama program meminta pengguna untuk memasukan jumlah simpul dalam graph. Kemudian pengguna memasukan nama setiap simpul, masukan bobot antara setiap pasangan simpul, mengisi matriks ketetanggaan dan program menampilkan matriks ketetanggaan yang menunjukan bobot antara setiap pasangan simpul dalam graf.



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 diinputkan !

### SOURCE CODE

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

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

// Pointer global
Pohon *root_2311102075;

// Inisialisasi
void init() {
    root_2311102075 = NULL;
}

bool isEmpty() {
    return root_2311102075 == 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_2311102075 = 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 : " << root_2311102075->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 != root_2311102075) {
                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_2311102075) {
                delete root_2311102075;
            }
        }
    }
}

```

```

        root_2311102075 = 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_2311102075);
        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_2311102075);
    int h = height(root_2311102075);
    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', root_2311102075);
    nodeC = insertRight('C', root_2311102075);
    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);

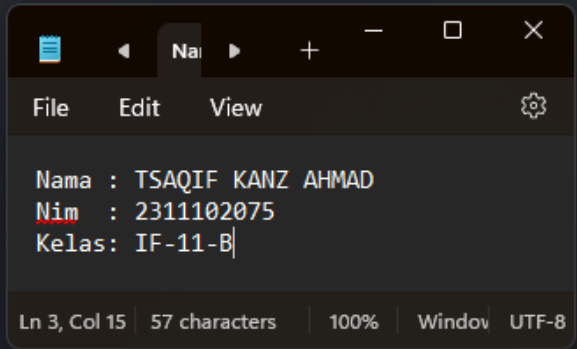
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\nPreOrder :" << endl;
    preOrder(root_2311102075);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root_2311102075);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root_2311102075);
    cout << "\n" << endl;
    characteristic();
    displayChild(nodeE);
    displayDescendant(nodeB);
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root_2311102075);
    cout << "\n" << endl;

```

```
} characteristic();
```

## SCREENSHOOT OUTPUT

```
Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri dari A
Node C berhasil ditambahkan ke child kanan dari A
Node D berhasil ditambahkan ke child kiri dari B
Node E berhasil ditambahkan ke child kanan dari B
Node F berhasil ditambahkan ke child kiri dari C
Node G berhasil ditambahkan ke child kiri dari E
Node H berhasil ditambahkan ke child kanan dari E
Node I berhasil ditambahkan ke child kiri dari G
Node J berhasil ditambahkan ke child kanan dari 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 memiliki 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

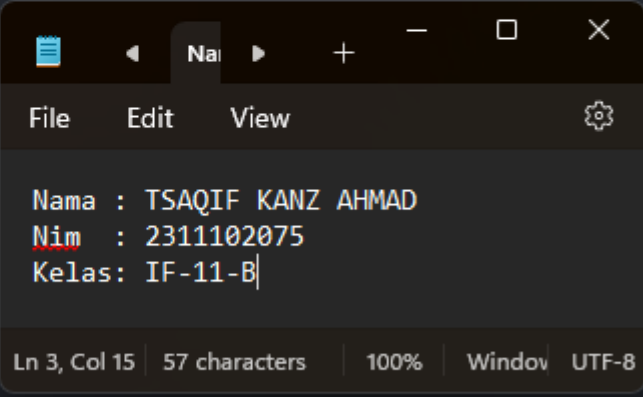
Child dari node E adalah: G H

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\ACER\AppData\Local\Temp>
```



## DESKRIPSI PORGRAM

*Program tersebut adalah program tree yang sama seperti guided diatas tetapi dimodifikasi dengan program menu menggunakan input data tree dari pengguna dan beberapa fungsi yaitu menggunakan variabel secara global dalam fungsi-fungsi lainnya, sehingga tidak perlu dideklarasikan secara lokal setiap fungsi, kemudian penulisan fungsi-fungsi menggunakan notasi lebih sederhana sehingga beberapa fungsi tidak lagi menggunakan pengecekan kondisi secara eksplisit, kemudian penggunaan default parameter digunakan untuk fungsi-fungsi penelusuran (preOrder, inOrder dan postOrder) sehingga program lebih fleksibel, kemudian penulisan fungsi 'charateristic' membuat sistem lebih sederhana, kemudian penggunaan else yang memisahkan block else secara tersendiri, penanganan penghapusan node pada penghapusan subtree dilakukan lebih sederhana menggunakan fungsi 'deleteSub'.*

## KESIMPULAN

**Graph dan Tree** adalah struktur data yang digunakan untuk merepresentasikan hubungan antar objek. Graph terdiri dari titik-titik (node) dan garis (edge) yang menghubungkan node-node tersebut. Edge dapat memiliki bobot, yang merepresentasikan nilai seperti biaya atau waktu. Ciri-ciri pada graph tidak memiliki struktur hierarki, memiliki banyak komponen dan memiliki sirkuit. Contoh penggunaan pada graph seperti jaringan sosial, peta jalan dan jaringan komputer. Sedangkan Tree adalah struktur data hierarkis yang terdiri dari node yang disebut root dan node-node lain yang disebut anak (children). Setiap node dapat memiliki maksimal satu parent (induk). Ciri-ciri pada Tree memiliki struktur hierarki dengan root sebagai node teratas, selalu terhubung dan tidak memiliki sirkuit. Contoh penggunaan pada tree seperti struktur organisasi, file sistem dan pohon genealogi.

## DAFTAR PUSTAKA

- [1] Asisten Praktikum, "Modul 9 Graph dan Tree" : [MODUL 9 - GRAPH DAN TREE.pdf](#)
- [2] Trivusi – Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya : <https://www.trivusi.web.id/2022/07/struktur-data-graph.html>
- [3] Trivusi - Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya : <https://www.trivusi.web.id/2022/07/struktur-data-tree.html#comments>