

**LAPORAN PRAKTIKUM  
STRUKTUR DATA PEMOGRAMAN**

**MODUL IX  
“GRAPH DAN TREE”**



**Disusun Oleh:**

NAMA : Trie Nabilla Farhah

NIM : 2311102071

**Dosen Pengampu:**

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

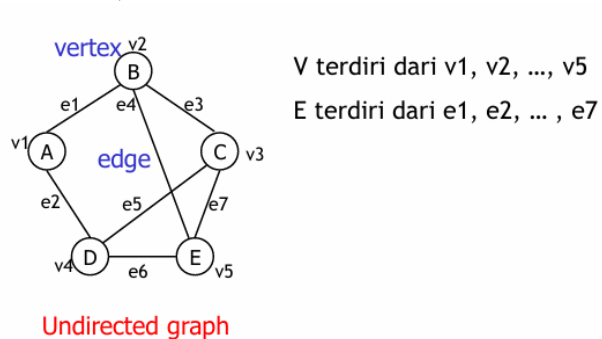
**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO  
PURWOKERTO**

**2024**

## A. DASAR TEORI

### 1. GRAPH

Graph adalah struktur data yang memiliki relasi many to many, yaitu tiap element dapat memiliki 0 atau lebih dari 1 cabang. Konsep graph adalah konsep struktur data yang terdiri dari node (vertex atau vertices) dan garis penghubung (arc atau edge). Vertex disimbolkan dengan “V” dan edge disimbolkan dengan “E”. Node digunakan untuk menyimpan data, sedangkan Edge, untuk menghubungkan node satu dengan node lain. Keterhubungan graph dapat membentuk relasi One to-One, One-to-Many, Many-to-One dan Many-to-Many. Contoh: Informasi topologi jaringan, keterhubungan antar kota-kota, dll.

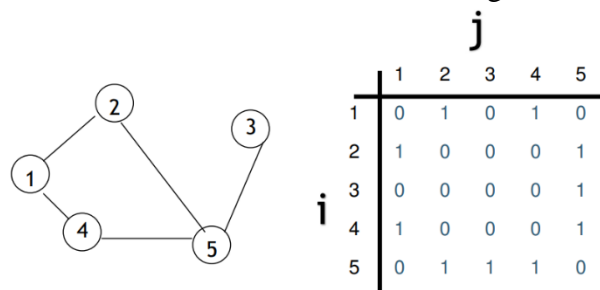


Dalam kehidupan sehari-hari Graph banyak digunakan untuk menggambarkan jaringan dan peta jalan, jalan kereta api, lintasan pesawat, system perpipaan, saluran telepon, koneksi elektrik, ketergantungan diantara task pada sistem manufaktur dan lain-lain. Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan graph.

Representasi graph dibedakan menjadi 2 yaitu **Adjacency Matrices** (Array Based) : implementasi menggunakan array matriks, dan **Adjacency List** (Linked List) : implementasi menggunakan linked list.

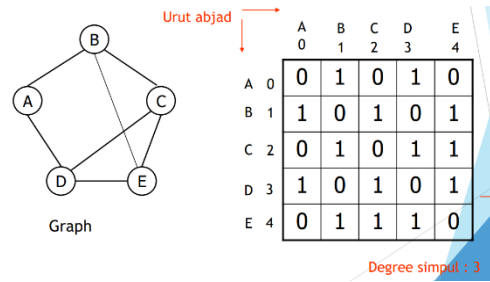
#### a. Adjacency Matrix

Representasi Graph berupa Matrik ordo  $n \times n$ . dimana  $n$  = node. Baris berisi Node asal, sedangkan kolom berisi Node tujuan. Jika graph tidak berbobot, maka nilai matriks diisi dengan 1 atau 0. Nilai 1 jika ada edge, dan 0 jika tidak ada edge antar node.  $A(i,j) = 1$ , jika antara node  $i$  dan node  $j$  terdapat edge/terhubung. Jika graph berbobot, maka nilai matriks diisi dengan bobot dari edge.  $A(i,j)$  = nilai bobot.

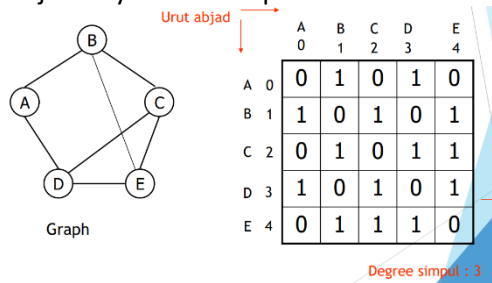


Representasi Graph Dalam Bentuk Matrix

- Adjacency Matrix Graph tak berarah

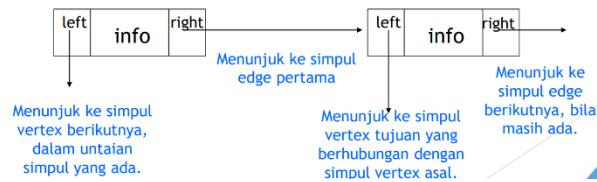


- Adjacency Matrix Graph berarah



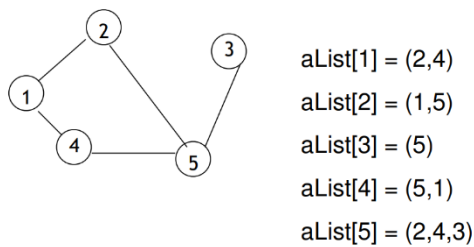
## b. Adjacency Lists

Representasinya dengan linked list. Node asal akan disebut Successor. Isi dari successor akan diisi list node yang berkaitan, node terakhir akan diisi pada nextnya.

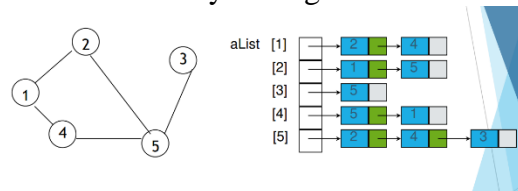


Direpresentasikan dengan linked list atau array

- Array list : array dua dimensi namun tidak ber-ordo nxn.



- Linked list : array of single linked list

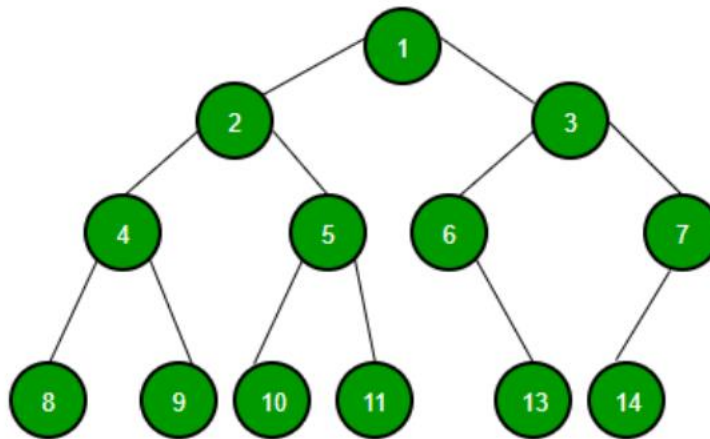


## 2. TREE

Dalam ilmu komputer, pohon biner adalah struktur data pohon di mana setiap simpul memiliki paling banyak dua anak, yang disebut sebagai anak kiri dan anak kanan. Sebuah definisi rekursif hanya menggunakan teori himpunan gagasan adalah bahwa (non-kosong) pohon biner adalah tuple  $(L, S, R)$ , di mana  $L$  dan  $R$  adalah pohon biner atau himpunan kosong dan  $S$  adalah himpunan tunggal.

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.



### 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

**4. Root**

Root atau akar adalah simpul teratas dari sebuah tree.

**5. Internal node**

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

**6. 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.

**7. Height of node**

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

**8. Depth of node**

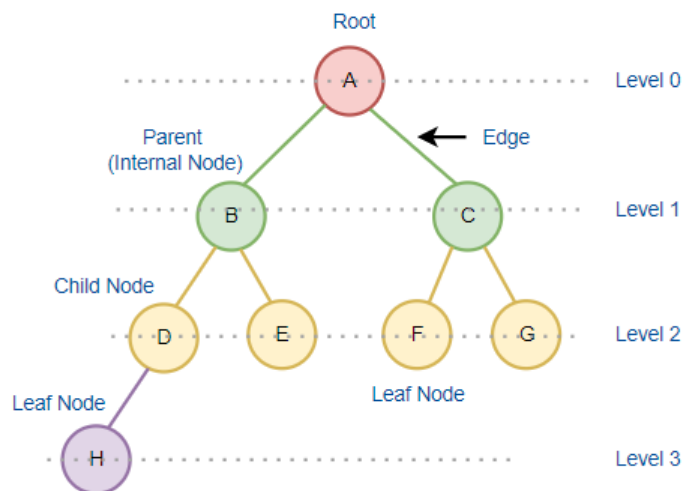
Depth of node adalah banyaknya edge dari root ke sebuah node.

**9. Height of tree**

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

**10. Subtree**

Subtree adalah setiap simpul dari tree beserta turunannya.



**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

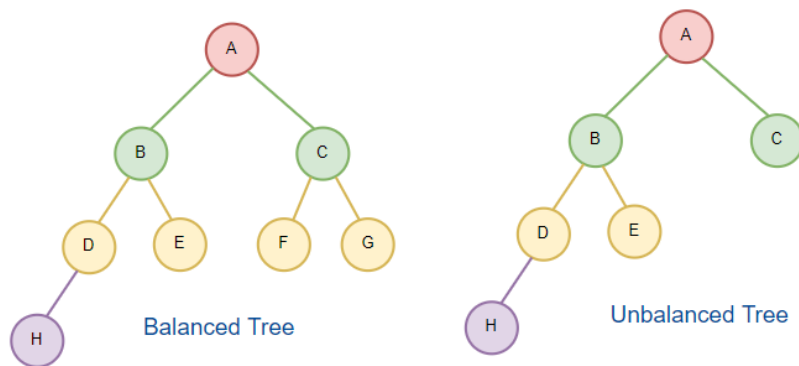
Struktur data tree yang tidak memiliki batasan jumlah node pada hierarki tree disebut General tree. Setiap simpul atau node bebas memiliki berapapun child node. Tree jenis adalah superset dari semua jenis tree.

### 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.

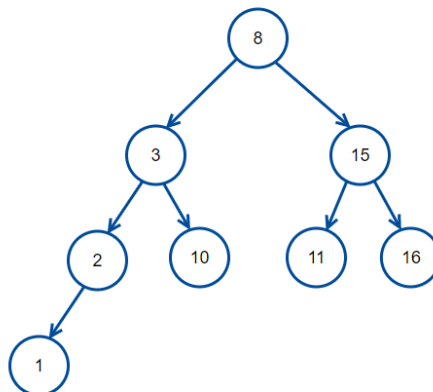
### 3. Balanced tree

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



## B. GUIDED

### GUIDED 1 : GRAPH

#### 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

```

PROBLEMS OUTPUT TERMINAL PORTS
WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-yepsompp.dcj' '--stdout=Microsoft-MIEngine-Out-tzyusqd2.jvm' '--stderr=Microsoft-MIEngine-Error-clw
siy2m.ua0' '--pid=Microsoft-MIEngine-Pid-sid4ndnu.mwp' '--dbgExe=C:\msys64\bin\gdb.exe' '--interpreter=mi'
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS D:\Semester2\Struktur Data\Praktikum9>

```

## Deskripsi Program

Program C++ diatas adalah implementasi sederhana untuk menampilkan representasi graf dalam bentuk matriks ketetanggaan (adjacency matrix). Program ini memodelkan kota-kota dan jarak antar kota dalam sebuah graf, dengan simpul (nodes) mewakili kota dan busur (edges) mewakili jarak antar kota. Program memulai dengan mendefinisikan ukuran array "n" dan menginisialisasikannya dengan nilai "{9, 4, 1, 7, 5, 12, 4, 13, 4, 10}". Selain itu, program mendefinisikan nilai target "cari" ke nilai "10".

- ‘Simpul’ adalah array yang mengandung nama kota yang akan digunakan sebagai simpul pada graf.
- ‘Busur’ adalah matriks ketetanggaan yang mencatat jarak antara dua kota. Nilai busur[i][j] menunjukkan jarak antara kota i dan j, dan nilai 0 menunjukkan bahwa tidak ada hubungan langsung antara kota i dan j.
- Fungsi graph
  - Masuk ke setiap baris matriks busur.



- Menampilkan nama kota di setiap kolom baris saat ini.
- Nilai busur [baris][kolom] tidak sama dengan 0 menunjukkan bahwa ada hubungan langsung antara kota yang diwakili oleh baris dan kota yang diwakili oleh kolom.
- Tampilkan kemudian kota tujuan dan jarak dari sana.
- Nama kota ditampilkan dengan lebar 15 karakter dan rata kiri dengan menggunakan setw(15) dan setiosflags(ios::left).

## GUIDED 2 : TREE

### Source Code

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

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();
    }

```

## Output

```

PS D:\Semester2\Struktur Data\Praktikum9> & 'c:\Users\Asus\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-zkkbjg2g.een' '--stdout=Microsoft-MIEngine-Out-nbsv1an1.gtg' '--stderr=Microsoft-MIEngine-Error-hrsq05sm.xfu' '--pid=Microsoft-MIEngine-Pid-ppxv1lvr.cdg' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'

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,

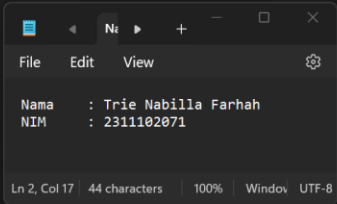
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2

PS D:\Semester2\Struktur Data\Praktikum9>
```



## Deskripsi Program

Program ini adalah contoh sederhana dari pohon biner dalam C++. Berikut adalah penjelasan menyeluruh tentang cara kerja program ini:

- Pohon adalah struktur data yang mewakili setiap node dalam pohon biner.
- Setiap node memiliki data (karakter), pointer ke anak kiri (kiri), anak kanan (kanan), dan orang tua (orang tua).
- Root adalah pointer sementara yang digunakan untuk membuat node baru, dan root adalah pointer ke node akar dari pohon biner.
- Fungsi init mengaktifkan pohon dengan mengubah pangkalan ke NULL, yang berarti pohon kosong.
- Fungsi isEmpty menentukan apakah pohon kosong dengan memeriksa apakah root adalah NULL.
- Jika pohon kosong, fungsi buatNode membuat node baru sebagai root. Jika pohon sudah ada, pesan "Pohon sudah dibuat" ditampilkan.
- Jika belum ada anak kiri, fungsi insertLeft menambah node kiri ke node yang telah ditentukan sebelumnya.
- Jika belum ada anak kanan, fungsi insertRight menambah node kanan ke node yang ditentukan.
- Data di node tertentu diubah melalui fungsi update.
- Data dari node tertentu ditampilkan melalui fungsi retrieve.
- Data node, root, parent, sibling, dan anak-anaknya disertakan dalam detail node yang ditentukan dengan fungsi find.
- Dalam urutan pre-order, fungsi pre-order mencari pohon.
- Fungsi inOrder melacak pohon dalam urutan in-order.
- Fungsi PostOrder Menuelusuri phon dalam urutan port-order.
- Fungsi deleteTree menghapus pohon atau subtree tertentu secara keseluruhan.
- Fungsi deleteSub menghilangkan subtree dari node tertentu.
- Seluruh pohon dihapus dengan fungsi clear.

- Jumlah node dalam pohon dihitung dengan fungsi size.
- Menghitung tinggi pohon adalah fungsi height.
- Fungsi characteristic menunjukkan ukuran, tinggi, dan rata-rata node pohon.

## C. UNGUIDED

### Unguided 1

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

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int trienabilla_2311102071;
    cout << "Silakan masukan jumlah simpul : ";
    cin >> trienabilla_2311102071;

    vector<string> simpul(trienabilla_2311102071);
    cout << "Silakan masukan nama simpul\n";
    for (int i = 0; i < trienabilla_2311102071; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    vector<vector<int>> bobot(trienabilla_2311102071,
vector<int>(trienabilla_2311102071));
    cout << "Silakan masukan bobot antar simpul\n";
    for (int i = 0; i < trienabilla_2311102071; i++) {
        for (int j = 0; j < trienabilla_2311102071; j++) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << "\n";
    cout << "\t";
    for (int i = 0; i < trienabilla_2311102071; i++) {
        cout << simpul[i] << "\t";
    }
    cout << endl;
    for (int i = 0; i < trienabilla_2311102071; i++) {
        cout << simpul[i] << "\t";
```

```

        for (int j = 0; j < trienabilla_2311102071; j++) {
            cout << bobot[i][j] << "\t";
        }
        cout << endl;
    }

    return 0;
}

```

## Output

```

PS D:\Semester2\Struktur Data\Praktikum9> & 'c:\Users\Asus\.vscode\extensions\ms-vscode.cpptools-1.28.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-suvwzluv.isu' '--stdout=Microsoft-MIEngine-Out-54235bca.ty' '--stderr=Microsoft-MIEngine-Error-3egt3p50.fdh' '--pid=Microsoft-MIEngine-Pid-utpdabwn.e0j' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : jakarta
Simpul 2 : bali
Silakan masukkan bobot antar simpul
jakarta-> jakarta = 2
jakarta-> bali = 7
bali-> jakarta = 8
bali-> bali = 3

      jakarta bali
jakarta 2      7
bali    8      3
PS D:\Semester2\Struktur Data\Praktikum9>

```

## Deskripsi Program

Program di atas merupakan program C++ yang membuat matriks untuk merepresentasikan bobot antar node dalam suatu grafik. Berikut rincian programnya:

- Pertama, program meminta pengguna untuk memasukkan jumlah node (n) yang ada di dalam grafik.
- Setelah itu, aplikasi meminta pengguna untuk memberi tahu nama setiap node (simpul[i]).
- Selanjutnya, pengguna diminta untuk memasukkan bobot antara setiap pasangan node dalam aplikasi (bobot[i][j]). Node1 memasukkan bobot dan node2 menghitungnya.
- Setelah itu, program mengeluarkan matriks bobot dalam format terstruktur. Nama node ditunjukkan sebagai header dibaris pertama output. Setiap baris berikutnya menunjukkan bobot antara setiap node dan setiap node lainnya. Tab ( ) membedakan bobot.
- membuat variabel integer n untuk menyimpan jumlah node.

- `cout << "Silakan masukan jumlah simpul : ";` mengatakan kepada pengguna untuk memasukkan jumlah node.
- `cin>> n;` membaca input pengguna dan menyimpannya dalam `n`.
- `vektor simpul(n);` mendeklarasikan vektor simpul yang digunakan untuk menyimpan nama node.
- Selanjutnya, program mengulangi setiap node dan meminta pengguna untuk memberikan nama menggunakan `cout` dan `cin`.
- `vector<vector> bobot(n, vector(n));` mendeklarasikan vektor bobot dua dimensi yang menyimpan bobot antara node.
- Selanjutnya, program mengulangi setiap pasangan node dan meminta pengguna untuk memasukkan bobot antara mereka dengan menggunakan `cout` dan `cin`. Akhirnya, program menggunakan `cout` untuk mengeluarkan matriks bobot dalam format yang terstruktur.

## Unguided 2

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

### Source Code

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    string data;
    Node* left;
    Node* right;
};

Node* createNode(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

Node* insertNode(Node* root, string data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data <= root->data) {
        root->left = insertNode(root->left, data);
    } else {
```

```

        root->right = insertNode(root->right, data);
    }
    return root;
}

void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    cout << root->data << " ";
    inorderTraversal(root->right);
}

void displayChild(Node* root, string parent) {
    if (root == NULL) return;
    if (root->data == parent) {
        if (root->left != NULL)
            cout << "Child kiri dari " << parent << ": " <<
root->left->data << endl;
        if (root->right != NULL)
            cout << "Child kanan dari " << parent << ": " <<
root->right->data << endl;
        return;
    }
    displayChild(root->left, parent);
    displayChild(root->right, parent);
}

void displayDescendant(Node* root, string parent) {
    if (root == NULL) return;
    if (root->data == parent) {
        cout << "Descendant dari " << parent << ": ";
        inorderTraversal(root->left);
        inorderTraversal(root->right);
        cout << endl;
        return;
    }
    displayDescendant(root->left, parent);
    displayDescendant(root->right, parent);
}

void trienabilla_2311102071() {
    Node* root = NULL;
    int choice;
    string data, parent;

    do {

```

```

        cout << "\n-----"
<< endl;
        cout << "                MENU                "
<< endl;
        cout << "-----"
<< endl;
        cout << "1. Masukan Node"
<< endl;
        cout << "2. Menampilkan inorder traversal"
<< endl;
        cout << "3. Menampilkan child of a node"
<< endl;
        cout << "4. Menampilkan descendant of a node"
<< endl;
        cout << "5. Keluar"
<< endl;
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data untuk node baru: ";
                cin >> data;
                root = insertNode(root, data);
                break;
            case 2:
                cout << "Inorder traversal tree: ";
                inorderTraversal(root);
                cout << endl;
                break;
            case 3:
                cout << "Masukkan nama node yang ingin
ditampilkan child-nya: ";
                cin >> parent;
                displayChild(root, parent);
                break;
            case 4:
                cout << "Masukkan nama node yang ingin
ditampilkan descendant-nya: ";
                cin >> parent;
                displayDescendant(root, parent);
                break;
            case 5:
                cout << "Terima kasih!\n";
                break;
            default:
                cout << "Pilihan tidak valid!\n";

```



```

    }
    } while (choice != 5);
}

int main() {
    trienabilla_2311102071();
    return 0;
}

```

## Output

```

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: a

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: b

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: c

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 2
Inorder traversal tree: a b c

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 3
Masukkan nama node yang ingin ditampilkan child-nya: b
Child kanan dari b: c

-----
MENU
-----
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan child of a node
4. Menampilkan descendant of a node
5. Keluar
Pilihan Anda: 4
Masukkan nama node yang ingin ditampilkan descendant-nya: a
Descendant dari a: b c

```

```
-----  
MENU  
-----  
1. Masukan Node  
2. Menampilkan inorder traversal  
3. Menampilkan child of a node  
4. Menampilkan descendant of a node  
5. Keluar  
Pilihan Anda: 4  
Masukkan nama node yang ingin ditampilkan descendant-nya: a  
Descendant dari a: b c  
  
-----  
MENU  
-----  
1. Masukan Node  
2. Menampilkan inorder traversal  
3. Menampilkan child of a node  
4. Menampilkan descendant of a node  
5. Keluar  
Pilihan Anda: 5  
Terima kasih!  
PS D:\Semester2\Struktur Data\Praktikum9>
```

## Deskripsi Program

Program ini berfungsi sebagai struktur data pohon, yang memungkinkan pengguna melakukan berbagai operasi pada pohon tersebut. Berikut adalah penjelasan tentang cara kerja program:

- Program dimulai dengan menciptakan struct node dengan tiga fitur: data untuk menyimpan nilai node, kiri untuk menyimpan pointer ke node anak kiri, dan kanan untuk menyimpan pointer ke node anak kanan.
- Untuk membuat node baru dengan nilai tertentu, gunakan fungsi `createNode`.
- Nilai node baru dapat dimasukkan ke dalam pohon dengan menggunakan fungsi `insertNode`, yang akan membandingkan nilai node baru dengan nilai node yang sudah ada di pohon dan kemudian memasukkan node baru ke anak kiri atau anak kanan node yang sesuai.
- Nilai node dalam batang dapat ditampilkan dengan menggunakan fungsi `inorderTraversal`, yang berarti mengunjungi node kiri terlebih dahulu, kemudian node induk, dan terakhir node kanan.
- Untuk menampilkan anak dari node tertentu, fungsi `displayChild` mencari node yang sesuai dengan nama yang diberikan dan menampilkan anak kiri dan kanan dari node tersebut.
- Fungsi `displayDescendant` menampilkan turunan dari sebuah node tertentu. Fungsi ini akan menemukan node yang sesuai dengan nama yang diberikan dan kemudian menampilkan semua node yang berada di bawah node tersebut.
- Fungsi `trienabilla_2311102071` adalah fungsi inti program, yang akan memberi pengguna tampilan menu. Pengguna dapat memasukkan node baru, menampilkan traversal inorder, menampilkan anak node, menampilkan descendant node, atau keluar dari program.
- Sampai pengguna memilih untuk keluar dari program, program akan terus berjalan.

#### **D. KESIMPULAN**

Dalam ilmu komputer, dua struktur data utama adalah graf dan tree. Graph terdiri dari simpul yang menghubungkan pasangan simpul dan tepi. Graph tak berarah (undirected) dan berarah adalah dua kategori utama. Ada dua pilihan untuk representasi adjacency graph: matrix adjacency atau list adjacency. DFS, BFS, dan Dijkstra adalah algoritma penting untuk jalur terpendek; Kruskal dan Prim adalah algoritma untuk Minimum Spanning Tree (MST).

Tree adalah struktur data hierarkis yang tidak memiliki cycle. Root, parent, child, dan leaf adalah komponen utama, dengan karakteristik seperti tinggi dan kedalaman. Binary tree, binary search tree (BST), seimbang AVL tree, dan B-tree untuk database adalah beberapa jenis tree. Untuk tree, algoritma penting meliputi traversal (preorder, inorder, postorder), insertion dan deletion, dan balancing. Sementara tree baik untuk hierarki dan pencarian cepat, grafik lebih baik untuk hubungan kompleks. Sangat penting untuk memahami kedua struktur ini untuk efisiensi komputasi.

## E. REFERENSI

Asisten Praktikum, “*Modul 9 Graph dan Tree*”, learning Management System, 2024

Study With Student. “Konsep Struktur Data Graph Pada Pemrograman (Struktur Data)”.  
Diakses dari <https://youtu.be/vX-CEAmQm5I?si=i9QgbHDXS67PrjFp>

Trivusi.(2022, 16 September). *Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya*.  
Diakses pada 9 Juni 2024, dari <https://www.trivusi.web.id/2022/07/struktur-data-tree.html>