

# **LAPORAN PRAKTIKUM**

## **MODUL IX**

### **GRAF DAN TREE**



**Disusun oleh:**  
**Aryo Tegar Sukarno**  
**NIM: 2311102018**

**Dosen Pengampu:**  
**Wahyu Andi Saputra, S.Pd., M.Eng**

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

## **BAB I TUJUAN PRAKTIKUM**

- Mahasiswa diharapkan mampu memahami Graph and Tree
- Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

## BAB II

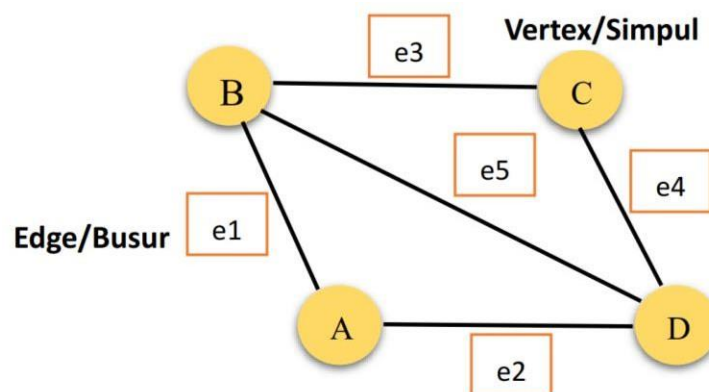
### DASAR TEORI

#### 1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde dan digambarkan seperti berikut:

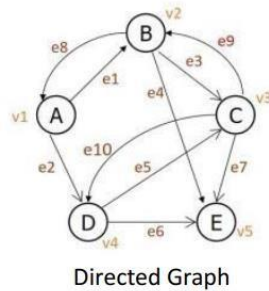


Graph juga dapat digunakan dalam berbagai bidang lainnya, seperti analisis jaringan untuk mempelajari pola koneksi dalam sistem kompleks, analisis peringkat dalam mesin pencari untuk menentukan relevansi dan otoritas halaman web. Dengan kemampuannya yang serbaguna, Graph menjadi alat yang sangat berharga dalam memahami dan menganalisis hubungan serta pola yang ada dalam berbagai konteks.

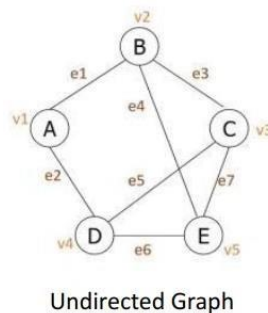
##### a. Jenis-jenis Graph

Graph memiliki berbagai jenis yang umumnya sering digunakan, antara lain:

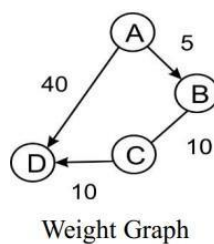
1) **Graph berarah (directed graph)**: Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.



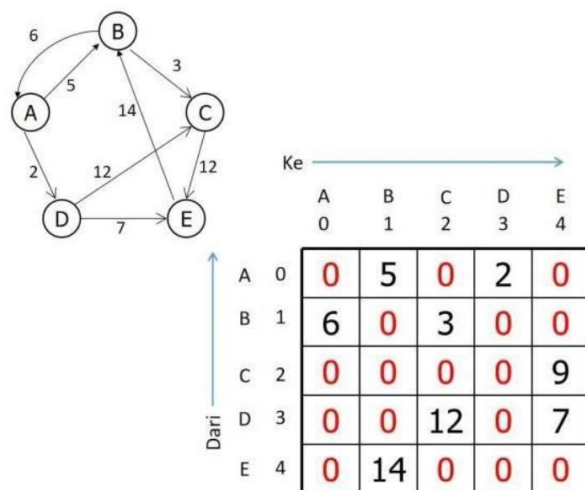
2) **Graph tak berarah (undirected graph)**: Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.



3) **Weight Graph** : Graph yang mempunyai nilai pada tiap edgenya.



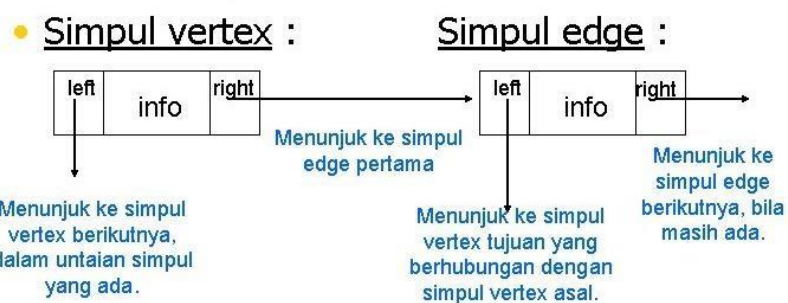
## b. Representasi Graph dengan Matriks



Gambar Representasi Graph dengan Matriks

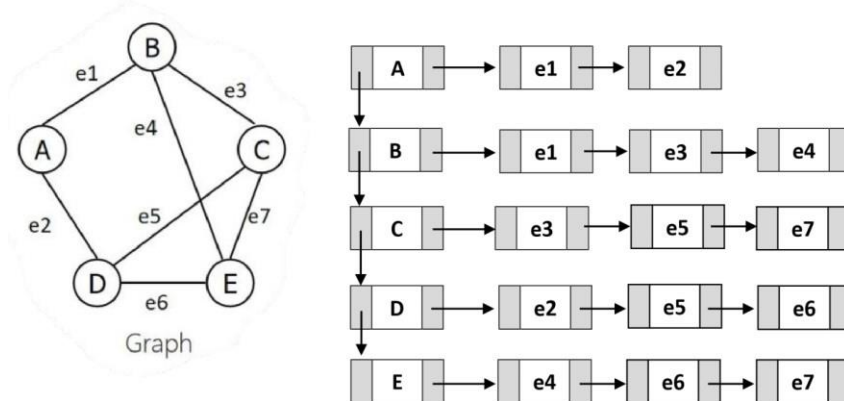
Representasi Graph dengan matriks, dikenal sebagai Matriks Adjasi, menggunakan matriks dua dimensi. Representasi Graph menggunakan matriks memiliki kelebihan akses efisien, penghitungan derajat simpul yang cepat, dan kesederhanaan. Namun, kekurangannya adalah penggunaan ruang yang besar, batasan skalabilitas, dan ketidakefisienan untuk Graph yang jarang terhubung. Representasi ini cocok untuk Graph kecil dengan hubungan yang padat, tetapi mungkin tidak efisien untuk Graph besar atau jarang terhubung.

### c. Representasi Graph dengan Linked List



Dalam membuat representasi Graph dalam bentuk linked list, penting untuk memperhatikan perbedaan antara simpul vertex dan simpul edge. Simpul vertex merujuk pada simpul atau vertex dalam Graph, sedangkan simpul edge merujuk pada busur atau hubungan antar simbol. Meskipun struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, umumnya disamakan. Akan tetapi, perbedaan

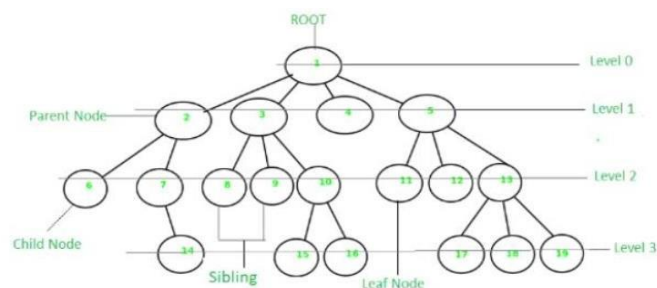
antara simpul vertex dan simpul edge terletak pada asumsi dan fungsi yang melekat pada masing-masing simpul.



Gambar Representasi Graph dengan Linked List

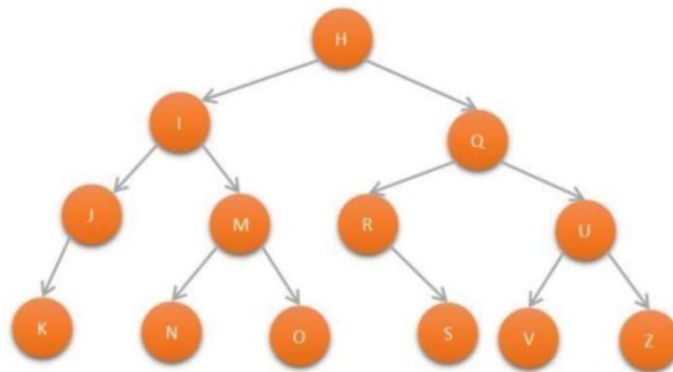
## 2. Tree atau Pohon

Dalam ilmu komputer, pohon merupakan struktur data yang umum dan kuat yang menyerupai pohon dalam kehidupan nyata. Struktur pohon terdiri dari kumpulan node yang saling terhubung, di mana setiap node memiliki paling banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data pohon digunakan untuk menyimpan data secara hierarkis, seperti pohon keluarga, skema pertandingan, dan struktur organisasi. Istilah-istilah yang terkait dengan struktur data pohon meliputi:



<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

Binary Tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

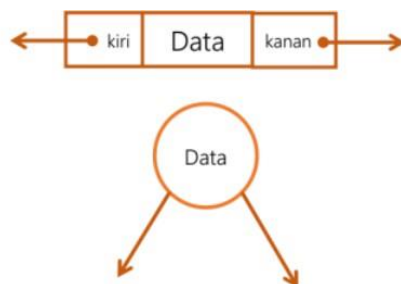


*Gambar Struktur Data Binary Tree*

Untuk membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```

struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
  
```



### 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. Retrive:** 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 lenght-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

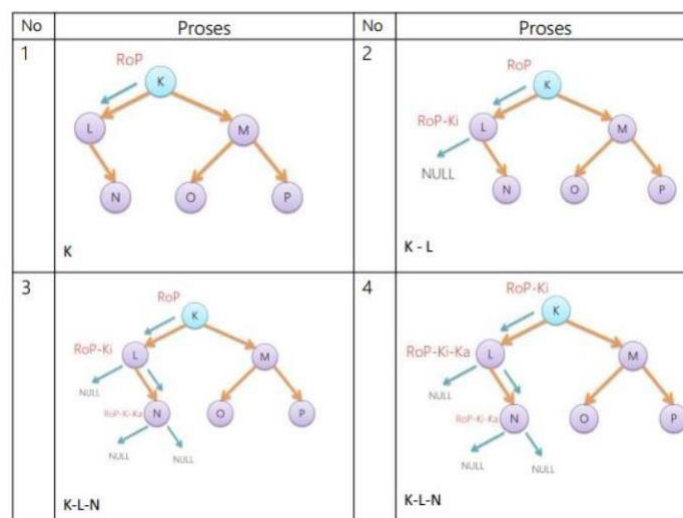
Penelusuran secara pre-order memiliki alur:

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

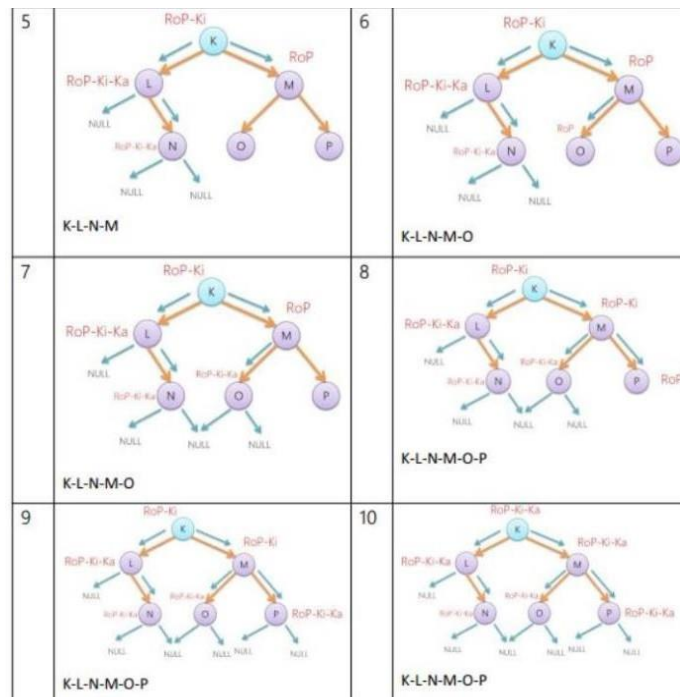
Root (print) - Kiri - Kanan

RoP - Ki - Ka

## Alur Pre-Order



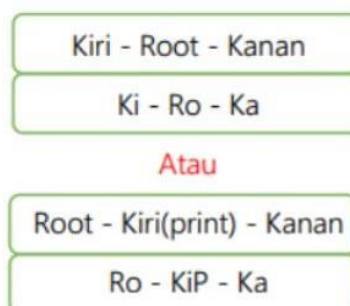




## 2. In-Order

Penelusuran secara in-order memiliki alur:

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



### 3. Post-Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
- b. Secara rekursif mencetak seluruh data pada subpohon kanan
- c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



## BAB III

### GUIDED

#### 1. Guided 1

##### Source code

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

```

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

int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

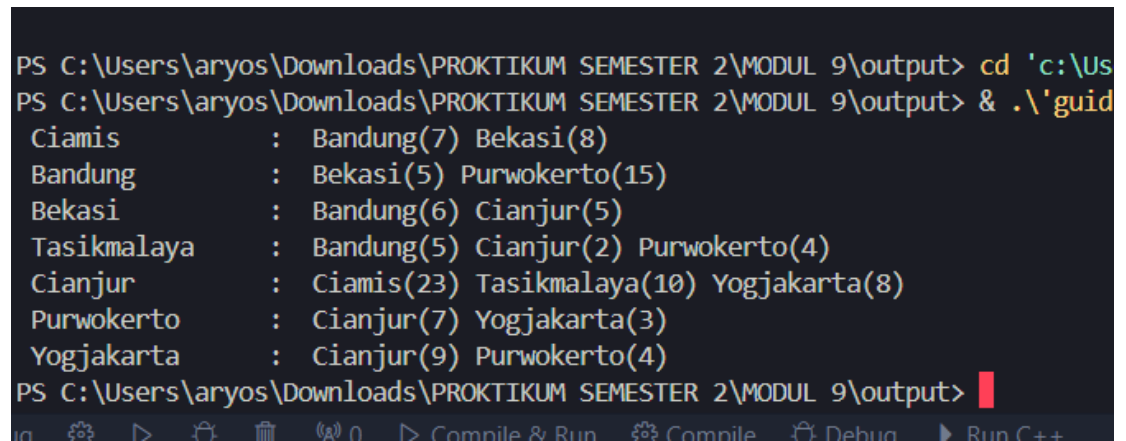
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
             << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "(" << busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}

int main()

```

```
{  
    tampilGraph();  
    return 0;  
}
```

### Screenshoot program



```
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 9\output> cd 'c:\Us  
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 9\output> & .\guid  
Ciamis      : Bandung(7) Bekasi(8)  
Bandung     : Bekasi(5) Purwokerto(15)  
Bekasi      : Bandung(6) Cianjur(5)  
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)  
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)  
Purwokerto  : Cianjur(7) Yogyakarta(3)  
Yogyakarta  : Cianjur(9) Purwokerto(4)  
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 9\output> 
```

### DESKRIPSI PROGRAM

## 1. Guided

### 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;
    else
        return 0;
    // true
    // 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()

```

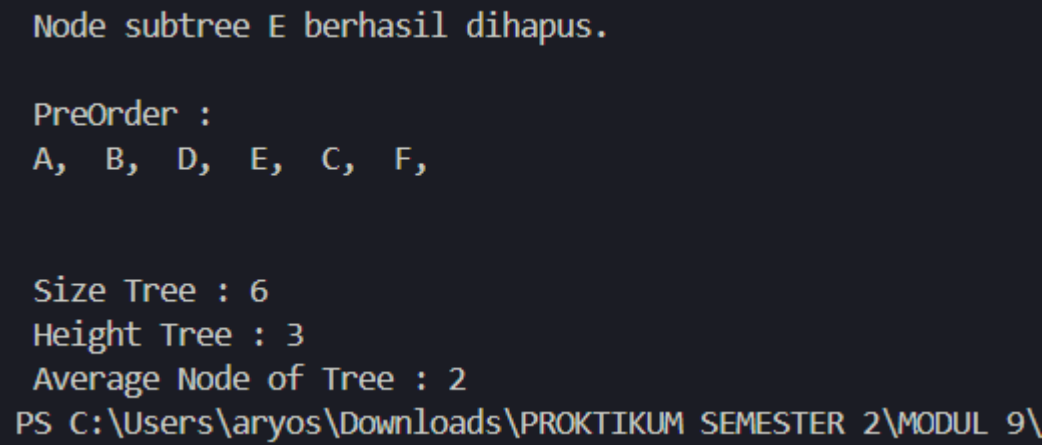
```

{
    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();  
}
```

### Screenshoot program



```
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\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 9\
```

### Deskripsi Program

Program tersebut adalah implementasi dari pohon biner, program ini memungkinkan pembuatan, pemodifikasi, dan penelusuran pohon biner.



## LATIHAN KELAS - UNGUIDED

### 1. Unguided 1

#### Source code

```
#include <iostream>

#include <vector>

#include <map>

using namespace std;

typedef map<string, int> DistanceMap; typedef
map<string, DistanceMap> Graph;

void addEdge(Graph &graph, const string &source, const string
&destination, int distance)
{
    graph[source][destination] = distance;
}

void printGraph(const Graph &graph)
{
    cout << " ";
```

```

        for (const auto &pair : graph)
        {
            cout << pair.first << "
                                   ";
        }
        cout << endl;

        for (const auto &pair :
              graph)
        {
            cout << pair.first;
            cout.width(20 - pair.first.length()); cout << " ";
            for (const auto &innerPair : pair.second)
            {
                cout << innerPair.second; cout.width(10);
            }
            cout << endl;
        }
    }

int main()
{
    int numNodes_2211102171;
    cout << "Silahkan masukkan jumlah simpul: "; cin
    >> numNodes_2211102171;

    vector<string> nodes(numNodes_2211102171);
    cout << "Silahkan masukkan nama simpul:\n"; for
    (int i = 0; i < numNodes_2211102171; ++i)

```

```

    {
        cout << "Simpul " << i + 1 << ": "; cin >>
        nodes[i];
    }

    Graph graph;
    cout << "Silahkan masukkan bobot antar simpul\n"; for (int
    i = 0; i < numNodes_2211102171; ++i)
    {
        for (int j = 0; j < numNodes_2211102171; ++j)
        {
            string source = nodes[i]; string
            destination = nodes[j]; int distance;
            cout << source << " --> " << destination << " =
";
            cin >> distance;
            addEdge(graph, source, destination, distance);
        }
    }

    printGraph(graph);
    return 0;
}

int main()
{
    Tree tree;

```

```

while (true)
{
    cout << "Menu:\n";
    cout << "1. Tambah Node\n";
    cout << "2. Cetak Tree\n"; cout
    << "3. Keluar\n";
    cout << "Pilih menu (1-3): ";

    int choice; cin
    >> choice;

    if (choice == 1)
    {
        string parent, child;
        cout << "Masukkan nama parent node: "; cin
        >> parent;
        cout << "Masukkan nama child node: "; cin
        >> child;
        addNode(tree, parent, child);
        cout << "Node berhasil ditambahkan.\n";
    }
    else if (choice == 2)
    {
        cout << "Cetak Tree:\n"; if
        (tree.empty())
        {
            cout << "Tree kosong.\n";

```

```

        }
        else
        {
            string root;
            cout << "Masukkan nama root node: "; cin
            >> root;
            printTree(tree, root);
        }
    }
    else if (choice == 3)
    {
        cout << "Terima kasih, program selesai.\n"; break;
    }
    else
    {
        cout << "Menu tidak valid. Silakan pilih menu 1-
3.\n";
    }

    cout << endl;
}

return 0;
}

```

## Screenshoot program

```
PS C:\Users\ACER> & 'c:\Users\ACER\.vscode\extensions\ms-vscode.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-2dkn2u1p.w2v' '--stdout=Microsoft-MIEngine-Out-1tgn2sm2.2gz' '--stderr=Microsoft-MIEngine-Error-1jho0gry.eqb' '--pid=Microsoft-MIEngine-Pid-dwfb yvuo.45h' '--dbgExe=C:\Not System\College\MinGW\bin\gdb.exe' '--interpreter=mi'
Silahkan masukkan jumlah simpul: 3
Silahkan masukkan nama simpul:
Simpul 1: Brebes
Simpul 2: Purwokerto
Simpul 3: Magelang
Silahkan masukkan bobot antar simpul
Brebes --> Brebes = 0
Brebes --> Purwokerto = 1
Brebes --> Magelang = 4
Purwokerto --> Brebes = 1
Purwokerto --> Purwokerto = 0
Purwokerto --> Magelang = 3
Magelang --> Brebes = 4
Magelang --> Purwokerto = 3
Magelang --> Magelang = 0
Brebes Magelang Purwokerto
Brebes      0      4      1
Magelang    4      0      3
Purwokerto  1      3      0
PS C:\Users\ACER> 
```

## Deskripsi Program

Program tersebut adalah implementasi sederhana dari struktur data graf, program ini memungkinkan pengguna untuk memasukkan jumlah simpul, nama simpul, dan bobot antar simpul dalam graf. Kemudian, graf yang dibuat akan dicetak ke layar.

## 2. Unguided 2

### Source code

// Muhammad Rayhan Pratama/2211102179/IF-10-D
---

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

typedef map<string, vector<string>> Tree;

void addNode(Tree &tree, const string &parent, const string &child)
{
    tree[parent].push_back(child);
}

void printTree(const Tree &tree, const string &node, int level = 0)
{
    for (int i = 0; i < level; ++i)
    {
        cout << "  ";
    }
    cout << "|_" << node << endl;

    if (tree.count(node) > 0)
    {
        for (const auto &child : tree.at(node))
        {
            printTree(tree, child, level + 1);
        }
    }
}
```

```
int main()
{
    Tree tree;

    while (true)
    {
        cout << "Menu:\n";
        cout << "1. Tambah Node\n";
        cout << "2. Cetak Tree\n"; cout
        << "3. Keluar\n";
        cout << "Pilih menu (1-3): ";

        int choice; cin
        >> choice;

        if (choice == 1)
        {
            string parent, child;
            cout << "Masukkan nama parent node: "; cin
            >> parent;
            cout << "Masukkan nama child node: "; cin
            >> child;
            addNode(tree, parent, child);
            cout << "Node berhasil ditambahkan.\n";
        }
        else if (choice == 2)
```



```

        {
            cout << "Cetak Tree:\n"; if
            (tree.empty())
            {
                cout << "Tree kosong.\n";
            }
            else
            {
                string root;
                cout << "Masukkan nama root node: "; cin
                >> root;
                printTree(tree, root);
            }
        }
    else if (choice == 3)
    {
        cout << "Terima kasih, program selesai.\n"; break;
    }
    else
    {
        cout << "Menu tidak valid. Silakan pilih menu 1-
3.\n";
    }

    cout << endl;
}

return 0;

```

```
}
```

**Screenshoot program**

```

PS C:\Users\ACER> & 'c:\Users\ACER\.vscode\extensions\ms-vsco
de.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLa
uncher.exe' '--stdin=Microsoft-MIEngine-In-205hik05.a25' '--std
out=Microsoft-MIEngine-Out-zevpgz4g.dh5' '--stderr=Microsoft-M
IEngine-Error-okd0xodx.kg5' '--pid=Microsoft-MIEngine-Pid-kaew
0nwy.ifm' '--dbgExe=C:\Not System\College\MingW\bin\gdb.exe' '
--interpreter=mi'
Menu:
1. Tambah Node
2. Cetak Tree
3. Keluar
Pilih menu (1-3): 1
Masukkan nama parent node: A
Masukkan nama child node: B
Node berhasil ditambahkan.

Menu:
1. Tambah Node
2. Cetak Tree
3. Keluar
Pilih menu (1-3): 1
Masukkan nama parent node: B
Masukkan nama child node: C
Node berhasil ditambahkan.

Menu:
1. Tambah Node
2. Cetak Tree
3. Keluar
Pilih menu (1-3): 1
Masukkan nama parent node: B
Masukkan nama child node: D
Node berhasil ditambahkan.

```

## Deskripsi Program

Program tersebut adalah implementasi sederhana dari struktur data pohon (tree), program ini memungkinkan pengguna untuk menambahkan simpul (node) ke dalam pohon dan mencetak pohon tersebut

## BAB IV KESIMPULAN