

LAPORAN PRAKTIKUM

MODUL 9 GRAPH DAN TREE



Disusun oleh:
Nofita Fitriyani
NIM: 2311102001

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

- a. Mahasiswa diharapkan mampu memahami graph dan tree
- b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II

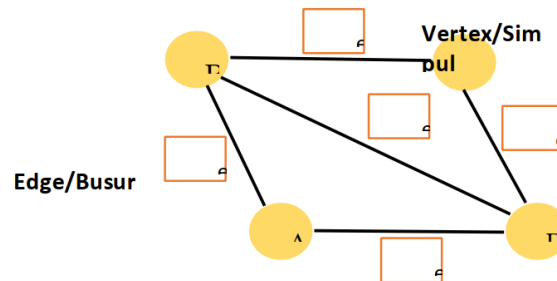
DASAR TEORI

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 sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

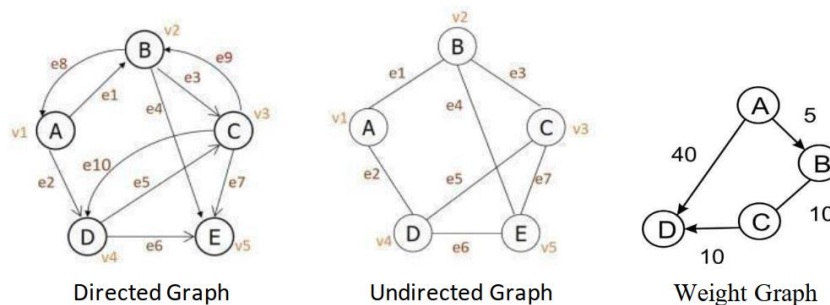
Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge. Dapat digambarkan:



Gambar 1 Contoh Graph

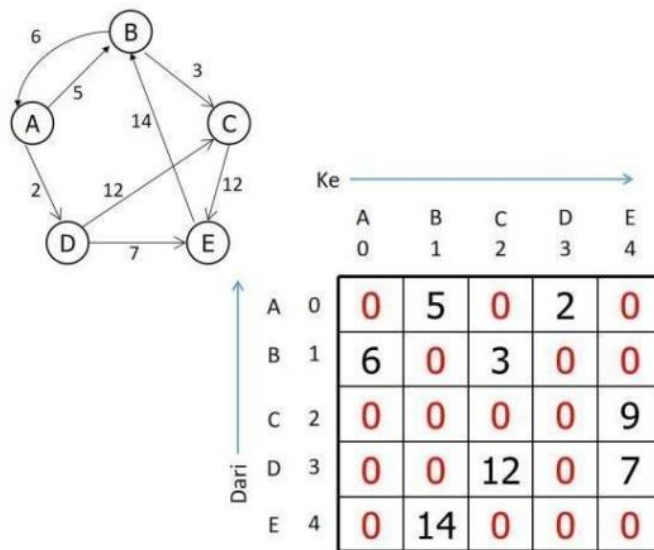
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph



- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
- Weight Graph :** Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks



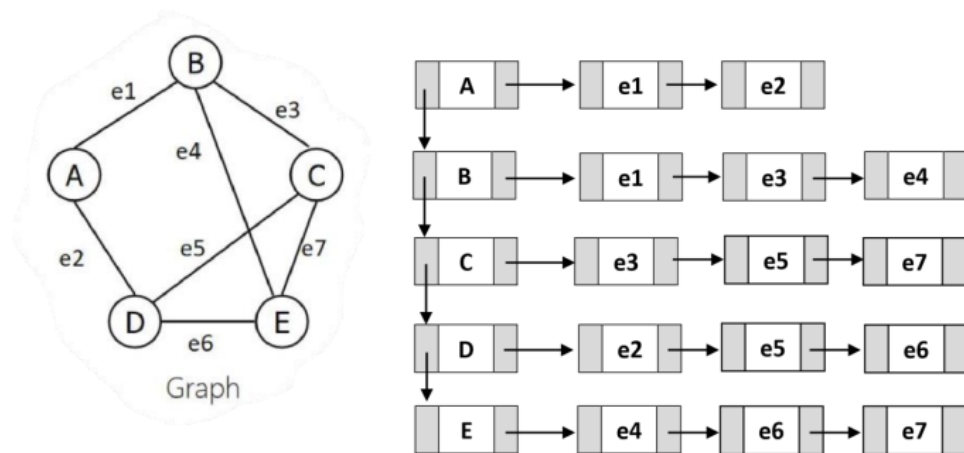
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

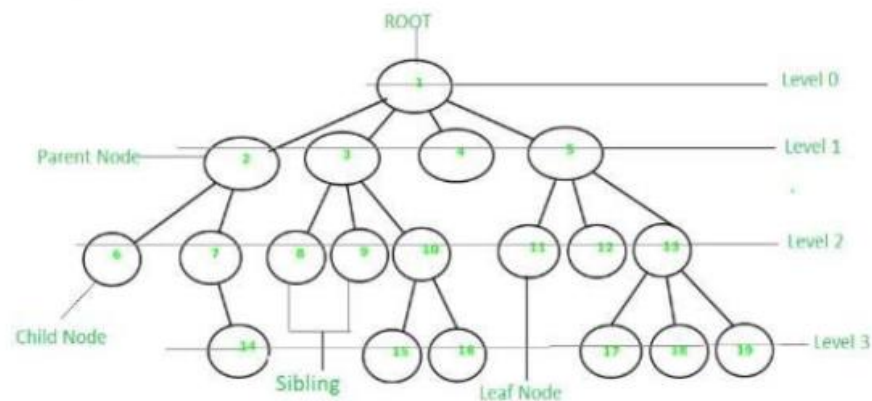
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

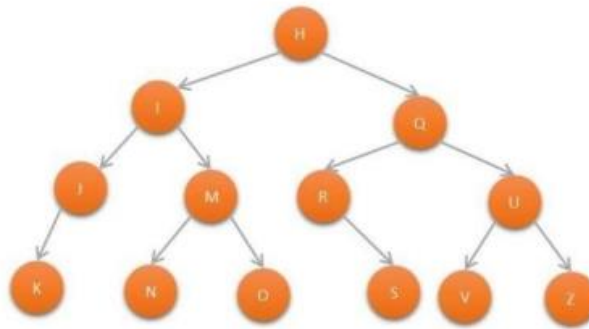
Dalam ilmu komputer, pohon/tree adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, dimana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



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

Tabel 1 Terminologi dalam Struktur Data Tree

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

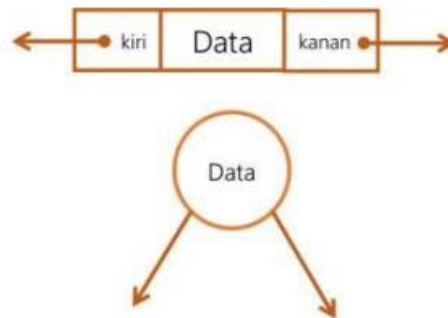


Gambar 1. Struktur Data Binary Tree

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;
  
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

- Create:** digunakan untuk membentuk binary tree baru yang masih kosong.
- Clear:** digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- isEmpty:** digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- Insert:** digunakan untuk memasukkan sebuah node kedalam tree.
- 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:

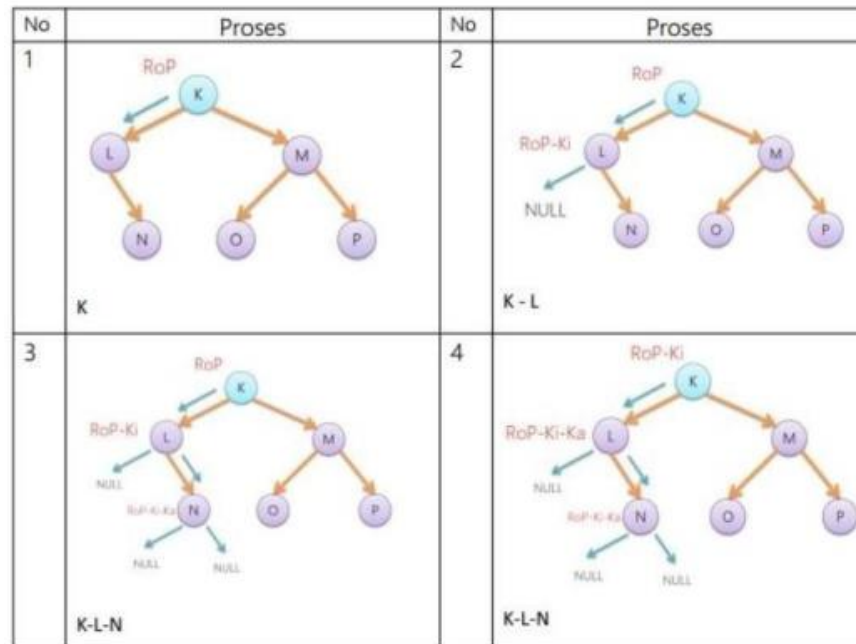
- a. Cetak data pada simpul root
- b. Secara rekursif mencetak seluruh data pada subpohon kiri
- c. Secara rekursif mencetak seluruh data pada subpohon kanan.

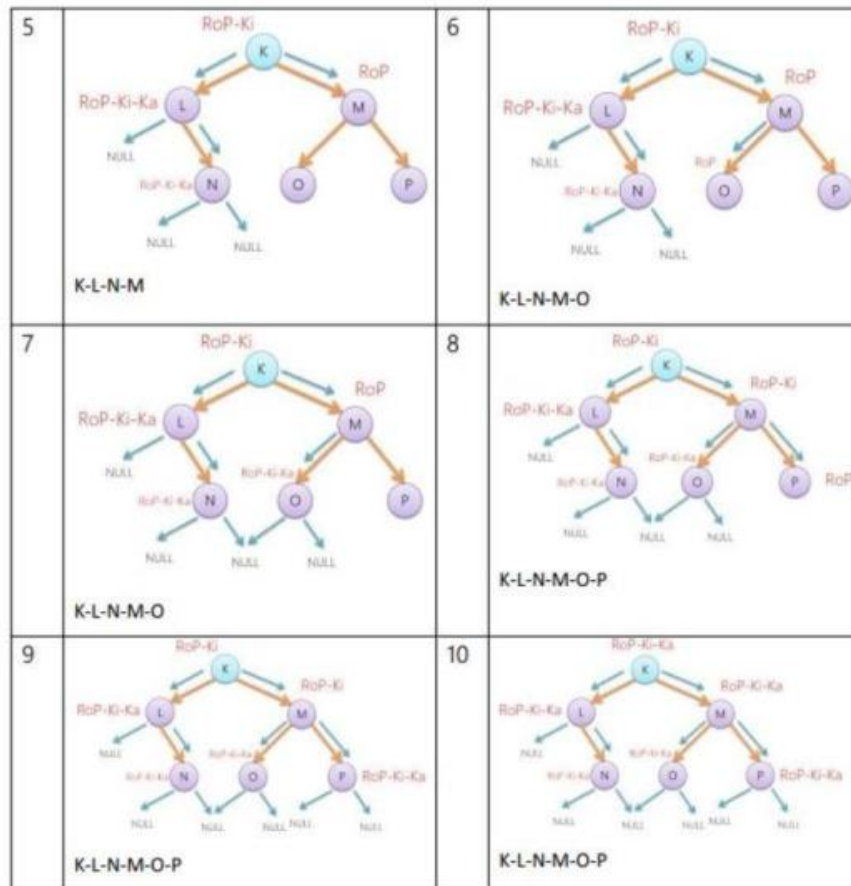
Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Alur Pre-Order



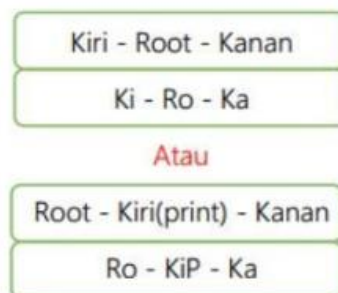


2. In-Order

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:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan
- Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Kanan - Root

Ki - Ka - Ro

Atau

Root - Kiri - Kanan(print)

Ro - Ki - KaP

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\LENOVO> cd "c:\Praktikum Struktur Data\Modul 9\" ; if
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:\Praktikum Struktur Data\Modul 9>

```

Deskripsi program

Program tersebut menampilkan graf berarah yang direpresentasikan dengan matriks ketetanggaan (adjacency matrix). Terdapat dua array utama dalam program ini: simpul yang berisi nama-nama simpul (node) dalam graf dan busur yang merupakan matriks dua dimensi yang menyimpan bobot (weight) dari setiap busur (edge) antara simpul-simpul tersebut. Matriks busur berukuran 7x7, di mana

elemen-elemen non-nol menunjukkan adanya busur dari simpul baris ke simpul kolom dengan bobot tertentu. Fungsi tampilGraph digunakan untuk mencetak graf tersebut ke layar dengan format yang mudah dibaca. Dalam fungsi ini, program mengiterasi setiap baris dari matriks busur, dan jika ditemukan nilai yang bukan nol, program akan mencetak simpul tujuan serta bobot dari busur yang menghubungkan kedua simpul tersebut. Fungsi main hanya memanggil tampilGraph untuk menampilkan graf ketika program dijalankan.

2. Guided 2

Source code

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

using namespace std;

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

Pohon *root, *baru;

void init()
{
    root = NULL;
}

bool isEmpty()
{
    return root == NULL;
}
```

```

void buatNode(char data)
{
    if (isEmpty())
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat sebagai
root."
                << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\n Node " << node->data << " sudah ada child
kiri !" << endl;
            return NULL;

```

```

    }
    else
    {
        Pohon *baru = new Pohon();
        baru->data = data;
        baru->left = NULL;
        baru->right = NULL;
        baru->parent = node;
        node->left = baru;
        cout << "\n Node " << data << " berhasil ditambahkan
kechild kiri " << baru->parent->data << endl;
        return baru;
    }
}

Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\n Node " << node->data << " sudah ada child
kanan !" << endl;
            return NULL;
        }
        else
        {
            Pohon *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;
    }
}

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

```

```

}

void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

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

```

```

}

void clear()
{
    if (isEmpty())
    {
        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 (isEmpty())
    {
        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 (isEmpty())
    {
        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 characteristic()

```



```

{
    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;

        characteristic();
        deleteSub(nodeE);
        cout << "\n PreOrder :" << endl;
        preOrder();
        cout << "\n" << endl;

        characteristic();
    }

```

Screenshot program

```

Node A berhasil dibuat sebagai root.

Node B berhasil ditambahkan kechild kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan kechild kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan kechild kiri C

Node G berhasil ditambahkan kechild kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan kechild kiri G

Node J berhasil ditambahkan ke child kanan 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 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 C:\Praktikum Struktur Data\Modul 9> █
```

Deskripsi program

Program ini merupakan implementasi pohon biner yang mendukung operasi pembuatan pohon, penambahan node, pengubahan nilai node, penelusuran (pre-

order, in-order, dan post-order), penghapusan subtree, serta analisis karakteristik pohon seperti ukuran dan tinggi. Struktur Pohon mendefinisikan node dengan data karakter dan pointer ke anak kiri, anak kanan, serta parent. Fungsi-fungsi utama meliputi inisialisasi, penambahan node kiri dan kanan, pengubahan dan pengambilan data node, pencarian informasi node, serta penelusuran pohon. Program juga dapat menghapus seluruh pohon atau subtree tertentu dan menghitung ukuran serta tinggi pohon untuk analisis lebih lanjut. Dalam fungsi main(), pohon dibangun dengan berbagai node, diubah, ditelusuri, dianalisis, dan dimodifikasi untuk mendemonstrasikan berbagai operasi yang didukung.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

int Nofita_2311102001; // Variabel dengan NIM

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

    vector<string> simpul(jumlahSimpul);
    vector<vector<int>>> busur(jumlahSimpul,
vector<int>(jumlahSimpul, 0));

    // Memasukkan nama-nama simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    // Memasukkan bobot antar simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }
}
```

```

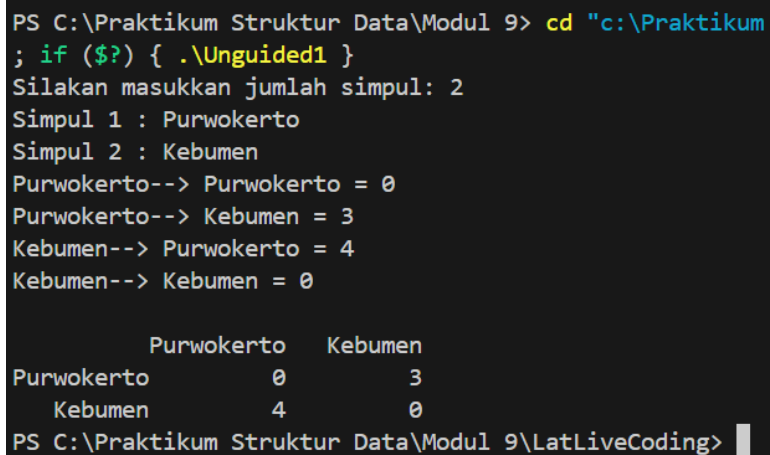
// Menampilkan matriks ketetanggaan
cout << endl;
cout << setw(10) << " ";
for (int i = 0; i < jumlahSimpul; i++) {
    cout << setw(10) << simpul[i];
}
cout << endl;

for (int i = 0; i < jumlahSimpul; i++) {
    cout << setw(10) << simpul[i];
    for (int j = 0; j < jumlahSimpul; j++) {
        cout << setw(10) << busur[i][j];
    }
    cout << endl;
}

return 0;
}

```

Screenshoot program



```

PS C:\Praktikum Struktur Data\Modul 9> cd "c:\Praktikum
; if ($?) { .\Unguided1 }
Silakan masukkan jumlah simpul: 2
Simpul 1 : Purwokerto
Simpul 2 : Kebumen
Purwokerto--> Purwokerto = 0
Purwokerto--> Kebumen = 3
Kebumen--> Purwokerto = 4
Kebumen--> Kebumen = 0

          Purwokerto   Kebumen
Purwokerto      0       3
Kebumen         4       0
PS C:\Praktikum Struktur Data\Modul 9\LatLiveCoding>

```

Deskripsi program

Program ini merepresentasikan graf dengan simpul dan bobotnya berdasarkan input dari pengguna. Program dimulai dengan meminta pengguna untuk

memasukkan jumlah simpul (node) yang ada dalam graf. Selanjutnya, pengguna diminta untuk memberikan nama masing-masing simpul. Setelah itu, pengguna diminta untuk mengisi bobot antar simpul, yang menunjukkan jarak atau koneksi antara dua simpul tertentu. Informasi yang dimasukkan pengguna disimpan dalam dua vektor: satu untuk menyimpan nama simpul dan satu lagi untuk menyimpan matriks ketetanggaan (adjacency matrix) yang menyimpan bobot antar simpul. Setelah semua data dimasukkan, program menampilkan matriks ketetanggaan tersebut dalam bentuk tabel yang rapi, di mana baris dan kolomnya merepresentasikan simpul, dan elemen-elemennya menunjukkan bobot atau jarak antar simpul. Program ini membantu dalam visualisasi graf untuk memahami hubungan dan jarak antar simpul dengan jelas.

2. Unguided 2

Source code

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

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

Pohon *root;

void init() {
    root = NULL;
}

bool isEmpty() {
    return root == 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 = 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 ada child
kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
        }
    }
}

```



```

        cout << "\nNode " << data << " berhasil ditambahkan
ke child kiri " << 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 ada child
kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan
ke child kanan " << node->data << endl;
            return baru;
        }
    }
}

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

```

```

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

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

void displayChild(Pohon *node) {
    if (node != NULL) {
        cout << "Parent: " << node->data << endl;
        if (node->left != NULL) {
            cout << "Child Kiri: " << node->left->data << endl;
        } else {
            cout << "Child Kiri: (tidak punya child kiri)" <<
endl;
        }
        if (node->right != NULL) {
            cout << "Child Kanan: " << node->right->data << endl;
        } else {
            cout << "Child Kanan: (tidak punya child kanan)" <<
endl;
        }
    }
}

```

```

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

Pohon *findNode(Pohon *node, char data) {
    if (node == NULL) return NULL;
    if (node->data == data) return node;
    Pohon *leftResult = findNode(node->left, data);
    if (leftResult != NULL) return leftResult;
    return findNode(node->right, data);
}

int main() {
    int choice;
    char data;

```

```

init();

while (true) {
    cout << "\nMenu:\n";
    cout << "1. Buat Node Root\n";
    cout << "2. Tambah Node Kiri\n";
    cout << "3. Tambah Node Kanan\n";
    cout << "4. Tampilkan PreOrder\n";
    cout << "5. Tampilkan InOrder\n";
    cout << "6. Tampilkan PostOrder\n";
    cout << "7. Tampilkan Child Node\n";
    cout << "8. Tampilkan Descendants Node\n";
    cout << "9. Keluar\n";
    cout << "Pilih opsi: ";
    cin >> choice;
    cin.ignore();
    switch (choice) {
        case 1:
            cout << "Masukkan data untuk root: ";
            cin >> data;
            buatNode(data);
            break;
        case 2: {
            cout << "Masukkan data untuk node kiri: ";
            cin >> data;
            cout << "Masukkan data parent: ";
            char parentData;
            cin >> parentData;
            Pohon *parent = findNode(root, parentData);
            if (parent != NULL) {
                insertLeft(data, parent);
            } else {
                cout << "\nParent tidak ditemukan!" << endl;
            }
        }
    }
}

```

```

        break;
    }
    case 3: {
        cout << "Masukkan data untuk node kanan: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        char parentData;
        cin >> parentData;
        Pohon *parent = findNode(root, parentData);
        if (parent != NULL) {
            insertRight(data, parent);
        } else {
            cout << "\nParent tidak ditemukan!" << endl;
        }
        break;
    }
    case 4:
        cout << "\nPreOrder: ";
        preOrder(root);
        cout << "\n";
        break;
    case 5:
        cout << "\nInOrder: ";
        inOrder(root);
        cout << "\n";
        break;
    case 6:
        cout << "\nPostOrder: ";
        postOrder(root);
        cout << "\n";
        break;
    case 7:
        cout << "Masukkan data node untuk menampilkan
child: ";

```

```

        cin >> data;
        {
            Pohon *node = findNode(root, data);
            if (node != NULL) {
                displayChild(node);
            } else {
                cout << "\nNode tidak ditemukan!" <<
endl;
            }
        }
        break;
    case 8:
        cout << "Masukkan data node untuk menampilkan
descendants: ";
        cin >> data;
        {
            Pohon *node = findNode(root, data);
            if (node != NULL) {
                displayDescendants(node);
            } else {
                cout << "\nNode tidak ditemukan!" <<
endl;
            }
        }
        break;
    case 9:
        return 0;
    default:
        cout << "Opsi tidak valid! Coba lagi." << endl;
    }
}

return 0;
}

```

Screenshot program

```
PS C:\Praktikum Struktur Data\Modul 9> cd ..
Unguided2 }
```

Menu:

1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan PreOrder
5. Tampilkan InOrder
6. Tampilkan PostOrder
7. Tampilkan Child Node
8. Tampilkan Descendants Node
9. Keluar

Pilih opsi: 1

Masukkan data untuk root: N

Node N berhasil dibuat menjadi root.

Pilih opsi: 2

Masukkan data untuk node kiri: O

Masukkan data parent: N

Node O berhasil ditambahkan ke child kiri N

Pilih opsi: 3

Masukkan data untuk node kanan: F

Masukkan data parent: N

Node F berhasil ditambahkan ke child kanan N

Pilih opsi: 4

PreOrder: N, O, F,

Pilih opsi: 5

InOrder: O, N, F,

Pilih opsi: 6

PostOrder: O, F, N,

Pilih opsi: 7

Masukkan data node untuk menampilkan child: F

Parent: F

Child Kiri: (tidak punya child kiri)

Child Kanan: (tidak punya child kanan)

Pilih opsi: 8

Masukkan data node untuk menampilkan descendants: N

Descendants of N:

```
Pilih opsi: 9
```

```
PS C:\Praktikum Struktur Data\Modul 9> █
```

Deskripsi program

Program tersebut adalah implementasi pohon biner yang memungkinkan pengguna untuk membuat, mengatur, dan menampilkan node di pohon. Program tersebut memberikan berbagai pilihan untuk pengguna untuk berinteraksi dengan pohon, termasuk membuat node root, menambahkan node anak kiri dan kanan, dan menampilkan pohon dalam traversal pre-order, in-order, dan post-order. Selain itu, program tersebut memungkinkan pengguna untuk menemukan dan menampilkan node anak dan keturunan dari node tertentu. Program tersebut menggunakan struktur data antrian untuk mengunjungi pohon dan menampilkan keturunan node.

BAB IV

KESIMPULAN

Graf dan pohon adalah dua struktur data yang penting dalam ilmu komputer. Graf terdiri dari simpul (node) dan sisi (edge) yang menghubungkan simpul-simpul tersebut, dan bisa berupa graf berarah, tak berarah, atau berbobot. Graf digunakan dalam berbagai aplikasi seperti jaringan sosial dan pemetaan jalan. Pohon adalah struktur data hierarkis yang terdiri dari node, di mana setiap node memiliki satu induk dan beberapa anak. Pohon biner, yang setiap nodenya memiliki maksimal dua anak, sering digunakan untuk menyimpan data hierarkis seperti pohon keluarga atau struktur organisasi. Operasi dasar pada pohon termasuk membuat, mengosongkan, memeriksa kekosongan, menyisipkan, mencari, mengubah, menghapus subpohon, dan melakukan traversal seperti pre-order, in-order, dan post-order. Memahami graf dan pohon serta operasi-operasinya sangat penting untuk pemrograman dan pemodelan data yang efektif.