LAPORAN PRAKTIKUM

MODUL 9
GRAPH AND TREE

Disusun oleh:
Bayu Kuncoro Adi
NIM: 2311102031

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. TUJUAN PRAKTIKUM

1. Mahasiswa diharapkan mampu memahami graph dan tree.
2. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman.

BAB II

DASAR TEORI

A. DASAR TEORI

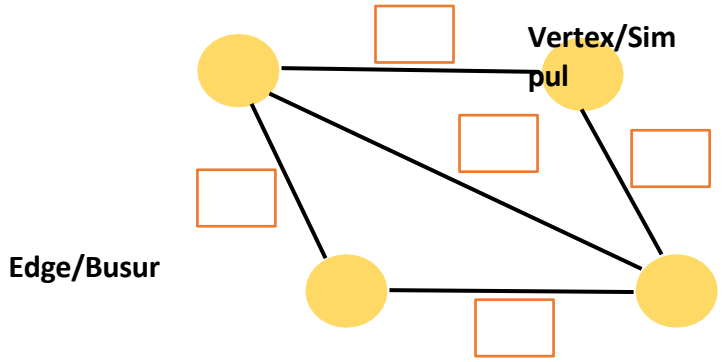
1. Graph

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.

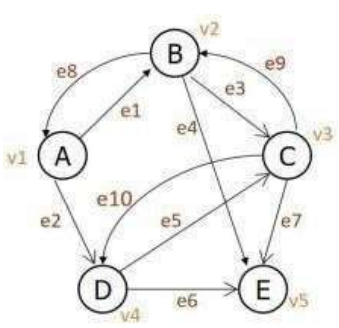
Dapat digambarkan:



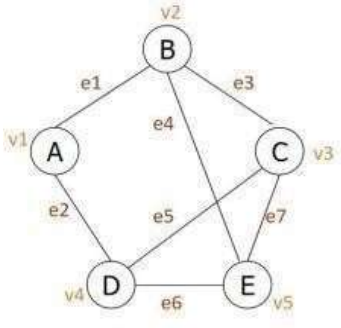
Gambar 1 Contoh Graph

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

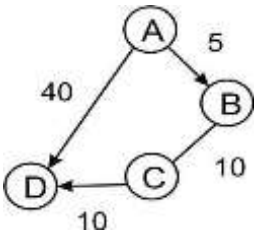
Jenis-jenis Graph



Directed Graph



Undirected Graph



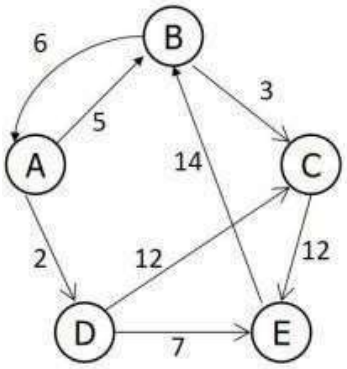
Weight Graph

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

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

c. Weight Graph : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks

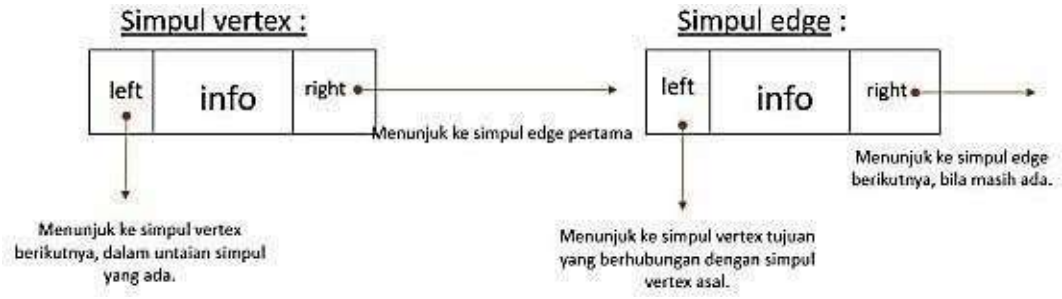


Ke →

Dari		A 0	B 1	C 2	D 3	E 4
A	0	0	5	0	2	0
B	1	6	0	3	0	0
C	2	0	0	0	0	9
D	3	0	0	12	0	7
E	4	0	14	0	0	0

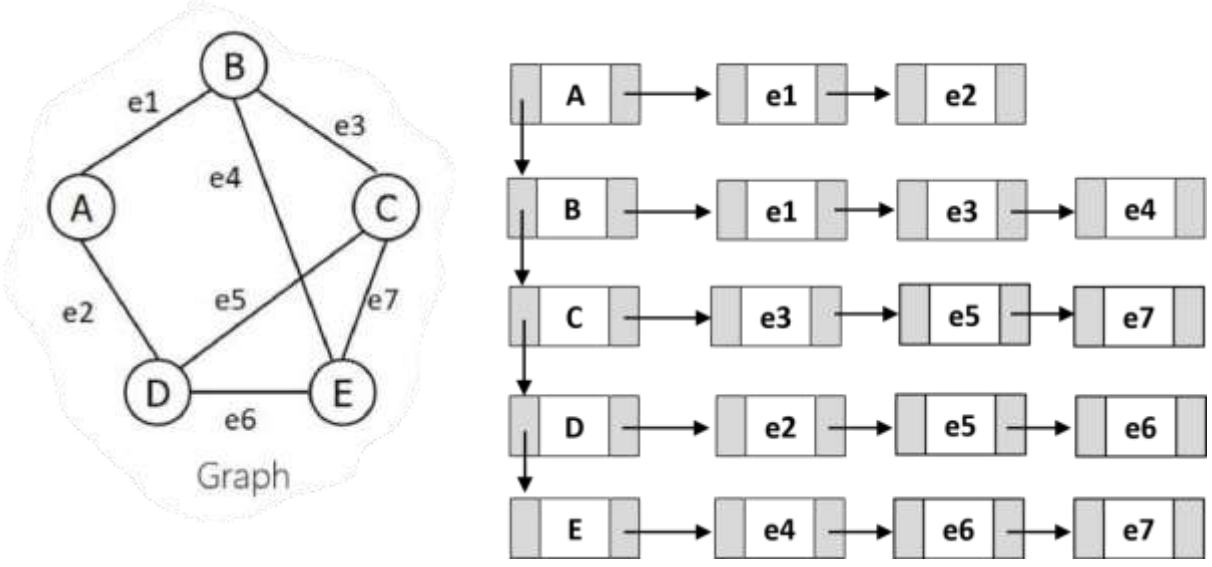
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

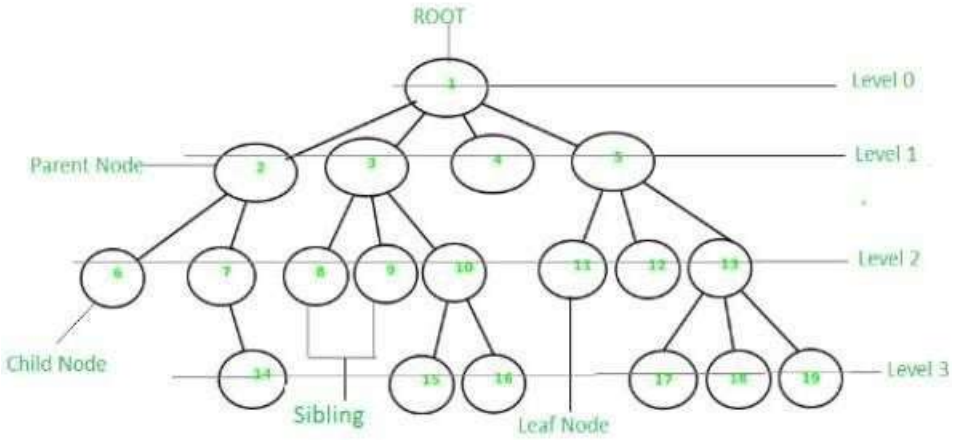
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung 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 :

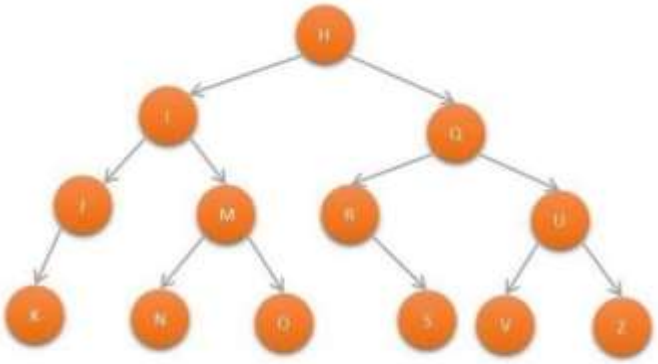


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor 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 diprasyaratkan memiliki simpul satu level di bawahnya (child)

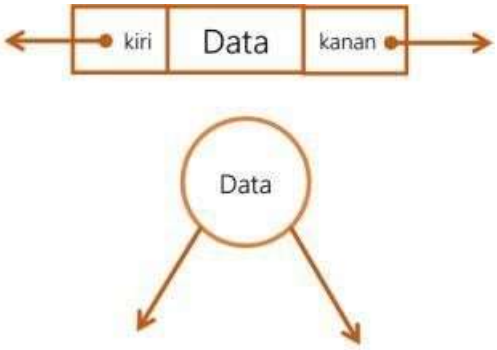
tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



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

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

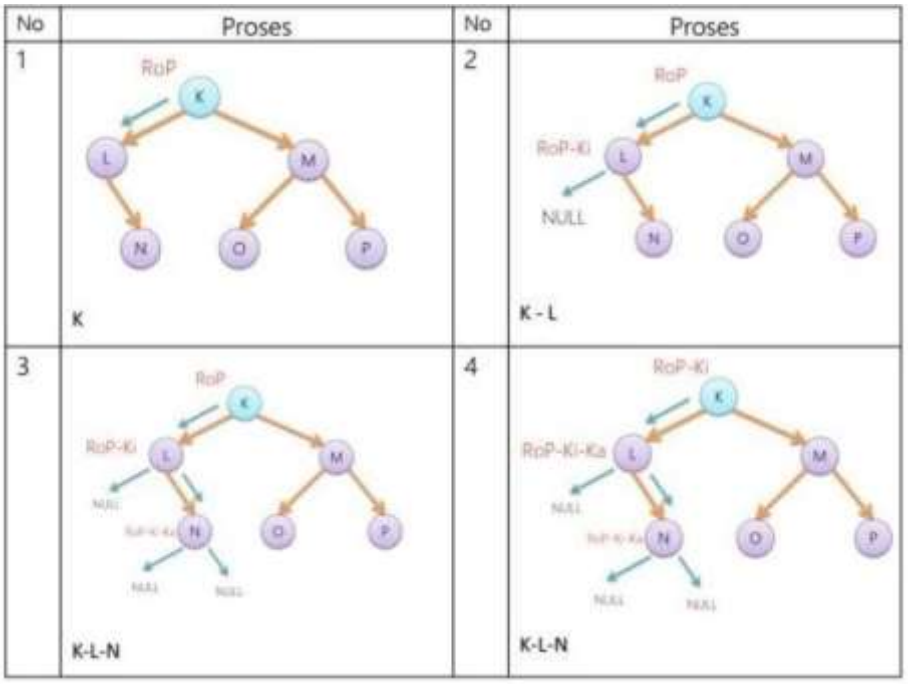
Penelusuran secara pre-order memiliki alur:

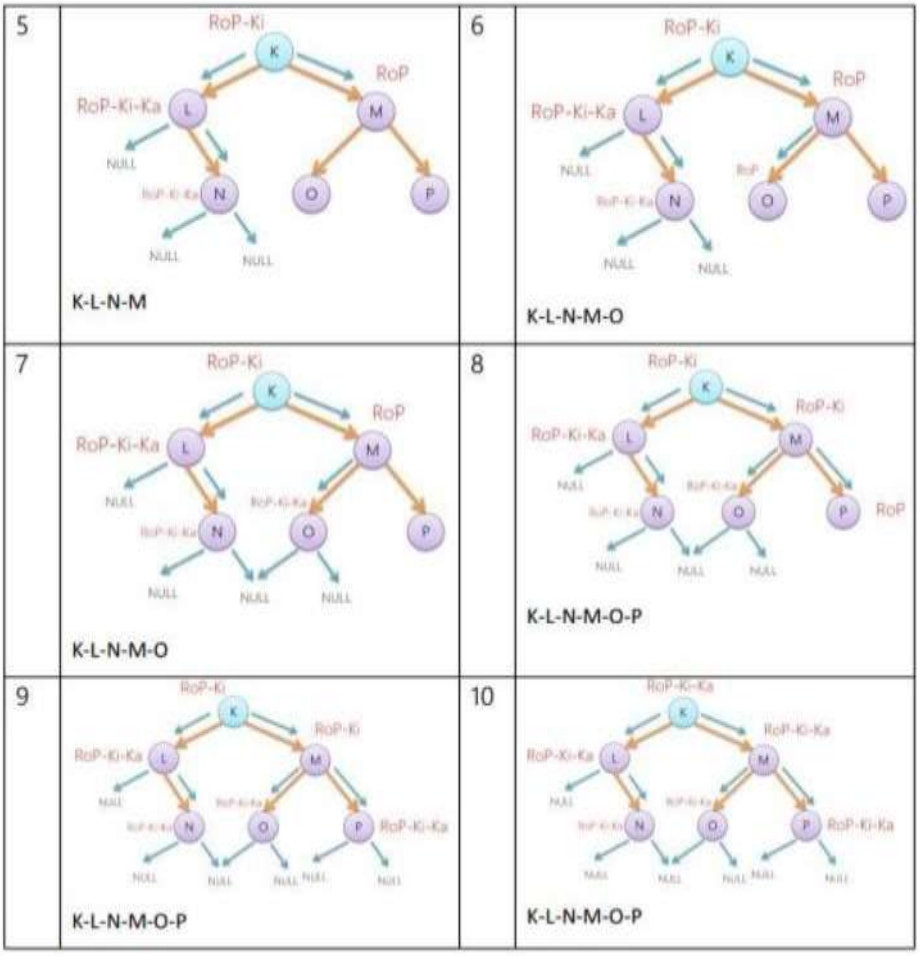
a. Cetak data pada simpul root
b. Secara rekursif mencetak seluruh data pada subpohon kiri
c. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan
RoP - Ki - Ka

Alur pre-order





3. **In-Order**

Penelusuran secara in-order memiliki alur:

1. Secara rekursif mencetak seluruh data pada subpohon kiri
2. Cetak data pada root
3. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

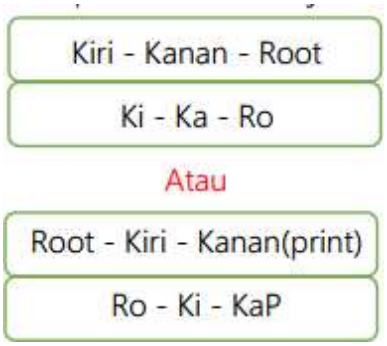
Ro - KiP - Ka

ii. Post Order

Penelusuran secara in-order memiliki alur:

1. Secara rekursif mencetak seluruh data pada subpohon kiri
2. Secara rekursif mencetak seluruh data pada subpohon kanan
3. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



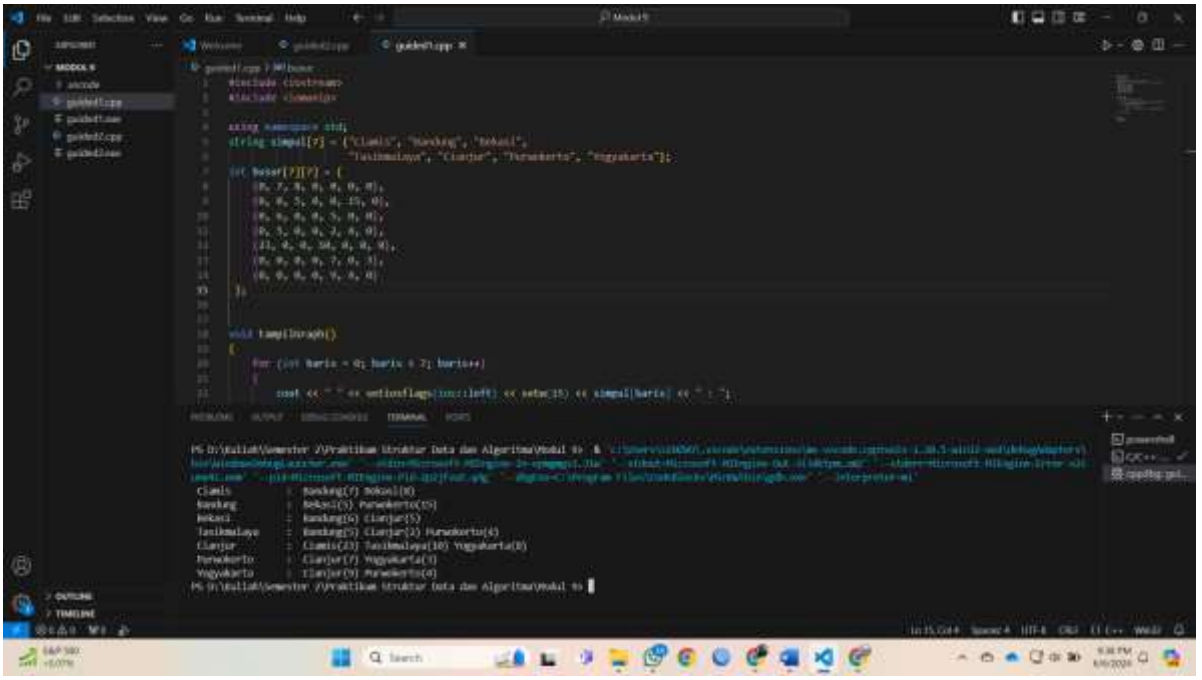
BAB III
GUIDED

1. Guided 1

Program:

```
#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:



Deskripsi Program:

Program di atas adalah implementasi sederhana dari sebuah graf berarah dalam bahasa C++. Graf ini direpresentasikan menggunakan matriks ketetanggaan (`busur`) yang menunjukkan bobot atau jarak antara simpul-simpul (node) yang diberi nama kota-kota di Indonesia seperti "Ciamis", "Bandung", "Bekasi", dan seterusnya. Fungsi `tampilGraph` digunakan untuk menampilkan graf dalam bentuk teks, di mana setiap baris menampilkan sebuah simpul diikuti oleh daftar simpul tujuan beserta bobot busur yang menghubungkannya. Dalam `main`, fungsi `tampilGraph` dipanggil untuk mencetak graf ke layar. Program ini menggunakan manipulasi I/O standar C++ seperti `setw` dan `setiosflags` untuk mengatur format output agar lebih mudah dibaca.

2. Guided 2

Program :

```
#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 << ", ";
```

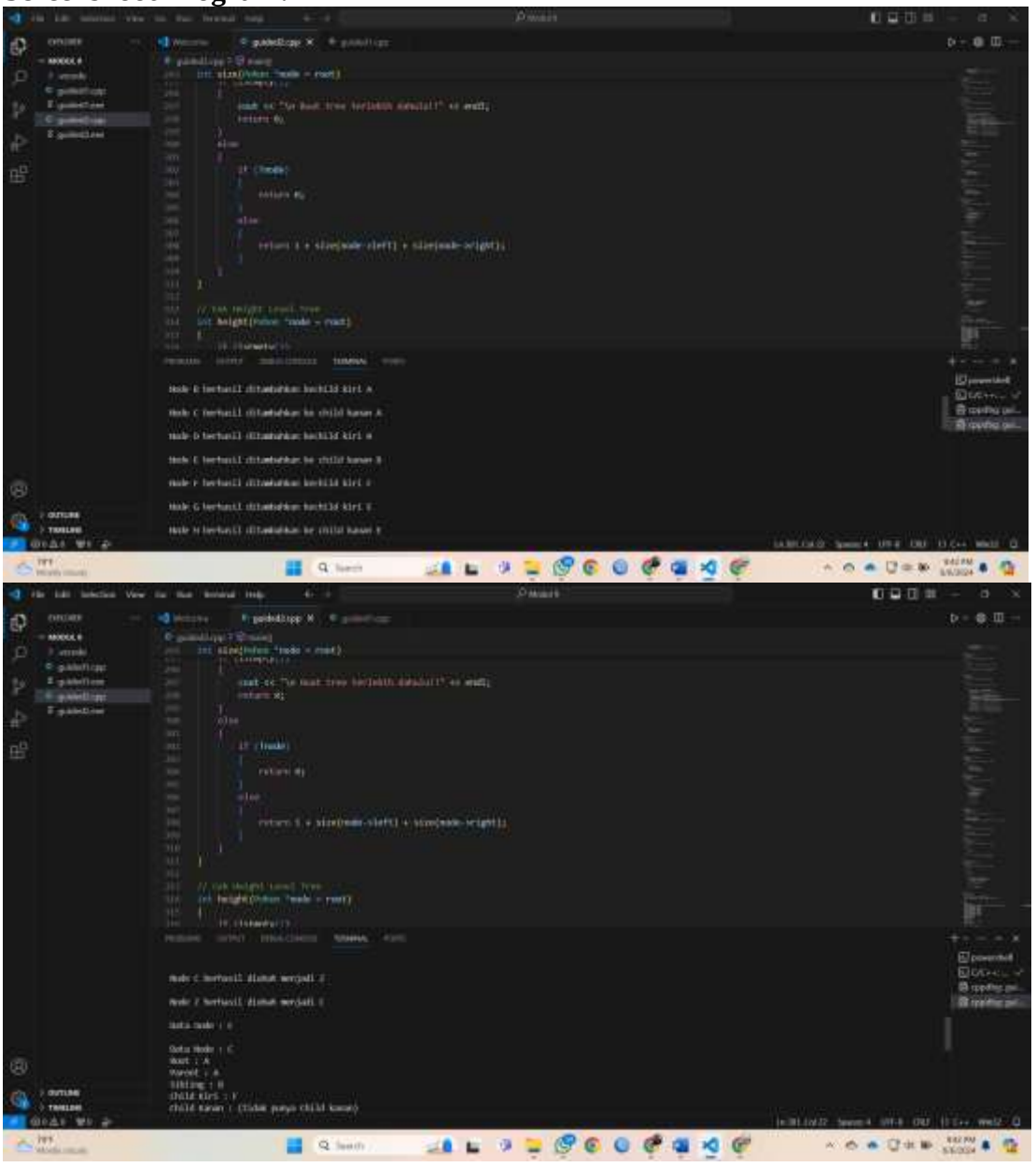
```
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// 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();
}
```

Screenshoot Program:



Deskripsi Program:

Program di atas adalah implementasi dari struktur data pohon biner dalam bahasa C++. Struktur data pohon ini terdiri dari simpul-simpul yang diimplementasikan menggunakan struktur `Pohon`, yang memiliki data, anak kiri, anak kanan, dan parent. Program ini menyediakan berbagai fungsi untuk memanipulasi pohon, termasuk membuat node baru (`buatNode`), menambah anak kiri (`insertLeft`) dan anak kanan (`insertRight`), memperbarui data node (`update`), mengambil data node (`retrieve`), menemukan informasi detail node (`find`), dan penelusuran pohon dengan metode pre-order, in-order, dan post-order. Selain itu, program juga dapat menghapus pohon secara keseluruhan (`clear`) atau sub-pohon tertentu (`deleteSub`), serta menghitung ukuran (`size`) dan tinggi pohon (`height`). Fungsi `characteristic` digunakan untuk menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Pada `main`, program ini mendemonstrasikan berbagai fungsi ini dengan membuat pohon, menambahkan beberapa

node, memperbarui node, melakukan penelusuran, menampilkan karakteristik pohon, dan menghapus sub-pohon.

LATIHAN KELAS-UNGUIDED

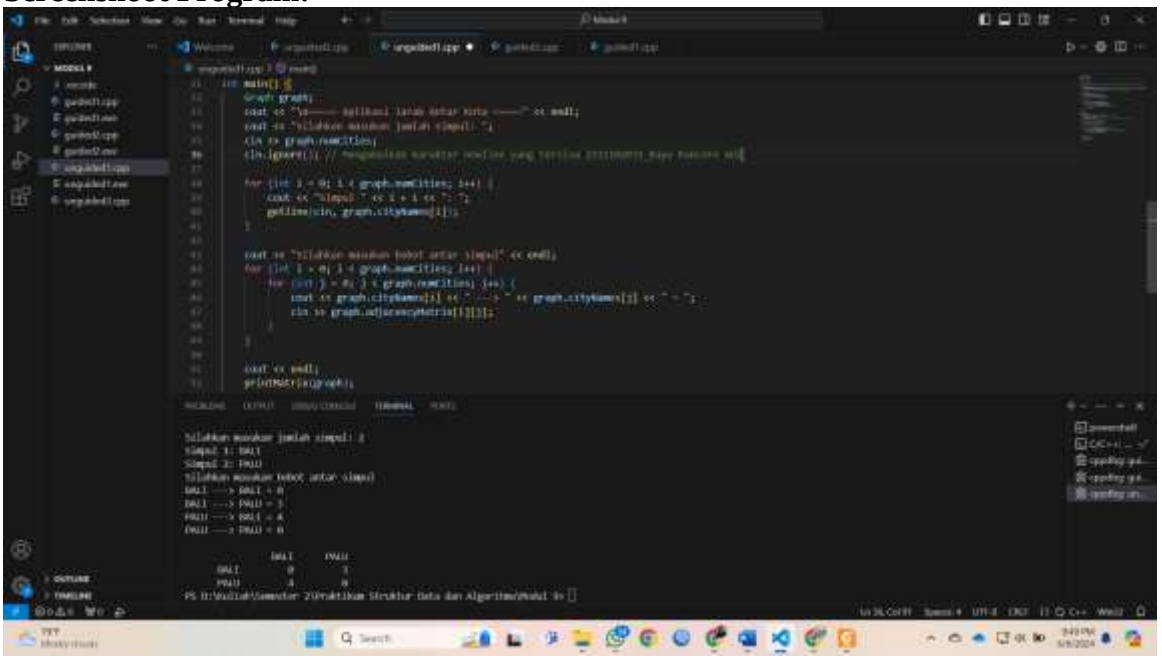
1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya. Yang memiliki output program seperti dibawah ini :

Program:

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

using namespace std;

const int MAX_CITIES = 10; //Maksimal banyak data yang dapat ditampung
pada variabel MAX_CITIES untuk struct Graph

struct Graph {
    string cityNames[MAX_CITIES]; // Banyaknya data string yang dapat
disimpan dalam string sesuai dengan kapasitas MAX_CITIES
    int adjacencyMatrix[MAX_CITIES][MAX_CITIES]; // Menggunakan array 2
dimensi, dengan ukuran maksimal sesuai MAX_CITIES
    int numCities; // Variabel yang digunakan untuk menyimpan jumlah
kota dalam graph
};

void printMatrix(const Graph& graph) {
    cout << setw(10) << " ";
    for (int i = 0; i < graph.numCities; i++) {
        cout << setw(10) << graph.cityNames[i];
    }
    cout << endl;

    for (int i = 0; i < graph.numCities; i++) {
        cout << setw(10) << graph.cityNames[i];
        for (int j = 0; j < graph.numCities; j++) {
            cout << setw(10) << graph.adjacencyMatrix[i][j];
        }
        cout << endl;
    }
}

int main() {
    Graph graph;
    cout << "\n===== Aplikasi Jarak Antar Kota =====" << endl;
    cout << "Silahkan masukan jumlah simpul: ";
    cin >> graph.numCities;
```

```
    cin.ignore(); // Mengabaikan karakter newline yang tersisa

    for (int i = 0; i < graph.numCities; i++) {
        cout << "Simpul " << i + 1 << ": ";
        getline(cin, graph.cityNames[i]);
    }

    cout << "Silahkan masukan bobot antar simpul" << endl;
    for (int i = 0; i < graph.numCities; i++) {
        for (int j = 0; j < graph.numCities; j++) {
            cout << graph.cityNames[i] << " ---> " <<
graph.cityNames[j] << " = ";
            cin >> graph.adjacencyMatrix[i][j];
        }
    }

    cout << endl;
    printMatrix(graph);

    return 0;
}
```

Screenshoot Program:



Deskripsi Program:

Program di atas adalah aplikasi sederhana dalam bahasa C++ yang memungkinkan pengguna untuk membuat graf berarah, dimana graf ini merepresentasikan jarak antar kota menggunakan matriks ketetanggaan. Program ini menggunakan struktur `Graph` yang memiliki array `cityNames` untuk menyimpan nama kota, matriks `adjacencyMatrix` untuk menyimpan bobot atau jarak antar kota, dan variabel `numCities` untuk menyimpan jumlah kota dalam graf. Dalam fungsi `main`, pengguna diminta untuk memasukkan jumlah kota, nama masing-masing kota, dan bobot atau jarak antar kota. Matriks ketetanggaan kemudian dicetak menggunakan fungsi `printMatrix`, yang menampilkan nama-nama kota di baris dan kolom pertama serta jarak antar kota di

posisi yang sesuai dalam matriks. Program ini menggabungkan penggunaan manipulasi I/O standar C++ seperti `setw` untuk mengatur lebar tampilan output agar lebih teratur dan mudah dibaca.

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!

Program:

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

struct Pohon {
    char data;
    Pohon *left, *right, *parent; // pointer
};
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 pohonnya dulu!" << 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 pohonnya dulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah ada di 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 update(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat pohonya dulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang mau diganti gak ada!!" << endl;
        else {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah menjadi " <<
data << endl;
        }
    }
}
void retrieve(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat pohonnya dulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk gak ada!" << endl;
        else {
```

```
            cout << "\nData node : " << node->data << endl;
        }
    }
}
void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat Pohonnya dulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk gak ada!" << endl;
        else {
            cout << "\nData 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;
        }
    }
}
void showChild(Pohon *node) {
    if (isEmpty()) {
        cout << "\n Buat treenya dulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk gak ada!" << endl;
        } else {
            cout << "\nNode " << node->data << " Child: " << endl;
            if (node->left)
                cout << " Child Kiri : " << node->left->data << endl;
            else
                cout << "Child Kiri : (tidak punya Child kiri)" << endl;
```

```
            if (node->right)
                cout << "Child Kanan : " << node->right->data << endl;
            else
                cout << "Child Kanan : (tidak punya Child kanan)" <<
endl;
        }
    }
}
void preOrder(Pohon *node) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

void showDescendants(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat pohonnya dulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk gak ada!" << endl;
        } else {
            cout << "\nKeturunan dari node " << node->data << " : ";
            preOrder(node);
            cout << endl;
        }
    }
}
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 deleteTree(Pohon *node) {
    if (node != NULL) {
        if (node != root) {
            if (node->parent->left == node)
                node->parent->left = NULL;
            else if (node->parent->right == node)
                node->parent->right = NULL;
```

```
        }
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root) {
            delete root;
            root = NULL;
        } else {
            delete node;
        }
    }
}
void deleteSub(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat pohonnya dulu!" << endl;
    } else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode cabang pohon " << node->data << " berhasil
dihapus." << endl;
    }
}
void clear() {
    if (isEmpty()) {
        cout << "\nBuat pohon dulu!!" << endl;
    } else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}
int size(Pohon *node) {
    if (!node) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}
int height(Pohon *node) {
    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;
        }
    }
}
void characteristic() {
```

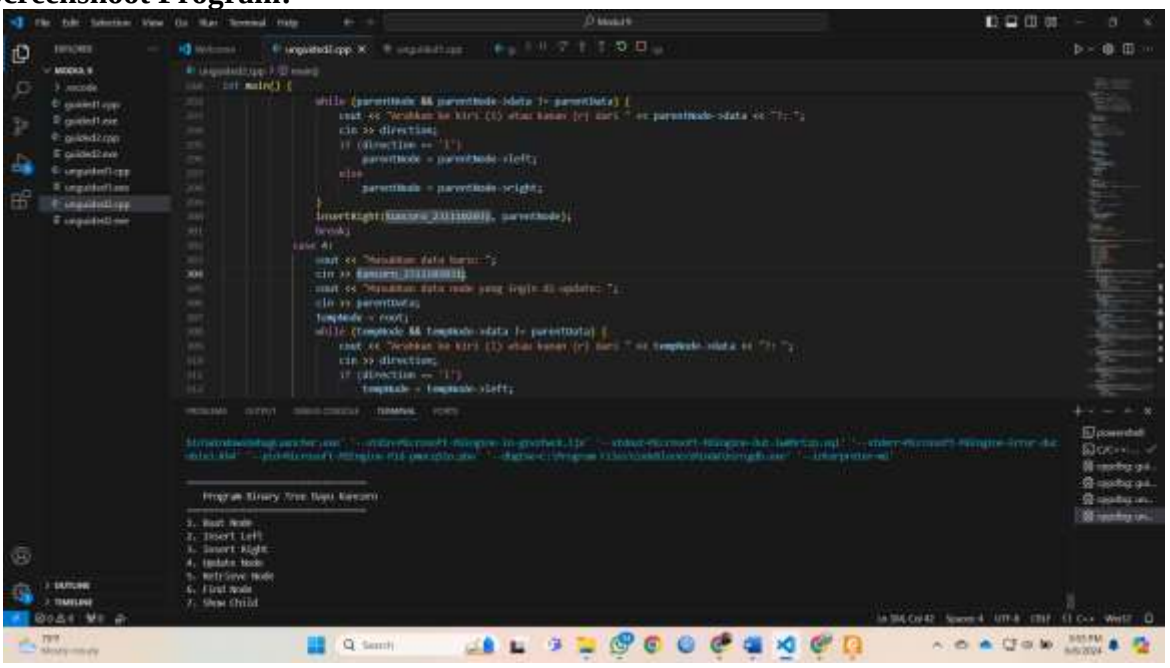
```
    int s = size(root);
    int h = height(root);
    cout << "\nUkuran Pohon : " << s << endl;
    cout << "Berat Pohon : " << h << endl;
    if (h != 0)
        cout << "Rata-rata simpul pohon : " << s / h << endl;
    else
        cout << "Rata-rata simpul pohon : 0" << endl;
}

int main() {
    int choice;
    char Kuncoro_2311102031, parentData, direction;
    Pohon *parentNode, *tempNode;

    do {
        cout << "\n================================\n";
        cout << "   Program Binary Tree Bayu Kuncoro  \n";
        cout << "================================\n";
        cout << "1. Buat Node\n";
        cout << "2. Insert Left\n";
        cout << "3. Insert Right\n";
        cout << "4. Update Node\n";
        cout << "5. Retrieve Node\n";
        cout << "6. Find Node\n";
        cout << "7. Show Child\n";
        cout << "8. Show Descendants\n";
        cout << "9. Clear Tree\n";
        cout << "10. Tree Characteristics\n";
        cout << "11. Exit\n";
        cout << "================================\n";
        cout << "Masukkan pilihan: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data: ";
                cin >> Kuncoro_2311102031;
                buatNode(Kuncoro_2311102031);
                break;
            case 2:
                cout << "Masukkan data: ";
                cin >> Kuncoro_2311102031;
                cout << "Masukkan parent data: ";
                cin >> parentData;
                parentNode = root;
                while (parentNode && parentNode->data != parentData) {
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
parentNode->data << "?: ";
                    cin >> direction;
```

```
                    if (direction == 'l')
                        parentNode = parentNode->left;
                    else
                        parentNode = parentNode->right;
                }
                insertLeft(Kuncoro_2311102031, parentNode);
                break;
            case 3:
                cout << "Masukkan data: ";
                cin >> Kuncoro_2311102031;
                cout << "Masukkan parent data: ";
                cin >> parentData;
                parentNode = root;
                while (parentNode && parentNode->data != parentData) {
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
parentNode->data << "?: ";
                    cin >> direction;
                    if (direction == 'l')
                        parentNode = parentNode->left;
                    else
                        parentNode = parentNode->right;
                }
                insertRight(Kuncoro_2311102031, parentNode);
                break;
            case 4:
                cout << "Masukkan data baru: ";
                cin >> Kuncoro_2311102031;
                cout << "Masukkan data node yang ingin di-update: ";
                cin >> parentData;
                tempNode = root;
                while (tempNode && tempNode->data != parentData) {
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
tempNode->data << "?: ";
                    cin >> direction;
                    if (direction == 'l')
                        tempNode = tempNode->left;
                    else
                        tempNode = tempNode->right;
                }
                update(Kuncoro_2311102031, tempNode);
                break;
            case 5:
                cout << "Masukkan data node yang ingin di-retrieve: ";
                cin >> Kuncoro_2311102031;
                tempNode = root;
                while (tempNode && tempNode->data != Kuncoro_2311102031)
{
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
tempNode->data << "?: ";
                    cin >> direction;
```

```
                    if (direction == 'l')
                        tempNode = tempNode->left;
                    else
                        tempNode = tempNode->right;
                }
                retrieve(tempNode);
                break;
            case 6:
                cout << "Masukkan data node yang ingin di-find: ";
                cin >> Kuncoro_2311102031;
                tempNode = root;
                while (tempNode && tempNode->data != Kuncoro_2311102031)
{
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
tempNode->data << "?: ";
                    cin >> direction;
                    if (direction == 'l')
                        tempNode = tempNode->left;
                    else
                        tempNode = tempNode->right;
                }
                find(tempNode);
                break;
            case 7:
                cout << "Masukkan data node yang ingin di-show child: ";
                cin >> Kuncoro_2311102031;
                tempNode = root;
                while (tempNode && tempNode->data != Kuncoro_2311102031)
{
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
tempNode->data << "?: ";
                    cin >> direction;
                    if (direction == 'l')
                        tempNode = tempNode->left;
                    else
                        tempNode = tempNode->right;
                }
                showChild(tempNode);
                break;
            case 8:
                cout << "Masukkan data node yang ingin di-show
descendants: ";
                cin >> Kuncoro_2311102031;
                tempNode = root;
                while (tempNode && tempNode->data != Kuncoro_2311102031)
{
                    cout << "Arahkan ke kiri (l) atau kanan (r) dari " <<
tempNode->data << "?: ";
                    cin >> direction;
                    if (direction == 'l')
```

```
                        tempNode = tempNode->left;
                    else
                        tempNode = tempNode->right;
                }
                showDescendants(tempNode);
                break;
            case 9:
                clear();
                break;
            case 10:
                characteristic();
                break;
            case 11:
                cout << "Keluar dari program." << endl;
                break;
            default:
                cout << "Pilihan tidak valid." << endl;
        }
    } while (choice != 11);

    return 0;
}
```

Screenshoot Program:



Deskripsi Program:

Program di atas adalah implementasi dari struktur data pohon biner dalam bahasa C++. Program ini menyediakan berbagai fungsi untuk mengelola pohon biner, termasuk inisialisasi pohon, penambahan node ke kiri atau kanan, pembaruan data node, pengambilan data node, pencarian node, menampilkan anak-anak node, menampilkan keturunan node, dan menghitung karakteristik pohon seperti ukuran dan tinggi pohon. Program ini juga memiliki fungsi untuk menghapus sub-pohon dan keseluruhan pohon. Di dalam fungsi `main`, terdapat menu interaktif yang memungkinkan pengguna untuk memilih operasi yang ingin dilakukan pada pohon biner dengan memasukkan pilihan

yang sesuai.

BAB IV
KESIMPULAN

Graf dan pohon adalah dua struktur data fundamental yang banyak digunakan dalam ilmu komputer untuk merepresentasikan hubungan dan hierarki antara objek-objek. Graf terdiri dari simpul (vertex) dan sisi (edge) yang menghubungkan simpul-simpul tersebut, dimana graf bisa bersifat berarah (directed graph) atau tak berarah (undirected graph). Pada graf berarah, urutan simpul dalam sebuah sisi penting, sedangkan pada graf tak berarah, urutan tersebut tidak penting. Selain itu, terdapat graf berbobot (weighted graph) dimana setiap sisi memiliki nilai atau bobot tertentu yang sering digunakan untuk merepresentasikan jarak atau biaya antara simpul-simpul.

Representasi graf dapat dilakukan dengan dua cara utama: matriks ketetanggaan dan daftar terhubung (linked list). Matriks ketetanggaan menggunakan matriks dua dimensi untuk merepresentasikan apakah ada sisi antara dua simpul dan berapa bobotnya jika ada. Sebaliknya, daftar terhubung menggunakan koleksi daftar untuk setiap simpul, di mana setiap daftar berisi simpul-simpul tetangga yang terhubung dengan sisi ke simpul tersebut. Pemahaman yang baik tentang kedua metode ini penting karena masing-masing memiliki kelebihan dan kekurangan tergantung pada kebutuhan aplikasi dan operasi yang paling sering dilakukan pada graf.

Pohon adalah struktur data yang menyerupai graf namun dengan aturan tambahan: setiap simpul kecuali simpul akar (root) memiliki tepat satu induk (parent), dan simpul akar tidak memiliki induk. Pohon biner adalah jenis pohon di mana setiap simpul memiliki paling banyak dua anak. Pohon digunakan untuk merepresentasikan data hirarkis seperti pohon keluarga, struktur organisasi, dan banyak aplikasi lainnya. Operasi dasar pada pohon meliputi penciptaan pohon baru, pengosongan pohon, pemeriksaan apakah pohon kosong, penyisipan node, pencarian node, pembaruan data node, penghapusan sub-pohon, dan penelusuran dengan metode pre-order, in-order, dan post-order.

Implementasi graf dan pohon dalam bahasa pemrograman seperti C++ memerlukan pemahaman tentang struktur data dan algoritma. Graf berarah dapat direpresentasikan menggunakan matriks ketetanggaan yang mencatat bobot atau jarak antara simpul-simpul. Pohon biner diimplementasikan menggunakan struktur data yang memiliki pointer untuk anak kiri dan kanan serta pointer ke induk. Program dalam C++ dapat menyediakan berbagai fungsi untuk memanipulasi dan mengelola pohon biner, termasuk penambahan, pembaruan, penghapusan, penelusuran, dan karakteristik seperti ukuran dan tinggi pohon. Melalui implementasi ini, pengguna dapat melakukan berbagai operasi pada graf dan pohon dengan lebih efektif dan efisien.

DAFTAR PUSTAKA

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications

Modul 9 Graph and Tree Praktikum Struktur Data dan Algoritma Pemrograman

Struktur Data Graph diakses dari https://dif.telkomuniversity.ac.id/pengertian-struktur-data-graf/#:~:text=Secara%20lebih%20rinci%2C%20struktur%20data,bersifat%20many%2Dto%2Dmany.

Struktur Data diakses dari https://github.com/yunusfebriansyah/struktur-data/tree/main/Circular%20Single%20Linked%20List