

LAPORAN PRAKTIKUM

MODUL IX GRAPH DAN TREE



**Disusun oleh:
Andika Indra Prastawa
NIM: 2311102033**

Dosen Pengampu:

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

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023**

BAB I

TUJUAN PRAKTIKUM

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

BAB II

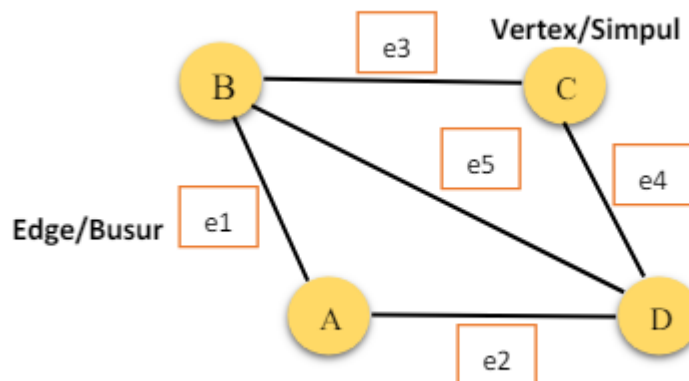
DASAR TEORI

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 :

$$G = (V, E)$$

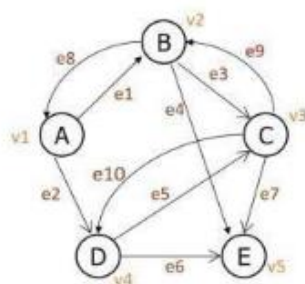
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:



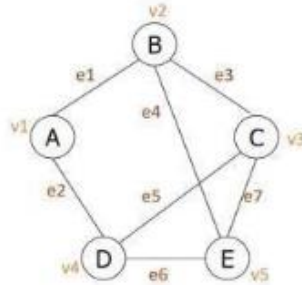
Gambar 1 Contoh Graph

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

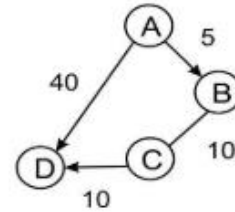
Jenis-jenis Graph



Directed Graph



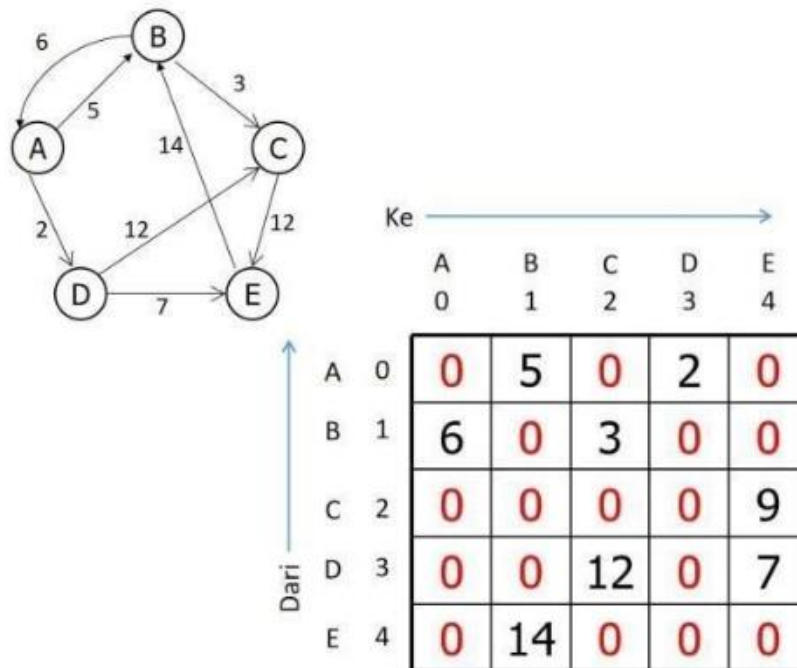
Undirected Graph



Weight Graph

- 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

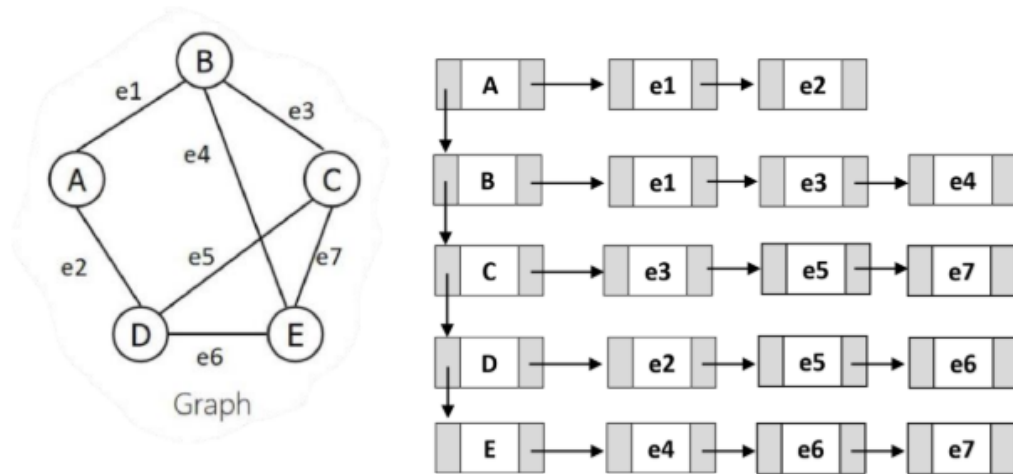
Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

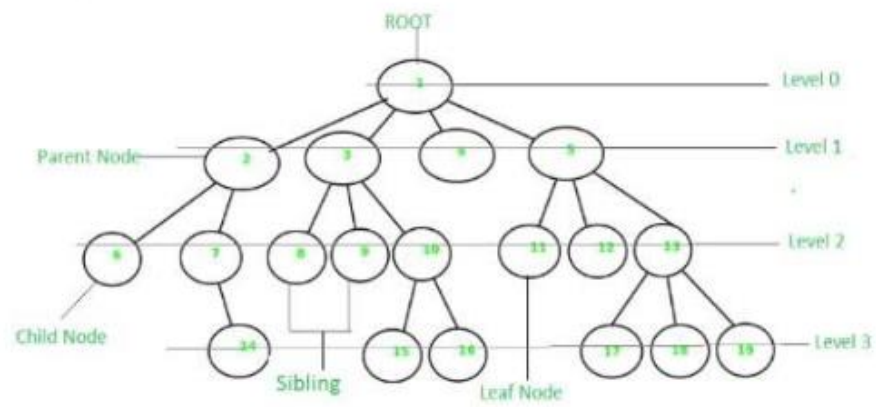
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun

biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

Tree atau Pohon Dalam ilmu komputer, pohon 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, di mana 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 hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



BAB III

GUIDED

1. Guided 1

Source code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};

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

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



```

        cout << " " << simpul[kolom] << "("
            << busur[baris][kolom] << ")";

    }

}

cout << endl;

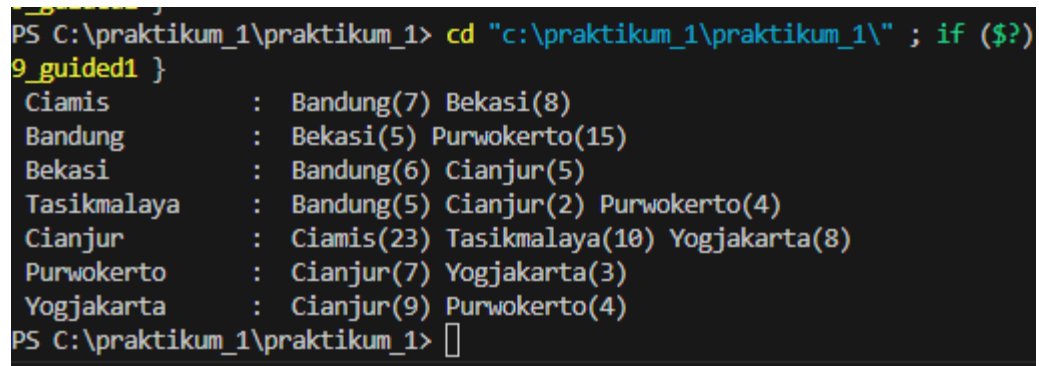
}

}

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

```

Screenshoot program



```

PS C:\praktikum_1\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { 9_guided1 }
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_1\praktikum_1>

```

Deskripsi program

Program mendefinisikan dan menampilkan graf yang direpresentasikan menggunakan array dua dimensi untuk menyimpan busur (edges) antara simpul (vertices). Simpul-simpul graf disimpan dalam array simpul, yang mencakup kota-kota seperti Ciamis, Bandung, Bekasi, dan lainnya. Busur antar simpul disimpan dalam array busur, di mana nilai selain nol menunjukkan adanya sambungan dan nilai tersebut merupakan bobot dari sambungan tersebut. Fungsi tampilGraph() digunakan untuk menampilkan graf dengan mencetak setiap simpul beserta simpul-simpul yang terhubung dengannya dan bobot masing-masing busur. Program dimulai dengan memanggil fungsi tampilGraph() di dalam fungsi main(),

yang menghasilkan output graf yang terstruktur dengan informasi tentang setiap koneksi dan bobotnya.

2. Guided 2

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;
}

```

```

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

```



```

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

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

```

```

        }
        else
        {
            delete node;
        }
    }
}

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

// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node = root)

```

```

{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {

```

```

        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        if (heightKiri >= heightKanan)
        {
            return heightKiri + 1;
        }
        else
        {
            return heightKanan + 1;
        }
    }
}

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

int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);

```

```

nodeJ = insertRight('J', nodeG);
update('Z', nodeC);
update('C', nodeC);
retrieve(nodeC);
find(nodeC);
cout << "\n PreOrder :" << endl;
preOrder(root);
cout << "\n"
    << endl;
cout << " InOrder :" << endl;
inOrder(root);
cout << "\n"
    << endl;
cout << " PostOrder :" << endl;
postOrder(root);
cout << "\n"
    << endl;
charateristic();
deleteSub(nodeE);
cout << "\n PreOrder :" << endl;
preOrder();
cout << "\n"
    << endl;
charateristic();
}

```

Screenshoot program

```
Yogyakarta : Cianjur(9) Purwokerto(4)  
PS C:\praktikum_1\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { g++  
9_guided2 }
```

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child 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_1\praktikum_1>

```

Deskripsi program

Program ini menyediakan berbagai fungsi untuk mengelola pohon biner, termasuk inisialisasi pohon, pengecekan apakah pohon kosong, pembuatan node baru, penambahan node di kiri dan kanan, pengubahan data node, penelusuran data node, pencarian data node, dan berbagai metode traversal seperti pre-order, in-order, dan post-order. Selain itu, terdapat juga fungsi untuk menghapus seluruh pohon atau subpohon, serta menghitung ukuran dan tinggi pohon. Di dalam fungsi main(), program membuat pohon dengan root 'A' dan beberapa node lainnya, mengubah data node, melakukan penelusuran dan pencarian, menampilkan

karakteristik pohon, serta menghapus subpohon dan menampilkan hasilnya setelah penghapusan.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

int main() {
    int jumlah_simpul;

    cout << "Silakan masukan jumlah simpul : ";
    cin >> jumlah_simpul;
    string nama_simpul[jumlah_simpul];

    cout << "Silakan masukan nama simpul\n";

    cin.ignore();
    for (int i = 0; i < jumlah_simpul; i++) {
        cout << "Simpul " << i+1 << " : ";
        getline(cin, nama_simpul[i]);
    }

    int simpul[jumlah_simpul][jumlah_simpul];
```



```

        cout << "Silakan masukkan bobot antar simpul\n";

        for (int i = 0; i < jumlah_simpul; i++) {
            for (int k = 0; k < jumlah_simpul; k++) {
                cout << nama_simpul[i] << " --> " << nama_simpul[k]
<< " = ";
                cin >> simpul[i][k];
            }
        }

        int formatting = nama_simpul[0].length()+1;

        for (int i = 0; i < jumlah_simpul-1; i++) {
            if (nama_simpul[i].length() < nama_simpul[i+1].length())
        {
                formatting = nama_simpul[i+1].length()+1;
            }
        }

        cout << left << setw(formatting) << " ";
        for (int i = 0; i < jumlah_simpul; i++) {
            cout << left << setw(nama_simpul[i].length()) <<
nama_simpul[i]<< "    ";
        }
        cout << endl;

        int Andikaindraprastawa_2311102033 = sizeof(nama_simpul) /
sizeof(*nama_simpul);

        for (int i = 0; i < jumlah_simpul; i++) {
            cout << left << setw(formatting) << nama_simpul[i];
            for (int k = 0; k < Andikaindraprastawa_2311102033; k++)
        {

```

```

        cout << right << setw(nama_simpul[k].length()) <<
simpul[i][k] << "    ";
    }
    cout << endl;
}
}

```

Screenshoot program

```

PS C:\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { g++
ed1 }
Silakan masukan jumlah simpul : 3
Silakan masukan nama simpul
Simpul 1 : andika
Simpul 2 : indra
Simpul 3 : prastawa
Silakan masukkan bobot antar simpul
andika --> andika = 2
andika --> indra = 3
andika --> prastawa = 4
indra --> andika = 2
indra --> indra = 3
indra --> prastawa = 4
prastawa --> andika = 2
prastawa --> indra = 3
prastawa --> prastawa = 4
      andika    indra    prastawa
andika      2      3      4
indra       2      3      4
prastawa    2      3      4
PS C:\praktikum_1\praktikum_1> 

```

Deskripsi program

Program mengimplementasi untuk menerima dan menampilkan graf berbobot dari masukan pengguna. Pengguna diminta untuk memasukkan jumlah simpul, kemudian memasukkan nama untuk setiap simpul. Selanjutnya, pengguna memasukkan bobot antara setiap pasangan simpul. Program ini kemudian menghitung panjang nama simpul terpanjang untuk menentukan format keluaran

yang rapi. Setelah itu, program menampilkan matriks bobot yang diatur berdasarkan nama simpul yang telah diberikan. Matriks ini menunjukkan bobot dari setiap hubungan antara simpul, dengan nama simpul di sepanjang baris dan kolom untuk memudahkan interpretasi visual.

2. Unguided 2

Source code

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

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

Pohon *root = nullptr;

void init() {
    root = NULL;
}

int isEmpty() {
    return (root == NULL) ? 1 : 0;
}

Pohon* buatNode(char data) {
    Pohon* newNode = new Pohon();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
```

```

        newNode->parent = NULL;

        return newNode;
    }

Pohon* insertLeft(Pohon* parent, Pohon* child) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->left != NULL) {
            cout << "\nNode " << parent->left->data << " sudah
ada child kiri!" << endl;
            return NULL;
        } else {
            child->parent = parent;
            parent->left = child;

            return child;
        }
    }
}

Pohon* insertRight(Pohon* parent, Pohon* child) {
    if (root == NULL) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->right != NULL) {
            cout << "\nNode " << parent->right->data << " sudah
ada child kanan!" << endl;
            return NULL;
        }
    }
}

```

```

        } else {
            child->parent = parent;
            parent->right = child;
            return child;
        }
    }
}

void update(char data, Pohon *node) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" <<
endl;
        else {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}

void retrieve(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

```

```

}

void find(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->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 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 deleteTree(Pohon *node) {
    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;
        }
    }
}

void deleteSub(Pohon *node) {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

void clear() {

```



```

        if (!root)
            cout << "\nBuat tree terlebih dahulu!!" << endl;
        else {
            deleteTree(root);
            cout << "\nPohon berhasil dihapus." << endl;
        }
    }

int size(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

int height(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return (heightKiri >= heightKanan) ? heightKiri + 1 :
heightKanan + 1;
    }
}

void charateristic() {
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
}

```

```

        cout << "Average Node of Tree : " << (size(root) /
(float)height(root)) << endl;
    }

int main() {
    root = buatNode('A');
    int menu, part, part2;
    char Andikaindraprastawa_2311102033;

    vector<Pohon*> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));

    insertLeft(root, nodes[0]);
    insertRight(root, nodes[1]);
    insertLeft(nodes[0], nodes[2]);
    insertRight(nodes[0], nodes[3]);
    insertLeft(nodes[1], nodes[4]);
    insertLeft(nodes[3], nodes[5]);
    insertRight(nodes[3], nodes[6]);
    insertLeft(nodes[5], nodes[7]);
    insertRight(nodes[5], nodes[8]);

    do
    {
        cout << "\n----- PROGHRAM GRAAAAPH ----- \n"

```

```

"1. Tambah node\n"
"2. Tambah di kiri\n"
"3. Tambah di kanan\n"
"4. Lihat karakteristik tree\n"
"5. Lihat isi data tree\n"
"6. Cari data tree\n"
"7. Penelurusan (Traversal) preOrder\n"
"8. Penelurusan (Traversal) inOrder\n"
"9. Penelurusan (Traversal) postOrder\n"
"10. Hapus subTree\n"
"0. KELUAR\n"
"\nPilih : ";
cin >> menu;
cout << "-----Running Command...\n";
switch (menu) {
    case 1:
        cout << "\n Nama Node (Character) : ";
        cin >> Andikaindraprastawa_2311102033;

nodes.push_back(buatNode(Andikaindraprastawa_2311102033));
        break;
    case 2:
        cout << "\nMasukkan nomor untuk node parent : ";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertLeft(nodes[part], nodes[part2]);
        break;
    case 3:
        cout << "\nMasukkan nomor untuk node parent : ";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertRight(nodes[part], nodes[part2]);

```

```
        break;
    case 4:
        charateristic();
        break;
    case 5:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        retrieve(nodes[part]);
        break;
    case 6:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        find(nodes[part]);
        break;
    case 7:
        cout << "\nPreOrder :" << endl;
        preOrder(root);
        cout << "\n" << endl;
        break;
    case 8:
        cout << "\nInOrder :" << endl;
        inOrder(root);
        cout << "\n" << endl;
        break;
    case 9:
        cout << "\nPostOrder :" << endl;
        postOrder(root);
        cout << "\n" << endl;
        break;
    case 10:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        deleteSub(nodes[part]);
        break;
```

```

        default:
            break;
    }
} while (menu != 0);
}

```

Screenshoot program

```

PS C:\praktikum_1\praktikum_1> cd C:\praktikum_1\praktikum_1\ ; if ($?) { g+
prak9_unguided2 }

----- PROGHRAM GRAAAAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 4
-----Running Command...

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

```

----- PROGHRAM GRAAAAPH -----

1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 5

-----Running Command...

Masukkan nomor node : 1

Data node : C

```
PS C:\praktikum_1\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { g++ lap  
prak9_unguided2 }
```

```
----- PROGHRAM GRAAAAPH -----
```

1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 6

-----Running Command...

Masukkan nomor node : 8

Data Node : J

Root : A

Parent : G

Sibling : I

Child Kiri : (tidak punya Child kiri)

Child Kanan : (tidak punya Child kanan)

```
PS C:\praktikum_1\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { g++ laprak9_unguided2.cpp ; if ($?) { ./praktikum_1.exe ; } } }
```

```
----- PROGHAM GRAAAAPH -----
```

1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 7

-----Running Command...

PreOrder :

A, B, D, E, G, I, J, H, C, F,

```
PS C:\praktikum_1\praktikum_1> cd "c:\praktikum_1\praktikum_1\" ; if ($?) { g++ laprak9_unguided2.cpp ; if ($?) { ./praktikum_1.exe ; } } }
```

```
----- PROGHAM GRAAAAPH -----
```

1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 8

-----Running Command...

InOrder :

D, B, I, G, J, E, H, A, F, C,


```

----- PROGHRAM GRAAAAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 9
-----Running Command...

PostOrder :
D, I, J, G, H, E, B, F, C, A,

```

```

----- PROGHRAM GRAAAAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 10
-----Running Command...

Masukkan nomor node : 7

Node subtree I berhasil dihapus.

```

Deskripsi program

Program mengimplementasikan struktur data pohon biner. Pada awalnya, program menginisialisasi pohon dan mendefinisikan berbagai fungsi untuk operasi pada pohon, termasuk membuat node baru (`buatNode`), menambahkan node ke kiri atau kanan dari node induk (`insertLeft` dan `insertRight`), memperbarui data node (`update`), mengambil data node (`retrieve`), dan menemukan informasi detail dari node tertentu (`find`). Selain itu, program juga menyediakan fungsi untuk melakukan traversal pohon dalam tiga urutan berbeda: pre-order (`preOrder`), in-order (`inOrder`), dan post-order (`postOrder`). Ada juga fungsi untuk menghapus seluruh pohon atau subtree tertentu (`deleteTree` dan `deleteSub`), serta mendapatkan karakteristik pohon seperti ukuran dan tinggi pohon (`characteristic`).

Program utama (`main`) dimulai dengan membuat root node dan beberapa node lainnya yang kemudian dihubungkan satu sama lain untuk membentuk struktur pohon biner. Selanjutnya, program menyediakan antarmuka pengguna berbasis teks yang memungkinkan pengguna untuk menambahkan node baru, menambahkan node ke kiri atau kanan node yang ada, melihat karakteristik pohon, mengambil atau mencari data node tertentu, melakukan traversal pohon, dan menghapus subtree. Menu tersebut berulang dalam loop `do-while` hingga pengguna memilih untuk keluar dengan memasukkan angka 0. Dengan ini, pengguna dapat secara interaktif membangun dan memanipulasi pohon biner serta mengeksplorasi berbagai operasi yang dapat dilakukan pada pohon tersebut.

BAB IV

KESIMPULAN

Graf dan pohon adalah dua struktur data penting dalam ilmu komputer yang digunakan dalam berbagai aplikasi seperti jaringan sosial, pemetaan jalan, dan pemodelan data hierarkis. Representasi dan operasi pada graf dan pohon memungkinkan manipulasi dan analisis data yang efisien. Pemahaman mendalam tentang konsep dan operasi pada graf dan pohon memungkinkan pengembang perangkat lunak merancang dan mengimplementasikan solusi efektif untuk berbagai masalah. Graf, direpresentasikan sebagai $G=(V,E)$, terdiri dari simpul (vertex) dan sambungan (edge), yang dapat berarah atau tak berarah serta berbobot. Representasi graf dapat menggunakan matriks atau linked list, di mana simpul vertex dan edge dibedakan untuk memudahkan pemrosesan. Pohon adalah struktur data hierarkis di mana setiap simpul memiliki satu induk dan nol atau lebih anak, dengan binary tree sebagai jenis yang setiap simpulnya memiliki maksimal dua anak. Operasi pada pohon meliputi pembuatan, penghapusan, penelusuran, dan penampilan karakteristik seperti ukuran dan tinggi, serta traversal yang dilakukan dengan metode pre-order, in-order, dan post-order.

DAFTAR PUSTAKA

[1] Learning Management System ,MODUL 9 GRAPH DAN TREE.