LAPORAN PRAKTIKUM

MODUL IX
GRAPH DAN TREE

Disusun oleh:
Brian Farrel Evandhika
NIM: 2311102037

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

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

A. TUJUAN PRAKTIKUM

a. Mahasiswa diharapkan mampu memahami graph dan tree

b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

B. Dasar Teori

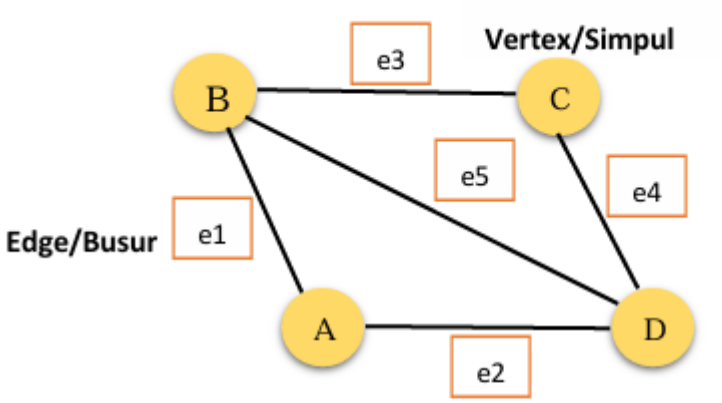
1. Graph

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

G = (V, E)

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

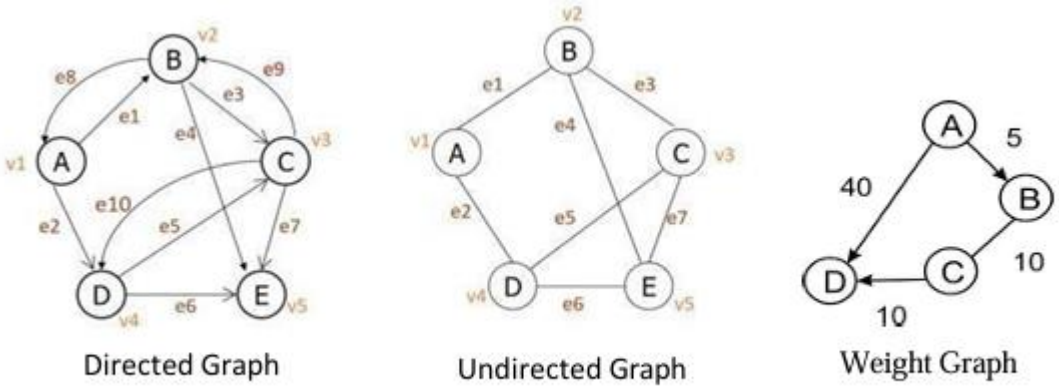
Dapat digambarkan:



Gambar 1 Contoh Graph

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

Jenis-jenis Graph



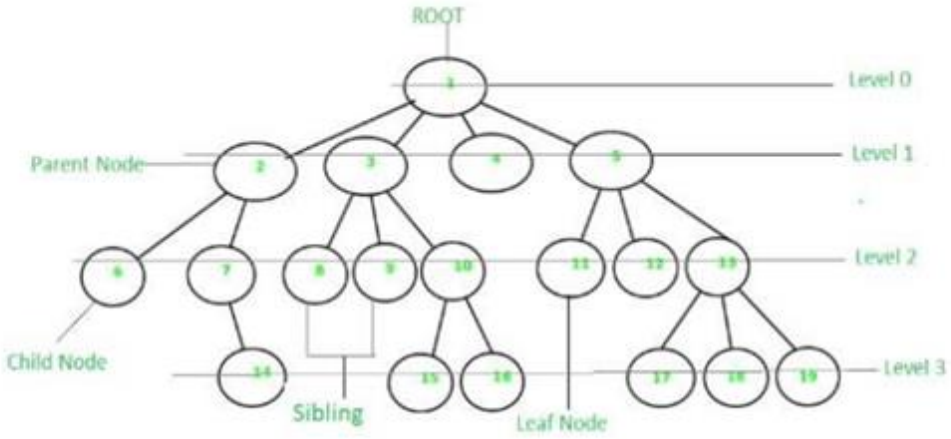
a. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.

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

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

2. 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 :



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.

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

2. In-Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Cetak data pada root

c. Secara rekursif mencetak seluruh data pada subpohon kanan

3. Post Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri

b. Secara rekursif mencetak seluruh data pada subpohon kanan

c. Cetak data pada root

GUIDED

Guided 1

```
#include <iostream> // Library standar yang digunakan untuk input dan output
#include <iomanip>  // Library standar yang digunakan untuk manipulasi input
dan output

using namespace std; // Untuk mempersingkat penulisan kode program

string simpul[7] =
    {
        "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto",
"Yogyakarta"}; // Array yang berisi nama-nama simpul

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}}; // Array yang berisi bobot dari busur

void tampilGraph() // Fungsi untuk menampilkan graph
{
    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() // Fungsi utama program
{
    tampilGraph();
    return 0;
}
```

Screenshot Program

```
PS C:\Users\MSI GAMING>  & 'c:\Users\MSI GAMING\.vscode\extensions\ms
-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLaunc
her.exe' '--stdin=Microsoft-MIEngine-In-btx4gzej.yhh' '--stdout=Micro
soft-MIEngine-Out-atjahfff.tqv' '--stderr=Microsoft-MIEngine-Error-32
hxed5n.fg4' '--pid=Microsoft-MIEngine-Pid-44jhowk3.1e0' '--dbgExe=C:\
msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
 Ciamis          :  Bandung(7)  Bekasi(8)
 Bandung         :  Bekasi(5)  Purwokerto(15)
 Bekasi          :  Bandung(6)  Cianjur(5)
 Tasikmalaya     :  Bandung(5)  Cianjur(2)  Purwokerto(4)
 Cianjur         :  Ciamis(23)  Tasikmalaya(10)  Yogyakarta(8)
 Purwokerto      :  Cianjur(7)  Yogyakarta(3)
 Yogyakarta      :  Cianjur(9)  Purwokerto(4)
PS C:\Users\MSI GAMING>
```

Deskripsi Program

Program ini mendefinisikan dan menampilkan representasi sebuah graf yang terdiri dari simpul dan busur. Graf ini direpresentasikan menggunakan array simpul yang menyimpan nama-nama simpul, serta matriks busur yang menyimpan bobot antara simpul-simpul tersebut. Fungsi tampilGraph() digunakan untuk menampilkan graf dengan iterasi melalui setiap simpul dan menampilkan busur yang terhubung beserta bobotnya. Pada fungsi main(), tampilGraph() dipanggil untuk menampilkan struktur graf ke layar, memperlihatkan koneksi antara kota-kota yang direpresentasikan sebagai simpul dalam graf.

Guided 2

```cpp
#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
```

Deskripsi Program

Program ini mengimplementasikan struktur data pohon biner yang dimulai dengan deklarasi struktur pohon yang mencakup data dan pointer ke anak kiri, anak kanan, serta induk. Fungsi-fungsi utama dalam program ini meliputi inisialisasi pohon (init), pengecekan apakah pohon kosong (isEmpty), pembuatan node baru (buatNode), penambahan node pada anak kiri (insertLeft) dan kanan (insertRight), pengubahan data node (update), serta penampilan data node (retrieve) dan pencarian node (find). Selain itu, terdapat fungsi-fungsi untuk traversal pohon seperti pre-order, in-order, dan post-order, penghapusan pohon atau subtree (deleteTree, deleteSub), serta pemeriksaan karakteristik pohon (size, height, characteristic). Pada fungsi main, pohon biner dibangun dengan root 'A' dan beberapa node tambahan, diikuti oleh berbagai operasi yang hasilnya ditampilkan ke layar.

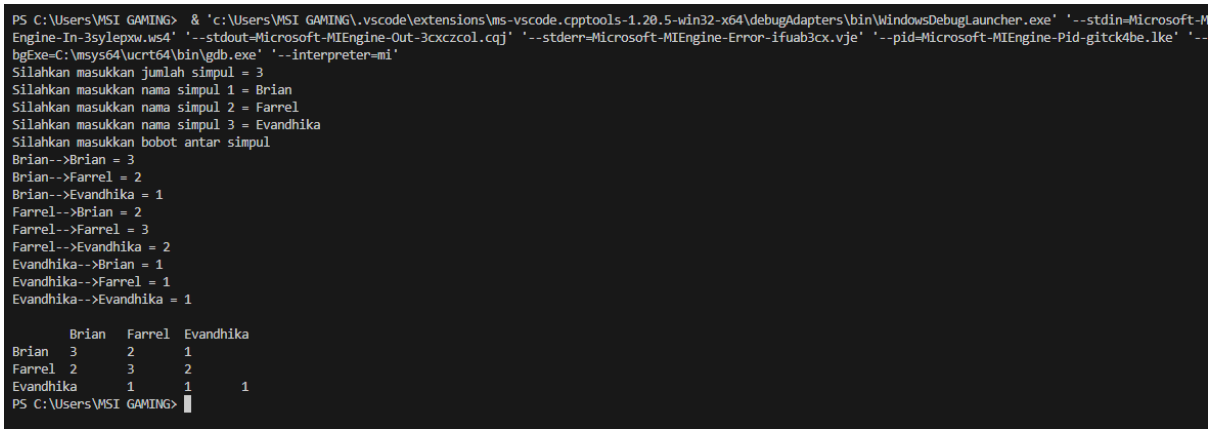
Unguided

Unguided 1

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

int main() {
    int jmlSimpulBrianFarrelEvandhika_2311102037;
    cout << "Silahkan masukkan jumlah simpul = ";
    cin >> jmlSimpulBrianFarrelEvandhika_2311102037;

    vector<string>
namaSimpulBrianFarrelEvandhika_2311102037(jmlSimpulBrianFarrelEvandhika_231110
2037);
    vector<vector<int>> bobot(jmlSimpulBrianFarrelEvandhika_2311102037,
vector<int>(jmlSimpulBrianFarrelEvandhika_2311102037));

    for (int i = 0; i < jmlSimpulBrianFarrelEvandhika_2311102037; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << " = ";
        cin >> namaSimpulBrianFarrelEvandhika_2311102037[i];
    }

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

    for (int i = 0; i < jmlSimpulBrianFarrelEvandhika_2311102037; ++i) {
        for (int j = 0; jmlSimpulBrianFarrelEvandhika_2311102037 && j <
jmlSimpulBrianFarrelEvandhika_2311102037; ++j) {
            cout << namaSimpulBrianFarrelEvandhika_2311102037[i] << "-->" <<
namaSimpulBrianFarrelEvandhika_2311102037[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << "\n\t";
    for (int i = 0; i < jmlSimpulBrianFarrelEvandhika_2311102037; ++i) {
        cout << namaSimpulBrianFarrelEvandhika_2311102037[i] << "\t";
    }
    cout << "\n";

    for (int i = 0; i < jmlSimpulBrianFarrelEvandhika_2311102037; ++i) {
        cout << namaSimpulBrianFarrelEvandhika_2311102037[i] << "\t";
        for (int j = 0; j < jmlSimpulBrianFarrelEvandhika_2311102037; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }
```

```
    return 0;
}
```

Screenshot Program



Deskripsi Program

Program ini meminta pengguna untuk memasukkan jumlah simpul pada sebuah graf, lalu memasukkan nama masing-masing simpul tersebut. Setelah itu, pengguna diminta untuk menginput bobot antara simpul-simpul, yang disimpan dalam matriks bobot. Setelah semua data dimasukkan, program menampilkan tabel yang menunjukkan bobot antara simpul dengan nama simpul sebagai baris dan kolom. Program ini menggunakan vector untuk menyimpan nama simpul dan bobot antara simpul, serta melakukan iterasi melalui input dan output menggunakan loop

Unguided 2

```
#include <iostream>  // Library standar yang digunakan untuk input dan output
#include <queue>     // Library standar yang digunakan untuk queue
using namespace std; // Untuk mempersingkat penulisan kode program

struct Pohon // Struct yang berisi data node tree
{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root; // Pointer root tree

void init() // Fungsi untuk inisialisasi tree
{
    root = NULL;
}

bool isEmpty() // Fungsi untuk mengecek apakah tree kosong
{
    return root == NULL;
}

void buatNode(char data) // Fungsi untuk membuat node tree
{
    if (isEmpty())
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\nNode " << data << " berhasil dibuat sebagai root." << endl;
    }
    else
    {
        cout << "\nTree sudah ada!" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node) // Fungsi untuk menambahkan node ke
child kiri
{
    if (node->left != NULL)
    {
        cout << "\nNode " << 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 << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
baru->parent->data << endl;
        return baru;
    }
}

Pohon *insertRight(char data, Pohon *node) // Fungsi untuk menambahkan node ke
child kanan
{
    if (node->right != NULL)
    {
        cout << "\nNode " << 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 << "\nNode " << data << " berhasil ditambahkan ke child kanan "
<< baru->parent->data << endl;
        return baru;
    }
}

void update(char data, Pohon *node) // Fungsi untuk mengubah data node
{
    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) // Fungsi untuk menampilkan data node
{
    if (!node)
    {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    }
    else
    {
        cout << "\nData node: " << node->data << endl;
    }
}

void find(Pohon *node) // Fungsi untuk mencari node
{
    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) // Fungsi untuk melakukan preOrder traversal
```

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

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

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

void deleteTree(Pohon *node) // Fungsi untuk menghapus tree
{
    if (node != NULL)
    {
        if (node != root)
        {
            if (node->parent->left == node)
                node->parent->left = NULL;
            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) // Fungsi untuk menghapus subtree
{
    if (!isEmpty())
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil dihapus." <<
endl;
    }
}

void clear() // Fungsi untuk menghapus tree
{
    if (!isEmpty())
    {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

int size(Pohon *node = root) // Fungsi untuk menghitung jumlah node tree
{
    if (!node)
    {
        return 0;
    }
    else
    {
        return 1 + size(node->left) + size(node->right);
    }
}

int height(Pohon *node = root) // Fungsi untuk menghitung tinggi tree
{
    if (!node)
    {
        return 0;
    }
    else
    {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
```

```
        return max(heightKiri, heightKanan) + 1;
    }
}

void characteristic() // Fungsi untuk menampilkan karakteristik tree
{
    cout << "\nSize Tree: " << size() << endl;
    cout << "Height Tree: " << height() << endl;
    if (height() != 0)
    {
        cout << "Average Node of Tree: " << (double)size() / height() << endl;
    }
}
void TampilChild_Descendants(Pohon *node) // Fungsi untuk menampilkan child
dan descendant dari node
{
    if (!node)
    {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    }
    else
    {
        cout << "\nNode: " << node->data << 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;

        cout << "Descendants:" << endl;
        queue<Pohon *> q;
        if (node->left)
            q.push(node->left);
        if (node->right)
            q.push(node->right);
        while (!q.empty())
        {
            Pohon *current = q.front();
            q.pop();
            cout << " " << current->data << ", ";
            if (current->left)
                q.push(current->left);
            if (current->right)
                q.push(current->right);
```

```
        }
        cout << endl;
    }
}

void brianfarrelevandhika_2311102037() // Fungsi untuk menampilkan menu
{
    int choice;
    do
    {
        cout << "\nMenu:\n";
        cout << "1. Buat Node\n";
        cout << "2. Update Node\n";
        cout << "3. Retrieve Node\n";
        cout << "4. Find Node\n";
        cout << "5. Travers Data\n";
        cout << "6. Hapus Subtree\n";
        cout << "7. Clear Tree\n";
        cout << "8. Karakteristik Tree\n";
        cout << "9. Display Child dan Descendants\n";
        cout << "10. Exit\n";
        cout << "Pilih opsi: ";
        cin >> choice;
        char data, parentData;
        Pohon *parentNode = nullptr;
        switch (choice)
        {
        case 1: // Buat Node
            if (isEmpty())
            {
                cout << "Masukkan data root: ";
                cin >> data;
                buatNode(data);
            }
            else // Insert Left atau Insert Right
            {
                cout << "Masukkan data node baru: ";
                cin >> data;
                cout << "Masukkan data parent node: ";
                cin >> parentData;
                cout << "Pilih 1 untuk Insert Left, 2 untuk Insert Right: ";
                char insertChoice;
                cin >> insertChoice;
                parentNode = nullptr;
                queue<Pohon *> q;
                q.push(root);
                while (!q.empty())
                {
```

```
                Pohon *node = q.front();
                q.pop();
                if (node->data == parentData)
                {
                    parentNode = node;
                    break;
                }
                if (node->left)
                    q.push(node->left);
                if (node->right)
                    q.push(node->right);
            }
            if (parentNode)
            {
                if (insertChoice == '1')
                {
                    insertLeft(data, parentNode);
                }
                else if (insertChoice == '2')
                {
                    insertRight(data, parentNode);
                }
                else
                {
                    cout << "Opsi tidak valid!" << endl;
                }
            }
            else
            {
                cout << "Parent node tidak ditemukan!" << endl;
            }
        }
        break;
    case 2: // Update Node
        if (!isEmpty())
        {
            cout << "Masukkan data node baru: ";
            cin >> data;
            cout << "Masukkan data node yang ingin diupdate: ";
            cin >> parentData;
            parentNode = nullptr;
            // Cari node yang ingin diupdate menggunakan BFS
            queue<Pohon *> q;
            q.push(root);
            while (!q.empty())
            {
                Pohon *node = q.front();
                q.pop();
```

```
                    if (node->data == parentData)
                    {
                        parentNode = node;
                        break;
                    }
                    if (node->left)
                        q.push(node->left);
                    if (node->right)
                        q.push(node->right);
                }
                if (parentNode)
                {
                    update(data, parentNode);
                }
                else
                {
                    cout << "Node yang ingin diupdate tidak ditemukan!" <<
endl;
                }
            }
            else
            {
                cout << "Buat tree terlebih dahulu!" << endl;
            }
            break;
        case 3: // Retrieve Node
            if (!isEmpty())
            {
                cout << "Masukkan data node yang ingin diretrieve: ";
                cin >> data;
                parentNode = nullptr;
                // Cari node yang ingin diretrieve menggunakan BFS
                queue<Pohon *> q;
                q.push(root);
                while (!q.empty())
                {
                    Pohon *node = q.front();
                    q.pop();
                    if (node->data == data)
                    {
                        parentNode = node;
                        break;
                    }
                    if (node->left)
                        q.push(node->left);
                    if (node->right)
                        q.push(node->right);
                }
```

```
                if (parentNode)
                {
                    retrieve(parentNode);
                }
                else
                {
                    cout << "Node yang ingin diretrieve tidak ditemukan!" <<
endl;
                }
            }
            else
            {
                cout << "Buat tree terlebih dahulu!" << endl;
            }
            break;
        case 4: // Find Node
            if (!isEmpty())
            {
                cout << "Masukkan data node yang ingin dicari: ";
                cin >> data;
                parentNode = nullptr;
                // Cari node yang ingin dicari menggunakan BFS
                queue<Pohon *> q;
                q.push(root);
                while (!q.empty())
                {
                    Pohon *node = q.front();
                    q.pop();
                    if (node->data == data)
                    {
                        parentNode = node;
                        break;
                    }
                    if (node->left)
                        q.push(node->left);
                    if (node->right)
                        q.push(node->right);
                }
                if (parentNode)
                {
                    find(parentNode);
                }
                else
                {
                    cout << "Node yang ingin dicari tidak ditemukan!" << endl;
                }
            }
            else
```

```
        {
            cout << "Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 5: // Travers Data
        if (!isEmpty())
        {
            cout << "\nPreOrder Traversal:" << endl;
            preOrder(root);
            cout << "\n";
            cout << "\nInOrder Traversal:" << endl;
            inOrder(root);
            cout << "\n";
            cout << "\nPostOrder Traversal:" << endl;
            postOrder(root);
            cout << "\n";
        }
        else
        {
            cout << "Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 6: // Hapus Subtree
        if (!isEmpty())
        {
            cout << "Masukkan data node subtree yang ingin dihapus: ";
            cin >> data;
            parentNode = nullptr;
            // Cari node subtree yang ingin dihapus menggunakan BFS
            queue<Pohon *> q;
            q.push(root);
            while (!q.empty())
            {
                Pohon *node = q.front();
                q.pop();
                if (node->data == data)
                {
                    parentNode = node;
                    break;
                }
                if (node->left)
                    q.push(node->left);
                if (node->right)
                    q.push(node->right);
            }
            if (parentNode)
            {
                deleteSub(parentNode);
```

```
                }
                else
                {
                    cout << "Node subtree yang ingin dihapus tidak ditemukan!"
<< endl;
                }
            }
            else
            {
                cout << "Buat tree terlebih dahulu!" << endl;
            }
            break;
        case 7: // Clear Tree
            clear();
            break;
        case 8: // Karakteristik Tree
            characteristic();
            break;
        case 9: // Display Child dan Descendants
            if (!isEmpty())
            {
                cout << "Masukkan data node yang ingin ditampilkan child dan
descendants: ";
                cin >> data;
                parentNode = nullptr;
                // Cari node yang ingin ditampilkan child dan descendants
menggunakan BFS
                queue<Pohon *> q;
                q.push(root);
                while (!q.empty())
                {
                    Pohon *node = q.front();
                    q.pop();
                    if (node->data == data)
                    {
                        parentNode = node;
                        break;
                    }
                    if (node->left)
                        q.push(node->left);
                    if (node->right)
                        q.push(node->right);
                }
                if (parentNode)
                {
                    TampilChild_Descendants(parentNode);
                }
                else
```

```
                {
                    cout << "Node yang ingin ditampilkan child dan descendants
tidak ditemukan!" << endl;
                }
            }
            else
            {
                cout << "Buat tree terlebih dahulu!" << endl;
            }
            break;
            case 10: // Exit
            cout << "Terima kasih telah menggunakan program ini!" << endl;
            break;
        default: // Opsi tidak valid
            cout << "Opsi tidak valid!" << endl;
            break;
        }
    } while (choice != '9');
}

int main() // Fungsi utama program
{
    init();
    brianfarrelevandhika_2311102037();
    return 0;
}
```

Screenshot Program

```
Node F berhasil ditambahkan ke child kanan I

Menu:
1. Buat Node
2. Update Node
3. Retrieve Node
4. Find Node
5. Travers Data
6. Hapus Subtree
7. Clear Tree
8. Karakteristik Tree
9. Display Child dan Descendants
10. Exit
Pilih opsi: 1
Masukkan data node baru: E
Masukkan data parent node: R
Pilih 1 untuk Insert Left, 2 untuk Insert Right: 1

Node E berhasil ditambahkan ke child kiri R

Menu:
1. Buat Node
2. Update Node
3. Retrieve Node
4. Find Node
5. Travers Data
6. Hapus Subtree
7. Clear Tree
8. Karakteristik Tree
9. Display Child dan Descendants
10. Exit
Pilih opsi: 5

PreOrder Traversal:
 B,  I,  N,  F,  R,  E,  A,

InOrder Traversal:
 N,  I,  F,  B,  E,  R,  A,

PostOrder Traversal:
 N,  F,  I,  E,  A,  R,  B,
```

Deskripsi Program

Program di atas adalah implementasi dari struktur data pohon biner dengan menu interaktif untuk pengguna. Program ini mendefinisikan struktur Pohon yang merepresentasikan node dalam pohon biner dan menyediakan berbagai fungsi untuk menginisialisasi pohon (init), membuat node baru (buatNode), menambah node kiri (insertLeft) dan kanan (insertRight), mengubah data node (update), melihat isi node (retrieve), mencari node (find), serta melakukan traversal pohon dalam urutan pre-order, in-order, dan post-order. Selain itu, program ini memiliki fungsi untuk menghapus subtree (deleteSub) dan seluruh pohon (clear), serta menampilkan karakteristik pohon seperti ukuran dan tinggi. Melalui menu interaktif, pengguna dapat memasukkan data secara dinamis, melakukan berbagai operasi pada pohon, dan melihat hasilnya secara langsung.

KESIMPULAN

Graph dan tree adalah dua struktur data fundamental dalam ilmu komputer yang digunakan untuk merepresentasikan hubungan antara objek dan menyimpan data secara hierarki. Graph terdiri dari node (simpul) dan edge (busur), yang menggambarkan hubungan antar objek. Ada berbagai jenis graph seperti directed graph, undirected graph, dan weighted graph, yang biasanya direpresentasikan menggunakan matriks atau linked list. Di sisi lain, tree adalah struktur hierarki dengan node yang terhubung menyerupai bentuk pohon, dengan operasi seperti create, clear, insert, dan delete subtree, serta metode traversal seperti Pre-Order, In-Order, dan Post-Order. Kedua struktur data ini penting dalam pemodelan sistem kompleks dan representasi jaringan, dengan graph lebih fleksibel untuk menggambarkan hubungan umum antar objek, sementara tree sangat efektif untuk data hierarki dan operasi berbasis struktur.

DAFTAR PUSTAKA

[1] Asisten Praktikum. 2024. Modul 9 “GRAPH DAN TREE”. Diakses 10 Juni 2024, 19:00 WIB. https://lms.ittelkom-pwt.ac.id/