LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL IX
GRAPH DAN TREE

Disusun Oleh :
NAMA : D’sharlendita Febianda Aurelia
NIM : 2311102069

Dosen :
Wahyu Andi Saputra, S.pd., M,Eng

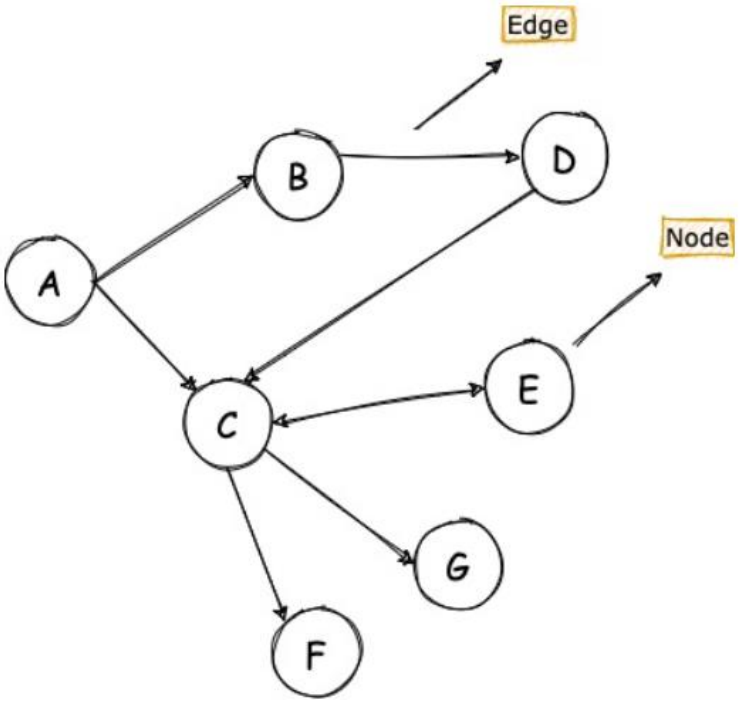
PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. Dasar Teori

1. Graph

Graph terdiri dari beberapa kumpulan titik (node) dan garis (edge).



Node, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri.

Edge, adalah penghubung antara satu node dengan node yang lain. Sebuah garis harus diawali dan diakhiri titik.

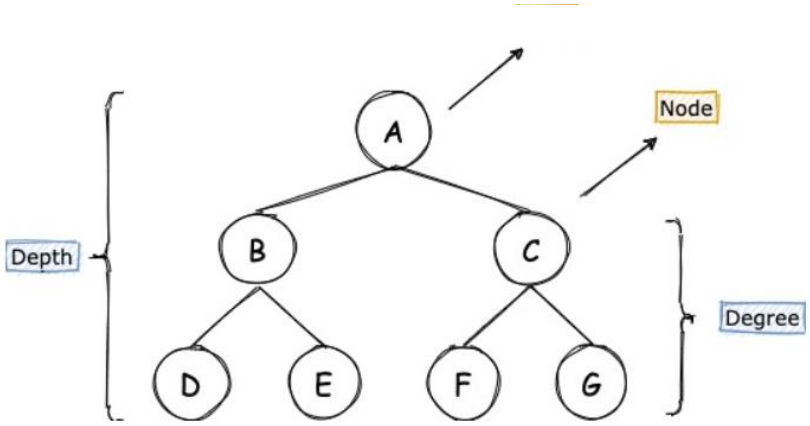
Path adalah jalur dari satu titik ke titik lain. Sebuah path yang diawali dan diakhiri dengan titik yang sama disebut juga dengan simpul tertutup.

Berdasarkan orientasi arah sisi nya, graph dapat dibedakan menjadi 2 yaitu:

- Directed graph atau graf berarah adalah graph yang setiap sisi nya memiliki orientasi arah.
- Undirected graph atau graf tak berarah adalah graph yang sisi nya tidak memiliki orientasi arah.

2. Tree

Tree adalah struktur data non linier berbentuk hierarki yang terdiri dari sekumpulan node yang berbeda.



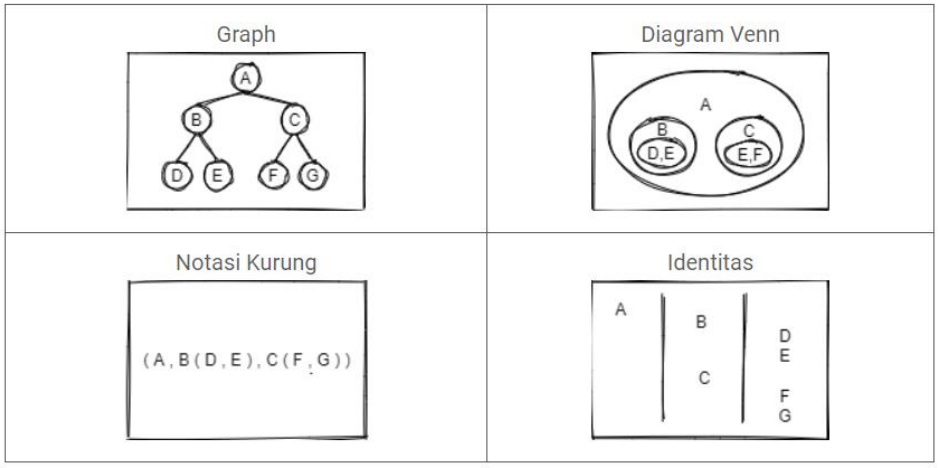
Node, adalah struktur yang berisi sebuah nilai atau suatu kondisi atau menggambarkan sebuah struktur data terpisah atau sebuah bagian pohon itu sendiri.

Root, adalah sebuah node yang terletak di posisi tertinggi atau urutan pertama dari suatu tree.

Depth, adalah jarak atau ketinggian antara root dan node.

Degree, adalah banyaknya anak atau turunan dari suatu node.

Ada beberapa cara untuk menggambar sebuah tree, diantaranya dapat dengan :



Tree yang hanya memiliki maksimal dua child (anak) disebut dengan binary tree atau pohon biner.

Operasi yang terdapat pada binary tree berdasarkan gambar diatas umumnya dapat dilakukan dengan urutan – urutan sebagai berikut :

- Pre Order (DFS – Depth First Search) : A, B, D, E, C, F, G
- In Order : D, B, E, A, F, C, G
- Last Order : D, E, B, F, G, C, A
- Level Order (BFS – Bread First Search): A, B, C, D, E, F, G

B. Guided

Guided 1

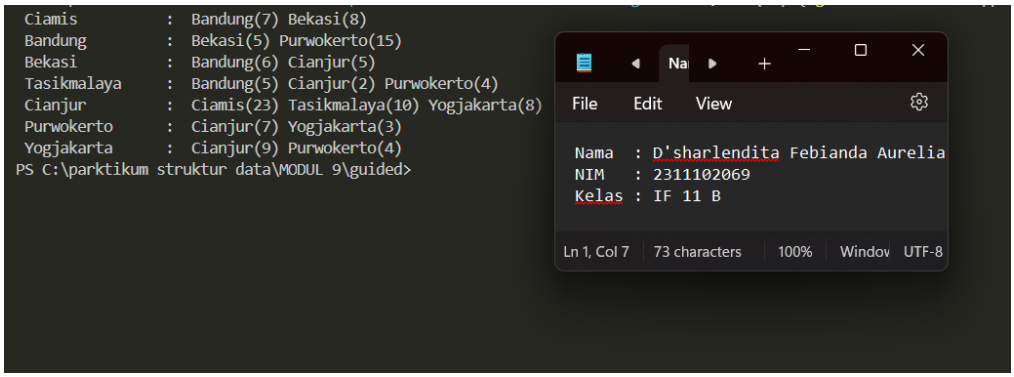
Program Graph

Source Code:

```
// D'sharlendita Febianda Aurelia
//2311102069

#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;
    }
}
```

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

Screenshots Output :



Deskripsi:

Program ini merepresentasikan graf berbobot dengan tujuh kota menggunakan matriks ketetanggaan di C++. Setiap elemen matriks menunjukkan bobot (jarak atau biaya) antara dua kota, 0 berarti tidak ada koneksi. Fungsi tampilGraph() mencetak setiap kota beserta kota-kota yang terhubung dan bobotnya. main() memanggil fungsi tampilGraph () untuk menampilkan graf tersebut.

Guided 2

Program Binary Tree

Source Code:

```
// D’sharlendita Febianda Aurelia
// 2311102069

#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; // true
    else
        return 0; // 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)
        {
```

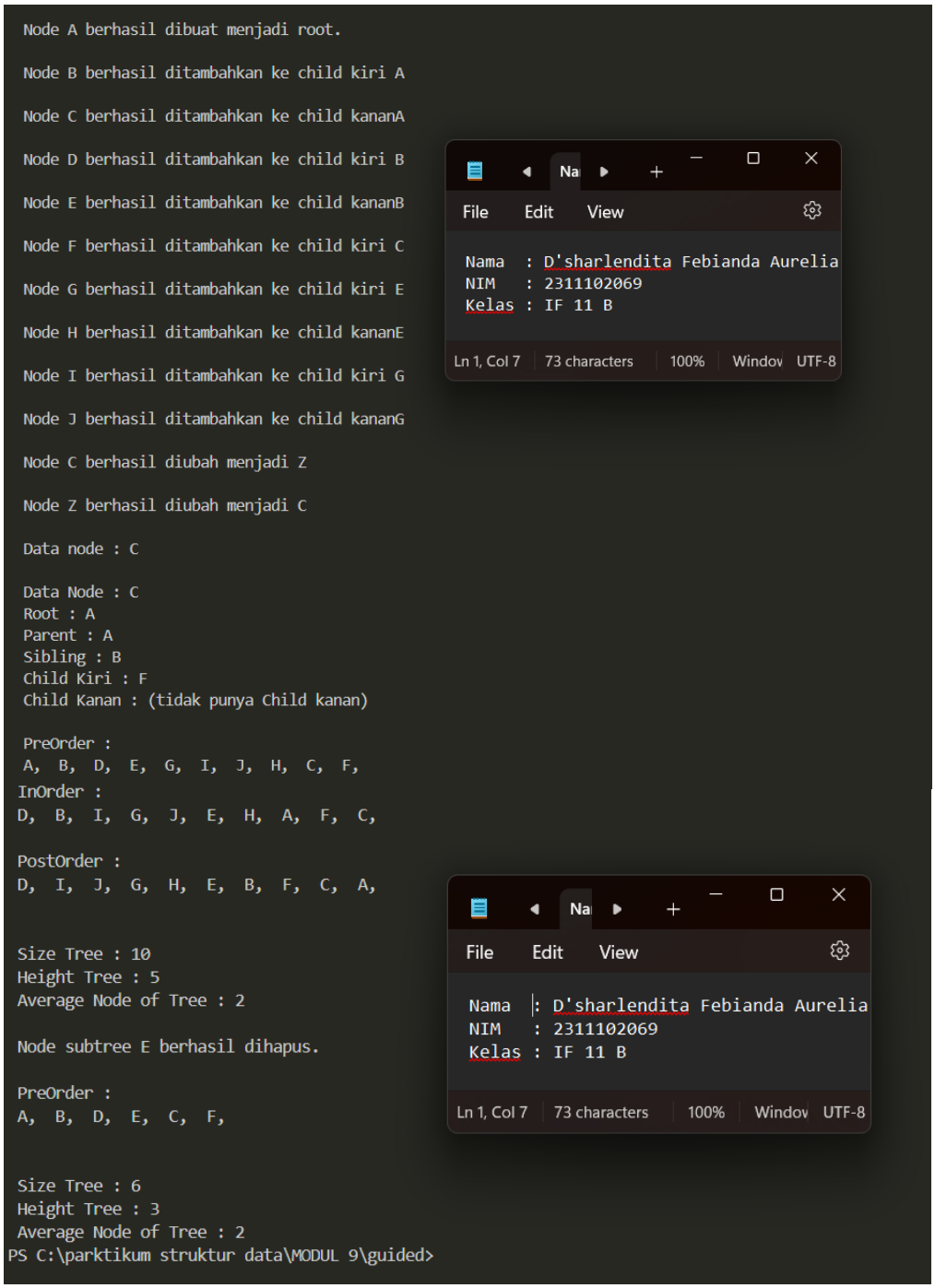
```
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                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();
}
```

Screenshot Output:



Deksripsi:

Program ini mengimplementasikan pohon biner dalam C++, dengan struktur Pohon yang menyimpan data, serta pointer ke anak kiri, anak kanan, dan parent. Fungsi-fungsi utama mencakup inisialisasi pohon, pengecekan apakah pohon kosong, pembuatan node baru, penambahan anak kiri atau kanan, pengubahan data node, penelusuran (pre-order, in-order, post-order), dan penghapusan node atau subtree. Program juga menghitung ukuran dan tinggi pohon, serta menampilkan karakteristik pohon. Di dalam main(), program membuat pohon dengan root 'A', menambahkan beberapa node, melakukan berbagai operasi, dan menghapus subtree tertentu, sambil menampilkan hasilnya.

C. Unguided

*Cantumkan NIM pada salah satu variabel di dalam program.\

Contoh : int nama_22102003

Unguided 1

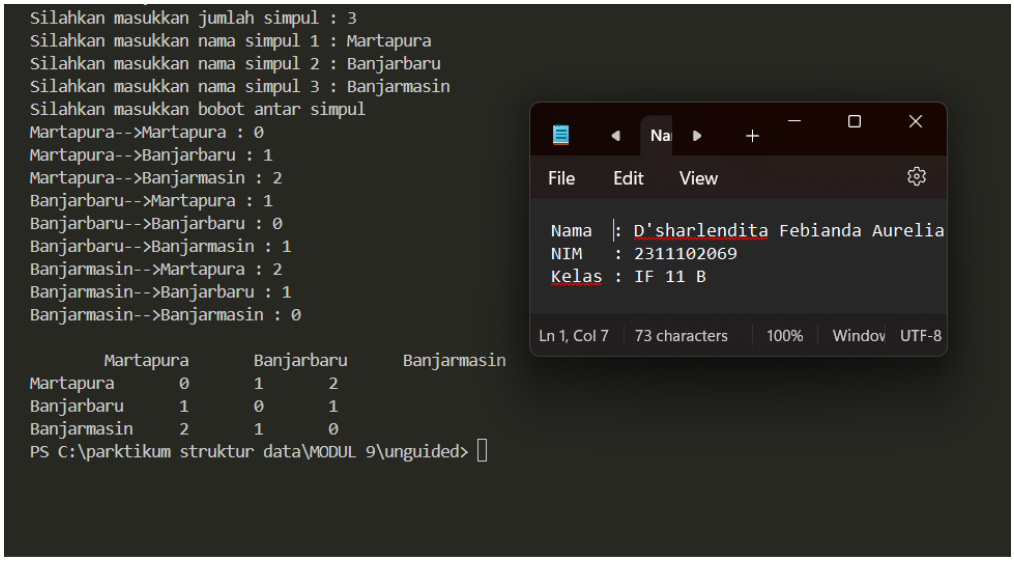
Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

Source Code :

```
//D'sharlendita Febianda Aurelia
//2311102069

// Buatlah program graph dengan menggunakan inputan user untuk
menghitung jarak dari sebuah kota ke kota lainnya

#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> jumlah_simpul;
    vector<string>
DsharlenditaFebiandaAurelia_2311102069(jumlah_simpul);
    vector<vector<int>> bobot(jumlah_simpul,
vector<int>(jumlah_simpul));
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << " :
";
        cin >> DsharlenditaFebiandaAurelia_2311102069[i];
    }
    cout << "Silahkan masukkan bobot antar simpul\n";
    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << DsharlenditaFebiandaAurelia_2311102069[i] <<
"-->" << DsharlenditaFebiandaAurelia_2311102069[j] << " : ";
            cin >> bobot[i][j];
        }
    }
    cout << "\n\t";
```

```
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << DsharlenditaFebiandaAurelia_2311102069[i] <<
"\t";
    }
    cout << "\n";
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << DsharlenditaFebiandaAurelia_2311102069[i] <<
"\t";
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }
    return 0;
}
```

Screenshots Output:



Deskripsi:

Program ini meminta pengguna untuk memasukkan jumlah simpul (kota) dan nama-nama simpul tersebut. Setelah itu, pengguna diminta untuk memasukkan bobot (jarak) antar simpul. Data tersebut disimpan dalam vektor dan matriks. Program kemudian menampilkan matriks bobot yang menunjukkan jarak antara setiap pasangan simpul. Input dan output dilakukan melalui terminal, dan matriks bobot ditampilkan dalam bentuk tabel untuk memudahkan pemahaman.

Unguided 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!

Source Code:

```
// D'sharlendita Febianda Aurelia
//2311102069

//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!

#include <iostream>
#include <vector>

using namespace std;

// Declaring the Tree structure
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root = nullptr;

// Initialize the tree
void init() {
    root = NULL;
}

// Check if the tree is empty
int isEmpty() {
    return (root == NULL) ? 1 : 0;
}

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

```
    newNode->parent = NULL;
    return newNode;
}

// Insert a node to the left
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;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kiri " << child->parent->data << endl;
            return child;
        }
    }
}

// Insert a node to the right
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;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kanan " << child->parent->data << endl;
            return child;
        }
    }
}
```

```
// Update node data
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;
        }
    }
}

// Retrieve node data
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;
        }
    }
}

// Find node and display its properties
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;
        }
    }
}

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

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

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

// Delete the entire tree
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;
        }
    }
}

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

// Clear the entire tree
void clear() {
    if (!root)
```

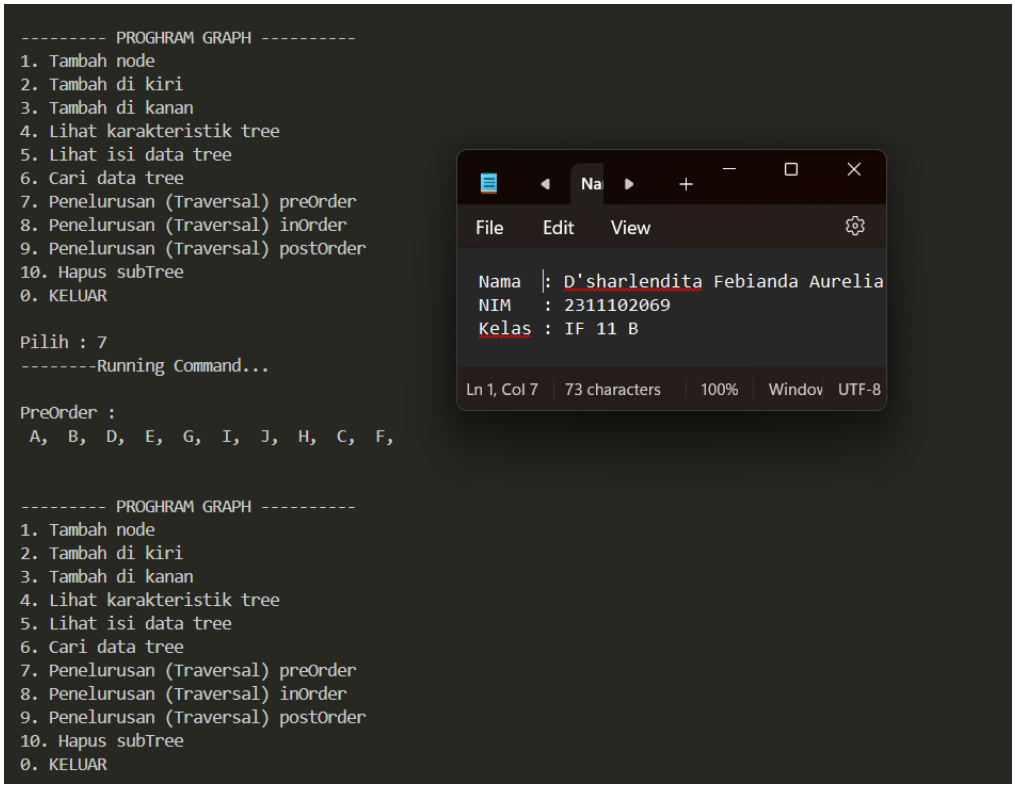
```
    cout << "\nBuat tree terlebih dahulu!!" << endl;
    else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

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

// Get the height of the tree
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;
    }
}

// Display tree characteristics
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 DsharlenditaFebiandaAurelia_2311102069;
    vector<Pohon*> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
```

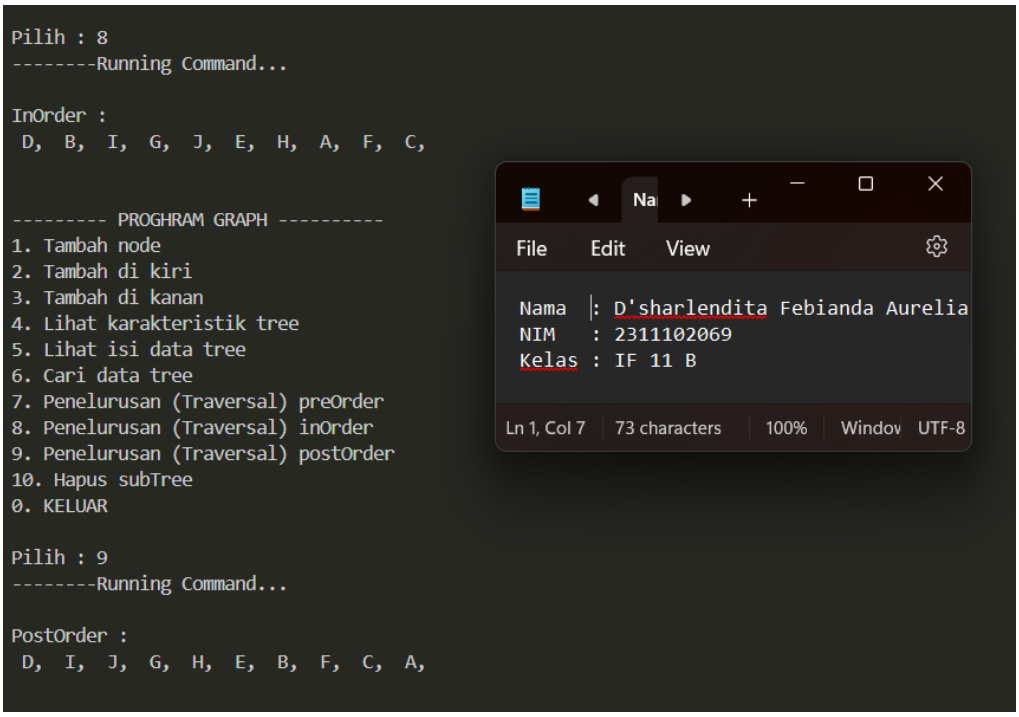
```
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));
    insertLeft(root, nodes[0]);
    insertRight(root, 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 GRAPH ----------\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 >> DsharlenditaFebiandaAurelia_2311102069;


nodes.push_back(buatNode(DsharlenditaFebiandaAurelia_2311102069))
;
                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);
}
```

Screenshot Output:





Deskripsi:

Program ini adalah implementasi dari pohon biner dengan berbagai fungsi manipulasi dan penelusuran. Pengguna dapat membuat dan memanipulasi pohon melalui menu interaktif. Program ini memungkinkan pengguna untuk menambah node, menambah node di kiri atau kanan, menampilkan karakteristik pohon, melihat isi data node, mencari node, dan melakukan penelusuran pohon (pre-order, in-order, dan post-order). Selain itu, pengguna juga dapat menghapus subtree dari node tertentu. Semua operasi dilakukan melalui antarmuka berbasis teks, dan data pohon disimpan dalam struktur pohon biner dengan setiap node memiliki pointer ke parent, left child, dan right child.

D. Kesimpulan

Praktikum Modul IX mengenai materi Graph dan Tree memberikan pemahaman mendalam mengenai struktur data graf dan pohon serta penggunaannya dalam pemrograman. Melalui praktikum ini, saya memahami konsep dasar dari kedua struktur data ini.

Graf adalah struktur data yang terdiri dari simpul-simpul yang saling terhubung melalui sisi-sisi, digunakan untuk merepresentasikan hubungan atau koneksi antar entitas. Dalam praktikum graf, saya mempelajari cara menghitung jarak antara kota-kota menggunakan input dari pengguna, yang melibatkan pembuatan matriks bobot antar simpul.

Sebaliknya, pohon adalah struktur data hirarkis yang terdiri dari simpul-simpul yang memiliki hubungan "parent-child". Pohon biner, khususnya, memiliki setiap simpul dengan maksimal dua anak. Dalam praktikum pohon, saya mengimplementasikan berbagai operasi seperti penambahan node, penelusuran (pre-order, in-order, post-order), serta manipulasi subtree. Implementasi ini memberikan wawasan tentang cara mengelola dan menavigasi data secara efisien dalam struktur hierarkis.

Dengan memahami kedua struktur data ini, saya dapat mengaplikasikan algoritma yang tepat untuk berbagai masalah pemrograman, baik yang melibatkan hubungan kompleks (graf) maupun struktur hierarkis (pohon).

E. Referensi

Asisten Praktikum. (2024, 29 Mei). “Modul 9 Graph dan Tree”. Diakses pada 29 Mei 2024, dari Learning Management System. 2024

Nurul, Ramdan. (2020, 10 November). Data Structure : Mengenal Graph & Tree.Diakses pada 11 Juni 2024, dari https://ramdannur.wordpress.com/2020/11/10/data-structure-mengenal-graph-tree/

Hadari, Ahmad. (2019, 31 Mei). Graf (Graph) dan Pohon (Tree) – Algoritma Pemprogaman 2. Diakses pada 11 Juni 2024, dari https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html