

LAPORAN PRAKTIKUM

MODUL 9 GRAPH & TREE



**Disusun oleh:
Rakha Yudhistira
NIM: 2311102010**

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

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2024**

BAB I

TUJUAN PRAKTIKUM

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

BAB II

DASAR TEORI

1. Graph

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan **verteks (V)**, sedangkan sisi yang menghubungkan antar verteks disebut **edge (E)**. Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y.

2. Tree

Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one-to-many) dan tidak linier antara elemen-elemennya.

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 D:\ITTP\Semester 2\Struktur data & Algoritma\Code\Praktikum\Modul 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 D:\ITTP\Semester 2\Struktur data & Algoritma\Code\Praktikum\Modul 9>

```

Deskripsi program

Program di atas mendefinisikan dan menampilkan representasi graf berbobot menggunakan dua array: satu untuk menyimpan nama-nama simpul (node) dan satu lagi untuk menyimpan bobot dari busur (edge) yang menghubungkan simpul-simpul tersebut. Array simpul menyimpan nama-nama lokasi, sementara matriks busur menyimpan bobot hubungan antar simpul dalam bentuk matriks ketetanggaan. Fungsi tampilGraph menampilkan graf dengan mencetak setiap simpul beserta semua simpul tetangganya yang terhubung, lengkap dengan bobot masing-masing busur. Program ini menggambarkan graf di mana setiap simpul memiliki hubungan langsung ke simpul lain jika terdapat bobot (nilai) yang tidak

nol di matriks ketetanggaan, menunjukkan arah dan bobot dari masing-masing hubungan. Ketika dijalankan, program akan mencetak setiap simpul bersama dengan simpul-simpul tetangga yang terhubung dan bobot dari busur yang menghubungkannya.

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

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

Deskripsi program

Program di atas mengimplementasikan struktur data pohon biner (binary tree) dalam C++ dengan berbagai operasi yang dapat dilakukan pada pohon tersebut. Program mendefinisikan struktur Pohon untuk node pohon yang memiliki data, pointer ke child kiri dan kanan, serta pointer ke parent. Fungsi-fungsi yang ada meliputi inisialisasi pohon, pengecekan apakah pohon kosong, penambahan node baru sebagai root, penambahan node ke child kiri dan kanan dari node yang ada, pengubahan data node, penelusuran node, dan pencarian node dengan mencetak informasi lengkap node tersebut termasuk parent dan sibling-nya jika ada. Selain itu, program juga mencakup fungsi untuk penelusuran pohon dalam urutan pre-order, in-order, dan post-order, penghapusan subtree atau seluruh pohon, serta penghitungan ukuran (size) dan tinggi (height) pohon. Pada bagian main, program membangun pohon dengan beberapa node dan menunjukkan cara penggunaan fungsi-fungsi tersebut, termasuk menampilkan traversal pohon dan karakteristik pohon sebelum dan setelah penghapusan subtree.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

using namespace std;

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    vector<string> kota(n);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> kota[i];
    }

    vector<vector<int>> graf(n, vector<int>(n));
    cout << "Silakan masukan bobot antar simpul" << endl;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) {
                graf[i][j] = 0;
            } else {
                cout << kota[i] << "-->" << kota[j] << " = ";
                cin >> graf[i][j];
            }
        }
    }
}
```

```

    }

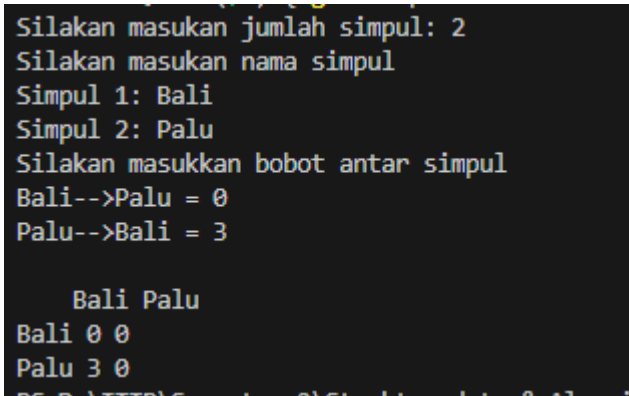
    // Output adjacency matrix
    cout << endl << "      ";
    for (int i = 0; i < n; ++i) {
        cout << kota[i] << " ";
    }
    cout << endl;

    for (int i = 0; i < n; ++i) {
        cout << kota[i] << " ";
        for (int j = 0; j < n; ++j) {
            cout << graf[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}

```

Screenshoot program



```

Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul
Simpul 1: Bali
Simpul 2: Palu
Silakan masukkan bobot antar simpul
Bali-->Palu = 0
Palu-->Bali = 3

      Bali Palu
Bali 0 0
Palu 3 0

```

Deskripsi program

Program di atas adalah sebuah implementasi dalam C++ yang memungkinkan pengguna untuk membuat graf berbobot yang direpresentasikan dengan matriks ketetanggaan (adjacency matrix). Pertama, pengguna diminta untuk memasukkan jumlah simpul yang diinginkan. Selanjutnya, pengguna memasukkan nama-nama simpul tersebut. Setelah itu, pengguna diminta untuk memasukkan bobot antar simpul, dengan input bobot antara dua simpul yang sama diatur menjadi 0 untuk menghindari loop ke diri sendiri. Matriks ketetanggaan kemudian dihasilkan dan dicetak, dengan baris dan kolom yang diberi label nama-nama simpul yang dimasukkan sebelumnya. Program ini memudahkan visualisasi dan representasi graf berbobot dengan menampilkan semua hubungan antar simpul dalam bentuk tabel yang terstruktur.

2. Unguided 2

Source code

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

struct Pohon {
    char data;
    Pohon *kiri, *kanan, *inti;
};

Pohon *root;

void init() {
    root = NULL;
}

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

```

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->kiri = NULL;
    node->kanan = NULL;
    node->inti = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi
root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertkiri(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->kiri != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->inti = node;
            node->kiri = baru;
        }
    }
}

```

```

        cout << "\nNode " << data << " berhasil ditambahkan
ke child kiri " << node->data << endl;
        return baru;
    }
}

Pohon *insertkanan(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->kanan != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->inti = node;
            node->kanan = baru;
            cout << "\nNode " << data << " berhasil ditambahkan
ke child kanan " << node->data << endl;
            return baru;
        }
    }
}

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

```

```

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

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

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

```



```

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

Pohon *findNode(Pohon *node, char data) {
    if (node == NULL) return NULL;
    if (node->data == data) return node;
    Pohon *kiriResult = findNode(node->kiri, data);
    if (kiriResult != NULL) return kiriResult;
    return findNode(node->kanan, data);
}

int main() {
    int choice;
    char data;
    init();

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Buat Node Root\n";
    }
}

```

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

```

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

        cin >> data;
        {
            Pohon *node = findNode(root, data);
            if (node != NULL) {
                displayChild(node);
            }
        }
    }
}

```

```

        } else {
            cout << "\nNode tidak ditemukan!" <<
endl;

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

return 0;
}

```

Screenshoot program

Menu:

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

Pilih opsi: 1

Masukkan data untuk root: A

Node A berhasil dibuat menjadi root.

Pilih opsi: 2

Masukkan data untuk node kiri: B

Masukkan data inti: A

Node B berhasil ditambahkan ke child kiri A

Pilih opsi: 3

Masukkan data untuk node kanan: C

Masukkan data inti: A

Node C berhasil ditambahkan ke child kanan A

Pilih opsi: 2

Masukkan data untuk node kiri: D

Masukkan data inti: C

Node D berhasil ditambahkan ke child kiri C

Pilih opsi: 3

Masukkan data untuk node kanan: E

Masukkan data inti: C

Node E berhasil ditambahkan ke child kanan C

Pilih opsi: 4

PreOrder: A, B, C, D, E,

Pilih opsi: 5

InOrder: B, A, D, C, E,

Pilih opsi: 6

PostOrder: B, D, E, C, A,

Pilih opsi: 7

Masukkan data node untuk menampilkan child: A
inti: A

Child Kiri: B

Child Kanan: C

Pilih opsi: 8

Masukkan data node untuk menampilkan descendants: C

Descendants of C: D E

Deskripsi program

Program di atas adalah implementasi dari sebuah struktur data pohon biner dalam C++ yang memungkinkan pengguna untuk melakukan berbagai operasi seperti menambah node, menampilkan traversal (pre-order, in-order, dan post-order), serta menampilkan child dan descendants dari node tertentu. Program ini menggunakan struktur Pohon untuk merepresentasikan node dalam pohon, dengan pointer ke child kiri, kanan, dan parent. Pengguna dapat memasukkan node baru sebagai root, atau sebagai child kiri atau kanan dari node yang ada. Selain itu, program ini menyediakan menu interaktif yang memudahkan pengguna dalam menavigasi dan menjalankan operasi-operasi yang tersedia pada pohon biner tersebut. Fungsi-fungsi seperti preOrder, inOrder, dan postOrder digunakan untuk penelusuran pohon, sedangkan fungsi displayChild dan displayDescendants digunakan untuk menampilkan child dan descendants dari node tertentu.

BAB IV

KESIMPULAN

Graph dan tree adalah dua struktur data penting yang sering digunakan dalam ilmu komputer dan pemrograman untuk merepresentasikan hubungan antara objek. Graph terdiri dari simpul-simpul yang saling terhubung melalui busur yang bisa berarah atau tidak berarah, berbobot atau tidak berbobot. Graph digunakan untuk berbagai aplikasi seperti jaringan komputer, pemetaan rute, dan analisis jejaring sosial. Tree adalah jenis khusus dari graph yang tidak mengandung siklus dan memiliki struktur hierarkis dengan satu root node dan child nodes yang tersusun secara berjenjang. Pohon biner, jenis khusus dari tree, membatasi setiap node memiliki maksimal dua child nodes dan digunakan dalam berbagai aplikasi seperti struktur data biner, ekspresi aritmetika, dan sistem file. Kedua struktur ini menyediakan cara yang efisien untuk mengelola dan memproses data kompleks serta mendukung berbagai algoritma penting dalam pemrograman dan ilmu komputer.

DAFTAR PUSTAKA

Asisten Praktikum. (2024). MODUL 9 GRAPH & TREE. Learning Managament System

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