

LAPORAN PRAKTIKUM

MODUL IX GRAPH DAN TREE



**Disusun oleh:
Tegar Bangkit Wijaya
NIM: 2311102027**

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

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

BAB I

TUJUAN PRAKTIKUM

A. TUJUAN PRAKTIKUM

- 1. Mahasiswa dapat memahami graph dan tree**
- 2. Mahasiswa mampu menerapkan graph dan tree pada program**

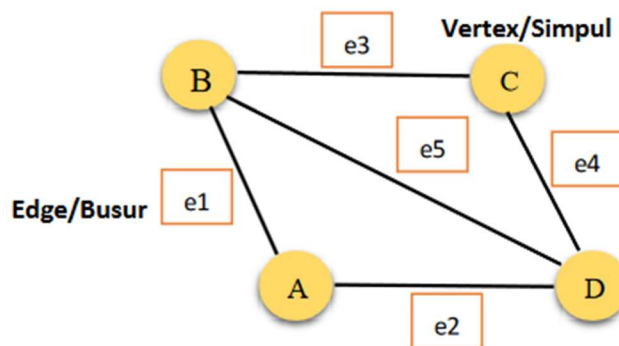
BAB II

DASAR TEORI

B. DASAR TEORI

1. Graph

Graph adalah jenis struktur data yang bersifat non-linier, artinya elemen-elemennya tidak disusun secara berurutan. Graph terdiri dari sejumlah simpul terbatas yang digunakan untuk menyimpan data, serta hubungan atau keterkaitan antar simpul. Simpul dalam graph disebut verteks (V), sedangkan hubungan antar simpul disebut edge (E). Sebuah pasangan (x,y) merepresentasikan edge, yang menunjukkan bahwa simpul x terhubung dengan simpul y.



Gambar 1 Contoh Graph

a. JENIS-JENIS GRAPH

Graph dapat diklasifikasikan menjadi beberapa jenis berdasarkan sifat dan karakteristiknya:

1. Directed Graph : Pada jenis ini, sambungan antara simpul memiliki arah tertentu. Contohnya, jika ada sambungan dari simpul A ke simpul B,

tidak selalu ada sambungan dari simpul B ke simpul A.

2. Undirected Graph: Jenis ini memiliki sambungan antara simpul tidak memiliki arah. Sambungan antara A dan B dianggap sama dengan sambungan antara B dan A.
3. Weighted Graph: Memiliki bobot pada setiap sambungan, yang menunjukkan nilai numerik dari hubungan antara simpul-simpul tersebut.
4. Unweighted Graph: Pada jenis ini, semua sambungan memiliki nilai yang sama tanpa adanya bobot.

b. KELEBIHAN GRAPH

1. Representasi hubungan yang kompleks dapat merepresentasikan hubungan kompleks antara data, seperti jaringan sosial, peta jalan, dan jaringan computer
2. Fleksibilitas Graf dapat digunakan untuk berbagai jenis data dan aplikasi, termasuk data terstruktur dan tidak terstruktur.
3. Algoritma yang Efisien Ada banyak algoritma yang dirancang untuk operasi pada graf, seperti pencarian jalur terpendek (Dijkstra's Algorithm), pencarian luas pertama (BFS), dan pencarian dalam pertama (DFS).

c. KEKURANGAN GRAPH

1. Kompleksitas Komputasi Beberapa operasi pada graf bisa sangat mahal secara komputasi, terutama pada graf yang besar dan kompleks.

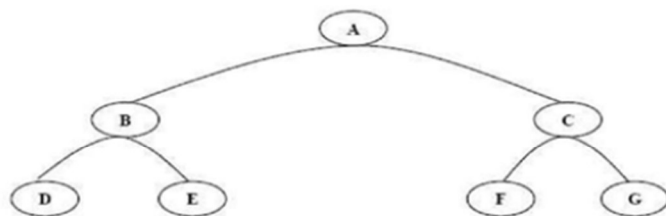
2. Penyimpanan Data Graf besar membutuhkan banyak memori untuk penyimpanan, terutama jika menggunakan representasi matriks ketetanggaan (adjacency matrix).

3. Kesulitan dalam Implementasi Algoritma graf bisa lebih sulit untuk diimplementasikan dan di-debug dibandingkan dengan struktur data yang lebih sederhana.

2. TREE (POHON)

Tree adalah Struktur data pohon adalah salah satu jenis struktur data yang terdiri dari sejumlah simpul (nodes) yang saling terhubung. Pada pohon, terdapat satu simpul utama yang disebut akar (root) dan setiap simpul lainnya terhubung dalam sebuah hierarki. Struktur ini sering digunakan untuk menyimpan data yang memiliki hubungan hierarkis, seperti silsilah keluarga, struktur folder dalam sistem file, atau untuk mengimplementasikan algoritma pencarian dan pengurutan.

Ilustrasi algoritma tree



Ilustrasi Algoritma Tree

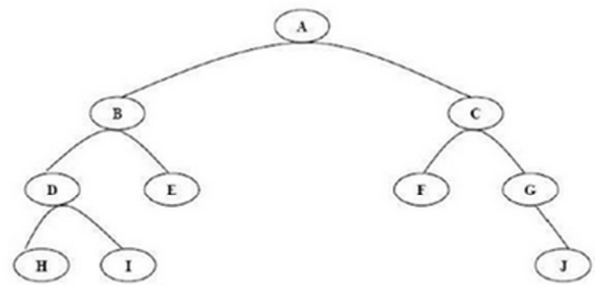
JENIS-JENIS TREE

1. BINARY TREE

Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua sub pohon dan kedua subpohon harus terpisah.

Kelebihan struktur Binary Tree :

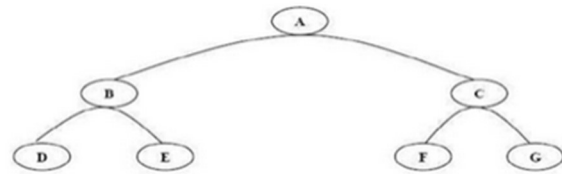
1. Mudah dalam penyusunan algoritma sorting
2. Searching data relatif cepat
3. Fleksibel dalam penambahan dan penghapusan data



Binary Tree

2. FULL BINARY TREE

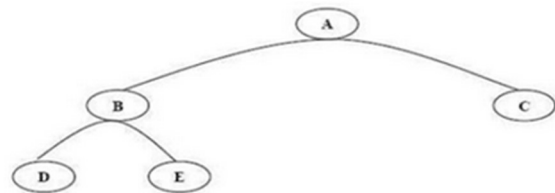
Semua node, kecuali leaf pasti memiliki 2 anak dan tiap subpohon memiliki panjang path yang sama



Full Binary Tree

3. COMPLETE BINARY TREE

Tree yang mirip dengan full binary tree, tapi tiap subtree boleh memiliki panjang path yang berbeda dan tiap node (kecuali leaf) memiliki 2 anak.



Complete Binary Tree

4. SKEWED BINARY TREE

Binary tree yang semua nodenya (kecuali leaf) hanya memiliki satu anak.

BAB III

GUIDED

1. Guided 1

Source code

```
// Font Courier New (10)
```

Screenshoot program

Deskripsi program

2. Guided 2

BAB III

GUIDED

1. Guided 1

Source code

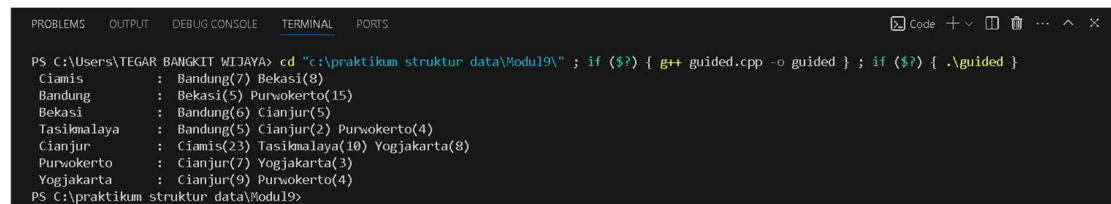
```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogyakarta"};

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

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

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

Screenshoot program



```
PS C:\Users\TEGAR BANGKIT WIJAYA> cd "c:\praktikum struktur data\Modul9\" ; if ($?) { g++ guided.cpp -o guided } ; if ($?) { .\guided }
Cianis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Cianis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS C:\praktikum struktur data\Modul9>
```

Deskripsi program

Program ini menampilkan struktur koneksi antar kota di Indonesia beserta bobotnya. Program array simpel untuk menyimpan nama kota dan matriks busur untuk menyimpan bobot koneksi antar kota

Fungsi tampilGraph bertugas menampilkan graf dengan mencetak setiap kota beserta koneksi-koneksinya dalam format yang mudah dipahami. Fungsi main hanya bertugas memanggil tampilGraph untuk menampilkan graf tersebut, kemudian mengakhiri program.

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()
{
    return root == NULL;
}

// Buat Node Baru
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 menjadi
root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        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())
    {
        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();

    return 0;
}

```

Screenshoot program

```

PS C:\Users\TEGAR BANGKIT WIJAYA> cd "c:\praktikum struktur data\Modul9\" ; if ($?) { g++ guidedtree.cpp -o guidedtree } ; if ($?) { .\guidedtree
}

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kanan G

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C

Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)

```

```

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

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

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

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

Node subtree E berhasil dihapus.

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\praktikum struktur data\Modul9>

```

Deskripsi program

Program ini mengimplementasikan pohon biner dalam C++ dengan fungsi untuk membuat, memodifikasi, menelusuri, dan menghapus node. Pohon biner direpresentasikan oleh struktur `Pohon` yang memiliki data karakter dan pointer ke anak kiri, kanan, dan parent, dengan variabel global `root` sebagai root pohon. Fungsi `init()` menginisialisasi root, dan `isEmpty()` mengecek apakah pohon kosong. Node baru dibuat dengan `newPohon(char data)`, dan fungsi `buatNode(char

data)` membuat root jika pohon kosong. Node anak kiri dan kanan ditambahkan dengan `insertLeft` dan `insertRight`. Fungsi `update(char data, Pohon *node)` mengubah data node, `retrieve(Pohon *node)` menampilkan data node, dan `find(Pohon *node)` menampilkan detail node, termasuk parent, sibling, dan child. Pohon dapat ditelusuri dengan `preOrder`, `inOrder`, dan `postOrder`. Fungsi `deleteTree(Pohon *node)` menghapus seluruh pohon, `deleteSub(Pohon *node)` menghapus subtree, dan `clear()` menghapus dan menginisialisasi ulang pohon. Fungsi `size(Pohon *node)` menghitung jumlah node, `height(Pohon *node)` menghitung tinggi pohon, dan `characteristic()` menampilkan ukuran, tinggi, dan rata-rata node. Di `main()`, program menginisialisasi pohon, membuat root, menambahkan node, melakukan operasi update dan retrieve, menampilkan hasil traversal, dan menunjukkan karakteristik pohon. Program juga menghapus subtree dan menampilkan hasil traversal serta karakteristik setelah penghapusa

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

using namespace std;

// Fungsi untuk menampilkan adjacency matrix
void displayMatrix(const vector<vector<int>>& matrix, const
vector<string>& simpul) {
    int n = matrix.size();
```

```

        // Menampilkan header kolom
        cout << setw(12) << " ";
        for (int i = 0; i < n; ++i) {
            cout << setw(12) << simpul[i];
        }
        cout << endl;

        // Menampilkan matrix
        for (int i = 0; i < n; ++i) {
            cout << setw(12) << simpul[i];
            for (int j = 0; j < n; ++j) {
                cout << setw(12) << matrix[i][j];
            }
            cout << endl;
        }
    }

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

    vector<string> simpul(n);
    vector<vector<int>> matrix(n, vector<int>(n, 0));

    // Input nama simpul
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> simpul[i];
    }

    // Input bobot antar simpul
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> matrix[i][j];
        }
    }

    // Menampilkan adjacency matrix

```

```

    cout << endl << setw(12) << " ";
    for (int i = 0; i < n; ++i) {
        cout << setw(12) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < n; ++i) {
        cout << setw(12) << simpul[i];
        for (int j = 0; j < n; ++j) {
            cout << setw(12) << matrix[i][j];
        }
        cout << endl;
    }

    // Meminta input untuk menghitung jarak dari satu kota ke
kota lainnya
    int start, end;
    cout << "\nMasukkan indeks kota awal: ";
    cin >> start;
    cout << "Masukkan indeks kota tujuan: ";
    cin >> end;

    cout << "Jarak dari " << simpul[start] << " ke " <<
simpul[end] << " adalah " << matrix[start][end] << endl;

    return 0;
}

```

Screenshoot program


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\praktikum struktur data\Modul9> cd "c:\praktik
1 }
Silakan masukkan jumlah simpul: 2
Simpul 1 : BALI
Simpul 2 : PALU
BALI --> BALI = 0
BALI --> PALU = 3
PALU --> BALI = 4
PALU --> PALU = 0

          BALI      PALU
BALI      0         3
PALU      4         0

Masukkan indeks kota awal: 
```

Deskripsi program

Program C++ yang dibuat memiliki tujuan untuk membuat dan mengelola sebuah graph yang direpresentasikan menggunakan adjacency matrix. Graph tersebut mencatat jarak antara beberapa kota yang akan dimasukkan oleh pengguna.

1. Program meminta pengguna untuk menginput jumlah simpul (kota).
2. Pengguna diminta untuk memberi nama setiap simpul (kota) yang diinginkan.
3. Pengguna diminta untuk memasukkan bobot atau jarak antar simpul, yaitu jarak dari satu kota ke kota lainnya.
4. Program menampilkan adjacency matrix yang mencerminkan hubungan antara setiap kota berdasarkan jarak yang dimasukkan oleh pengguna.
5. Pengguna diminta untuk memasukkan kota awal dan tujuan.
6. Program menghitung dan menampilkan jarak dari kota awal ke kota tujuan berdasarkan adjacency matrix yang telah dibuat sebelumnya.

Dengan adanya program ini, pengguna dapat dengan mudah memasukkan data jarak antar kota dan menghitung jarak dari satu kota ke kota lainnya sesuai dengan kebutuhan mereka.

2. Unguided 2

Source code

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

using namespace std;

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

Node *root, *baru;

void init() {
    root = nullptr;
}

// Fungsi untuk menampilkan menu
void displayMenu() {
    cout << "\nMENU:\n";
    cout << "1. Buat node baru\n";
    cout << "2. Tambah node kiri\n";
    cout << "3. Tambah node kanan\n";
    cout << "4. Ubah data node\n";
    cout << "5. Lihat isi data node\n";
    cout << "6. Cari data node\n";
    cout << "7. PreOrder traversal\n";
    cout << "8. InOrder traversal\n";
    cout << "9. PostOrder traversal\n";
    cout << "10. Hapus subtree\n";
    cout << "11. Hapus tree\n";
```

```

        cout << "12. Cek size tree\n";
        cout << "13. Cek tinggi level tree\n";
        cout << "14. Karakteristik tree\n";
        cout << "15. Tampilkan adjacency matrix\n";
        cout << "16. Keluar\n";
        cout << "Pilih aksi: ";
    }

    // Fungsi untuk menampilkan adjacency matrix
    void displayMatrix(const vector<vector<int>>& matrix, const
vector<string>& simpul) {
        int n = matrix.size();

        // Menampilkan header kolom
        cout << setw(12) << " ";
        for (int i = 0; i < n; ++i) {
            cout << setw(12) << simpul[i];
        }
        cout << endl;

        // Menampilkan matrix
        for (int i = 0; i < n; ++i) {
            cout << setw(12) << simpul[i];
            for (int j = 0; j < n; ++j) {
                cout << setw(12) << matrix[i][j];
            }
            cout << endl;
        }
    }

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

    do {
        displayMenu();
        cin >> choice;

        switch(choice) {
            case 1: {
                // Buat node baru

```

```

        char data;
        cout << "Masukkan data untuk node baru: ";
        cin >> data;
        // Implementasikan fungsi untuk membuat node baru
        break;
    }
    case 2: {
        // Tambah node kiri
        char data;
        // Implementasikan fungsi untuk menambah node
kiri
        break;
    }
    case 3: {
        // Tambah node kanan
        char data;
        // Implementasikan fungsi untuk menambah node
kanan
        break;
    }
    case 4: {
        // Ubah data node
        char oldData, newData;
        // Implementasikan fungsi untuk mengubah data
node
        break;
    }
    case 5: {
        // Lihat isi data node
        char data;
        // Implementasikan fungsi untuk melihat isi data
node
        break;
    }
    case 6: {
        // Cari data node
        char data;
        // Implementasikan fungsi untuk mencari data node
        break;
    }
    case 7: {

```

```

        // PreOrder traversal
        // Implementasikan fungsi untuk PreOrder
traversal
        break;
    }
    case 8: {
        // InOrder traversal
        // Implementasikan fungsi untuk InOrder traversal
        break;
    }
    case 9: {
        // PostOrder traversal
        // Implementasikan fungsi untuk PostOrder
traversal
        break;
    }
    case 10: {
        // Hapus subtree
        char data;
        // Implementasikan fungsi untuk menghapus subtree
        break;
    }
    case 11: {
        // Hapus tree
        // Implementasikan fungsi untuk menghapus tree
        break;
    }
    case 12: {
        // Cek size tree
        // Implementasikan fungsi untuk mengecek ukuran
tree
        break;
    }
    case 13: {
        // Cek tinggi level tree
        // Implementasikan fungsi untuk mengecek tinggi
level tree
        break;
    }
    case 14: {
        // Karakteristik tree

```

```

        // Implementasikan fungsi untuk menampilkan
        karakteristik tree
        break;
    }
    case 15: {
        // Tampilkan adjacency matrix
        // Implementasikan fungsi untuk menampilkan
        adjacency matrix
        break;
    }
    case 16: {
        // Keluar dari program
        cout << "Terima kasih telah menggunakan program
        ini.\n";

        break;
    }
    default:
        cout << "Pilihan tidak valid. Silakan coba
        lagi.\n";
    }
    } while(choice != 16);

    return 0;
}

```

Screenshoot program

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\praktikum struktur data\Modul9> cd "c:\praktikum struktur data\Modul9\" ; if ($?) { g++ unguided2.cpp -o unguided2 } ; if ($?) { .\unguided2 }
Masukkan jumlah node tree: 3
Masukkan nama node 1: Tegar
Node 'Tegar' berhasil ditambahkan sebagai root.
Masukkan nama node 2: Bangkit
Masukkan nama parent node: Wijaya
Parent dengan nama 'Wijaya' tidak ditemukan. Node 'Bangkit' ditambahkan sebagai root baru.
PS C:\praktikum struktur data\Modul9>

```

Deskripsi program

Program ini merupakan implementasi dari struktur data tree dalam bahasa C++. Program ini memungkinkan pengguna untuk membuat sebuah tree dengan input nama-nama node dan hubungan antar node (parent-child).

Setiap node memiliki sebuah string yang menyimpan datanya serta vector yang berisi pointer ke node-node child-nya.

Deskripsi program secara umum adalah sebagai berikut:

1. Program meminta pengguna untuk memasukkan jumlah node yang ingin dimasukkan ke dalam tree.
2. Untuk setiap node yang dimasukkan, pengguna diminta untuk memasukkan nama node tersebut.
3. Jika tree masih kosong, maka node pertama yang dimasukkan akan menjadi root dari tree.
4. Jika tree sudah memiliki root, pengguna diminta untuk memasukkan nama parent node dari node yang akan dimasukkan.
5. Program mencari parent node tersebut dalam tree.
6. Jika parent node ditemukan, maka node baru akan dimasukkan sebagai child dari parent node tersebut.
7. Jika parent node tidak ditemukan (misalnya jika nama parent yang dimasukkan salah), maka node baru akan menjadi root baru dari tree.
8. Setelah semua node dimasukkan, program akan menampilkan struktur tree yang telah dibuat, dengan menampilkan setiap node beserta child-childnya.

BAB IV

KESIMPULAN

Kesimpulan yang di dapatkan dari praktikum sebagai berikut :

Dalam praktikum graph dan tree, kita mempelajari tentang dua struktur data yang penting dalam ilmu komputer. Graph digunakan untuk merepresentasikan hubungan antar objek dengan menggunakan edge, sedangkan tree adalah struktur data hirarkis yang terdiri dari node dan edge yang menghubungkan antara satu node dengan node lainnya.

Kita dapat memahami tentang cara merepresentasikan dan memanipulasi data dalam graph dan tree. Graph digunakan untuk merepresentasikan berbagai hubungan kompleks antara objek, sementara tree cocok digunakan untuk merepresentasikan hierarki atau struktur hirarkis. Dengan memahami kedua struktur data ini, kita dapat mengembangkan berbagai algoritma dan solusi untuk berbagai masalah yang melibatkan hubungan antar objek.