LAPORAN PRAKTIKUM

MODUL 14

GRAPH

Nama :

Candra Dinata (2311104061)

Kelas :

S1SE 07 02

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Tujuan pembelajaran praktikum tentang graph adalah untuk memberikan pemahaman mendalam kepada mahasiswa mengenai konsep dasar teori graph, termasuk komponen-komponen seperti simpul (nodes), sisi (edges), derajat, dan berbagai jenis graph seperti graph berarah (directed graph), graph tak berarah (undirected graph), serta graph berbobot (weighted graph). Praktikum ini juga bertujuan untuk melatih kemampuan mahasiswa dalam merepresentasikan graph menggunakan berbagai metode seperti matriks ketetanggaan (adjacency matrix), daftar ketetanggaan (adjacency list), dan struktur data lainnya yang sesuai. Selain itu, mahasiswa akan diajak untuk mengimplementasikan algoritma-algoritma penting dalam graph, seperti algoritma pencarian jalur terpendek (Shortest Path), algoritma pencarian dalam graph (DFS dan BFS), serta algoritma untuk minimum spanning tree (MST) seperti Kruskal dan Prim. Melalui praktikum ini, diharapkan mahasiswa dapat menganalisis dan memecahkan permasalahan nyata yang melibatkan graph, meningkatkan keterampilan pemrograman, serta memahami bagaimana graph dapat diterapkan dalam berbagai bidang seperti jaringan komputer, pengolahan data, dan pengembangan sistem kompleks.

II. DASAR TEORI

Teori graph adalah cabang dari matematika diskret yang membahas tentang hubungan antara objek yang direpresentasikan sebagai simpul (nodes) dan sisi (edges). Sebuah graph terdiri dari himpunan simpul dan sisi yang dapat menghubungkan pasangan simpul, di mana sisi tersebut dapat bersifat berarah (directed) atau tidak berarah (undirected). Graph juga dapat memiliki bobot (weighted graph), yaitu nilai tertentu yang melekat pada setiap sisi untuk merepresentasikan biaya atau jarak. Representasi graph sering dilakukan melalui matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list), yang masing-masing memiliki kelebihan tergantung pada kebutuhan penyimpanan dan kecepatan akses data. Konsep dasar ini menjadi fondasi bagi berbagai algoritma penting seperti pencarian jalur terpendek, penelusuran graph (DFS dan BFS), serta penghitungan minimum spanning tree (MST). Teori graph digunakan secara luas dalam berbagai bidang, termasuk jaringan komputer, analisis data, pemodelan sistem, dan optimasi logistik.

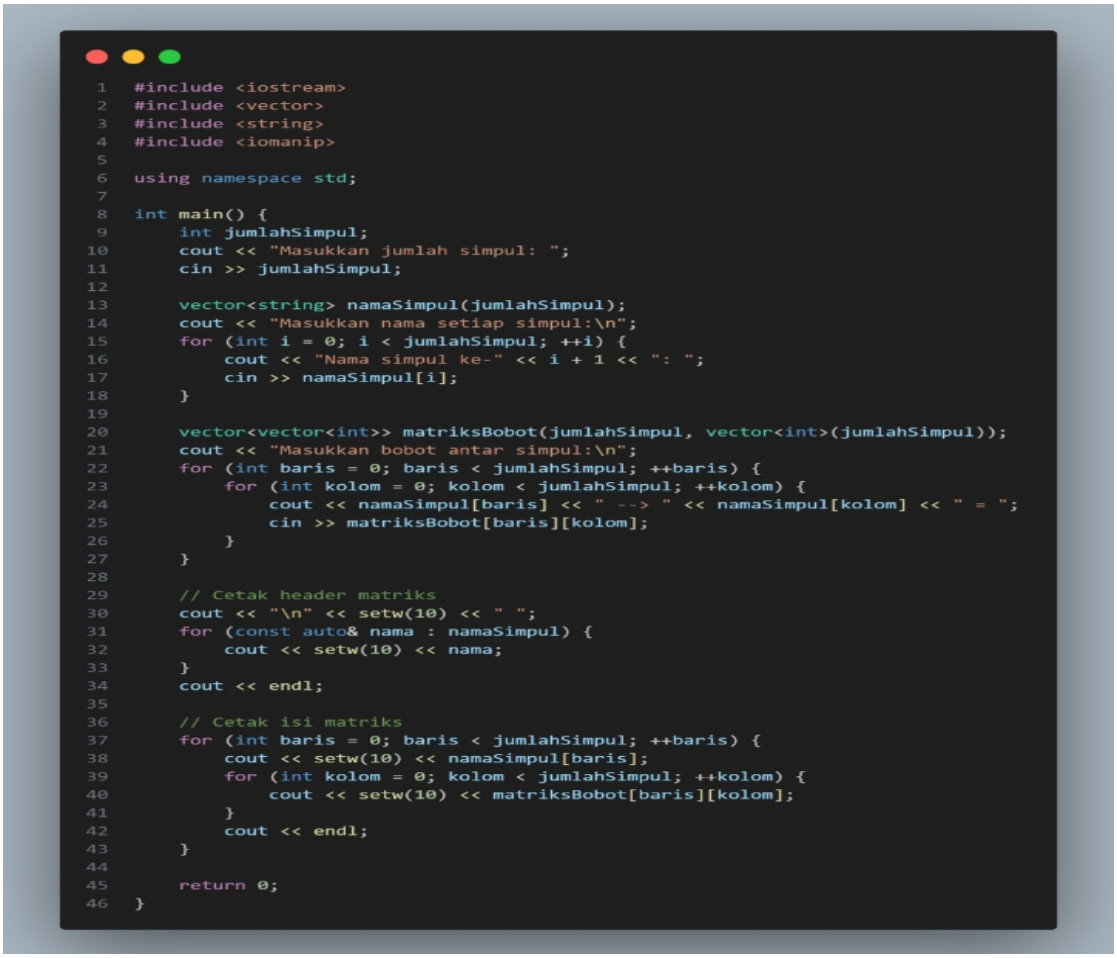
III. GUIDED

1.

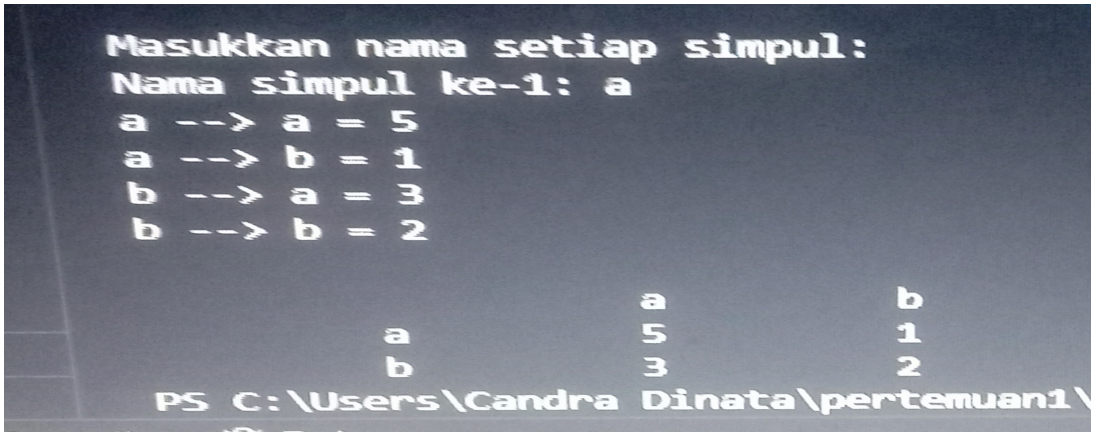
Kode di atas merupakan implementasi struktur data graph menggunakan adjacency list untuk merepresentasikan hubungan antar node (simpul) dan mendukung traversal menggunakan algoritma Depth First Search (DFS) dan Breadth First Search (BFS). Struktur Graph terdiri dari simpul awal (first), sementara simpul direpresentasikan oleh struktur ElmNode yang memiliki atribut informasi (info), penanda kunjungan (visited), daftar sisi pertama (firstEdge), dan simpul berikutnya (Next). Struktur ElmEdge digunakan untuk merepresentasikan sisi yang menghubungkan simpul-simpul dalam graph. Fungsi CreateGraph digunakan untuk menginisialisasi graph kosong, sedangkan InsertNode menambahkan simpul baru ke graph, dan ConnectNode membuat hubungan (edge) antar simpul. Fungsi ResetVisited mengatur ulang status kunjungan semua simpul untuk memastikan traversal dapat diulang. Traversal DFS dilakukan dengan fungsi rekursif PrintDFS, sedangkan traversal BFS memanfaatkan struktur antrian (queue) pada fungsi PrintBFS. Pada main(), graph dengan simpul-simpul A hingga H dibuat dan dihubungkan sesuai dengan struktur yang ditentukan, kemudian traversal DFS dan BFS dilakukan mulai dari simpul A, dengan hasil traversal dicetak ke layar. Kode ini memberikan pemahaman mendalam tentang manipulasi dan eksplorasi graph secara manual menggunakan struktur data dasar di C++.

IV. UNGUIDED

1.

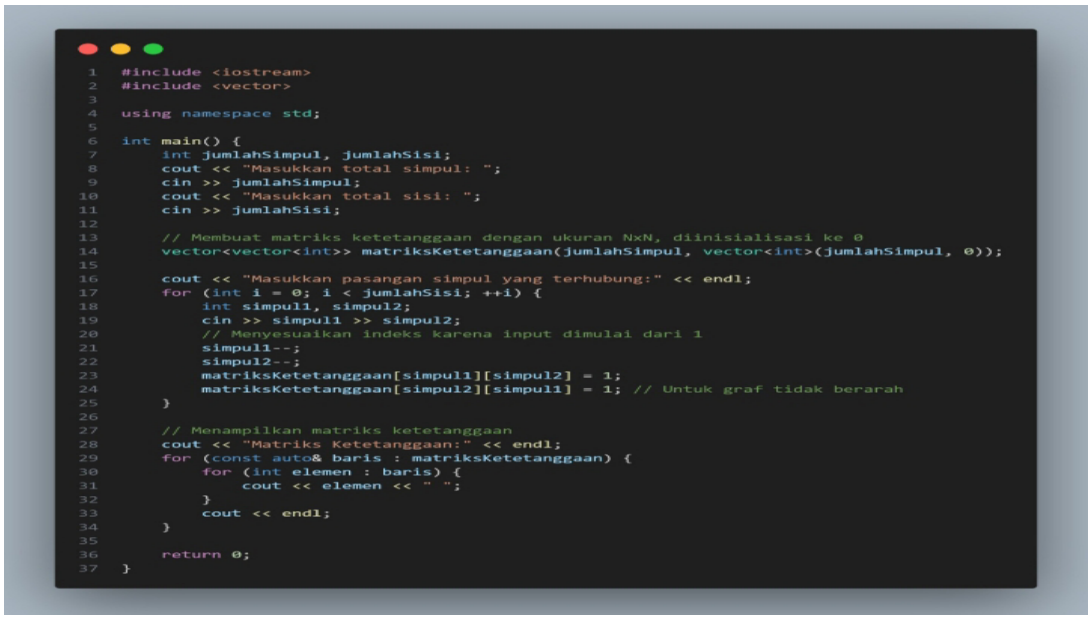


```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

using namespace std;

int main() {
    int jumlahSimpul;
    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> namaSimpul(jumlahSimpul);
    cout << "Masukkan nama setiap simpul:\n";
    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << "Nama simpul ke-" << i + 1 << ": ";
        cin >> namaSimpul[i];
    }

    vector<vector<int>> matriksBobot(jumlahSimpul, vector<int>(jumlahSimpul));
    cout << "Masukkan bobot antar simpul:\n";
    for (int baris = 0; baris < jumlahSimpul; ++baris) {
        for (int kolom = 0; kolom < jumlahSimpul; ++kolom) {
            cout << namaSimpul[baris] << " --> " << namaSimpul[kolom] << " = ";
            cin >> matriksBobot[baris][kolom];
        }
    }

    // Cetak header matriks
    cout << "\n" << setw(10) << " ";
    for (const auto& nama : namaSimpul) {
        cout << setw(10) << nama;
    }
    cout << endl;

    // Cetak isi matriks
    for (int baris = 0; baris < jumlahSimpul; ++baris) {
        cout << setw(10) << namaSimpul[baris];
        for (int kolom = 0; kolom < jumlahSimpul; ++kolom) {
            cout << setw(10) << matriksBobot[baris][kolom];
        }
        cout << endl;
    }

    return 0;
}
```

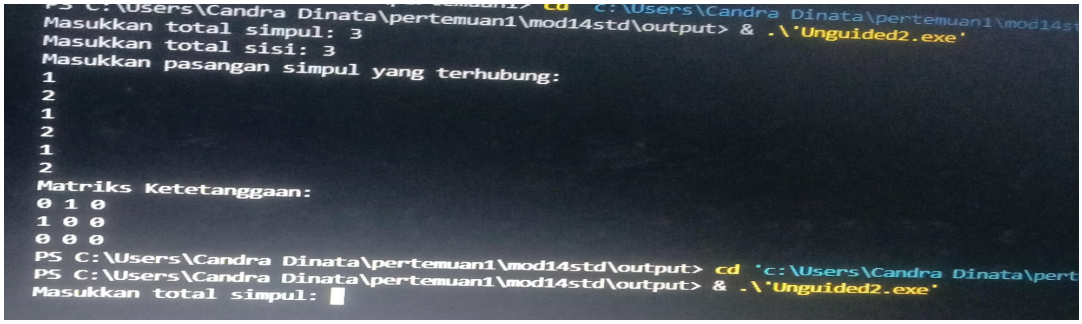


Kode di atas adalah program C++ yang meminta pengguna untuk memasukkan jumlah simpul (node) dalam sebuah graf, nama-nama simpul, dan bobot hubungan antar simpul. Pertama, program meminta pengguna memasukkan jumlah simpul, lalu menyimpan nama-nama simpul tersebut dalam sebuah vektor bernama namaSimpul. Selanjutnya, program menggunakan matriks berbentuk vektor dua dimensi matriksBobot untuk merepresentasikan hubungan (edge) antar simpul dengan bobotnya, yang dimasukkan oleh pengguna. Bobot hubungan dari setiap simpul ke simpul lainnya dimasukkan dalam loop bersarang, di mana nama simpul digunakan untuk membantu pengguna memahami relasi yang dimaksud. Setelah semua data dimasukkan, program mencetak matriks bobot dalam bentuk tabel yang rapi. Nama simpul dicetak sebagai header baris dan kolom, sementara elemen matriks mewakili bobot antar simpul. Penggunaan fungsi seperti setw dari pustaka <iomanip> memastikan output terlihat terformat dengan baik dan mudah dibaca.

2.



```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int jumlahSimpul, jumlahSisi;
    cout << "Masukkan total simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan total sisi: ";
    cin >> jumlahSisi;

    // Membuat matriks ketetanggaan dengan ukuran NxN, diinisialisasi ke 0
    vector<vector<int>> matriksKetetanggaan(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    cout << "Masukkan pasangan simpul yang terhubung:" << endl;
    for (int i = 0; i < jumlahSisi; ++i) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;
        // Menyesuaikan indeks karena input dimulai dari 1
        simpul1--;
        simpul2--;
        matriksKetetanggaan[simpul1][simpul2] = 1;
        matriksKetetanggaan[simpul2][simpul1] = 1; // Untuk graf tidak berarah
    }

    // Menampilkan matriks ketetanggaan
    cout << "Matriks Ketetanggaan:" << endl;
    for (const auto& baris : matriksKetetanggaan) {
        for (int elemen : baris) {
            cout << elemen << " ";
        }
        cout << endl;
    }

    return 0;
}
```



Kode di atas adalah implementasi representasi graf menggunakan matriks ketetanggaan (adjacency matrix) di C++. Program dimulai dengan meminta input jumlah simpul (jumlahSimpul) dan jumlah sisi (jumlahSisi) dari pengguna. Matriks ketetanggaan diinisialisasi menggunakan struktur data vector dua dimensi dengan ukuran NxN, di mana semua elemen awalnya diatur ke 0 untuk merepresentasikan tidak adanya koneksi antar simpul. Selanjutnya, program meminta pengguna untuk memasukkan pasangan simpul yang terhubung, di mana setiap koneksi diperbarui dalam matriks dengan nilai 1, baik untuk simpul1 ke simpul2 maupun sebaliknya, karena graf dianggap tidak berarah. Setelah semua sisi dimasukkan, program mencetak matriks ketetanggaan, di mana setiap baris dan kolom merepresentasikan simpul, dan elemen bernilai 1 menunjukkan adanya hubungan antara dua simpul tersebut. Kode ini mempermudah visualisasi struktur graf kecil melalui pendekatan berbasis matriks.

V. KESIMPULAN

Dari praktikum tentang graph menggunakan bahasa C++, dapat disimpulkan bahwa graph adalah struktur data yang sangat berguna untuk merepresentasikan hubungan antar objek dalam berbagai konteks. Representasi graph dapat dilakukan menggunakan adjacency list atau adjacency matrix, masing-masing memiliki kelebihan tergantung pada efisiensi penyimpanan dan operasi pencarian. Implementasi adjacency list lebih hemat memori untuk graph yang jarang (sparse graph), sementara adjacency matrix mempermudah akses langsung ke koneksi antar simpul tetapi memerlukan lebih banyak memori. Dalam praktiknya, traversal graph seperti Depth First Search (DFS) dan Breadth First Search (BFS) sangat penting untuk eksplorasi semua simpul yang terhubung dan menemukan pola tertentu dalam graph. Melalui implementasi algoritma dan pengelolaan

data graph, kita dapat memahami konsep dasar teori graph serta bagaimana struktur ini diterapkan dalam pemrograman untuk memecahkan berbagai permasalahan seperti jalur terpendek, jaringan, dan optimasi.