LAPORAN PRAKTIKUM
PERTEMUAN 14
GRAPH

Nama :

Haza Zaidan Zidna Fann

(2311104056)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

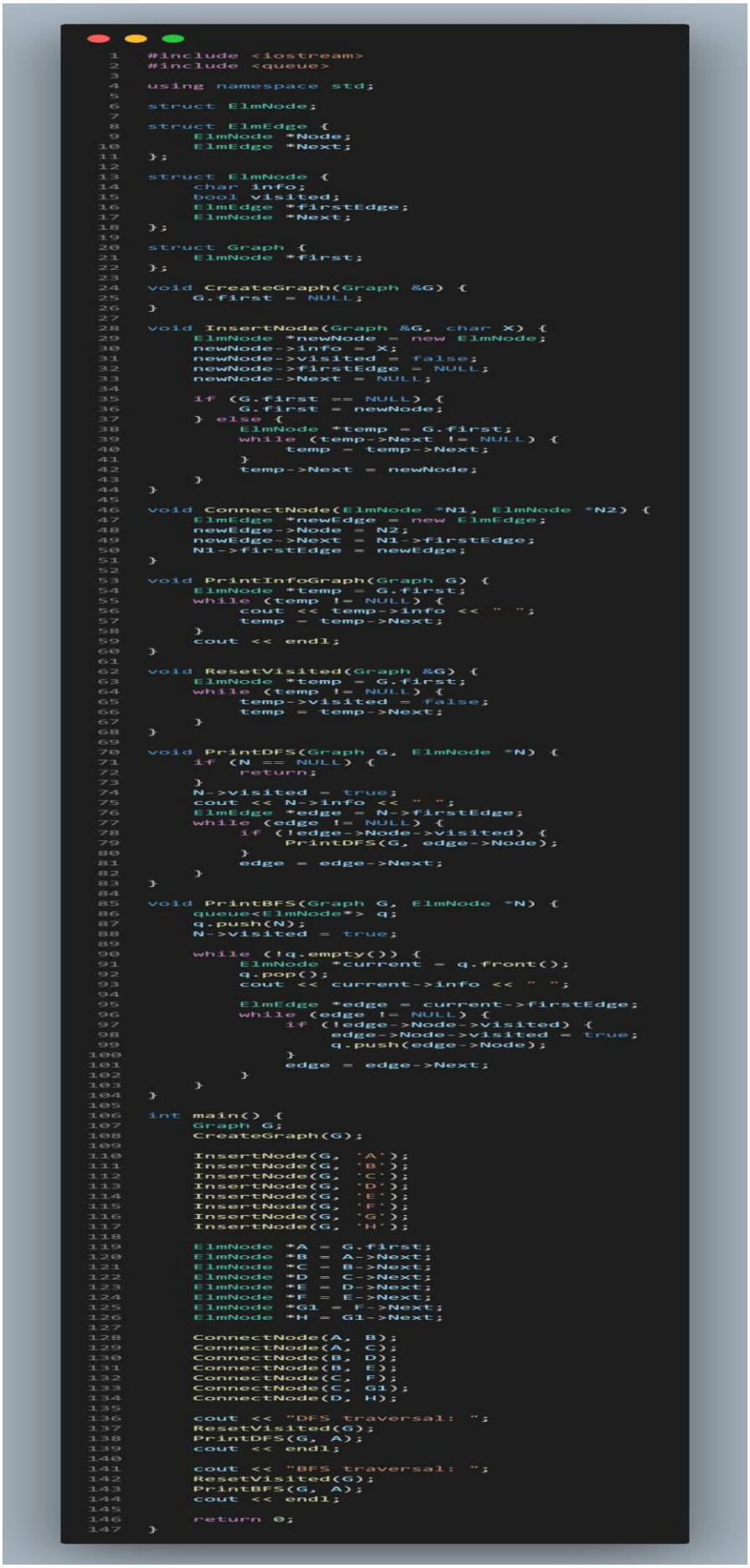
I. TUJUAN

Memahami konsep graph
Mengimplementasikan graph dengan menggunakan pointer

II. LANDASAN TEORI

Graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek-objek dalam bentuk simpul (node) dan sisi (edge). Sisi bisa berbobot atau tidak, dan dapat bersifat terarah (directed) atau tidak terarah (undirected). Selain itu, graf juga bisa mengandung siklus atau tidak. Operasi dasar yang dapat dilakukan pada graf termasuk pencarian jalur antar simpul, penambahan atau penghapusan simpul dan sisi, serta perhitungan jarak terpendek. Implementasi graf bisa menggunakan matriks ketetanggaan, daftar ketetanggaan, atau struktur lain yang lebih efisien tergantung pada kebutuhan aplikasi. Graf sering diterapkan dalam pemecahan masalah algoritma, seperti dalam jaringan komputer, analisis data sosial, dan algoritma pencarian.

III. GUIDED



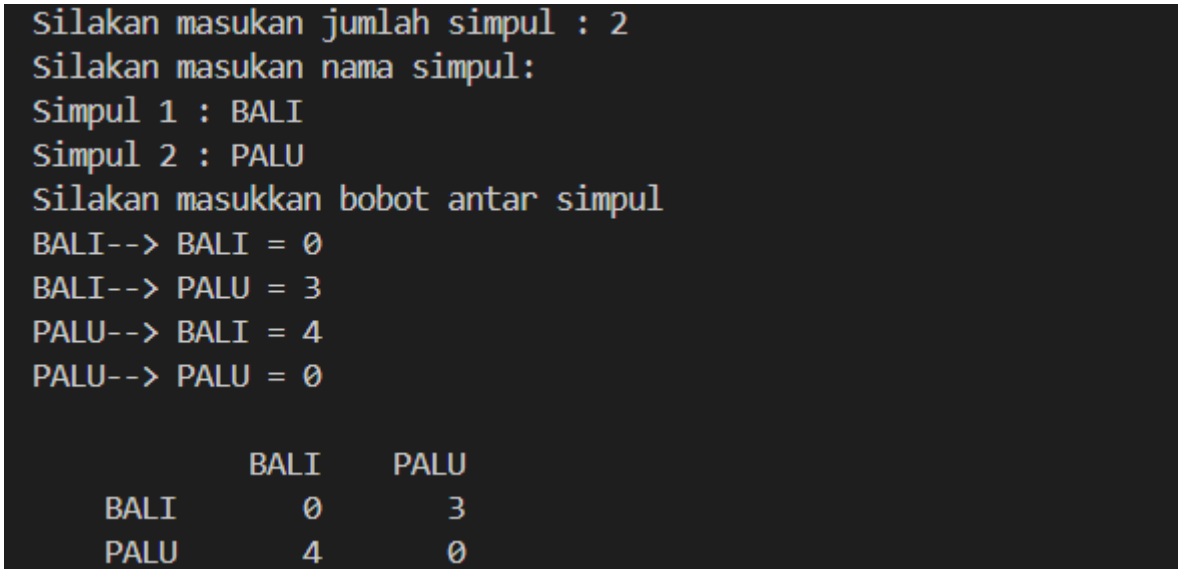
```cpp
#include <iostream>
#include <queue>

using namespace std;

struct ElmNode;

struct ElmEdge {
    ElmNode *Node;
    ElmEdge *Next;
};

struct ElmNode {
    char info;
    bool visited;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
};

void CreateGraph(Graph &G) {
    G.first = NULL;
}

void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    newNode->visited = false;
    newNode->firstEdge = NULL;
    newNode->Next = NULL;

    if (G.first == NULL) {
        G.first = newNode;
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next;
        }
        temp->Next = newNode;
    }
}

void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->firstEdge;
    N1->firstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " ";
        temp = temp->Next;
    }
    cout << endl;
}

void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false;
        temp = temp->Next;
    }
}

void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return;
    }
    N->visited = true;
    cout << N->info << " ";
    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node);
        }
        edge = edge->Next;
    }
}

void PrintBFS(Graph G, ElmNode *N) {
    queue<ElmNode*> q;
    q.push(N);
    N->visited = true;

    while (!q.empty()) {
        ElmNode *current = q.front();
        q.pop();
        cout << current->info << " ";

        ElmEdge *edge = current->firstEdge;
        while (edge != NULL) {
            if (!edge->Node->visited) {
                edge->Node->visited = true;
                q.push(edge->Node);
            }
            edge = edge->Next;
        }
    }
}

int main() {
    Graph G;
    CreateGraph(G);

    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    ElmNode *A = G.first;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);

    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

Kode ini adalah implementasi graf menggunakan struktur data linked list untuk simpul dan sisi, dengan dua algoritma pencarian: DFS (Depth-First Search) dan BFS (Breadth-First Search).
Graf dibuat dengan tipe data Graph yang berisi pointer ke simpul pertama. Fungsi InsertNode menambah simpul baru dengan informasi yang ditentukan, dan ConnectNode menghubungkan dua simpul dengan sisi. Fungsi PrintDFS dan PrintBFS digunakan untuk melakukan pencarian graf secara mendalam (DFS) atau secara luas (BFS), dengan memanfaatkan traversal rekursif dan antrian. Fungsi ResetVisited mengatur ulang status kunjungan pada setiap simpul sebelum pencarian dilakukan.
Graf yang dibangun terdiri dari simpul-simpul yang dihubungkan sesuai dengan data yang diberikan, kemudian dilakukan pencarian DFS dan BFS untuk menampilkan urutan traversal.

IV. UNGUIDED

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

using namespace std;

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

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

    vector<vector<int>> graph(V, vector<int>(V, 0));
    cout << "Silakan masukkan bobot antar simpul \n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout << cities[i] << "--> " << cities[j] << " = ";
            cin >> graph[i][j];
        }
    }

    // Print header for the matrix
    cout << "\n" << setw(8) << " ";
    for (int i = 0; i < V; i++) {
        cout << setw(8) << cities[i];
    }
    cout << endl;

    // Print matrix
    for (int i = 0; i < V; i++) {
        cout << setw(8) << cities[i];
        for (int j = 0; j < V; j++) {
            cout << setw(8) << graph[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

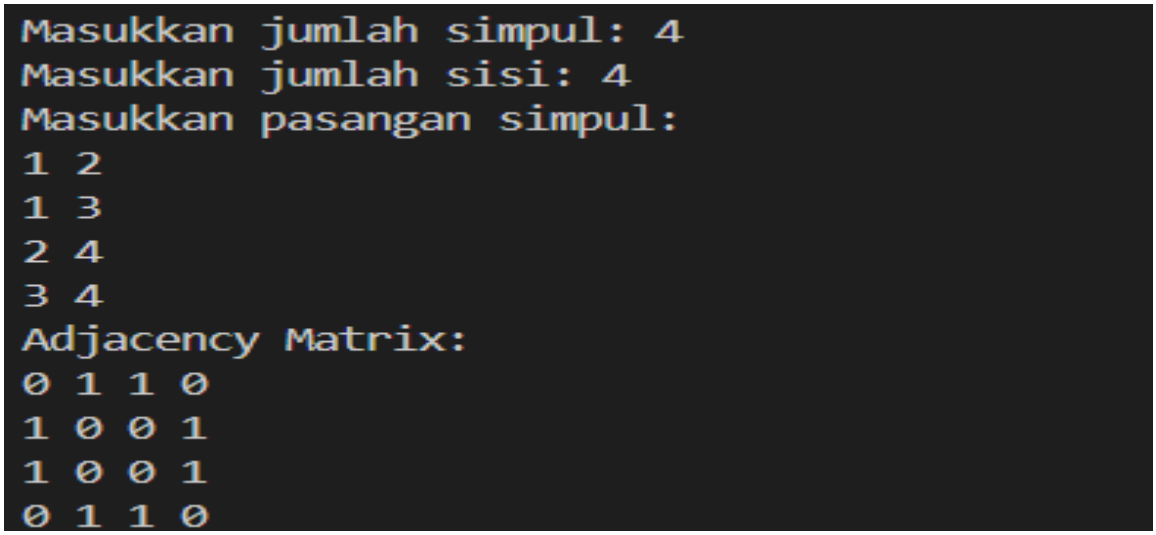


Kode ini bertujuan untuk membuat dan menampilkan graf berbobot dalam bentuk matriks ketetanggaan. Dimulai dengan meminta input jumlah simpul, diikuti dengan nama simpul-simpul yang akan dimasukkan ke dalam vektor cities. Kemudian, pengguna diminta untuk memasukkan bobot antar simpul, yang disimpan dalam matriks dua dimensi graph, di mana setiap elemen matriks mewakili bobot sisi yang menghubungkan dua simpul.

Setelah input selesai, program mencetak matriks ketetanggaan, yang mewakili hubungan antar simpul dalam graf. Fungsi setw digunakan untuk menata tampilan output agar rapi dengan memberikan lebar kolom yang konsisten saat menampilkan nama simpul dan bobotnya.

Secara keseluruhan, kode ini berfungsi untuk membangun graf berbobot yang dapat digunakan untuk berbagai aplikasi, seperti algoritma pencarian jalur terpendek atau analisis graf lainnya**.**

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

using namespace std;

int main() {
    int V, E;
    cout << "Masukkan jumlah simpul: ";
    cin >> V;
    cout << "Masukkan jumlah sisi: ";
    cin >> E;

    // Membuat adjacency matrix dengan ukuran VxV dan menginisialisasi dengan 0
    vector<vector<int>> adjacencyMatrix(V, vector<int>(V, 0));

    cout << "Masukkan pasangan simpul:" << endl;
    for (int i = 0; i < E; i++) {
        int u, v;
        cin >> u >> v;
        // Karena simpul mulai dari 1, bukan 0, kurangi 1 untuk indeks matriks
        u--;
        v--;
        adjacencyMatrix[u][v] = 1;
        adjacencyMatrix[v][u] = 1; // Graf tidak berarah
    }

    // Menampilkan adjacency matrix
    cout << "Adjacency Matrix:" << endl;
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout << adjacencyMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```



Kode ini membuat graf tidak berarah menggunakan matriks ketetanggaan (adjacency matrix). Pengguna diminta memasukkan jumlah simpul (V) dan jumlah sisi (E), lalu memasukkan pasangan simpul yang dihubungkan oleh sisi. Matriks ketetanggaan diinisialisasi dengan nilai 0, dan setiap sisi yang dimasukkan mengubah nilai elemen terkait menjadi 1. Karena grafnya tidak berarah, matriks simetris (nilai di elemen [u][v] dan [v][u] diubah menjadi 1). Program kemudian menampilkan matriks ketetanggaan yang menggambarkan hubungan antar simpul.

V. KESIMPULAN

Graf digunakan untuk menggambarkan hubungan antar objek, di mana setiap objek diwakili sebagai simpul (node) dan hubungan antar objek sebagai sisi (edge). Terdapat graf terarah (arah sisi jelas) dan tidak terarah (sisi tanpa arah). Graf bermanfaat dalam berbagai aplikasi seperti pemodelan jaringan, pencarian jalur terpendek, dan analisis hubungan antar objek. Representasi graf umumnya menggunakan matriks ketetanggaan atau daftar ketetanggaan.