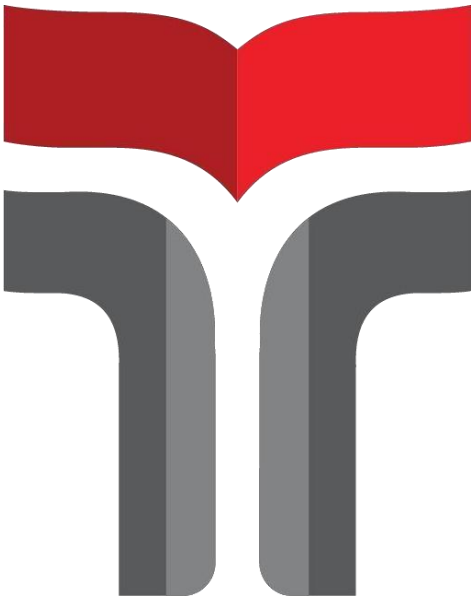
LAPORAN PRAKTIKUM
STRUKTUR DATA 14
"GRAPH"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 02

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2023/2024

I. TUJUAN

Dalam laporan ini, tujuan praktikum adalah untuk memahami, mengimplementasikan, dan menganalisis struktur data graf pada bahasa pemrograman C++. Adapun rincian tujuan yang ingin dicapai meliputi:

- Pemahaman Konsep Dasar Graph Peserta praktikum diharapkan mampu memahami konsep dasar graf, termasuk definisi, representasi (matriks ketetanggaan dan daftar ketetanggaan), serta klasifikasi graf berdasarkan sifat-sifatnya (berarah, tidak berarah, berbobot, dan tidak berbobot).

- Implementasi Graph di C++ Praktikan akan mempelajari cara mengimplementasikan graf menggunakan bahasa C++, baik melalui representasi matriks ketetanggaan maupun daftar ketetanggaan. Peserta juga diharapkan mampu mengembangkan fungsi-fungsi utama graf, seperti penambahan simpul, penambahan sisi, serta pencarian tetangga suatu simpul.

- Pemecahan Masalah dengan Algoritma pada Graph Peserta akan mempraktikkan beberapa algoritma populer yang berhubungan dengan graf, seperti Breadth-First Search (BFS), Depth-First Search (DFS), algoritma Dijkstra untuk pencarian jalur terpendek, dan algoritma Kruskal atau Prim untuk mencari Minimum Spanning Tree.

- Analisis Kompleksitas Algoritma Dalam laporan, peserta diminta untuk menganalisis kompleksitas waktu dan ruang dari algoritma yang telah diimplementasikan. Hal ini bertujuan untuk memberikan pemahaman mendalam tentang efisiensi algoritma tersebut dalam berbagai skenario penggunaan.

- Pemahaman Aplikasi Nyata Graph Peserta diharapkan dapat mengidentifikasi dan memahami berbagai aplikasi graf dalam kehidupan nyata, seperti dalam pemetaan jaringan transportasi, perencanaan rute, jaringan komputer, atau analisis data dalam jejaring sosial

- Pengembangan Keterampilan Pemrograman Dengan implementasi praktis ini, peserta diharapkan meningkatkan keterampilan pemrograman mereka, khususnya dalam menggunakan bahasa C++ untuk memecahkan permasalahan berbasis graf.

II. DASAR TEORI

Struktur data graf merupakan salah satu konsep penting dalam ilmu komputer yang digunakan untuk merepresentasikan hubungan antar objek. Graf banyak diaplikasikan dalam berbagai bidang, seperti jaringan komputer, pemetaan geografis, dan analisis jejaring sosial. Berikut adalah dasar teori yang relevan dengan implementasi graf dalam bahasa C++:

1. Definisi Graph

Graf adalah struktur data yang terdiri dari dua komponen utama:

Simpul (Vertex): Representasi dari objek atau entitas.
Sisi (Edge): Hubungan atau koneksi antara dua simpul. Graf dapat direpresentasikan sebagai , di mana:
adalah himpunan simpul.
adalah himpunan sisi.

2. Jenis-Jenis Graph

Graf Berarah (Directed Graph): Graf di mana setiap sisi memiliki arah.
Graf Tidak Berarah (Undirected Graph): Graf di mana sisi tidak memiliki arah.
Graf Berbobot (Weighted Graph): Graf di mana setiap sisi memiliki bobot tertentu.
Graf Tidak Berbobot (Unweighted Graph): Graf tanpa bobot pada sisi-sisinya.

3. Representasi Graph dalam C++

Ada dua metode umum untuk merepresentasikan graf:

Matriks Ketetanggaan (Adjacency Matrix): Menggunakan matriks dua dimensi untuk menyimpan informasi hubungan antar simpul. Jika terdapat sisi antara simpul dan , maka nilai matriks pada posisi adalah 1 (atau bobotnya).

Kelebihan: Implementasi sederhana.

Kekurangan: Menggunakan banyak memori, terutama untuk graf yang jarang (sparse graph).

Daftar Ketetanggaan (Adjacency List): Menggunakan daftar untuk menyimpan simpul yang bertetangga dengan setiap simpul.

Kelebihan: Lebih hemat memori.

Kekurangan: Implementasi lebih kompleks dibandingkan matriks ketetanggaan.

4. Algoritma pada Graph

Beberapa algoritma penting yang digunakan dalam manipulasi graf meliputi:

Breadth-First Search (BFS): Algoritma untuk traversing graf secara level-wise. Cocok untuk menemukan jalur terpendek di graf tidak berbobot.

Depth-First Search (DFS): Algoritma untuk traversing graf secara mendalam. Berguna dalam analisis keterhubungan dan siklus.

Dijkstra: Algoritma untuk mencari jalur terpendek dalam graf berbobot.

Kruskal dan Prim: Algoritma untuk mencari Minimum Spanning Tree (MST), yang digunakan dalam optimasi jaringan.

5. Aplikasi Graph

Graf memiliki banyak aplikasi praktis, antara lain:

Jaringan Transportasi: Penentuan rute tercepat.

Jaringan Komputer: Routing dan optimasi data.

Jejaring Sosial: Analisis hubungan antar pengguna.

Perencanaan Proyek: Representasi dependensi dalam manajemen proyek.

6. Keunggulan Bahasa C++ dalam Implementasi Graph

Efisiensi: Bahasa C++ memiliki performa yang tinggi, terutama untuk operasi graf yang kompleks.

Standard Template Library (STL): STL menyediakan struktur data bawaan seperti vector, list, dan priority_queue yang sangat membantu dalam implementasi graf.

Kontrol Memori: C++ memungkinkan pengelolaan memori manual, yang berguna untuk aplikasi graf skala besar.

Representasi struktur data *graph* pada multilist:

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
typedef int infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;
struct ElmNode{
infoGraph info;
int Visited;
adrEdge firstEdge;
adrNode Next;
};
struct ElmEdge{
adrNode Node;
adrEdge Next;
};
struct Graph {
adrNode First;
};
```

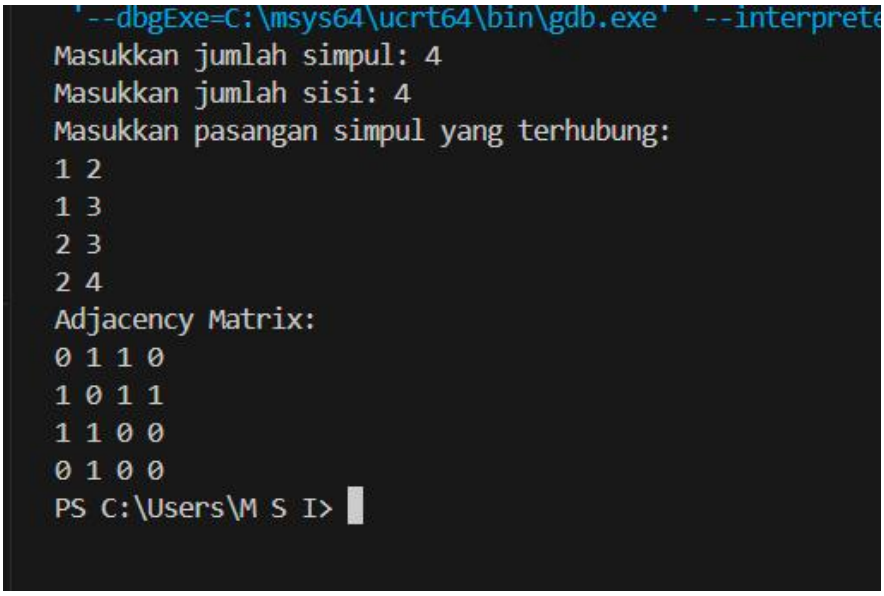
III. GUIDED

```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;
    }
}
```

```cpp
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;
}
```

IV. UNGUIDED

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

using namespace std;

int main() {
    int vertices, edges;

    // Input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> vertices;
    cout << "Masukkan jumlah sisi: ";
    cin >> edges;

    // Membuat matriks ketetanggaan
    vector<vector<int>> adjacencyMatrix(vertices, vector<int>(vertices, 0));

    cout << "Masukkan pasangan simpul yang terhubung:\n";
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        adjacencyMatrix[u - 1][v - 1] = 1;
        adjacencyMatrix[v - 1][u - 1] = 1; // Karena graf tidak berarah
    }

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

    return 0;
}

```



V. KESIMPULAN

Graf adalah struktur data yang esensial dalam ilmu komputer dan banyak digunakan dalam berbagai aplikasi nyata. Representasi graf dalam C++ melalui matriks ketetanggaan atau daftar ketetanggaan memungkinkan efisiensi dalam pengelolaan data. Dengan bantuan algoritma seperti BFS, DFS, dan Dijkstra, berbagai permasalahan kompleks dapat diselesaikan. Implementasi graf dalam C++ juga didukung oleh efisiensi performa dan fitur-fitur STL, menjadikannya pilihan yang optimal untuk pengembangan aplikasi berbasis graf. Keseluruhan, graf menawarkan solusi komprehensif untuk berbagai tantangan komputasi yang melibatkan hubungan antar elemen.