LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL I4
GRAPH

Disusun Oleh :
NAMA : HERDIAN ABDILLAH PURNOMO
NIM : 103112430048

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph merupakan struktur data non-linier yang digunakan untuk merepresentasikan hubungan antar data dalam bentuk simpul (vertex) dan sisi (edge). Setiap vertex mewakili suatu objek, sedangkan edge menunjukkan hubungan atau keterkaitan antar objek tersebut. Graph dapat bersifat berarah (directed graph) atau tidak berarah (undirected graph), serta dapat memiliki bobot (weighted graph) yang menyatakan jarak, biaya, atau nilai tertentu pada setiap sisi. Struktur graph banyak digunakan untuk memodelkan permasalahan dunia nyata seperti jaringan komputer, peta jalan, hubungan sosial, dan sistem rekomendasi.

Graph memungkinkan representasi hubungan yang kompleks dan saling terhubung, di mana satu vertex dapat terhubung dengan banyak vertex lainnya. Implementasi graph dapat dilakukan menggunakan adjacency matrix atau adjacency list, tergantung pada kebutuhan efisiensi memori dan operasi. Operasi dasar pada graph meliputi penambahan vertex dan edge, pencarian jalur, serta traversal menggunakan algoritma seperti Depth First Search (DFS) dan Breadth First Search (BFS). Dengan kemampuannya merepresentasikan relasi banyak-ke-banyak, graph menjadi struktur data yang penting dalam pemrograman dan pengolahan data berbasis jaringan.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided

graph.h

```
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

#include <iostream>
using namespace std;

typedef char infoGraph;

struct ElmNode;
struct ElmEdge;

typedef ElmNode *adrNode;
typedef ElmEdge *adrEdge;

struct ElmNode
{
    infoGraph info;
    int visited;
    adrEdge firstEdge;
```

```
    adrNode next;
};

struct ElmEdge
{
    adrNode node;
    adrEdge next;
};

struct Graph
{
    adrNode first;
};

void CreateGraph(Graph &G);
adrNode AllocateNode(infoGraph X);
adrEdge AllocateEdge(adrNode N);

void InsertNode(Graph &G, infoGraph X);
adrNode FindNode(Graph G, infoGraph X);

void ConnectNode(Graph &G, infoGraph A, infoGraph B);

void PrintInfoGraph(Graph G);

void ResetVisited(Graph &G);
void PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

graph.cpp

```
#include "graph.h"
#include <queue>
#include <stack>

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

adrNode AllocateNode(infoGraph X)
```

```
{
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}

adrEdge AllocateEdge(adrNode N)
{
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

void InsertNode(Graph &G, infoGraph X)
{
    adrNode P = AllocateNode(X);
    P->next = G.first;
    G.first = P;
}

adrNode FindNode(Graph G, infoGraph X)
{
    adrNode P = G.first;
    while (P != NULL)
    {
        if (P->info == X)
            return P;
        P = P->next;
    }
    return NULL;
}

void ConnectNode(Graph &G, infoGraph A, infoGraph B)
{
    adrNode N1 = FindNode(G, A);
    adrNode N2 = FindNode(G, B);

    if (N1 == NULL || N2 == NULL)
    {
```

```
        cout << "Node tidak ditemukan!\n";
        return;
    }

    adrEdge E1 = AllocateEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    adrEdge E2 = AllocateEdge(N1);
    E2->next = N2->firstEdge;
    N2->firstEdge = E2;
}

void PrintInfoGraph(Graph G)
{
    adrNode P = G.first;
    while (P != NULL)
    {
        cout << P->info << " -> ";
        adrEdge E = P->firstEdge;
        while (E != NULL)
        {
            cout << E->node->info << " ";
            E = E->next;
        }
        cout << endl;
        P = P->next;
    }
}

void ResetVisited(Graph &G)
{
    adrNode P = G.first;
    while (P != NULL)
    {
        P->visited = 0;
        P = P->next;
    }
}

void PrintDFS(Graph &G, adrNode N)
{
    if (N == NULL)
```

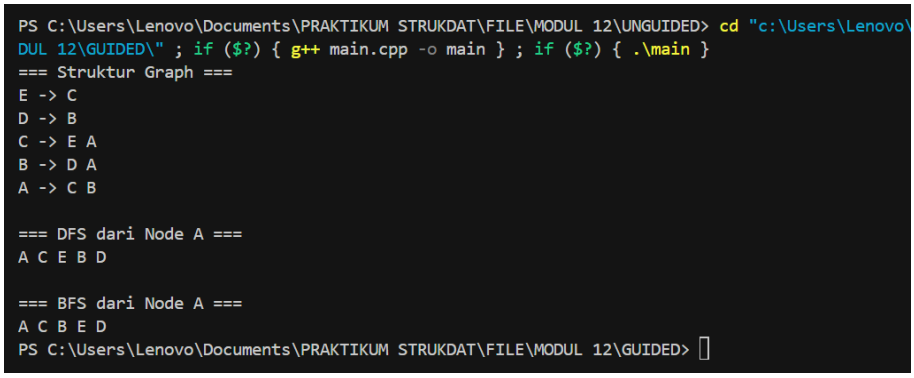
```
        return;

    N->visited = 1;
    cout << N->info << " ";

    adrEdge E = N->firstEdge;
    while (E != NULL)
    {
        if (E->node->visited == 0)
        {
            PrintDFS(G, E->node);
        }
        E = E->next;
    }
}

void PrintBFS(Graph &G, adrNode N)
{
    if (N == NULL)
        return;

    queue<adrNode> Q;
    Q.push(N);

    while (!Q.empty())
    {
        adrNode curr = Q.front();
        Q.pop();

        if (curr->visited == 0)
        {
            curr->visited = 1;
            cout << curr->info << " ";

            adrEdge E = curr->firstEdge;
            while (E != NULL)
            {
                if (E->node->visited == 0)
                {
                    Q.push(E->node);
                }
                E = E->next;
            }
```

```
        }
    }
}
```

main.cpp

```
#include "graph.h"
#include "graph.cpp"
#include <iostream>
using namespace std;

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

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

    ConnectNode(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
    ConnectNode(G, 'C', 'E');

    cout << "=== Struktur Graph ===\n";
    PrintInfoGraph(G);

    cout << "\n=== DFS dari Node A ===\n";
    ResetVisited(G);
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n\n=== BFS dari Node A ===\n";
    ResetVisited(G);
    PrintBFS(G, FindNode(G, 'A'));

    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini merupakan implementasi struktur data graf tak berarah (undirected graph) menggunakan adjacency list yang memungkinkan pengguna untuk membangun, mengelola, serta menelusuri jaringan data secara sistematis. Selain menyediakan fungsi dasar seperti pembuatan simpul (node), penyambungan sisi (edge), dan pencarian data, program ini dilengkapi dengan mekanisme pengaturan ulang status kunjungan guna menjamin akurasi proses navigasi. Untuk keperluan analisis keterhubungan antar simpul, program mengimplementasikan dua algoritma penelusuran utama, yaitu Depth First Search (DFS) untuk eksplorasi jalur secara mendalam dan Breadth First Search (BFS) untuk eksplorasi jalur secara melebar atau per tingkat.

D. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
using namespace std;

typedef char 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;
};

void CreateGraph(Graph &G);
adrNode InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);
void ResetVisited(Graph &G);
void PrintDFS(Graph G, adrNode N);
void PrintBFS(Graph G, adrNode N);

#endif
```

graph.cpp

```
#include "graph.h"

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

adrNode InsertNode(Graph &G, infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;

    if (G.first == NULL) {
        G.first = P;
    } else {
        adrNode Q = G.first;
        while (Q->next != NULL) {
            Q = Q->next;
        }
        Q->next = P;
    }
    return P;
}

void ConnectNode(adrNode N1, adrNode N2) {
    adrEdge E1 = new ElmEdge;
    E1->node = N2;
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    adrEdge E2 = new ElmEdge;
    E2->node = N1;
    E2->next = N2->firstEdge;
    N2->firstEdge = E2;
}


void PrintInfoGraph(Graph G) {
    adrNode P = G.first;
    while (P != NULL) {
        cout << P->info << " -> ";
```

```
        adrEdge E = P->firstEdge;
        while (E != NULL) {
            cout << E->node->info << " ";
            E = E->next;
        }
        cout << endl;
        P = P->next;
    }
}

void ResetVisited(Graph &G) {
    adrNode P = G.first;
    while (P != NULL) {
        P->visited = 0;
        P = P->next;
    }
}

void PrintDFS(Graph G, adrNode N) {
    if (N == NULL || N->visited == 1) return;

    cout << N->info << " ";
    N->visited = 1;

    adrEdge E = N->firstEdge;
    while (E != NULL) {
        PrintDFS(G, E->node);
        E = E->next;
    }
}

void PrintBFS(Graph G, adrNode N) {
    if (N == NULL) return;

    adrNode queue[100];
    int front = 0, rear = 0;

    queue[rear++] = N;
    N->visited = 1;

    while (front < rear) {
        adrNode P = queue[front++];
        cout << P->info << " ";
```

```
        adrEdge E = P->firstEdge;
        while (E != NULL) {

            if (E->node->visited == 0) {
                E->node->visited = 1;
                queue[rear++] = E->node;
            }
            E = E->next;
        }
    }
}
```

main.cpp

```
#include "graph.h"
#include "graph.cpp"

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

    adrNode A = InsertNode(G, 'A');
    adrNode B = InsertNode(G, 'B');
    adrNode C = InsertNode(G, 'C');
    adrNode D = InsertNode(G, 'D');
    adrNode E = InsertNode(G, 'E');
    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(C, D);
    ConnectNode(D, E);

    cout << "=== GRAPH ===" << endl;
    PrintInfoGraph(G);

    cout << "\nDFS mulai dari A: ";
    ResetVisited(G);
    PrintDFS(G, A);

    cout << "\n\nBFS mulai dari A: ";
    ResetVisited(G);
    PrintBFS(G, A);
```

```
    cout << endl;
    return 0;
}
```

Screenshots Output 1

```
=== GRAPH ===
A -> C B
B -> D A
C -> D A
D -> E C B
E -> D
```

Screenshots Output 2

```
=== GRAPH ===
A -> C B
B -> D A
C -> D A
D -> E C B
E -> D

DFS mulai dari A: A C D E B
```

Screenshots Output 3

```
=== GRAPH ===
A -> C B
B -> D A
C -> D A
D -> E C B
E -> D

DFS mulai dari A: A C D E B

BFS mulai dari A: A C B D E
PS C:\Users\Lenovo\Documents\PRAKTIKUM STRUKDAT\FILE\MODUL 12\UNGUIDED>
```

Deskripsi:

Program ini memodelkan jaringan antar kota menggunakan konsep graf, di mana jalan raya dua arah menjadi penghubung utamanya. Pengguna dapat secara fleksibel menambah kota, membangun koneksi, serta memantau seluruh rute yang terbentuk. Fitur unggulannya terletak pada dua metode pencarian: **DFS** yang mengeksplorasi jalur hingga titik terjauh (seperti penelusuran lorong), dan **BFS** yang memprioritaskan area sekitar terlebih dahulu (seperti pencarian bertahap).

E. Kesimpulan

Teori dan implementasi graf membuktikan bahwa struktur data non-linear ini adalah instrumen krusial dalam merepresentasikan keterkaitan antar entitas melalui Node dan Edge. Penggunaan Adjacency List memastikan optimasi ruang penyimpanan, sementara algoritma DFS dan BFS menawarkan solusi sistematis untuk navigasi jalur, baik secara mendalam maupun bertahap. Fleksibilitas ini menjadikan graf sebagai basis utama dalam memecahkan masalah kompleks seperti optimasi rute dan pemodelan jejaring sosial.

F. Referensi

https://www.geeksforgeeks.org/cpp/implementation-of-graph-in-cpp/

https://onecompiler.com/cpp/3y6ganqqd

https://www.w3schools.com/dsa/dsa_theory_graphs.php