LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 12
GRAPH

Disusun Oleh :
NAMA : FARIS WALID AWWAL AIDI
NIM : 103112430133

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalah himpunan tidak kosong dari node (vertex) dan garis penghubung (edge). Node merupakan elemen atau titik dalam graph, sedangkan edge adalah garis yang menghubungkan antara dua node. Secara umum, graph dibagi menjadi Graph Berarah (Directed Graph), di mana setiap edge memiliki arah spesifik ke mana node tersebut dihubungkan, dan Graph Tidak Berarah (Undirected Graph), di mana edge menghubungkan dua node tanpa arah spesifik, artinya koneksi bersifat bolak-balik. Selain arah, edge juga sering ditambahkan beban atau nilai (weight) yang merepresentasikan informasi seperti panjang atau biaya. Suatu node A dikatakan bertetangga (adjacent) dengan node B jika keduanya dihubungkan langsung oleh sebuah edge. Dalam implementasinya, graph dapat direpresentasikan menggunakan Matriks Ketetanggaan (Adjacency Matrices) (berupa Array 2 Dimensi) atau yang lebih dinamis, Multi Linked List. Representasi Multi Linked List terdiri dari list leader (horizontal) yang berisikan informasi node dan list successor (vertikal, disebut Trailer) yang berisikan alamat node tujuan. Graph juga dapat digunakan untuk menyelesaikan masalah Topological Sort, yaitu proses untuk mendapatkan urutan linear dari elemen-elemen yang memiliki keterurutan parsial (misalnya, prerequisite mata kuliah). Terakhir, terdapat dua metode utama dalam penelusuran graph: Breadth First Search (BFS) yang mengunjungi node berdasarkan kedalaman (depth) dari root (level per level) dan menggunakan struktur data Queue , serta Depth First Search (DFS) yang menelusuri secara rekursif sejauh mungkin ke subtree dari node sebelum kembali, menggunakan struktur data Stack.

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

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;
};

// Primitif Graph
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);

// Traversal
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;
    }

    // Buat edge dari N1 ke N2
    adrEdge E1 = AllocateEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    // Karena undirected -> buat edge balik
    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) {
```

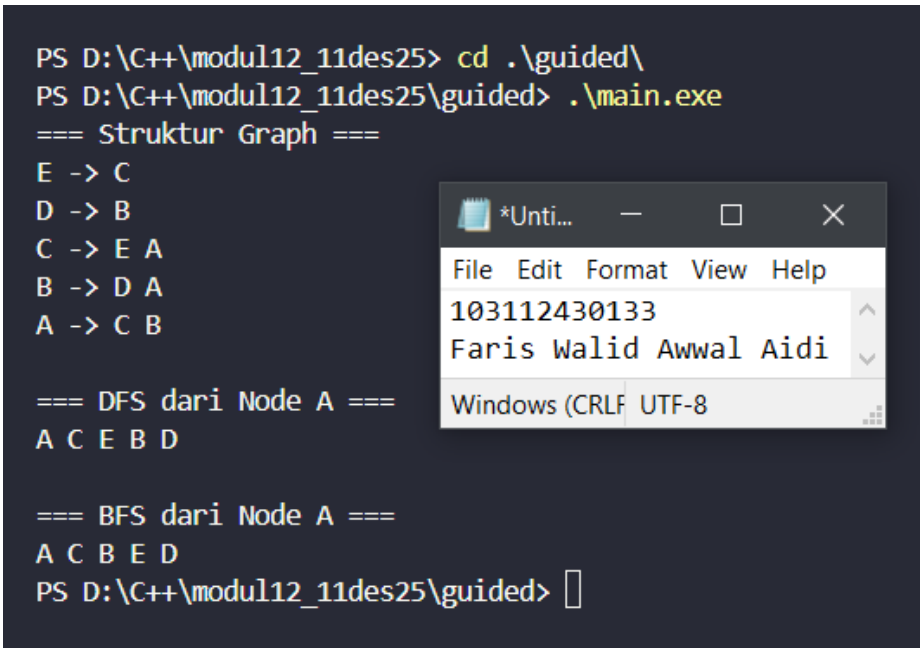
```
                    Q.push(E->node);
                }
                E = E->next;
            }
        }
    }
}
```

main.cpp

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

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

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

    // Hubungkan node (Graph tidak berarah)
    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 mengimplementasikan struktur data Graph Tidak Berarah menggunakan pendekatan Multi Linked List dalam bahasa C++. Graph direpresentasikan oleh struct Graph yang menunjuk ke list node utama (ElmNode). Setiap ElmNode memiliki informasi node, status kunjungan (visited), pointer ke list edge yang terhubung (firstEdge), dan pointer ke node berikutnya dalam list utama (next). Koneksi antar node disimpan dalam list edge (ElmEdge), di mana setiap edge menunjuk ke node tujuan. Fungsi-fungsi dasar seperti CreateGraph, AllocateNode, InsertNode, FindNode, dan PrintInfoGraph digunakan untuk mengelola dan menampilkan struktur graph. Karena graph diimplementasikan sebagai tidak berarah, prosedur ConnectNode secara otomatis membuat dua edge yang saling berlawanan (misalnya A ke B dan B ke A).

Selain itu, program ini menyediakan fungsi penelusuran graph: Depth First Search (DFS) yang menggunakan pendekatan rekursif dan Breadth First Search (BFS) yang menggunakan Queue dari Standard Template Library (STL) untuk mengunjungi semua node yang dapat dijangkau dari node awal, memastikan node yang sudah dikunjungi (visited = 1) tidak dikunjungi kembali.

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

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 saat menghubungkan: " << A
<< " dan " << B << endl;
        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;
    if (P == NULL)
    {
        cout << "Graph kosong." << endl;
        return;
    }
    while (P != NULL)
    {
        cout << "Node " << P->info << " (Terhubung ke): ";
        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)
    {
```

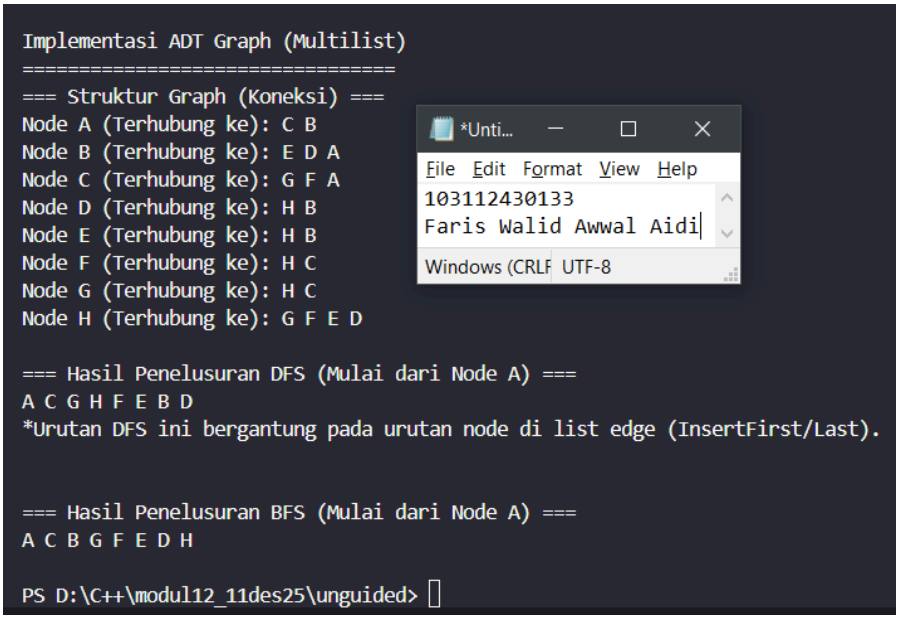
```
        if (E->node->visited == 0)
        {
            PrintDFS(G, E->node);
        }
        E = E->next;
    }
}

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

    queue<adrNode> Q;

    Q.push(N);
    N->visited = 1;

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

        cout << curr->info << " ";

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

            if (E->node->visited == 0)
            {
                E->node->visited = 1;
                Q.push(E->node);
            }
            E = E->next;
        }
    }
}
```

main.cpp

```cpp
#include "graph.h"
#include <iostream>
#include "graph.cpp"

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

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

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

    cout << "================================\n";
    cout << "Implementasi ADT Graph (Multilist)\n";
    cout << "================================\n";

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

    cout << "\n=== Hasil Penelusuran DFS (Mulai dari Node A)
===\n";

    ResetVisited(G);
    adrNode StartNodeDFS = FindNode(G, 'A');
```

```
    if (StartNodeDFS != NULL)
    {
        PrintDFS(G, StartNodeDFS);
    }
    else
    {
        cout << "Node awal (A) tidak ditemukan." << endl;
    }
    cout << "\n*Urutan DFS ini bergantung pada urutan node di list
edge (InsertFirst/Last)." << endl;

    cout << "\n\n=== Hasil Penelusuran BFS (Mulai dari Node A)
===\n";

    ResetVisited(G);
    adrNode StartNodeBFS = FindNode(G, 'A');
    if (StartNodeBFS != NULL)
    {
        PrintBFS(G, StartNodeBFS);
    }
    else
    {
        cout << "Node awal (A) tidak ditemukan." << endl;
    }
    cout << "\n"
         << endl;

    return 0;
}
```

Screenshot Output



Deskripsi:

Program ini mengimplementasikan struktur data Graph Tidak Berarah menggunakan pendekatan Multi Linked List dalam bahasa C++. Graph direpresentasikan oleh struct Graph yang menunjuk ke list node utama (ElmNode). Setiap ElmNode memiliki informasi node (info), status kunjungan (visited), pointer ke list edge yang terhubung (firstEdge), dan pointer ke node berikutnya dalam list utama (next). Koneksi antar node disimpan dalam list edge (ElmEdge), di mana setiap edge menunjuk ke node tujuan (node) dan pointer ke edge berikutnya (next). Fungsi-fungsi dasar seperti CreateGraph, AllocateNode, InsertNode, FindNode, dan PrintInfoGraph digunakan untuk mengelola dan menampilkan struktur graph. Karena graph diimplementasikan sebagai tidak berarah, prosedur ConnectNode secara otomatis membuat dua edge yang saling berlawanan (misalnya A ke B dan B ke A). Selain itu, program ini menyediakan fungsi penelusuran graph: Depth First Search (DFS) yang menggunakan pendekatan rekursif untuk mengunjungi node sejauh mungkin ke dalam sebelum mundur, dan Breadth First Search (BFS) yang menggunakan Queue dari Standard Template Library (STL) untuk mengunjungi node berdasarkan kedalaman atau level secara berurutan. Kedua metode penelusuran ini memanfaatkan atribut visited pada ElmNode untuk memastikan setiap node hanya dikunjungi satu kali, dengan fungsi ResetVisited digunakan sebelum setiap penelusuran baru.

D. Kesimpulan

Kesimpulan dari program ini adalah implementasi lengkap dari Abstract Data Type (ADT) Graph Tidak Berarah menggunakan struktur data Multi Linked List di C++. Representasi ini memungkinkan pengelolaan node dan edge secara dinamis, di mana setiap node (ElmNode) terhubung ke list edge (ElmEdge) yang menunjuk ke node-node tetangga. Program ini berhasil mengimplementasikan operasi dasar seperti pembuatan, penyisipan node, dan penghubungan node (ConnectNode), yang secara spesifik pada graph tidak berarah, membuat koneksi bolak-balik. Lebih lanjut, program juga berhasil mengaplikasikan dua metode penelusuran graph yang fundamental: Depth First Search (DFS), yang diimplementasikan secara rekursif, dan Breadth First Search (BFS), yang memanfaatkan antrian (Queue) untuk menelusuri graph secara level-per-level, menunjukkan fungsionalitas ADT graph secara keseluruhan.

E. Referensi

Raharjo, Budi. 2025. *Buku Pemrograman C++ Mudah dan Cepat Menjadi Master C.*

Wikipedia contributors. (2024, 8 Mei). C++. Wikipedia, Ensiklopedia Bebas. Diakses pada 2 Desember 2025, dari https://id.wikipedia.org/wiki/C%2B%2B