LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIV

GRAPH

Disusun Oleh :

NAMA : Taufik Hafit Zakaria
NIM : 103112430093

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Graph adalah struktur data non-linear yang merepresentasikan hubungan antar objek. Struktur ini terdiri dari himpunan Node (Vertex) yang berisi data dan himpunan Edge (Sisi) yang menghubungkan sepasang node. Berbeda dengan Tree yang hirarkis, Graph memungkinkan hubungan yang lebih bebas (Many-to-Many) dan dapat membentuk siklus. Dalam praktikum ini, Graph diimplementasikan menggunakan pendekatan dinamis berbasis Multi-list.

Karakteristik dan Jenis Graph

1. Jenis Berdasarkan Arah: Terbagi menjadi Directed Graph (berarah), di mana koneksi memiliki arah spesifik (A ke B tidak sama dengan B ke A), dan Undirected Graph (tak berarah), di mana koneksi berlaku dua arah secara otomatis.
2. Representasi Multi-list: Graph dimodelkan menggunakan dua komponen list berkait:
    1. List Node: List utama yang menyimpan elemen verteks.
    2. List Edge: List sekunder yang terhubung pada setiap node, menyimpan informasi ketetanggaan (koneksi) ke node lain.

Metode Penelusuran (Traversal)

Untuk mengunjungi seluruh elemen dalam Graph, terdapat dua algoritma utama:

1. BFS (Breadth First Search): Penelusuran melebar per level kedalaman menggunakan struktur data Queue.
2. DFS (Depth First Search): Penelusuran mendalam menelusuri satu cabang hingga habis sebelum backtrack, menggunakan struktur data Stack.

Keuntungan Representasi Multi-list

1. Efisiensi Memori: Lebih hemat memori dibandingkan representasi Adjacency Matrix (Array 2D), karena hanya mengalokasikan ruang untuk koneksi (edge) yang benar-benar ada.
2. Dinamis: Memudahkan penambahan atau penghapusan node dan edge secara fleksibel tanpa batasan ukuran statis.

Keterbatasan

1. Kompleksitas Implementasi: Membutuhkan manajemen pointer yang rumit untuk menghubungkan list node dengan list edge.
2. Kecepatan Akses: Pengecekan hubungan antar dua node memerlukan penelusuran linear pada list edge, sehingga relatif lebih lambat dibandingkan akses indeks langsung pada representasi matriks.

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

Guided 1

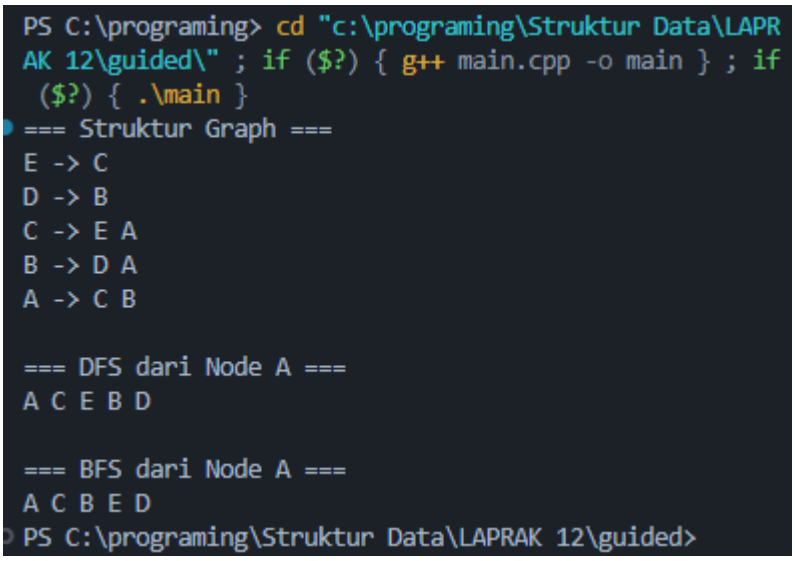
```
//Taufik Hafit Zakaria
//103112430093
//graf.h

#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
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//graf.cpp

#include "graf.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;
}

```

```cpp
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;
            }
        }
    }
}
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//main.cpp

#include "graf.h"
#include "graf.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 yang saya praktikkan ini adalah simulasi struktur data Graph menggunakan bahasa C++. Konsepnya menggunakan teknik "List di dalam List" (Multi Linked List), di mana terdapat daftar utama untuk menyimpan titik data (disebut Node) dan daftar tambahan untuk menyimpan sambungan atau garis hubung (disebut Edge). Karena jenis graf yang saya buat adalah graf tidak berarah (*Undirected*), maka setiap kali saya menghubungkan dua titik (misalnya A ke B), program secara otomatis membuat hubungan timbal balik (A terhubung ke B, dan B terhubung ke A). Kode program dipisahkan menjadi tiga file yakni graf.h, graf.cpp, dan main.cpp.

Dalam percobaan di fungsi main, saya memulai dengan membuat graf kosong lalu menambahkan lima titik data, yaitu A, B, C, D, dan E. Setelah titik-titik tersebut masuk, saya menghubungkannya satu sama lain sesuai pola yang diinginkan (misalnya A ke B, B ke D, dst). Program kemudian menampilkan daftar hubungan tersebut ke layar untuk memastikan strukturnya sudah benar. Terakhir, saya melakukan simulasi penelusuran data dimulai dari titik A menggunakan dua cara berbeda: DFS (metode yang menelusuri satu cabang sampai ujung terdalam dulu) dan BFS (metode yang menelusuri semua tetangga terdekat secara melebar), di mana keduanya berhasil mengunjungi seluruh titik yang terhubung.

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

Unguided 1

```cpp
//Taufik Hafit Zakaria
//103112430093
//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 sesuai soal latihan 1
void CreateGraph(Graph &G);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

// Primitif tambahan untuk mendukung
adrNode AllocateNode(infoGraph X);
adrEdge AllocateEdge(adrNode N);
adrNode FindNode(Graph G, infoGraph X);
void ResetVisited(Graph &G);

// Soal 2 dan 3
void PrintDFS(Graph G, adrNode N);
void PrintBFS(Graph G, adrNode N);

#endif
```

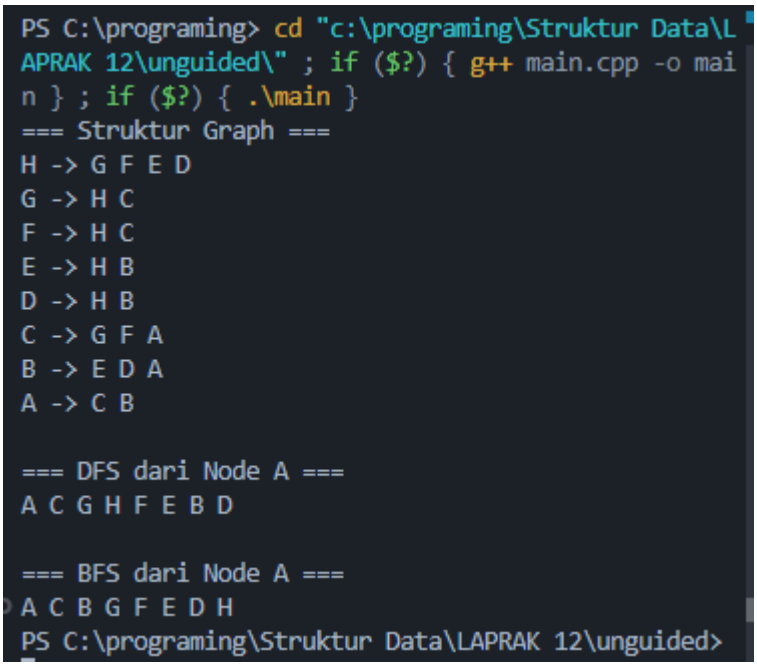
```cpp
//Taufik Hafit Zakaria
//103112430093
//graph.cpp

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

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(adrNode N1, adrNode N2) {
    if (N1 == NULL || N2 == NULL) {
        return;
    }

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

    // Edge dari N2 ke N1 (undirected)
    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;
    }
}
```

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

// Soal 2: Prosedur PrintDFS
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;
    }
}

// Soal 3: Prosedur PrintBFS
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;
            }
        }
    }
}
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//main.cpp

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

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

    // Tambah node sesuai Gambar pada soal
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    // Hubungkan node sesuai Gambar pada soal
    adrNode A = FindNode(G, 'A');
    adrNode B = FindNode(G, 'B');
    adrNode C = FindNode(G, 'C');
    adrNode D = FindNode(G, 'D');
    adrNode E = FindNode(G, 'E');
    adrNode F = FindNode(G, 'F');
    adrNode GNode = FindNode(G, 'G');
    adrNode H = FindNode(G, 'H');

    // Koneksi sesuai gambar
    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, GNode);
    ConnectNode(D, H);
    ConnectNode(E, H);
    ConnectNode(F, H);
    ConnectNode(GNode, H);

    // Tampilkan struktur graph
    cout << "=== Struktur Graph ===" << endl;
    PrintInfoGraph(G);

    // Soal 2: DFS
    cout << "\n=== DFS dari Node A ===" << endl;
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    // Soal 3: BFS
    cout << "\n=== BFS dari Node A ===" << endl;
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini adalah implementasi praktikum struktur data Graph Tak Berarah (*Undirected Graph*) yang saya kerjakan menggunakan bahasa C++. Representasi datanya menggunakan konsep "List di dalam List" (Multi Linked List), di mana terdapat daftar utama untuk menyimpan data titik (Node) dan daftar tambahan yang menempel pada setiap titik untuk menyimpan jalur hubungannya (Edge).

Dalam simulasi di fungsi main, saya membangun sebuah graf lengkap sesuai dengan gambar pada soal latihan. Prosesnya dimulai dengan mendaftarkan titik-titik dari huruf 'A' hingga 'H'. Setelah titik dibuat, saya menghubungkannya satu sama lain (seperti menghubungkan A dengan B, B dengan D, hingga bermuara ke H) menggunakan fungsi ConnectNode. Karena ini adalah graf tak berarah, setiap kali saya membuat hubungan, program otomatis mencatatnya sebagai jalur bolak-balik. Terakhir, saya menampilkan struktur graf tersebut dan melakukan simulasi penelusuran dimulai dari titik A menggunakan dua metode: DFS (penelusuran mendalam satu jalur sampai habis) dan BFS (penelusuran melebar ke semua tetangga terdekat), untuk melihat perbedaan urutan titik yang dikunjungi.

D. Kesimpulan

Dari hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data Graph dengan representasi *Adjacency List* merupakan solusi yang sangat efisien untuk memodelkan hubungan antar objek yang bersifat non-linear dan kompleks (relasi *Many-to-Many*). Melalui implementasi program ini, konsep pengembangan dari Multi Linked List dapat dipahami secara mendalam, di mana list utama berperan sebagai himpunan verteks (Node) dan list sekunder berfungsi sebagai himpunan sisi (Edge) yang menyimpan informasi konektivitas antar simpul. Praktikum ini secara khusus mendemonstrasikan bahwa manipulasi Graph Tak Berarah (*Undirected*) menuntut ketelitian logika pointer yang tinggi, karena setiap pembentukan relasi antar dua node mewajibkan pembuatan simpul edge secara timbal balik (dua arah) pada masing-masing list edge node terkait.

Selain itu, program ini membuktikan bahwa representasi *Adjacency List* menawarkan efisiensi memori yang lebih baik dibandingkan representasi Matriks (*Adjacency Matrix*) berbasis array 2 dimensi, terutama untuk graf yang jarang (*sparse graph*), karena memori hanya dialokasikan untuk hubungan yang benar-benar ada. Eksperimen dengan algoritma penelusuran *Depth First Search* (DFS) dan *Breadth First Search* (BFS) juga memberikan pemahaman krusial mengenai strategi kunjungan data, baik secara mendalam menggunakan tumpukan (*stack*) maupun melebar menggunakan antrean (*queue*). Struktur data ini sangat relevan dan aplikatif untuk memecahkan masalah dunia nyata, seperti simulasi rute navigasi antar kota, pemetaan topologi jaringan komputer, hingga analisis hubungan dalam jejaring sosial.

E. Referensi

Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Pearson.

GeeksforGeeks. (2024). Graph Data Structure And Algorithms. Diakses pada tahun 2025 dari https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Programiz. (2024). Depth First Search (DFS). Diakses pada tahun 2025 dari https://www.programiz.com/dsa/graph-dfs

Programiz. (2024). Breadth First Search (BFS). Diakses pada tahun 2025 dari https://www.programiz.com/dsa/graph-bfs