LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XII
GRAPH

Disusun Oleh :
NAMA : ABYAN RAHMAN AL FARIZ
NIM : 103112430021

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan verteks (V), sedangkan sisi yang menghubungkan antar verteks disebut edge (E). Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y.

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

Guided 1 (graph.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;
}

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;
            }
        }
    }
}
```

Guided 1 (graph.h)

```
#ifndef GRAF_H_INCLUDED
#define GRAF_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 DFS(Graph &G, adrNode N);
```

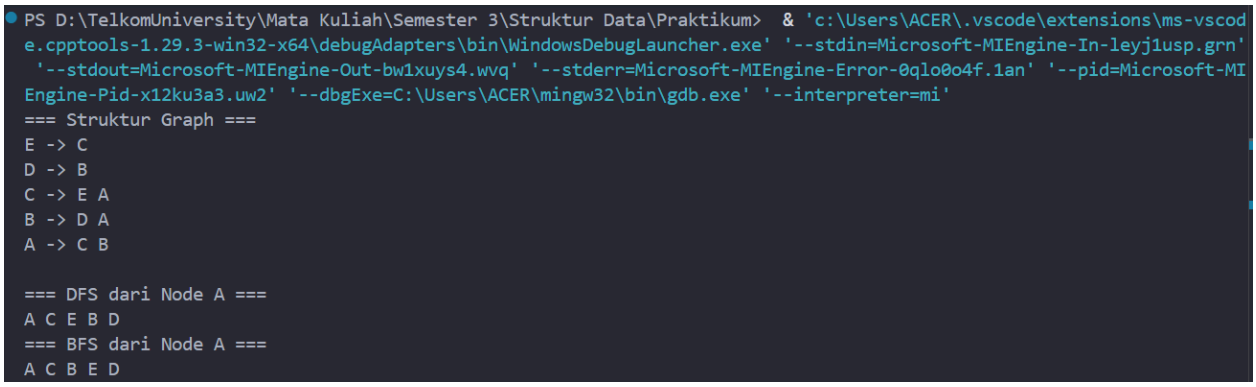
```
void PrintBFS(Graph &G, adrNode N);

#endif
```

Guided 1 (main.cpp)

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

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

    //Tambah node
    InsertNode(G, 'A'); //0
    InsertNode(G, 'B'); //1
    InsertNode(G, 'C'); //2
    InsertNode(G, 'D'); //3
    InsertNode(G, 'E'); //4

    //Hubungkan node (Graph tidak berarah)
    ConnectNode(G, 'A', 'B'); //0 -> 1
    ConnectNode(G, 'A', 'C'); //0 -> 2
    ConnectNode(G, 'B', 'D'); //1 -> 3
    ConnectNode(G, 'C', 'E'); //2 -> 4

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

    cout << "\n=== DFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n=== BFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintBFS(G, FindNode(G, 'A'));
```

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

Screenshots Output:



Deskripsi:

Program ini mengimplementasikan struktur data Graph tidak berarah (undirected graph) menggunakan pendekatan adjacency list. Setiap simpul (node) direpresentasikan oleh struktur ElmNode yang menyimpan informasi berupa karakter, penanda kunjungan (visited), pointer ke edge pertama, serta pointer ke node berikutnya dalam daftar node. Sementara itu, hubungan antar node direpresentasikan oleh struktur ElmEdge yang menunjuk ke node tujuan. Program menyediakan fungsi-fungsi dasar seperti CreateGraph untuk inisialisasi graph, InsertNode untuk menambahkan node, ConnectNode untuk menghubungkan dua node secara dua arah, serta PrintInfoGraph untuk menampilkan struktur graph beserta hubungan antar node. Selain itu, program juga mengimplementasikan dua metode penelusuran graph, yaitu Depth First Search (DFS) yang menelusuri graph secara mendalam menggunakan rekursi, dan Breadth First Search (BFS) yang menelusuri graph secara melebar menggunakan struktur queue. Fungsi ResetVisited digunakan untuk mengatur ulang status kunjungan node agar traversal dapat dilakukan kembali dengan benar. Program utama (main.cpp) mendemonstrasikan pembuatan graph, penghubungan node, serta proses traversal DFS dan BFS yang dimulai dari node tertentu.

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

Unguided 1 (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 E = new ElmEdge;
    E->Node = N;
    E->Next = NULL;
    return E;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = AllocateNode(X);
    if (G.First == NULL) {
        G.First = P;
    } else {
        adrNode Q = G.First;
        while (Q->Next != NULL)
            Q = Q->Next;
        Q->Next = 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) {
    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 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) {
    queue<adrNode> Q;
    Q.push(N);
    N->visited = 1;

    while (!Q.empty()) {
        adrNode P = Q.front();
        Q.pop();
        cout << P->info << " ";

        adrEdge E = P->firstEdge;
        while (E != NULL) {
            if (E->Node->visited == 0) {
                E->Node->visited = 1;
                Q.push(E->Node);
            }
            E = E->Next;
        }
    }
}
```

Unguided 1 (graph.h)

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE

#include <iostream>
using namespace std;

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;

struct ElmEdge {
    adrNode Node;
    adrEdge Next;
};

struct ElmNode {
    infoGraph info;
```

```
    int visited;
    adrEdge firstEdge;
    adrNode Next;
};

struct Graph {
    adrNode First;
};

void CreateGraph(Graph &G);
adrNode AllocateNode(infoGraph X);
adrEdge AllocateEdge(adrNode N);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
adrNode FindNode(Graph G, infoGraph X);
void PrintInfoGraph(Graph G);

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

#endif
```

Unguided 1 (main.cpp)

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

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(FindNode(G,'A'), FindNode(G,'B'));
    ConnectNode(FindNode(G,'A'), FindNode(G,'C'));
```

```
    ConnectNode(FindNode(G,'B'), FindNode(G,'D'));
    ConnectNode(FindNode(G,'C'), FindNode(G,'E'));

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

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

    adrNode P = G.First;
    while (P != NULL) {
        P->visited = 0;
        P = P->Next;
    }

    cout << "\nBFS mulai dari A: ";
    PrintBFS(G, FindNode(G,'A'));

    return 0;
}
```

Screenshots Output:

```
PS D:\TelkomUniversity\Mata Kuliah\Semester 3\Struktur Data\Praktikum>  & 'c:\Users\ACER\.vscode\extensions\ms-vscode.cpptools-1.29.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-vevuddxq.2gv' '--stdout=Microsoft-MIEngine-Out-v2ibv5kf.u4e' '--stderr=Microsoft-MIEngine-Error-ddqpk2ra.dws' '--pid=Microsoft-MIEngine-Pid-apzfpayn.wqy' '--dbgExe=C:\Users\ACER\mingw32\bin\gdb.exe' '--interpreter=mi'
=== GRAPH ===
A : C B
B : D A
C : E A
D : B
E : C

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

Deskripsi:

Program latihan ini membahas penerapan struktur data Graph menggunakan representasi adjacency list untuk menyimpan hubungan antar simpul. Setiap simpul direpresentasikan sebagai node yang memiliki informasi data, penanda kunjungan (visited), serta pointer ke daftar edge yang menunjukkan node-node tetangganya. Hubungan antar simpul bersifat tidak berarah (undirected), sehingga setiap koneksi dibuat dua arah. Program menyediakan operasi dasar seperti pembuatan graph, penambahan node, pencarian node, dan penghubungan antar node. Selain itu, program

juga mengimplementasikan dua metode traversal graph, yaitu Depth First Search (DFS) untuk menelusuri graph secara mendalam dan Breadth First Search (BFS) untuk menelusuri graph secara melebar berdasarkan urutan antrian.

D. Kesimpulan

Kesimpulannya, **graph** merupakan struktur data yang digunakan untuk merepresentasikan hubungan atau keterkaitan antar data dalam bentuk simpul (vertex) dan sisi (edge). Graph sangat fleksibel karena mampu memodelkan berbagai permasalahan nyata seperti jaringan komputer, peta jalan, hubungan sosial, dan sistem navigasi.

Dalam penerapannya, graph dapat direpresentasikan menggunakan adjacency list atau adjacency matrix, serta dapat bersifat berarah maupun tidak berarah. Proses penelusuran graph umumnya dilakukan menggunakan algoritma **Depth First Search (DFS)** dan **Breadth First Search (BFS)**, yang masing-masing memiliki karakteristik berbeda dalam menjelajahi simpul. Dengan memahami konsep dan operasi graph, proses pemodelan dan analisis hubungan data menjadi lebih terstruktur dan efisien.

E. Referensi

https://www.trivusi.web.id/2022/07/struktur-data-graph.html

https://bif.telkomuniversity.ac.id/mengenal-struktur-data-graph-definisi-dan-manfaat/

https://www.exabytes.co.id/blog/jenis-struktur-data-graph-dan-penerapannya/