LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL XIV
GRAPH

Disusun Oleh :
NAMA : Ridha Akifah
NIM : 103112400132

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph merupakan salah satu struktur data yang digunakan untuk merepresentasikan hubungan antar objek, yang terdiri dari sekumpulan simpul (vertex) dan sisi (edge) yang menghubungkan antar simpul tersebut. Graph dapat digunakan untuk memodelkan berbagai permasalahan nyata seperti jaringan komputer, peta jalan, hubungan antar data, serta alur proses. Berdasarkan arah sisi, graph dibedakan menjadi graph berarah (directed graph) dan graph tidak berarah (undirected graph). Dalam implementasinya, graph dapat direpresentasikan menggunakan matriks ketetanggaan (adjacency matrix) maupun multilist, di mana multilist lebih fleksibel karena bersifat dinamis. Selain itu, graph mendukung berbagai metode penelusuran seperti Breadth First Search (BFS) dan Depth First Search (DFS) yang digunakan untuk mengunjungi seluruh simpul dalam graph secara sistematis sesuai dengan aturan tertentu.

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

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;

};


void createGraph(Graph &G);

adrNode AllocatedNode(infoGraph X);

adrEdge AllocatedEdge(adrNode N);


void insertNode(Graph &G, infoGraph X);

void 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 "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;
    }

    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;
    }
}
```

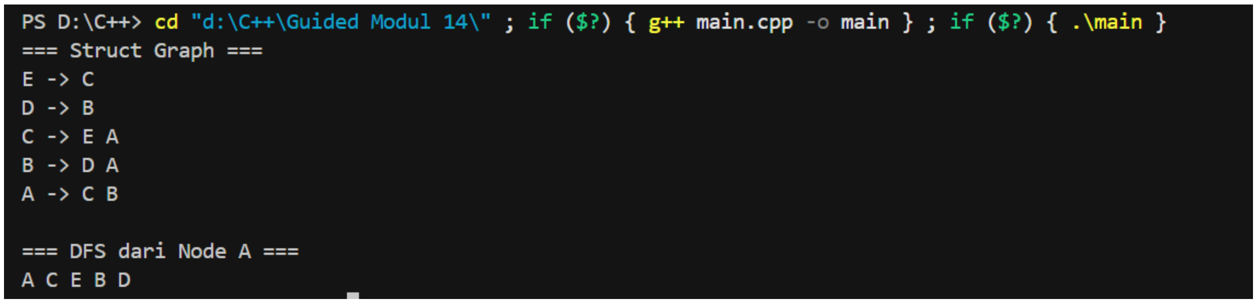
```
void ResetVisited(Graph &G)
{
    adrNode P = G.first;
    while (P != NULL)
    {
        P->visited = 0;
        P = P->next;
    }
}

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 "graf.h"
```

```cpp
#include "graf.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 << "=== Struct Graph ===\n";

    PrintInfoGraph(G);


    cout << "\n=== DFS dari Node A ===\n";

    ResetVisited(G);

    PrintDFS(G, findNode(G, 'A'));


    cout << endl;

    return 0;
}
```

Screenshoot Output



```
PS D:\C++> cd "d:\C++\Guided Modul 14\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
=== Struct Graph ===
E -> C
D -> B
C -> E A
B -> D A
A -> C B

=== DFS dari Node A ===
A C E B D
```

Deskripsi:

Program ini mengimplementasikan struktur data graph tidak berarah menggunakan pointer dalam bahasa C++, dengan fitur penambahan node, penghubungan antar node, serta penelusuran graph menggunakan metode DFS dan BFS.

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

Unguided 1

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 AllocatedNode(infoGraph X);
adrEdge AllocatedEdge(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 AllocatedNode(infoGraph X)
{
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}
adrEdge AllocatedEdge(adrNode N)
{
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}
void InsertNode(Graph &G, infoGraph X)
{
    adrNode P = AllocatedNode(X);
    P->next = G.first;
    G.first = P;
}
adrNode findNode(Graph G, infoGraph X)
{
    adrNode P = G.first;
    while (P != NULL)
```

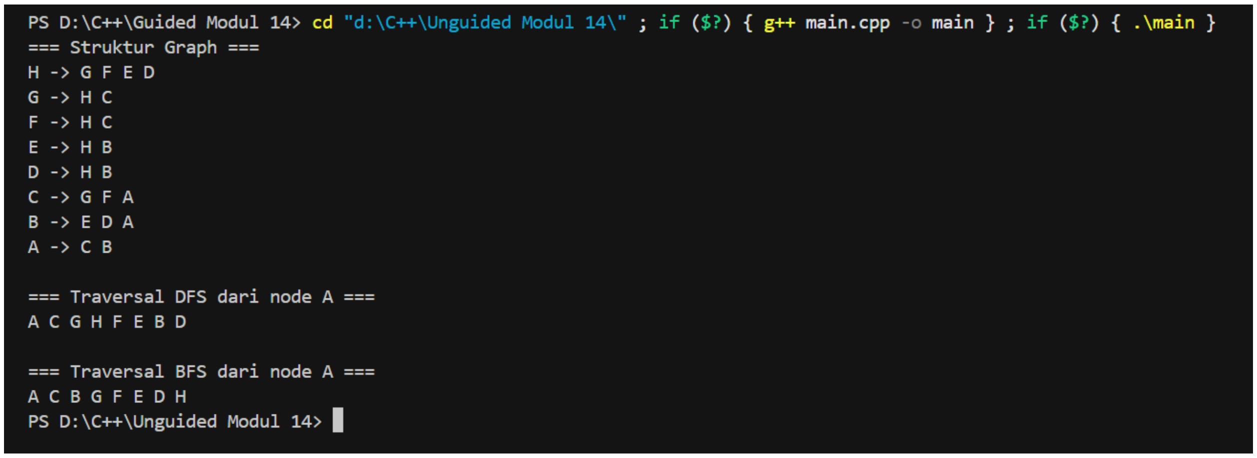
```cpp
    {
        if (P->info == X)
            return P;
        P = P->next;
    }
    return NULL;
}
void ConnectNode(Graph &G, infoGraph A, infoGraph B)
{
    adrNode N1 = findNode(G, A);
    adrNode N2 = findNode(G, B);
    if (N1 == NULL || N2 == NULL)
    {
        cout << "Node tidak ditemukan!" << endl;
        return;
    }
    // buat edge dari N1 ke N2
    adrEdge E1 = AllocatedEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;
    // karena undirected -> buat edge balik
    adrEdge E2 = AllocatedEdge(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;
    }
}
// ditambahkan di soal no 2
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;

    }

}

// ditambahkan di soal no 3

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)

                {
```

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

main.cpp

```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');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');
    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 << "=== Struktur Graph ===" << endl;
    PrintInfoGraph(G);
    cout << "\n=== Traversal DFS dari node A ===" << endl;
    ResetVisited(G);
    printDFS(G, findNode(G, 'A'));
    cout << "\n\n=== Traversal BFS dari node A ===" << endl;
    ResetVisited(G);
    printBFS(G, findNode(G, 'A'));
    cout << endl;
    return 0;
}
```

Screenshoot Output



```
PS D:\C++\Guided Modul 14> cd "d:\C++\Unguided Modul 14\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
=== Struktur Graph ===
H -> G F E D
G -> H C
F -> H C
E -> H B
D -> H B
C -> G F A
B -> E D A
A -> C B

=== Traversal DFS dari node A ===
A C G H F E B D

=== Traversal BFS dari node A ===
A C B G F E D H
PS D:\C++\Unguided Modul 14> 
```

Deskripsi:

Program ini dirancang untuk memahami konsep struktur data graph tidak berarah yang diimplementasikan menggunakan pointer. Melalui program ini, pengguna dapat membentuk simpul-simpul graph serta menghubungkannya antar simpul sesuai kebutuhan. Selain itu, program menyediakan metode penelusuran Depth First Search (DFS) dan Breadth First Search (BFS) yang digunakan untuk menelusuri setiap simpul dalam graph secara sistematis.

D. Kesimpulan

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data graph digunakan untuk merepresentasikan hubungan antar simpul dalam suatu sistem secara terstruktur. Melalui implementasi graph menggunakan multilist, hubungan antar node dapat dikelola secara dinamis dan efisien. Selain itu, metode penelusuran graph seperti Breadth First Search (BFS) dan Depth First Search (DFS) memungkinkan proses penelusuran simpul dilakukan secara sistematis sesuai dengan kebutuhan. Praktikum ini membantu memahami konsep graph serta penerapannya dalam pemrograman, khususnya dalam pengolahan data yang memiliki keterkaitan antar elemen.

E. Referensi

https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170

https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd