LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 12

GRAPH

Disusun Oleh :
NAMA : Muhammad Omar Nadiv
NIM : 103112430063

Dosen
Fahrudin Mukti Wibowo, S.Kom., M.Eng.

PROGRAM STUDI TEKNIK INFORMATIKA
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 1 (graf.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 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
```

Guided 1 (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;
}

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)
```

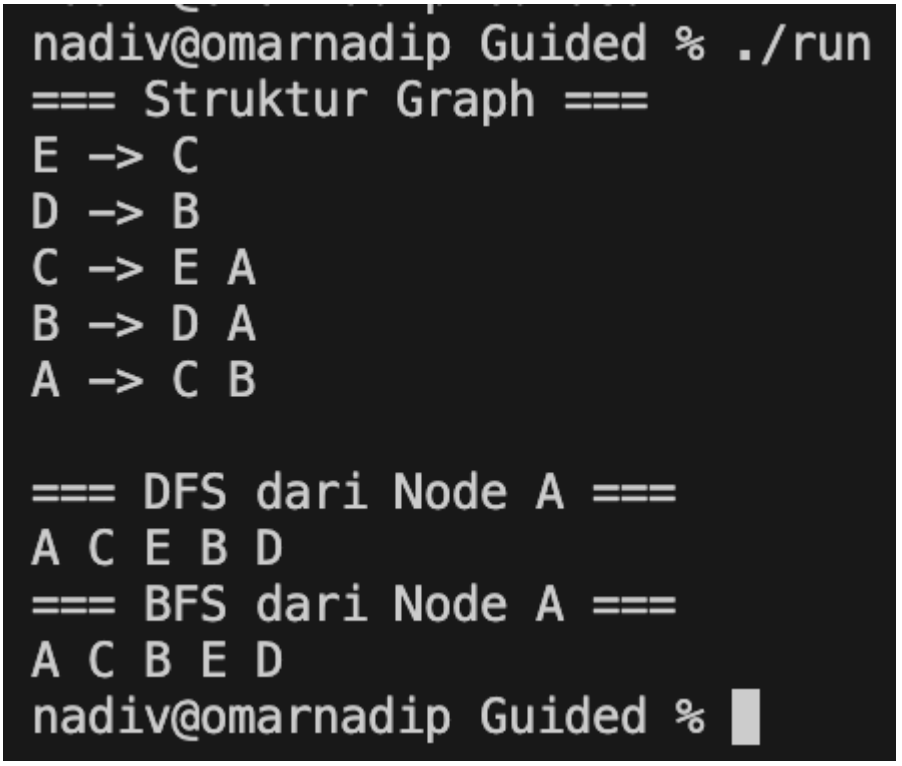
```
        {
            curr->visited = 1;
            cout << curr->info << " ";

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

Guided 1 (main.cpp)

```
#include "graf.h"
#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

    //Tambah edge
    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 menggunakan pendekatan adjacency list dengan linked list. Setiap node (vertex) menyimpan informasi berupa karakter, penanda kunjungan (visited),

serta pointer ke daftar edge yang terhubung. Edge digunakan untuk merepresentasikan hubungan antar node dan bersifat undirected, sehingga setiap koneksi dibuat dua arah. Program menyediakan operasi dasar seperti pembuatan graph, penambahan node dan edge, pencarian node, serta penampilan struktur graph. Selain itu, traversal graph diimplementasikan menggunakan Depth First Search (DFS) dan Breadth First Search (BFS) untuk menelusuri node mulai dari titik awal tertentu.

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

Unguided 1 (file main.cpp)

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

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

    cout << "=== Insert Node ===" << endl;
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');

    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');

    cout << "\n=== Connect Node (Undirected) ===" << endl;
    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(C, E);
```

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

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

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

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

(file 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;

    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)
{
```

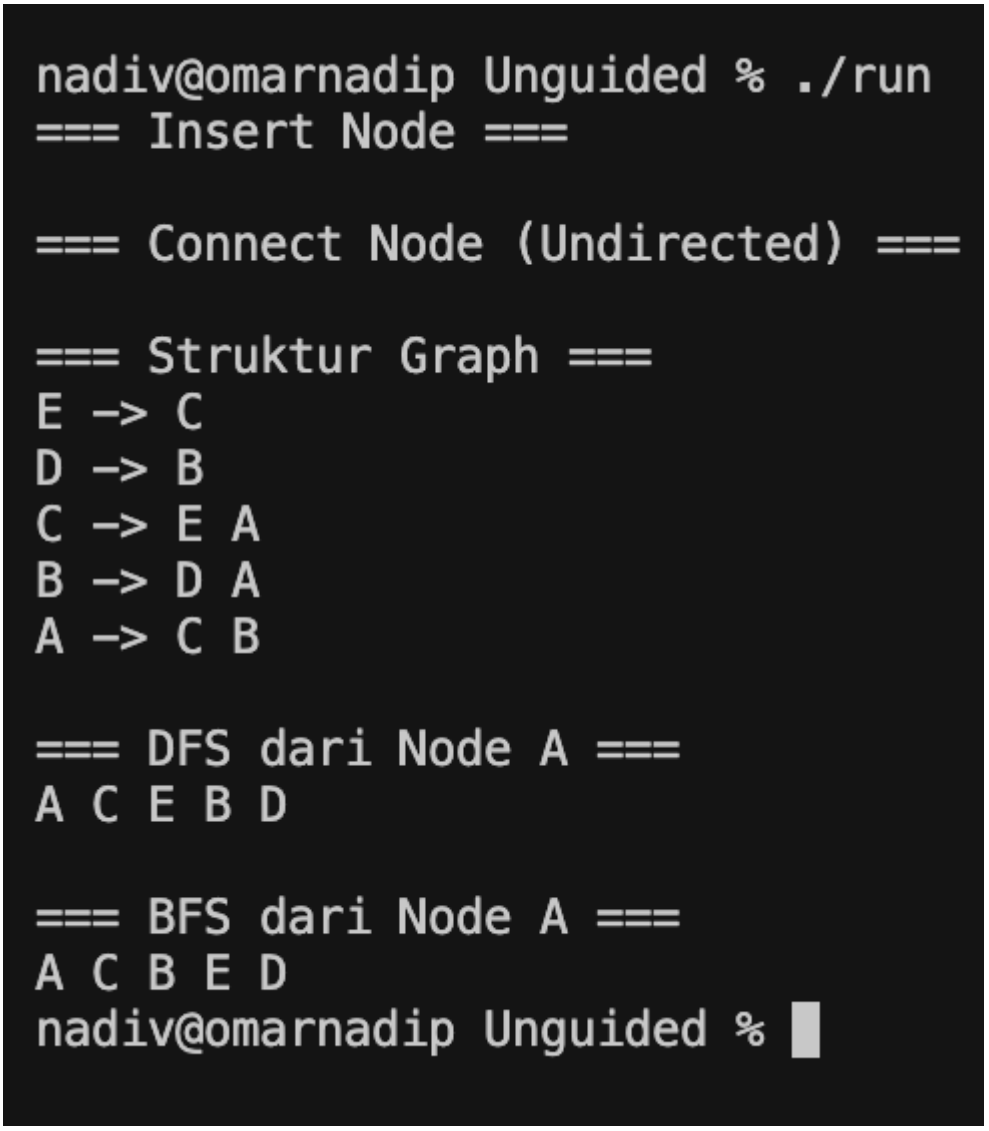
```
    if (N == NULL)
        return;

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

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

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

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

(file 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 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);
adrNode FindNode(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
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data Graph tidak berarah menggunakan representasi adjacency list. Setiap node menyimpan informasi berupa karakter, penanda kunjungan (visited), serta pointer ke daftar edge yang terhubung dengan node lain. Graph dibangun dengan menambahkan node dan menghubungkannya melalui edge dua arah sehingga hubungan antar node bersifat timbal balik. Program juga menyediakan fungsi untuk menampilkan struktur graph sehingga keterhubungan antar node dapat terlihat dengan jelas.

Selain itu, program menerapkan algoritma Depth First Search (DFS) dan Breadth First Search (BFS) untuk melakukan penelusuran graph mulai dari node tertentu. DFS menelusuri graph secara mendalam ke setiap cabang terlebih dahulu, sedangkan BFS menelusuri graph secara melebar berdasarkan level. Melalui implementasi ini, program menunjukkan cara kerja traversal graph serta pemanfaatan struktur graph dalam merepresentasikan hubungan data yang saling terhubung.

D. Kesimpulan

Pada modul ini, saya mempelajari konsep dasar Graph sebagai struktur data non-linier yang digunakan untuk merepresentasikan hubungan antar data. Saya memahami bagaimana membangun graph tidak berarah menggunakan adjacency list, serta melakukan operasi dasar seperti penambahan node dan edge. Selain itu, saya juga mempelajari dan mempraktikkan algoritma penelusuran graph menggunakan Depth First Search (DFS) dan Breadth First Search (BFS) untuk menelusuri seluruh node dalam graph. Modul ini membantu saya memahami bagaimana graph digunakan untuk memodelkan hubungan yang kompleks dalam berbagai permasalahan pemrograman.

E. 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