LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 14
PENGENALAN CODE BLOCKS

Disusun Oleh :
NAMA : Muhamad Ilham Syahid
NIM : 13112400155

Dosen
WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalah struktur data non-linear yang terdiri dari node (vertex) dan edge yang menghubungkan antar node. Graph dapat berupa graph berarah maupun tidak berarah, serta direpresentasikan menggunakan adjacency matrix atau multilist. Untuk menelusuri graph digunakan algoritma Depth First Search (DFS) yang menelusuri graph secara mendalam dan Breadth First Search (BFS) yang menelusuri graph berdasarkan level. Struktur data graph banyak digunakan untuk merepresentasikan hubungan antar data dalam berbagai bidang seperti jaringan, peta, dan sistem prasyarat.

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

Guided 1

Graph.h

```h
#ifndef GRAPH_H
#define GRAPH_H

#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;
};

// PROTOTYPE (WAJIB ADA SEMUA)
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
```

Guided 2

Graph.cpp

```
#include "graph.h"

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 && P->info != X)
        P = P->Next;
    return P;
}

// Graph tidak berarah → dua arah
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 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 || N->visited == 1)
        return;

    N->visited = 1;
    cout << N->info << " ";

    adrEdge E = N->firstEdge;
```

```
    while (E != NULL) {
        PrintDFS(G, E->Node);
        E = E->Next;
    }
}

#include <queue>
void PrintBFS(Graph G, adrNode N) {
    if (N == NULL) return;

    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;
        }
    }
}
```

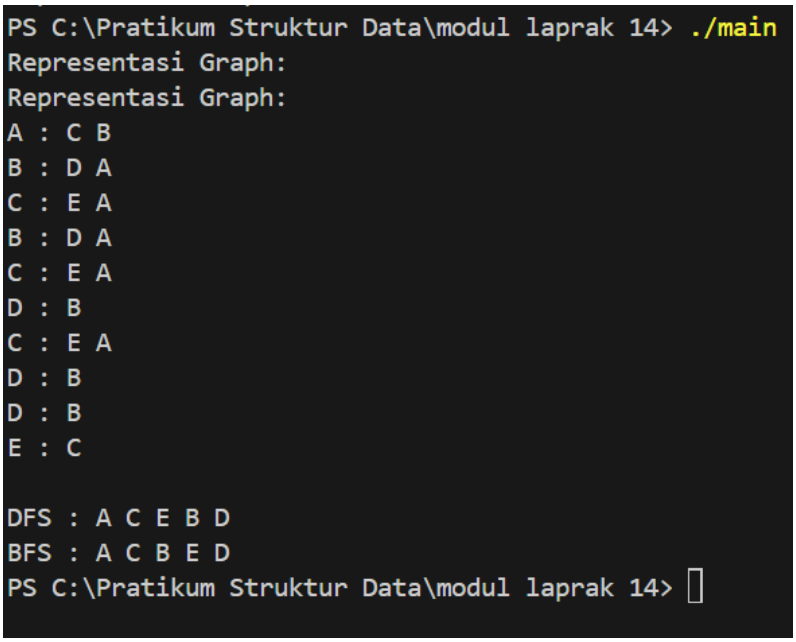
Guided 3

Main.cpp

```
#include "graph.h"

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

    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');

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(C, E);

    cout << "Representasi Graph:\n";
    PrintInfoGraph(G);

    cout << "\nDFS : ";
    ResetVisited(G);
    PrintDFS(G, A);

    cout << "\nBFS : ";
    ResetVisited(G);
    PrintBFS(G, A);

    return 0;
}
```

Screenshots Output



Deskripsi:

Modul 14 mempelajari struktur data graph yang terdiri dari node dan edge. Modul ini

membahas jenis graph, representasi graph menggunakan multilist, serta metode penelusuran DFS dan BFS. Implementasi graph dilakukan menggunakan bahasa pemrograman untuk memahami hubungan antar data secara terstruktur.

C. Kesimpulan

Graph adalah struktur data yang terdiri dari node dan edge untuk merepresentasikan hubungan antar data. Modul ini memperkenalkan jenis graph, representasi multilist, serta metode penelusuran DFS dan BFS. Praktikum ini membantu memahami cara kerja graph dan penerapannya dalam pemrograman.

D. Referensi

Modul 14