LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIV
PENGENALAN CODE BLOCKS

Disusun Oleh :
NAMA : Sinta Sintiani
NIM : 103112430047

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalah struktur data non-linier yang digunakan untuk merepresentasikan hubungan atau keterkaitan antar objek, di mana objek-objek tersebut disebut simpul (vertex) dan hubungan antar simpul disebut sisi (edge). Graph sering digunakan untuk memodelkan berbagai permasalahan nyata seperti jaringan komputer, peta jalan, hubungan sosial, dan sistem yang melibatkan relasi antar data.

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 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
```

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;
    }

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

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

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

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

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\mod 14> cd "d:\SEM 3\STRUKTUR DATA\Modul 14\" ; i
}
=== Struktur Graph ===
E -> C
D -> B
C -> E A
B -> D A
A -> C B

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

Deskripsi:

Program tersebut digunakan untuk membangun dan mengelola struktur data graph, yang memungkinkan penyimpanan data berupa node, penghubungan antar node, serta penelusuran hubungan antar node menggunakan metode DFS dan BFS.

Unguided 1

graph.h

```
#ifndef GRAPH_H
#define GRAPH_H
#include <iostream>

using namespace std;

typedef char infoGraph;

typedef struct ElmNode *adrNode;
typedef struct 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);
void ConnectNode(Graph &G, adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

adrNode FindNode(Graph G, infoGraph X);
void PrintDFS(Graph G, adrNode N);
void PrintBFS(Graph G, adrNode N);
void ResetVisited(Graph &G);

#endif
```

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);
    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(Graph &G, adrNode N1, adrNode N2) {
    if (N1 != NULL && N2 != NULL) {
        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) {
```

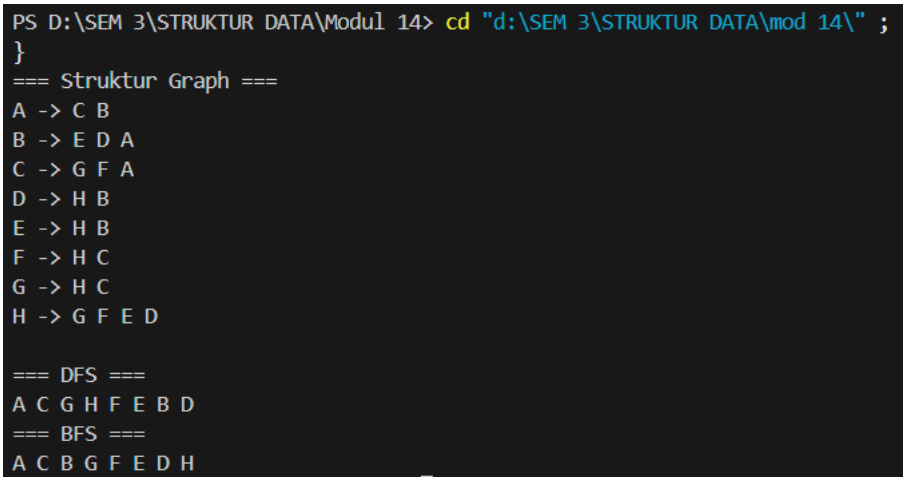
```
        P->visited = 0;
        P = P->next;
    }
}

void PrintDFS(Graph G, adrNode N) {
    if (N == NULL || N->visited == 1)
        return;

    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) {
    if (N == NULL)
        return;

    queue<adrNode> Q;
    N->visited = 1;
    Q.push(N);

    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;
        }
    }
}
```

Main.cpp

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

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');

    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 Gg = FindNode(G, 'G');
    adrNode H = FindNode(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, Gg);
    ConnectNode(G, D, H);
    ConnectNode(G, E, H);
    ConnectNode(G, F, H);
    ConnectNode(G, Gg, H);

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

    cout << "\n=== DFS ===\n"; // no 2
    ResetVisited(G);
    PrintDFS(G, A);

    cout << "\n=== BFS ===\n"; // no 3
    ResetVisited(G);
    PrintBFS(G, A);
```

```
    return 0;
}
```

Screenshots Output



Deskripsi:

Program tersebut mengimplementasikan graph tidak berarah di C++ menggunakan adjacency list, yang mendukung pembuatan node, penghubungan antar node, penampilan struktur graph, serta penelusuran DFS dan BFS dari node awal.

B. Kesimpulan

Melalui praktikum ini, kita dapat memahami konsep dan implementasi struktur data graph menggunakan bahasa C++, mulai dari pembuatan node, penghubungan antar node, hingga proses penelusuran graph menggunakan metode DFS dan BFS untuk merepresentasikan dan menganalisis hubungan antar data.

C. Referensi

https://www.geeksforgeeks.org/dsa/graph-data-structure-and-algorithms/

https://www.programiz.com/dsa/graph