LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL XIV

GRAPH

Disusun Oleh :

Nama : Izzah Minkhotun Fannisa

NIM : 103112400198

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Struktur data adalah metode untuk menyusun dan menyimpan informasi di dalam sistem komputer agar bisa diambil dan dikelola dengan efektif. Memilih struktur data yang sesuai sangat memengaruhi performa aplikasi, khususnya terkait konsumsi memori dan waktu pemrosesan informasi.

Graph adalah struktur data non-linear yang terdiri dari beberapa simpul *(vertex/node)* dan sisi *(edge)* yang menghubungkan antar simpul. Graph digunakan untuk menggambarkan hubungan atau keterkaitan antar objek, seperti jaringan komputer, peta, serta relasi sosial. Berdasarkan arah sisi, graph dapat dibagi menjadi dua jenis yaitu graph berarah *(directed graph)* dan graph tidak berarah *(undirected graph).* Pada graph tidak berarah, hubungan antar simpul memiliki sifat dua arah.

Adjacency list merupakan salah satu metode untuk merepresentasikan graf dengan menyimpan daftar simpul yang terhubung ke setiap simpul lainnya, di mana setiap node memiliki pointer ke daftar *edge* yang menunjukkan node-node tetangganya; representasi ini lebih efisien dalam penggunaan memori dibandingkan *adjacency matrix,* khususnya untuk graf dengan jumlah sisi yang relatif sedikit.

Node atau simpul berperan sebagai elemen utama graf yang menyimpan data, sedangkan *edge* atau sisi mewakili hubungan antar node; dalam implementasi *adjacency list*, setiap *edge* disimpan sebagai struktur yang menunjuk ke node tujuan, dan pada graf tidak berarah, setiap hubungan dicatat dua kali, yakni dari node pertama ke node kedua dan sebaliknya.

Di samping itu, *Depth First Search* adalah algoritma penelusuran graf yang beroperasi dengan mengunjungi satu node, kemudian menjelajah sedalam mungkin ke node tetangganya sebelum melakukan backtracking; DFS umumnya diimplementasikan secara rekursif dan memanfaatkan penanda kunjungan untuk mencegah pengulangan node.

Sementara itu*, Breadth First* Search merupakan algoritma penelusuran graf yang dilakukan secara melebar, dimulai dari node awal dengan mengunjungi seluruh node tetangga sebelum beralih ke tingkat berikutnya; BFS menggunakan struktur data *queue* untuk mengatur urutan kunjungan dan memastikan setiap node hanya dikunjungi satu kali.

B. Guide

1. Guide 1

a. Source Code

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 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) {
        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 = AllocatedEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    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;
    }
}

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 <iostream>
#include "graf.h"
#include "graf.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(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
```

```
    ConnectNode(G, 'C', 'E');

    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;
}
```

b. Screenshot Output

```
=== Struktur Graph ===
E->C
D->B
C->E A
B->D A
A->C B

=== Traversal DFS dari node A ===
A C E B D

=== Traversal BFS dari node A ===
A C B E D
```

c. Deskripsi Program

Program ini adalah implementasi struktur data graf tak berarah *(undirected graph)* dengan memanfaatkan *adjacency list* dalam bahasa pemrograman C++. Setiap node menyimpan data dalam bentuk karakter dan memiliki daftar tepi (edge) yang terhubung dengannya. Selain itu, program menyediakan fungsi untuk menjelajahi graf melalui algoritma *Depth First Search (DFS)* dan *Breadth First Search (BFS)*.

Struktur graf dikembangkan secara dinamis dengan bantuan pointer dan alokasi memori, yang berguna untuk memahami konsep struktur data non-linier serta penerapan teknik traversal pada graf.

Pembagian program :

a. File graf.h

a) Tipe data dan struktur

Bagian ini berisi deklarasi struktur data dan prototype fungsi yang digunakan dalam program.

- infoGraph : tipe data untuk menyimpan informasi node (berupa char)
- ElmNode : merepresentasikan simpul graph
- ElmEdge : merepresentasikan sisi *(edge)* antar node
- Graph : struktur utama graph yang menyimpan pointer ke node pertama

b) Deklarasi fungsi

- createGraph() : inisialisasi graph kosong
- AllocatedNode() : alokasi node baru
- AllocatedEdge() : alokasi edge baru
- InsertNode() : menambahkan node ke graph
- FindNode() : mencari node berdasarkan info
- ConnectNode() : menghubungkan dua node
- printInfoGraph() : menampilkan struktur graph
- ResetVisited() : mengatur ulang status kunjungan node
- printDFS() : traversal graph dengan DFS
- printBFS() : traversal graph dengan BFS

C. Unguide

1. Unguide 1

a. Source Code

graf.h

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

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 AlokasiNode(infoGraph X);
adrEdge AlokasiEdge(adrNode N);
void InsertNode(Graph &G, infoGraph X);
adrNode FindNode(Graph G, infoGraph X);
void ConnectNode(Graph &G, adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

#endif
```

graf.cpp

```
#include "graf.h"

void CreateGraph(Graph &G) {
    G.first = NULL;
}

adrNode AlokasiNode(infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}

adrEdge AlokasiEdge(adrNode N) {
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = AlokasiNode(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, adrNode N1, adrNode N2) {
    if (N1 == NULL || N2 == NULL)
        return;

    adrEdge E1 = AlokasiEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    adrEdge E2 = AlokasiEdge(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;
    }
}
```

main.cpp

```
#include <iostream>
#include "graf.h"
#include "graf.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');
    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 G1 = 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, G1);
    ConnectNode(G, D, H);
    ConnectNode(G, E, H);
    ConnectNode(G, F, H);
    ConnectNode(G, G1, H);

    cout << "Struktur Graph:" << endl;
    PrintInfoGraph(G);

    return 0;
}
```

b. Screenshot Output

```
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
```

c. Deskripsi Program

Program ini menerapkan struktur data graph tidak berarah dengan menggunakan representasi adjacency list dalam bahasa pemrograman C++. Graph tersusun atas node dan edge, di mana setiap node menyimpan informasi

berupa karakter serta daftar hubungan dengan node lain. Karena graph bersifat tidak berarah, setiap hubungan antar node dicatat secara dua arah.

Pada bagian pendefinisian struktur data, digunakan ElmNode untuk merepresentasikan node dan ElmEdge untuk merepresentasikan hubungan antar node. Struktur Graph berfungsi sebagai pengelola utama yang menyimpan alamat node pertama. Program menyediakan fungsi untuk menginisialisasi graph kosong, membuat node dan edge secara dinamis, menambahkan node ke dalam graph, mencari node tertentu, serta menghubungkan dua node.

Proses penghubungan node dilakukan dengan menambahkan edge pada masing-masing node yang saling terhubung. Setelah graph terbentuk, struktur graph ditampilkan dengan menampilkan setiap node beserta node-node yang terhubung dengannya. Program utama digunakan untuk melakukan pengujian dengan menambahkan node dari A hingga H, menghubungkan node-node tersebut sesuai pola yang ditentukan, dan menampilkan hasil akhir struktur graph ke layar.

2. Unguide 2

a. Source Code

graf.h

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

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 AlokasiNode(infoGraph X);
adrEdge AlokasiEdge(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);

#endif
```

graf.cpp

```
#include "graf.h"

void CreateGraph(Graph &G) {
    G.first = NULL;
}

adrNode AlokasiNode(infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}

adrEdge AlokasiEdge(adrNode N) {
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = AlokasiNode(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) return;

    adrEdge E1 = AlokasiEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    adrEdge E2 = AlokasiEdge(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;
    }
}
```

main.cpp

```
#include "graf.h"
#include "graf.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');

    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 << endl << "Hasil DFS dari node A:" << endl;
    ResetVisited(G);
    PrintDFS(G, FindNode(G,'A'));
    cout << endl;

    return 0;
}
```

b. Screenshot Output

```
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

Hasil DFS dari node A:
A C G H F E B D
```

c. Deskripsi Program

Program ini bertujuan untuk membuat dan mengelola struktur data graph tidak berarah menggunakan *adjacency list*. Setiap node direpresentasikan sebagai simpul yang menyimpan data berupa karakter, sedangkan hubungan antar node direpresentasikan sebagai edge.

Program bekerja dengan cara menginisialisasi graph kosong, kemudian menambahkan beberapa node ke dalam graph. Setelah itu, node-node tersebut dihubungkan satu sama lain secara dua arah untuk membentuk graph tidak berarah. Struktur graph ditampilkan dengan menunjukkan setiap node beserta node-node yang terhubung dengannya.

Selain menampilkan struktur graph, program juga melakukan penelusuran graph menggunakan algoritma *Depth First Search (DFS)*. Proses DFS dimulai dari node A dan berjalan secara rekursif dengan menandai node yang sudah dikunjungi agar tidak diproses kembali. Hasil traversal DFS kemudian ditampilkan ke layar sebagai urutan kunjungan node.

Unguide 3

a. Source Code

graf.h

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

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 PrintBFS(Graph &G, adrNode N);

#endif
```

graf.cpp

```
#include "graf.h"
#include <queue>

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 E = new ElmEdge;
    E->node = N;
    E->next = NULL;
    return E;
}

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) {
        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;
    }

    adrEdge E1 = AllocatedEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    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;
    }
}

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;
            }
        }
    }
}
```

main.cpp

```
#include <iostream>
#include "graf.h"
#include "graf.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');
    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 << endl << "Hasil BFS dari node A:" << endl;
    ResetVisited(G);
    PrintBFS(G, FindNode(G,'A'));
    cout << endl;

    return 0;
}
```

b. Screenshot Output

```
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

Hasil BFS dari node A:
A C B G F E D H
```

c. Deskripsi Program

Program ini dibuat untuk menerapkan struktur data graph tidak berarah dengan menggunakan representasi daftar keterhubungan node serta

menampilkan hasil penelusuran *Breadth First Search (BFS*). Untuk menjaga kerapian dan kemudahan pemahaman, implementasi program disusun secara terpisah berdasarkan fungsinya. Struktur graph didefinisikan melalui node dan edge, di mana setiap node menyimpan data berupa karakter, penanda kunjungan, serta hubungan ke node lain sehingga memungkinkan satu node terhubung dengan banyak node sekaligus.

Proses pembentukan graph dilakukan dengan menginisialisasi graph kosong, mengalokasikan node dan *edge* secara dinamis, menambahkan node ke dalam graph, serta menghubungkan node-node tersebut secara dua arah. Penelusuran graph dilakukan menggunakan algoritma BFS dengan bantuan struktur antrian *(queue),* di mana penelusuran dimulai dari node awal dan berkembang ke node-node tetangga secara bertahap sambil menandai node yang telah dikunjungi.

Pada tahap pengujian, graph dibentuk dengan node A hingga H sesuai dengan hubungan yang ditentukan, kemudian struktur graph ditampilkan dan hasil penelusuran BFS dari node A ditunjukkan setelah status kunjungan seluruh node diatur ulang

D. Kesimpulan

Berdasarkan hasil implementasi dan pengujian program, dapat disimpulkan bahwa struktur data graf tidak berarah dapat dibangun dengan efektif menggunakan representasi *adjacency list*. Pendekatan ini memungkinkan pengelolaan hubungan antar simpul secara dinamis dan efisien dalam hal penggunaan memori. Program ini berhasil menerapkan proses pembentukan graf, penambahan simpul, serta penghubungan simpul secara dua arah.

Selain itu, algoritma DFS dan BFS dapat diterapkan untuk menelusuri graf dengan karakteristik berbeda, di mana DFS menjelajahi graf secara mendalam, sedangkan BFS menjelajahinya secara melebar. Implementasi ini membantu memperkuat pemahaman tentang konsep struktur data non-linier, penggunaan pointer, serta penerapan algoritma traversal pada graf.

E. Referensi

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3rd ed.)*. MIT Press.

Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C++ (4th ed.).* Pearson Education.

GeeksforGeeks. (n.d.). *Graph Data Structure*. https://www.geeksforgeeks.org/graph-data-structure/

GeeksforGeeks. (n.d.). *Breadth First Search (BFS).* https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

GeeksforGeeks. (n.d.). *Depth First Search (DFS)*. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/