

**LAPORAN
MODUL 14
GRAPH**



Disusun Oleh :

NAMA :

Devi Nurliana

NIM :

103112400144

Dosen :

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO
2025**

A. DASAR TEORI

Struktur data *Graph* adalah tipe struktur data yang tidak berbentuk garis lurus, digunakan untuk menggambarkan hubungan antar data dalam bentuk simpul dan sisi. Setiap simpul mewakili suatu objek, sedangkan sisi menunjukkan hubungan atau koneksi antar objek tersebut. *Graph* bisa berupa graf berarah, di mana arah hubungan antar simpul diatur, atau graf tidak berarah, di mana hubungan tidak memiliki arah. *Graph* juga bisa memiliki bobot, yaitu nilai yang menunjukkan tingkat atau ukuran hubungan antar simpul, atau tidak memiliki bobot tergantung pada kebutuhan masalah yang ingin dipelajari. Dalam penerapannya, graph biasanya disajikan dalam bentuk matriks ketetanggaan atau daftar ketetanggaan. Daftar ketetanggaan lebih efektif digunakan ketika jumlah sisi dalam graph tidak terlalu banyak. Struktur data *Graph* digunakan dalam berbagai situasi nyata seperti jaringan komputer, sistem transportasi, peta, hubungan sosial, dan rekomendasi sistem. Struktur ini juga mendukung berbagai operasi seperti menambah simpul atau sisi, menelusuri *graph*, dan mencari jalur terbaik.

B. GUIDED

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;
    }

    // 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;
    }
}

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);

    //Tambah node
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');

    // hubungakan node (graf tidak berarah)
    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 ini berfungsi untuk mengimplementasikan struktur data graph tidak berarah menggunakan adjacency list serta melakukan traversal DFS dan BFS.

1. Pendefinisian struktur graph

Pada file *graf.h*, didefinisikan struktur node, edge, dan graph. Setiap node menyimpan data, penanda kunjungan, serta pointer ke edge dan node lain yang terhubung.

2. Pembuatan dan penghubungan node

Pada file *graf.cpp*, program menyediakan fungsi untuk membuat graph, menambahkan node, dan menghubungkan dua node. Prosedur *ConnectNode* membentuk graph tidak berarah dengan membuat hubungan dua arah antar node.

3. Traversal graph

Program mendukung dua metode penelusuran, yaitu DFS dan BFS. DFS dilakukan secara rekursif, sedangkan BFS menggunakan struktur data *queue*. Status kunjungan node diatur menggunakan variabel *visited*.

4. Penampilan hasil

Pada file *main.cpp*, program menampilkan struktur graph serta hasil traversal DFS dan BFS dari node awal, sehingga urutan kunjungan setiap node dapat diamati.

C. UNGUIDED

1. Unguided 1

a. Source Code

- *graf.h*

```

#ifndef GRAPH_H
#define GRAPH_H

#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 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;

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

    // edge N2 ke N1 (tidak berarah)
    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);  
  
    // Membuat node A sampai H  
    InsertNode(G, 'A');  
    InsertNode(G, 'B');  
    InsertNode(G, 'C');  
    InsertNode(G, 'D');  
    InsertNode(G, 'E');  
    InsertNode(G, 'F');  
    InsertNode(G, 'G');  
    InsertNode(G, 'H');  
  
    // Ambil alamat node  
    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');  
  
    // Hubungan sesuai ilustrasi  
    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 ini dibuat untuk mengimplementasikan struktur data *graph* tidak berarah menggunakan *linked list* sebagai representasi hubungan antar simpul.

1. Pendefinisian struktur graph

Pada file *graf.h*, program mendefinisikan struktur *ElmNode*, *ElmEdge*, dan *Graph*. Setiap node menyimpan informasi berupa karakter serta pointer ke daftar edge dan node berikutnya, sedangkan edge menyimpan hubungan antar node.

2. Proses pembuatan dan alokasi graph

Pada file *graf.cpp*, prosedur *CreateGraph*, *AlokasiNode*, dan *AlokasiEdge* digunakan untuk menginisialisasi graph kosong serta mengalokasikan memori untuk node dan edge yang akan membentuk graph.

3. Proses penyisipan dan penghubungan node

Node ditambahkan ke dalam graph menggunakan prosedur *InsertNode*. Hubungan antar node dibentuk dengan prosedur *ConnectNode* yang membuat edge dua arah sehingga graph bersifat tidak berarah.

4. Penampilan struktur graph

Pada file *main.cpp*, seluruh node dan hubungan yang telah dibuat ditampilkan menggunakan prosedur *PrintInfoGraph* sehingga struktur graph dapat dilihat sesuai dengan ilustrasi yang dibentuk.

2. Unguided 2

- a. Source Code
- *graf.h*

```

#ifndef GRAPH_H
#define GRAPH_H

#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 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, 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 ini bertujuan untuk mengimplementasikan struktur data *graph tidak berarah* menggunakan representasi *adjacency list*, serta menampilkan hubungan antar simpul dan hasil penelusuran *Depth First Search (DFS)*. Implementasi dibagi ke dalam tiga file utama, yaitu *graf.h*, *graf.cpp*, dan *main.cpp*.

1. Pendefinisian struktur graph

Pada file *graf.h*, program mendefinisikan tipe data *graph* yang terdiri dari node dan edge. Setiap node menyimpan informasi bertipe karakter, penanda kunjungan (*visited*), pointer ke edge pertama, dan pointer ke node berikutnya. Struktur ini memungkinkan *graph* menyimpan hubungan antar simpul secara dinamis.

2. Proses alokasi dan penyisipan node

Pada file *graf.cpp*, fungsi *AlokasiNode* dan *InsertNode* digunakan untuk membuat node baru dan memasukkannya ke dalam graph. Node disimpan dalam bentuk linked list sehingga penambahan simpul dapat dilakukan dengan fleksibel tanpa batas ukuran tetap.

3. Pembentukan hubungan antar node

Prosedur *ConnectNode* menghubungkan dua node sehingga membentuk graph tidak berarah. Setiap hubungan dibuat dua arah dengan menambahkan edge pada masing-masing node, sehingga keterhubungan antar simpul dapat ditelusuri dari kedua sisi.

4. Penelusuran dan penampilan graph

Program menyediakan prosedur *PrintInfoGraph* untuk menampilkan struktur graph dalam bentuk daftar ketetanggaan. Selain itu, prosedur *PrintDFS* digunakan untuk menampilkan hasil penelusuran graph secara DFS mulai dari node tertentu, dengan bantuan prosedur *ResetVisited* untuk mengatur ulang status kunjungan node sebelum penelusuran dimulai.

3. Unguided 3

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);
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
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 ini bertujuan untuk mengimplementasikan struktur data Graph tidak berarah menggunakan representasi adjacency list serta menampilkan hasil penelusuran BFS (Breadth First Search). Implementasi dibagi ke dalam beberapa file agar struktur program rapi dan mudah dipahami.

1. Pendefinisian struktur Graph

Pada file *graf.h*, program mendefinisikan struktur *Graph*, *ElmNode*, dan *ElmEdge*. Setiap node menyimpan informasi bertipe karakter, penanda kunjungan (*visited*), pointer ke edge pertama, serta pointer ke node berikutnya. Struktur ini memungkinkan satu node terhubung ke banyak node lain.

2. Proses alokasi dan pembentukan Graph

Pada file *graf.cpp*, disediakan fungsi *CreateGraph*, *AllocatedNode*, dan *AllocatedEdge* untuk menginisialisasi graph kosong serta mengalokasikan node dan edge baru. Prosedur *InsertNode* digunakan untuk menambahkan node ke dalam graph, sedangkan *ConnectNode* digunakan untuk menghubungkan dua node secara tidak berarah.

3. Penelusuran Graph menggunakan BFS

Masih pada file *graf.cpp*, prosedur *PrintBFS* diimplementasikan menggunakan struktur *queue*. Penelusuran dilakukan dengan mengunjungi node secara melebar dari node awal, menandai node yang sudah dikunjungi, dan memasukkan node tetangga ke dalam antrian.

4. Pengujian dan penampilan hasil

Pada file *main.cpp*, graph dibentuk dengan node A sampai H sesuai ilustrasi, kemudian antar node dihubungkan. Program menampilkan struktur graph menggunakan *PrintInfoGraph* dan menampilkan hasil penelusuran BFS dari node A setelah status kunjungan di-reset menggunakan *ResetVisited*.

D. KESIMPULAN

Berdasarkan ketiga implementasi yang dibuat, dapat disimpulkan bahwa struktur data *graph tidak berarah* dapat direpresentasikan secara efektif menggunakan *adjacency list* berbasis *linked list*. Implementasi ini memungkinkan pengelolaan node dan edge secara dinamis melalui proses alokasi memori, penyisipan node, serta pembentukan hubungan dua arah antar simpul. Selain menampilkan struktur graph, program juga menunjukkan kemampuan penelusuran graph melalui dua metode, yaitu *Depth First Search (DFS)* dan *Breadth First Search (BFS)*, yang masing-masing memiliki karakteristik penelusuran berbeda. Dengan pembagian program ke dalam file *graf.h*, *graf.cpp*, dan *main.cpp*, struktur kode menjadi lebih terorganisir, mudah dipahami, serta memudahkan pengembangan dan pengujian fitur graph secara bertahap.

E. REFERENSI

- Modul 14 Struktur Data. Graph. Program Studi Teknik Informatika. Universitas Telkom, 2025
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms. MIT Press, 2009.