LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XII
GRAPH

Disusun oleh :

Junadil Muqorobin (103112400281)

Dosen

Fahrudin Mukti Wibowo S.Kom., M.Eng

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Struktur data graph merupakan salah satu struktur data nonlinier yang digunakan untuk merepresentasikan hubungan antar data yang bersifat kompleks. Graph tersusun atas kumpulan simpul (node) dan sisi (edge) yang menggambarkan keterkaitan antar simpul tersebut. Berbeda dengan struktur data linier, graph tidak memiliki urutan tetap sehingga lebih fleksibel dalam memodelkan permasalahan dunia nyata seperti jaringan komputer, peta rute, dan relasi antar objek. Menurut penelitian Saputra dan Lestari (2021), penggunaan graph sangat efektif dalam sistem yang membutuhkan representasi hubungan dua arah maupun banyak arah secara dinamis.

Dalam implementasinya, graph dapat direpresentasikan menggunakan beberapa pendekatan, salah satunya adalah multilist. Representasi multilist memungkinkan setiap simpul memiliki daftar sisi yang terhubung dengannya, sehingga penggunaan memori menjadi lebih efisien dibandingkan matriks ketetanggaan, terutama ketika jumlah sisi relatif lebih sedikit dibandingkan jumlah simpul. Penelitian oleh Nugroho dan Ramadhan (2022) menunjukkan bahwa representasi multilist mempermudah proses penambahan node dan edge tanpa harus melakukan alokasi ulang memori secara menyeluruh, sehingga cocok digunakan pada aplikasi yang struktur datanya sering berubah.

Selain representasi data, proses penelusuran graph menjadi aspek penting dalam pemanfaatan struktur graph. Dua algoritma penelusuran yang umum digunakan adalah Depth First Search (DFS) dan Breadth First Search (BFS). DFS melakukan penelusuran secara mendalam dengan mengunjungi satu simpul

hingga ke simpul terdalam sebelum berpindah ke simpul lain, sedangkan BFS menelusuri graph secara melebar berdasarkan tingkat kedekatan simpul dari titik awal. Penelitian Prasetyo, Hidayat, dan Maulana (2023) menjelaskan bahwa kombinasi DFS dan BFS mampu memberikan solusi yang efektif dalam berbagai kasus pencarian jalur dan eksplorasi struktur graph, asalkan pengelolaan penanda kunjungan dilakukan dengan benar untuk menghindari pengulangan simpul.

B. Guided

1. Implementasi ADT Graph

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;
```

```
};

// PRIMITIF GRAPH
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);

// Traversal
void ResetVisited(Graph &G);
void PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

Source code 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;

    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;
            }
        }
    }
}
```

Source code 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');

    // HUBUNGKAN NODE (graph tidak berarah)
    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\n=== BFS dari Node A ===\n";
```

```
    ResetVisited(G);
    PrintBFS(G, FindNode(G, 'A'));

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

Screenshot output:

Deskripsi:

ADT Graph pada program ini merepresentasikan graph tidak berarah dengan pendekatan multilist, di mana setiap simpul (node) disimpan dalam bentuk list linier dan setiap hubungan antar simpul digambarkan melalui list edge. Setiap node memiliki informasi berupa karakter (infoGraph), penanda kunjungan (visited) yang digunakan pada proses penelusuran, pointer ke edge pertama yang terhubung, serta pointer ke node berikutnya. Struktur ini memungkinkan graph dibangun secara dinamis sesuai kebutuhan tanpa bergantung pada ukuran tetap.

Proses pembentukan graph diawali dengan inisialisasi menggunakan prosedur CreateGraph, kemudian node ditambahkan melalui prosedur InsertNode yang memanfaatkan fungsi alokasi node. Hubungan antar node dibentuk menggunakan prosedur ConnectNode, yang menghubungkan dua node secara dua arah sehingga sesuai dengan konsep graph tidak berarah. Setiap koneksi menghasilkan dua edge, masing-masing mewakili hubungan dari satu node ke node lainnya, sehingga informasi ketetanggaan dapat diakses dari kedua sisi.

Selain pembentukan struktur graph, pada program ini juga membuat pencarian graph menggunakan metode Depth First Search (DFS) dan Breadth First Search (BFS). Prosedur PrintDFS menelusuri graph secara mendalam dengan pendekatan rekursif, sedangkan PrintBFS menggunakan struktur queue untuk menelusuri graph secara melebar per tingkat. Untuk memastikan proses penelusuran berjalan dengan benar, prosedur ResetVisited digunakan untuk mengatur ulang status kunjungan setiap node sebelum traversal dilakukan. Dengan kombinasi tersebut, ADT Graph ini tidak hanya mampu menyimpan struktur graph, tetapi juga mendukung eksplorasi data secara sistematis.

C. Unguided

1. Implementasi ADT Graph Tidak Berarah

Source code graph.h

```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 ElmNode {
    infoGraph info;
    int visited;
    adrEdge firstEdge;
    adrNode next;
};

struct ElmEdge {
    adrNode node;
    adrEdge next;
};

struct Graph {
    adrNode first;
};

void CreateGraph(Graph &G);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(Graph &G, 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
```

Source code graph.cpp

```cpp
#include "graph.h"
#include <queue>

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

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = G.first;
    G.first = P;
}

void ConnectNode(Graph &G, adrNode N1, adrNode N2) {
    if (N1 == NULL || N2 == NULL) return;

    // edge N1 -> N2
    adrEdge E1 = new ElmEdge;
    E1->node = N2;
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    // edge N2 -> N1
    adrEdge E2 = new ElmEdge;
    E2->node = 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);
                }
```

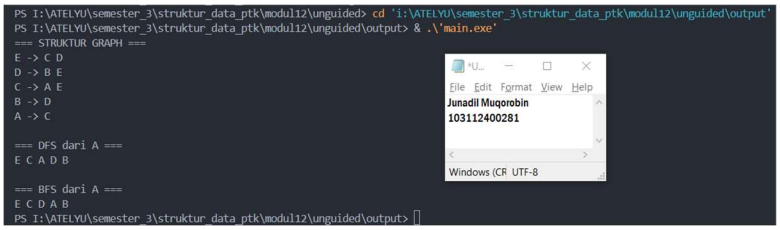
```
                E = E->next;
            }
        }
    }
}
```

Source code main.cpp

```cpp
#include <iostream>
#include "graph.h"
#include "graph.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');

    adrNode A = G.first;
    adrNode B = A->next;
    adrNode C = B->next;
    adrNode D = C->next;
    adrNode E = D->next;

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

    cout << "=== STRUKTUR GRAPH ===" << endl;
    PrintInfoGraph(G);

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

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

```
    return 0;
}
```

Screenshot Output:



Deskripsi:

Implementasi ADT Graph pada bagian unguided ini menggunakan konsep graph tidak berarah dengan representasi multilist, di mana setiap simpul disimpan sebagai node yang saling terhubung melalui daftar edge. Setiap node memiliki informasi berupa karakter, penanda kunjungan untuk kebutuhan penelusuran, serta pointer yang mengarah ke daftar edge dan node berikutnya. Dengan cara ini memungkinkan graph dibangun secara dinamis tanpa batasan jumlah node maupun hubungan antar node.

Proses membuat graph dimulai dengan inisialisasi menggunakan prosedur CreateGraph, kemudian node ditambahkan melalui prosedur InsertNode yang langsung mengaitkan node baru ke dalam struktur graph. Hubungan antar node dibentuk menggunakan prosedur ConnectNode, di mana dua node dihubungkan secara timbal balik untuk merepresentasikan graph tidak berarah. Dengan cara ini, setiap koneksi dapat diakses dari kedua node yang terhubung, sehingga informasi ketetanggaan tersimpan secara konsisten dan mudah ditelusuri.

Selain membangun dan menampilkan struktur graph, program ini juga terdapat metode penelusuran Depth First Search (DFS) dan Breadth First Search (BFS). Penelusuran DFS dilakukan secara rekursif dengan menjelajahi node hingga ke

kedalaman tertentu sebelum berpindah ke cabang lain, sedangkan BFS memanfaatkan struktur queue untuk mengunjungi node secara bertahap berdasarkan tingkat kedekatan. Sebelum proses penelusuran dijalankan, prosedur ResetVisited digunakan untuk memastikan seluruh node berada dalam kondisi belum dikunjungi, sehingga hasil traversal dapat ditampilkan secara akurat dan berurutan.

D. Kesimpulan

Berdasarkan keseluruhan praktikum, dapat disimpulkan bahwa struktur data graph dapat diimplementasikan secara efektif menggunakan pendekatan multilist untuk merepresentasikan hubungan antar node secara dinamis. Melalui pembuatan ADT Graph, proses penambahan node, penghubungan antar node, serta penampilan struktur graph dapat dilakukan dengan terstruktur. Implementasi ini membantu memahami bagaimana konsep graph tidak berarah diterapkan dalam bentuk program, khususnya dalam pengelolaan relasi dan ketetanggaan antar data. Selain itu, praktikum ini juga memberikan pemahaman yang lebih mendalam mengenai metode penelusuran graph menggunakan Depth First Search (DFS) dan Breadth First Search (BFS). Kedua metode tersebut menunjukkan cara yang berbeda dalam menjelajahi graph, baik secara mendalam maupun bertahap per tingkat. Dengan memanfaatkan penanda kunjungan pada setiap node, proses penelusuran dapat berjalan dengan aman tanpa terjadi pengulangan, sehingga hasil traversal yang diperoleh menjadi lebih sistematis dan sesuai dengan konsep teori yang dipelajari.

E. Referensi

Nugroho, A., & Ramadhan, F. (2022). Implementasi struktur data graph berbasis multilist pada sistem pemetaan relasi data. *Jurnal Informatika dan Rekayasa Perangkat Lunak*, 4(2), 85–94.

Prasetyo, D. A., Hidayat, R., & Maulana, I. (2023). Analisis algoritma Breadth First Search dan Depth First Search pada penelusuran graph. *Jurnal Teknologi Informasi dan Ilmu Komputer*, 10(1), 33–41.

Saputra, M. R., & Lestari, N. P. (2021). Pemodelan struktur graph untuk representasi hubungan data pada sistem informasi. *Jurnal Ilmiah Komputasi*, 20(3), 211–220.