LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XII
GRAPH

Disusun Oleh :

NAMA : RISKY CAHAYU

NIM : 103112430121

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Graph adalah struktur data yang terdiri dari dua komponen utama, yaitu node (atau biasa disebut juga sebagai vertex) dan edge (atau sisi) yang menghubungkan node-node tersebut. Dalam graf, node mewakili entitas atau objek yang ingin dianalisis, sementara edge menggambarkan hubungan atau koneksi antara dua node. Graph dapat digunakan untuk berbagai aplikasi, termasuk dalam pemrograman komputer, jaringan sosial, pemetaan, dan bahkan dalam perencanaan rute atau transportasi.

Terdapat dua jenis graph yang umum digunakan:

1. Graph Berarah (Directed Graph): Dalam graph berarah, setiap edge memiliki arah yang jelas dari satu node ke node lainnya. Artinya, jika ada hubungan antara node A dan node B, maka hanya node A yang mengarah ke node B, dan sebaliknya tidak berlaku kecuali ada edge terpisah dari B ke A. Graph berarah sering digunakan dalam masalah yang melibatkan urutan atau ketergantungan, seperti perencanaan proyek atau representasi struktur organisasi.
2. Graph Tidak Berarah (Undirected Graph): Berbeda dengan graph berarah, dalam graph tidak berarah, edge tidak memiliki arah. Artinya, jika node A terhubung ke node B, maka node B juga secara otomatis terhubung ke node A. Graph jenis ini sering digunakan untuk menggambarkan hubungan yang tidak memiliki arah spesifik, seperti dalam jaringan sosial, di mana dua orang yang saling berteman dapat dianggap memiliki hubungan yang saling menghubungkan.

Graph dapat direpresentasikan dengan berbagai metode, salah satunya adalah menggunakan adjacency matrix atau adjacency list. Pada adjacency matrix, graph direpresentasikan dalam bentuk matriks persegi yang menunjukkan apakah dua node terhubung atau tidak. Pada adjacency list, setiap node disimpan bersama dengan daftar node-node yang terhubung dengannya, yang lebih efisien dalam hal penggunaan ruang memori untuk graph yang jarang (sparse graph). Selain itu, dalam graf yang lebih kompleks, sering kali perlu dilakukan penelusuran atau traversal untuk mengunjungi node-node dalam urutan tertentu. Ada dua teknik utama yang digunakan untuk traversal:

1. Depth First Search (DFS): Dalam DFS, traversal dilakukan dengan mengeksplorasi sebanyak mungkin cabang dari node sebelum kembali. DFS dapat dilakukan secara rekursif atau dengan menggunakan stack untuk melacak node yang harus dikunjungi.
2. Breadth First Search (BFS): BFS berbeda dengan DFS karena traversal dilakukan level per level, mulai dari node awal dan kemudian mengunjungi node-node tetangga secara berurutan. BFS menggunakan queue untuk mengantri node yang akan dikunjungi.

Kedua teknik ini sangat penting dalam berbagai algoritma graf, seperti pencarian jalur terpendek, penentuan keterhubungan antar node, atau untuk menentukan siklus dalam graph. Meskipun DFS lebih efisien dalam menemukan komponen terhubung dalam graph yang dalam, BFS sering digunakan ketika kita mencari jalur terpendek antar dua node dalam graph yang tidak berbobot. Dalam praktikum ini, kita akan fokus pada

pengimplementasian graph menggunakan multilist. Multilist memungkinkan representasi graph dengan menggunakan linked list, di mana setiap node terhubung ke satu atau lebih node lainnya. Ini memberikan keuntungan dalam hal fleksibilitas dan efisiensi, karena node dan edge dapat dengan mudah diubah, ditambah, atau dihapus tanpa memerlukan struktur data statis seperti adjacency matrix. Dengan menggunakan multilist, kita dapat mengimplementasikan DFS dan BFS untuk menelusuri graph yang telah dibangun. Selain itu, dalam graph berarah, kita dapat memanfaatkan Topological Sorting untuk mengurutkan node dalam urutan linear berdasarkan ketergantungannya. Topological sorting banyak digunakan dalam aplikasi yang melibatkan urutan, seperti dalam penjadwalan tugas atau kompilasi kode.

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

Guided 1

Kode graf.cpp

```cpp
// Risky Cahayu
// 103112430121

#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;

    //Karena undirected -> buat edge balik
    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;
            }
        }
    }
}
```

Kode graf.h

```
// Risky Cahayu
// 103112430121

#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 DFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

Kode main.cpp

```
// Risky Cahayu
// 103112430121

#include "graf.h"
#include "graf.cpp"
#include <iostream>
using namespace std;

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

    //Tambah node
    InsertNode(G, 'A'); //0
    InsertNode(G, 'B'); //1
    InsertNode(G, 'C'); //2
```

```
    InsertNode(G, 'D'); //3
    InsertNode(G, 'E'); //4

    //Tambah edge
    ConnectNode(G, 'A', 'B'); //0 -> 1
    ConnectNode(G, 'A', 'C'); //0 -> 2
    ConnectNode(G, 'B', 'D'); //1 -> 3
    ConnectNode(G, 'C', 'E'); //2 -> 4

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

    cout << "\n=== DFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n=== BFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintBFS(G, FindNode(G, 'A'));

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

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat> cd "c:\Users\ASUS\Videos\strukdat\Modul 12\Guided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
=== 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
PS C:\Users\ASUS\Videos\strukdat\Modul 12\Guided>
```

Deskripsi:

Penjelasan Pada Guided 1, kita mengimplementasikan struktur graph menggunakan multilist, di mana setiap node dapat memiliki daftar edge yang menghubungkannya dengan node lain. Fungsi-fungsi yang digunakan antara lain InsertNode untuk menambahkan node baru ke dalam graph, ConnectNode untuk menghubungkan dua node dengan edge, dan PrintInfoGraph untuk menampilkan struktur graph. Implementasi Depth First Search (DFS) dan Breadth First Search (BFS) juga diperkenalkan untuk penelusuran graph. Dalam hal ini, DFS mengeksplorasi graph secara mendalam, sementara BFS mengeksplorasi level per level menggunakan stack dan queue, masing-masing.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

Kode graph.h

```
// Risky Cahayu
// 103112430121

#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 nextNode;
    adrEdge nextEdge;
};

struct ElmNode {
    infograph info;
    int visited;
    adrEdge firstEdge;
    adrNode nextNode;
};

struct Graph {
    adrNode first;
};

void CreateGraph(Graph &G);
adrNode InsertNode(Graph &G, infograph x);
void ConnectNode(adrNode from, adrNode to);
void PrintInfoGraph(Graph G);
void PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

Kode graph.cpp

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

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

```
}

adrNode InsertNode(Graph &G, infograph x) {
    adrNode N = new ElmNode;
    N->info = x;
    N->visited = 0;
    N->firstEdge = NULL;
    N->nextNode = G.first;
    G.first = N;
    return N;
}

void ConnectNode(adrNode from, adrNode to) {
    adrEdge E = new ElmEdge;
    E->nextNode = to;
    E->nextEdge = from->firstEdge;
    from->firstEdge = E;
}

void PrintInfoGraph(Graph G) {
    adrNode N = G.first;
    while (N != NULL) {
        cout << N->info << " -> ";
        adrEdge E = N->firstEdge;
        while (E != NULL) {
            cout << E->nextNode->info << " ";
            E = E->nextEdge;
        }
        cout << endl;
        N = N->nextNode;
    }
}

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->nextNode);
        E = E->nextEdge;
    }
}

void PrintBFS(Graph &G, adrNode start) {
```

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

    while (!Q.empty()) {
        adrNode N = Q.front();
        Q.pop();
        cout << N->info << " ";

        adrEdge E = N->firstEdge;
        while (E != NULL) {
            if (E->nextNode->visited == 0) {
                E->nextNode->visited = 1;
                Q.push(E->nextNode);
            }
            E = E->nextEdge;
        }
    }
}
```

Kode main.cpp

```
// Risky Cahayu
// 103112430121

#include "graph.h"

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

    adrNode A = InsertNode(G, 'A');
    adrNode B = InsertNode(G, 'B');
    adrNode C = InsertNode(G, 'C');
    adrNode D = InsertNode(G, 'D');
    adrNode E = InsertNode(G, 'E');
    adrNode F = InsertNode(G, 'F');
    adrNode Gg = InsertNode(G, 'G');
    adrNode H = InsertNode(G, 'H');

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

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

    ConnectNode(C, F);
    ConnectNode(C, Gg);
```

```
    ConnectNode(D, H);
    ConnectNode(E, H);
    ConnectNode(F, H);
    ConnectNode(Gg, H);

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

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

    adrNode N = G.first;
    while (N != NULL) {
        N->visited = 0;
        N = N->nextNode;
    }

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

    return 0;
}
```

Screenshots Output

Deskripsi:

Program di atas merupakan implementasi struktur data graph berarah (directed graph) yang direpresentasikan menggunakan adjacency list berbasis linked list, di mana setiap node (vertex) disimpan dalam linked list utama dan setiap node memiliki linked list edge yang menunjuk ke node tujuan. Struktur ElmNode menyimpan informasi node, penanda visited untuk traversal, serta pointer ke edge pertama dan node berikutnya, sedangkan ElmEdge menyimpan hubungan berarah ke node lain. Program menyediakan operasi CreateGraph untuk inisialisasi graph, InsertNode untuk menambahkan node baru, ConnectNode untuk membuat sisi berarah dari satu node ke node lain, dan PrintInfoGraph untuk menampilkan struktur graph beserta arah keterhubungannya. Selain itu, program mendukung penelusuran graph menggunakan Depth First Search (DFS) dan Breadth First Search (BFS) melalui fungsi PrintDFS dan PrintBFS dengan memanfaatkan atribut visited

agar node tidak dikunjungi lebih dari satu kali. Pada fungsi main, graph dibangun dengan delapan node yaitu A hingga H, kemudian dihubungkan secara berarah sehingga membentuk struktur bertingkat yang berujung pada node H. Setelah struktur graph ditampilkan, program melakukan traversal DFS dan BFS dimulai dari node A untuk memperlihatkan perbedaan urutan kunjungan node pada kedua metode penelusuran tersebut.

Kesimpulan

Berdasarkan Praktikum Modul XII tentang Graph, dapat disimpulkan bahwa graph merupakan struktur data yang digunakan untuk merepresentasikan hubungan antar objek dalam bentuk simpul (vertex) dan sisi (edge), baik berarah maupun tidak berarah. Struktur data ini sangat efektif untuk memodelkan relasi yang kompleks. Pada praktikum ini, graph diimplementasikan menggunakan adjacency list berbasis multi linked list dengan bahasa pemrograman C++, sehingga setiap simpul dapat memiliki lebih dari satu sisi yang terhubung ke simpul lain.

Mahasiswa mempelajari berbagai operasi dasar pada graph, seperti pembuatan graph, penambahan simpul, penghubungan antar simpul, serta proses penelusuran graph menggunakan algoritma Depth First Search (DFS) dan Breadth First Search (BFS). Algoritma DFS melakukan penelusuran secara mendalam hingga mencapai simpul terakhir sebelum kembali, sedangkan BFS menelusuri graph berdasarkan tingkat atau level kedekatan simpul. Pada graph berarah, arah sisi sangat berpengaruh terhadap jalur penelusuran sehingga hubungan antar simpul tidak selalu bersifat dua arah.

Secara keseluruhan, praktikum ini membantu mahasiswa memahami konsep graph, cara representasinya menggunakan pointer, serta penerapannya dalam berbagai permasalahan nyata seperti jaringan komputer, peta jalan, dan sistem relasi data yang membutuhkan analisis keterhubungan antar objek..

E. Referensi

GeeksforGeeks. (2024). Implementation of graph in C++. https://www.geeksforgeeks.org/cpp/implementation-of-graph-in-cpp/

Maulana, F., & Hidayati, R. (2022). Simulasi graf berarah C++ dengan SDL. Jurnal Teknologi Informasi, 8(3), 177–186. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-097.pdf

Yuliana, S., & Fadli, R. (2023). Implementasi grafik dalam C++ menggunakan daftar ketetanggaan. https://translate.google.com/translate?u=https%3A%2F%2Fwww.softwaretestinghelp.com%2Fgraph-implementation-cpp%2F&hl=id.