

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 14
GRAPH**



Disusun Oleh :
NAMA : AGUNG RAMADHAN
NIM : 103112430060

Dosen
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antar objek yang terdiri dari simpul (vertex) dan sisi (edge). Setiap simpul merepresentasikan suatu objek, sedangkan sisi menunjukkan hubungan antar simpul tersebut. Graph dapat berbentuk berarah atau tidak berarah, serta dapat memiliki bobot yang menyatakan nilai tertentu seperti jarak atau biaya. Graph banyak digunakan dalam berbagai permasalahan nyata seperti jaringan komputer, peta, sistem transportasi, dan manajemen data karena mampu memodelkan hubungan secara fleksibel dan efisien.

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

Guided 1

Kode graf.h

```
// AGUNG RAMADHAN
// 103112430060

#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 graf.cpp

```

// AGUNG RAMADHAN
// 103112430060

#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 main.cpp

```

// AGUNG RAMADHAN
// 103112430060

#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'));

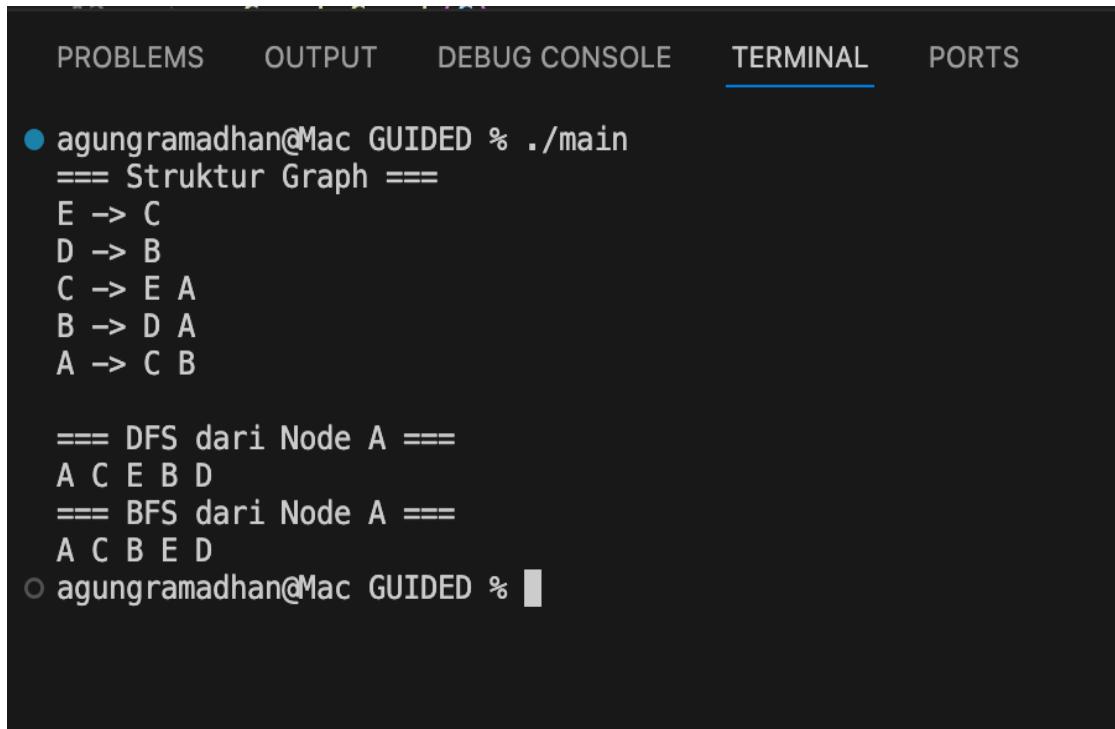
    cout << "\n==== BFS dari Node A ====\n";

```

```
    ResetVisited(G); //Reset visited semua node
    PrintBFS(G, FindNode(G, 'A'));

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

Screenshot:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● agungramadhan@Mac GUIDED % ./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
○ agungramadhan@Mac GUIDED %
```

Deskripsi:

ADT Graph di atas merepresentasikan graph tidak berarah menggunakan struktur multilist yang terdiri dari simpul (node) dan sisi (edge). Setiap node menyimpan informasi data, penanda kunjungan (visited), serta pointer ke daftar edge yang terhubung, sedangkan edge menyimpan alamat node tujuan. ADT ini menyediakan operasi dasar seperti pembuatan graph, penambahan node dan edge, pencarian node, serta penampilan struktur graph. Selain itu, ADT juga mendukung penelusuran graph menggunakan algoritma Depth First Search (DFS) dan Breadth First Search (BFS) untuk mengunjungi seluruh node berdasarkan hubungan ketetanggaannya.

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

Unguided 1

Kode graph.h

```
// AGUNG RAMADHAN
// 103112430060

#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 Node;
    adrEdge NextEdge;
};

struct elmNode
{
    infoGraph info;
    int visited;
    adrEdge firstEdge;
    adrNode Next;
};

struct Graph
{
    adrNode first;
};

// Prosedur & fungsi
void CreateGraph(Graph &G);
adrNode InsertNode(Graph &G, infoGraph x);
void ConnectNode(Graph &G, adrNode P, adrNode Q);
void PrintInfoGraph(Graph G);

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

#endif
```

Kode graph.cpp

```
// AGUNG RAMADHAN
// 103112430060

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

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

adrNode InsertNode(Graph &G, infoGraph x)
{
    adrNode P = new elmNode;
    P->info = x;
    P->visited = 0;
    P->firstEdge = NULL;
    P->Next = NULL;

    if (G.first == NULL)
    {
        G.first = P;
    }
    else
    {
        adrNode Q = G.first;
        while (Q->Next != NULL)
        {
            Q = Q->Next;
        }
        Q->Next = P;
    }
    return P;
}

void ConnectNode(Graph &G, adrNode P, adrNode Q)
{
    // P -> Q
    adrEdge E1 = new elmEdge;
    E1->Node = Q;
    E1->NextEdge = P->firstEdge;
    P->firstEdge = E1;

    // Q -> P (tidak berarah)
    adrEdge E2 = new elmEdge;
    E2->Node = P;
    E2->NextEdge = Q->firstEdge;
    Q->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->NextEdge;
        }
        cout << endl;
        P = P->Next;
    }
}

// ===== DFS =====
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->NextEdge;
    }
}

// ===== BFS =====
void PrintBFS(Graph G, adrNode N)
{
    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->NextEdge;
}
}
}

```

Kode main.cpp

```

// AGUNG RAMADHAN
// 103112430060

#include "graph.h"

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

    // Membuat node
    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');

    // Koneksi sesuai gambar
    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 << "Adjacency List:" << endl;
    PrintInfoGraph(G);

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

    // reset visited
    adrNode P = G.first;
}

```

```

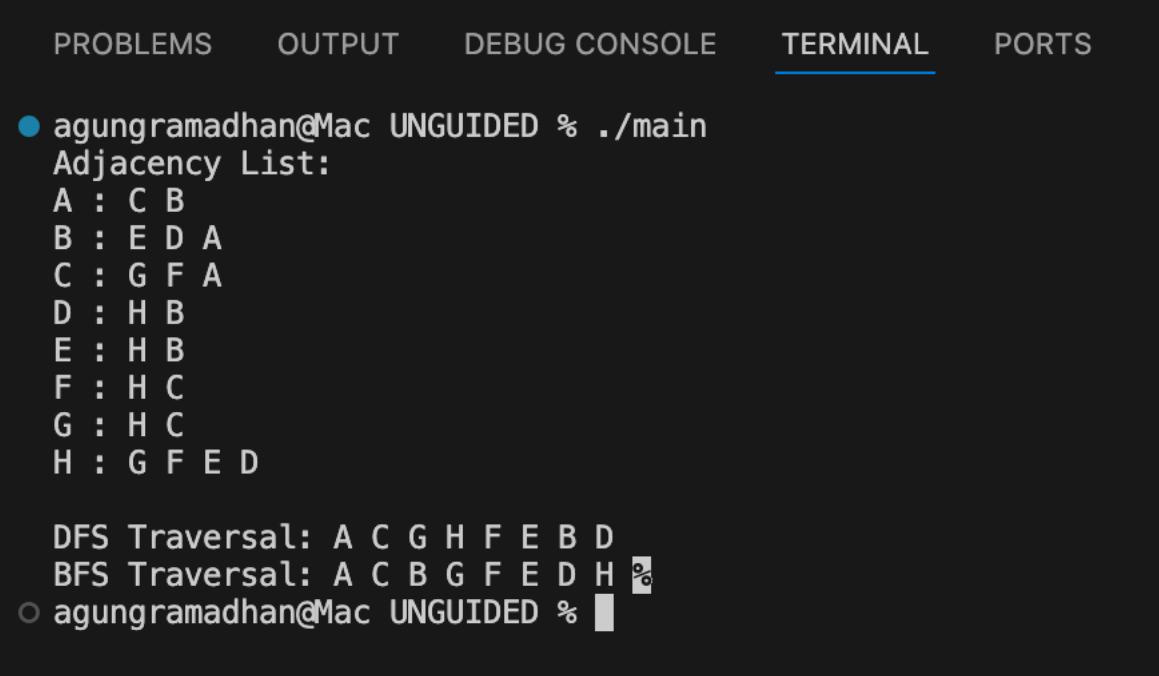
    while (P != NULL)
    {
        P->visited = 0;
        P = P->Next;
    }

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

    return 0;
}

```

Screenshots Output



PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

- agungramadhan@Mac UNGUIDED % ./main

Adjacency List:

A : C B

B : E D A

C : G F A

D : H B

E : H B

F : H C

G : H C

H : G F E D

DFS Traversal: A C G H F E B D

BFS Traversal: A C B G F E D H %
- agungramadhan@Mac UNGUIDED %

Deskripsi:

Kode di atas merupakan implementasi ADT Graph tidak berarah yang direpresentasikan menggunakan adjacency list berbasis pointer (multilist). Setiap node menyimpan informasi data, penanda kunjungan (*visited*), serta pointer ke daftar edge yang menghubungkannya dengan node lain. Graph mendukung operasi dasar seperti pembuatan graph, penambahan node, penghubungan antar node, dan penampilan struktur graph. Selain itu, kode ini mengimplementasikan metode penelusuran Depth First Search (DFS) dan Breadth First Search (BFS) untuk menelusuri seluruh node berdasarkan hubungan ketetanggaannya. Pada program utama, graph dibangun sesuai struktur yang ditentukan, kemudian ditampilkan dalam bentuk adjacency list serta hasil traversal DFS dan BFS dari node awal.

D. Kesimpulan

Kesimpulan dari implementasi ADT Graph ini adalah bahwa struktur data graph dapat digunakan secara efektif untuk merepresentasikan hubungan antar data dengan memanfaatkan adjacency list yang bersifat dinamis. Melalui operasi penambahan node, penghubungan node, serta penelusuran menggunakan algoritma DFS dan BFS, graph mampu menampilkan keterhubungan dan urutan kunjungan antar simpul dengan jelas. Implementasi ini menunjukkan bahwa ADT Graph memudahkan pemodelan masalah yang melibatkan relasi kompleks serta mendukung proses penelusuran data secara sistematis dan efisien.

E. Referensi

Setialana, P., & Ardiansyah, M. N. (2020). *Traversal Struktur Data Bipartite Graph dalam Graph Database menggunakan DFS*. ELINVO Journal.

Andriati, D. A., Dariato, E., & Hafizh, R. (2025). *Implementasi Teori Graf dan Optimisasi Algoritma Dijkstra, BFS dan DFS dalam Menentukan Rute Terpendek*. Jurnal Multimedia & Teknologi Informasi.

Khan, M. S., Ware, A., Habib, U., Khalid, M. J., & Bahoo, N. (2022). *Skeleton Based Human Action Recognition Using Doubly Linked List*. International Journal of Computer Trends and Technology, 70(2), 18-21.

Lohmann, S., & Tutsch, D. (2024). *The Doubly Linked Tree of Singly Linked Rings: Providing Hard Real-Time Database Operations on an FPGA*. Computers, 13(1), 8.