LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL I4
Graph

Disusun Oleh :
NAMA : Loh Suwargi Nitis Hamengku Bintang
NIM : 103112400116

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalah struktur data yang merepresentasikan himpunan objek (vertex/nodes) dan relasi antar-objek tersebut (edges). Dua representasi yang paling umum adalah *adjacency matrix* dan *adjacency list*. Untuk graf yang relatif jarang (sparse) seperti kebanyakan aplikasi dunia nyata adjacency list lebih efisien dalam penggunaan memori dan traversal karena hanya menyimpan tetangga yang ada; sebaliknya adjacency matrix memerlukan ruang $O(V^2)$ tetapi memberi keuntungan akses konstanta untuk memeriksa keberadaan edge. Pilihan antara keduanya berdampak langsung pada performa operasi dasar seperti iterasi tetangga, penambahan atau penghapusan edge, dan kompleksitas algoritma traversal.

Dua algoritma penelusuran fundamental pada graph adalah Depth-First Search (DFS) dan Breadth-First Search (BFS). DFS menelusuri sedalam mungkin di satu cabang sebelum backtracking biasanya diimplementasikan rekursif atau menggunakan stack dan berguna untuk mendeteksi komponen terhubung, menemukan *topological order*, atau mendeteksi siklus. BFS menggunakan queue dan menelusuri level demi level dari node awal, sehingga cocok untuk menemukan jalur terpendek pada graf tak-berbobot dan untuk analisis berjenjang (level-order). Kedua metode menandai node yang sudah dikunjungi (visited) agar tidak terjadi pengulangan kunjungan.

Struktur penyimpanan graph dan teknik traversal terus berkembang untuk memenuhi kebutuhan analisis graf skala besar dan temporal. Misalnya, penelitian menggabungkan teknik adjacency list dengan representasi CSR/kompak agar mendapat trade-off antara kecepatan akses baca dan efisiensi memori; pada saat bersamaan DFS/BFS tetap menjadi dasar di banyak aplikasi seperti segmentasi citra, sistem pakar, dan praproses untuk graph neural networks (GNNs). Implementasi dan optimasi struktur data graph yang efisien kini banyak dibahas dalam literatur karena pengaruhnya besar pada kemampuan pemrosesan graf besar dan aplikasi data science modern.

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

Guided 1

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
```

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()){
```

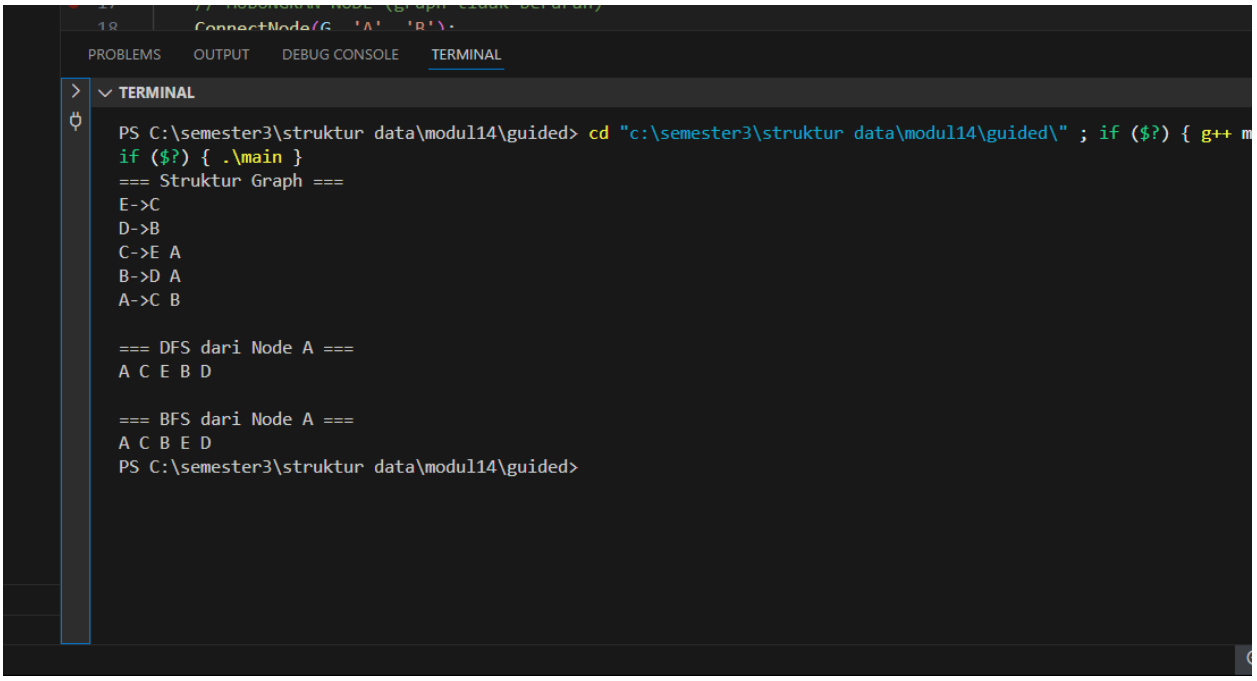
```
        adrNode curr = Q.front();
        Q.pop();

        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;
            }
        }
    }
}
```

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');

    // 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;
}
```

Screenshots Output



Deskripsi: program ini bertujuan untuk membangun sebuah ADT Graph tidak berarah dengan represntasi adjacency list. Setiap simpul (node) menyimpan informasi berupa karakter serta penanda kunjungan (visited), sedangkan hubungan antar simpul direpresentasikan dalam bentuk edge yang saling terhubung dua arah. Program ini juga menggunakan metode DFS dan BFS dimulai dari Node A

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

Unguided 1

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 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);

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 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;
   }

   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){
```

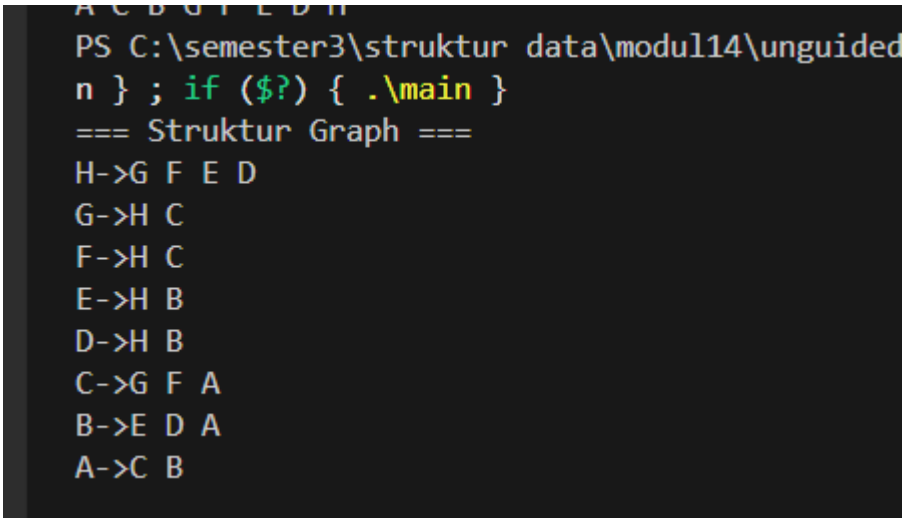
```
            if(E->node->visited == 0){
                Q.push(E->node);
            }
            E = E->next;
        }
    }
  }
}
```

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');
    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 ===\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;
}
```

Screenshots Output



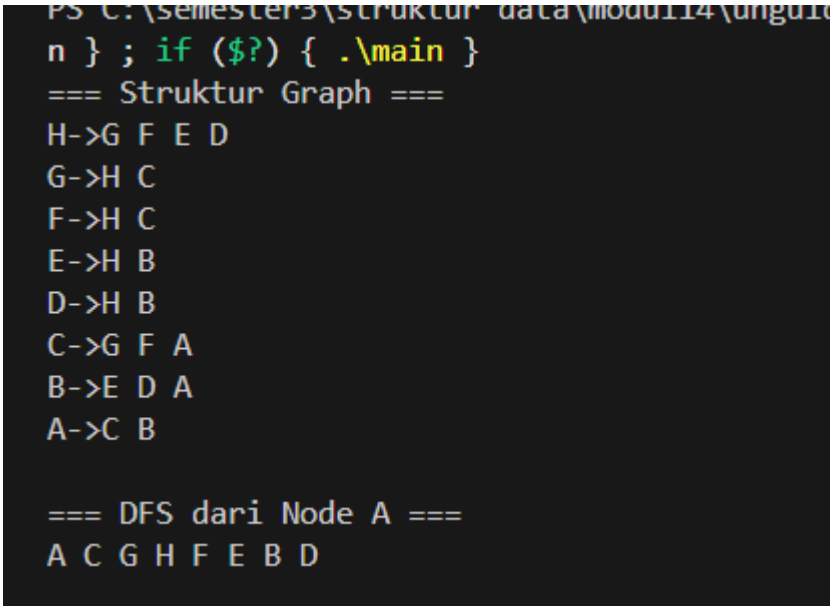
Deskripsi: program ini bertujuan untuk membangun sebuah ADT Graph tidak berarah dengan represntasi adjacency list. Setiap simpul (node) menyimpan informasi berupa karakter serta penanda kunjungan (visited), sedangkan hubungan antar simpul direpresentasikan dalam bentuk edge yang saling terhubung dua arah.

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

Unguided 2

Untuk program nomor 2 digabung menjadi 1 program saja

Screenshots Output



Deskripsi: prosedur penelusuran graph menggunakan metode DFS. Proses penelusuran

dimulai dari sebuah node awal dan dilanjutkan secara rekursif ke node tetangga yang belum dikunjungi hingga mencapai node terdalam. Setiap node yang dikunjungi akan ditandai sebagai telah dikunjungi untuk mencegah pengulangan penelusuran.

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

Unguided 3

Untuk program soal nomer 3 itu digabung dengan soal nomor 1

Screenshots Output

```
PS C:\semester3\struktur data\modu
n } ; if ($?) { .\main }
=== 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

=== DFS dari Node A ===
A C G H F E B D

=== BFS dari Node A ===
A C B G F E D H
```

F. Deskripsi: prosedur penelusuran graph menggunakan metode BFS. Penelusuran dilakukan dengan memanfaatkan struktur data queue, sehingga node yang berada pada tingkat terdekat dari node awal akan dikunjungi terlebih dahulu. Setiap node yang telah diproses ditandai sebagai visited agar tidak diproses ulang.

G. Kesimpulan

Penerapan konsep graph, menunjukkan bahwa representasi adjacency list merupakan pilihan yang tepat untuk membangun graph tidak berarah secara efisien. Struktur ADT yang digunakan mampu mengelola node dan edge dengan baik serta menyediakan operasi dasar yang mendukung proses pengolahan graph. Hal ini membuktikan bahwa pemahaman teori graph dapat diimplementasikan secara langsung ke dalam program dengan struktur yang jelas dan terorganisasi.

Implementasi algoritma Depth First Search (DFS) dan Breadth First Search (BFS) pada soal 2 dan 3 berhasil menampilkan hasil penelusuran sesuai karakteristik masing-masing metode. DFS menelusuri graph secara mendalam, sedangkan BFS menelusuri graph berdasarkan tingkat kedekatan node. Hasil yang diperoleh menunjukkan bahwa program telah berjalan dengan benar dan selaras dengan dasar teori yang digunakan.

H. Referensi

Arul, S. M., Senthil, G., Jayasudha, S., Alkhayyat, A., Azam, K., & Elangovan, R. (2023). Graph theory and algorithms for network analysis. In *E3S Web of Conferences* (Vol. 399, p. 08002). EDP Sciences.

Aranski, A. W. (2022). Depth First Search Algorithm In Solving the Shortest Route Using the Concept of Generate and Test. *IJISTECH (International Journal of Information System and Technology)*, *6*(3), 353-360.

Nicolas, F., & Suryantara, I. G. N. (2022). Implementasi algoritma breadth first search dan depth first search pada aplikasi kimia hidrokarbon berbasis augmented reality. *CogITo Smart Journal*, *8*(1), 194-205.