LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIV
GRAPH

Disusun Oleh :
NAMA : HILMI HAKIM RAMADANI
NIM : 103112430016

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph merupakan himpunan tidak kosong dari node (vertec) dan garis penghubung (edge). Contoh sederhana tentang graph, yaitu antara Tempat Kost Anda dengan Common Lab. Tempat Kost Anda dan Common Lab merupakan node (vertec). Jalan yang menghubungkan tempat Kost dan Common Lab merupakan garis penghubung antara keduanya (edge).

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

Guided

#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');

    ConnectNode(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
    ConnectNode(G, 'C', 'D');
    ConnectNode(G, 'D', '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;
}
```

#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); // diperbaiki
   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 E = AllocateEdge(N2);
    E->next = N1->firstEdge;
    N1->firstEdge = E;

    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;
    }
```

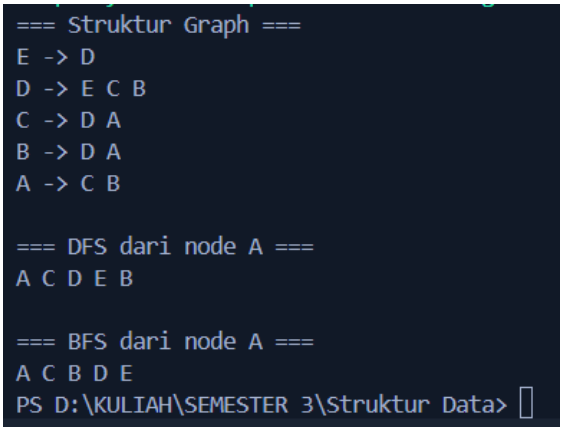
```
}

void PrintBFS(Graph &G, adrNode N)
{
   if (N == NULL)
      return;

   queue<adrNode> Q;
   Q.push(N); // diperbaiki

   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); // diperbaiki
            }
            E = E->next;
         }
      }
   }
}
```

#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; // <<< sesuaikan dengan graf.cpp (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); // <<< diperbaiki (bukan void)

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
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data graph tak berarah menggunakan adjacency list. Program membuat beberapa node (A–E), menghubungkannya dengan edge dua arah, lalu menampilkan struktur graph. Selain itu, program juga menerapkan

algoritma Depth First Search (DFS) dan Breadth First Search (BFS) yang dimulai dari node A, dengan memanfaatkan penanda `visited` untuk mencegah pengunjungan ulang node, sehingga traversal graph dapat berjalan dengan benar dan terstruktur.

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

Unguided

#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');

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

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

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

   adrNode P = G.First;
   while (P != NULL)
   {
      P->visited = 0;
      P = P->Next;
   }

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

   return 0;
}
```

#graf.cpp

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

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 E = new ElmEdge;
   E->Node = N;
   E->Next = NULL;
   return E;
}

void InsertNode(Graph &G, infoGraph X)
{
   adrNode P = AllocateNode(X);
   if (G.First == NULL)
   {
      G.First = P;
   }
   else
   {
      adrNode Q = G.First;
      while (Q->Next != NULL)
         Q = Q->Next;
      Q->Next = 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(adrNode N1, adrNode 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 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->Next;
    }
}

void PrintBFS(Graph G, adrNode N)
```

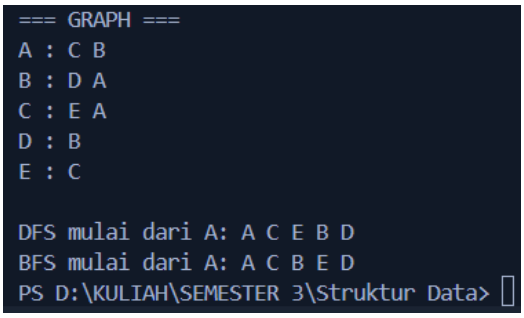
```
{
   queue<adrNode> Q;
   Q.push(N);
   N->visited = 1;

   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->Next;
      }
   }
}
```

#graf.h

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE

#include <iostream>
using namespace std;

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;

struct ElmEdge
{
   adrNode Node;
   adrEdge Next;
};

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

```
struct Graph
{
   adrNode First;
};

void CreateGraph(Graph &G);
adrNode AllocateNode(infoGraph X);
adrEdge AllocateEdge(adrNode N);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
adrNode FindNode(Graph G, infoGraph X);
void PrintInfoGraph(Graph G);

void PrintDFS(Graph G, adrNode N);
void PrintBFS(Graph G, adrNode N);

#endif
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan graph tak berarah menggunakan adjacency list dengan node bertipe karakter. Program membuat graph, menambahkan beberapa simpul (A–E), menghubungkannya dengan edge dua arah, lalu menampilkan struktur graph. Selain itu, program juga melakukan traversal DFS (Depth First Search) dan BFS (Breadth First Search) yang dimulai dari node A dengan memanfaatkan atribut visited untuk menandai node yang sudah dikunjungi agar tidak diproses berulang.

D. Kesimpulan

Berdasarkan praktikum Modul tentang Graph, dapat disimpulkan bahwa struktur data graph mampu merepresentasikan hubungan antar objek secara fleksibel melalui simpul (vertex) dan sisi (edge). Implementasi graph menggunakan adjacency list terbukti efisien dalam penyimpanan dan pengelolaan data, khususnya untuk graph tak berarah. Melalui

penerapan algoritma Depth First Search (DFS) dan Breadth First Search (BFS), mahasiswa dapat memahami perbedaan cara penelusuran graph secara mendalam dan melebar, serta pentingnya penggunaan penanda visited untuk mencegah kunjungan node secara berulang. Praktikum ini membantu meningkatkan pemahaman konsep graph dan traversalnya, serta melatih kemampuan implementasi struktur data dan algoritma dalam bahasa pemrograman C++.

E. Referensi

Modul 14 GRAPH