LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL XIV (14)
GRAPH

Disusun Oleh :

NAMA : Gien Darrel Adli NIM : 10312430008

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Algoritma merupakan fondasi dalam pembuatan program komputer. Secara sederhana, algoritma adalah serangkaian langkah logis dan sistematis yang disusun untuk menyelesaikan suatu masalah. C++, berfungsi sebagai alat untuk mengimplementasikan algoritma tersebut agar dapat dimengerti dan dieksekusi oleh komputer. C++ sering digunakan sebagai bahasa pengantar untuk mempelajari konsep pemrograman dasar karena strukturnya yang terorganisir dan kemampuannya untuk menangani operasi tingkat rendah.

Untuk membangun logika dalam program sesuai dengan algoritma yang dirancang, C++ menyediakan struktur kontrol, yang terbagi menjadi dua jenis utama:

Struktur Percabangan (Conditional): Digunakan untuk pengambilan keputusan, di mana program akan menjalankan blok kode tertentu jika suatu kondisi terpenuhi. Struktur ini mencakup if-else untuk mengevaluasi kondisi boolean dan switch-case untuk memilih blok kode berdasarkan nilai dari sebuah variabel.

Struktur Perulangan (Looping): Digunakan untuk mengeksekusi blok kode yang sama secara berulang kali selama kondisi tertentu masih terpenuhi. C++ menyediakan tiga jenis perulangan utama: for, while, dan do-while, yang masing-masing memiliki karakteristik penggunaan yang spesifik dalam implementasi algoritma.

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;     // <<< 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
```

Deskripsi:

File graf.h ini berisi definisi struktur data graf (node, edge, dan graph) serta deklarasi fungsi-fungsi untuk membuat graf, menambah dan menghubungkan node, mencari node, menampilkan isi graf, dan melakukan traversal DFS serta BFS, sehingga graf dapat dikelola dan diuji sesuai konsep dasar graf.

Guided 2 (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;
            }
         }
      }
}
```

Deskripsi:

File **graf.cpp** ini berisi implementasi fungsi-fungsi untuk mengelola graf berbasis adjacency list. Program dimulai dengan membuat graf kosong, lalu menyediakan fungsi untuk alokasi node dan edge. Selanjutnya, node dapat ditambahkan ke dalam graf dan dihubungkan satu sama lain secara dua arah. File ini juga mengatur penampilan isi graf serta traversal menggunakan DFS dan BFS dengan memanfaatkan penanda visited agar node tidak dikunjungi berulang.

Guided 3 (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');
```

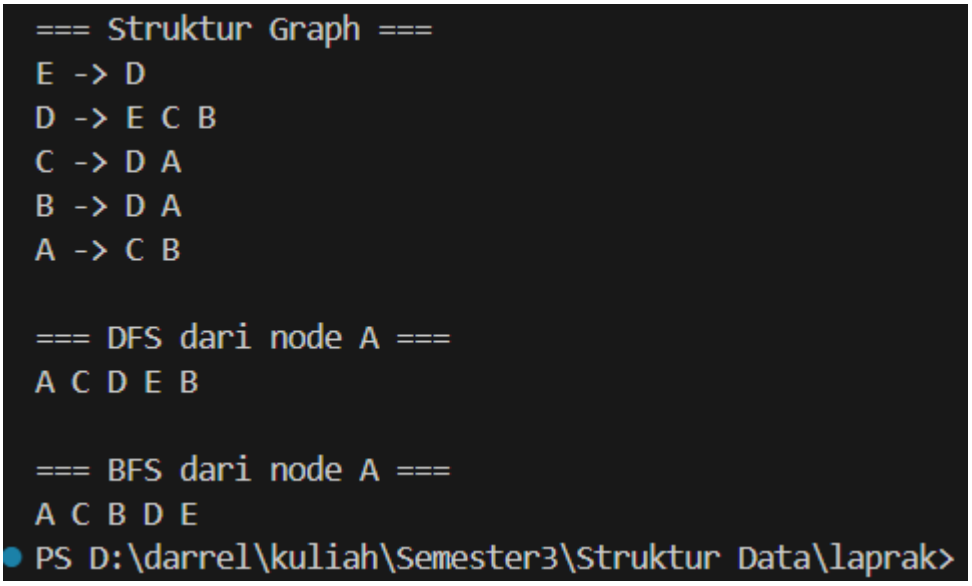
```
    insertNode(G, 'D');
    insertNode(G, 'E');

    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;
}
```

Deskripsi:

Program main.cpp ini berfungsi untuk menguji implementasi graf berbasis adjacency list. Program pertama membuat graf kosong, lalu menambahkan beberapa node (A, B, C, D, E) menggunakan insertNode. Setelah itu, node-node tersebut dihubungkan menggunakan ConnectNode untuk membentuk graf sesuai struktur yang diinginkan.

Selanjutnya, program menampilkan seluruh isi graf dengan PrintInfoGraph untuk melihat hubungan antar node. Kemudian, program melakukan traversal DFS dan BFS mulai dari node A, dengan memanggil ResetVisited sebelum traversal agar status kunjungan node kembali ke awal. Hasilnya menunjukkan urutan kunjungan node sesuai algoritma DFS dan BFS. Program ini bertujuan memastikan bahwa operasi graf, penghubungan node, dan traversal berjalan dengan benar**.**

Screenshots Output :



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

graph.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
```

Deskripsi:

File graph.h berisi definisi struktur node dan edge untuk graf berbasis adjacency list, serta deklarasi fungsi untuk membuat graf, menambah dan menghubungkan node, mencari node, menampilkan graf, dan melakukan traversal DFS/BFS.

graph.cpp

```
#include "graph.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;
      }
   }
}
```

Deskripsi:

File graph.cpp berisi implementasi fungsi-fungsi untuk mengelola graf berbasis adjacency list. Fungsi-fungsi ini meliputi pembuatan graf kosong, alokasi node dan edge, penambahan node, penghubungan dua node, pencarian node, penampilan isi graf, serta traversal menggunakan DFS dan BFS dengan memanfaatkan penanda visited agar node tidak dikunjungi lebih dari sekali.

main.cpp

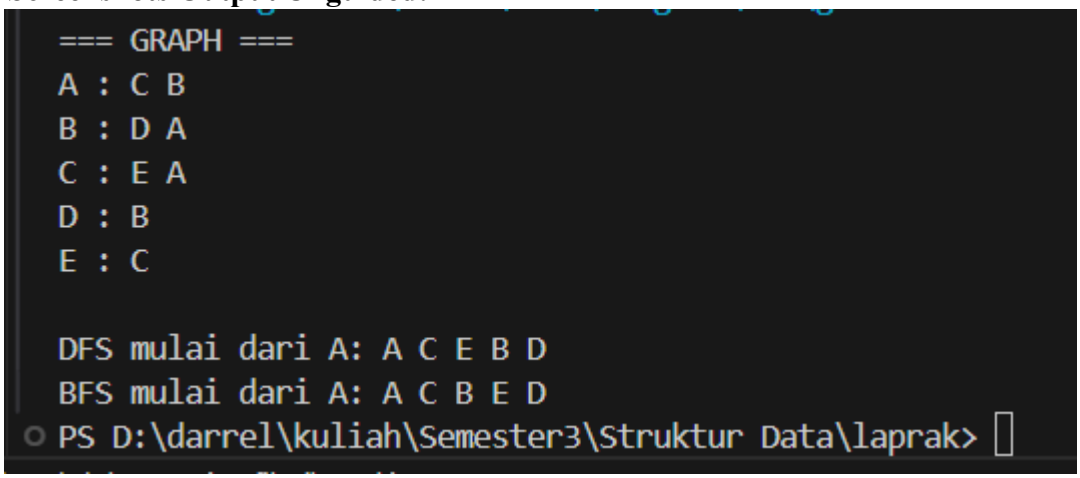
```
#include <iostream>
#include "graph.h"
#include "graph.cpp"

using namespace std;

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

```
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');

    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;
}
```

Deskripsi:

Program main.cpp ini berfungsi untuk menguji implementasi graf berbasis adjacency list. Program membuat graf kosong, menambahkan node A sampai E, lalu menghubungkan node-node tersebut dengan ConnectNode sesuai struktur graf yang diinginkan. Setelah itu, isi graf ditampilkan menggunakan PrintInfoGraph. Program kemudian melakukan traversal DFS dan BFS mulai dari node A, dengan mereset status visited sebelum BFS agar semua node dapat dikunjungi ulang. Hasilnya menunjukkan urutan kunjungan node sesuai algoritma DFS dan BFS.

Screenshots Output Unguided:



D. Kesimpulan

Kesimpulan modul 14 ini adalah mempelajari dan mengimplementasikan struktur data graf berbasis adjacency list menggunakan bahasa C++. Pada modul ini dibuat file header untuk mendefinisikan struktur node dan edge beserta pointer dan status kunjungan node, serta deklarasi fungsi-fungsi untuk membuat graf, menambah node, menghubungkan node, mencari node, menampilkan isi graf, dan melakukan traversal DFS dan BFS. File implementasi merealisasikan seluruh operasi tersebut secara terstruktur, sedangkan program utama digunakan untuk menguji pembuatan graf, penghubungan node, dan traversal sehingga konsep graf, relasi antar simpul, dan algoritma penelusuran dapat dipahami dan dijalankan sesuai teori.

E. Referensi

Kaswar, A. B., & Zain, S. G. (2021). Mudah Belajar Pemrograman Dasar C++. Syiah Kuala University Press.

Hanief, S., Jepriana, I. W., & Kom, S. (2020). Konsep Algoritme dan Aplikasinya dalam Bahasa Pemrograman C++. Penerbit Andi.

Imamuddin, A., & Sobarnas, M. A. (2021). PEMBELAJARAN JARAK JAUH PEMROGRAMAN DASAR MENGGUNAKAN BAHASA C++ UNTUK UMUM: SEBUAH PROGRAM PENGABDIAN KEPADA MASYARAKAT. BEMAS: Jurnal Bermasyarakat, 1(2), 59-67.