LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL XIV
GRAPH

Disusun Oleh :
NAMA : ADIKA AUNURFIKRI NOVIYANTO
NIM : 103112400195

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

C++ adalah pengembangan dari dari bahasa c yang dibuat oleh Bjarne Stroustrup sekitar tahun 1980-an. C++ disebut bahasa multi-paradigma, artinya bisa dipakai dengan gaya prosedural (pakai fungsi biasa), beriorientasi objek (pakai class dan object), atau bahkan gabungan keduanya. C++ punya dasar-dasar seperti variabel, operator percabangan (if, switch), perulangan (for, while), dan bisa memakai class untuk membuat objek.

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

Guided 1

graph.h

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct 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);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);
adrNode FindNode(Graph G, infoGraph X);

#endif
```

graph.cpp

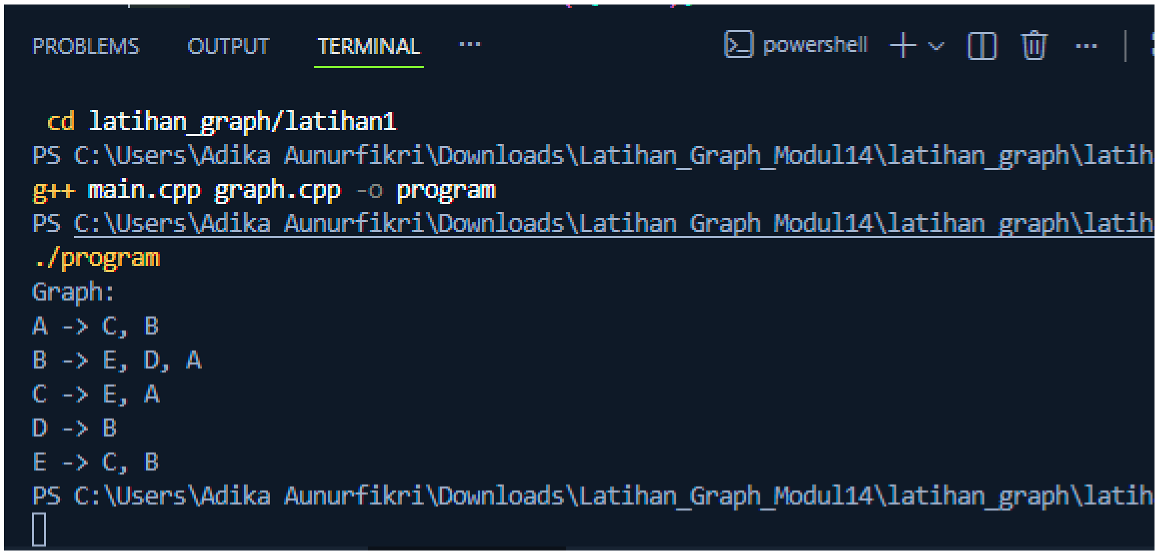
```
#include <iostream>
#include "graph.h"
using namespace std;

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

void InsertNode(Graph &G, infoGraph X) {
   adrNode newNode = new ElmNode;
   newNode->info = X;
   newNode->visited = 0;
   newNode->firstEdge = NULL;
   newNode->Next = NULL;

   if (G.first == NULL) {
      G.first = newNode;
   } else {
      adrNode temp = G.first;
      while (temp->Next != NULL) {
         temp = temp->Next;
      }
      temp->Next = newNode;
   }
}

adrNode FindNode(Graph G, infoGraph X) {
   adrNode temp = G.first;
   while (temp != NULL) {
      if (temp->info == X) {
         return temp;
      }
      temp = temp->Next;
   }
```

```
    return NULL;
}

void ConnectNode(adrNode N1, adrNode N2) {
    adrEdge newEdge1 = new ElmEdge;
    newEdge1->Node = N2;
    newEdge1->Next = N1->firstEdge;
    N1->firstEdge = newEdge1;

    adrEdge newEdge2 = new ElmEdge;
    newEdge2->Node = N1;
    newEdge2->Next = N2->firstEdge;
    N2->firstEdge = newEdge2;
}

void PrintInfoGraph(Graph G) {
    adrNode tempNode = G.first;

    cout << "Graph:" << endl;
    while (tempNode != NULL) {
        cout << tempNode->info << " -> ";
        adrEdge tempEdge = tempNode->firstEdge;
        while (tempEdge != NULL) {
            cout << tempEdge->Node->info;
            if (tempEdge->Next != NULL) {
                cout << ", ";
            }
            tempEdge = tempEdge->Next;
        }
        cout << endl;
        tempNode = tempNode->Next;
    }
}
```

Main.cpp

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

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

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

    adrNode nodeA = FindNode(G, 'A');
    adrNode nodeB = FindNode(G, 'B');
    adrNode nodeC = FindNode(G, 'C');
    adrNode nodeD = FindNode(G, 'D');
    adrNode nodeE = FindNode(G, 'E');

    ConnectNode(nodeA, nodeB);
    ConnectNode(nodeA, nodeC);
    ConnectNode(nodeB, nodeD);
    ConnectNode(nodeB, nodeE);
    ConnectNode(nodeC, nodeE);

    PrintInfoGraph(G);

    return 0;
}
```

Screenshots Output:



```
 cd latihan_graph/latihan1
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\latih
g++ main.cpp graph.cpp -o program
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\latih
./program
Graph:
A -> C, B
B -> E, D, A
C -> E, A
D -> B
E -> C, B
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\latih
```

Guided 2

graph.h

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct 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);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);
```

```
adrNode FindNode(Graph G, infoGraph X);
void ResetVisited(Graph &G);
void PrintDFS(Graph G, adrNode N);

#endif
```

graph..cpp

```
#include <iostream>
#include "graph.h"
using namespace std;

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

void InsertNode(Graph &G, infoGraph X) {
   adrNode newNode = new ElmNode;
   newNode->info = X;
   newNode->visited = 0;
   newNode->firstEdge = NULL;
   newNode->Next = NULL;

   if (G.first == NULL) {
      G.first = newNode;
   } else {
      adrNode temp = G.first;
      while (temp->Next != NULL) {
         temp = temp->Next;
      }
      temp->Next = newNode;
   }
}

adrNode FindNode(Graph G, infoGraph X) {
   adrNode temp = G.first;
```

```
    while (temp != NULL) {
        if (temp->info == X) {
            return temp;
        }
        temp = temp->Next;
    }
    return NULL;
}

void ConnectNode(adrNode N1, adrNode N2) {
    adrEdge newEdge1 = new ElmEdge;
    newEdge1->Node = N2;
    newEdge1->Next = N1->firstEdge;
    N1->firstEdge = newEdge1;

    adrEdge newEdge2 = new ElmEdge;
    newEdge2->Node = N1;
    newEdge2->Next = N2->firstEdge;
    N2->firstEdge = newEdge2;
}

void PrintInfoGraph(Graph G) {
    adrNode tempNode = G.first;

    cout << "Graph:" << endl;
    while (tempNode != NULL) {
        cout << tempNode->info << " -> ";
        adrEdge tempEdge = tempNode->firstEdge;
        while (tempEdge != NULL) {
            cout << tempEdge->Node->info;
            if (tempEdge->Next != NULL) {
                cout << ", ";
            }
            tempEdge = tempEdge->Next;
        }
        cout << endl;
        tempNode = tempNode->Next;
    }
}

void ResetVisited(Graph &G) {
    adrNode temp = G.first;
    while (temp != NULL) {
        temp->visited = 0;
        temp = temp->Next;
    }
}

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

    N->visited = 1;
    cout << N->info << " ";

    adrEdge tempEdge = N->firstEdge;
    while (tempEdge != NULL) {
        if (tempEdge->Node->visited == 0) {
            PrintDFS(G, tempEdge->Node);
        }
        tempEdge = tempEdge->Next;
    }
}
```

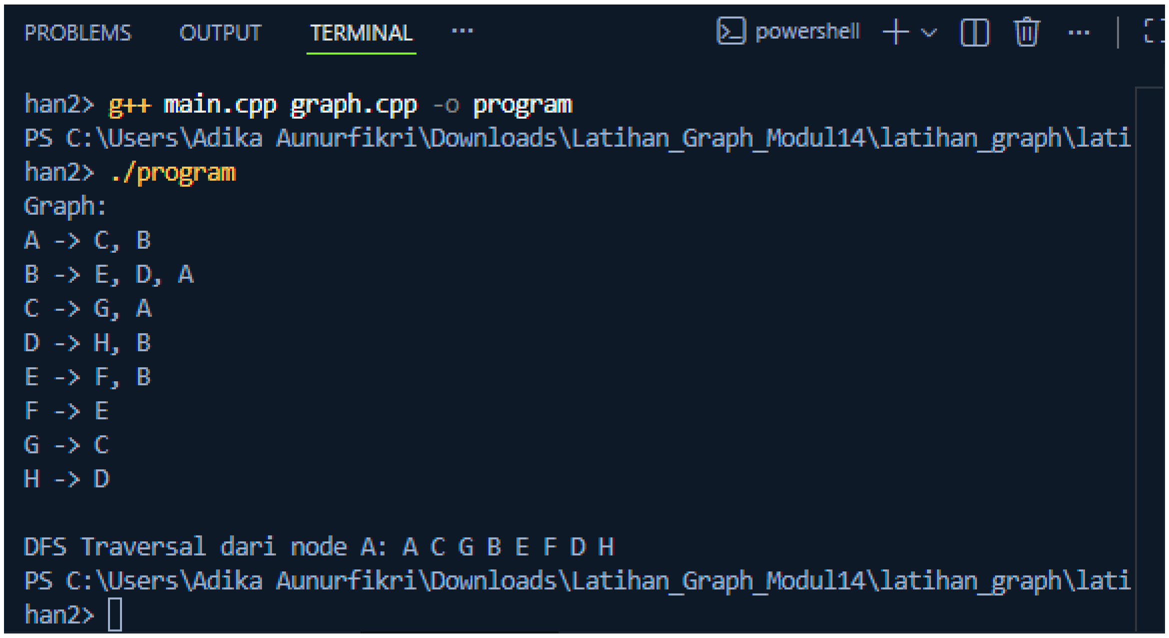
Main.cpp

```
#include <iostream>
#include "graph.h"
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');

adrNode nodeA = FindNode(G, 'A');
adrNode nodeB = FindNode(G, 'B');
adrNode nodeC = FindNode(G, 'C');
adrNode nodeD = FindNode(G, 'D');
adrNode nodeE = FindNode(G, 'E');
adrNode nodeF = FindNode(G, 'F');
adrNode nodeG = FindNode(G, 'G');
adrNode nodeH = FindNode(G, 'H');

ConnectNode(nodeA, nodeB);
ConnectNode(nodeA, nodeC);
ConnectNode(nodeB, nodeD);
ConnectNode(nodeB, nodeE);
ConnectNode(nodeD, nodeH);
ConnectNode(nodeE, nodeF);
ConnectNode(nodeC, nodeG);

PrintInfoGraph(G);

cout << "\nDFS Traversal dari node A: ";
ResetVisited(G);
PrintDFS(G, nodeA);
cout << endl;
```

```
    return 0;
}
```

Screenshots Output:



```
han2> g++ main.cpp graph.cpp -o program
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\lati
han2> ./program
Graph:
A -> C, B
B -> E, D, A
C -> G, A
D -> H, B
E -> F, B
F -> E
G -> C
H -> D

DFS Traversal dari node A: A C G B E F D H
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\lati
han2>
```

Guided 3

graph.h

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct 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);
void InsertNode(Graph &G, infoGraph X);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);
adrNode FindNode(Graph G, infoGraph X);
void ResetVisited(Graph &G);
void PrintBFS(Graph G, adrNode N);

#endif
```

graph..cpp

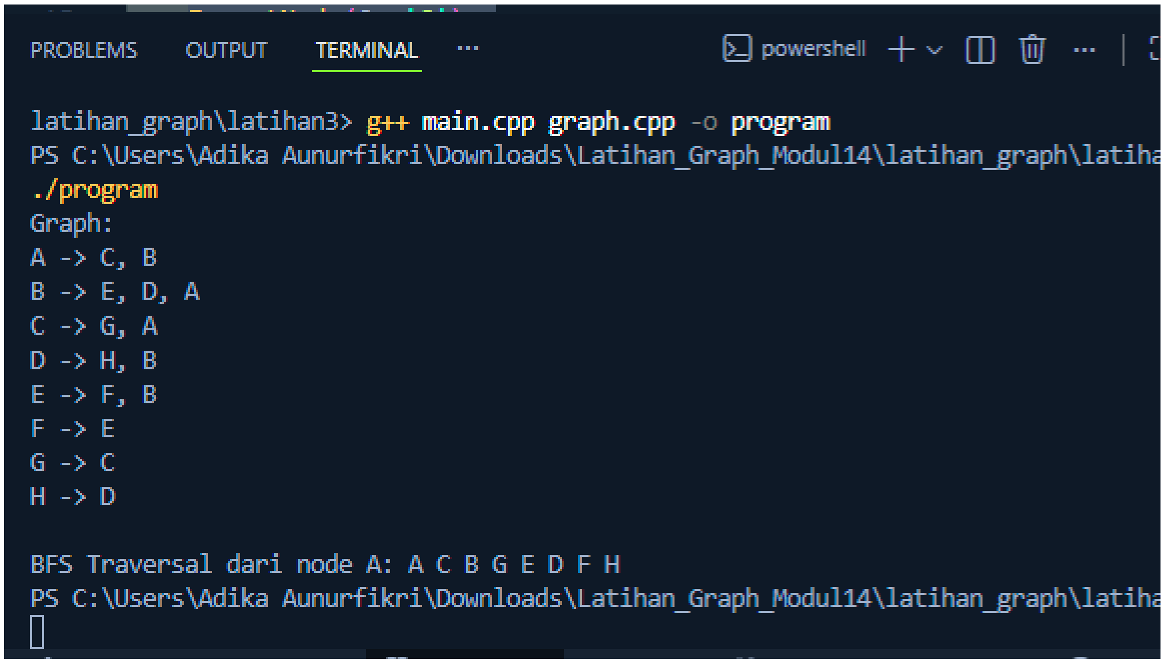
```
#include <iostream>
#include <queue>
#include "graph.h"
using namespace std;

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

void InsertNode(Graph &G, infoGraph X) {
   adrNode newNode = new ElmNode;
   newNode->info = X;
   newNode->visited = 0;
   newNode->firstEdge = NULL;
   newNode->Next = NULL;

   if (G.first == NULL) {
      G.first = newNode;
   } else {
      adrNode temp = G.first;
      while (temp->Next != NULL) {
         temp = temp->Next;
      }
      temp->Next = newNode;
   }
```

```
}

adrNode FindNode(Graph G, infoGraph X) {
    adrNode temp = G.first;
    while (temp != NULL) {
        if (temp->info == X) {
            return temp;
        }
        temp = temp->Next;
    }
    return NULL;
}

void ConnectNode(adrNode N1, adrNode N2) {
    adrEdge newEdge1 = new ElmEdge;
    newEdge1->Node = N2;
    newEdge1->Next = N1->firstEdge;
    N1->firstEdge = newEdge1;

    adrEdge newEdge2 = new ElmEdge;
    newEdge2->Node = N1;
    newEdge2->Next = N2->firstEdge;
    N2->firstEdge = newEdge2;
}

void PrintInfoGraph(Graph G) {
    adrNode tempNode = G.first;

    cout << "Graph:" << endl;
    while (tempNode != NULL) {
        cout << tempNode->info << " -> ";
        adrEdge tempEdge = tempNode->firstEdge;
        while (tempEdge != NULL) {
            cout << tempEdge->Node->info;
            if (tempEdge->Next != NULL) {
                cout << ", ";
            }
            tempEdge = tempEdge->Next;
        }
        cout << endl;
        tempNode = tempNode->Next;
    }
}

void ResetVisited(Graph &G) {
    adrNode temp = G.first;
    while (temp != NULL) {
        temp->visited = 0;
        temp = temp->Next;
    }
}

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

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

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

        adrEdge tempEdge = current->firstEdge;
        while (tempEdge != NULL) {
            if (tempEdge->Node->visited == 0) {
                tempEdge->Node->visited = 1;
                Q.push(tempEdge->Node);
            }
            tempEdge = tempEdge->Next;
        }
    }
}
```

Main.cpp

```
#include <iostream>
#include "graph.h"
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');

    adrNode nodeA = FindNode(G, 'A');
    adrNode nodeB = FindNode(G, 'B');
    adrNode nodeC = FindNode(G, 'C');
    adrNode nodeD = FindNode(G, 'D');
    adrNode nodeE = FindNode(G, 'E');
    adrNode nodeF = FindNode(G, 'F');
    adrNode nodeG = FindNode(G, 'G');
    adrNode nodeH = FindNode(G, 'H');

    ConnectNode(nodeA, nodeB);
    ConnectNode(nodeA, nodeC);
    ConnectNode(nodeB, nodeD);
    ConnectNode(nodeB, nodeE);
    ConnectNode(nodeD, nodeH);
    ConnectNode(nodeE, nodeF);
    ConnectNode(nodeC, nodeG);

    PrintInfoGraph(G);

    cout << "\nBFS Traversal dari node A: ";
    ResetVisited(G);
    PrintBFS(G, nodeA);
    cout << endl;

    return 0;
}
```

Screenshots Output:



PROBLEMS OUTPUT TERMINAL ··· powershell

```
latihan_graph\latihan3> g++ main.cpp graph.cpp -o program
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\latiha
./program
Graph:
A -> C, B
B -> E, D, A
C -> G, A
D -> H, B
E -> F, B
F -> E
G -> C
H -> D

BFS Traversal dari node A: A C B G E D F H
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Graph_Modul14\latihan_graph\latiha
```

Deskripsi:

Program ini merupakan implementasi struktur data Graph tidak berarah (Undirected Graph) menggunakan representasi multilist atau adjacency list dengan bahasa pemrograman C++. Graph dibangun menggunakan dua struktur utama yaitu ElmNode yang merepresentasikan simpul/vertex dan ElmEdge yang merepresentasikan sisi/edge penghubung antar simpul. Dalam implementasinya, program menyediakan tiga latihan utama yang mencakup pembuatan Abstract Data Type (ADT) Graph lengkap dengan operasi dasar seperti CreateGraph, InsertNode, dan ConnectNode untuk membentuk graf tidak berarah dimana setiap koneksi antar node bersifat dua arah. Selanjutnya program juga mengimplementasikan dua algoritma penelusuran graf yang fundamental yaitu Depth First Search (DFS) yang menggunakan pendekatan rekursif untuk menelusuri graf secara mendalam terlebih dahulu sebelum backtracking, dan Breadth First Search (BFS) yang menggunakan struktur data queue untuk menelusuri graf secara melebar per level dari node awal. Setiap latihan dirancang secara modular dalam folder terpisah untuk memudahkan pemahaman, testing, dan dokumentasi dengan output yang jelas menampilkan adjacency list serta urutan kunjungan node sesuai algoritma yang digunakan.

C. Kesimpulan

Implementasi struktur data Graph menggunakan multilist terbukti efektif dan fleksibel dalam merepresentasikan hubungan antar node dengan sifat dinamis yang memungkinkan penambahan dan penghapusan node secara efisien tanpa batasan ukuran seperti pada array. Melalui tiga latihan yang telah dikerjakan, dapat disimpulkan bahwa pemahaman tentang ADT Graph, algoritma DFS, dan algoritma BFS sangat penting dalam menyelesaikan berbagai permasalahan yang berkaitan dengan graf seperti pencarian jalur, deteksi siklus, dan analisis keterhubungan dalam jaringan. Perbedaan mendasar antara DFS dan BFS terletak pada strategi penelusurannya dimana DFS lebih cocok untuk masalah yang memerlukan eksplorasi mendalam seperti pencarian jalur atau deteksi siklus, sedangkan BFS lebih optimal untuk mencari jalur terpendek atau menganalisis hubungan antar node berdasarkan jarak dari node awal. Program yang telah dibuat menunjukkan bahwa implementasi graf menggunakan pointer dan linked list memberikan fleksibilitas tinggi dalam pengelolaan memori dan kemudahan dalam pengembangan algoritma-algoritma graf lanjutan seperti Dijkstra, Kruskal, atau Prim untuk berbagai aplikasi nyata dalam bidang jaringan komputer, sistem transportasi, dan social network analysis.

D. Referensi

- W3Resource. (2020). *C++ String Exercises: Convert digit/number to words*.
- GeeksforGeeks. (2020). *Loops in C++ (for, while, do-while).*