LAPORAN PRAKTIKUM

- **Identitas Praktikum**

  Nama MK : Struktur Data
  Kode MK : CCK2AAB4
  Bobot SKS : 4 SKS
  Tempat : L-Program, Gedung DC, lantai 3
  Hari, tanggal : Selasa, 24 Desember 2024
  Jam : 12:30-15:30 WIB
  Topik praktikum : Modul-14 GRAPH

- **Identitas Mahasiswa**

  Nama lengkap : Afad Fath Musyarof Halim
  NIM : 2211104030
  Program Studi : S-1 Software Engineering

- **Hasil Praktikum**

13. GRAPH

13.1. Pengertian

Graph adalah sebuah kumpulan yang terdiri dari node (vertec) dan garis penghubung (edge).

13.2. Representasi Graph

```
#include <iostream>
#include <vector>
using namespace std;

// Kelas untuk merepresentasikan graf
class Graph {
private:
    int vertices;                   // Jumlah simpul (vertices)
    vector<vector<int>> adjList;    // Adjacency list

public:
    // Konstruktor
    Graph(int v) {
        vertices = v;
        adjList.resize(v);
```

```
        }

        // Menambahkan edge ke graf (graf tak berarah)
        void addEdge(int u, int v) {
            adjList[u].push_back(v);
            adjList[v].push_back(u); // Hapus ini jika graf berarah
        }

        // Menampilkan graf
        void displayGraph() {
            for (int i = 0; i < vertices; ++i) {
                cout << "Vertex " << i << ": ";
                for (int neighbor : adjList[i]) {
                    cout << neighbor << " ";
                }
                cout << endl;
            }
        }
    };

    int main() {
        // Membuat graf dengan 5 simpul
        Graph g(5);

        // Menambahkan sisi
        g.addEdge(0, 1);
        g.addEdge(0, 4);
        g.addEdge(1, 2);
        g.addEdge(1, 3);
        g.addEdge(1, 4);
        g.addEdge(2, 3);
        g.addEdge(3, 4);

        // Menampilkan adjacency list
        cout << "Adjacency List Representation of the Graph:" << endl;
        g.displayGraph();
        return 0;
}
```

13.3. Latihan

- Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

struct ElmNode;
struct ElmEdge;

typedef ElmNode* adrNode;
typedef ElmEdge* adrEdge;

struct Graph {
   adrNode first;
};

struct ElmNode {
   char info;
   adrNode next;
   adrEdge firstEdge;
};

struct ElmEdge {
   adrNode node;
   adrEdge next;
};

void CreateGraph(Graph &G);
void InsertNode(Graph &G, char info);
adrNode FindNode(Graph G, char info);
void ConnectNode(adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

void PrintBFS(Graph &G, adrNode start);
void PrintDFS(Graph &G, adrNode start);


#endif
```

- Graph.cpp

```
#include "graph.h"
#include <iostream>
#include <queue>
#include <stack>
#include <unordered_set>

using namespace std;

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

void InsertNode(Graph &G, char info) {
    adrNode newNode = new ElmNode;
    newNode->info = info;
    newNode->next = G.first;
    newNode->firstEdge = nullptr;
    G.first = newNode;
}

adrNode FindNode(Graph G, char info) {
    adrNode currentNode = G.first;
    while (currentNode != nullptr) {
        if (currentNode->info == info) {
            return currentNode;
        }
        currentNode = currentNode->next;
    }
    return nullptr;
}

adrEdge AllocateEdge(adrNode N) {
    adrEdge newEdge = new ElmEdge;
    newEdge->node = N;
    newEdge->next = nullptr;
    return newEdge;
}
```

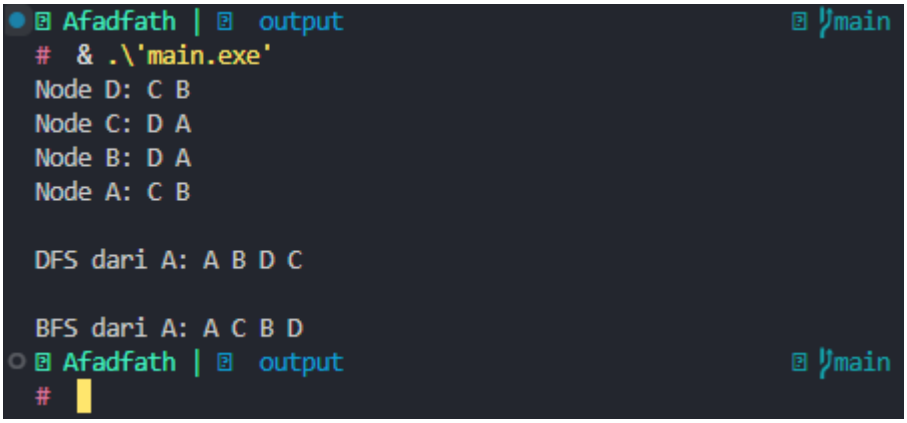
```
void ConnectNode(adrNode N1, adrNode N2) {
    adrEdge newEdge1 = AllocateEdge(N2);
    newEdge1->next = N1->firstEdge;
    N1->firstEdge = newEdge1;

    adrEdge newEdge2 = AllocateEdge(N1);
    newEdge2->next = N2->firstEdge;
    N2->firstEdge = newEdge2;
}

void PrintInfoGraph(Graph G) {
    adrNode currentNode = G.first;
    while (currentNode != nullptr) {
        cout << "Node " << currentNode->info << ": ";
        adrEdge currentEdge = currentNode->firstEdge;
        while (currentEdge != nullptr) {
            cout << currentEdge->node->info << " ";
            currentEdge = currentEdge->next;
        }
        cout << endl;
        currentNode = currentNode->next;
    }
}

void PrintBFS(Graph &G, adrNode start) {
    if (start == nullptr) return;

    queue<adrNode> q;
    unordered_set<adrNode> visited;

    q.push(start);
    visited.insert(start);

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

        adrEdge currentEdge = currentNode->firstEdge;
```

```
            while (currentEdge != nullptr) {
                if (visited.find(currentEdge->node) == visited.end()) {
                    q.push(currentEdge->node);
                    visited.insert(currentEdge->node);
                }
                currentEdge = currentEdge->next;
            }
        }
        cout << endl;
    }

    void PrintDFS(Graph &G, adrNode start) {
        if (start == nullptr) return;

        stack<adrNode> s;
        unordered_set<adrNode> visited;

        s.push(start);

        while (!s.empty()) {
            adrNode currentNode = s.top();
            s.pop();

            if (visited.find(currentNode) == visited.end()) {
                cout << currentNode->info << " ";
                visited.insert(currentNode);

                adrEdge currentEdge = currentNode->firstEdge;
                while (currentEdge != nullptr) {
                    if (visited.find(currentEdge->node) == visited.end()) {
                        s.push(currentEdge->node);
                    }
                    currentEdge = currentEdge->next;
                }
            }
        }
        cout << endl;
}
```

- Main.cpp

```
#include "graph.cpp"

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

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

    adrNode A = FindNode(G, 'A');
    adrNode B = FindNode(G, 'B');
    adrNode C = FindNode(G, 'C');
    adrNode D = FindNode(G, 'D');

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(C, D);

    PrintInfoGraph(G);

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

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

    return 0;
}
```

- Output


Afadfath | output main
& .\'main.exe'
Node D: C B
Node C: D A
Node B: D A
Node A: C B
DFS dari A: A B D C
BFS dari A: A C B D
Afadfath | output main
#