LAPORAN PRAKTIKUM

MODUL 14

GRAPH

Disusun Oleh :

Farhan Kurniawan (2311104073)

Kelas:

SE-07-2

Dosen :

Wahyu Andi Saputra, S.Pd, M.Eng,

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

1. Memahami konsep *graph.*
2. Mengimplementasikan *graph* dengan menggunakan *pointer*.

II. LANDASAN TEORI

Graph adalah struktur data non-linear yang terdiri dari himpunan simpul (node atau vertex) dan garis penghubung (edge) di antara simpul-simpul tersebut. Graph dapat dibedakan menjadi graph berarah (directed graph), di mana setiap edge memiliki arah, dan graph tak berarah (undirected graph), di mana edge tidak memiliki arah. Selain itu, terdapat graph berbobot (weighted graph) yang setiap edge-nya memiliki bobot, serta graph tidak berbobot (unweighted graph) yang hanya merepresentasikan koneksi antar simpul. Dalam representasi komputer, graph dapat diimplementasikan menggunakan adjacency matrix atau adjacency list. Graph banyak digunakan dalam berbagai bidang, seperti jaringan komputer untuk merepresentasikan hubungan antar perangkat, sistem transportasi untuk memetakan rute, media sosial untuk hubungan pertemanan, serta algoritma pencarian jalur terpendek seperti Dijkstra atau BFS untuk menemukan rute optimal. Contoh sederhana graph adalah hubungan antara Tempat Kost dan Common Lab, di mana keduanya merupakan simpul (node) yang dihubungkan oleh garis penghubung (edge) berupa jalan di antara keduanya.

III. GUIDED

1. Input Program:

```cpp
#include <iostream>
#include <queue>

using namespace std;

struct ElmNode;

struct ElmEdge {
    ElmNode *Node;
    ElmEdge *Next;
};

struct ElmNode {
    char info;
    bool visited;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
};

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

void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    newNode->visited = false;
    newNode->firstEdge = NULL;
    newNode->Next = NULL;

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

void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->firstEdge;
    N1->firstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " ";
        temp = temp->Next;
    }
    cout << endl;
}

void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false;
        temp = temp->Next;
    }
}
```

```cpp
void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return;
    }
    N->visited = true;
    cout << N->info << " ";
    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node);
        }
        edge = edge->Next;
    }
}

void PrintBFS(Graph G, ElmNode *N) {
    queue<ElmNode*> q;
    q.push(N);
    N->visited = true;

    while (!q.empty()) {
        ElmNode *current = q.front();
        q.pop();
        cout << current->info << " ";

        ElmEdge *edge = current->firstEdge;
        while (edge != NULL) {
            if (!edge->Node->visited) {
                edge->Node->visited = true;
                q.push(edge->Node);
            }
            edge = edge->Next;
        }
    }
}

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');

    ElmNode *A = G.first;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);

    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

2. Output Program:

```
PS D:\Praktukum\C++\Modul14> cd "d:\Praktukum\C++\Modul14\" ; 
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS D:\Praktukum\C++\Modul14>
```

IV. UNGUIDED

1. Input Program:

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

using namespace std;

int main() {
    int jumlahSimpul;

    // Meminta jumlah simpul dari pengguna
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> simpul(jumlahSimpul);
    vector<vector<int>> bobot(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    // Meminta nama simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Silakan masukkan nama simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    // Meminta bobot antar simpul
    cout << "Silakan masukkan bobot antar simpul:\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            if (i == j) {
                bobot[i][j] = 0; // Bobot dari simpul ke dirinya sendiri adalah 0
            } else {
                cout << simpul[i] << " --> " << simpul[j] << ": ";
                cin >> bobot[i][j];
            }
        }
    }

    // Menampilkan matriks bobot
    cout << "\nMatriks Bobot:\n";
    cout << setw(10) << " ";
    for (const auto& namaSimpul : simpul) {
        cout << setw(10) << namaSimpul;
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Output Program:

```
PS D:\Praktukum\C++\Modul14> cd "d:\Praktukum\C++\Modul14\Unguided\" ;
Silakan masukkan jumlah simpul: 2
Silakan masukkan nama simpul 1: BALI
Silakan masukkan nama simpul 2: PALU
Silakan masukkan bobot antar simpul:
BALI --> PALU: 3
PALU --> BALI: 4
Matriks Bobot:
                BALI       PALU
       BALI        0          3
       PALU        4          0
```

2. Input Program:

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

int main() {
    int jumlahSimpul, jumlahSisi;

    // Meminta input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

    // Inisialisasi adjacency matrix dengan nilai 0
    vector<vector<int>> adjacencyMatrix(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    // Meminta pasangan simpul yang terhubung
    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < jumlahSisi; i++) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;

        // Karena graf tak berarah, set adjacencyMatrix[simpul1][simpul2] dan adjacencyMatrix[simpul2][simpul1] ke 1
        adjacencyMatrix[simpul1 - 1][simpul2 - 1] = 1;
        adjacencyMatrix[simpul2 - 1][simpul1 - 1] = 1;
    }

    // Menampilkan adjacency matrix
    cout << "\nAdjacency Matrix:\n";
    for (const auto& row : adjacencyMatrix) {
        for (const auto& val : row) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output Program:

```
PS D:\Praktukum\C++\Modul14> cd "d:\Praktukum\C++\Modul14\Unguided\" ;
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul:
1 2
1 3
2 4
3 4
```

```
Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
```

V. KESIMPULAN

Graph adalah struktur data non-linear yang sangat berguna untuk merepresentasikan hubungan atau koneksi antar elemen, baik dalam bentuk berarah maupun tak berarah, berbobot maupun tidak berbobot. Dengan representasi seperti adjacency matrix atau adjacency list, graph dapat diterapkan di berbagai bidang, seperti jaringan komputer, sistem transportasi, media sosial, dan algoritma pencarian jalur terpendek. Pemahaman tentang graph memberikan fondasi yang kuat untuk mengatasi masalah kompleks yang melibatkan koneksi antar elemen, seperti memetakan hubungan tempat atau mencari rute optimal dalam kehidupan sehari-hari.