# LAPORAN PRAKTIKUM
# Modul 14
# “Graph”


**Disusun Oleh:**
**Dimastian Aji Wibowo (2311104058)**
**SE-07-02**

**Dosen :**
**Wahyu Andi Saputra, S.Pd., M.Eng.**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**

**2024**

1. **Tujuan**
    - Memahami konsep graph
    - Mengimplementasikan graph dengan menggunakan pointer

2. **Landasan Teori**

**A. Pengertian**

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. Jenis – jenis graph**

**1. Graph Berarah**

Merupakan graph dimana tiap node memiliki edge yang memiliki arah, kemana node tersebut dihubungkan.

a. Representasi Graph

Pada dasarnya representasi dari graph berarah sama dengan graph tak-berarah. Perbedaannya apabila graph tak-berarah terdapat node A dan node B yang terhubung, secara otomatis terbentuk panah bolak balik dari A ke B dan B ke A. Pada Graph berarah node A terhubung dengan node B, belum tentu node B terhubung dengan node A.

b. Topological Sort

Diberikan urutan partial dari elemen suatu himpunan, dikehendaki agar elemen yang terurut parsial tersebut mempunyai keterurutan linier. Contoh dari keterurutan parsial banyak dijumpai dalam kehidupan sehari-hari, misalnya:

- Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisite mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalamkurikulum.
- Dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dulu dari pekerjaan lain (misalnya membuat fondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela, dsb.
- Dalam sebuah program Pascal, pemanggilan prosedur harus

sedemikian rupa, sehingga peletakan prosedur pada teks program harus seuai dengan urutan (partial) pemanggilan. Dalam pembuatan tabel pada basis data, tabel yang di-refer oleh tabel lain harus dideklarasikan terlebih dulu. Jika suatu aplikasi terdiri dari banyak tabel, maka urutan pembuatan tabel harus sesuai dengan definisinya.

2. **Graph Tidak Berarah**

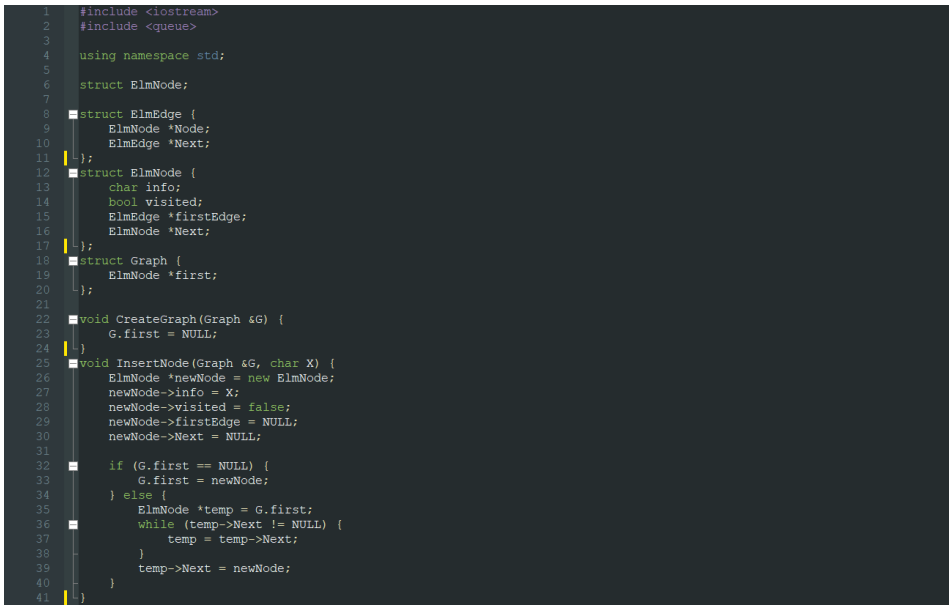
   Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah. Selain arah, beban atau nilai sering ditambahkan pada edge . Misalnya nilai yang merepresentasikan panjang, atau biaya transportasi, dan lain-lain. Hal mendasar lain yang perlu diketahui adalah, suatu node A dikatakan bertetangga dengan node B jika antara node A dan node B dihubungkan langsung dengan sebuah edge.

   Dari definisi graph dapat kita simpulkan bahwa graph dapat direpresentasikan dengan Matrik Ketetanggaan (Adjacency Matrices), yaitu matrik yang menyatakan keterhubungan antar node dalam graph. Implementasi matrik ketetanggaan dalam bahasa pemrograman dapat berupa : Array 2 Dimensi dan Multi Linked List. Graph dapat direpresentasikan dengan matrik n x n, dimana n merupakan jumlah node dalam graph tersebut.

## 3. Guided

1. Membuat struct ElmNode digunakan untuk merepresentasikan setiap simpul (node) dalam graf dengan char info untuk menyimpan informasi atau label pada simpul, bool visited untuk menandai apakah simpul telah dikunjungi dalam traversal, ElmEdge* firstEdge sebagai pointer ke daftar tepi, dan ElmNode* Next sebagai pointer menuju simpul berikutnya.
2. Membuat struct ElmEdge digunakan untuk merepresentasikan setiap tepi (edge) dalam graf dengan ElmNode* Node sebagai pointer menuju simpul dan ElmEdge* Next sebagai pointer ke tepi berikutnya.
3. Membuat struct Graph digunakan untuk merepresentasikan graf dengan ElmNode* first untuk menyimpan pointer ke simpul pertama dalam daftar simpul graf.
4. Fungsi CreateGraph bertujuan menginisialisasi graf dengan mengatur pointer first pada graf ke NULL, menandakan graf kosong.

5. Fungsi InsertNode menambahkan simpul baru ke dalam graf dengan membuat simpul baru dan menginisialisasi atributnya, jika graf kosong maka simpul baru menjadi simpul pertama, dan jika tidak maka simpul baru ditambahkan diakhir daftar simpul.



```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;
    }
}
```

6. Fungsi ConnectNode membuat koneksi antara dua simpul dengan membuat tepi baru dan menambahkan tepi kee daftar tepi simpul.
7. Fungsi PrintInfoGraph mencetak informasi semua simpul dalam graf secara berurutan berdasarkan daftar simpul.
8. Fungsi ResetVisited untuk mengatur ulang status visited semua simpul dalam graf menjadi false.
9. Fungsi PrintDFS untuk melakukan traversal Depth-First Search (DFS) pada graf dengan menandai simpul saat ini sebagai telah dikunjungi, mencetak informasi simpul tersebut, dan mengunjungi semua simpul yang terhubung melalui tepi.

```
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;
    }
}
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;
    }
}
```

10. Fungsi PrintBFS melakukan traversal Breadth-First Search (BFS) pada graf dengan menggunakan queue untuk memproses impul, menandai dan mencetak simpul yang diproses, lalu menambahkan simpul-simpul tetangga yang belum dikunjungi ke antrean.

```
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;
        }
    }
}
```

11. Main memanggil fungsi fungsi sebelumnya.

```cpp
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;
}
```

## 4. Unguided

### A. Unguided 1

1. Membuat variabel jumlahSimpul yang digunakan untuk menyimpan jumlah simpul dalam graf. Nilai ini akan dimasukkan oleh pengguna melalui fungsi cin.
2. Membuat vector<string> namaSimpul untuk menyimpan nama-nama simpul yang dimasukkan oleh pengguna. Vektor ini berukuran sesuai dengan jumlah simpul yang telah ditentukan.
3. Menggunakan perulangan for untuk meminta pengguna memasukkan nama setiap simpul, lalu menyimpannya ke dalam vector namaSimpul berdasarkan indeks simpul.
4. Membuat vector<vector<int>> adjacencyMatrix yang merupakan matriks 2D untuk menyimpan bobot antar simpul. Matriks ini diinisialisasi dengan nilai 0 menggunakan konstruktor vector.
5. Membuat nested loop for untuk meminta pengguna memasukkan bobot antar simpul. Jika simpul yang diproses adalah simpul yang sama (misalnya A ke A), maka bobot secara otomatis diatur ke 0, karena simpul tidak memiliki hubungan ke dirinya sendiri. Jika berbeda, bobot diminta dari pengguna dan disimpan dalam matriks.
6. Menampilkan judul "Adjacency Matrix" sebagai header, diikuti dengan menampilkan nama-nama simpul di baris pertama

menggunakan setw(10) untuk merapikan tampilan tabel.

7. Menggunakan nested loop untuk menampilkan isi matriks ketetanggaan. Baris pertama menunjukkan nama simpul, sementara kolom diisi dengan nilai bobot dari adjacencyMatrix. Setiap elemen ditampilkan menggunakan setw(10) agar tata letak tabel rapi.
8. Membuat fungsi main() sebagai entry point program. Fungsi ini mengatur alur kerja mulai dari input jumlah simpul, input nama simpul, input bobot antar simpul, hingga menampilkan matriks ketetanggaan sebagai output.

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

int main() {
    int jumlahSimpul;

    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> namaSimpul(jumlahSimpul);
    cout << "Silakan masukkan nama simpul:\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> namaSimpul[i];
    }
    vector<vector<int>> adjacencyMatrix(jumlahSimpul, vector<int>(jumlahSimpul, 0));
    cout << "Silakan masukkan bobot antar simpul (gunakan 0 jika tidak ada hubungan):\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            if (i == j) {
                adjacencyMatrix[i][j] = 0;
            } else {
                cout << namaSimpul[i] << " --> " << namaSimpul[j] << " = ";
                cin >> adjacencyMatrix[i][j];
            }
        }
    }
    cout << "\nAdjacency Matrix:\n";
    cout << setw(10) << " ";
    for (const auto &nama : namaSimpul) {
        cout << setw(10) << nama;
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(10) << adjacencyMatrix[i][j];
        }
        cout << endl;
    }
    return 0;
}
```

9. Berikut merupakan output dari program tersebut.

```
Masukkan jumlah simpul: 2
Silakan masukkan nama simpul:
Simpul 1: Bali
Simpul 2: Palu
Silakan masukkan bobot antar simpul (gunakan 0 jika tidak ada hubungan):
Bali --> Palu = 2
Palu --> Bali = 2

Adjacency Matrix:
                Bali      Palu
      Bali         0         2
      Palu         2         0
```

## B. Unguided 2

1. Membuat dua variabel jumlahSimpul dan jumlahSisi yang digunakan untuk menyimpan jumlah simpul dalam graf dan jumlah sisi (hubungan antar simpul) dengan pengguna diminta untuk memasukan nilai – nilai tersebut.

2. Membuat vector<vector<int>> adjacencyMatrix dengan ukuran jumlahSimpul x jumlahSimpul untuk menyimpan hubungan antar simpul dalam bentuk matriks ketetanggaan. Matriks diinisialisasi dengan nilai 0 menggunakan konstruktor vektor bawaan.
3. Menggunakan perulangan for sebanyak jumlahSisi untuk membaca pasangan simpul yang memiliki hubungan. Dengan pada setiap iterasi pengguna pengguna diminta memasukkan dua simpul, simpul1 dan simpul2, yang memiliki hubungan. Indeks simpul dikurangi 1 (simpul1-- dan simpul2--) karena matriks menggunakan indeks berbasis nol, sementara pengguna cenderung memasukkan indeks berbasis satu.
4. Menambahkan hubungan antar simpul dalam matriks ketetanggaan dengan Elemen adjacencyMatrix[simpul1][simpul2] dan adjacencyMatrix[simpul2][simpul1] diatur menjadi 1 untuk menunjukkan bahwa kedua simpul saling terhubung. Dan Penugasan dua arah dilakukan karena graf ini adalah graf tidak berarah (undirected graph).
5. Menggunakan perulangan untuk menampilkan isi matriks ketetanggaan dengan setiap baris matriks dicetak dalam bentuk deretan angka yang dipisahkan oleh spasi, mewakili hubungan antar simpul.
6. Membuat fungsi main() sebagai entry point program. Fungsi ini mengatur seluruh alur eksekusi mulai dari input jumlah simpul dan sisi, membaca pasangan simpul, hingga menampilkan matriks ketetanggaan.

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

int main() {
    int jumlahSimpul, jumlahSisi;

    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

    vector<vector<int>> adjacencyMatrix(jumlahSimpul, vector<int>(jumlahSimpul, 0));

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

        simpul1--;
        simpul2--;

        adjacencyMatrix[simpul1][simpul2] = 1;
        adjacencyMatrix[simpul2][simpul1] = 1;
    }
    cout << "Adjacency Matrix:\n";
    for (const auto &row : adjacencyMatrix) {
        for (const auto &value : row) {
            cout << value << " ";
        }
        cout << endl;
    }
    return 0;
}
```

7. Berikut merupakan output dari program tersebut

```
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
```

## 5. Kesimpulan

Graf direpresentasikan menggunakan simpul (node) dan tepi (edge), dengan memanfaatkan pointer untuk menghubungkan simpul dan membentuk struktur graf berarah. Melalui praktikum ini, kami berhasil memahami konsep dasar graf, seperti penambahan simpul dan tepi, serta traversal menggunakan metode Depth-First Search (DFS) dan Breadth-First Search (BFS). Implementasi ini menunjukkan bagaimana struktur graf dapat digunakan untuk merepresentasikan hubungan antar elemen secara efisien. Selain itu, kami juga menyadari pentingnya pengelolaan status kunjungan simpul dalam proses traversal untuk menghindari perulangan atau kesalahan logika. Praktikum ini memberikan dasar yang kuat untuk memahami konsep lebih lanjut dalam teori graf dan aplikasinya pada permasalahan dunia nyata.