

**LAPORAN PRAKTIKUM**  
**MODUL 14**  
**GRAPH**



**DISUSUN OLEH :**  
**TIURMA GRACE ANGELINA – 2311104042**  
**SE0702**

**DOSEN :**  
**WAHYU ANDI SAPUTRA**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

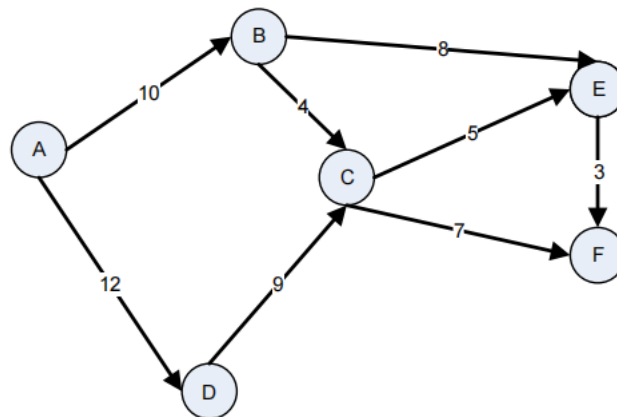
## 1. TUJUAN

- Memahami konsep Graph.
- Mengimplementasikan Graph dengan menggunakan pointer.

## 2. Landasan Teori

Graph adalah salah satu struktur data yang digunakan untuk merepresentasikan hubungan antar elemen berupa node (simpul) yang dihubungkan oleh edge (sisi). Graph terbagi menjadi dua jenis utama, yaitu graph berarah, di mana setiap edge memiliki arah tertentu yang menunjukkan hubungan dari satu node ke node lainnya, dan graph tidak berarah, di mana edge tidak memiliki arah sehingga hubungan antar node bersifat dua arah. Dalam praktikum ini, graph direpresentasikan menggunakan multilist karena sifatnya yang dinamis, sehingga memungkinkan penanganan data yang fleksibel dan efisien. Operasi dasar pada graph meliputi penambahan node dan edge, penghapusan elemen, serta penelusuran node menggunakan algoritma seperti Breadth First Search (BFS) dan Depth First Search (DFS). Algoritma BFS bekerja dengan menelusuri graph secara level per level menggunakan struktur data queue, sementara algoritma DFS menelusuri graph hingga kedalaman tertentu sebelum kembali menggunakan stack atau rekursi. Selain itu, topological sorting juga dipelajari dalam praktikum ini. Topological sorting merupakan proses pengurutan node pada graph berarah sesuai dengan dependensi antar elemen. Teknik ini sering digunakan dalam berbagai kasus, seperti menentukan urutan pekerjaan dalam proyek atau pengelolaan dependensi tabel dalam basis data. Dengan memahami konsep dasar graph, cara merepresentasikannya, serta implementasi algoritma-algoritma tersebut, praktikum ini memberikan pemahaman yang mendalam tentang bagaimana struktur data graph dapat digunakan untuk memecahkan berbagai permasalahan dalam dunia nyata, terutama yang melibatkan hubungan antar elemen secara kompleks.

Contoh Graph:



Gambar 14-11 *Graph Jarak Antar kota*

### 3. Guided

#### 1. Guided 1

Code :

```
#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;
        }
    }

    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;
}

```

### Output :

```

"C:\Users\USER\Downloads\2311104042_Tiurma Grace Angelina\GUIDED\bin\Debug\GUIDED.exe"
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H

Process returned 0 (0x0)   execution time : 0.098 s
Press any key to continue.

```

### Penjelasan :

- ElmEdge: Struktur untuk menyimpan sisi/edge graf, berisi pointer ke node tujuan dan pointer ke edge berikutnya
- ElmNode: Struktur untuk menyimpan node/simpul graf, berisi:
  - info: nilai/informasi node (berupa karakter)
  - visited: penanda apakah node sudah dikunjungi
  - firstEdge: pointer ke edge pertama
  - Next: pointer ke node berikutnya
- Graph: Struktur utama graf yang menyimpan pointer ke node pertama
- Fungsi-fungsi Dasar:
  - CreateGraph: Menginisialisasi graf kosong
  - InsertNode: Menambahkan node baru ke graf di akhir list
  - ConnectNode: Membuat edge/sisi yang menghubungkan dua node
- Fungsi Traversal:
  - PrintDFS: Melakukan traversal graf secara Depth-First Search (DFS)
    - Menggunakan rekursi
    - Mengunjungi node saat ini, tandai sebagai visited
    - Rekursif ke semua node tetangga yang belum dikunjungi
  - PrintBFS: Melakukan traversal graf secara Breadth-First Search (BFS)
    - Menggunakan queue
    - Mengunjungi node level per level
    - Menambahkan semua tetangga yang belum dikunjungi ke queue

- Fungsi Utilitas:
  - PrintInfoGraph: Mencetak semua info node dalam graf
  - ResetVisited: Mereset status visited semua node menjadi false
- Main Program:
  - Membuat graf dengan 8 node (A-H)
  - Membuat koneksi sesuai struktur graf yang diinginkan
  - Menampilkan hasil traversal DFS dan BFS
- Graf yang dibuat memiliki struktur:
  - A terhubung ke B dan C
  - B terhubung ke D dan E
  - C terhubung ke F dan G
  - D terhubung ke H

Output program akan menampilkan urutan node yang dikunjungi saat traversal DFS dan BFS.

#### 4. Unguided

##### 1. Unguided 1

Code :

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

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

struct ElmNode {
    string info;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
    int numNodes;
};

void CreateGraph(Graph &G) {
    G.first = NULL;
    G.numNodes = 0;
}

void InsertNode(Graph &G, string X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    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;
    }
    G.numNodes++;
}
```



```

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

```

```

ElmNode* FindNode(Graph G, string name) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        if (temp->info == name) return temp;
        temp = temp->Next;
    }
    return NULL;
}

```

```

void PrintMatrix(Graph G) {

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

    // Print matrix
    ElmNode *row = G.first;
    while (row != NULL) {
        cout << row->info << "    ";
        ElmNode *col = G.first;
        while (col != NULL) {
            bool found = false;
            int weight = 0;
            ElmEdge *edge = row->firstEdge;
            while (edge != NULL) {
                if (edge->Node == col) {
                    found = true;
                    weight = edge->weight;
                    break;
                }
                edge = edge->Next;
            }
            cout << weight << "    ";

```

```

        col = col->Next;
    }
    cout << endl;
    row = row->Next;
}
}

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

    int numNodes;
    cout << "Silakan masukan jumlah simpul : ";
    cin >> numNodes;

    cout << "Silakan masukan nama simpul\n";
    for (int i = 0; i < numNodes; i++) {
        string nodeName;
        cout << "Simpul " << i + 1 << " : ";
        cin >> nodeName;
        InsertNode(G, nodeName);
    }

    cout << "Silakan masukkan bobot antar simpul\n";
    ElmNode *node1 = G.first;
    while (node1 != NULL) {
        ElmNode *node2 = G.first;
        while (node2 != NULL) {
            int weight;
            cout << node1->info << "-->" << node2->info << " = ";
            cin >> weight;
            if (weight != 0) {
                ConnectNode(node1, node2, weight);
            }
            node2 = node2->Next;
        }
        node1 = node1->Next;
    }

    cout << endl;
    PrintMatrix(G);

    return 0;
}

```

**Output :**

```
"C:\Users\USER\Downloads\2311104042_Tiurma Grace Angelina\UNGUIDED\bin\Debug\UNGUIDED.exe"
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukan bobot antar simpul
BALI-->BALI = 0
BALI-->PALU = 3
PALU-->BALI = 4
PALU-->PALU = 0

      BALI    PALU
BALI   0      3
PALU   4      0

Process returned 0 (0x0)   execution time : 15.136 s
Press any key to continue.
```

### Penjelasan :

Program ini membuat graf yang menyimpan data menggunakan daftar (list) untuk menunjukkan hubungan antar kota. Meskipun data disimpan dalam bentuk daftar, program akan menampilkan hasilnya dalam bentuk tabel/matriks agar lebih mudah dibaca.

- **Struct ElmNode:** Merepresentasikan simpul (node) dalam graph.
  - info: Menyimpan nama simpul (string).
  - firstEdge: Pointer ke edge pertama yang terhubung ke simpul ini.
  - Next: Pointer ke simpul berikutnya dalam linked list.
- **Struct ElmEdge:** Merepresentasikan edge dalam graph.
  - Node: Pointer ke simpul yang dihubungkan oleh edge ini.
  - weight: Menyimpan bobot (jarak atau biaya) dari edge ini.
  - Next: Pointer ke edge berikutnya dari simpul asal.
- **Struct Graph:** Menyimpan graph itu sendiri.
  - first: Pointer ke simpul pertama dalam graph.
  - numNodes: Menyimpan jumlah simpul dalam graph.

### Fungsi-Fungsi Utama

#### a. CreateGraph(Graph &G)

Inisialisasi graph dengan membuat first menjadi NULL dan numNodes menjadi 0.

#### b. InsertNode(Graph &G, string X)

Menambahkan simpul baru ke graph:

1. Membuat simpul baru dengan informasi X.
2. Jika graph kosong, simpul baru menjadi first.
3. Jika graph sudah memiliki simpul, simpul baru ditambahkan di akhir linked list.
4. numNodes diperbarui untuk menghitung jumlah simpul.

#### c. ConnectNode(ElmNode \*N1, ElmNode \*N2, int weight)

Menghubungkan dua simpul:

1. Membuat edge baru yang menghubungkan N1 ke N2 dengan bobot weight.
2. Menambahkan edge ini ke daftar edge simpul N1.

#### d. FindNode(Graph G, string name)

Mencari simpul berdasarkan nama (info):

1. Menelusuri linked list simpul di dalam graph.
2. Jika ditemukan simpul dengan info sesuai, mengembalikan pointer simpul tersebut.
3. Jika tidak ditemukan, mengembalikan NULL.

#### **e. PrintMatrix(Graph G)**

Mencetak matriks adjacency dari graph:

1. Menampilkan header tabel (kolom simpul).
2. Untuk setiap simpul sebagai baris:
  - Mencetak bobot dari simpul tersebut ke semua simpul lainnya.
  - Jika ada edge antara dua simpul, mencetak bobot edge tersebut.
  - Jika tidak ada edge, mencetak 0.

### **main() Function**

#### **a. Input Jumlah dan Nama Simpul**

1. Meminta pengguna memasukkan jumlah simpul (numNodes).
2. Meminta pengguna memasukkan nama untuk setiap simpul (InsertNode).

#### **b. Input Bobot Antar Simpul**

1. Untuk setiap pasangan simpul (N1, N2), meminta pengguna memasukkan bobot.
2. Jika bobot tidak 0, fungsi ConnectNode dipanggil untuk menghubungkan kedua simpul.

#### **c. Cetak Matriks Adjacency**

Fungsi PrintMatrix dipanggil untuk mencetak matriks adjacency dengan format yang telah ditentukan.

## **2. Unguided 2**

**Code :**

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

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

struct ElmNode {
    int info;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
```

```

    int numNodes;
};

void CreateGraph(Graph &G) {
    G.first = NULL;
    G.numNodes = 0;
}

void InsertNode(Graph &G, int X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    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;
    }
    G.numNodes++;
}

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

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

ElmNode* GetNode(Graph G, int index) {
    ElmNode *temp = G.first;
    for(int i = 1; i < index; i++) {
        if(temp == NULL) return NULL;
        temp = temp->Next;
    }
    return temp;
}

```

```

void PrintMatrix(Graph G) {
    cout << "Adjacency Matrix:" << endl;
    for(int i = 1; i <= G.numNodes; i++) {
        ElmNode* node1 = GetNode(G, i);
        for(int j = 1; j <= G.numNodes; j++) {
            bool connected = false;
            ElmNode* node2 = GetNode(G, j);
            ElmEdge* edge = node1->firstEdge;
            while(edge != NULL) {
                if(edge->Node == node2) {
                    connected = true;
                    break;
                }
                edge = edge->Next;
            }
            cout << (connected ? "1" : "0") << " ";
        }
        cout << endl;
    }
}

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

    int vertices, edges;
    cout << "Masukkan jumlah simpul: ";
    cin >> vertices;

    for(int i = 1; i <= vertices; i++) {
        InsertNode(G, i);
    }

    cout << "Masukkan jumlah sisi: ";
    cin >> edges;

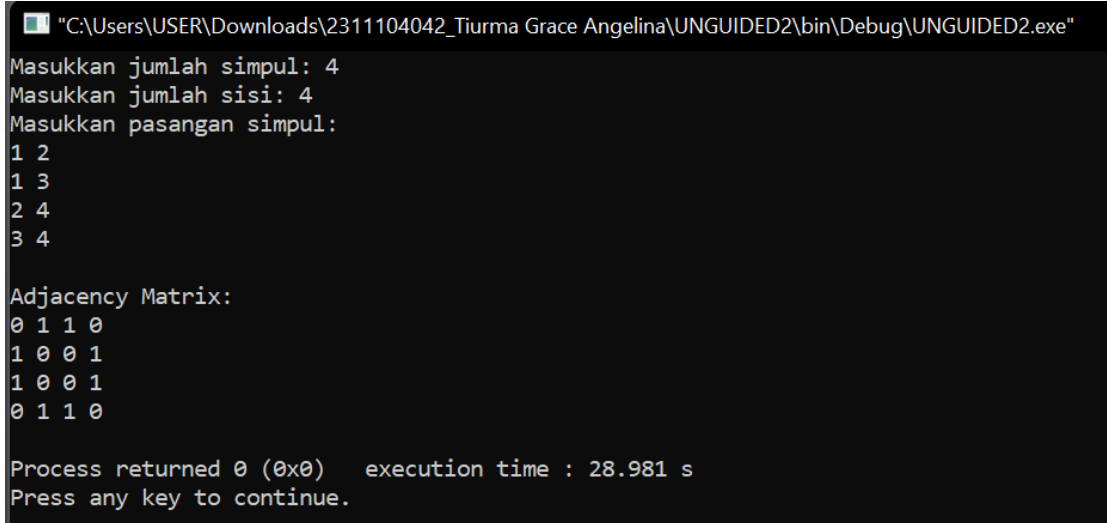
    cout << "Masukkan pasangan simpul: " << endl;
    for(int i = 0; i < edges; i++) {
        int v1, v2;
        cin >> v1 >> v2;
        ElmNode *node1 = GetNode(G, v1);
        ElmNode *node2 = GetNode(G, v2);
        if(node1 != NULL && node2 != NULL) {
            ConnectNode(node1, node2);
        }
    }
}

```

```
        cout << endl;
        PrintMatrix(G);

        return 0;
    }
```

#### Output :



```
"C:\Users\USER\Downloads\2311104042_Tiurma Grace Angelina\UNGUIDED2\bin\Debug\UNGUIDED2.exe"
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

Process returned 0 (0x0)   execution time : 28.981 s
Press any key to continue.
```

#### Penjelasan :

Program ini menggunakan **struct** untuk merepresentasikan graph sebagai *undirected graph* (graph tak berarah) dengan **Adjacency Matrix**.

##### a. Struct ElmNode

- **info**: Menyimpan data dari simpul (dalam hal ini berupa angka).
- **firstEdge**: Pointer ke daftar edge pertama yang terhubung dengan simpul ini.
- **Next**: Pointer ke simpul berikutnya dalam linked list.

##### b. Struct ElmEdge

- **Node**: Pointer ke simpul tujuan yang terhubung dengan edge ini.
- **Next**: Pointer ke edge berikutnya dalam daftar edge.

##### c. Struct Graph

- **first**: Pointer ke simpul pertama dalam graph.
- **numNodes**: Jumlah simpul dalam graph.

#### Fungsi Utama

##### a. CreateGraph(Graph &G)

Menginisialisasi graph:

- first di-set menjadi NULL (graph kosong).
- numNodes di-set menjadi 0 (belum ada simpul).

#### **b. InsertNode(Graph &G, int X)**

Menambahkan simpul baru ke graph:

1. Membuat simpul baru dengan nilai info yang diberikan.
2. Jika graph kosong, simpul menjadi simpul pertama (first).
3. Jika tidak, simpul baru ditambahkan di akhir linked list.
4. Jumlah simpul (numNodes) diperbarui.

#### **c. ConnectNode(ElmNode \*N1, ElmNode \*N2)**

Menghubungkan dua simpul secara *bidirectional*:

1. Membuat edge baru dari N1 ke N2 dan menambahkannya ke daftar edge simpul N1.
2. Membuat edge baru dari N2 ke N1 dan menambahkannya ke daftar edge simpul N2.

#### **d. GetNode(Graph G, int index)**

Mengambil simpul ke-index (berdasarkan urutan penambahan):

1. Menelusuri linked list simpul sebanyak index langkah.
2. Mengembalikan pointer ke simpul jika ditemukan, atau NULL jika tidak ada.

#### **e. PrintMatrix(Graph G)**

Mencetak **Adjacency Matrix** dari graph:

1. Looping untuk setiap pasangan simpul (node1, node2).
2. Mengecek apakah ada edge yang menghubungkan kedua simpul dengan menelusuri daftar edge node1.
3. Jika edge ditemukan, mencetak 1, jika tidak, mencetak 0.

### **Fungsi main()**

#### **a. Input Jumlah Simpul**

1. Program meminta jumlah simpul (vertices).
2. Menambahkan simpul ke graph dengan nilai 1 hingga vertices menggunakan InsertNode.



**b. Input Jumlah dan Pasangan Sisi**

1. Program meminta jumlah sisi (edges).
2. Untuk setiap sisi, program meminta pasangan simpul (v1 dan v2).
3. Menghubungkan simpul v1 dan v2 menggunakan ConnectNode.

**c. Cetak Adjacency Matrix**

Memanggil PrintMatrix untuk mencetak matriks adjacency dari graph.

## **5. Kesimpulan**

Laporan praktikum ini membahas implementasi graph yang dapat direpresentasikan menggunakan adjacency matrix dan adjacency list, dengan kelebihan dan kekurangan masing-masing. Algoritma BFS cocok untuk pencarian jalur terpendek, sedangkan DFS lebih efektif untuk eksplorasi menyeluruh. Pemilihan struktur dan algoritma yang tepat bergantung pada karakteristik graph dan kebutuhan masalah, di mana adjacency list lebih efisien untuk graph yang sparse. Implementasi algoritma traversal ini dapat diterapkan pada berbagai kasus nyata seperti jaringan sosial dan pencarian rute.