

**LAPORAN PRAKTIKUM**  
**Modul XIV**  
**“GRAPH”**



**Disusun Oleh:**  
**Dewi Atika Muthi -2211104042**  
**SE-07-02**

**Dosen:**  
**Wahyu Andi Saputra**

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

## 1. Tujuan

Berikut ini adalah tujuan praktikum ini:

- a. Memahami konsep graph dan implementasinya dalam pemrograman
- b. Mengimplementasikan graph menggunakan pointer dengan struktur data multilist

## 2. Landasan Teori

**Graph** merupakan struktur data non-linier yang terdiri dari himpunan tidak kosong dari node (vertex) dan garis penghubung (edge). Graph dapat digunakan untuk merepresentasikan berbagai hubungan antar objek, seperti peta jalan, jaringan komputer, atau hubungan pertemanan di media sosial.

### Jenis-jenis Graph:

#### 1. Graph Berarah (Directed Graph)

- Merupakan graph dimana tiap node memiliki edge dengan arah tertentu
- Edge menunjukkan arah hubungan antar node
- Hubungan dari node A ke B tidak otomatis berarti ada hubungan dari B ke A

#### 2. Graph Tidak Berarah (Undirected Graph)

- Merupakan graph dimana edge yang menghubungkan antar node tidak memiliki arah
- Jika node A terhubung dengan node B, maka node B juga terhubung dengan node A
- Edge dapat memiliki bobot/nilai yang merepresentasikan jarak, biaya, atau metric lainnya

### Representasi Graph:

Graph dapat direpresentasikan dengan dua cara utama:

#### 1. Matriks Ketetanggaan (Adjacency Matrix)

- Menggunakan array 2 dimensi berukuran  $n \times n$  ( $n$  = jumlah node)
- Simpel untuk diimplementasikan tapi membutuhkan memori yang besar
- Cocok untuk graph yang padat (dense graph)

#### 2. Multilist

- Menggunakan struktur data linked list
- Lebih efisien dalam penggunaan memori
- Cocok untuk graph yang jarang (sparse graph)
- Bersifat dinamis sehingga ukuran dapat disesuaikan

### Metode Penelusuran Graph:

### 1. Breadth First Search (BFS)

- Menelusuri graph level per level
- Menggunakan struktur data Queue
- Mengunjungi semua node pada level yang sama sebelum pindah ke level berikutnya

### 2. Depth First Search (DFS)

- Menelusuri graph dengan mengutamakan kedalaman
- Menggunakan struktur data Stack
- Mengunjungi node sampai ke ujung cabang sebelum backtrack

## 3. Guided

### 1) Implementasi Graph Menggunakan Pointer

#### Source code guided.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;
    }
}

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);

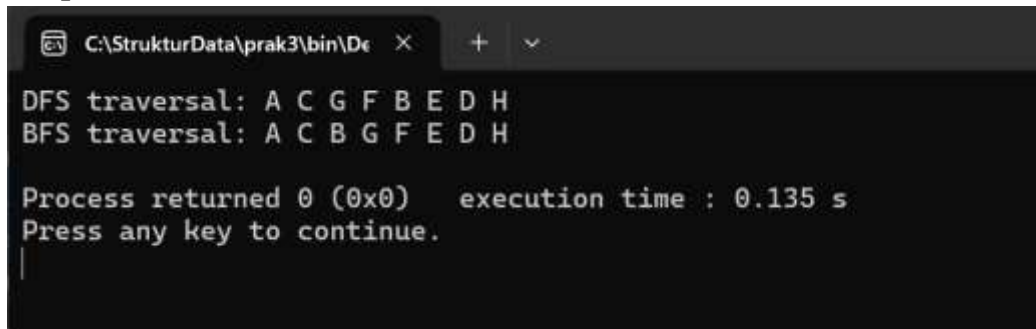
        cout << "DFS traversal: ";
        ResetVisited(G);
        PrintDFS(G, A);
        cout << endl;

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

        return 0;
    }

```

### Output:



```

C:\StrukturData\prak3\bin\De x + v
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 : 0.135 s
Press any key to continue.
|

```

### Deskripsi program:

Program ini merupakan implementasi struktur data Graph menggunakan pointer dalam C++. Program mendemonstrasikan pembuatan graph, penambahan node, penghubungan antar node, dan dua metode traversal graph yaitu DFS (Depth First Search) dan BFS (Breadth First Search).

### Struktur Data yang Digunakan:

- Struct ElmEdge: Menyimpan pointer ke node tujuan dan edge berikutnya
- Struct ElmNode: Menyimpan informasi node (char), status kunjungan, pointer ke edge pertama, dan node berikutnya

- Struct Graph: Menyimpan pointer ke node pertama

Fungsi-fungsi yang Diimplementasikan:

1. CreateGraph(Graph &G)
  - Inisialisasi graph kosong dengan mengeset first = NULL
2. InsertNode(Graph &G, char X)
  - Menambahkan node baru ke dalam graph
  - Node baru ditambahkan di akhir list
3. ConnectNode(ElmNode \*N1, ElmNode \*N2)
  - Menghubungkan dua node dengan membuat edge dari N1 ke N2
  - Edge baru ditambahkan di awal list edge
4. PrintInfoGraph(Graph G)
  - Mencetak informasi semua node dalam graph
5. ResetVisited(Graph &G)
  - Mereset status kunjungan semua node menjadi false
6. PrintDFS(Graph G, ElmNode \*N)
  - Implementasi traversal Depth First Search
  - Menggunakan rekursi untuk mengunjungi node-node
7. PrintBFS(Graph G, ElmNode \*N)
  - Implementasi traversal Breadth First Search
  - Menggunakan queue untuk mengunjungi node-node

#### 4. Unguided

- 1) Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

##### Source code unguided1.cpp:

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

const int MAX = 10;

class Graph {
private:
    int numVertices;
    string vertices[MAX];
    int adjMatrix[MAX][MAX];

public:
    Graph() {
        numVertices = 0;
        for(int i = 0; i < MAX; i++) {
            for(int j = 0; j < MAX; j++) {
                adjMatrix[i][j] = 0;
            }
        }
    }

    void addVertex(string name) {
        if(numVertices < MAX) {
            vertices[numVertices] = name;
            numVertices++;
        }
    }
}
```

```

void addEdge(int start, int end, int weight) {
    adjMatrix[start][end] = weight;
}

void printMatrix() {
    cout << setw(10) << " ";
    for(int i = 0; i < numVertices; i++) {
        cout << setw(10) << vertices[i];
    }
    cout << endl;

    for(int i = 0; i < numVertices; i++) {
        cout << setw(10) << vertices[i];
        for(int j = 0; j < numVertices; j++) {
            cout << setw(10) << adjMatrix[i][j];
        }
        cout << endl;
    }
}

int getNumVertices() {
    return numVertices;
}

string getVertex(int index) {
    return vertices[index];
}
};

int main() {
    Graph graph;
    int numVertices;
    string vertexName;

    cout << "Silakan masukan jumlah simpul : ";
    cin >> numVertices;

    cout << "Silakan masukan nama simpul" << endl;
    for(int i = 0; i < numVertices; i++) {
        cout << "Simpul " << i+1 << " : ";
        cin >> vertexName;
        graph.addVertex(vertexName);
    }

    cout << "Silakan masukkan bobot antar simpul" << endl;
    for(int i = 0; i < numVertices; i++) {
        for(int j = 0; j < numVertices; j++) {
            string start = graph.getVertex(i);
            string end = graph.getVertex(j);
            cout << start << "---> " << end << " = ";
            int weight;
            cin >> weight;
            graph.addEdge(i, j, weight);
        }
    }

    cout << endl;
    graph.printMatrix();

    return 0;
}

```

## Output:

```
C:\StrukturData\prak3\bin\De x + v
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 : 22.567 s
Press any key to continue.
```

## Deskripsi Program:

Program mengimplementasikan graph berbobot menggunakan adjacency list

Menggunakan struct Node untuk menyimpan informasi kota

Menggunakan struct Edge untuk menyimpan informasi jarak antar kota

Program menggunakan konsep graph tidak berarah, sehingga jarak A ke B sama dengan B ke A

Cara kerja program:

- Program meminta user memasukkan jumlah simpul
- User diminta memasukkan nama untuk setiap simpul
- Program meminta user memasukkan bobot untuk setiap pasangan simpul
- Program menampilkan matriks adjacency yang berisi bobot antar simpul

2) Buatlah sebuah program C++ untuk merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program harus dapat:

- Menerima input jumlah simpul dan jumlah sisi.
- Menerima input pasangan simpul yang terhubung oleh sisi.
- Menampilkan adjacency matrix dari graf tersebut.

## Source code unguided2.cpp:

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

const int MAX_VERTICES = 10;

class Graph {
private:
    int vertices;
    int matrix[MAX_VERTICES][MAX_VERTICES];
```



```

public:
    Graph(int v) {
        vertices = v;
        for(int i = 0; i < vertices; i++) {
            for(int j = 0; j < vertices; j++) {
                matrix[i][j] = 0;
            }
        }
    }

    void addEdge(int v1, int v2) {
        matrix[v1-1][v2-1] = 1;
        matrix[v2-1][v1-1] = 1;
    }

    void printMatrix() {
        cout << "\nAdjacency Matrix:" << endl;
        for(int i = 0; i < vertices; i++) {
            for(int j = 0; j < vertices; j++) {
                cout << matrix[i][j] << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    int vertices, edges;

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

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

    Graph graph(vertices);

    cout << "\nMasukkan pasangan simpul:\n";
    for(int i = 0; i < edges; i++) {
        int v1, v2;
        cin >> v1 >> v2;
        graph.addEdge(v1, v2);
    }

    graph.printMatrix();

    return 0;
}

```

**Output:**

```
C:\StrukturData\prak3\bin\De x + - □ X
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 : 23.127 s
Press any key to continue.
```

### Deskripsi Program:

Mengimplementasikan graph tidak berarah menggunakan matriks adjacency

Menggunakan array 2D untuk merepresentasikan hubungan antar simpul

**Matrix[i][j] = 1** jika ada edge antara simpul i dan j, 0 jika tidak ada

Program ini dapat menerima input jumlah simpul dan sisi, menambah edge antar simpul, serta menampilkan matriks adjacency.

## 5. Kesimpulan

Pada praktikum ini kita mempelajari graph, dimana graph merupakan struktur data yang terdiri dari node dan edge untuk merepresentasikan hubungan antar objek dan dapat diimplementasikan menggunakan multilist dengan pointer. Pemahaman tentang graph dan implementasinya sangat penting dalam pengembangan aplikasi yang membutuhkan representasi hubungan antar objek, seperti sistem navigasi, jaringan sosial, atau optimasi rute.