# Jobsheet 12 Graph Algoritma Struktur Data

**Muhammad Farrel Caesarian**
**TI-1A/20**
**2341720053**
**POLITEKNIK NEGERI MALANG**

## Percobaan 1

Kode

```java
import java.util.Scanner;
public class GraphMain20{
    Run | Debug
    public static void main(String[] args) throws Exception {
        Graph20 gedung = new Graph20(v:6);
        Scanner input20 = new Scanner(System.in);
        gedung.addEdge(asal:0, tujuan:1, jarak:50);
        gedung.addEdge(asal:0, tujuan:2, jarak:100);
        gedung.addEdge(asal:1, tujuan:3, jarak:70);
        gedung.addEdge(asal:2, tujuan:3, jarak:40);
        gedung.addEdge(asal:3, tujuan:4, jarak:60);
        gedung.addEdge(asal:4, tujuan:5, jarak:80);
        gedung.degree(asal:0);
        gedung.printGraph();
        gedung.removeEdge(asal:1, tujuan:3);
        gedung.printGraph();

        System.out.print(s:"Masukkan Gedung asal : ");
        int asal = input20.nextInt();
        System.out.print(s:"Masukkan Gedung tujuan : ");
        int tujuan = input20.nextInt();

        boolean isDirect = gedung.isDirectlyConnected(asal, tujuan);
        if (isDirect) {
            System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan) +
        } else {
            System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan) +
        }
    }
}
```

```java
public class Graph20{
    int vertex;
    DoubleLinkedLists20 list[];

    public Graph20(int v){
        vertex = v;
        list = new DoubleLinkedLists20[v];
        for(int i=0;i<v;i++){
            list[i] = new DoubleLinkedLists20();
        }
    }
    public void addEdge(int asal, int tujuan, int jarak){
        list[asal].addFirst(tujuan, jarak);
        list[tujuan].addFirst(asal, jarak);
    }
    public void degree(int asal) throws Exception{
        int k, totalIn=0, totalOut=0;
        for(int i=0;i<vertex;i++){
            //inDegree
            for(int j=0;j<list[i].size();j++){
                if(list[i].get(j) == asal){
                    ++totalIn;
                }
            }
            //outDegree
            for(k=0;k<list[asal].size();k++){
```

```
                list[asal].get(k);
            }
            totalOut=k;
        }
        System.out.println("InDegree dari Gedung " +(char) ('A' +asal)
+ ":" +totalIn);
        System.out.println("OutDegree dari Gedung " +(char) ('A' +asal)
+ ":" +totalOut);
        System.out.println("Degree dari Gedung " +(char) ('A' +asal) +
":" +(totalIn+totalOut));
    }
    public void removeEdge(int asal, int tujuan) throws Exception {
        list[asal].remove(tujuan);
        list[tujuan].remove(asal);
    }
    public void removeAllEdge(){
        for(int i=0;i<vertex;i++){
            list[i].clear();
        }
        System.out.println("Graf berhasil dikosongkan");
    }
    public void printGraph() throws Exception {
        for (int i = 0; i < vertex; i++) {
            if (list[i].size() > 0) {
                System.out.print("Gedung " + (char) ('A' + i) + "
Terhubung dengan: ");
                for (int j = 0; j < list[i].size(); j++) {
                    System.out.print((char) ('A' + list[i].get(j)) + "
(" + list[i].getJarak(j) + " m), ");
                }
                System.out.println();
            }
        }
        System.out.println();
    }
public boolean isDirectlyConnected(int asal, int tujuan) throws
Exception {
        for (int i = 0; i < list[asal].size(); i++) {
            if (list[asal].get(i) == tujuan) {
                return true;
            }
        }
        return false;
    }
}
```

Output

```
InDegree dari Gedung A:0
OutDegree dari Gedung A:2
Degree dari Gedung A:2
Gedung ATerhubung dengan
C (100 m), B (50 m),
Gedung BTerhubung dengan
D (70 m), B (50 m),
Gedung CTerhubung dengan
D (40 m), C (100 m),
Gedung DTerhubung dengan
E (60 m), D (40 m), D (70 m),
Gedung ETerhubung dengan
F (80 m), E (60 m),
Gedung FTerhubung dengan
F (80 m),
```

**2.1.3 Pertanyaan**

1. Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!

Jawab:

```
InDegree dari Gedung A:2
OutDegree dari Gedung A:2
Degree dari Gedung A:4
Gedung A Terhubung dengan C (100 m), B (50 m),
Gedung B Terhubung dengan D (70 m), A (50 m),
Gedung C Terhubung dengan D (40 m), A (100 m),
Gedung D Terhubung dengan E (60 m), C (40 m), B (70 m),
Gedung E Terhubung dengan F (80 m), D (60 m),
Gedung F Terhubung dengan E (80 m),

Gedung A Terhubung dengan C (100 m), B (50 m),
Gedung B Terhubung dengan A (50 m),
Gedung C Terhubung dengan D (40 m), A (100 m),
Gedung D Terhubung dengan E (60 m), C (40 m),
Gedung E Terhubung dengan F (80 m), D (60 m),
Gedung F Terhubung dengan E (80 m),
```

2. Pada class Graph, terdapat atribut **list[]** bertipe DoubleLinkedList. Sebutkan tujuan pembuatan variabel tersebut!

Jawab: Variabel list[] dalam class Graph digunakan untuk mengimplementasikan adjacency list, yang berfungsi untuk menyimpan informasi tetangga setiap simpul secara efisien dan mendukung operasi-operasi graf yang diperlukan.

3. Jelaskan alur kerja dari method **removeEdge**!

Jawab:

-Menghapus Edge dari Vertex Asal ke Vertex Tujuan: **list[asal].remove(tujuan)**: Mencari node dalam DoubleLinkedLists20 yang diwakili oleh list[asal] yang memiliki data tujuan. Menghapus node tersebut dari daftar, mengatur ulang pointer prev dan next dari node tetangga yang relevan, dan mengurangi ukuran list.

-Menghapus Edge dari Vertex Tujuan ke Vertex Asal: **list[tujuan].remove(asal)**: Mencari node dalam DoubleLinkedLists20 yang diwakili oleh list[tujuan] yang memiliki data asal. Menghapus node tersebut dari daftar, mengatur ulang pointer prev dan next dari node tetangga yang relevan, dan mengurangi ukuran list.

4. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain saat digunakan pada method **addEdge** pada class Graph?
Jawab: Karena Kompleksitas waktu yang rendah (O(1)) dimana hal tersebut memastikan penambahan elemen dilakukan dengan cepat. Sederhana dan konsisten, dengan selalu menambahkan elemen di awal list tanpa perlu memikirkan urutan atau posisi yang tepat.

5. Modifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).

```
Masukkan gedung asal: 2
Masukkan gedung tujuan: 3
Gedung C dan D bertetangga

Masukkan gedung asal: 2
Masukkan gedung tujuan: 5
Gedung C dan F tidak bertetangga
```

Jawab:

```
Masukkan Gedung asal : 2
Masukkan Gedung tujuan : 3
Gedung C dan Gedung D bertetangga
Masukkan Gedung asal : 2
Masukkan Gedung tujuan : 5
Gedung C dan Gedung F tidak bertetangga
PS E:\Semester 2\SecondSemester\Jobsheet Graph>
```

## Percobaan 2

Kode

```java
import java.util.Scanner;
public class GraphMain20{
    Run | Debug
    public static void main(String[] args) throws Exception {
        Graph20 gedung = new Graph20(v:6);
        GraphMatriks20 gdg = new GraphMatriks20(v:6);
        Scanner input20 = new Scanner(System.in);

        gdg.makeEdge(asal:0, tujuan:1, jarak:50);
        gdg.makeEdge(asal:1, tujuan:0, jarak:60);
        gdg.makeEdge(asal:1, tujuan:2, jarak:70);
        gdg.makeEdge(asal:2, tujuan:1, jarak:80);
        gdg.makeEdge(asal:2, tujuan:3, jarak:40);
        gdg.makeEdge(asal:3, tujuan:0, jarak:90);
        gdg.printGraph();
        System.out.println(x:"Hasil setelah penghapusan edge");
        gdg.removeEdge(asal:2, tujuan:1);
        gdg.printGraph();
        /*gedung.addEdge(0, 1, 50);
        gedung.addEdge(0, 2, 100);
```

```java
public class GraphMatriks20{
    int vertex;
    int[][] matriks;

    public GraphMatriks20(int v){
        vertex = v;
        matriks = new int[v][v];
    }

    public void makeEdge(int asal, int tujuan, int jarak){
        matriks[asal][tujuan] = jarak;
    }

    public void removeEdge(int asal, int tujuan){
        matriks[asal][tujuan] = -1;
    }

    public void printGraph(){
        for(int i=0;i<vertex;i++){
            System.out.print("Gedung " +(char)('A' +i)+": ");
            for(int j=0;j<vertex;j++){
                if(matriks[i][j] !=-1){
                    System.out.print("Gedung " +(char)('A' +j)+" ("+matriks[i][j]+" m), ");
                }
            }
            System.out.println();
        }
    }
}
```

Output

```
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung C: Gedung A (0 m), Gedung B (80 m), Gedung C (0 m), Gedung D (40 m), Gedung E (0 m), Gedung F (0 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung E: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung F: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Hasil setelah penghapusan edge
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung C: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (40 m), Gedung E (0 m), Gedung F (0 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung E: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung F: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
```

**2.2.3 Pertanyaan**

1. Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!

Jawab:

```
public void removeEdge(int asal, int tujuan){
    matriks[asal][tujuan] = -1;
}
```

Before

```
public void removeEdge(int asal, int tujuan){
    matriks[asal][tujuan] = 0;
}
```

After

2. Apa jenis graph yang digunakan pada Percobaan 2?

Jawab: Graph Representasi Matriks Adjacency (Adjacency Matrix Representation). karena Graph di representasikan dengan matriks dua dimensi dan Nilai matriks [i][j] menyimpan jarak dari vertex i ke vertex j.

3. Apa maksud dari dua baris kode berikut?

```
gdg.makeEdge(1, 2, 70);
gdg.makeEdge(2, 1, 80);
```

Jawab: Membuat Edge dengan asal 1 tujuan 2 jarak 70 dan Edge dengan asal 2 tujuan 1 jarak 80.

4. Modifikasi kode program sehingga terdapat method untuk menghitung degree, termasuk inDegree dan outDegree!

Jawab:

```
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung C: Gedung A (0 m), Gedung B (80 m), Gedung C (0 m), Gedung D (40 m), Gedung E (0 m), Gedung F (0 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung E: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung F: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Hasil setelah penghapusan edge
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung C: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (40 m), Gedung E (0 m), Gedung F (0 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung E: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Gedung F: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m), Gedung E (0 m), Gedung F (0 m),
Out Degree : 1
In Degree : 2
Degree : 3
```

```
public int outDegree(int v) {
    int outDegree = 0;
    for (int j = 0; j < vertex; j++) {
        if (matriks[v][j] != 0) {
            outDegree++;
        }
    }
    return outDegree;
}

public int inDegree(int v) {
    int inDegree = 0;
    for (int i = 0; i < vertex; i++) {
        if (matriks[i][v] != 0) {
            inDegree++;
        }
    }
    return inDegree;
}

public int degree(int v) {
    return inDegree(v) + outDegree(v);
}
```

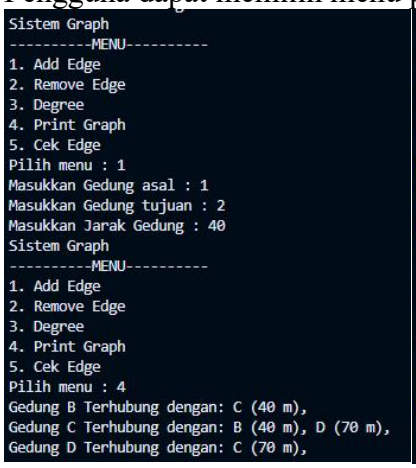
## 3. Latihan Praktikum

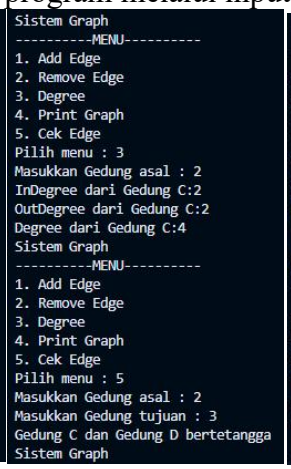
**Waktu percobaan: 90 menit**

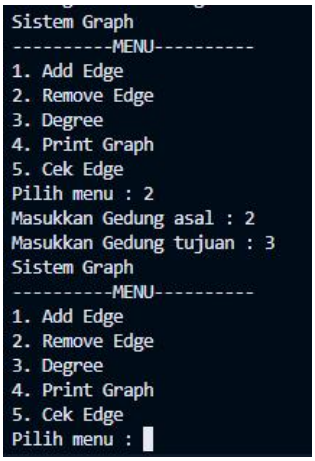
1. Modifikasi kode program pada class **GraphMain** sehingga terdapat menu program yang bersifat dinamis, setidaknya terdiri dari:

a) Add Edge
b) Remove Edge
c) Degree
d) Print Graph
e) Cek Edge

Pengguna dapat memilih menu program melalui input Scanner




```
int choice;
        do {
            System.out.println("Sistem Graph");
            System.out.println("----------MENU----------");
            System.out.println("1. Add Edge");
            System.out.println("2. Remove Edge");
            System.out.println("3. Degree");
            System.out.println("4. Print Graph");
            System.out.println("5. Cek Edge");
            System.out.println("6. Keluar");
            System.out.print("Pilih menu : ");
            choice = input20.nextInt();
            switch (choice) {
            case 1:
                System.out.print("Masukkan Gedung asal : ");
                int asal1 = input20.nextInt();
                System.out.print("Masukkan Gedung tujuan : ");
                int tujuan1 = input20.nextInt();
                System.out.print("Masukkan Jarak Gedung : ");
                int jarak1 = input20.nextInt();
                gedung.addEdge(asal1, tujuan1, jarak1);
                break;
            case 2:
                System.out.print("Masukkan Gedung asal : ");
                int asal2 = input20.nextInt();
                System.out.print("Masukkan Gedung tujuan : ");
```

```
                int tujuan2 = input20.nextInt();
                gedung.removeEdge(asal2, tujuan2);
                break;
            case 3:
                System.out.print("Masukkan Gedung asal : ");
                int asal3 = input20.nextInt();
                gedung.degree(asal3);
                break;
            case 4:
                gedung.printGraph();
                break;
            case 5:
                System.out.print("Masukkan Gedung asal : ");
                int asal5 = input20.nextInt();
                System.out.print("Masukkan Gedung tujuan : ");
                int tujuan5 = input20.nextInt();

                boolean isDirect = gedung.isDirectlyConnected(asal5,
tujuan5);
                if (isDirect) {
                System.out.println("Gedung " + (char) ('A' + asal5) + "
dan Gedung " + (char) ('A' + tujuan5) + " bertetangga");
            } else {
                System.out.println("Gedung " + (char) ('A' + asal5) + "
dan Gedung " + (char) ('A' + tujuan5) + " tidak bertetangga");
            }
                break;
            case 6:
                System.exit(0);
                break;
            default:
                break;
        }
    } while (choice !=-1);
```

2. Tambahkan method **updateJarak** pada Percobaan 1 yang digunakan untuk mengubah jarak antara dua node asal dan tujuan!

```
public void setJarak(int index, int jarak) {
    Node20 temp = head;
    int count = 0;
    while (temp != null) {
        if (count == index) {
            temp.jarak = jarak;
            break;
        }
        count++;
        temp = temp.next;
    }
}
```

```java
public void updateJarak(int asal, int tujuan, int jarakBaru) throws Exception {
    for (int i = 0; i < list[asal].size(); i++) {
        if (list[asal].get(i) == tujuan) {
            list[asal].setJarak(i, jarakBaru);
            break;
        }
    }
    for (int i = 0; i < list[tujuan].size(); i++) {
        if (list[tujuan].get(i) == asal) {
            list[tujuan].setJarak(i, jarakBaru);
            break;
        }
    }
    System.out.println("Jarak antara Gedung " + (char)('A' + asal) + " dan Gedung " + (char)('A' +
    tujuan) + " berhasil diperbarui menjadi " + jarakBaru + " m.");
}
```

3. Tambahkan method **hitungEdge** untuk menghitung banyaknya edge yang terdapat di dalam graf!

```java
public int hitungEdge() {
    int jumlahEdge = 0;
    for (int i = 0; i < vertex; i++) {
        jumlahEdge += list[i].size();
    }
    // Karena setiap edge terhitung dua kali (asal ke tujuan dan tujuan ke asal), maka dibagi 2
    return jumlahEdge / 2;
}
```