JOBSHEET XII

Graph

1. Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model graph
2. membuat dan mendeklarasikan struktur algoritma graph
3. menerapkan algoritma dasar graph dalam beberapa studi kasus

2. Praktikum

2.1 Percobaan 1: Implementasi Graph menggunakan Linked List

Sebuah universitas membuat program untuk memodelkan graf **berarah berbobot** yang mewakili gedung-gedung dan jarak antar gedung tersebut menggunakan Linked List. Setiap gedung dihubungkan dengan jalan yang memiliki jarak tertentu (dalam meter). Perhatikan class diagram Graph berikut ini.

Graph<NoAbsen>
vertex: int DoubleLinkedList: list[]
addEdge(asal: int, tujuan: int): void degree(asal: int): void removeEdge(asal: int, tujuan: int): void removeAllEdges(): void printGraph(): void

2.1.1 Langkah-langkah Percobaan

Waktu percobaan (90 menit)

1. Buka text editor. Buat class **Node<NoAbsen>.java** dan class **DoubleLinkedList<NoAbsen>.java** sesuai dengan praktikum **Double Linked List.**

   A. Class Node

   Kode program yang terdapat pada class **Node** belum dapat mengakomodasi kebutuhan pembuatan graf berbobot, sehingga diperlukan sedikit modifikasi. Setelah Anda menyalin kode program dari class **Node** pada praktikum Double Linked List, tambahkan atribut **jarak** bertipe **int** untuk menyimpan bobot graf

```
int data;
Node prev, next;
int jarak;

Node(Node prev, int data, int jarak, Node next) {
    this.prev = prev;
    this.data = data;
    this.next = next;
    this.jarak = jarak;
}
```

```
public class Node26 {
    int data;
    Node26 prev, next;
    int jarak;

    Node26 (Node26 prev, int data, int jarak,Node26 next) {
        this.data = data;
        this.prev = prev;
        this.next = next;
        this.jarak = jarak;
    }
}
```

B. Class DoubleLinkedList

Setelah Anda menyalin kode program dari class **DoubleLinkedList** pada praktikum Double Linked List, lakukan modifikasi pada method **addFirst** agar dapat menerima parameter **jarak** dan digunakan saat instansiasi Node

```
public void addFirst(int item, int jarak) {
    if (isEmpty()) {
        head = new Node(null, item, jarak, null);
    } else {
        Node newNode = new Node(null, item, jarak, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}
```

```
public void addFirst(int item, int jarak) {
    if (isEmpty()) {
        head = new Node26(prev:null, item, jarak, next:null);
    } else {
        Node26 newNode = new Node26(prev:null, item, jarak, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}
```

Selanjutnya buat method **getJarak** (hampir sama seperti method **get**) yang digunakan untuk mendapatkan nilai jarak edge antara dua node.

```
public int getJarak(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception(message:"Nilai indeks di luar batas");
    }
    Node tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.jarak;
}
```

```
public int getJarak(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception(message:"Nilai index di luar batas");
    }
    Node26 tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.jarak;
}
```

Modifikasi method **remove** agar dapat melakukan penghapusan edge sesuai dengan node asal dan tujuan pada graf. Pada praktikum Double Linked List, parameter **index** digunakan untuk menghapus data sesuai posisi pada indeks tertentu, sedangkan pada Graf ini, penghapusan didasarkan pada data node tujuan, sehingga modifikasi kode diperlukan untuk menghindari index out of bound.

```
public void remove(int index) {
    Node current = head;
        while (current != null) {
            if (current.data == index) {
                if (current.prev != null) {
                    current.prev.next = current.next;
                } else {
                    head = current.next;
                }
                if (current.next != null) {
                    current.next.prev = current.prev;
                }
                break;
            }
            current = current.next;
        }
}
```

```java
public void remove(int index) {
    Node26 current = head;
    while (current != null) {
        if (current.data == index) {
            if (current.prev != null) {
                current.prev.next = current.next;
            } else {
                head = current.next;
            }
            if (current.next != null) {
                current.next.prev = current.prev;
            }
            break;
        }
        current = current.next;
    }
}
```

C. Class Graph

2. Buat file baru, beri nama **Graph<NoAbsen>.java**
3. Lengkapi class **Graph** dengan atribut yang telah digambarkan di dalam pada class diagram, yang terdiri dari atribut **vertex** dan **DoubleLinkedList**

```java
int vertex;
DoubleLinkedList list[];
```

```java
public class Graph26 {
    int vertex;
    DoubleLinkedList26[] list;
}
```

4. Tambahkan konstruktor default untuk menginisialisasi variabel **vertex** dan menambahkan perulangan jumlah vertex sesuai dengan panjang array yang telah ditentukan.

```java
public Graph(int v) {
    vertex = v;
    list = new DoubleLinkedList[v];
    for (int i = 0; i < v; i++) {
        list[i] = new DoubleLinkedList();
    }
}
```

```
Graph26(int v) {
    vertex = v;
    list = new DoubleLinkedList26[v];
    for (int i = 0; i < v; i++) {
        list[i] = new DoubleLinkedList26();
    }
}
```

5. Tambahkan method **addEdge()** untuk menghubungkan dua node. Baris kode program berikut digunakan untuk graf berarah (directed).

```
public void addEdge(int asal, int tujuan, int jarak) {
    list[asal].addFirst(tujuan, jarak);
}
```

Apabila graf yang dibuat adalah undirected graph, maka tambahkan kode berikut.

```
list[tujuan].addFirst(asal, jarak);
```

```
void addEdge(int asal, int tujuan, int jarak) {
    list[asal].addFirst(tujuan, jarak);
    //list[tujuan].addFirst(asal, jarak); // Apabila graf yang dibuat adalah undirected graph
}
```

6. Tambahkan method **degree()** untuk menampilkan jumlah derajat lintasan pada setiap vertex. Kode program berikut digunakan untuk menghitung degree pada graf berarah

```
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));
}
```

Apabila graf yang dibuat adalah undirected graph, maka cukup gunakan kode berikut.

```
System.out.println("Degree dari Gedung " + (char) ('A' +asal) + ": " + list[asal].size());
```

```
    void degree(int asal) throws Exception {
        int k, totalIn = 0, totalOut = 0;
        for (int i = 0; i < vertex; i++) {
            for (int j = 0; j < list[i].size(); j++) {
                if (list[i].get(j) == asal) {
                    ++totalIn;
                }
            }
            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' + vertex) + ": " + (totalIn + totalOut));
        // System.out.println("Degree dari gedung" + (char)('A' + asal) + ": " + list[asal].size());
    }
}
```

7. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graph. Penghapusan membutuhkan 2 parameter yaitu node **asal** dan **tujuan**.

```
public void removeEdge(int asal, int tujuan) throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (i == tujuan) {
            list[asal].remove(tujuan);
        }
    }
}
```

```
void removeEdge(int asal, int tujuan) throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (i == tujuan) {
            list[asal].remove(tujuan);
        }
    }
}
```

8. Tambahkan method **removeAllEdges()** untuk menghapus semua vertex yang ada di dalam graf.

```
public void removeAllEdges() {
    for (int i = 0; i < vertex; i++) {
        list[i].clear();
    }
    System.out.println("Graf berhasil dikosongkan");
}
```

```
void removeAllEdges() {
    for (int i = 0; i < vertex; i++) {
        list[i].clear();
    }
    System.out.println(x:"Graph berhasil dikosongkan");
}
```

9. Tambahkan method **printGraph()** untuk mencetak graf.

```
public void printGraph() throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (list[i].size() > 0) {
            System.out.println("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(x:"");
        }
    }
    System.out.println(x:"");
}
```

```
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();
}
```

D. Class Utama

10. Buat file baru, beri nama **GraphMain<NoAbsen>.java**

11. Tuliskan struktur dasar bahasa pemrograman Java yang terdiri dari fungsi **main**

12. Di dalam fungsi **main**, lakukan instansiasi object Graph bernama **gedung** dengan nilai parameternya adalah 6.

```
Graph gedung = new Graph(6);
```

13. Tambahkan beberapa edge pada graf, tampilkan degree salah satu node, kemudian tampilkan grafnya.

```
Graph gedung = new Graph(6);
gedung.addEdge(0, 1, 50);
gedung.addEdge(0, 2, 100);
gedung.addEdge(1, 3, 70);
gedung.addEdge(2, 3, 40);
gedung.addEdge(3, 4, 60);
gedung.addEdge(4, 5, 80);
gedung.degree(0);
gedung.printGraph();
```

```
public static void main(String[] args) throws Exception {
    Graph26 gedung = new Graph26(v:6);
    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();
```

14. Compile dan run program.

    Catatan: Degree harus disesuaikan dengan jenis graf yang digunakan. Pada kasus ini, digunakan directed weighted graph

15. Tambahkan pemanggilan method **removeEdge()**, kemudian tampilkan kembali graf tersebut.

```
gedung.removeEdge(1, 3);
gedung.printGraph();
```

16. **Commit dan push kode program ke Github**

17. Compile dan run program.

2.1.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

Hasil running pada langkah 14

```
InDegree dari Gedung A: 0
OutDegree dari Gedung A: 2
Degree dari Gedung A: 2
Gedung A terhubung dengan
C (100 m), B (50 m),
Gedung B terhubung dengan
D (70 m),
Gedung C terhubung dengan
D (40 m),
Gedung D terhubung dengan
E (60 m),
Gedung E terhubung dengan
F (80 m),
```

Hasil running pada langkah 17

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

```
InDegree dari Gedung A: 0
OutDegree dari Gedung A: 2
Degree dari Gedung G: 2
Gedung A terhubung dengan: C (100 m), B (50 m),
Gedung B terhubung dengan: D (70 m),
Gedung C terhubung dengan: D (40 m),
Gedung D terhubung dengan: E (60 m),
Gedung E terhubung dengan: F (80 m),
```

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

2.1.3 Pertanyaan

1. Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!
2. Pada class Graph, terdapat atribut **list[]** bertipe DoubleLinkedList. Sebutkan tujuan pembuatan variabel tersebut!

   Atribut list[] digunakan untuk menyimpan daftar tetangga (adjacency list) dari setiap node dalam graf. Dengan menggunakan DoubleLinkedList, operasi penambahan dan penghapusan edge menjadi lebih efisien.

3. Jelaskan alur kerja dari method **removeEdge**!
   - Menentukan node asal dan node tujuan dari edge yang akan dihapus.
   - Mengakses adjacency list dari node asal.
   - Mencari dan menghapus node tujuan dalam adjacency list tersebut.
   - Jika graf tidak berarah, ulangi langkah yang sama untuk adjacency list dari node tujuan.

4. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain saat digunakan pada method **addEdge** pada class Graph?

   addFirst() digunakan untuk menambahkan node baru di awal DoubleLinkedList agar operasi penambahan lebih cepat (O(1) waktu). Metode ini lebih efisien dibandingkan metode penambahan lain yang mungkin memerlukan traversal seluruh daftar (O(n) waktu).

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
```

```
    void isConnected(int start, int end) throws Exception {
        path = false; // Set path ke false di awal
        boolean[] visited = new boolean[vertex];
        connected(start, end, visited);
        if (path) {
            System.out.println("Gedung " + (char) ('A' + start) + " Bertetangga dengan " + (char) ('A' + end));
        } else {
            System.out.println("gedung " + (char) ('A' + start) + " Tidak Bertetangga dengan gedung2 " + (char) ('A' + end));
        }
    }

    void connected(int current, int end, boolean[] visited) throws Exception {
        if (current == end) {
            path = true;
            return;
        }
        visited[current] = true;
        for (int i = 0; i < list[current].size(); i++) {
            int neighbor = list[current].get(i);
            if (!visited[neighbor]) {
                connected(neighbor, end, visited);
                if (path) {
                    return;
                }
            }
        }
    }
}
```

```
import java.util.Scanner;

public class Graphmain26 {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Graph26 gedung = new Graph26(v:6);
        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);

        Scanner scanner = new Scanner(System.in);
        System.out.print(s:"Masukkan node asal (0-5): ");
        int start = scanner.nextInt();
        System.out.print(s:"Masukkan node tujuan (0-5): ");
        int end = scanner.nextInt();
        scanner.close();

        gedung.isConnected(start, end);

        // gedung.degree(0);
        // gedung.printGraph();
        // gedung.removeEdge(1, 3);
        // gedung.printGraph();
    }
}
```

```
Masukkan node asal (0-5): 2
Masukkan node tujuan (0-5): 3
Gedung C Bertetangga dengan D
```

```
Masukkan node asal (0-5): 2
Masukkan node tujuan (0-5): 5
Gedung C Bertetangga dengan F
```

2.2 Percobaan 2: Implementasi Graph menggunakan Matriks

Dengan menggunakan kasus yang sama dengan Percobaan 1, pada percobaan ini implementasi graf dilakukan dengan menggunakan matriks dua dimensi.

2.2.1 Langkah-langkah Percobaan

Waktu percobaan: 60 menit

1. Buat file baru, beri nama **GraphMatriks<NoAbsen>.java**
2. Lengkapi class **GraphMatriks** dengan atribut **vertex** dan **matriks**

```
int vertex;
int[][] matriks;
```

```
    int vertex;
    int matriks[][];
```

3. Tambahkan konstruktor default untuk menginisialisasi variabel **vertex** dan menginstansiasi panjang array dua dimensi yang telah ditentukan.

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

```
GraphMatriks26(int v){
    vertex = v;
    matriks = new int[v][v];
}
```

4. Untuk membuat suatu lintasan yang menghubungkan dua node, maka dibuat method **makeEdge()** sebagai berikut.

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

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

5. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graf.

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

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

6. Tambahkan method printGraph() untuk mencetak graf.

```
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();
    }
}
```

```
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();
    }
}
```

7. Tambahkan kode berikut pada file **GraphMain<NoAbsen>.java** yang sudah dibuat pada Percobaan 1.

```
GraphMatriks gdg = new GraphMatriks(4);
gdg.makeEdge(0, 1, 50);
gdg.makeEdge(1, 0, 60);
gdg.makeEdge(1, 2, 70);
gdg.makeEdge(2, 1, 80);
gdg.makeEdge(2, 3, 40);
gdg.makeEdge(3, 0, 90);
gdg.printGraph();
System.out.println("Hasil setelah penghapusan edge");
gdg.removeEdge(2, 1);
gdg.printGraph();
```

```java
public class GraphMain26 {

    Run | Debug
    public static void main(String[] args) {
        GraphMatriks26 gdg = new GraphMatriks26(v:4);
        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();
    }

}
```

8. **Commit dan push kode program ke Github**
9. Compile dan run program.

2.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

```
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m),
Gedung C: Gedung A (0 m), Gedung B (80 m), Gedung C (0 m), Gedung D (40 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (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 B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m),
Gedung C: Gedung A (0 m), Gedung B (0 m), Gedung C (0 m), Gedung D (40 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m),
```


```
Gedung A: Gedung A (0 m), Gedung B (50 m), Gedung C (0 m), Gedung D (0 m),
Gedung B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m),
Gedung C: Gedung A (0 m), Gedung B (80 m), Gedung C (0 m), Gedung D (40 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (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 B: Gedung A (60 m), Gedung B (0 m), Gedung C (70 m), Gedung D (0 m),
Gedung C: Gedung A (0 m), Gedung C (0 m), Gedung D (40 m),
Gedung D: Gedung A (90 m), Gedung B (0 m), Gedung C (0 m), Gedung D (0 m),
```

2.2.3 Pertanyaan

1. Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!
2. Apa jenis graph yang digunakan pada Percobaan 2?

   Graph yang digunakan adalah directed weighted graph (graf terarah berbobot).

3. Apa maksud dari dua baris kode berikut?

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

   Baris pertama (`gdg.makeEdge(1, 2, 70)`) menambahkan edge dari node 1 ke node 2 dengan bobot 70. Baris kedua (`gdg.makeEdge(2, 1, 80)`) menambahkan edge dari node 2 ke node 1 dengan bobot 80. Ini menunjukkan bahwa graf adalah terarah dan bobot antara dua node

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

```
public int inDegree(int node) {
    int inDegree = 0;
    for (int i = 0; i < vertex; i++) {
        if (matriks[i][node] != -1) {
            inDegree++;
        }
    }
    return inDegree;
}

public int outDegree(int node) {
    int outDegree = 0;
    for (int j = 0; j < vertex; j++) {
        if (matriks[node][j] != -1) {
            outDegree++;
        }
    }
    return outDegree;
}
}
```

GraphMatriks26:

```
GraphMatriks26(int v) {
    vertex = v;
    matriks = new int[v][v];
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            matriks[i][j] = -1;
        }
    }
}
```

GraphMain26:

```
    gdg.printGraph();

    for (int i = 0; i < gdg.vertex; i++) {
        System.out.println("Gedung " + (char)('A' + i) + ": inDegree = " + gdg.inDegree(i) + ", outDegree = " + gdg.outDegree(i));
    }

    System.out.println(x:"Hasil setelah penghapusan edge");
    gdg.removeEdge(asal:2, tujuan:1);
    gdg.printGraph();

    for (int i = 0; i < gdg.vertex; i++) {
        System.out.println("Gedung " + (char)('A' + i) + ": inDegree = " + gdg.inDegree(i) + ", outDegree = " + gdg.outDegree(i));
    }
}
```

Hasil:

```
Gedung A: Gedung B (50 m),
Gedung B: Gedung A (60 m), Gedung C (70 m),
Gedung C: Gedung B (80 m), Gedung D (40 m),
Gedung D: Gedung A (90 m),
Gedung A: inDegree = 2, outDegree = 1
Gedung B: inDegree = 2, outDegree = 2
Gedung C: inDegree = 1, outDegree = 2
Gedung D: inDegree = 1, outDegree = 1
Hasil setelah penghapusan edge
Gedung A: Gedung B (50 m),
Gedung B: Gedung A (60 m), Gedung C (70 m),
Gedung C: Gedung D (40 m),
Gedung D: Gedung A (90 m),
Gedung A: inDegree = 2, outDegree = 1
Gedung B: inDegree = 1, outDegree = 2
Gedung C: inDegree = 1, outDegree = 1
Gedung D: inDegree = 1, outDegree = 1
```

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

```
switch (choice) {
    case 1:
        System.out.print(s:"Masukkan node asal: ");
        int asal = scanner.nextInt();
        System.out.print(s:"Masukkan node tujuan: ");
        int tujuan = scanner.nextInt();
        System.out.print(s:"Masukkan jarak: ");
        int jarak = scanner.nextInt();
        gedung.addEdge(asal, tujuan, jarak);
        break;
    case 2:
        System.out.print(s:"Masukkan node asal untuk dihapus edge: ");
        int asalRemove = scanner.nextInt();
        System.out.print(s:"Masukkan node tujuan untuk dihapus edge: ");
        int tujuanRemove = scanner.nextInt();
        gedung.removeEdge(asalRemove, tujuanRemove);
        break;
    case 3:
        System.out.print(s:"Masukkan node untuk diketahui derajatnya: ");
        int nodeDegree = scanner.nextInt();
        gedung.degree(nodeDegree);
        break;
    case 4:
        gedung.printGraph();
        break;
    case 5:
        System.out.print(s:"Masukkan node asal: ");
        int asalCheck = scanner.nextInt();
        System.out.print(s:"Masukkan node tujuan: ");
        int tujuanCheck = scanner.nextInt();
        gedung.isConnected(asalCheck, tujuanCheck);
        break;
    case 0:
        System.out.println(x:"Terima kasih!");
        break;
    default:
        System.out.println(x:"Pilihan tidak valid. Silakan pilih kembali.");
```

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

```
void updateJarak(int asal, int tujuan, int jarakBaru) throws Exception {
    boolean edgeFound = false;
    for (int i = 0; i < list[asal].size(); i++) {
        if (list[asal].get(i) == tujuan) {
            list[asal].getJarak(i);
            edgeFound = true;
            break;
        }
    }
```

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

```
    int hitungEdge() {
    int totalEdge = 0;
    for (int i = 0; i < vertex; i++) {
        totalEdge += list[i].size();
    }
    return totalEdge;
    }
```