Nama : Romy Mahardika Pangestu Lazuardi
No : 25
Kelas : 1H

JOBSHEET XII

Graph

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;
}
```

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++;
}
```

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;
}
```

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;
        }
}
```

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**

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

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

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

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);
```

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());
```

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);
        }
    }
}
```

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");
}
```

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:"");
}
```

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();
```

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.

```
public class Node25 {
    int data;
    int jarak;
    Node25 prev, next;

    Node25(Node25 prev, int data, int jarak, Node25 next) {
        this.prev = prev;
        this.data = data;
        this.jarak = jarak;
        this.next = next;
    }
}
```

```
public class DoubleLinkedList25 {
    Node25 head;
    int size;

    public DoubleLinkedList25() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }
```

```
public void addFirst(int item, int jarak) {
    if (isEmpty()) {
        head = new Node25(null, item, jarak, null);
    } else {
        Node25 newNode = new Node25(null, item, jarak, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}

public int getJarak(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception("Nilai indeks di luar batas");
    }
    Node25 tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.jarak;
}

public void remove(int index) {
    Node25 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;
            }
            size--;
            break;
        }
        current = current.next;
    }
}

public void clear() {
    head = null;
    size = 0;
}

public int size() {
    return size;
```

```
    }

    public int get(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Nilai indeks di luar batas");
        }
        Node25 tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }
}
```

```
public class Graph25 {
    int vertex;
    DoubleLinkedList25 list[];

    public Graph25(int v) {
        vertex = v;
        list = new DoubleLinkedList25[v];
        for (int i = 0; i < v; i++) {
            list[i] = new DoubleLinkedList25();
        }
    }

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

    public void degree(int asal) throws Exception {
        int 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;
                }
            }
        }

        totalOut = list[asal].size();

        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);
    }

    public void removeAllEdges() {
        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();
            }
        }
    }
}
```

```
public class GraphMain25 {
    public static void main(String[] args) throws Exception {
        Graph25 gedung = new Graph25(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();

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

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

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!
Variabel list[] bertipe DoubleLinkedList25 digunakan untuk menyimpan kumpulan node-node yang saling terhubung dalam graf. Setiap indeks dalam list[] merepresentasikan sebuah node dalam graf, dan setiap node tersebut merupakan linked list yang berisi informasi mengenai node-node yang terhubung dengan node tersebut.

3. Jelaskan alur kerja dari method **removeEdge!**
Method removeEdge digunakan untuk menghapus edge (sambungan) antara dua node dalam graf.

Pada implementasinya, method ini menghapus simpul pertama dari linked list yang merepresentasikan node asal, sehingga efektif menghapus sambungan langsung dari node asal ke node tujuan.

4. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain saat digunakan pada method **addEdge** pada class Graph?
Method addFirst() digunakan dalam method addEdge() karena saat menambahkan edge baru, node baru yang merepresentasikan sambungan ke node tujuan harus ditambahkan di awal linked list. Hal ini dilakukan untuk memudahkan akses dan operasi penghapusan nantinya, karena simpul baru menjadi simpul pertama yang terhubung dengan node asal.

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
```

```
import java.util.Scanner;

public void checkPath(Graph25 gedung) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Masukkan gedung asal: ");
    int asal = scanner.nextInt();
    System.out.print("Masukkan gedung tujuan: ");
    int tujuan = scanner.nextInt();

    boolean pathExists = isPathExist(gedung, asal, tujuan);
    if (pathExists) {
        System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan) + "
bertetangga");
    } else {
        System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan) + " tidak
bertetangga");
    }
}

public boolean isPathExist(Graph25 gedung, int asal, int tujuan) {
    for (int i = 0; i < gedung.list[asal].size(); i++) {
        if (gedung.list[asal].get(i) == tujuan) {
            return true;
        }
    }
    return false;
}
```

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;
```

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];
}
```

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;
}
```

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

```
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();
    }
}
```

7. Tambahkan kode berikut pada file **GraphMain.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();
```

8. Commit dan push kode program ke Github

9. Compile dan run program.

```
public class GraphMatriks25 {
    int vertex;
    int[][] matriks;
```

```
    public GraphMatriks25(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] = 0;
            }
        }
    }

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

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

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

```
public class GraphMain25 {
    public static void main(String[] args) {
        GraphMatriks25 gdg = new GraphMatriks25(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);

        System.out.println("Graf sebelum penghapusan edge:");
        gdg.printGraph();

        System.out.println("Hasil setelah penghapusan edge:");
        gdg.removeEdge(2, 1);
        gdg.printGraph();
    }
```

```
}
```

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),
```

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?
Jenis graph yang digunakan adalah "Directed Graph" (graf berarah), di mana setiap edge memiliki arah dari node asal ke node tujuan.

3. Apa maksud dari dua baris kode berikut?

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

Baris pertama: Membuat edge dari node 1 ke node 2 dengan jarak 70.

Baris kedua: Membuat edge dari node 2 ke node 1 dengan jarak 80.

Artinya, node 1 memiliki sambungan ke node 2 dengan jarak 70, dan sebaliknya node 2 memiliki sambungan ke node 1 dengan jarak 80. Ini menunjukkan bahwa graph adalah directed dan edge memiliki bobot yang bisa berbeda untuk arah yang berbeda.

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

Method degree(int asal) menghitung inDegree dan outDegree dari node asal.

inDegree dihitung dengan menjumlahkan semua edge yang menuju node asal.

outDegree dihitung dengan menjumlahkan semua edge yang berangkat dari node asal.

Kemudian, method ini mencetak nilai inDegree, outDegree, dan total degree dari node tersebut.

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

```
import java.util.Scanner;

public class GraphMain25 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        GraphMatriks25 gdg = new GraphMatriks25(4);

        boolean running = true;
        while (running) {
            System.out.println("\nMenu:");
            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. Update Jarak");
            System.out.println("7. Hitung Edge");
            System.out.println("8. Exit");
            System.out.print("Pilih menu: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Masukkan gedung asal: ");
                    int asal = scanner.nextInt();
                    System.out.print("Masukkan gedung tujuan: ");
                    int tujuan = scanner.nextInt();
                    System.out.print("Masukkan jarak: ");
                    int jarak = scanner.nextInt();
                    gdg.makeEdge(asal, tujuan, jarak);
                    break;
                case 2:
                    System.out.print("Masukkan gedung asal: ");
                    asal = scanner.nextInt();
                    System.out.print("Masukkan gedung tujuan: ");
                    tujuan = scanner.nextInt();
                    gdg.removeEdge(asal, tujuan);
                    break;
                case 3:
```

```
                System.out.print("Masukkan gedung: ");
                asal = scanner.nextInt();
                gdg.degree(asal);
                break;
            case 4:
                gdg.printGraph();
                break;
            case 5:
                System.out.print("Masukkan gedung asal: ");
                asal = scanner.nextInt();
                System.out.print("Masukkan gedung tujuan: ");
                tujuan = scanner.nextInt();
                if (gdg.cekEdge(asal, tujuan)) {
                    System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan)
+ " bertetangga");
                } else {
                    System.out.println("Gedung " + (char) ('A' + asal) + " dan Gedung " + (char) ('A' + tujuan)
+ " tidak bertetangga");
                }
                break;
            case 6:
                System.out.print("Masukkan gedung asal: ");
                asal = scanner.nextInt();
                System.out.print("Masukkan gedung tujuan: ");
                tujuan = scanner.nextInt();
                System.out.print("Masukkan jarak baru: ");
                jarak = scanner.nextInt();
                gdg.updateJarak(asal, tujuan, jarak);
                break;
            case 7:
                System.out.println("Jumlah edge dalam graf: " + gdg.hitungEdge());
                break;
            case 8:
                running = false;
                break;
            default:
                System.out.println("Pilihan tidak valid.");
        }
    }
    scanner.close();
  }
}
```

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

```
public void updateJarak(int asal, int tujuan, int jarak) {
  for (int i = 0; i < list[asal].size(); i++) {
```

```
        if (list[asal].get(i) == tujuan) {
            try {
                list[asal].getNode(i).jarak = jarak;
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
            return;
        }
    }
    System.out.println("Edge tidak ditemukan!");
}
```

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

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