LAPORAN HASIL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA

Oleh:
DZULFIKAR MUHAMMAD AL GHIFARI
NIM. 2341760071
SIB-1F / 08
D-IV SISTEM INFORMASI BISNIS
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG

PRAKTIKUM 14

14.2 Percobaan 1

14.2.1 Langkah Langkah percobaan

1. Membuat Class Node

```
public class Node08 {
    int data;
    Node08 prev, next;
    int jarak;

    Node08 (Node08 prev, int data, int jarak, Node08 next) {
        this.prev = prev;
        this.data = data;
        this.next = next;
        this. jarak = jarak;
    }
}
```

2. Menyalin class double linkedlist dan menyesuaikan beberapa method

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void addFirst(int item, int jarak) {
    if (isEmpty()) {
        head = new Node08(prev:null, item, jarak, next:null);
    } else {
        Node08 newNode = new Node08(prev:null, item, jarak, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public int getJarak(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception(message:"Nilai indeks di luar batas");
    }

    Node08 tmp = head;

    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }

    return tmp. jarak;
}
```

```
public void remove(int index) {
    Node08 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;
    }
}
```

3. Membuat class graph

```
public class Graph08 {
    int vertex;
    DoubleLinkedLists08 list[];
}
```

4. Membuat konnstruktor class graph

```
public Graph08(int v) {
    vertex = v;
    list = new DoubleLinkedLists08[v];
    for (int i = 0; i < v; i++) {
        list[i] = new DoubleLinkedLists08();
    }
}
```

5. Menambahkan method addedge

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void addEdge(int asal, int tujuan, int jarak) {
    list[asal].addFirst(tujuan, jarak);
}
```

6. Menambahkan method addFirst

```
public void addFirst(int item) {
    if (isEmpty()) {
        head = new Node08(prev:null, item, next:null);
    } else {
        Node08 newNode = new Node08(prev:null, item, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}
```

7. Menambahkan method degree

```
Codeium: Refactor | Explain | Generate Javadoc | ×
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));
}
```

8. Menambahkan method remove edge

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void removeEdge(int asal, int tujuan) throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (i == tujuan) {
            list[asal].remove(tujuan) ;
        }
    }
}
```

9. Membuat method print graph

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void printGraph() throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (list[i].size() > 0) {
            System.out.println("Gedung " + (char) ('A' + 1) +" terhubung dengan
            ");
            for (int j = 0; j < list[i].size(); j++) {
                System.out.print((char) ('A' + list[1].get(j)) + " (" + list[i].
                getJarak(j) +" m), ");
            }
            System.out.println(x:" ");
        }
    }
    System.out.println(x:" ");
}
```

10. Membuat class main

```
Codeium: Refactor | Explain
public class GraphMain08 {
    Run | Debug | Run main | Debug main | Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) throws Exception {
        Graph08 gedung = new Graph08(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();
    }
}
```

VERIFIKASI HASIL PERCOBAAN 14.2.2

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

dzf@dzf-14ARE05:/media/dzf/D
```

```
erTemuan14_b814d9047bin GraphMain08
InDegree dari Gedung A: 0
OutDegree dari Gedung A: 2
Degree dari Gedung A: 2
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),
```

PERTANYAAN 14.2.3

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!

Untuk menyimpan daftar tetangga (adjacency list) darisetiap simpul (vertex) dalam graf

3. Jelaskan alur kerja dari method removeEdge!

Melakukan iterasi melalui daftar tetangga (list) dari simpul asal.
Pada setiap iterasi, membandingkan indeks (i) dengan simpul tujuan.
Jika indeks (i) sama dengan simpul tujuan, maka menghapus simpul tujuan dari daftar tetangga (list[asal]) menggunakan metode remove.

4. Apakah alasan pemanggilan method addFirst() untuk menambahkan data, bukan method add

jenis lain saat digunakan pada method addEdge pada class Graph?

Kemudahan manipulasi tepi, yaitu menempatkan tepi baru di awal daftar tetangga memudahkan akses dan penelusuran algoritma.

5. Modifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara

suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).

```
public boolean isAdjacent(int asal, int tujuan) {
    for (int i = 0; i < list[asal].size(); i++) {
        try {
            if (list[asal].get(i) == tujuan) {
                System.out.print("Gedung " + (char) ('A' + i) + " bertetanggan
                dengan Gedung " + (char) ('A' + tujuan) + "\n");
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return false;
}
```

```
Scanner scanner = new Scanner(System.in);
int asal, tujuan, jarak;
System.out.print(s:"Masukkan gedung asal : ");
asal = scanner.nextInt();
System.out.print(s:"Masukkan gedung tujuan : ");
tujuan = scanner.nextInt();
gedung.isAdjacent(asal, tujuan);
```

```
Masukkan gedung asal : 1
Masukkan gedung tujuan : 3
Gedung A bertetanggan dengan Gedung D
dzf@dzf-14ARE05:/media/dzf/DATA/1POLINEMA
```

14.3 Percobaan 2

14.3.1 Langkah-langkah Percobaan

1. Membuat class graphMatrik

```
Codeium: Refactor | Explain
public class GraphMatriks08 {
    int vertex;
    int [][] matriks;


    public GraphMatriks08(int v) {
        vertex = v;
        matriks = new int[v][v];
    }
}
```

2. Menambahkan method makeEdge

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void makeEdge(int asal, int tujuan, int jarak) {
    matriks[asal][tujuan] = jarak;
}
```

3. Menambahkan method removeEdge

```
int tujuan
Codeium: Refactor | Explain
public void remove
    int tujuan - GraphMatriks08.removeEdge(int, int)
    matriks[asal][tujuan] = -1;
}
```

4. Menambahkan method print

```
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();
    }
}
```

5. Menambahkan kode pada method main

```
GraphMatriks08 gdg = new GraphMatriks08(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();
}
```

VERIFIKASI HASIL PERCOBAAN 14.3.2

```
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),
```

PERTANYAAN 14.3.3

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

2. Apa jenis graph yang digunakan pada Percobaan 2?

Graf matriks adalah cara untuk merepresentasikan struktur graf menggunakan matriks, yaitu susunan bilangan yang tersusun dalam baris dan kolom.

3. Apa maksud dari dua baris kode berikut?

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

Untuk menambahkan edge diantara dua node dalam graf berbasis matriks adjacency

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

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

Codeium: Refactor | Explain | Generate Javadoc | ×
public int outDegree(int node) {
    int outDegreeCount = 0;
    for (int j = 0; j < vertex; j++) {
        if (matriks[node][j] != 0) {
            outDegreeCount++;
        }
    }
    return outDegreeCount;
}
```

14.4 Percobaan 3

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

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

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

```
Scanner scanner = new Scanner(System.in);
Graph08 graph = new Graph08(vertex:6); // Ganti jumlah vertex ses
kebutuhan

int pil;
do {
    System.out.println(x:"\nMenu :");
    System.out.println(x:"1. Add Edge");
    System.out.println(x:"2. Remove Edge");
    System.out.println(x:"3. Degree");
    System.out.println(x:"4. Print Graph");
    System.out.println(x:"5. Cek Edge");
    System.out.println(x:"6. Update Jarak");
    System.out.println(x:"7. Hitung Edge");
    System.out.println(x:"0. Exit");
    System.out.print(s:"Masukkan pilihan: ");
    pil = scanner.nextInt();

    switch (pil) {
```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public void updateJarak(int asal, int tujuan, int jarakBaru) {
    matriks[asal][tujuan] = jarakBaru;
}
```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
public int hitungEdge() {
    int edgeCount = 0;
    for (int i = 0; i < vertex; i++) {
        for (int j = 0; j < vertex; j++) {
            if (matriks[i][j] != 0) {
                edgeCount++;
            }
        }
    }
    return edgeCount;
}
```

```
5. Cek Edge
6. Update Jarak
7. Hitung Edge
0. Exit
Masukkan pilihan: 1
Masukkan gedung asal: 1
Masukkan gedung tujuan: 2
Masukkan jarak: 60
```

```
5. Cek Edge
6. Update Jarak
7. Hitung Edge
0. Exit
Masukkan pilihan: 4
Gedung B terhubung dengan: D(90m), C(60m),
Gedung C terhubung dengan: B(60m),
Gedung D terhubung dengan: B(90m),

Menu Program:
```

```
4. Print Graph
5. Cek Edge
6. Update Jarak
7. Hitung Edge
0. Exit
Masukkan pilihan: 5
Masukkan gedung asal: 1
Masukkan gedung tujuan: 3
Gedung B dan Gedung D bertetangga
```