LAPORAN HASIL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA
PRAKTIKUM 14

Oleh:

RANDA HERU KUSUMA

NIM. 2341760009

SIB-1F / 25

D-IV SISTEM INFORMASI BISNIS JURUSAN

TEKNOLOGI INFORMASI POLITEKNIK

NEGERI MALANG

Link github: https://github.com/randaheru/Praktikum14.git

2.1.1 Langkah-langkah Percobaan

Waktu percobaan (90 menit)

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

```
class Node25 {
    int data;
    int jarak; // New field for jarak
    Node25 prev;
    Node25 next;

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

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 Node25(prev:null, item, jarak, next:null);
    } else {
        Node25 newNode25 = new Node25(prev:null, item, jarak, head);
        head.prev = newNode25;
        head = newNode25;
    }
    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 || index < 0) {
        throw new Exception(message:"Nilai indeks di luar batas.");
    }

    Node25 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

```
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;
```

C. Class Graph

Buat file baru, beri nama **Graph25.java**

Lengkapi class **Graph** dengan atribut yang telah digambarkan di dalam pada class diagram, yang terdiri dari atribut **vertex** dan **DoubleLinkedList**

```
public class Graph25 {
    int vertex;
    DoubleLinkedLists25[] list;
```

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

```
public Graph25(int v) {
    vertex = v;
    list = new DoubleLinkedLists25[v];
    for (int i = 0; i < v; i++) {
        list[i] = new DoubleLinkedLists25();
    }
}
```

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

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

```java
public void degree(int asal) throws Exception {
    int totalIn = 0, totalOut = 0;

    // Calculate in-degree
    for (int i = 0; i < vertex; i++) {
        for (int j = 0; j < list[i].size(); j++) {
            if (list[i].get(j) == asal) {
                ++totalIn;
            }
        }
    }
```

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

```java
public void removeEdge(int asal, int tujuan) throws Exception {
    list[asal].remove(tujuan);
}
```

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

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

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

```java
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();
        }
```

D. Class Utama

Buat file baru, beri nama **GraphMain25.java**

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

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

```java
Graph25 gedung = new Graph25(v:6);
```

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

```java
Graph25 gedung = new Graph25(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);
```

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

```
gedung.removeEdge(asal:1, tujuan:3);
gedung.printGraph();
```

2.1.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

Hasil running pada langkah 14

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

Hasil running pada langkah 17

```
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),
```

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! Memiliki tujuan utama untuk menyimpan daftar tetangga (adjacency list) dari setiap simpul (vertex) dalam graf
3. Jelaskan alur kerja dari method **removeEdge**!

Metode removeEdge bekerja dengan memanggil metode remove dari DoubleLinkedLists25 untuk menghapus tepi yang mengarah dari simpul asal ke simpul tujuan. Metode remove pada DoubleLinkedLists25 mencari node yang sesuai dan menghapusnya dari linked list dengan memperbarui referensi prev dan next dari node yang berdekatan, serta menyesuaikan ukuran linked list.

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

Dengan menambahkan elemen di awal, setiap tepi baru selalu ditempatkan di posisi yang sama (yaitu, di awal linked list). Ini memberikan kesederhanaan dan konsistensi dalam penanganan elemen baru, membuat implementasi dan debug menjadi lebih mudah

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
```

Menambahkan Metod

```
public boolean isAdjacent(int asal, int tujuan) {
    for (int i = 0; i < list[asal].size(); i++) {
        try {
            if (list[asal].get(i) == tujuan) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return false;
}
```

```
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
```

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 **GraphMatriks25.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 GraphMatriks25(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<NoAbsen>.java** yang sudah dibuat pada

Percobaan 1.

```
import java.util.Scanner;

public class GraphMain25 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) throws Exception {
        // Example usage
        Graph25 gedung = new Graph25vertex:(6);
        gedung.addEdge(0, 1, 50);
        gedung.addEdge(0, 2, 100);
        gedung.addEdge(1, 3, 70);
        gedung.addEdge(2, 3, 40x:);
        gedung.addEdge(3, 4, 60x:);
        gedung.addEdge(4, 5, 80x:);
        x:
        gedung.degree(0);x:
        gedung.printGraph();x:
```

2.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

```
Masukkan gedung asal: 2
Masukkan gedung tujuan: 5
Gedung C dan Gedung F tidak bertetangga
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),
PS C:\Users\ACER\Documents\Graph>
```

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?

Representasi graf menggunakan matriks adjacency. Kelas ini memiliki beberapa metode untuk operasi dasar pada graf berbentuk matriks adjacency, seperti menambah edge, menghapus edge, dan mencetak graf.

3. Apa maksud dari dua baris kode berikut?

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

Dua baris kode tersebut adalah pemanggilan metode makeEdge dari objek gdg yang merupakan instansiasi dari kelas GraphMatriks25. Metode makeEdge digunakan untuk menambahkan edge antara 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;
}

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

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

```
public static void main(String[] args) throws Exception {
    Scanner scanner = new Scanner(System.in);
    Graph25 graph = new Graph25(6); // Ganti jumlah vertex sesuai kebutuhan

    int choice;
    do {
        System.out.println("\nMenu Program:");
        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("0. Exit");
        System.out.print("Masukkan pilihan: ");
        choice = scanner.nextInt();
```

```
Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Cek Edge
6. Update Jarak
7. Hitung Edge
0. Exit
Masukkan pilihan:
```

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!

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

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

```
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;
}
```