Laporan Teori Algoritma dan Struktur Data

Jobsheet 15 - Graph

Dosen Pengampu : Triana Fatmawati, S.T., M.T.

Nama : **Annisa**

Nim : **2341760032**

Kelas : **SIB 1E**

Prodi : **D-IV Sistem Informasi Bisnis**

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

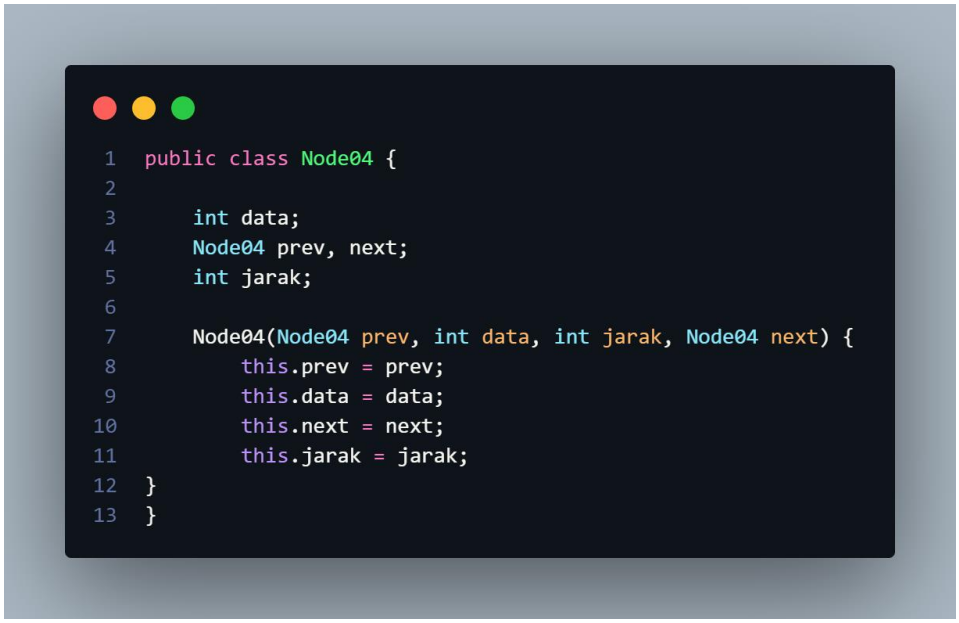
2023/2024

2. Praktikum

2.1 Percobaan 1: Implementasi Graph menggunakan Linked List

2.1.1 Langkah-langkah Percobaan

Berikut merupakan kode program class Node



```java
public class Node04 {

    int data;
    Node04 prev, next;
    int jarak;

    Node04(Node04 prev, int data, int jarak, Node04 next) {
        this.prev = prev;
        this.data = data;
        this.next = next;
        this.jarak = jarak;
}
}
```

Class GraphMain

```java
public class GraphMain04 {

    public static void main(String[] args) throws Exception {
        Graph04 gedung = new Graph04(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();
    }
}
```

Class DoubleLinked List

```java
public class DoubleLinkedLists04 {

        Node04 head;
        int size;

        public DoubleLinkedLists04() {
            head = null;
            size = 0;
        }

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

            public void addFirst (int item, int jarak) {
                if (isEmpty()) {
                    head = new Node04(null, item, jarak,  null);
                }else {
                    Node04 newNode04 = new Node04(null, item, jarak, head);
                    head.prev = newNode04;
                    head = newNode04;
                }
                size++;
                }

                public int getJarak(int index) throws Exception {
                    if (isEmpty() || index >= size) {
                        throw new Exception("Nilai indeks diluar batas.");
                    }
                    Node04 tmp = head;
                    for (int i = 0; i < index; i++) {
                        tmp = tmp.next;
                    }
                    return tmp.jarak;
                }

                public void remove(int index) {
                    Node04 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 int size() {
                    return size;
                }

                public int get(int index) throws Exception {
                    if (isEmpty() || index >= size) {
                        throw new Exception("Nilai indeks di luar batas.");
                    }
                    Node04 tmp = head;
                    for (int i = 0; i < index; i++) {
                        tmp = tmp.next;
                    }
                    return tmp.data;
                    }

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

Class Graph

```java
public class Graph04 {

    int vertex;
    DoubleLinkedLists04[] list;

    public Graph04(int v) {
        vertex = v;
        list = new DoubleLinkedLists04[v];
        for (int i = 0; i < v; i++) {
            list[i] = new DoubleLinkedLists04();
        }
    }

    public void addEdge(int asal, int tujuan, int jarak) {
        list[asal].addFirst(tujuan, jarak);
        //list[tujuan].addFirst(tujuan, 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));
        //System.out.println("Degree dari Gedung " + (char) ('A' + asal) + ": " + list[asal].size());
    }

    public void removeEdge(int asal, int tujuan) throws Exception {
        for(int i = 0; i < vertex; i++) {
            if (i == tujuan) {
                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.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();
            }
        }
        System.out.println();
    }
}

```

2.1.2 Verifikasi Hasil Percobaan

```
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),

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!

Perbaikan kode program yang error yaitu dengan menambahkan size—pada method remove

```
public void remove(int index) {
                    Node04 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;
                    }
                }
```

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

Pada class Graph, atribut list[] yang bertipe DoubleLinkedList digunakan untuk merepresentasikan graf sebagai kumpulan dari beberapa list.

3. Jelaskan alur kerja dari method removeEdge!

Method removeEdge dalam kelas Graph berfungsi untuk menghapus sebuah edge antara dua simpul dalam graf. Alur kerja method ini melibatkan penghapusan entri yang sesuai dari daftar adjacency (linked list) untuk simpul asal (asal). Metode removeEdge bertujuan untuk menghapus sebuah edge (sisi) yang menghubungkan dua node dalam graf.

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

Pemanggilan addFirst dalam metode addEdge dipilih karena alasan efisiensi, kemudahan implementasi, dan konsistensi dalam pengelolaan adjacency list. Dengan addFirst, penambahan edge dilakukan dalam waktu konstan, menjaga performa graf tetap optimal tanpa perlu traversal yang tidak perlu atau penanganan urutan khusus.

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
```

Berikut merupakan hasil modifikasi daro studi kasus di atas yaitu dengan menambahkan kode program pada class graph dan graphmain

Class Graph

```java
public boolean ifTrue(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;
    }
}
```

Class GraphMain

```java
Scanner sc04 = new Scanner(System.in);
        int asal, tujuan;

        System.out.print("Masukkan inputan: ");
        int input = sc04.nextInt();

        for (int i = 0; i < input; i++) {
            System.out.print("Masukkan gedung asal: ");
            asal = sc04.nextInt();
            System.out.print("Masukkan gedung tujuan: ");
            tujuan = sc04.nextInt();
            if (gedung.ifTrue(asal, tujuan)) {
                System.out.println("Gedung " + (char) ('A' + asal) + " dan " + (char) ('A' + tujuan) + " bertetanga");
        }else {
            System.out.println("Gedung " + (char) ('A' + asal) + " dan " + (char) ('A' + tujuan) + " tidak bertetangga");
        }
        System.out.println();
    }
    sc04.close();
}
}
```

Output

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

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

2.2 Percobaan 2: Implementasi Graph menggunakan Matriks

2.2.1 Langkah-langkah Percobaan

Berikut merupakan Class GraphMatriks

```java
public class GraphMatriks04 {
    int vertex;
    int[][] matriks;

    public GraphMatriks04(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();
        }
    }
}
```

Class GraphMain

```java
public class GraphMain04 {

    public static void main(String[] args) {
        GraphMatriks04 gdg = new GraphMatriks04(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();
    }
}
```

2.2.2 Verifikasi Hasil Percobaan

```
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?

Menggunakan graph berarah dan berbobot. Dikarenakan cara pengaturan nilai pada matriks adjacency dalam metode makeEdge di mana nilai jarak (yang merupakan bobot) disimpan pada posisi matriks[asal][tujuan]. Sedangkan dalam metode removeEdge untuk mengindikasikan penghapusan edge juga mendukung bahwa graph ini berarah, karena hanya arah dari asal ke tujuan yang dihapus, bukan sebaliknya

3. **Apa maksud dari dua baris kode berikut?**

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

Maksut dari dua bari tersbeut yaitu pada baris pertama untuk menambahkan sebuah edge dari vertex 1 ke vertex 2 dengan bobot (jarak) 70. Ini berarti ada sebuah koneksi dari node 1 menuju node 2 dengan jarak atau bobot sebesar 70. Sedangkan pada baris kedua di maksudkan untuk menambahkan sebuah edge dari vertex 2 ke vertex 1 dengan bobot (jarak) 80. Ini berarti ada sebuah koneksi dari node 2 menuju node 1 dengan jarak atau bobot sebesar 80.

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

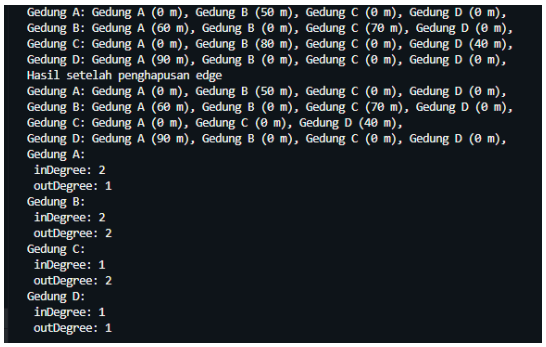
Class GraphMatriks

```
public int outDegree(int vertex) {
        int degree = 0;
        for (int i = 0; i < this.vertex; i++) {
            if (matriks[vertex][i] != 0) {
                degree++;
            }
        }
        return degree;
    }

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

Class GraphMain

```
for (int i = 0; i < 4; i++) {
        System.out.println("Gedung " + (char) ('A' + i) + ":");
        System.out.println(" inDegree: " + gdg.inDegree(i));
        System.out.println(" outDegree: " + gdg.outDegree(i));

    }
```

Output



3. Latihan Praktikum

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!

Berikut merupakan kode program hasil modifikasi di atas

```java
public class GraphMatriks04 {
    int vertex;
    int[][] matriks;

    public GraphMatriks04(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] = 0;
    }

    public boolean hasEgde(int asal, int tujuan) {
        return matriks[asal][tujuan] != 0;
    }

    public void updateJarak(int asal, int tujuan, int jarak) {
        if (matriks[asal][tujuan] != 0) {
            matriks[asal][tujuan] = jarak;
        }else {
            System.out.println("edge tidak ditemukan.");
        }
    }

    public int hitungEdge() {
        int jumlah = 0;
        for (int i = 0; i < vertex; i++) {
            for (int j = 0; j < vertex; j++) {
            if (matriks[i][j] != 0) {
                jumlah++;

            }
        }
    }
    return jumlah;
}

    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();
        }
    }

    public int outDegree(int vertex) {
        int degree = 0;
        for (int i = 0; i < this.vertex; i++) {
            if (matriks[vertex][i] != 0) {
                degree++;
            }
        }
        return degree;
    }

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

```java
import java.util.Scanner;

public class GraphMain04 {

    public static void main(String[] args) throws Exception{
        Scanner sc04 = new Scanner(System.in);
        GraphMatriks04 gdg = new GraphMatriks04(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();

    for (int i = 0; i < 4; i++) {
        System.out.println("Gedung " + (char) ('A' + i) + ":");
        System.out.println(" inDegree: " + gdg.inDegree(i));
        System.out.println(" outDegree: " + gdg.outDegree(i));

    }

    while (true) {
        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. Update Jarak");
        System.out.println("7. Hitung Edge");
        System.out.println("8. Exit");
        System.out.print("Pilih menu: ");
        int menu = sc04.nextInt();

        switch (menu) {
            case 1:
            System.out.print("Masukkan asal: ");
            int asal = sc04.nextInt();
            System.out.print("Masukkan tujuan: ");
            int tujuan = sc04.nextInt();
            System.out.print("Masukkan jarak: ");
            int jarak = sc04.nextInt();
            gdg.makeEdge(asal, tujuan, jarak);
                break;
            case 2:
            System.out.print("Masukkan asal: ");
            asal = sc04.nextInt();
            System.out.print("Masukkan tujuan: ");
            tujuan = sc04.nextInt();
            gdg.removeEdge(asal, tujuan);
            break;
            case 3:
            for (int i = 0; i < 4; i++) {
                System.out.println("Gedung " + (char) ('A' + i) + ":");
                System.out.println(" inDegree: " + gdg.inDegree(i));
                System.out.println(" outDegree: " + gdg.outDegree(i));
            }
            break;
            case 4:
            gdg.printGraph();
            break;
            case 5:
            System.out.print("Masukkan asal: ");
            asal = sc04.nextInt();
            System.out.print("Masukkan tujuan: ");
            tujuan = sc04.nextInt();
            if (gdg.hasEgde(asal, tujuan)) {
                System.out.println("Edge ada.");
            }else {
                System.out.println("Edge tidak ada.");
            }
            break;
            case 6:
            System.out.print("Masukkan asal: ");
            asal = sc04.nextInt();
            System.out.print("Masukkan tujuan: ");
            tujuan = sc04.nextInt();
            System.out.println("Masukkan jarak baru: ");
            jarak = sc04.nextInt();
            gdg.updateJarak(asal, tujuan, jarak);
            break;
            case 7:
            System.out.println("Jumlah edge dalam graf: " + gdg.hitungEdge());
            break;
            case 8:
            sc04.close();
            System.exit(0);
            default:
            System.out.println("Pilihan tidak valid!");
        }
    }
}
}
```

Output

```
Menu:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Cek Edge
6. Update Jarak
7. Hitung Edge
8. Exit
Pilih menu: 4
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),
```

```
Pilih menu: 6
Masukkan asal: 0
Masukkan tujuan: 1
Masukkan jarak baru: 40
```

```
Pilih menu: 4
Gedung A: Gedung A (0 m), Gedung B (40 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),
```

```
Pilih menu: 7
Jumlah edge dalam graf: 5
```