LAPORAN PRAKTIKUM

MATA KULIAH ALGORITMA DAN STRUKTUR DATA

PERTEMUAN 15 : GRAPH

KAYLA RACHMAUDINA SATITI PUTRI

2341760103

D-IV SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI POLITEKNIK

NEGERI MALANG

2024

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

```java
public class Node15 {
    int data;
    Node15 prev, next;
    int jarak;

    Node15(Node15 prev, int data, int jarak, Node15 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

```java
public void addFirst(int item, int jarak) {
    if (isEmpty()) {
        head = new Node15(prev:null, item, jarak, next:null);
    } else {
        Node15 newNode = new Node15(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.

```java
public int getJarak(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception(message:"Nilai indeks di luar batas");
    }
    Node15 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) {
    Node15 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;
DoubleLinkedList15 list[];
```

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

```
public Graph15(int v) {
    vertex = v;
    list = new DoubleLinkedList15[v];
    for (int i = 0; i < v; i++) {
    list[i] =new DoubleLinkedList15();
    }
}
```

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);
}
```

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));
```

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(x:"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.

```
Graph15 gedung = new Graph15(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();
```

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 (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),
```

Hasil running pada langkah 17

```
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!

untuk merepresentasikan adjacency list dari graph tersebut agar lebih dapat menyimpan, mengakses, dan memanipulasi struktur graph dengan efisien dan terorganisir.

3. Jelaskan alur kerja dari method **removeEdge**!

   menghapus node `tujuan` dari adjacency list di `asal` menggunakan method `remove` dari `DoubleLinkedList`.

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

   Penggunaan `addFirst()` dalam `addEdge` pada kelas `Graph` merupakan pilihan yang **efisien** dan **sesuai** dengan struktur data dan semantik graph, memungkinkan penambahan tepi baru dengan cara yang mudah diakses dan dikelola.

5. Modifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).

```java
public boolean adjacent(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) + " dan " + (char) ('A' + tujuan) + " bertetangga " + "\n");
                return true;
            } else {
                System.out.print("Gedung "+ (char) ('A' + i) + " dan " + (char) ('A' + tujuan) + " tidak bertetangga " + "\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return false;
}
```

```java
Scanner sc = new Scanner(System.in);
int asal, tujuan, jarak;
System. out.print( "Masukkan gedung asal : ");
asal = sc.nextInt();
System. out.print ( "Masukkan gedung tujuan : ");
tujuan = sc.nextInt();
gedung.adjacent(asal, tujuan);
```

```
Masukkan gedung asal: 2
Masukkan gedung tujuan: 3
Gedung A dan D bertetangga
```

```
Masukkan gedung asal: 2
Masukkan gedung tujuan: 5
Gedung A dan 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 **GraphMatriks<NoAbsen>.java**
2. Lengkapi class **GraphMatriks** dengan atribut **vertex** dan **matriks**

```
public class GraphMatriks15 {
    int vertex;
    int[][] matriks;
```

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

```
    public GraphMatriks15(int v) {
        vertex = v;
        matriks = new int[v][v];
    }
```

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

```
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.

```
GraphMatriks15 gdg = new GraphMatriks15(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();
```

2.2.2 Verifikasi Hasil Percobaan

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

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?

   Graf matriks yang merupakan representasi dari sebuah graf yang menggunakan matriks.
3. Apa maksud dari dua baris kode berikut?

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

   Kode tersebut menambahkan dua sisi berarah pada graf, satu dari simpul 1 ke simpul 2 dengan bobot 70, dan satu lagi dari simpul 2 ke simpul 1 dengan bobot 80.
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

   Pengguna dapat memilih menu program melalui input Scanner

```java
while (true) {
    System.out.println();
    System.out.println("------------------------");
    System.out.println("|         MENU         |");
    System.out.println("------------------------");
    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("------------------------");
    System.out.print("Pilih Menu : ");
    int choice = sc.nextInt();

    switch (choice) {
        case 1:
            System.out.println();
            System.out.print("Masukkan gedung asal  : ");
            int asal = sc.nextInt();
            System.out.print("Masukkan gedung tujuan: ");
            int tujuan = sc.nextInt();
            System.out.print("Masukkan jarak        : ");
            int jarak = sc.nextInt();
            gedung.addEdge(asal, tujuan, jarak);
            break;
        case 2:
            System.out.println();
            System.out.print("Masukkan gedung asal  : ");
            asal = sc.nextInt();
            System.out.print("Masukkan gedung tujuan: ");
            tujuan = sc.nextInt();
            gedung.removeEdge(asal, tujuan);
            break;
        case 3:
            System.out.println();
            System.out.print("Masukkan gedung yang akan dicek derajatnya: ");
            asal = sc.nextInt();
            gedung.degree(asal);
            break;
        case 4:
            System.out.println();
            gedung.printGraph();
            break;
        case 5:
            System.out.println();
            System.out.print("Masukkan gedung asal  : ");
            asal = sc.nextInt();
            System.out.print("Masukkan gedung tujuan: ");
            tujuan = sc.nextInt();
            if (gedung.adjacent(asal, tujuan)) {
                System.out.println("Edge exists.");
            } else {
                System.out.println("Edge doesn't exist.");
            }
            break;
```

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

```
-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 1

Masukkan gedung asal  : 2
Masukkan gedung tujuan: 1
Masukkan jarak        : 80

-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 2

Masukkan gedung asal  : 1
Masukkan gedung tujuan: 0
```

```
-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 4

Gedung A terhubung dengan B (50m),
Gedung B terhubung dengan C (70m),
Gedung C terhubung dengan B (80m),


-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 3

Masukkan gedung yang akan dicek derajatnya: 0
InDegree dari Gedung A: 0
OutDegree dari Gedung A: 1
Degree dari Gedung A: 1
```

```
-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 5

Masukkan gedung asal  : 2
Masukkan gedung tujuan: 1
Gedung A dan B bertetangga
Edge exists.

-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 6

Masukkan gedung asal  : 0
Masukkan gedung tujuan: 1
Masukkan jarak baru   : 40
```

```
-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 7

Total edge dalam graf : 3

-----------------------
|        MENU         |
-----------------------
| 1. | Add Edge       |
| 2. | Remove Edge    |
| 3. | Degree         |
| 4. | Print Graph    |
| 5. | Cek Edge       |
| 6. | Update Jarak   |
| 7. | Hitung Edge    |
| 8. | Keluar         |
-----------------------
Pilih Menu : 8

Keluar dari program.
```