JOBSHEET XII

Graph

Name: Rizqi Bagus Andrean

Kelas: TI-1D

Absen: 25

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

```
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.

2.1.2 Verifikasi Hasil Percobaan

No 14

```
iles\Zulu\zulu-11\bin\java.exe' '-cp' 'C:\Users\Acer\AppData
spaceStorage\68b1dac99ad985332ca5f8e419bd84d0\redhat.java\jd
f2\bin' 'percobaan1.Graph'
Indegree dari Gedung  A : 0
Outdegree dari Gedung A : 2
degree dari Gedung A : 2
Gedunnf A terhubung dengan: C (100m),
B (50m),

Gedunnf B terhubung dengan: D (70m),

Gedunnf C terhubung dengan: D (40m),

Gedunnf D terhubung dengan: E (60m),

Gedunnf E terhubung dengan: F (80m),


PS C:\Users\Acer\Tugas Kuliah\Semester 2\Praktek Algoritma\j
```

No 17

```
Files\Zulu\zulu-11\bin\java.exe  -cp  C:\Users\Acer\AppData
spaceStorage\68b1dac99ad985332ca5f8e419bd84d0\redhat.java\jd
f2\bin' 'percobaan1.Graph'
Indegree dari Gedung  A : 0
Outdegree dari Gedung A : 2
degree dari Gedung A : 2
Gedunnf A terhubung dengan: C (100m),
B (50m),

Gedunnf C terhubung dengan: D (40m),

Gedunnf D terhubung dengan: E (60m),

Gedunnf E terhubung dengan: F (80m),
```

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),
```

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!

   Atribut `list[]` bertipe `DLL[]` dalam kelas `Graph` digunakan untuk menyimpan daftar adjacency dari setiap vertex dalam graf. Ini memungkinkan representasi graf yang efisien dan memfasilitasi operasi seperti penambahan, penghapusan, dan pencarian edge di graf.
3. Jelaskan alur kerja dari method **removeEdge**!
   Fungsi ini memiliki 2 parameter asal dan tujuan, lalu melakukan perulangan dari 0 sampai jumlah vertexnya. Lalu disetiap perulangan memeriksa apakah i sama dengan asal yang mau dihapus, jika iya maka list tujuan dengan asal tersebut akan dihapus, dengan memanhggil fungsi remove yang ada di DLL.
4. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain saat digunakan pada method **addEdge** pada class Graph?

   Metode `addFirst()` digunakan dalam metode `addEdge` pada kelas `Graph` karena menyediakan cara yang efisien dan sederhana untuk menambahkan elemen ke adjacency list. Operasi ini cepat, tidak bergantung pada ukuran list, dan mengurangi kompleksitas implementasi tanpa mempengaruhi fungsi utama graf dalam hal representasi dan manipulasi adjacency list.
5. Modifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).
   Tambahkan fungsi berikut

```
boolean isPathExist(int source, int destination) {
        boolean[] visited = new boolean[vertex];
        return dfs(source, destination, visited);
    }

    // Metode DFS rekursif untuk mengecek jalur
    private boolean dfs(int current, int destination, boolean[] visited) {
        // Tandai node saat ini sebagai telah dikunjungi
        visited[current] = true;

        // Jika kita sudah sampai di tujuan, kembalikan true
        if (current == destination) {
            return true;
        }

        // Iterasi melalui tetangga dari node saat ini
```

```
        for (int i = 0; i < list[current].size(); i++) {
            int neighbor = list[current].get(i);

            // Jika tetangga belum dikunjungi, lakukan DFS pada tetangga tersebut
            if (!visited[neighbor]) {
                if (dfs(neighbor, destination, visited)) {
                    return true;
                }
            }
        }

        // Jika semua tetangga sudah dijelajahi dan tidak menemukan tujuan, kembalikan
false
        return false;
    }
```

```
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
```

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<NoAbsen>.java** yang sudah dibuat pada Percobaan 1.

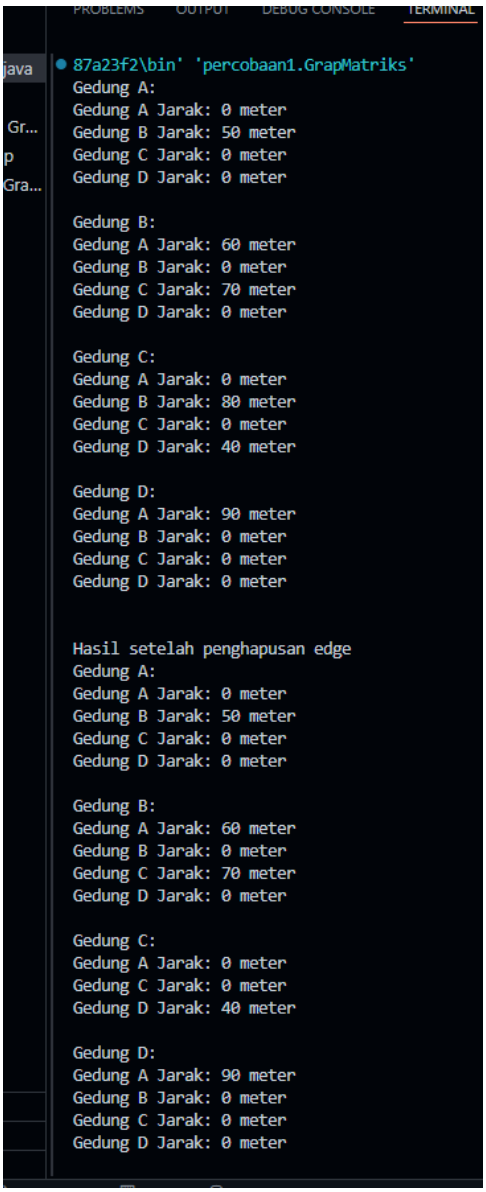
```
GraphMatriks gdg = new GraphMatriks(4);
gdg.makeEdge(0, 1, 50);
gdg.makeEdge(1, 0, 60);
gdg.makeEdge(1, 2, 70);
gdg.makeEdge(2, 1, 80);
gdg.makeEdge(2, 3, 40);
gdg.makeEdge(3, 0, 90);
gdg.printGraph();
System.out.println("Hasil setelah penghapusan edge");
gdg.removeEdge(2, 1);
gdg.printGraph();
```

8. **Commit dan push kode program ke Github**
9. Compile dan run program.

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),
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
87a23f2\bin' 'percobaan1.GrapMatriks'
Gedung A:
Gedung A Jarak: 0 meter
Gedung B Jarak: 50 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 0 meter
Gedung B:
Gedung A Jarak: 60 meter
Gedung B Jarak: 0 meter
Gedung C Jarak: 70 meter
Gedung D Jarak: 0 meter
Gedung C:
Gedung A Jarak: 0 meter
Gedung B Jarak: 80 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 40 meter
Gedung D:
Gedung A Jarak: 90 meter
Gedung B Jarak: 0 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 0 meter
Hasil setelah penghapusan edge
Gedung A:
Gedung A Jarak: 0 meter
Gedung B Jarak: 50 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 0 meter
Gedung B:
Gedung A Jarak: 60 meter
Gedung B Jarak: 0 meter
Gedung C Jarak: 70 meter
Gedung D Jarak: 0 meter
Gedung C:
Gedung A Jarak: 0 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 40 meter
Gedung D:
Gedung A Jarak: 90 meter
Gedung B Jarak: 0 meter
Gedung C Jarak: 0 meter
Gedung D Jarak: 0 meter

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?

   Directed graph
3. Apa maksud dari dua baris kode berikut?

   Menambahkan edge dari 1 ke 2 dengan bobot 70

   Menambahkan edge dari 2 ke 1 dengan bobot 80

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

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

```
3.
4.      int calculateInDegree(int vertex) {
5.          int inDegree = 0;
6.          for (int i = 0; i < this.vertex; i++) {
7.              if (matriks[i][vertex] != -1) {
8.                  inDegree++;
9.              }
10.         }
11.         return inDegree;
12.     }
13.
14.     int calculateOutDegree(int vertex) {
15.         int outDegree = 0;
16.         for (int i = 0; i < this.vertex; i++) {
17.             if (matriks[vertex][i] != -1) {
18.                 outDegree++;
19.             }
20.         }
21.         return outDegree;
22.     }
23.     int calculateTotalDegree(int vertex) {
24.         return calculateInDegree(vertex) + calculateOutDegree(vertex);
25.     }
26.
```

```
    void printDegrees() {
        for (int i = 0; i < vertex; i++) {
            int inDegree = calculateInDegree(i);
            int outDegree = calculateOutDegree(i);
            System.out.println("Gedung " + (char) ('A' + i) + " -> InDegree: " + inDegree + ", OutDegree: " + outDegree + ", Total Degree: " + (inDegree + outDegree));
        }
    }

```

```
Gedung A -> InDegree: 4, OutDegree: 4, Total Degree: 8
Gedung B -> InDegree: 3, OutDegree: 4, Total Degree: 7
Gedung C -> InDegree: 4, OutDegree: 3, Total Degree: 7
Gedung D -> InDegree: 4, OutDegree: 4, Total Degree: 8
PS C:\Users\Acer\Tugas Kuliah\Semester 2\Praktek Algoritma\jobsheet 1
```

35. 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

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        GraphModification graph = new GraphModification(6);
        boolean exit = false;

        while (!exit) {
            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. Check Edge Existence");
            System.out.println("6. Exit");
            System.out.print("Pilih menu (1-6): ");

            int choice = scanner.nextInt();
            int source, destination, jarak;
```

```
            switch (choice) {
                case 1:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    System.out.print("Masukkan jarak: ");
                    jarak = scanner.nextInt();
                    graph.addEdge(source, destination, jarak);
                    break;
                case 2:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    try {
                        graph.removeEdge(source, destination);
                    } catch (Exception e) {
                        System.out.println("Edge tidak ditemukan.");
                    }
                    break;
                case 3:
                    System.out.print("Masukkan vertex untuk diperiksa degree-nya (0-
5): ");
                    source = scanner.nextInt();
                    try {
                        graph.degree(source);
                    } catch (Exception e) {
                        System.out.println("Vertex tidak valid.");
                    }
                    break;
                case 4:
                    try {
                        graph.printGraph();
                    } catch (Exception e) {
                        System.out.println("Error dalam mencetak graph.");
                    }
                    break;
                case 5:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    boolean pathExists = graph.isPathExist(source, destination);
                    System.out.println("Apakah ada jalur dari " + (char) ('A' +
source) + " ke " + (char) ('A' + destination) + ": " + pathExists);
                    break;
                case 6:
                    exit = true;
                    System.out.println("Program berakhir.");
                    break;
                default:
                    System.out.println("Pilihan tidak valid. Silakan pilih 1-6.");
```

```
            }
        }
        scanner.close();
    }
```

```
Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Exit
Pilih menu (1-6): 4
Gedung A terhubung dengan: B (20m),


Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Exit
Pilih menu (1-6): 1
Masukkan source vertex (0-5): 2
Masukkan destination vertex (0-5): 1
Masukkan jarak: 30

Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Exit
Pilih menu (1-6): 4
Gedung A terhubung dengan: B (20m),
Gedung C terhubung dengan: B (30m),


Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Exit
Pilih menu (1-6):
```

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

   Tambahkan method setJarak di DLL

```
  void setJarak(int index, int newJarak) throws Exception {
      if (isEmpty() || index >= size) {
          throw new Exception("Index out of bound");
```

```
        } else {
            Node current = head;
            for (int i = 0; i < index; i++) {
                current = current.next;
            }
            current.jarak = newJarak;
        }
    }
```

Tambahkan method update jarak di Graph

```
    void updateJarak(int source, int destination, int newJarak) throws Exception {
        for (int i = 0; i < list[source].size(); i++) {
            if (list[source].get(i) == destination) {
                list[source].setJarak(i, newJarak);
                return;
            }
        }
        throw new Exception("Edge tidak ditemukan.");
    }
```

Tambahkan case baru di switch main

```
case 6:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    System.out.print("Masukkan jarak baru: ");
                    newJarak = scanner.nextInt();
                    try {
                        graph.updateJarak(source, destination, newJarak);
                    } catch (Exception e) {
                        System.out.println("Update jarak gagal: " + e.getMessage());
                    }
                    break;
```

```
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Update Jarak
7. Exit
Pilih menu (1-7): 4
Gedung A terhubung dengan: B (20m),
Gedung C terhubung dengan: D (40m),


Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Update Jarak
7. Exit
Pilih menu (1-7): 6
Masukkan source vertex (0-5): 0
Masukkan destination vertex (0-5): 1
Masukkan jarak baru: 30

Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Update Jarak
7. Exit
Pilih menu (1-7): 4
Gedung A terhubung dengan: B (30m),
Gedung C terhubung dengan: D (40m),
```

3. Tambahkan method **hitungEdge** untuk menghitung banyaknya edge yang terdapat di dalam graf!
   Full Source Code Graph (include hitung edge)

```java
package percobaan1;

import java.util.Scanner;

public class GraphModification {
    int vertex;
    DLL list[];

    GraphModification(int vertex) {
        this.vertex = vertex;
        list = new DLL[vertex];
        for (int i = 0; i < vertex; i++) {
            list[i] = new DLL();
        }
    }

    void addEdge(int source, int destination, int jarak) {
        // Directed
```

```
        list[source].addFirst(destination, jarak);
        // Undirected (optional)
        // list[destination].addFirst(source, jarak);
    }

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

        // InDegree
        for (int i = 0; i < vertex; i++) {
            for (int j = 0; j < list[i].size(); j++) {
                if (list[i].get(j) == asal) {
                    ++totalOut;
                }
            }
        }

        // OutDegree
        int k = list[asal].size();
        totalIn = 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));
    }

    void removeEdge(int source, int destination) throws Exception {
        list[source].remove(destination);
    }

    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("");
            }
        }
        System.out.println("");
    }

    boolean isPathExist(int source, int destination) {
        boolean[] visited = new boolean[vertex];
        return dfs(source, destination, visited);
    }
```

```
    private boolean dfs(int current, int destination, boolean[] visited) {
        visited[current] = true;

        if (current == destination) {
            return true;
        }

        for (int i = 0; i < list[current].size(); i++) {
            int neighbor = list[current].get(i);
            if (!visited[neighbor]) {
                if (dfs(neighbor, destination, visited)) {
                    return true;
                }
            }
        }
        return false;
    }


    void updateJarak(int source, int destination, int newJarak) throws Exception {
        for (int i = 0; i < list[source].size(); i++) {
            if (list[source].get(i) == destination) {
                list[source].setJarak(i, newJarak);
                return;
            }
        }
        throw new Exception("Edge tidak ditemukan.");
    }

    void hitungEdge() {
        int totalEdge = 0;
        for (int i = 0; i < vertex; i++) {
            totalEdge += list[i].size();
        }
        // Jika graph undirected, hitungan edge harus dibagi dua
        System.out.println("Total edge: " + totalEdge);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        GraphModification graph = new GraphModification(6);
        boolean exit = false;

        while (!exit) {
            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. Check Edge Existence");
            System.out.println("6. Update Jarak");
            System.out.println("7. Hitung Edge");
            System.out.println("8. Exit");
```

```
            System.out.print("Pilih menu (1-7): ");

            int choice = scanner.nextInt();
            int source, destination, jarak, newJarak;

            switch (choice) {
                case 1:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    System.out.print("Masukkan jarak: ");
                    jarak = scanner.nextInt();
                    graph.addEdge(source, destination, jarak);
                    break;
                case 2:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    try {
                        graph.removeEdge(source, destination);
                    } catch (Exception e) {
                        System.out.println("Edge tidak ditemukan.");
                    }
                    break;
                case 3:
                    System.out.print("Masukkan vertex untuk diperiksa degree-nya (0-
5): ");
                    source = scanner.nextInt();
                    try {
                        graph.degree(source);
                    } catch (Exception e) {
                        System.out.println("Vertex tidak valid.");
                    }
                    break;
                case 4:
                    try {
                        graph.printGraph();
                    } catch (Exception e) {
                        System.out.println("Error dalam mencetak graph.");
                    }
                    break;
                case 5:
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    boolean pathExists = graph.isPathExist(source, destination);
                    System.out.println("Apakah ada jalur dari " + (char) ('A' +
source) + " ke " + (char) ('A' + destination) + ": " + pathExists);
                    break;
                case 6:
```

```
                    System.out.print("Masukkan source vertex (0-5): ");
                    source = scanner.nextInt();
                    System.out.print("Masukkan destination vertex (0-5): ");
                    destination = scanner.nextInt();
                    System.out.print("Masukkan jarak baru: ");
                    newJarak = scanner.nextInt();
                    try {
                        graph.updateJarak(source, destination, newJarak);
                    } catch (Exception e) {
                        System.out.println("Update jarak gagal: " + e.getMessage());
                    }
                    break;
                case 7:
                    graph.hitungEdge();
                    break;
                case 8:
                    exit = true;
                    System.out.println("Program berakhir.");
                    break;
                default:
                    System.out.println("Pilihan tidak valid. Silakan pilih 1-7.");
            }
        }

        scanner.close();
    }
}
```

```
Masukkan jarak: 80

Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Update Jarak
7. Hitung Edge
8. Exit
Pilih menu (1-7): 4
Gedung A terhubung dengan: B (10m),
Gedung B terhubung dengan: C (40m),
Gedung C terhubung dengan: H (80m),


Menu Program:
1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Check Edge Existence
6. Update Jarak
7. Hitung Edge
8. Exit
Pilih menu (1-7): 7
Total edge: 3
```