**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

# Praktikum

* 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 |

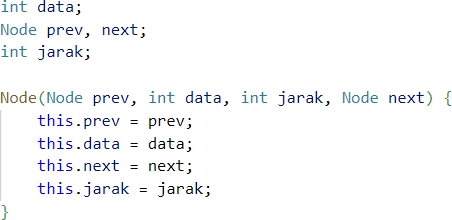
# Langkah-langkah Percobaan Waktu percobaan (90 menit)

* + - 1. Buka text editor. Buat class **Node<NoAbsen>.java** dan class

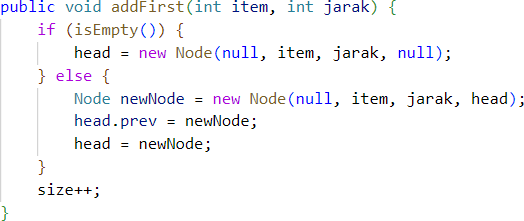
**DoubleLinkedList<NoAbsen>.java** sesuai dengan praktikum **Double Linked List.**

# 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



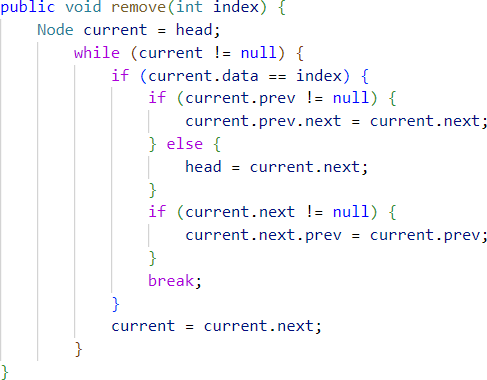
# Class DoubleLinkedList

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

Selanjutnya buat method **getJarak** (hampir sama seperti method **get**) yang digunakan untuk mendapatkan nilai jarak edge antara dua node.



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.

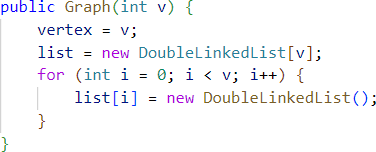


# Class Graph

* + - 1. Buat file baru, beri nama **Graph<NoAbsen>.java**
      2. Lengkapi class **Graph** dengan atribut yang telah digambarkan di dalam pada class diagram, yang terdiri dari atribut **vertex** dan **DoubleLinkedList**



* + - 1. Tambahkan konstruktor default untuk menginisialisasi variabel **vertex** dan menambahkan perulangan jumlah vertex sesuai dengan panjang array yang telah ditentukan.



* + - 1. Tambahkan method **addEdge()** untuk menghubungkan dua node. Baris kode program berikut digunakan untuk graf berarah (directed).

Apabila graf yang dibuat adalah undirected graph, maka tambahkan kode berikut.



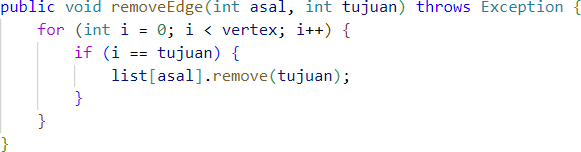
* + - 1. Tambahkan method **degree()** untuk menampilkan jumlah derajat lintasan pada setiap vertex. Kode program berikut digunakan untuk menghitung degree pada graf berarah

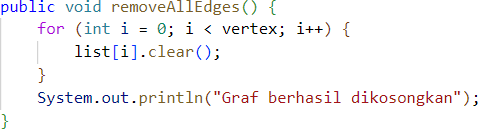


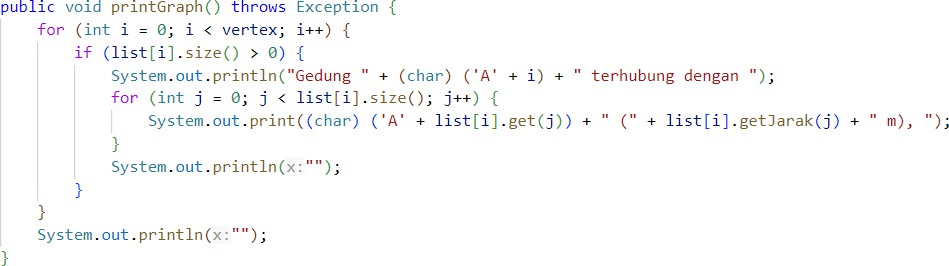
Apabila graf yang dibuat adalah undirected graph, maka cukup gunakan kode berikut.



* + - 1. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graph. Penghapusan membutuhkan 2 parameter yaitu node **asal** dan **tujuan**.

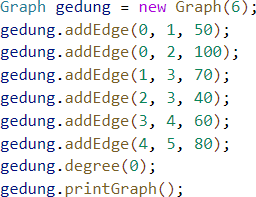


* + - 1. Tambahkan method **removeAllEdges()** untuk menghapus semua vertex yang ada di dalam graf.
      2. Tambahkan method **printGraph()** untuk mencetak graf.



# D. Class Utama

* + - 1. Buat file baru, beri nama **GraphMain<NoAbsen>.java**
      2. Tuliskan struktur dasar bahasa pemrograman Java yang terdiri dari fungsi **main**
      3. Di dalam fungsi **main**, lakukan instansiasi object Graph bernama **gedung** dengan nilai parameternya adalah 6.
      4. Tambahkan beberapa edge pada graf, tampilkan degree salah satu node, kemudian tampilkan grafnya.



* + - 1. Compile dan run program.

Catatan: Degree harus disesuaikan dengan jenis graf yang digunakan. Pada kasus ini, digunakan directed weighted graph

* + - 1. Tambahkan pemanggilan method **removeEdge()**, kemudian tampilkan kembali graf tersebut.

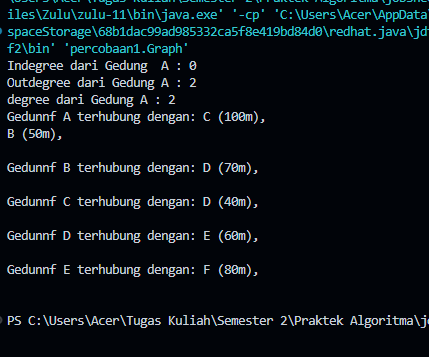


# Commit dan push kode program ke Github

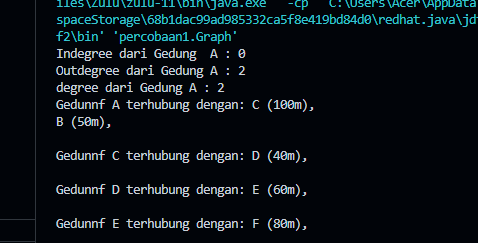
* + - 1. Compile dan run program.

# Verifikasi Hasil Percobaan

# No 14



No 17



Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

|  |  |
| --- | --- |
| **Hasil running pada langkah 14** | **Hasil running pada langkah 17** |
| A screenshot of a computer  Description automatically generated |  |

# 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.

1. 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.

1. 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.

1. 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;

    }

A close up of black text  Description automatically generatedA close up of black text  Description automatically generated

# Percobaan 2: Implementasi Graph menggunakan Matriks

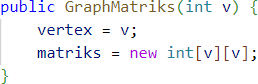
Dengan menggunakan kasus yang sama dengan Percobaan 1, pada percobaan ini implementasi graf dilakukan dengan menggunakan matriks dua dimensi.

# 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**



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



1. Untuk membuat suatu lintasan yang menghubungkan dua node, maka dibuat method

**makeEdge()** sebagai berikut.

A black and blue text  Description automatically generated

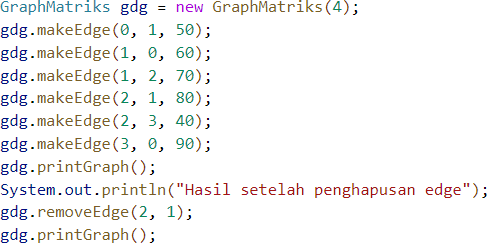
1. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graf.

A white background with black and blue text  Description automatically generated

1. Tambahkan method printGraph() untuk mencetak graf.



1. Tambahkan kode berikut pada file **GraphMain<NoAbsen>.java** yang sudah dibuat pada Percobaan 1.

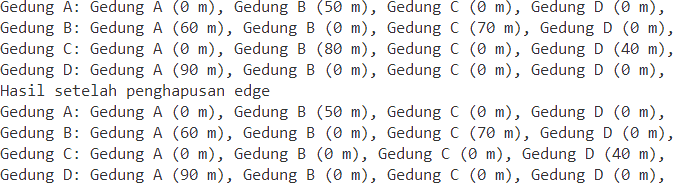


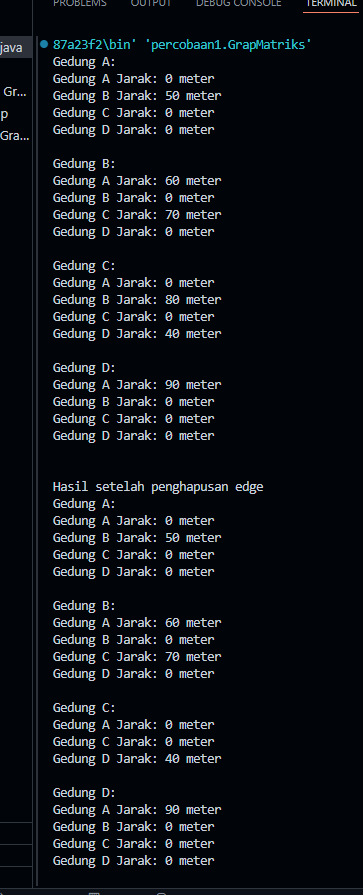
# Commit dan push kode program ke Github

1. Compile dan run program.

# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.





# 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

* + - 1. Apa maksud dari dua baris kode berikut?

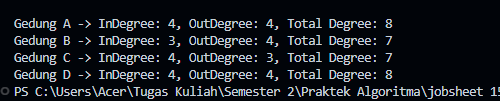
Menambahkan edge dari 1 ke 2 dengan bobot 70

Menambahkan edge dari 2 ke 1 dengan bobot 80



* + - 1. Modifikasi kode program sehingga terdapat method untuk menghitung degree, termasuk inDegree dan outDegree!

1. int calculateInDegree(int vertex) {
2. int inDegree = 0;
3. for (int i = 0; i < this.vertex; i++) {
4. if (matriks[i][vertex] != -1) {
5. inDegree++;
6. }
7. }
8. return inDegree;
9. }
10. int calculateOutDegree(int vertex) {
11. int outDegree = 0;
12. for (int i = 0; i < this.vertex; i++) {
13. if (matriks[vertex][i] != -1) {
14. outDegree++;
15. }
16. }
17. return outDegree;
18. }
19. int calculateTotalDegree(int vertex) {
20. return calculateInDegree(vertex) + calculateOutDegree(vertex);
21. }
22. void printDegrees() {
23. for (int i = 0; i < vertex; i++) {
24. int inDegree = calculateInDegree(i);
25. int outDegree = calculateOutDegree(i);
26. System.out.println("Gedung " + (char) ('A' + i) + " -> InDegree: " + inDegree + ", OutDegree: " + outDegree + ", Total Degree: " + (inDegree + outDegree));
27. }
28. }



# Latihan Praktikum

**Waktu percobaan: 90 menit**

1. Modifikasi kode program pada class **GraphMain** sehingga terdapat menu program yang bersifat dinamis, setidaknya terdiri dari:
   1. Add Edge
   2. Remove Edge
   3. Degree
   4. Print Graph
   5. 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:

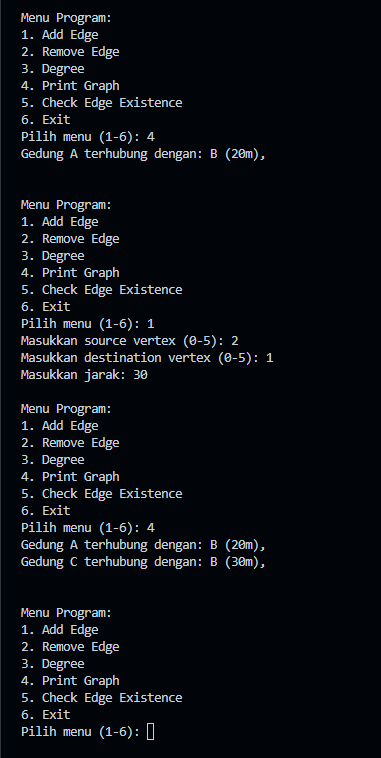
                    System.out.println("Pilihan tidak valid. Silakan pilih 1-6.");

            }

        }

        scanner.close();

    }



1. 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 {

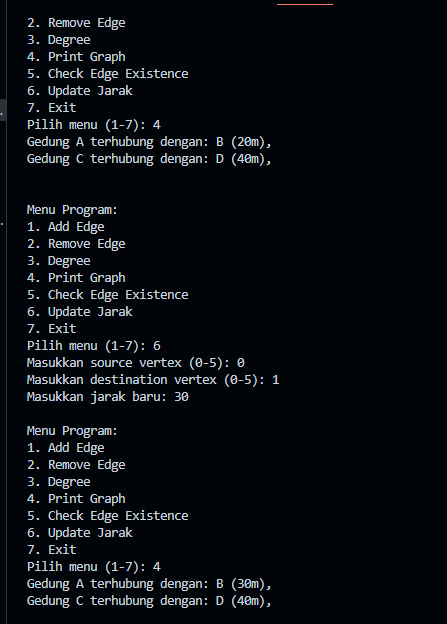
                        graph.updateJarak(source, destination, newJarak);

                    } catch (Exception e) {

                        System.out.println("Update jarak gagal: " + e.getMessage());

                    }

                    break;



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

Full Source Code Graph (include hitung edge)

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();

    }

}

