

SKRIPSI

MODULARISASI ALGORITMA SHORTEST PATH PADA PERANGKAT LUNAK KIRI MENGGUNAKAN STRATEGY PATTERN



Muhammad Aldi Rivandi

NPM: 6182001029

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2025

UNDERGRADUATE THESIS

**MODULARIZATION OF THE SHORTEST PATH ALGORITHM
ON KIRI SOFTWARE USING STRATEGY PATTERN**



Muhammad Aldi Rivandi

NPM: 6182001029

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2025**

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	5
2.1 KIRI [1]	5
2.2 Design Pattern dan Strategy Pattern [2]	6
2.2.1 Contoh Kode Program	8
2.3 MySQL [3]	9
2.3.1 LineString [3]	10
2.4 Graf [4]	11
2.4.1 Graf Berarah [4]	12
2.4.2 Graf Tidak Berarah [4]	13
2.4.3 Graf Berbobot [4]	13
2.4.4 Graf Tidak Berbobot [4]	14
2.5 Algoritma Shortest Path	15
2.5.1 Algoritma Dijkstra [5]	15
2.5.2 Algoritma Floyd-Warshall [5]	18
2.5.3 Algoritma A* [6]	20
3 ANALISIS	23
3.1 Analisis Sistem Kini	23
3.1.1 Analisis Kelas	24
3.1.1.1 Main.java	24
3.1.1.2 AdminListener	25
3.1.1.3 NewMenjanganServer	26
3.1.1.4 ServiceListener	27
3.1.1.5 Worker	27
3.1.1.6 DataPuller	30
3.1.1.7 DataPuller.RouteResult	30
3.1.1.8 DataPullerException	30
3.1.1.9 LatLon	31
3.1.1.10 UnrolledLinkedList	32
3.1.1.11 UnrolledLinkedList.FastLinkedListIterator	32
3.1.1.12 Track	33

3.1.1.13	GraphEdge	34
3.1.1.14	GraphNode	35
3.1.1.15	Graph	36
3.1.1.16	Dijkstra	37
3.1.1.17	Dijkstra.NodeInfo	38
3.1.2	Pemodelan Graf	38
3.2	Analisis Sistem Usulan	39
3.2.1	Implementasi Strategy Pattern	40
3.2.2	Implementasi Algoritma A* dan Algoritma Floyd-Warshall	40
4	PERANCANGAN	41
4.1	Rancangan Diagram Kelas	41
DAFTAR REFERENSI		43
A	KODE PROGRAM	45
B	HASIL EKSPERIMEN	47

DAFTAR GAMBAR

1.1	Tampilan halaman perangkat lunak KIRI	1
1.2	Tampilan perangkat lunak KIRI, setelah menerima masukan	2
2.1	Tampilan awal perangkat lunak KIRI	5
2.2	Tampilan perangkat lunak KIRI, setelah menerima masukan	6
2.3	Struktur Strategy Pattern	7
2.4	Graf	12
2.5	Graf Berarah	12
2.6	Graf Tidak Berarah	13
2.7	Graf Berbobot	14
2.8	Graf Tidak Berbobot	14
3.1	Struktur Kelas NewMenjangan	23
3.2	Rute Ciroyom – Cicaheum	39
3.3	Struktur Strategy Pattern	40
4.1	Rancangan Diagram Kelas	41
B.1	Hasil 1	47
B.2	Hasil 2	47
B.3	Hasil 3	47
B.4	Hasil 4	47

1

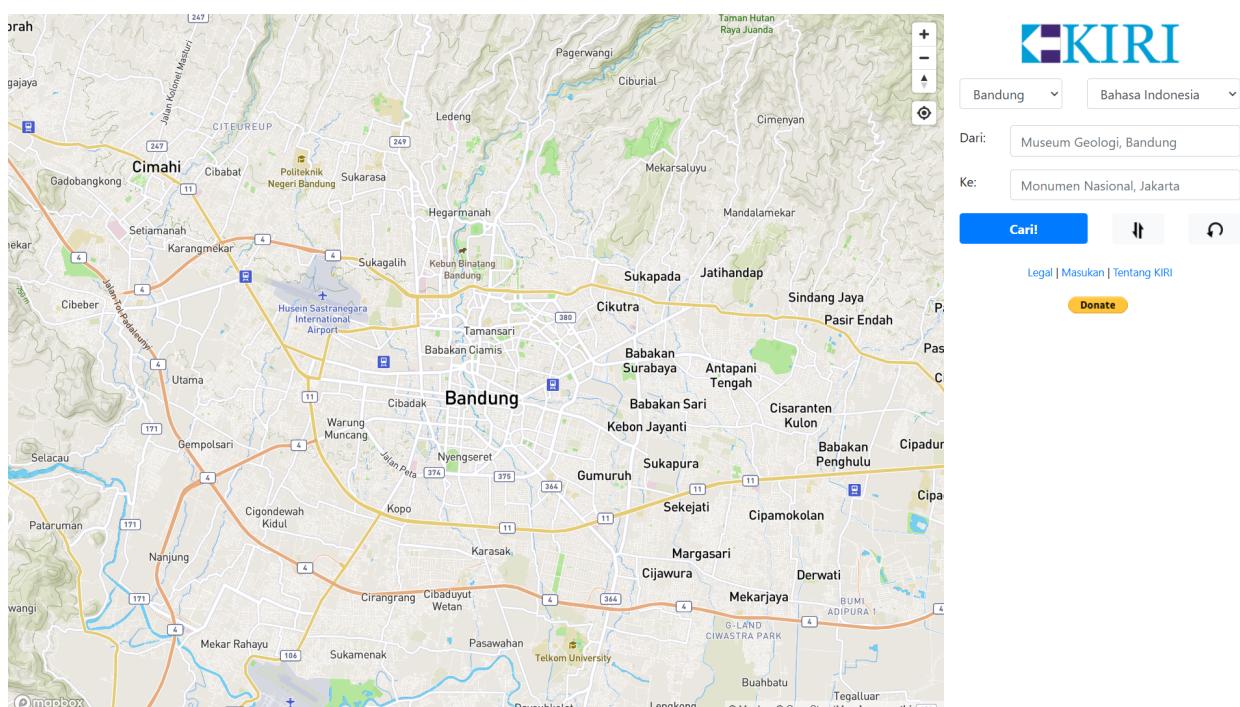
BAB 1

2

PENDAHULUAN

3 1.1 Latar Belakang

- 4 Perangkat lunak KIRI¹ (lihat Gambar 1.1) adalah perangkat lunak berbasis *web* yang dirancang
5 untuk membantu pengguna menemukan rute perjalanan ketika menggunakan angkot untuk di
6 Bandung serta TransJakarta dan Commuterline untuk di DKI Jakarta. Pada perangkat lunak
7 KIRI, pengguna dapat memasukkan titik awal perjalanan dan titik tujuan. KIRI kemudian akan
mencari berbagai alternatif rute yang bisa digunakan untuk mencapai tujuan tersebut.

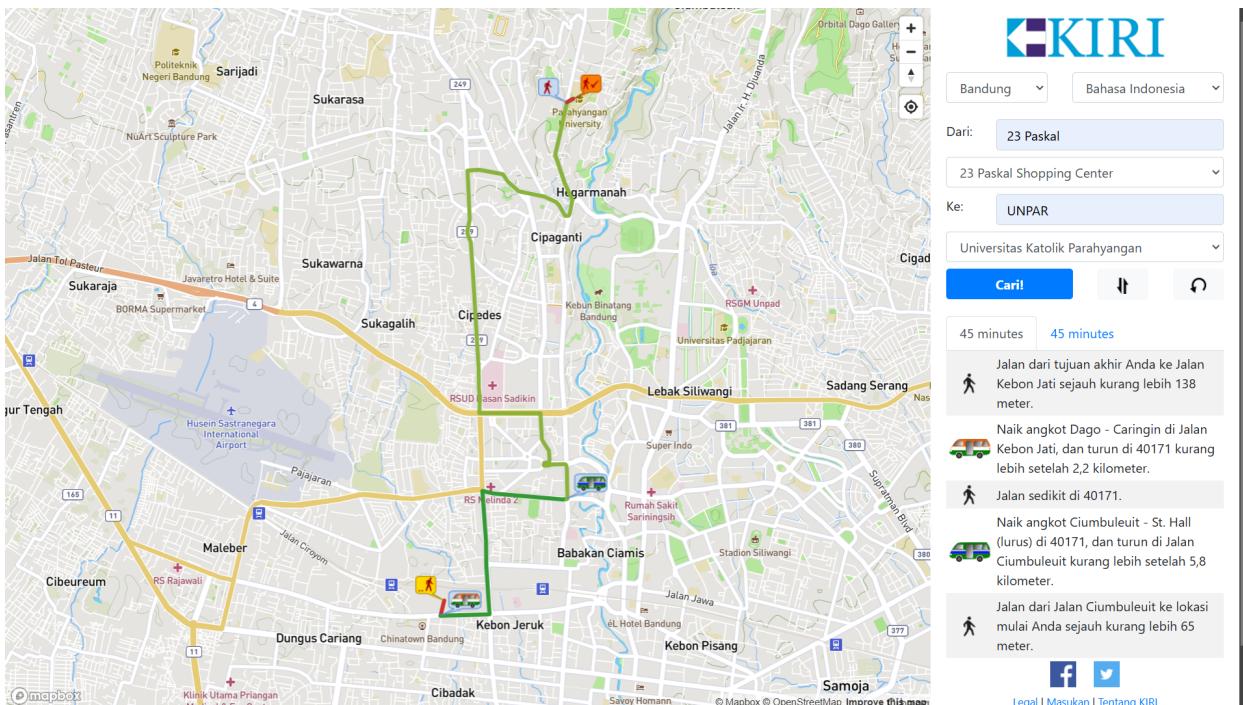


Gambar 1.1: Tampilan halaman perangkat lunak KIRI

8

- 9 KIRI akan memberikan informasi mengenai langkah-langkah yang harus ditempuh oleh pengguna
10 yang akan bepergian dari suatu tempat ke tempat tujuannya, mulai dari seberapa jauh pengguna
11 harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun
12 angkot tersebut, seberapa jauh lagi pengguna harus berjalan sampai ke lokasi tujuan, dan seberapa
13 lama estimasi waktu perjalanan yang akan ditempuh (lihat Gambar 1.2).

¹<https://projectkiri.id/> (diakses 14 Februari 2025)



Gambar 1.2: Tampilan perangkat lunak KIRI, setelah menerima masukan

1 Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Bagian *frontend*, yang dinamakan
 2 Tirtayasa dan dibangun menggunakan bahasa pemrograman PHP serta mengandalkan basis data
 3 MySQL untuk menyimpan serta mengelola data. Selain itu, Tirtayasa juga menggunakan framework
 4 CodeIgniter 3. Saat menerima permintaan pencarian, Tirtayasa meneruskannya ke bagian *backend*,
 5 yaitu NewMenjangan. Hasil dari NewMenjangan kemudian diformat agar dapat dibaca dengan
 6 baik oleh pengguna. Bagian ini diimplementasikan dalam bahasa pemrograman Java dan berperan
 7 penting dalam perhitungan rute optimal. [1]
 8 NewMenjangan merupakan program *daemon* yang berjalan secara otomatis saat server dinyalakan
 9 dan terus beroperasi hingga server dimatikan. *Daemon* sendiri adalah program komputer yang
 10 berjalan dilatar belakang dan tidak berinteraksi langsung dengan pengguna². Pada saat eksekusi,
 11 NewMenjangan terhubung ke basis data MySQL untuk mengambil data rute angkot yang tersimpan
 12 dalam format LineString. LineString adalah salah satu tipe data geometris dalam MySQL yang
 13 mewakili satu atau lebih segmen garis yang terhubung. LineString terdiri dari urutan titik (*point*)
 14 yang membentuk jalur atau lintasan³. Setiap titik pada LineString merepresentasikan lokasi
 15 potensial untuk penumpang naik atau turun. Dari data tersebut, NewMenjangan membangun
 16 *weighted graph* dalam memori (RAM) dalam bentuk *adjacency list* dan melakukan prakomputasi.
 17 Setiap titik pada LineString menjadi akan *node*, dan antara titik ke-*i* dan titik ke-(*i*+1) dihubungkan
 18 dengan *edge*. Jika ada dua titik dari rute angkot berbeda yang berdekatan (jarak di bawah konstanta
 19 tertentu), maka dibuatkan juga *edge*, yang menunjukkan kemungkinan seseorang dapat turun dari
 20 suatu angkot dan naik ke angkot lainnya untuk meneruskan perjalanan.

²<https://www.ibm.com/docs/en/aix/7.1?topic=processes-> (diakses 14 Februari 2025)

³<https://dev.mysql.com/doc/refman/8.4/en/gis-linestring-property-functions.html> (diakses 14 Februari 2025)

1 Saat NewMenjangan menerima permintaan pencarian dari titik A ke titik B, kedua titik tersebut
2 dijadikan *node* sementara, dan dibuatkan *edge* sementara ke *node-node* yang sudah ada sebelumnya,
3 jika jaraknya di bawah konstanta tertentu. Pencarian jarak terdekat pada graf tersebut dilakukan
4 menggunakan algoritma Dijkstra versi teroptimasi (*priority queue* dengan struktur data *heap*).
5 Proses ini dapat dilakukan secara paralel dengan aman (*thread-safe*) tanpa mengubah graf utama.
6 Seperti pada judul tugas akhir ini, akan dilakukan modularisasi algoritma *shortest path* pada
7 perangkat lunak KIRI menggunakan strategy pattern. Modularisasi adalah membagi perangkat
8 lunak menjadi beberapa komponen yang terpisah, yang disebut modul, yang masing-masing dapat
9 diakses dan diberi nama secara independen tetapi bekerja bersama untuk memenuhi kebutuhan
10 sistem [7]. Saat ini algoritma yang digunakan KIRI masih terikat dengan algoritma Dijkstra. Oleh
11 karena itu, pada tugas akhir ini akan diimplementasikan algoritma lainnya, yaitu algoritma A* dan
12 Floyd-Warshall sebagai *concrete strategy*. Selain itu, akan dilakukan juga penerapan arsitektur kelas
13 *strategy pattern* sehingga aplikasi KIRI akan menjadi lebih fleksibel dalam pemilihan algoritma
14 *shortest path* yang akan digunakan dan juga memudahkan apabila akan dilakukan perubahan atau
15 perbaikan pada suatu algoritma yang digunakan.

16 **1.2 Rumusan Masalah**

- 17 1. Bagaimana cara melakukan perubahan kode pada NewMenjangan untuk menerapkan strategy
18 pattern?
19 2. Bagaimana mengimplementasikan algoritma A* dan Floyd Warshall sebagai *concrete strategy*?

20 **1.3 Tujuan**

- 21 1. Melakukan perubahan arsitektur kelas dengan menerapkan strategy pattern.
22 2. Melakukan implementasi algoritma A* dan Floyd Warshall.

23 **1.4 Batasan Masalah**

24 ...

25 **1.5 Metodologi**

- 26 Metodologi yang akan digunakan dalam pembuatan tugas akhir ini adalah sebagai berikut:
27 1. Melakukan eksplorasi fungsi-fungsi dan cara kerja perangkat lunak KIRI.
28 2. Mempelajari modul-modul yang terdapat pada Tirtayasa dan NewMenjangan.
29 3. Mempelajari bahasa pemrograman PHP dan framework CodeIgniter 3.
30 4. Melakukan studi literatur mengenai penerapan arsitektur kelas strategy pattern.
31 5. Mempelajari cara kerja algoritma Dijkstra, A*, dan Floyd Warshall.
32 6. Mengubah implementasi algoritma Dijkstra yang sudah ada ke dalam strategy pattern.
33 7. Mengimplementasikan algoritma A-star dan Floyd Warshall.
34 8. Melakukan pengujian dan eksperimen.
35 9. Menulis dokumen tugas akhir.

1 1.6 Sistematika Pembahasan

2 Tugas akhir ini akan disusun menjadi beberapa bab sebagai berikut:

- 3 • **Bab 1:** Pendahuluan

4 Bab ini berisi latar belakang, rumusan masalah,tujuan, batasan masalah, metodologi, dan
5 sistematika pembahasan.

- 6 • **Bab 2:** Landasan Teori

7 Bab ini berisi dasar dari teori-teori yang dibutuhkan dalam penelitian ini, seperti KIRI, *Design*
8 *Pattern* dan *Strategy Pattern*, MySQL, LineString, Graf, dan juga algoritma-algoritma *shortest*
9 *path* yang akan diimplementasikan, yaitu algoritma Dijkstra, algoritma Floyd-Warshall, dan
10 algoritma A*.

- 11 • **Bab 3:** Analisis

12 Bab ini berisi analisis dari sistem KIRI saat ini yang belum mengimplementasikan *Strategy*
13 *Pattern* serta hanya mengimplementasikan algoritma Dijkstra dan juga analisi sistem usulan
14 yang akan mengimplementasikan *Strategy Pattern* serta 2 algoritma *shortest path* lainnya,
15 yaitu algoritma Floyd-Warshall, dan algoritma A*.

- 16 • **Bab 4:** Perancangan

17 Bab ini berisi rancangan dari pengembangan yang akan dilakukan perangkat lunak KIRI.

- 18 • **Bab 5:** Implementasi dan Pengujian

19 Bab ini berisikan pengimplementasian yang akan dilakukan untuk pengembangan perangkat
20 lunak KIRI. Selain itu, bab ini juga berisi pengujian fungsi-fungsi pada perangkat lunak KIRI
21 setelah dilakukan pengembangan.

- 22 • **Bab 6:** Kesimpulan dan Saran

23 Bab ini berisi kesimpulan dari hasil pengembangan perangkat lunak kiri dan juga saran untuk
24 pengembangan berikutnya.

1

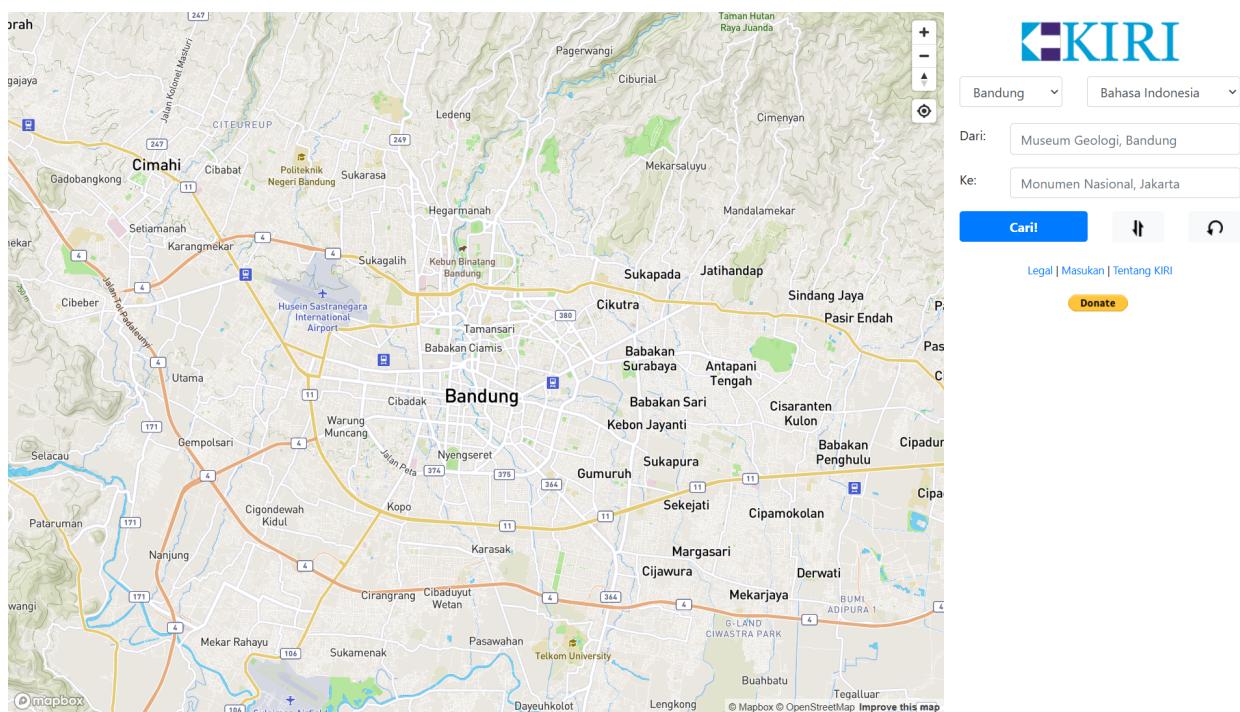
BAB 2

2

LANDASAN TEORI

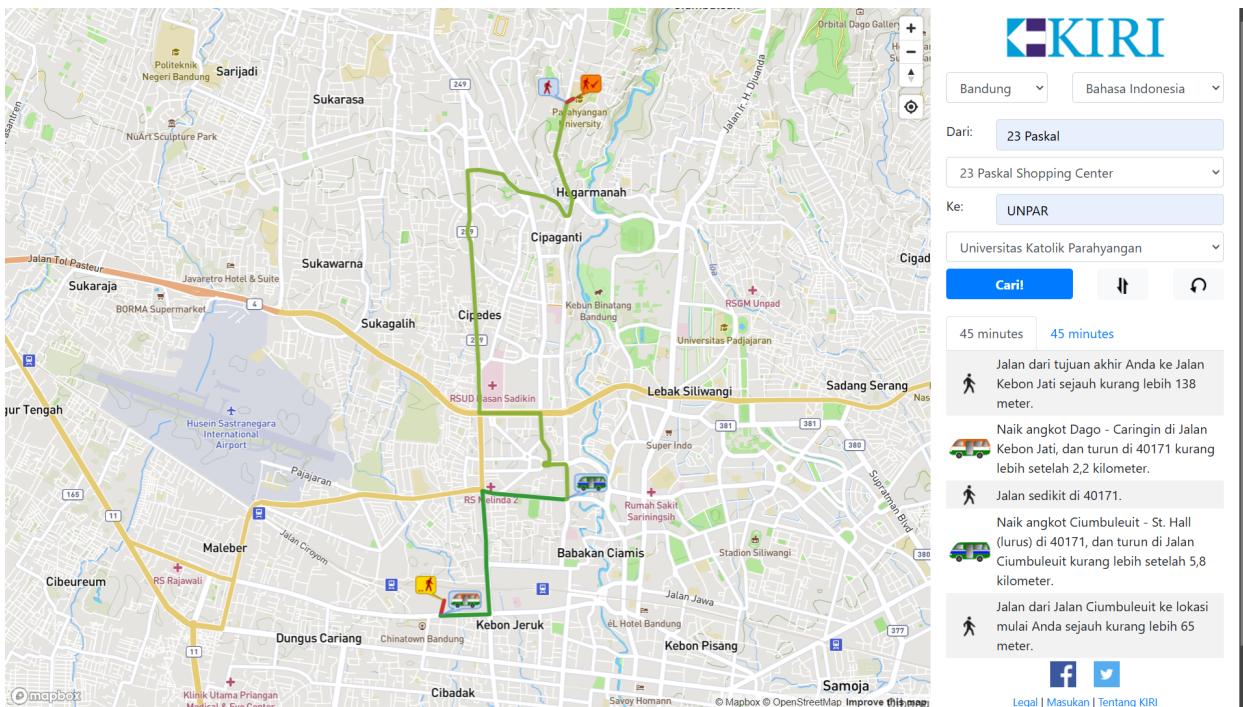
3 2.1 KIRI [1]

- 4 KIRI (lihat Gambar 2.1) adalah aplikasi navigasi transportasi umum berbasis web yang menyediakan
5 rute antara dua lokasi geografis menggunakan transportasi publik. KIRI dirancang untuk melayani
6 kebutuhan pengguna angkot (angkutan kota) di Bandung serta TransJakarta dan Commuterline di
7 DKI Jakarta. Salah satu keunggulan KIRI dibandingkan layanan seperti Google Maps atau Moovit
8 adalah kemampuannya memahami karakteristik transportasi publik, di mana penumpang dapat
9 naik atau turun di sepanjang jalan tanpa terbatas pada halte tertentu.



Gambar 2.1: Tampilan awal perangkat lunak KIRI

- 10 KIRI akan memberikan informasi mengenai langkah-langkah yang harus ditempuh oleh pengguna
11 yang akan berpergian dari satu tempat ke tempat tujuannya, mulai dari seberapa jauh pengguna
12 harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun
13 angkot tersebut, seberapa jauh lagi pengguna harus berjalan sampai ke lokasi tujuan, dan seberapa
14 lama estimasi waktu perjalanan yang akan ditempuh (lihat Gambar 2.2).



Gambar 2.2: Tampilan perangkat lunak KIRI, setelah menerima masukan

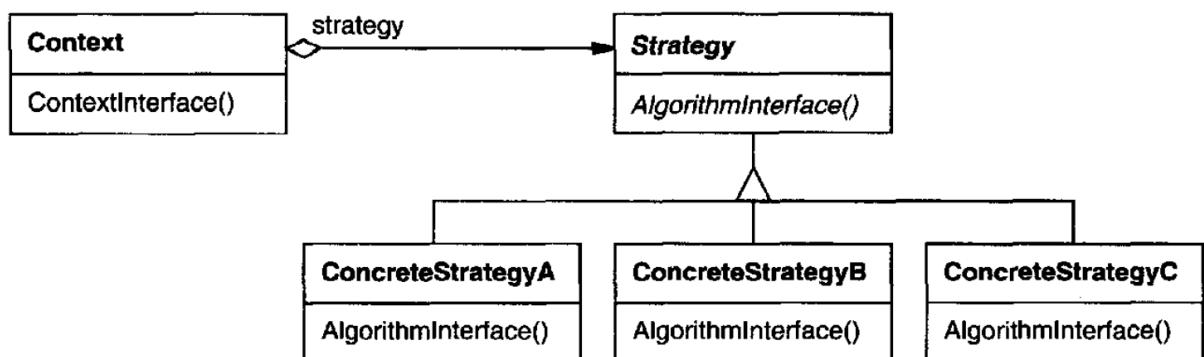
1 Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Yang pertama, yaitu Tirtayasa yang
 2 merupakan bagian *frontend* dari KIRI, dan bertanggung jawab sebagai antarmuka pengguna untuk
 3 browser web. Komponen ini mengubah nama tempat yang dimasukkan pengguna menjadi koordinat
 4 geografis dengan menggunakan bantuan Google Maps. Tirtayasa sendiri dibangun menggunakan
 5 PHP dan Framework CodeIgniter 4.
 6 Selanjutnya, NewMenjangan, yang merupakan bagian *backend* dari KIRI dan dibangun dengan
 7 bahasa pemrograman Java serta digunakan untuk memproses permintaan navigasi. Komponen ini
 8 memuat semua jalur transportasi umum dalam bentuk graf dan menggunakan algoritma Dijkstra
 9 untuk menghitung rute optimal. Algoritma ini dipercepat dengan penggunaan struktur data heap,
 10 yang membuatnya efisien untuk jalur yang kompleks.

11 2.2 Design Pattern dan Strategy Pattern [2]

12 *Design Pattern* adalah solusi umum yang telah terbukti efektif untuk mengatasi masalah desain
 13 berulang dalam pengembangan perangkat lunak berorientasi objek. Solusi ini dirancang agar dapat
 14 digunakan kembali di berbagai konteks tanpa harus disesuaikan secara berlebihan. Sebagai contoh,
 15 pola desain membantu memecah masalah desain menjadi struktur yang lebih modular dan fleksibel,
 16 sehingga mempermudah pengembangan dan pemeliharaan perangkat lunak.
 17 Pada dasarnya, design pattern memiliki empat elemen utama. Pertama, nama pola yang mem-
 18 berikan cara singkat untuk menyebut masalah desain tertentu, solusinya, dan konsekuensi dari
 19 penerapannya. Kedua, masalah, yaitu deskripsi konteks atau situasi di mana pola desain ini relevan.
 20 Ketiga, solusi, berupa abstraksi dari elemen-elemen desain dan kolaborasinya tanpa menyebutkan
 21 implementasi konkret. Keempat, konsekuensi, yang mencakup hasil dari penerapan pola, termasuk
 22 dampak pada fleksibilitas, efisiensi, dan pengelolaan sistem.

- 1 Penggunaan *design pattern* juga memungkinkan sistem menjadi lebih adaptif terhadap perubahan
 2 kebutuhan. Pola seperti *Strategy* mempermudah pergantian algoritma di runtime, sedangkan *Factory*
 3 *Method* membantu mengurangi ketergantungan pada implementasi spesifik dengan menyediakan
 4 cara fleksibel untuk membuat objek. Dengan demikian, design pattern mempermudah kolaborasi
 5 dan komunikasi antar tim pengembang.
- 6 Strategy Pattern merupakan salah satu pola desain perilaku yang dirancang untuk mendefinisikan
 7 serangkaian algoritma, mengenkapsulasi setiap algoritma, dan memungkinkan algoritma-algoritma
 8 tersebut untuk saling dipertukarkan. Pola ini memungkinkan algoritma untuk bervariasi. Dengan
 9 demikian, klien tidak perlu mengetahui detail implementasi dari algoritma yang digunakan, mela-
 10 inkan cukup berinteraksi melalui antarmuka umum yang disediakan oleh objek strategi.
- 11 Pola ini sangat berguna ketika terdapat kebutuhan untuk mendukung berbagai varian algoritma
 12 dalam menyelesaikan tugas yang sama. Strategy Pattern memindahkan setiap algoritma ke dalam
 13 kelas terpisah, yang disebut sebagai *concrete strategy*. Klien dapat memilih dan menentukan strategi
 14 yang sesuai ke dalam konteks pada waktu eksekusi, sehingga memberikan fleksibilitas yang tinggi
 15 dalam proses pengembangan perangkat lunak.
- 16 Manfaat utama dari strategy pattern adalah kemampuannya untuk menghilangkan kompleksitas
 17 yang diakibatkan oleh penggunaan pernyataan kondisional yang rumit dalam kode, serta kemudahan
 18 dalam menambahkan atau mengganti algoritma tanpa perlu memodifikasi kode klien atau konteks.
 19 Namun, penerapan pola ini juga memiliki kelemahan, seperti meningkatnya jumlah kelas dalam
 20 sistem dan potensi timbulnya *overhead* komunikasi antara konteks dan strategi. Oleh karena itu,
 21 penerapan strategy pattern sebaiknya dipertimbangkan dengan cermat, terutama dalam situasi di
 22 mana variasi algoritma memang diperlukan untuk memenuhi kebutuhan sistem.

Berikut merupakan struktur dari strategy pattern (Gambar 2.3).



Gambar 2.3: Struktur Strategy Pattern

2.2.1 Contoh Kode Program

Kode 2.1: IntegerSorter.java

```

2 abstract class IntegerSorter {
3     public abstract int[] sort(int[] arr);
4 }
5

```

Kode 2.2: ArraysSort.java

```

7 import java.util.*;
8
9 class ArraysSort extends IntegerSorter {
10    public int[] sort(int[] arr) {
11        Arrays.sort(arr);
12        return arr;
13    }
14 }
15

```

Kode 2.3: BubbleSort.java

```

17 class BubbleSort extends IntegerSorter {
18    public int[] sort(int[] arr) {
19        int n = arr.length;
20        for (int i = 0; i < n - 1; i++) {
21            for (int j = 0; j < n - i - 1; j++) {
22                if (arr[j] > arr[j + 1]) {
23                    int temp = arr[j];
24                    arr[j] = arr[j + 1];
25                    arr[j + 1] = temp;
26                }
27            }
28        }
29        return arr;
30    }
31 }
32

```

Kode 2.4: InsertionSort.java

```

34 class InsertionSort extends IntegerSorter {
35    public int[] sort(int[] arr) {
36        int n = arr.length;
37        for (int i = 1; i < n; i++) {
38            int key = arr[i];
39            int j = i - 1;
40
41            while (j >= 0 && arr[j] > key) {
42                arr[j + 1] = arr[j];
43                j = j - 1;
44            }
45            arr[j + 1] = key;
46        }
47        return arr;
48    }
49 }
50

```

Kode 2.5: Main.java

```

52 import java.util.*;
53
54 public class Main {
55     public static void main(String[] args) {
56         int[] arr = { 6, 4, 21, 9, 14, 17, 3 };
57
58         IntegerSorter arraysSort = new ArraysSort();
59         System.out.println("Arrays.sort: " + Arrays.toString(arrsSort.sort(arr.clone())));
60
61         IntegerSorter bubbleSort = new BubbleSort();
62         System.out.println("Bubble_Sort: " + Arrays.toString(bubbleSort.sort(arr.clone())));
63
64         IntegerSorter insertionSort = new InsertionSort();
65         System.out.println("Insertion_Sort: " + Arrays.toString(insertionSort.sort(arr.clone())));
66
67     }
68 }
69

```

1 2.3 MySQL [3]

2 MySQL merupakan sistem manajemen basis data relasional (*Relational Database Management
3 System/RDBMS*) bersifat open source yang dikembangkan oleh *Oracle Corporation*. SQL, yang
4 merupakan singkatan dari *Structured Query Language*, adalah bahasa pemrograman yang digunakan
5 untuk mengambil, memperbarui, menghapus, serta memanipulasi data pada basis data relasional.
6 Sebagai basis data relasional, MySQL menyimpan data dalam bentuk tabel yang terdiri atas baris
7 dan kolom, yang disusun dalam suatu skema. Skema ini bertugas mendefinisikan bagaimana data
8 diorganisasi dan disimpan, serta menjelaskan hubungan antara tabel-tabel yang ada di dalamnya.
9 Dalam MySQL, terdapat berbagai sintaks yang digunakan untuk mendukung pengelolaan basis
10 data. Sintaks-sintaks tersebut mencakup operasi penting, seperti pembuatan tabel, penyisipan data,
11 pembaruan data, penghapusan data, hingga pengambilan data. Setiap sintaks dirancang untuk
12 mempermudah pengguna dalam mengelola data secara efektif dan efisien sesuai kebutuhan sistem.
13 Berikut merupakan sintaks-sintaks dasar yang umum digunakan dalam MySQL.

14 • ***CREATE DATABASE***

15 1 **CREATE DATABASE database_name;**

18 Sintaks tersebut digunakan untuk membuat database baru dalam MySQL. **database_name**
19 diisi nama dari database baru yang akan dibuat.

20 • ***DROP DATABASE***

21 1 **DROP DATABASE database_name;**

24 Sintaks tersebut digunakan untuk menghapus database yang telah dibuat dalam MySQL.
25 **database_name** diisi nama dari database yang akan dihapus.

26 • ***CREATE TABLE***

27 1 **CREATE TABLE table_name (**
28 2 column1 datatype,
29 3 column2 datatype,
30 4 column3 datatype,
31 5
32 6);

35 Sintaks tersebut digunakan untuk membuat atau memasukan tabel baru kedalam sebuah
36 database. **table_name** diisi nama dari tabel yang akan dibuat, **column1**, **column2**, **column3**
37 dan seterusnya diisi dengan nama kolom didalam tabel yang akan dibuat, dan **datatype** diisi
38 dengan tipe data dari kolom yang akan dibuat, seperti **varchar**, **integer**, **date**, dan lain-lain.

39 • ***DROP TABLE***

40 1 **DROP TABLE table_name;;**

43 Sintaks tersebut digunakan untuk menghapus tabel yang telah dibuat dalam sebuah database.
44 **table_name** diisi nama dari tabel yang akan dihapus.

45 • ***SELECT***

46 1 **SELECT column1, column2, ...**
47 2 **FROM table_name;**

50 Sintaks tersebut digunakan untuk memilih atau mengambil data dari sebuah tabel dalam
51 database. **column1**, **column2** dan seterusnya diisi dengan nama kolom dari sebuah tabel
52 yang datanya akan diambil dan **table_name** diisi dengan nama tabel dimana kolom tersebut
53 berada.

• **WHERE**

```
1   SELECT column1, column2, ...
2   FROM table_name
3   WHERE condition;
```

Sintaks tersebut digunakan untuk memilih atau mengambil data dari sebuah tabel dalam database dengan sebuah kondisi tertentu yang bertujuan untuk memfilter data yang akan diambil. `column1`, `column2` dan seterusnya diisi dengan nama kolom dari sebuah tabel yang datanya akan diambil, `table_name` diisi dengan nama tabel dimana kolom tersebut berada, dan `condition` diisi dengan kondisi dari data yang akan diambil atau filter seperti apa yang dingin dilakukan ketika mengambil data.

• **INSERT INTO**

```
1   INSERT INTO table_name (column1, column2, column3, ...)
2   VALUES (value1, value2, value3, ...);
```

Sintaks tersebut digunakan untuk memasukan atau menambahkan data baru kedalam kolom dari sebuah tabel yang telah ada. `table_name` diisi nama tabel yang akan ditambahkan data baru, `column1`, `column2`, `column3` dan seterusnya diisi dengan nama kolom yang akan ditambahkan data baru, dan `value` diisi dengan nilai atau value dari data baru yang akan ditambahkan.

• **DELETE**

```
1   DELETE FROM table_name
2   WHERE condition;
```

Sintaks tersebut digunakan untuk menghapus data baru kedalam kolom dari sebuah tabel yang telah ada. `table_name` diisi nama tabel yang akan ditambahkan data baru, `column1`, `column2`, `column3` dan seterusnya diisi dengan nama kolom yang akan ditambahkan data baru, dan `value1`, `value2`, `value3` dan seterusnya diisi dengan nilai atau value dari data baru yang akan ditambahkan.

2.3.1 LineString [3]

LineString adalah tipe data geometris dalam MySQL yang mewakili jalur atau lintasan yang terdiri dari satu atau lebih segmen garis yang terhubung. Tipe data ini digunakan dalam Sistem Informasi Geografis (GIS) untuk merepresentasikan lintasan seperti jalan, sungai, atau rute perjalanan. Setiap *LineString* terdiri dari urutan titik (point) yang memiliki koordinat (x, y) dan minimal memiliki dua titik untuk membentuk garis.

Untuk memanipulasi dan menganalisis *LineString*, MySQL menyediakan sejumlah fungsi bawaan. Sebelum fungsi tersebut digunakan, objek *LineString* biasanya dikonversi ke bentuk geometris menggunakan fungsi `ST_GeomFromText()`. Fungsi ini menerima teks representasi geometris, seperti `LineString(x1y1, x2y2, ...)` dan mengubahnya menjadi objek geometris yang dapat diproses oleh fungsi GIS lainnya. Berikut adalah beberapa fungsi penting yang dapat digunakan untuk bekerja dengan *LineString*:

- `ST_EndPoint(ls)`

Mengembalikan titik akhir dari *LineString* `ls`. Contoh:

```
1   SET @ls = 'LineString(1,1,2,2,3,3)';
2   SELECT ST_AsText(ST_EndPoint(ST_GeomFromText(@ls)));
3   -- Hasil: 'POINT(3,3)'
```

- 1 • ST_IsClosed(ls)

2 Mengecek apakah *LineString* ls membentuk lintasan tertutup (titik awal dan akhir sama).

3 Contoh:

```
4
5     SET @ls = 'LineString(1_1,_2_2,_3_3,_1_1)';
6     SELECT ST_IsClosed(ST_GeomFromText(@ls));
7     -- Hasil: 1 (TRUE)
```

- 9 • ST_Length(ls)

10 Menghitung panjang total *LineString* ls. Contoh:

```
11
12     SET @ls = 'LineString(1_1,_2_2,_3_3)';
13     SELECT ST_Length(ST_GeomFromText(@ls));
14     -- Hasil: 2.828427
```

- 16 • ST_NumPoints(ls)

17 Mengembalikan jumlah titik yang membentuk *LineString* ls. Contoh:

```
18
19     SET @ls = 'LineString(1_1,_2_2,_3_3)';
20     SELECT ST_NumPoints(ST_GeomFromText(@ls));
21     -- Hasil: 3
```

- 23 • ST_Point(ls, N)

24 Mengembalikan titik ke-N pada *LineString* ls. Contoh:

```
25
26     SET @ls = 'LineString(1_1,_2_2,_3_3)';
27     SELECT ST_AsText(ST_PointN(ST_GeomFromText(@ls), 2));
28     -- Hasil: 'POINT(2_2)'
```

- 30 • ST_StartPoint(ls)

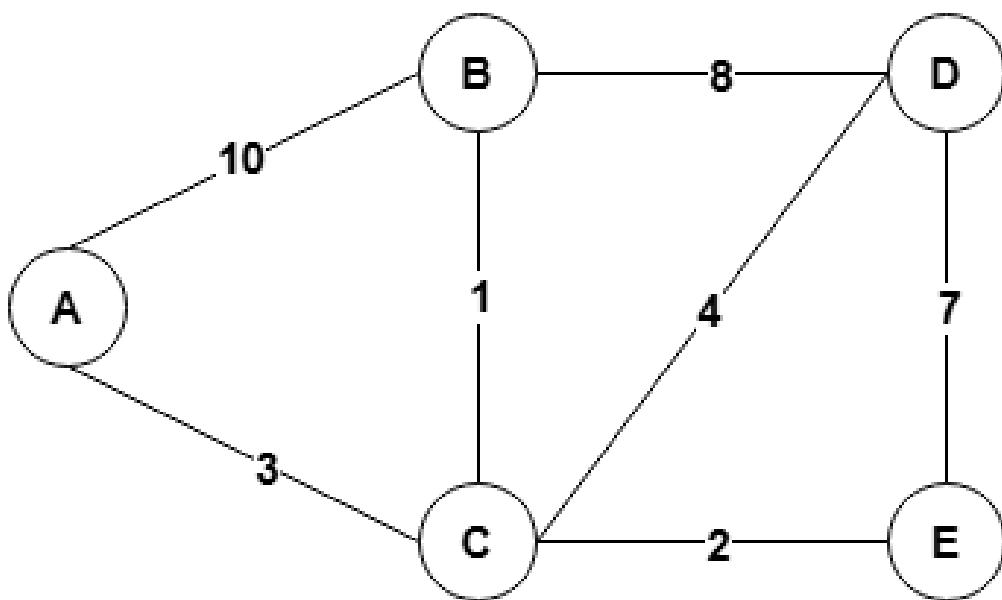
31 Mengembalikan titik awal dari *LineString* ls. Contoh:

```
32
33     SET @ls = 'LineString(1_1,_2_2,_3_3)';
34     SELECT ST_AsText(ST_StartPoint(ST_GeomFromText(@ls)));
35     -- Hasil: 'POINT(1_1)'
```

37 2.4 Graf [4]

38 Graf (G) adalah struktur yang terdiri dari simpul (*vertex*) dan sisi (*edge*), dimana sisi menghubungkan
 39 pasangan simpul (lihat Gambar 2.8). Sebuah graf direpresentasikan sebagai pasangan $G = (V, E)$,
 40 dengan V sebagai himpunan simpul dan E sebagai himpunan sisi. Sisi diwakili oleh pasangan
 41 simpul yang terhubung. Graf dapat bersifat terarah atau tidak terarah. Simpul-simpul dalam graf
 42 dapat memiliki derajat tertentu, yaitu jumlah sisi yang menghubunginya. Sebuah graf disebut
 43 terhubung jika terdapat jalur antara setiap pasangan simpul. Jalur ini adalah urutan simpul yang
 44 dihubungkan oleh sisi. Selain itu, siklus adalah jalur tertutup di mana simpul awal dan akhir adalah
 45 sama. Teori graf juga mencakup konsep seperti pohon (graf terhubung tanpa siklus), graf bipartit
 46 (simpul dibagi menjadi dua himpunan yang saling bebas sisi), dan subgraf (bagian dari graf yang
 47 tetap mempertahankan struktur graf).

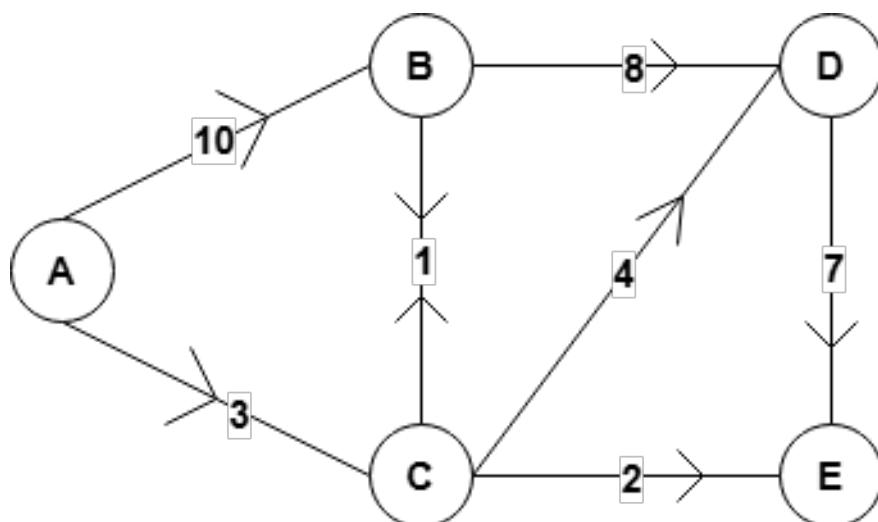
48 Graf digunakan dalam berbagai hal, seperti jaringan komputer, rute transportasi, dan analisis
 49 hubungan sosial. Teori graf menyediakan dasar matematis untuk mempelajari struktur ini dan
 50 memberikan cara untuk memodelkan dan menyelesaikan masalah kompleks di berbagai bidang.



Gambar 2.4: Graf

¹ 2.4.1 Graf Berarah [4]

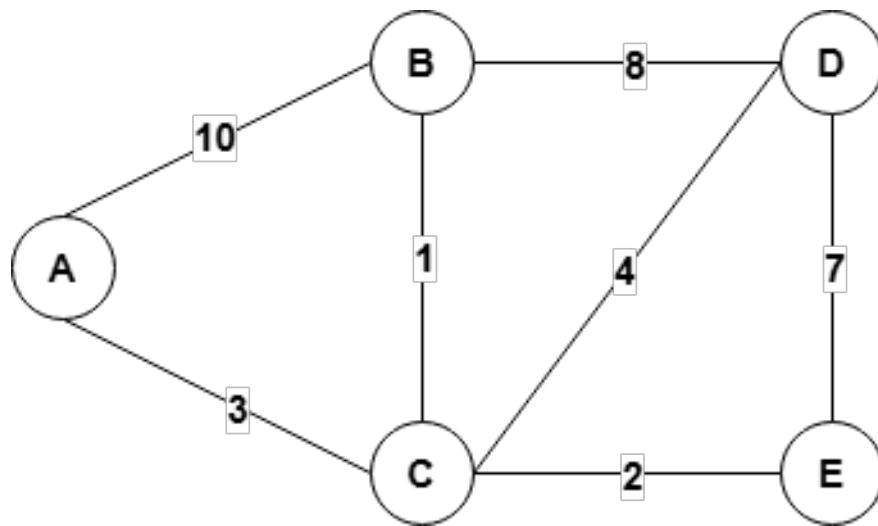
² Graf berarah adalah graf di mana setiap sisi memiliki arah tertentu. Dengan kata lain, setiap sisi
³ dalam graf ini diwakili oleh pasangan terurut dari simpul (*vertex*). Sebuah graf berarah didefinisikan
⁴ sebagai pasangan $G = (V, E)$, di mana V adalah himpunan simpul dan E adalah himpunan sisi
⁵ yang berbentuk pasangan terurut dari simpul. Setiap sisi e memiliki simpul awal $init(e)$ dan simpul
⁶ terminal $ter(e)$ yang menunjukkan arah perjalanan dari satu simpul ke simpul lainnya.
⁷ Graf berarah dapat memiliki berbagai karakteristik khusus. Jika terdapat lebih dari satu sisi dengan
⁸ arah yang sama antara dua simpul yang sama, sisi tersebut disebut sisi paralel. Jika sebuah sisi
⁹ memiliki awal dan terminal yang sama, maka sisi tersebut disebut loop. Selain itu, jika suatu graf
¹⁰ tidak berarah diberikan arah pada setiap sisinya, maka graf berarah yang dihasilkan disebut sebagai
¹¹ *oriented graph*.



Gambar 2.5: Graf Berarah

¹ 2.4.2 Graf Tidak Berarah [4]

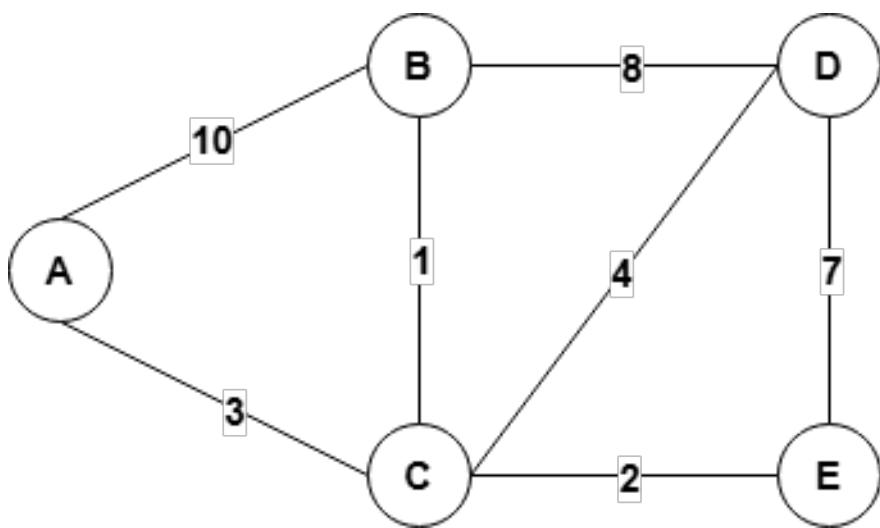
- ² Graf tidak berarah adalah jenis graf di mana setiap sisi hanya menghubungkan dua simpul tanpa
³ adanya arah tertentu. Dalam graf ini, sisi diwakili sebagai pasangan tidak terurut $\{u, v\}$ yang
⁴ berarti bahwa pergerakan dari simpul u ke v sama dengan pergerakan dari v ke u . Graf tidak
⁵ berarah dapat direpresentasikan dalam bentuk matriks ketetanggaan (*adjacency matrix*), di mana
⁶ matriks ini bersifat simetris.
- ⁷ Salah satu sifat penting dari graf tidak berarah adalah konsep keterhubungan. Sebuah graf dikatakan
⁸ terhubung jika terdapat jalur antara setiap pasangan simpulnya. Jika terdapat simpul yang tidak
⁹ dapat dijangkau dari simpul lain, maka graf tersebut disebut tidak terhubung. Graf tidak berarah
¹⁰ juga memiliki properti siklus, yaitu jalur tertutup di mana simpul awal dan simpul akhir adalah
¹¹ sama.



Gambar 2.6: Graf Tidak Berarah

¹² 2.4.3 Graf Berbobot [4]

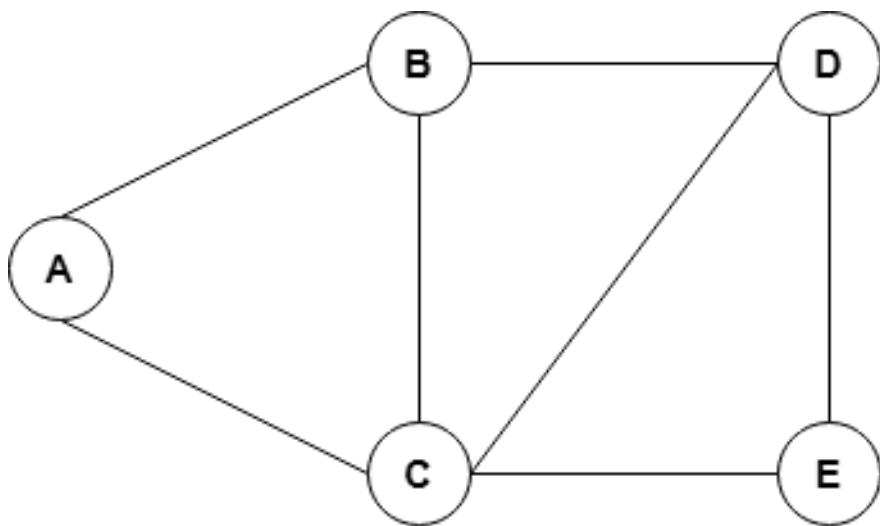
- ¹³ Graf berbobot adalah graf di mana setiap sisi memiliki bobot atau nilai numerik yang menyatakan
¹⁴ karakteristik tertentu seperti jarak, biaya, atau kapasitas. Graf berbobot direpresentasikan sebagai
¹⁵ $G = (V, E, w)$, di mana $w : E \rightarrow R$ adalah fungsi yang memberikan bobot pada setiap sisi.
- ¹⁶ Graf berbobot sering digunakan dalam berbagai aplikasi seperti algoritma pencarian jalur terpendek,
¹⁷ optimasi jaringan, dan model transportasi. Dalam beberapa kasus, graf berbobot dapat berarah
¹⁸ atau tidak berarah, tergantung pada apakah bobot tersebut memiliki arah tertentu atau tidak.



Gambar 2.7: Graf Berbobot

¹ 2.4.4 Graf Tidak Berbobot [4]

- ² Graf tidak berbobot adalah graf di mana semua sisi dianggap memiliki bobot yang sama, biasanya bernilai satu atau dianggap tidak berbobot sama sekali. Graf jenis ini sering digunakan dalam
- ³ analisis struktur jaringan, di mana hanya penting untuk mengetahui adanya hubungan antar simpul
- ⁴ tanpa mempertimbangkan seberapa kuat atau penting hubungan tersebut.
- ⁵ Dalam graf tak berbobot, algoritma seperti BFS (Breadth-First Search) dan DFS (Depth-First
- ⁶ Search) sangat berguna untuk menemukan keterhubungan, jalur terpendek dalam jumlah langkah,
- ⁷ serta untuk eksplorasi struktur graf lainnya.



Gambar 2.8: Graf Tidak Berbobot

¹ 2.5 Algoritma Shortest Path

² Ada berbagai jenis algoritma *shortest path* yang dirancang untuk menemukan lintasan terpendek
³ antara dua titik dalam sebuah graf. Algoritma ini memainkan peran penting dalam berbagai
⁴ aplikasi, seperti sistem navigasi, perencanaan jaringan, analisis data geografis, dan pemecahan
⁵ masalah rute optimal.
⁶ Setiap algoritma memiliki pendekatan, kelebihan, dan keterbatasan masing-masing, yang menjadinya
⁷ lebih sesuai untuk situasi tertentu. Sebagai contoh, algoritma seperti *Dijkstra* sangat cocok
⁸ untuk graf dengan bobot positif, sementara *Floyd-Warshall* lebih sesuai untuk menemukan lintasan
⁹ terpendek pada semua pasangan titik dalam graf kecil. Di sisi lain, algoritma *A** dirancang khusus
¹⁰ untuk mempercepat pencarian lintasan dengan memanfaatkan heuristik. Berikut pembahasan
¹¹ secara detail tiga algoritma *shortest path* yang digunakan, yaitu *Dijkstra*, *Floyd-Warshall*, dan *A**.

¹² 2.5.1 Algoritma Dijkstra [5]

¹³ Algoritma Dijkstra merupakan sebuah algoritma untuk menyelesaikan masalah *single-source shortest*
¹⁴ *path*, yaitu menemukan jalur terpendek dari satu titik asal ke semua titik lainnya dalam sebuah graf
¹⁵ berarah dengan bobot tepi non-negatif. Algoritma ini menggunakan sebuah struktur *min-priority*
¹⁶ *queue* (antrean prioritas minimum) yang menyimpan titik-titik dengan prioritas sesuai dengan
¹⁷ perkiraan jarak terpendek dari titik asal.

Algorithm 1 Dijkstra(G, w, s)

```

1: Initialize-Single-Source( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  do
5:    $u = \text{Extract-Min}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for each vertex  $v \in G.\text{Adj}[u]$  do
8:     Relax( $u, v, w$ )
9:   end for
10: end while
```

¹⁸ Pseudocode 1 merupakan pseudocode dari algoritma Dijkstra. Pada pseudocode terdapat beberapa
¹⁹ atribut diantarnya, yaitu G yang merepresentasikan graf, w merupakan bobot yang menyatakan
²⁰ jarak atau biaya antar dan s merepresentasikan simpul sumber yang merupakan titik awal pencarian.
²¹ Selain itu, S merepresentasikan kumpulan simpul yang sudah diproses yang diawali diinisialisasi
²² kosong, sedangkan Q merepresentasikan kumpulan simpul yang belum diproses, kemudian u
²³ merepresentasikan simpul yang sedang diproses dan v merepresentasikan simpul tetangga dari u .
²⁴ Algoritma Dijkstra dimulai dengan menginisialisasi perkiraan jarak terpendek dari titik asal s ke
²⁵ semua titik lain, kecuali s itu sendiri yang diinisialisasi dengan jarak 0 dan juga semua simpul
²⁶ dimasukkan ke dalam *min-priority queue* (Q), di mana prioritas ditentukan berdasarkan jarak
²⁷ terpendek yang diketahui. Selanjutnya, algoritma memproses simpul-simpul satu per satu dengan
²⁸ memilih simpul u dari Q yang memiliki jarak terpendek. Simpul tersebut kemudian ditambahkan
²⁹ ke dalam himpunan S .

- 1 Setelah simpul u diproses, algoritma akan memeriksa semua tetangga v dari u . Untuk setiap
 2 tetangga, algoritma melakukan proses *relaksasi*, yaitu membandingkan jarak saat ini ke v dengan
 3 jarak yang melewati u . Jika jalur melalui u memberikan jarak yang lebih pendek, jarak ke v
 4 diperbarui dengan jarak baru tersebut, dan simpul pendahulu v diatur menjadi u . Proses ini
 5 memastikan bahwa jalur terpendek ditemukan secara bertahap melalui iterasi. Algoritma akan
 6 terus berjalan hingga semua simpul telah diproses atau antrean Q kosong. Hasil akhir berupa jarak
 7 terpendek dari simpul sumber s ke setiap simpul lain dalam graf.
- 8 Graf pada gambar 2.8 dapat diselesaikan menggunakan Dijkstra dengan langkah-langkah berikut.

9 1. Inisialisasi

- 10 • Atur jarak semua simpul ke ∞ (tak hingga), kecuali simpul A yang diset ke 0.
 11 • Masukan semua simpul kedalam *Priority Queue*.
 12 • PQ : {(A, 0), (B, ∞), (C, ∞), (D, ∞), (E, ∞)}
 13

Simpul	Jarak Dari A	Jalur
A	0	A
B	∞	-
C	∞	-
D	∞	-
E	∞	-

14 2. Iterasi 1

- 15 • Ambil simpul A (0) dari PQ
 16 • Periksa dan Perbarui jarak ke tetangga
 – B = $\min(\infty, 0 + 10) = 10$
 – C = $\min(\infty, 0 + 3) = 3$
 17 • PQ : {(C, 3), (B, 10), (D, ∞), (E, ∞)}
 18

Simpul	Jarak Dari A	Jalur
A	0	A
B	10	A - B
C	3	A - C
D	∞	-
E	∞	-

19 3. Iterasi 2

- 20 • Ambil simpul C (3) dari PQ
 21 • Periksa dan Perbarui jarak ke tetangga
 – B = $\min(10, 3 + 1) = 4$
 – D = $\min(\infty, 3 + 4) = 7$
 – E = $\min(\infty, 3 + 2) = 5$
 22 • PQ : {(B, 4), (E, 5), (D, 7), (B, 10)}
 23
 24
 25

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

1 4. Iterasi 3

- 2 • Ambil simpul B (4) dari PQ
- 3 • Periksa dan Perbarui jarak ke tetangga
4 – D = $\min(7, 4 + 8) = 7$
- 5 • PQ : {(E, 5), (D, 7), (B, 10)}

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

6 5. Iterasi 4

- 7 • Ambil simpul E (5) dari PQ
- 8 • Periksa dan Perbarui jarak ke tetangga
9 – D = $\min(7, 5 + 7) = 7$
- 10 • PQ : {(D, 7)}

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

11 6. Iterasi 5

- 12 • Ambil simpul D (7) dari PQ
- 13 • Tidak ada tetangga atau PQ sudah kosong.

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

2.5.2 Algoritma Floyd-Warshall [5]

Algoritma Floyd-Warshall merupakan sebuah algoritma untuk menyelesaikan masalah jalur terpendek untuk semua pasangan titik dalam graf berarah dengan menggunakan pendekatan pemrograman dinamis. Algoritma ini sangat berguna untuk graf yang memiliki bobot sisi negatif, selama tidak terdapat siklus dengan bobot negatif dalam graf tersebut. Pendekatan ini menghitung jalur terpendek antara semua pasangan titik dengan menggunakan tabel bobot antar titik dan mengulanginya secara bertahap untuk mencapai solusi optimal.

Algorithm 2 Floyd-Warshall(W)

```

1:  $n = W.\text{rows}$ 
2:  $D^{(0)} = W$ 
3: for  $k = 1$  to  $n$  do
4:   Let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix
5:   for  $i = 1$  to  $n$  do
6:     for  $j = 1$  to  $n$  do
7:        $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
8:     end for
9:   end for
10: end for
11: return  $D^{(n)}$ 
```

Pseudocode 2 merupakan pseudocode dari algoritma Floyd-Warshall. Pada pseudocode terdapat beberapa atribut diantarnya, yaitu W yang merupakan sebuah matriks berbobot berukuran $n * n$ dan mewakili bobot dari setiap sisi pada graf, $D^{(0)}$ atau $D^{(k)}$ merupakan Matriks berukuran $n * n$ pada iterasi ke- k , n merepresentasikan banyaknya simpul dalam graf, diperoleh dari jumlah baris dari matriks W . Selain itu, terdapat $d_{ij}^{(k)}$ yang merupakan elemen dari matriks $D^{(k)}$ yang menunjukkan jarak terpendek antara simpul i dan j pada iterasi ke- k . Algoritma Floyd-Warshall dimulai dengan menginisialisasi n yang diinisialisasi dengan nilai baris pada matriks W dan juga $D^{(0)}$ yang diinisialisasi dengan matriks W .

Selama n iterasi, algoritma memperbarui matriks jarak terpendek dengan mempertimbangkan kemungkinan jalan melalui simpul antara $\{1, 2, \dots, k\}$. Pada setiap langkah, algoritma memeriksa apakah jarak dari i ke j dapat diperpendek dengan melalui simpul k , dibandingkan dengan jarak langsung antara i dan j . Proses ini menghasilkan solusi optimal, di mana $D^{(n)}$ mencakup semua jarak terpendek yang memungkinkan. Algoritma Floyd-Warshall memiliki kompleksitas waktu $O(n^3)$ karena terdiri dari tiga looping untuk semua titik dalam graf.

Graf pada gambar 2.8 dapat diselesaikan menggunakan Floyd-Warshall dengan langkah-langkah berikut.

24. 1. Inisialisasi Matriks Jarak
 - 25. • Setiap jarak awal diisi berdasarkan bobot sisi yang ada. Jika tidak ada sisi antara dua simpul, diisi dengan ∞ (tak hingga).

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

1 2. Iterasi 1 dengan A sebagai perantara(k) / k = A

- 2 • Tidak ada perubahan karena A hanya memiliki jalur langsung ke B dan C dan tidak ada
3 jalur baru yang lebih pendek yang bisa ditemukan.

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

4 3. Iterasi 2 dengan B sebagai perantara(k) / k = B

- 5 • Tidak ada jalur lebih pendek yang ditemukan melalui B sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

6 4. Iterasi 3 dengan C sebagai perantara (k) / k = C

- 7 • A - B dapat melalui C
8 – A - C - B = 3 + 1 = 4
9 – Jarak langsung A - B adalah 10, jadi jarak A - B diperbarui dengan jarak A - C - B.
10 • A - D dapat melalui C
11 – A - C - D = 3 + 4 = 7
12 – Jarak A - D diisi dengan jarak A - C - D.
13 • A - E dapat melalui C
14 – A - C - E = 3 + 2 = 5
15 – Jarak A - E diisi dengan jarak A - C - E.
16 • B - D dapat melalui C
17 – B - C - D = 1 + 4 = 5
18 – Jarak B - D diisi dengan jarak B - C - D.
19 • B - E dapat melalui C
20 – B - C - E = 1 + 2 = 4
21 – Jarak B - E diisi dengan jarak B - C - E.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

- 1 5. Iterasi 4 dengan D sebagai perantara (k) / k = D
- 2 • Tidak ada jalur lebih pendek yang ditemukan melalui D sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

- 3 6. Iterasi 5 dengan E sebagai perantara (k) / k = E
- 4 • Titik E sudah merupakan titik terakhir atau titik tujuan sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

5 2.5.3 Algoritma A* [6]

- 6 Algoritma A* adalah metode pencarian yang meminimalkan estimasi total biaya solusi dengan menggabungkan dua fungsi, yaitu $g(n)$ dan $h(n)$. Fungsi $g(n)$ menghitung biaya aktual dari titik awal hingga simpul n , sedangkan $h(n)$ memperkirakan biaya tersisa dari n ke tujuan. Kombinasi ini menghasilkan $f(n) = g(n) + h(n)$, yang memberikan perkiraan total biaya solusi jika rute melalui simpul n . Algoritma ini biasanya dipilih karena dapat mencapai solusi yang optimal dan lengkap, terutama jika fungsi heuristik $h(n)$ memenuhi kriteria tertentu.
- 7 Kondisi utama yang diperlukan agar algoritma A* memberikan solusi optimal adalah heuristik $h(n)$ yang bersifat *admissible*, yaitu tidak pernah melebih-lebihkan biaya ke tujuan, dan *consistent* atau *monotonic*, di mana nilai h tidak menurun di sepanjang jalur. Dengan adanya heuristik yang memenuhi syarat ini, algoritma A* dapat menghindari eksplorasi simpul-simpul yang tidak relevan, mengurangi waktu dan memori yang dibutuhkan.
- 8 Terdapat kendala utama dari algoritma A*, yaitu penggunaan memori yang besar karena algoritma ini perlu menyimpan semua simpul yang telah dihasilkan. Untuk mengatasi hal ini, terdapat varian A* seperti *Iterative-Deepening A** (IDA*) yang mengurangi kebutuhan memori tanpa mengorbankan optimalitas solusi, dengan biaya eksekusi yang sedikit lebih tinggi.
- 9 Graf pada gambar 2.8 dapat diselesaikan menggunakan A* dengan langkah-langkah berikut.
- 10 1. Inisialisasi

- 1 • Tentukan nilai heuristik setiap titik $h(n)$.
 - 2 – $h(A) = 7$
 - 3 – $h(B) = 6$
 - 4 – $h(C) = 2$
 - 5 – $h(D) = 4$
 - 6 – $h(E) = 0$
- 7 • Buat 2 buah list untuk menyimpan titik.
 - 8 – *Open List*, untuk menyimpan titik yang akan diperiksa.
 - 9 – *Closed List*, untuk menyimpan titik yang sudah diperiksa.
- 10 • Hitung total jarak untuk titik awal ($f(A)$).
 - 11 – $f(A) = 0 + 7 = 7$
- 12 • Update list.
 - 13 – *Open List*: {A}
 - 14 – *Closed List*: {}

15 2. Iterasi 1

- 16 • Pilih titik A, karena hanya ada titik A saja pada *Open List*
- 17 • Periksa titik-titik tetangganya.
 - 18 – $f(B) = 10 + 6 = 16$
 - 19 – $f(C) = 3 + 2 = 5$
- 20 • Update list.
 - 21 – *Open List*: {B, C}
 - 22 – *Closed List*: {A}

23 3. Iterasi 2

- 24 • Pilih titik C, karena memiliki total jarak terkecil pada *Open List*
- 25 • Periksa titik-titik tetangganya.
 - 26 – $f(B) = 4 + 6 = 10$
 - 27 – $f(D) = 7 + 4 = 11$
 - 28 – $f(E) = 5 + 0 = 5$
- 29 • Update list.
 - 30 – *Open List*: {B, D, E}
 - 31 – *Closed List*: {A, C}

32 4. Iterasi 3

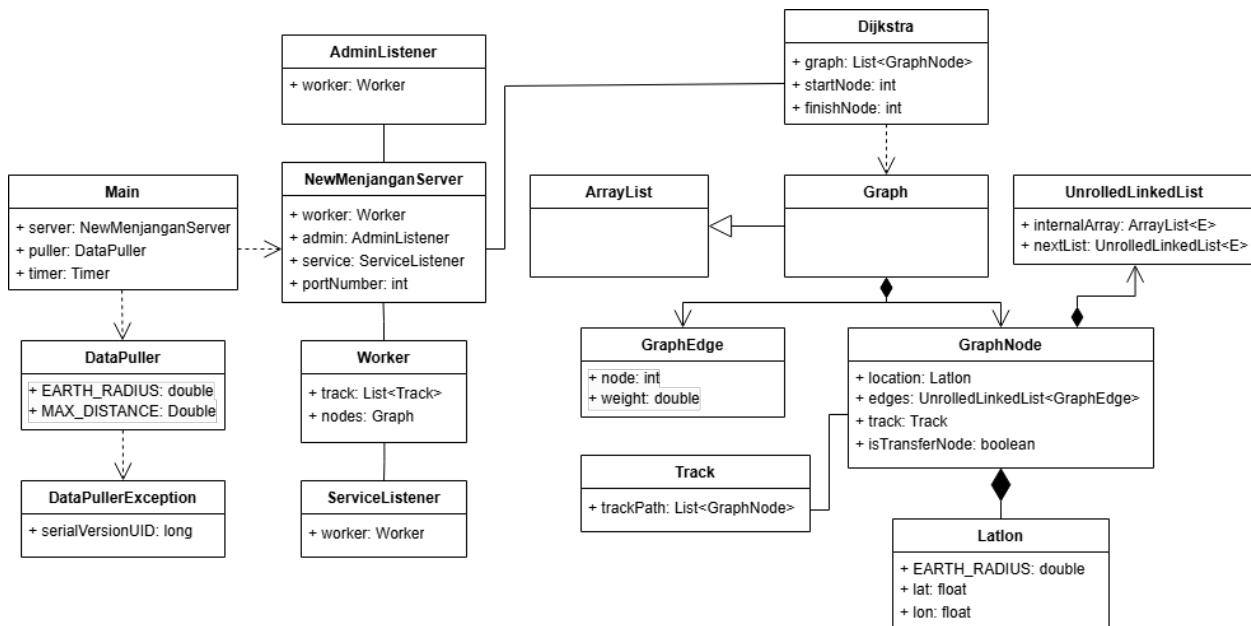
- 33 • Pilih titik E, karena memiliki total jarak terkecil pada *Open List*
- 34 • Iterasi selesai karena E adalah titik tujuan.
- 35 • *Closed List*: {A, C, E}

BAB 3

ANALISIS

3.1 Analisis Sistem Kini

Analisis akan dilakukan terhadap sistem backend dari aplikasi KIRI yang bernama NewMenjangan. Sistem ini bertanggung jawab untuk menangani berbagai fungsi backend yang mendukung layanan utama KIRI, termasuk pengolahan data, komunikasi dengan komponen lain, dan pengelolaan algoritma terkait penelusuran rute. Gambar 3.1, merupakan struktur kelas saat ini dari aplikasi KIRI yang digambarkan dalam diagram kelas. Saat ini, algoritma yang diterapkan adalah algoritma Dijkstra, yang digunakan untuk menemukan jalur terpendek dalam graf berbobot. Analisis ini mencakup tinjauan menyeluruh terhadap struktur kelas NewMenjangan dan juga pemodelan graf pada aplikasi KIRI.



Gambar 3.1: Struktur Kelas NewMenjangan

1 3.1.1 Analisis Kelas

2 Pada bagian ini, akan dilakukan analisis terhadap struktur kelas yang membentuk sistem backend
3 NewMenjangan. Analisis ini bertujuan untuk memahami peran dan hubungan antar kelas dalam
4 sistem, termasuk bagaimana kelas-kelas tersebut berinteraksi untuk menangani pencarian rute
5 menggunakan algoritma Dijkstra. Setiap kelas dalam diagram pada Gambar 3.1 akan dianalisis
6 berdasarkan fungsinya dalam sistem.

7 3.1.1.1 Main.java

8 Kelas ini berfungsi sebagai pusat kendali dari backend KIRI. Melalui kelas ini, server bisa dijalankan,
9 diperiksa statusnya, dimatikan, dan juga mengolah data. Pada kelas ini, terdapat 5 konstanta dan 5
10 atribut. Selain itu, terdapat *method - method* diimplementasikan yang memiliki penjelasan sebagai
11 berikut:

- 12 • Konstanta

- 13 – TRACKS_CONF, MYSQL_PROPERTIES, dan MJNSERVE_PROPERTIES

- 14 Konstanta-konstanta tersebut digunakan untuk mengarahkan pada file konfigurasi yang
15 diperlukan.

- 16 – LOGGING_PROPERTIES dan NEWMJNSERVE_LOG

- 17 Konstanta-konstanta tersebut digunakan untuk mengatur lokasi konfigurasi logging dan
18 file log.

- 19 • Atribut

- 20 – server, puller, dan timer

- 21 Atribut-atribut tersebut digunakan untuk mengelola server, menarik data, dan menjalankan tugas terjadwal.

- 23 – portNumber

- 24 Atribut ini berfungsi untuk menetapkan *port default* yang digunakan oleh server.

- 25 – homeDirectory

- 26 Atribut ini digunakan untuk menjadi direktori utama yang diambil dari variabel lingkungan
27 NEWMJNSERVE_HOME, yang diperlukan agar aplikasi berjalan.

- 28 • Method

- 29 – main(String[] args)

- 30 *Method* ini dirancang untuk memproses argumen yang diterima guna memeriksa status server atau menghentikannya. Ketika argumen **-c** diberikan, fungsi **sendCheckStatus** akan dipanggil untuk memastikan bahwa server sedang berjalan. Sebaliknya, jika argumen **-s** diberikan, fungsi **sendShutdown** akan bertugas mematikan server. Sebelum proses lebih lanjut dilakukan, program memeriksa apakah variabel lingkungan **NEWMJNSERVE_HOME** telah disetel. Jika tidak, aplikasi akan segera dihentikan. Setelah inisialisasi konfigurasi *logging* selesai, fungsi **pullData** dijalankan untuk menarik data yang diperlukan. Server kemudian dimulai melalui pemanggilan metode **server.start()**, dan sebuah **ShutdownHook** disertakan untuk memastikan penghentian server dilakukan dengan aman.

- 39 – sendCheckStatus(int portNumber) dan sendShutdown(int portNumber)

- 40 Keduanya menggunakan koneksi HTTP untuk mengirim permintaan ke server. *Method* **sendCheckStatus** berfungsi untuk mengecek status server, sementara *method*

- **sendShutdown** mengirim permintaan untuk mematikan server.
- **pullData()**

Method ini bertugas untuk menarik data dari sumber SQL dan sumber eksternal lainnya. Jika terjadi perubahan pada file konfigurasi **tracks.conf**, data yang ada akan ditimpak dengan data baru yang diperbarui. Selain itu, proses pembaruan ini akan dicatat dalam log untuk pemantauan perubahan data yang terjadi. *Method* akan mengembalikan nilai **true** apabila penarikan data berhasil dan **false** apabila gagal.
- **fileEquals(File file1, File file2)**

Method ini dirancang untuk membandingkan dua file secara biner untuk menentukan kesamaan di antara keduanya. Jika kedua file memiliki panjang yang berbeda, maka file tersebut secara otomatis dianggap tidak identik. Selain itu, jika ditemukan perbedaan byte pada posisi tertentu selama proses pembandingan, posisi perbedaan tersebut akan dicatat dalam log. *Method* akan mengembalikan nilai **true** apabila file *identical* dan **false** apabila tidak.

3.1.1.2 AdminListener

16 Kelas ini berfungsi sebagai *handler* HTTP khusus yang menerima perintah-perintah administratif
17 untuk mengelola server *backend* KIRI. Kelas ini memungkinkan aplikasi *backend* menerima dan
18 menjalankan perintah administrasi dari localhost melalui HTTP. Pada kelas ini, terdapat 1 atribut
19 diinisialisasikan serta *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- Atribut

— worker

Variabel ini bertipe Worker yang merupakan sebuah kelas. worker ini diperlukan untuk menjalankan perintah-perintah tertentu, seperti tracksinfo.

- Method

```
- handle(String target, Request baseRequest, HttpServletRequest request,  
       HttpServletResponse response)
```

Method ini mengimplementasikan penanganan permintaan HTTP dengan memanfaatkan kelas **AbstractHandler** dari *library* Jetty. Ketika sebuah permintaan HTTP diterima, metode ini akan melakukan beberapa langkah. Pertama, sumber permintaan diperiksa untuk memastikan bahwa hanya permintaan dari localhost yang diterima. Selanjutnya, metode ini mengurai parameter query string dari URL untuk mengidentifikasi perintah yang diminta. Dengan pendekatan ini, setiap permintaan dapat diproses sesuai dengan parameter yang dikirimkan.

Method ini mendukung berbagai jenis perintah yang dapat diterima melalui permintaan HTTP. Salah satu perintah adalah **forceShutdown**, yang berfungsi untuk menghentikan server setelah jeda satu detik dengan menjalankan `System.exit(0)` dalam sebuah thread baru. Perintah lain, yaitu **tracksInfo**, akan mengembalikan informasi mengenai jalur jika worker telah diinisialisasi, menggunakan metode `worker.printTracksInfo()`. Selain itu, perintah ping akan mengembalikan string "pong" untuk memverifikasi bahwa server sedang aktif. Apabila perintah yang diterima tidak valid atau tidak dikenali, metode ini akan mengembalikan status dan pesan kesalahan yang sesuai.

Pengaturan status dan pesan respons dilakukan berdasarkan hasil dari setiap perintah yang diproses. Jika perintah berhasil dijalankan, status `HttpStatus.OK_200` akan dikembalikan. Jika permintaan berasal dari alamat selain localhost, status yang dikembalikan adalah `HttpStatus.FORBIDDEN_403`. Permintaan yang tidak mencantumkan perintah akan menerima status `HttpStatus.BAD_REQUEST_400`, sedangkan jika worker belum siap, status `HttpStatus.SERVICE_UNAVAILABLE_503` akan diberikan.

3.1.1.3 NewMenjanganServer

Kelas ini berfungsi untuk menginisialisasi dan menjalankan server HTTP yang mendengarkan permintaan pada backend KIRI. Server ini menggunakan *library* Jetty untuk menangani permintaan HTTP. Pada kelas ini, terdapat 1 konstanta dan 6 atribut. Selain itu, terdapat sebuah konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `DEFAULT_PORT_NUMBER`

Merupakan nomor port *default* yang digunakan jika tidak ada port yang ditentukan.

- **Atribut**

- `worker`

Merupakan instance dari kelas Worker.

- `admin`

Merupakan instance dari kelas AdminListener.

- `service`

Merupakan instance dari kelas ServiceListener.

- `httpServer`

Merupakan server Jetty yang akan mendengarkan permintaan HTTP.

- `portNumber`

Bertujuan untuk menyimpan port yang digunakan server.

- `homeDirectory`

Bertujuan untuk menyimpan direktori *home* yang digunakan oleh server.

- **Konstruktor**

- `NewMenjanganServer(int portNumber, String homeDirectory)`

Bertujuan untuk menginisialisasi komponen-komponen utama server. Pada tahap awal, sebuah objek `worker` dibuat dengan menerima `homeDirectory` sebagai parameter, sehingga memungkinkan `worker` mengakses file yang dibutuhkan. Selanjutnya, objek `admin` dan `service` diinisialisasi untuk menangani permintaan. Selain itu, sebuah *instance* `httpServer` dibuat dan dikonfigurasi agar dapat mendengarkan pada *port* yang telah ditentukan dan juga `AbstractHandler` ditambahkan ke `httpServer` untuk mengarahkan permintaan HTTP ke *method* yang sesuai.

- **Method**

- `clone()`

Method ini bertujuan untuk membuat salinan baru dari objek `NewMenjanganServer` dengan mempertahankan nilai yang sama untuk variabel `portNumber` dan `homeDirectory`. Dalam kasus di mana proses *cloning* gagal, metode ini akan mencatat pesan kesalahan

- 1 ke dalam *log* global untuk memastikan bahwa kegagalan tersebut tercatat.
2 – **start()** dan **stop()** Kedua metode ini berfungsi untuk mengelola server HTTP. Metode
3 start digunakan untuk menjalankan server sehingga dapat mulai mendengarkan dan
4 memproses permintaan yang masuk. Sebaliknya, metode stop bertugas menghentikan
5 server HTTP, memastikan bahwa semua aktivitas server dihentikan dengan aman.

6 **3.1.1.4 ServiceListener**

7 Kelas ini bertanggung jawab untuk menangani permintaan layanan pada server KIRI. Class ini
8 menerima permintaan HTTP untuk mencari rute dan transportasi terdekat berdasarkan parameter
9 yang diberikan. Pada kelas ini, terdapat 6 konstanta dan 1 atribut. Selain itu, terdapat sebuah
10 konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

11 • **Konstanta**

- 12 – `PARAMETER_START`, `PARAMETER_FINISH`, `PARAMETER_MAXIMUM_WALKING`,
13 `PARAMETER_WALKING_MULTIPLIER`, `PARAMETER_PENALTY_TRANSFER`, dan
14 `PARAMETER_TRACKTYPEID_BLACKLIST`

15 Semua konstanta tersebut digunakan untuk menentukan parameter permintaan.

16 • **Atribut**

- 17 – `worker`

18 Merupakan *instance* dari *class Worker*, yang bertanggung jawab untuk menemukan rute
19 dan transportasi

20 • **Method**

- 21 – `handle(String target, Request baseRequest, HttpServletRequest request,`
22 `HttpServletResponse response)`

23 *Method* ini bertanggung jawab untuk menangani permintaan HTTP dan menghasilkan
24 respons yang sesuai. Dalam prosesnya, variabel *query* digunakan untuk menyimpan
25 string parameter permintaan, sementara *params* adalah objek *Map* yang berisi parameter
26 permintaan yang telah diurai menggunakan *method parseQuery(query)*. Untuk me-
27 nentukan hasil yang akan dikirimkan, *method* ini menggunakan variabel *responseText*
28 untuk menyimpan teks respons dan *responseCode* untuk menyimpan status HTTP yang
29 akan dikembalikan kepada klien.

- 30 – `parseQuery(String query)`

31 *Method* ini bertugas untuk memproses *string query* dan mengonversinya menjadi sebuah
32 objek *Map* yang berisi pasangan kunci dan nilai. Proses dimulai dengan memecah *query*
33 berdasarkan karakter & untuk mendapatkan setiap pasangan kunci dan nilai secara
34 terpisah. Selanjutnya, setiap pasangan dipecah lebih lanjut berdasarkan karakter = untuk
35 memisahkan kunci dari nilainya. Jika *query* yang diterima bernilai *null*, *method* ini akan
36 melemparkan *NullPointerException* sebagai penanganan kesalahan.

37 **3.1.1.5 Worker**

38 Kelas ini bertanggung jawab untuk menangani permintaan routing (pencarian rute) menggunakan
39 algoritma pencarian jalur terpendek. Fungsinya adalah untuk memproses permintaan rute berda-
40 sarkan data graf, dengan mempertimbangkan parameter jarak berjalan kaki, penalti transfer, dan

1 lainnya. Pada kelas ini, terdapat 8 atribut. Selain itu, terdapat konstruktor dan method - method
2 diimplementasikan yang memiliki penjelasan sebagai berikut:

3 • **Atribut**

- 4 – `globalMaximumWalkingDistance`
5 Merepresentasikan jarak maksimal untuk berjalan kaki.
- 6 – `global_maximum_transfer_distance`
7 Merepresentasikan arak maksimal untuk *transfer node*.
- 8 – `globalMultiplierWalking`
9 Berfungsi sebagai faktor pengali jarak berjalan kaki.
- 10 – `globalPenaltyTransfer`
11 Merepresentasikan nilai penalti untuk *transfer node*.
- 12 – `numberOfRequests`
13 Merepresentasikan jumlah permintaan yang diproses.
- 14 – `totalProcessTime`
15 Merepresentasikan total waktu proses (dalam milidetik).
- 16 – `tracks`
17 Merepresentasikan daftar rute (jalur transportasi) yang tersedia.
- 18 – `nodes`
19 Representasi graf dari seluruh *node*.

20 • **Konstruktor**

- 21 – `public Worker(String homeDirectory)`
22 Konstruktor ini membaca file konfigurasi utama yang disebut **MJNSERVE_PROPERTIES**
23 untuk memuat pengaturan yang dibutuhkan. Selanjutnya, data graf jalur diambil dari
24 file konfigurasi tambahan, yaitu **TRACKS_CONF**, untuk membangun struktur jalur yang
25 akan digunakan. Setelah itu, *method linkAngkots()* dijalankan untuk menghubungkan
26 *node-node* angkot, memastikan keterhubungan jalur transportasi dalam graf. Terakhir,
27 metode `cleanUpMemory()` dipanggil untuk membersihkan memori sementara, sehingga
28 efisiensi dan stabilitas sistem tetap terjaga.

29 • **Method**

- 30 – `cleanUpMemory()`
31 *Method* ini berfungsi untuk membersihkan memori yang digunakan selama proses kom-
32 putasi.
- 33 – `readConfiguration(String filename)`
34 *Method* ini berfungsi untuk membaca konfigurasi dari file properti dan menyimpan nilai
35 ke dalam variabel global.
- 36 – `printTracksInfo()`
37 *Method* ini berfungsi untuk membuat ringkasan informasi tentang rute (*track*) dan *node*
38 yang dimuat dalam aplikasi.
- 39 – `findRoute(LatLon start, LatLon finish, Double customMaximumWalkingDistance,`
40 `Double customMultiplierWalking, Double customPenaltyTransfer, Set<String>`
41 `trackTypeIdBlacklist)`
42 *Method* ini dirancang untuk membangun graf virtual sebagai bagian dari proses pencarian

1 rute. Proses dimulai dengan menambahkan *node start* dan *end* ke dalam graf, yang me-
2 wakili titik awal dan tujuan perjalanan. Setelah itu, *node-node* dalam graf dihubungkan
3 menggunakan jarak berjalan kaki untuk mencerminkan kemungkinan pergerakan pejalan
4 kaki. Selanjutnya, algoritma Dijkstra dijalankan untuk menghitung dan menemukan rute
5 terpendek antara *node start* dan *end*. Sebagai hasil akhirnya, langkah-langkah rute yang
6 ditemukan disusun dalam format protokol Kalapa-Dago untuk memberikan panduan
7 perjalanan yang terstruktur dan dapat diinterpretasikan dengan mudah.

8 – **resetStatistics()**

9 *Method* ini bertujuan untuk mereset statistik pemrosesan server. Dalam prosesnya,
10 jumlah permintaan (**numberOfRequests**) diatur ulang menjadi nol untuk menghapus
11 data historis mengenai jumlah permintaan yang telah diterima. Selain itu, waktu
12 pemrosesan total (**totalProcessTime**) juga diatur ulang menjadi nol untuk menghapus
13 catatan akumulasi waktu yang digunakan dalam memproses permintaan.

14 – **getNumberOfRequests()**

15 *Method* ini mengembalikan jumlah permintaan yang telah diproses sejak statistik terakhir
16 diatur ulang.

17 – **getTotalProcessTime()**

18 *Method* ini mengembalikan total waktu pemrosesan semua permintaan dalam detik.

19 – **readGraph(String filename)**

20 *Method* ini dirancang untuk membangun graf dengan membaca data dari file yang berisi
21 informasi lintasan, node, dan koneksi antar node. Proses dimulai dengan membaca file
22 untuk mengambil detail lintasan, node, serta hubungan koneksi di antara keduanya. Graf
23 kemudian dibuat untuk merepresentasikan struktur jaringan yang akan digunakan dalam
24 proses pencarian rute. *Method* ini mengembalikan nilai **true** apabila berhasil dan **false**
25 apabila gagal.

26 – **linkAngkots()**

27 *Method* membangun *K-D Tree* yang digunakan untuk mempermudah pencarian node
28 terdekat. Metode ini juga menghubungkan node transfer yang berada dalam batas jarak
29 transfer maksimum, sehingga memastikan konektivitas antar node sesuai dengan aturan
30 jarak yang telah ditentukan.

31 – **toString()**

32 *Method* ini mengembalikan representasi string dari semua *track* yang dimuat.

33 – **findNearbyTransports(LatLon start, Double customMaximumWalkingDistance)**

34 *Method* ini bertujuan untuk menemukan transportasi terdekat dari lokasi tertentu dengan
35 menghitung jarak dari lokasi awal ke setiap *track* dalam graf. Setelah semua jarak dihitung,
36 *method* ini akan menentukan lintasan dengan jarak minimum yang masih berada dalam
37 batas yang dapat dijangkau dengan berjalan kaki. *Method* ini mengembalikan informasi
38 dari transportasi yang ditemukan dalam bentuk *string*.

1 **3.1.1.6 DataPuller**

2 Kelas ini bertanggung jawab untuk mengambil data jalur dari basis data dan memprosesnya dalam
3 bentuk yang diinginkan. Kelas ini menggunakan JDBC untuk koneksi ke basis data MySQL dan
4 mengubah data jalur menjadi koordinat. Selain itu, kelas ini menambahkan titik-titik virtual untuk
5 memenuhi jarak maksimum tertentu antar titik. Pada kelas ini, terdapat 2 konstanta serta *method* -
6 *method* diimplementasikan yang memiliki penjelasan sebagai berikut:

7 • **Konstanta**

8 – **EARTH_RADIUS**

9 Konstanta tersebut merupakan radius Bumi dalam kilometer dan digunakan untuk
10 menghitung jarak antar titik koordinat.

11 – **MAX_DISTANCE**

12 Konstanta tersebut merupakan jarak maksimum antar titik yang diizinkan, jika jarak
13 antar dua titik melebihi nilai ini, titik-titik virtual akan ditambahkan di antaranya.

14 • **Method**

15 – **pull(File sqlPropertiesFile, PrintStream output)**

16 Berfungsi untuk mengambil data dari tabel tracks di basis data, kemudian menuliskan
17 hasil format jalur dalam format yang ditentukan. *Method* ini memuat data dari file
18 properti, terhubung ke basis data, dan melakukan query untuk mengambil data yang
19 diperlukan. Hasil query diolah dan ditulis ke *output*.

20 – **lineStringToLngLatArray(String wktText)**

21 Mengubah data koordinat dalam format LINESTRING menjadi array LngLatAlt. Ini
22 menghilangkan teks LINESTRING dan tanda kurung, kemudian memecah data menjadi
23 objek koordinat texttLngLatAlt. *Method* ini mengembalikan array koordinat, yang
24 berisi array objek LngLatAlt yang mewakili koordinat LineString.

25 – **computeDistance(LngLatAlt p1, LngLatAlt p2)**

26 Menghitung jarak antara dua titik koordinat. *Method* ini mempertimbangkan kelengkungan
27 bumi dalam perhitungan jaraknya. *Method* ini mengembalikan jarak yang dihitung
28 antara dua titik.

29 – **formatTrack**

30 Mengonversi informasi jalur yang diambil dari basis data ke format konfigurasi yang
31 dibutuhkan. Metode ini mengatur titik transit, menambahkan titik virtual, dan menyusun
32 informasi jalur dalam format konfigurasi yang diinginkan.

33 **3.1.1.7 DataPuller.RouteResult**

34 Merupakan sebuah *static nested class* yang menyimpan hasil akhir dalam format konfigurasi sebagai
35 string **trackInConfFormat**, yang dapat diambil dengan *method* **getTrackInConfFormat()**.

36 **3.1.1.8 DataPullerException**

37 Kelas ini adalah kelas *custom exception* yang dibuat untuk menangani kesalahan khusus yang terjadi
38 saat pemrosesan data dalam kelas DataPuller. Kelas ini memperluas **RuntimeException**, sehingga
39 DataPullerException adalah *unchecked exception* dan tidak memerlukan penanganan eksplisit

1 dengan blok *try-catch* di tempat pemanggilannya. Pada kelas ini, terdapat sebuah konstanta serta
2 konstruktor yang memiliki penjelasan sebagai berikut:

3 • **Konstanta**

4 – **serialVersionUID**

5 Menyimpan ID *serial*. ID ini memastikan data yang disimpan atau dikirimkan tetap
6 cocok dengan versi kelas yang digunakan saat objek tersebut dibaca kembali. Hal ini
7 penting, terutama jika kelas ini mengalami perubahan struktur, agar versi yang berbeda
8 tetap dapat dikenali atau mencegah kesalahan jika struktur sudah tidak cocok.

9 • **Konstruktor**

10 – **DataPullerException(String message)**

11 Konstruktor ini menerima pesan kesalahan dalam bentuk String, yang kemudian dite-
12 ruskan ke konstruktor *superclass RuntimeException* untuk disimpan dan nantinya dapat
13 diambil dengan metode **getMessage()**. Pesan ini bertujuan untuk memberikan informasi
14 yang lebih rinci tentang kesalahan yang terjadi.

15 **3.1.1.9 LatLon**

16 Kelas ini berfungsi untuk merepresentasikan posisi geografis dengan koordinat lintang (*latitude*)
17 dan bujur (*longitude*). Pada kelas ini terdapat metode untuk menghitung jarak antara dua titik
18 koordinat berdasarkan jarak permukaan bumi. Penggunaannya bisa ditemui pada sistem yang
19 memerlukan perhitungan atau pengelolaan data geografis. Pada kelas ini, terdapat 1 konstanta dan
20 2 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki
21 penjelasan sebagai berikut:

22 • **Konstanta**

23 – **EARTH_RADIUS**

24 Menyimpan nilai jari-jari Bumi dalam satuan kilometer (6371.0 km). Konstanta ini
25 digunakan dalam perhitungan jarak antara dua titik geografis.

26 • **Atribut**

27 – **lat**

28 Merepresentasikan nilai lintang suatu titik.

29 – **lon**

30 Merepresentasikan nilai bujur suatu titik.

31 • **Konstruktor**

32 – **LatLon(float lat, float lon)**

33 Konstruktor ini menerima dua parameter, **lat** (*latitude*) dan **lon** (*longitude*), yang
34 disimpan langsung dalam atribut publik kelas.

35 – **LatLon(String latlon)**

36 Konstruktor ini menerima parameter tunggal berupa String dengan format *latitude,longitude*. String ini kemudian dipisah berdasarkan tanda koma (", "), lalu nilai-nilai
37 yang diperoleh diubah menjadi float dan disimpan dalam atribut **lat** dan **lon**.

38 • **Method**

39 – **toString()**

40 *Method* ini mengembalikan representasi string dari objek LatLon, berupa lat lon, yang

menyajikan lintang dan bujur dalam format yang sederhana.

- `distanceTo(LatLon target)`

Method ini menghitung jarak antara objek LatLon saat ini dengan objek LatLon lain (*target*). *Method* ini mengembalikan jarak yang dihitung antara objek LatLon saat ini dan objek LatLon target dalam kilometer.

3.1.1.10 UnrolledLinkedList

Kelas ini merupakan struktur data khusus berbasis linked list yang berfungsi sebagai implementasi daftar berantai (linked list) yang memanfaatkan ArrayList sebagai penyimpanan internal. Pada kelas ini, terdapat 2 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- `internalArray`

Menyimpan elemen-elemen di dalam setiap *node* `UnrolledLinkedList`.

- `nextList`

Referensi ke `UnrolledLinkedList` berikutnya (jika ada) untuk membentuk hubungan antara *node-node* `UnrolledLinkedList`.

- **Konstruktor**

- `UnrolledLinkedList()`

Inisialisasi `UnrolledLinkedList` dengan membuat `internalArray` kosong dan `nextList` sebagai `null`.

- **Method**

- `add(E e)`

Menambahkan elemen *e* ke `internalArray` pada `UnrolledLinkedList` saat ini.

- `iterator()`

Mengembalikan iterator (`FastLinkedListIterator`) yang dapat digunakan untuk menavigasi elemen-elemen di dalam `UnrolledLinkedList`.

- `addAll(UnrolledLinkedList<E> elements)`

Menambahkan semua elemen dari sebuah objek `UnrolledLinkedList<E>` lain (*elements*) ke dalam *list* yang sedang diproses (*this*).

- `size()`

Menghitung total elemen di seluruh `UnrolledLinkedList`, termasuk yang ada di `nextList` dan mengembalikan nilai total elemen tersebut.

- `cleanUpMemory()`

Mengurangi ukuran `internalArray` ke jumlah elemen yang bertujuan untuk mengurangi penggunaan memori.

3.1.1.11 UnrolledLinkedList.FastLinkedListIterator

Inner class ini merupakan iterator yang digunakan untuk menjelajahi atau mengambil elemen-elemen dalam `UnrolledLinkedList` satu per satu secara berurutan. Dalam *inner class* ini juga terdapat 3 atribut serta beberapa *method* yang diimplementasikan.

- **Atribut**

```
1   - currentList
2       Menunjuk pada UnrolledLinkedList saat ini yang sedang diiterasi.
3   - currentIndex
4       Posisi indeks di internalArray pada currentList.
5   - globalIndex
6       Posisi indeks global yang melacak elemen secara keseluruhan di seluruh UnrolledLinke-
7       dList.
```

- 8 • **Method**
- 9 - **hasNext()**
10 Mengecek apakah masih ada elemen berikutnya di dalam **internalArray** atau di
11 **nextList** dan mengembalikan nilai **true** apabila masih ada dan **false** apabila tidak.
- 12 - **next()**
13 Mengembalikan elemen berikutnya dalam iterasi. Jika sudah mencapai akhir **internalArray**,
14 beralih ke **nextList**.
- 15 - **hasPrevious()** dan **previous()**
16 Melempar **UnsupportedOperationException**.
- 17 - **nextIndex()** dan **previousIndex()**
18 Mengembalikan indeks global berikutnya atau sebelumnya.
- 19 - **remove()**
20 Melempar **UnsupportedOperationException**.
- 21 - **set(E e)**
22 Mengganti elemen di **currentIndex** dengan elemen baru **e**.
- 23 - **add(E e)**
24 Menambahkan elemen **e** ke dalam **UnrolledLinkedList**.

25 3.1.1.12 Track

26 Kelas Track merepresentasikan sebuah jalur transportasi umum, yaitu jalur angkot atau Transjakarta.
27 Kelas ini menghubungkan node-node dalam sebuah graf menggunakan jalur yang spesifik untuk
28 transportasi tersebut. Pada kelas ini, terdapat 4 atribut. Selain itu, terdapat konstruktor dan
29 *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- 30 • **Atribut**
- 31 - **trackTypeId**
32 Menyimpan jenis jalur transportasi, seperti "angkot" atau "transjakarta".
- 33 - **trackId**
34 Menyimpan ID spesifik dari jalur transportasi, seperti "kalapaledeng" atau "cicaheumci-
35 royom".
- 36 - **trackPath**
37 Menyimpan daftar node (**GraphNode**) yang membentuk jalur.
- 38 - **penalty**
39 Menyimpan nilai penalti yang akan digunakan untuk memengaruhi biaya jalur saat
40 algoritma pencarian jalur dijalankan.

- Konstruktor

– `public Track(String fullyQualifiedTrackId)`

Menginisialisasi sebuah objek `Track` menggunakan ID jalur lengkap dalam format `trackTypeId.trackId`.

- Method

– `getTrackId()`

Mengembalikan ID jalur (`trackId`).

– `getTrackTypeId()`

Mengembalikan jenis jalur (`trackTypeId`).

– `getPenalty()`

Mengembalikan nilai penalti.

– `setPenalty(double penalty)`

Mengatur nilai penalti untuk suatu jalur.

– `toString()`

Mengembalikan representasi string dari objek `Track`, meliputi jenis jalur, ID jalur, dan jumlah node.

– `addNode(GraphNode node)`

Menambahkan sebuah node ke dalam jalur (`trackPath`).

– `getNode(int idx)`

Mengambil dan mengembalikan node berdasarkan indeks dalam `trackPath`.

– `getSize()`

Mengembalikan jumlah node dalam jalur.

3.1.1.13 GraphEdge

Kelas ini mewakili sebuah *edge* (sisi) dalam struktur data graf. *Edge* ini digunakan dalam konteks perhitungan jalur atau algoritma graf lainnya. Pada kelas ini, terdapat 2 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- Atribut

– `node`

Menunjukkan simpul (*node*) yang dituju oleh *edge* ini. *Edge* ini menghubungkan dua *node* dengan arah tertentu dari satu *node* ke *node* yang lain, dan *node* menunjukkan simpul akhir yang menjadi tujuan.

– `weight`

Menunjukkan bobot dari *edge* ini. Dalam konteks jalur terpendek atau algoritma lainnya, bobot ini bisa berarti jarak, waktu tempuh, atau biaya yang diperlukan untuk bergerak dari *node* asal menuju *node* yang ditunjuk oleh *node*.

- Konstruktor

– `GraphEdge(int node, double weight)`

Konstruktor ini mengambil dua parameter, `node` dan `weight`, untuk menginisialisasi sebuah *edge*. Ini berarti bahwa setiap *edge* akan memiliki *node* tujuan dan bobotnya sendiri yang menunjukkan biaya atau jarak menuju *node* tersebut.

1 • **Method**

2 – `int getNode()`

3 *Method* ini mengembalikan nilai *node* yang dituju oleh *edge* ini yang bertujuan untuk
4 mendapatkan informasi tentang *node* tujuan.

5 – `double getWeight()`

6 *Method* ini mengembalikan bobot dari *edge* saat ini, yang dapat digunakan dalam
7 perhitungan atau penelusuran jalur dalam graf.

8 **3.1.1.14 GraphNode**

9 Kelas ini berfungsi untuk merepresentasikan sebuah *node* (simpul) dalam sebuah graf. *Node* ini
10 digunakan untuk menyimpan informasi tentang lokasi geografis, koneksi ke *node* lain, dan atribut
11 tertentu yang berhubungan dengan transportasi umum. Kelas ini dapat digunakan untuk membuat
12 struktur graf yang akan mencerminkan rute dan jalur transportasi, sehingga memudahkan pemodelan
13 dalam algoritma pencarian rute. Pada kelas ini, terdapat 4 atribut. Selain itu, terdapat konstruktor
14 dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

15 • **Atribut**

16 – `location`

17 Atribut ini menyimpan lokasi *node* dalam bentuk koordinat lintang dan bujur. `location`
18 merupakan objek dari `LatLon`, yang menyimpan posisi geografis dalam satuan derajat.

19 – `edges`

20 Atribut ini menyimpan daftar dari semua *edge* (sisi) atau jalur keluar dari node ini. Jalur
21 keluar ini mengarah ke node lain, menciptakan koneksi dalam graf. `edges` diimplementa-
22 sikan menggunakan `UnrolledLinkedList`.

23 – `track`

24 Atribut ini berfungsi sebagai referensi balik ke informasi rute (track) dari node. Dengan
25 atribut ini, sistem dapat mengetahui rute atau *track* mana yang terkait dengan node
26 tersebut.

27 – `isTransferNode`

28 Atribut ini menunjukkan apakah *node* tersebut adalah *node transfer*. Dalam konteks
29 transportasi umum, sebuah *node transfer* memungkinkan pengguna untuk turun atau
30 naik kendaraan umum dari node tersebut. Jika bernilai *true*, berarti *node* ini adalah
31 *node transfer*.

32 • **Konstruktor**

33 – `GraphNode(LatLon location, Track track)`

34 Konstruktor ini membuat instance baru dari kelas `GraphNode` dengan lokasi (`LatLon`)
35 dan referensi rute (`track`). Saat diinisialisasi, atribut `isTransferNode` diatur ke *false*
36 secara *default*, dan `edges` diinisialisasi sebagai daftar kosong dari objek `GraphEdge`.

37 • **Method**

38 – `getEdges()`

39 Mengembalikan daftar `edges`, yaitu daftar sisi keluar dari *node* ini. Daftar ini dapat
40 digunakan untuk mengetahui semua koneksi dari *node* ke *node* lain dalam graf.

41 – `push_back(int node, float weight)`

1 Method ini menambahkan sisi baru ke daftar *edges* dengan memasukkan informasi *node*
 2 tujuan dan bobotnya. Bobot (*weight*) mencerminkan jarak atau biaya perjalanan ke
 3 *node* lain.

4 – **link(GraphNode nextNode)**
 5 Method ini menghubungkan *node* ini dengan *node* lain (*nextNode*) dengan menambahkan
 6 semua sisi (*edges*) dari *node* berikut ke daftar sisi (*edges*) *node* ini.

7 – **getLocation()**
 8 Mengembalikan lokasi geografis dari *node* ini dalam bentuk objek **LatLon**

9 – **getTrack()**
 10 Mengembalikan referensi rute (*track*) yang terkait dengan *node* ini. Ini memungkinkan
 11 akses ke informasi rute dari *node*.

12 – **toString()**
 13 Mengembalikan representasi teks dari *node* ini, yang mencakup informasi lokasi dan
 14 status apakah *node* ini adalah *node transfer* atau bukan.

15 – **setTransferNode(boolean b)**
 16 Mengatur apakah *node* ini merupakan *node transfer* berdasarkan parameter . Jika *b*
 17 bernilai *true*, maka *node* akan dianggap sebagai *node transfer*.

18 – **isTransferNode()**
 19 Mengembalikan nilai *boolean* yang menunjukkan apakah *node* ini adalah *node transfer*
 20 (*true*) atau bukan (*false*).

21 3.1.1.15 Graph

22 Kelas ini adalah kelas yang merepresentasikan sebuah graf, yaitu kumpulan dari *node-node*
 23 (**GraphNode**). Dengan metode **rangeSearch**, kelas ini dapat mendukung pencarian node dalam
 24 radius tertentu, yang berguna dalam pemodelan rute. Kelas ini tidak memiliki atribut selain dari
 25 **ArrayList** yang diwarisi, karena kelas ini mewarisi semua fungsi dari **ArrayList** dan berfungsi
 26 untuk menyimpan node-node dalam bentuk **GraphNode**. Terdapat 2 konstruktor dan *method - method* -
 27 diimplementasikan yang memiliki penjelasan sebagai berikut:

- 28 • **Konstruktor**

- 29 – **Graph()** Konstruktor ini membuat sebuah objek **Graph** tanpa menentukan kapasitas awal.
 30 Dengan kata lain, kapasitas akan ditentukan berdasarkan elemen yang ditambahkan.
- 31 – **Graph(int capacity)**
 32 Konstruktor ini membuat objek **Graph** dengan kapasitas awal tertentu, sesuai dengan
 33 nilai **capacity** yang diberikan.

- 34 • **Method**

- 35 – **rangeSearch(LatLon center, double distance)**
 36 Method ini berfungsi untuk mencari semua node yang berada dalam jangkauan atau
 37 radius tertentu dari suatu titik pusat. Parameter **center** adalah objek **LatLon** yang
 38 menyatakan titik pusat dari area pencarian, dan **distance** adalah radius pencarian
 39 dalam satuan kilometer.
 40 Pada implementasi awal, *method* ini menggunakan perulangan sederhana melalui setiap
 41 **GraphNode** dalam graf (**for(GraphNode node : this)**). Untuk setiap *node* dalam graf,

1 *method* ini menghitung jarak antara **center** dan lokasi *node* (*node.location*) dengan
2 menggunakan metode **distanceTo** pada objek **LatLon**. Jika jarak antara **center** dan
3 *node* tersebut lebih kecil atau sama dengan **distance**, maka *node* tersebut dianggap
4 berada dalam jangkauan, dan dimasukkan ke dalam *list*. Metode ini mengembalikan *list*,
5 yaitu daftar **GraphNode** yang berada dalam jangkauan dari **center**.

6 3.1.1.16 Dijkstra

7 Kelas ini adalah implementasi dari algoritma Dijkstra yang digunakan untuk mencari jarak terpendek
8 antara dua titik dalam sebuah graf. Algoritma ini bekerja dengan mencari rute dengan bobot
9 terendah atau rute dengan jarak minimum antara node awal dan node tujuan. Pada kelas ini,
10 terdapat 1 konstanta dan 10 atribut. Selain itu, terdapat konstruktor dan *method - method*
11 diimplementasikan yang memiliki penjelasan sebagai berikut:

- 12 • Konstanta

- 13 – **DIJKSTRA_NULLNODE**
14 Nilai konstan -1 untuk menandakan node yang tidak valid.

- 15 • Atribut

- 16 – **graph**
17 Daftar *node* dalam graf yang merepresentasikan jalur.
- 18 – **startNode** dan **finishNode**
19 Menyimpan indeks *node* awal dan akhir dari pencarian.
- 20 – **nodeInfoLinks**
21 Array **NodeInfo** yang menyimpan informasi jarak terdekat untuk setiap node.
- 22 – **nodesMinHeap**
23 Array **NodeInfo** yang menyimpan node dalam urutan jarak terpendek untuk mendukung
24 operasi heap dalam algoritma Dijkstra.
- 25 – **heapsize**
26 Ukuran heap, menandakan jumlah node yang ada dalam heap.
- 27 – **numOfNodes**
28 Jumlah total node dalam graf.
- 29 – **memorySize**
30 Ukuran memori yang digunakan.
- 31 – **multiplierWalking** dan **penaltyTransfer**
32 Faktor pengali dan penalti yang digunakan untuk menghitung bobot tambahan pada
33 node yang terkait dengan jalur pejalan kaki atau transfer angkot.

- 34 • Method

- 35 – **runAlgorithm(Set<String> trackTypeIdBlacklist)**
36 Merupakan *method* utama untuk menjalankan algoritma Dijkstra yang dirancang untuk
37 menemukan jalur terpendek dalam sebuah graf. Langkah pertama adalah menginisialisasi
38 setiap objek **NodeInfo** dengan jarak awal bernilai **POSITIVE_INFINITY** dan menetapkan
39 jarak awal *node* sumber (**startNode**) menjadi 0. Setelah itu, struktur **nodesMinHeap**
40 diatur ulang menggunakan metode **heapify** untuk mengurutkan *node* berdasarkan jarak
terpendek. Proses pencarian dimulai dengan mengambil *node* saat ini (**currentNode**) dari

1 heap, dan pencarian dihentikan jika *node* tersebut adalah *node* tujuan (*finishNode*).
 2 Selama iterasi, untuk setiap objek **GraphEdge** yang terhubung dengan *currentNode*,
 3 jarak ke *node* tujuan dihitung menggunakan *method calculateWeight*. Jika jarak
 4 baru yang dihitung lebih pendek daripada jarak yang tercatat sebelumnya, dan tidak
 5 termasuk dalam **trackTypeIdBlacklist**, maka jarak tersebut diperbarui. Selanjutnya,
 6 node dengan jarak yang lebih kecil dipindahkan ke posisi yang sesuai dalam heap
 7 menggunakan *method heapPercolateUp*. Setelah semua langkah selesai, *method* ini akan
 8 mengembalikan jarak terpendek ke *node* tujuan. Jika tidak ada jalur yang tersedia, nilai
 9 **POSITIVE_INFINITY** akan dikembalikan untuk menandakan kegagalan menemukan jalur.

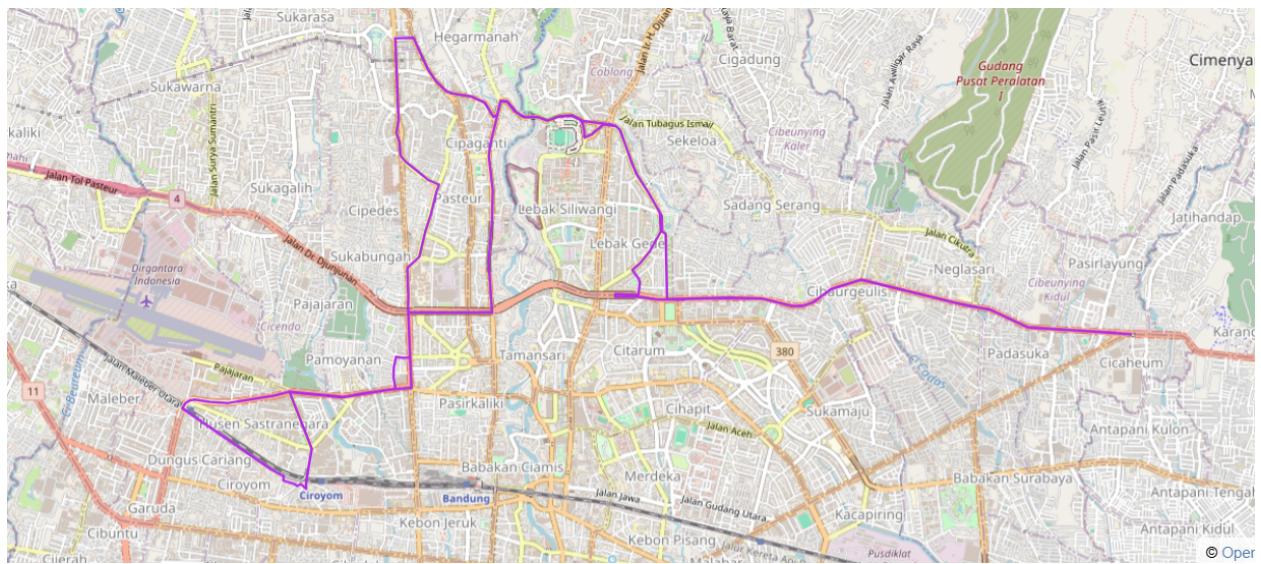
- 10 – **getParent(int node)**
 Mengembalikan *parent* (*node* asal) dari *node* yang dimaksud dalam rute terpendek.
- 11 – **getDistance(int node)**
 Mengembalikan jarak dari *node* awal ke *node* yang diminta.
- 12 – **calculateWeight(NodeInfo currentNode, GraphEdge edge)**
 Method ini bertugas menghitung bobot perjalanan dari *currentNode* ke *node* tujuan
 melalui sebuah *edge*. Perhitungan bobot dilakukan berdasarkan beberapa kondisi. Jika
 salah satu dari *node* tersebut merupakan jalur pejalan kaki, bobot dihitung dengan mene-
 rapkan nilai **multiplierWalking** untuk merepresentasikan jarak berjalan kaki. Jika node
 tersebut melibatkan *transfer* angkot, parameter **penaltyTransfer** akan ditambahkan ke
 bobot perjalanan. Namun, jika perjalanan tetap berada dalam angkot yang sama, bobot
 dihitung berdasarkan nilai bobot dari objek **GraphEdge** serta penalti yang berlaku pada
 jalur yang relevan. *Method* ini mengembalikan nilai Double yang mewakili bobot yang
 telah dihitung.
- 13 – **heapPercolateDown(int index)**
 Menjaga agar heap tetap terurut setelah penghapusan node dengan jarak terpendek.
- 14 – **heapPercolateUp(int index)**
 Memperbarui posisi node dalam heap setelah perubahan jarak.
- 15 – **heapDeleteMin()**
 Menghapus node dengan jarak terpendek dari heap, dan menyesuaikan urutan heap.
 Method ini mengembalikan variabel **ret**, yang berisi objek **NodeInfo**.
- 16 – **getString(int node)**
 Mengembalikan string dari **NodeInfo**.

33 3.1.1.17 Dijkstra.NodeInfo

34 Merupakan sebuah *static nested class* yang menyimpan data untuk setiap node, seperti **baseIndex**,
 35 **heapIndex**, **distance**, dan **parent**. Selain itu, diimplementasikan juga *method* **toString()** yang
 36 mengembalikan nilai string dari data-data tersebut yang telah diformat.

37 3.1.2 Pemodelan Graf

38 Pada bagian ini, akan dilakukan analisis mengenai bagaimana rute-rute yang terdapat dalam tabel
 39 "tracks" pada database dapat dimodelkan menjadi sebuah graf oleh NewMenjangan. Gambar 3.2
 40 merupakan visualisasi GIS (*Geographic Information System*) dari rute angkot cicaheum-ciroyom.



Gambar 3.2: Rute Ciroyom – Cicaheum

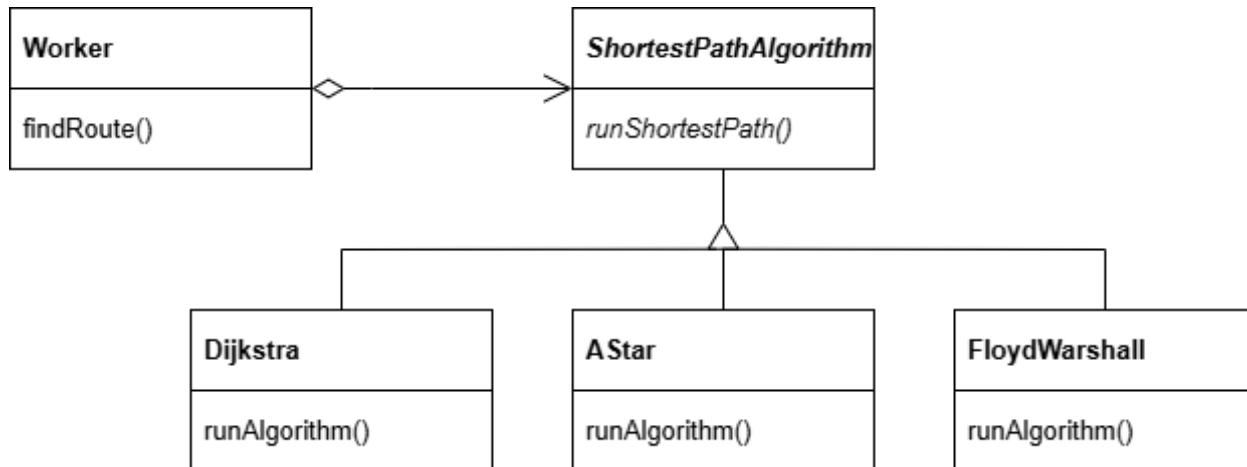
- 1 Pada saat NewMenjangan dijalankan data dari tabel "tracks" akan diambil menggunakan me-
 2 thod `pull()` pada kelas DataPuller 3.1.1.6 dalam format LineString 2.3.1. Kemudian format
 3 data diubah menjadi bentuk array `LngLatAlt` (*longitude* dan *latitude*) menggunakan method
 4 `lineStringToLngLatArray` yang juga terdapat pada kelas DataPuller 3.1.1.6. Dari Titik-titik
 5 yang terdapat pada array `LngLatAlt` akan dimodelkan menjadi graf dengan menjadikan setiap
 6 titik menjadi sebuah node dan akan dihubungkan dengan edge antara nodenya, selain itu juga
 7 apabila ada dua titik dari rute angkot berbeda atau disebut juga *transferNode* yang jaraknya
 8 dibawah dari konstanta yang telah ditentukan, maka akan dibuatkan juga sebuah edge untuk
 9 menghubungkannya. Pemodelan graf tersebut menggunakan method-method pada kelas Worker
 10 3.1.1.5 yang juga memanfaatkan method-method pada kelas Graph 3.1.1.15, GraphNode 3.1.1.14,
 11 LatLon 3.1.1.9, dan Track 3.1.1.12.

12 3.2 Analisis Sistem Usulan

- 13 Analisis ini dilakukan terhadap sistem usulan yang bertujuan untuk meningkatkan fleksibilitas
 14 dan efisiensi dalam proses pencarian rute terpendek pada perangkat lunak KIRI. Sistem yang
 15 diusulkan mengadopsi strategy pattern sebagai pola desain untuk memisahkan logika implementasi
 16 algoritma shortest path, sehingga memungkinkan penggantian atau penambahan algoritma dengan
 17 lebih mudah. Selain algoritma Dijkstra yang telah diterapkan pada sistem saat ini, sistem usulan
 18 juga mengimplementasikan dua algoritma tambahan, yaitu Floyd-Warshall dan A*. Penambahan
 19 kedua algoritma ini bertujuan untuk memperluas cakupan kebutuhan kasus penggunaan yang
 20 beragam, di mana Floyd-Warshall cocok untuk penghitungan semua pasangan titik, sedangkan A*
 21 menawarkan efisiensi untuk pencarian rute dengan heuristik tertentu. Dengan demikian, sistem
 22 usulan diharapkan dapat mengatasi keterbatasan sistem yang ada, yang saat ini hanya menggunakan
 23 satu algoritma dan belum mengadopsi pola desain modular seperti strategy pattern.

1 3.2.1 Implementasi Strategy Pattern

2 Implementasi strategy pattern pada NewMenjangan bertujuan untuk memberikan KIRI kebebasan
 3 untuk memilih dan menentukan strategi yang akan dipilih, dalam kasus ini adalah algoritma *shortest*
 4 *path*. Selain itu, juga memberikan fleksibilitas yang tinggi serta memberikan kemudahan dalam
 5 menambahkan atau mengganti algoritma ketika proses pengembangan perangkat lunak, seperti
 yang telah dijelaskan pada subbab 2.2.



Gambar 3.3: Struktur Strategy Pattern

6
 7 Gambar 3.3 menunjukkan struktur strategy pattern yang dirancang untuk implementasi yang akan
 8 dilakukan dalam tugas akhir ini. Gambar 3.3 merupakan lanjutan dari gambar 2.3 yang hanya
 9 menggambarkan struktur strategy pattern secara umum. Gambar 3.3 menjelaskan bagaimana
 10 strategy pattern diimplementasikan dengan algoritma Dijkstra, A*, dan Floyd-Warshall sebagai
 11 *concrete strategy*. Kemudian, kelas ShortestPathAlgorithm sebagai *Strategy* dan kelas Worker sebagai
 12 *Context*.

13 3.2.2 Implementasi Algoritma A* dan Algoritma Floyd-Warshall

14 Sistem usulan dirancang untuk meningkatkan performa dan fleksibilitas dalam penentuan rute
 15 terpendek dengan mengimplementasikan algoritma A* dan Floyd-Warshall. Pada sistem saat ini,
 16 hanya algoritma Dijkstra yang telah diimplementasikan. Algoritma A* dan Floyd-Warshall juga
 17 sama seperti algoritma Dijkstra yang termasuk algoritma *shortest path* yang digunakan untuk
 18 mencari rute terdekat, seperti yang sudah dijelaskan pada subbab 2.5.3 dan 2.5.2. Perubahan akan
 19 dilakukan pada NewMenjangan untuk mengimplementasikan algoritma A* dan Floyd-Warshall,
 20 yaitu dengan menambahkan 2 *class* java baru dengan nama Dijkstra.java dan AStar.java.

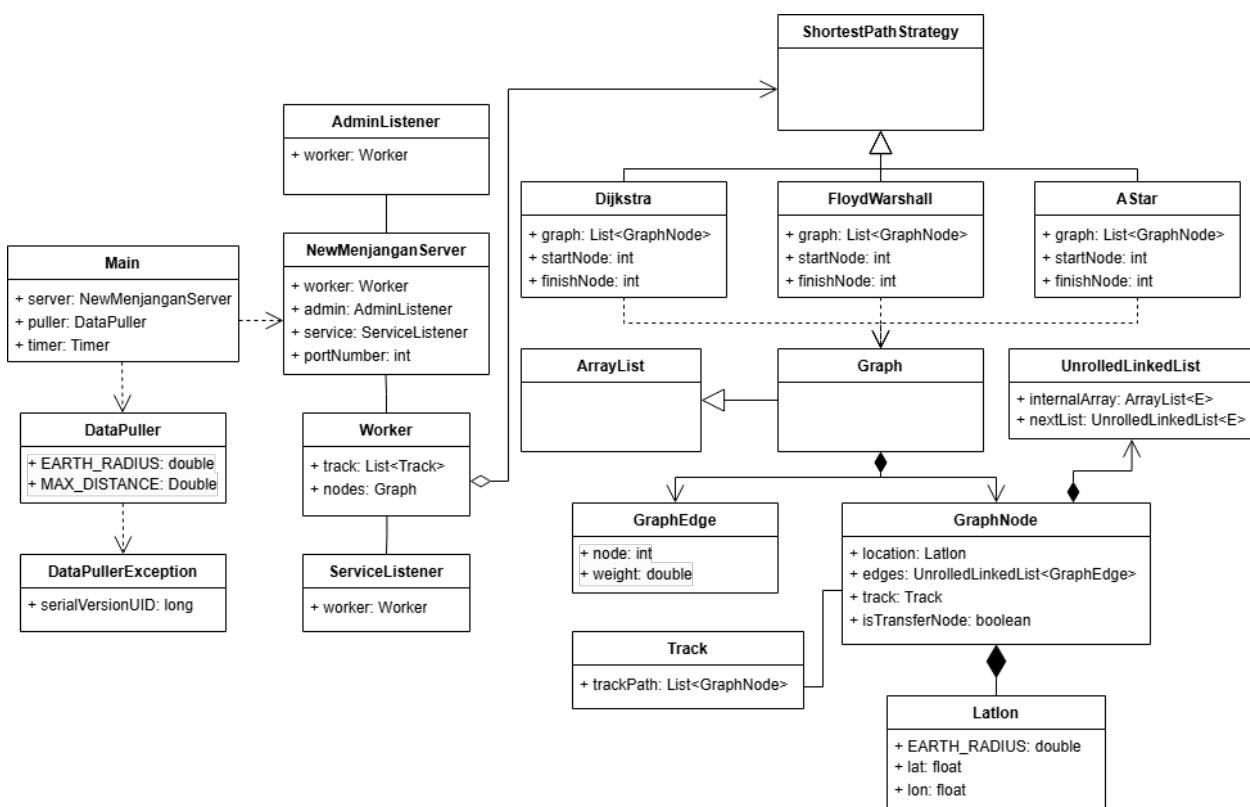
1

BAB 4

2

PERANCANGAN

3 4.1 Rancangan Diagram Kelas



Gambar 4.1: Rancangan Diagram Kelas

DAFTAR REFERENSI

- [1] Nugroho, P. A. dan Natali, V. (2017) Open sourcing proprietary application case study: Kiri website. *IJNMT (International Journal of New Media Technology)*, 4, 82–86.
- [2] Gamma, E., Helm, R., Johnson, R., dan Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
- [3] Version 8.4 (2024) *MySQL 8.4 Reference Manual - Including MySQL NDB Cluster 8.4*. Oracle Corporation. Austin, USA.
- [4] Diestel, R. (2017) *Graph Theory*, 5th edition. Springer, Berlin, Heidelberg.
- [5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., dan Stein, C. (2009) *Introduction to Algorithms*, 3rd edition. The MIT Press, Cambridge, MA.
- [6] Russell, S. dan Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, Upper Saddle River, NJ.
- [7] Pressman, R. S. dan Maxim, B. R. (2019) *Software Engineering: A Practitioner's Approach*, ninth edition. McGraw-Hill Education, New York, NY, USA.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[i] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = (--aaa + &dcaa) / (bbb++ - ccc % 2 );
14             strcpy(a,"hello_@?");
15         }
16     count = ~mask | 0x00FF00AA;
17 }
18
19 // Fonts for Displaying Program Code in LATEX
20 // Adrian P. Robson, nepsweb.co.uk
21 // 8 October 2012
22 // http://nepsweb.co.uk/docs/progfonts.pdf
23

```

Kode A.2: MyCode.java

```

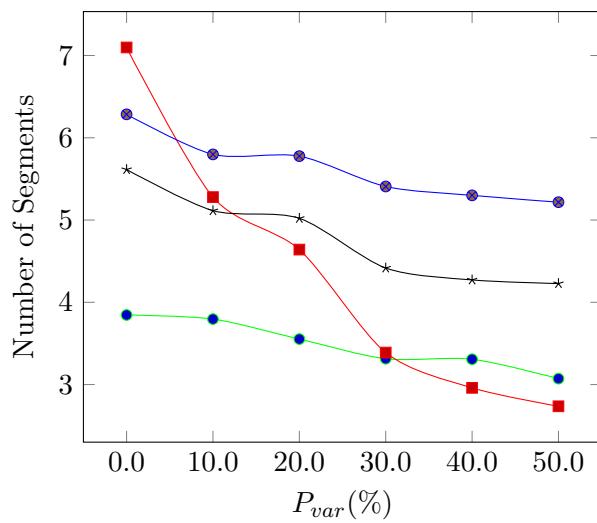
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                         //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35}
36

```

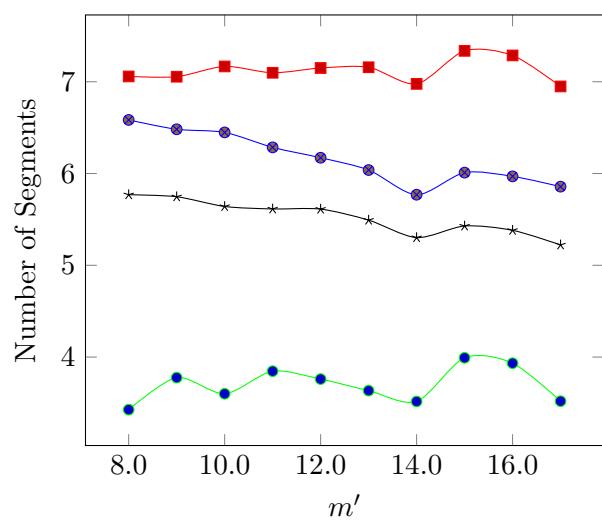

LAMPIRAN B

HASIL EKSPERIMENT

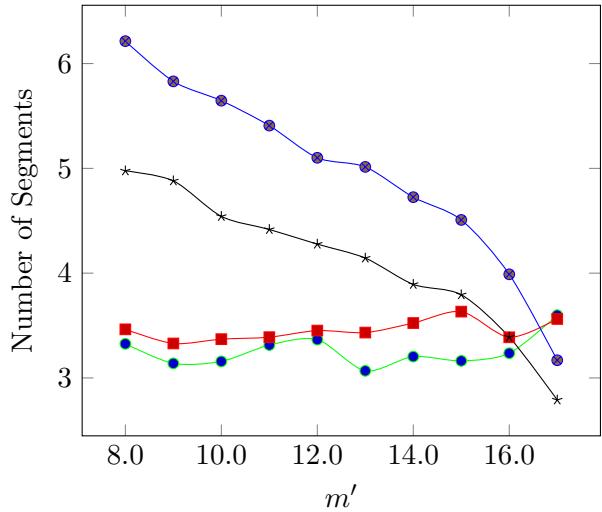
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



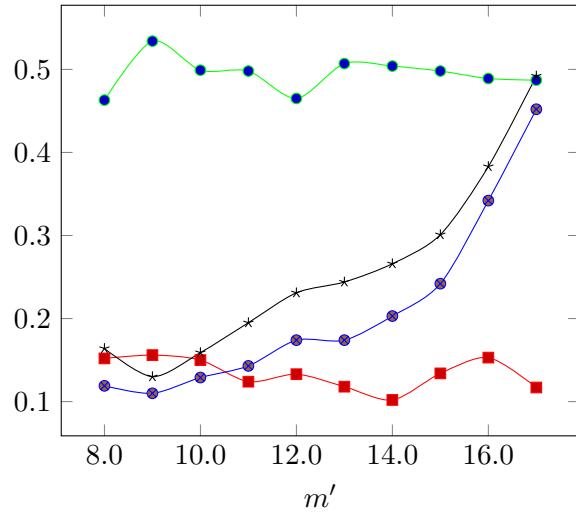
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4