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

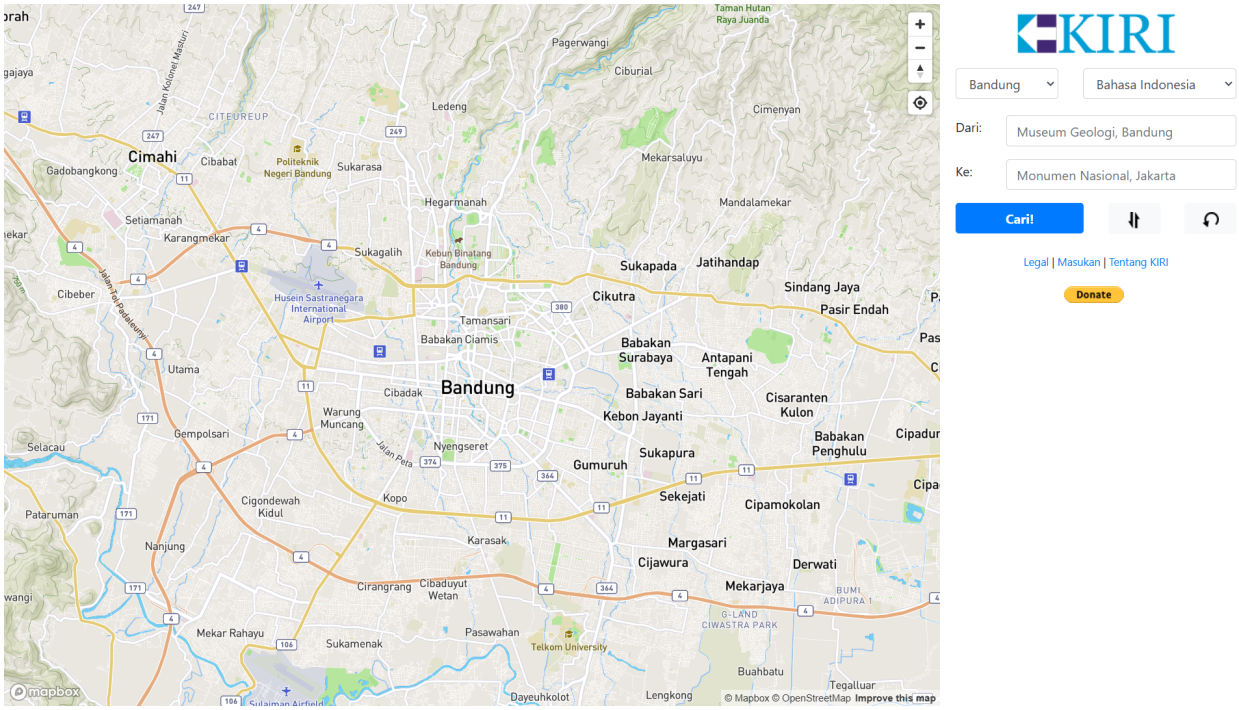
DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Perangkat lunak KIRI (lihat Gambar 1.1) adalah perangkat lunak berbasis web yang dirancang untuk membantu pengguna menemukan rute perjalanan menggunakan angkot. Pada perangkat lunak KIRI, pengguna dapat memasukkan titik awal perjalanan dan titik tujuan. KIRI kemudian akan mencarikan berbagai alternatif rute angkot yang bisa digunakan untuk mencapai tujuan tersebut. KIRI akan memberikan informasi mengenai langkah-langkah yang harus ditempuh oleh pengguna yang akan berpergian dari satu tempat ke tempat tujuannya, mulai dari seberapa jauh pengguna harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun angkot tersebut, seberapa jauh lagi pengguna harus berjalan sampai ke lokasi tujuan, dan seberapa lama estimasi waktu perjalanan yang akan ditempuh.



Gambar 1.1: Tampilan halaman perangkat lunak KIRI

Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Bagian frontend, yang dinamakan Tirtayasa, dibangun menggunakan bahasa pemrograman PHP dan mengandalkan basis data MySQL untuk menyimpan serta mengelola data. Selain itu, Tirtayasa juga menggunakan framework CodeIgniter 3. Saat menerima permintaan pencarian, Tirtayasa meneruskannya ke bagian backend, yaitu NewMenjangan. Hasil dari NewMenjangan kemudian diformat agar dapat dibaca dengan baik oleh pengguna. Bagian ini diimplementasikan dalam bahasa pemrograman Java dan berperan penting dalam perhitungan rute optimal.

NewMenjangan merupakan program daemon yang berjalan secara otomatis saat server dinyalakan dan terus beroperasi hingga server dimatikan. Daemon sendiri adalah program komputer yang berjalan di latar belakang dan tidak berinteraksi langsung dengan pengguna[1]. Pada saat eksekusi, NewMenjangan terhubung ke basis data MySQL untuk mengambil data rute angkot yang tersimpan dalam format *LineString*. *LineString* adalah salah satu tipe data geometris dalam MySQL yang mewakili satu atau lebih segmen garis yang terhubung. *LineString* terdiri dari urutan titik (point) yang membentuk jalur atau lintasan[2]. Setiap titik pada *LineString* merepresentasikan lokasi potensial untuk penumpang naik atau turun. Dari data tersebut, NewMenjangan membangun *weighted graph* dalam memori (RAM) dalam bentuk *adjacency list* dan melakukan prakomputasi. Setiap titik pada LineString menjadi *node*, dan antara titik ke-i dan titik ke-(i+1) dihubungkan dengan *edge*. Jika ada dua titik dari rute angkot berbeda yang berdekatan (jarak di bawah konstanta tertentu), maka dibuatkan juga *edge*, yang menunjukkan kemungkinan seseorang dapat turun dari suatu angkot dan naik ke angkot lainnya untuk meneruskan perjalanan.

Saat NewMenjangan menerima permintaan pencarian dari titik A ke titik B, kedua titik tersebut dijadikan *node* sementara, dan dibuatkan *edge* sementara ke *node-node* yang sudah ada sebelumnya, jika jaraknya di bawah konstanta tertentu. Pencarian jarak terdekat pada graf tersebut dilakukan menggunakan algoritma Dijkstra versi teroptimasi (*priority queue* dengan struktur data *heap*). Proses ini dapat dilakukan secara paralel dengan aman (*thread-safe*) tanpa mengubah graf utama. Pada saat ini algoritma yang digunakan KIRI masih terikat dengan algoritma Dijkstra. Oleh karena itu, pada tugas akhir ini akan diimplementasikan algoritma lainnya, yaitu algoritma A-star dan Floyd Warshall sebagai *concrete strategy*. Selain itu, akan dilakukan juga penerapan arsitektur kelas *strategy pattern* sehingga aplikasi KIRI akan menjadi lebih fleksibel dalam pemilihan algoritma *shortest path* yang akan digunakan dan juga memudahkan apabila akan dilakukan perubahan atau perbaikan pada suatu algoritma yang digunakan.

1.2 Rumusan Masalah

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

[1] https://www.ibm.com/docs/en/aix/7.1?topic=processes-
[2] https://dev.mysql.com/doc/refman/8.4/en/gis-linestring-property-functions.html

1.3 Tujuan

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

1.4 Batasan Masalah

...

1.5 Metodologi

Metodologi yang akan digunakan dalam pembuatan tugas akhir ini adalah sebagai berikut:

1. Melakukan eksplorasi fungsi-fungsi dan cara kerja perangkat lunak KIRI.
2. Mempelajari modul-modul yang terdapat pada Tirtayasa dan NewMenjangan.
3. Mempelajari bahasa pemrograman PHP dan framework CodeIgniter 3.
4. Melakukan studi literatur mengenai penerapan arsitektur kelas *strategy pattern*.
5. Mempelajari cara kerja algoritma Dijkstra, A-star, dan Floyd Warshall.
6. Mengubah implementasi algoritma Dijkstra yang sudah ada ke dalam *strategy pattern*.
7. Mengimplementasikan algoritma A-star dan Floyd Warshall.
8. Melakukan pengujian dan eksperimen.
9. Menulis dokumen tugas akhir.

1.6 Sistematika Pembahasan

Tugas akhir ini akan disusun menjadi beberapa bab sebagai berikut:

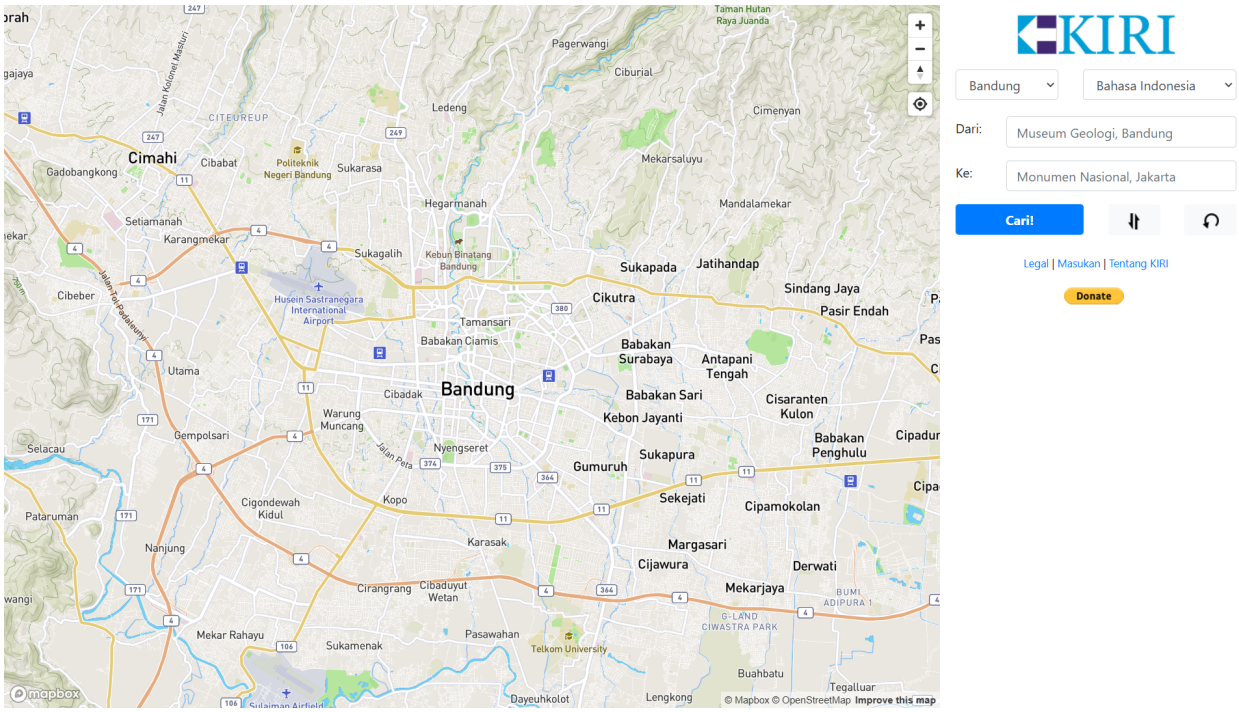
- **Bab 1:** Pendahuluan
  Bab ini berisi latar belakang, rumusan masalah,tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
- **Bab 2:** Landasan Teori ...
- **Bab 3:** Analisis ...

BAB 2

LANDASAN TEORI

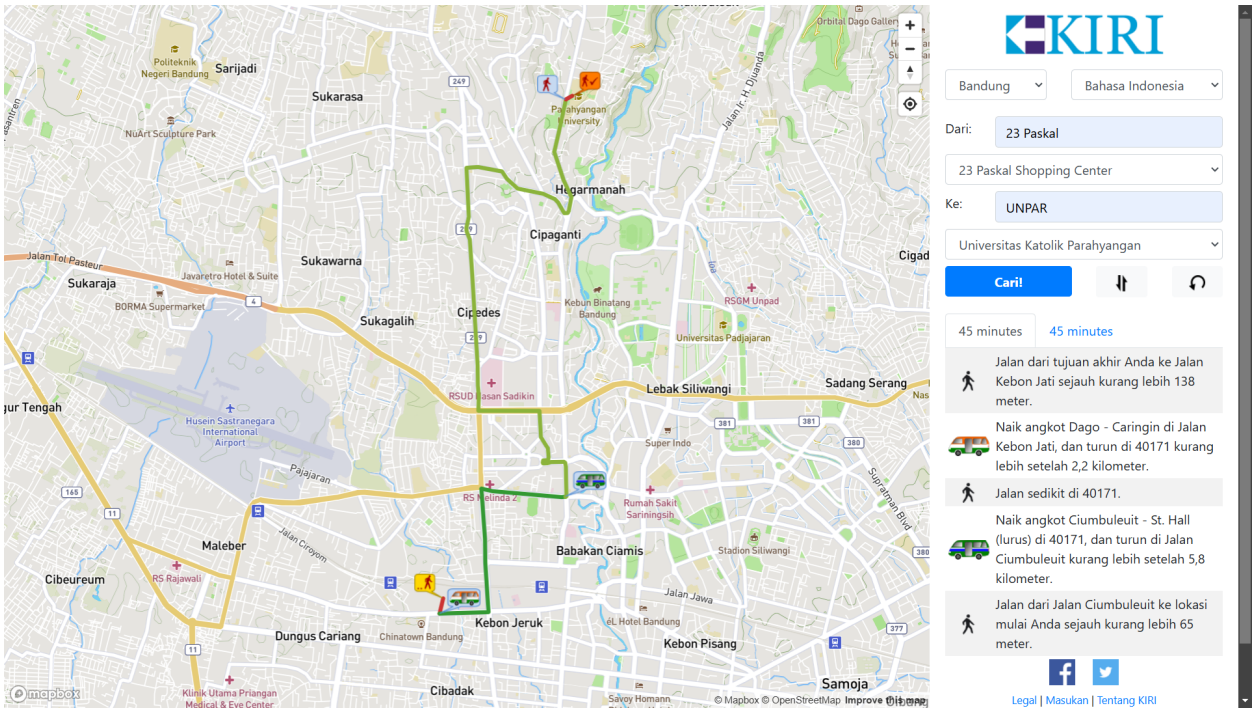
2.1 KIRI

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



Gambar 2.1: Tampilan awal perangkat lunak KIRI

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



Gambar 2.2: Tampilan perangkat lunak KIRI, setelah menerima masukan

Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Yang pertama, yaitu Tirtayasa yang merupakan bagian *frontend* dari KIRI, dan bertanggung jawab sebagai antarmuka pengguna untuk browser web. Komponen ini mengubah nama tempat yang dimasukkan pengguna menjadi koordinat geografis dengan menggunakan bantuan Google Maps. Tirtayasa sendiri dibangun menggunakan PHP dan Framework CodeIgniter 4.

Selanjutnya, NewMenjangan, yang merupakan bagian *backend* dari KIRI dan dibangun dengan bahasa pemrogrman Java serta digunakan untuk memproses permintaan navigasi. Komponen ini memuat semua jalur transportasi umum dalam bentuk graf dan menggunakan algoritma Dijkstra untuk menghitung rute optimal. Algoritma ini dipercepat dengan penggunaan struktur data heap, yang membuatnya efisien untuk jalur yang kompleks.

2.2 Design Pattern dan Strategy Pattern

[1] *Design Pattern* adalah solusi umum yang telah terbukti efektif untuk mengatasi masalah desain berulang dalam pengembangan perangkat lunak berorientasi objek. Solusi ini dirancang agar dapat digunakan kembali di berbagai konteks tanpa harus disesuaikan secara berlebihan. Sebagai contoh, pola desain membantu memecah masalah desain menjadi struktur yang lebih modular dan fleksibel, sehingga mempermudah pengembangan dan pemeliharaan perangkat lunak.

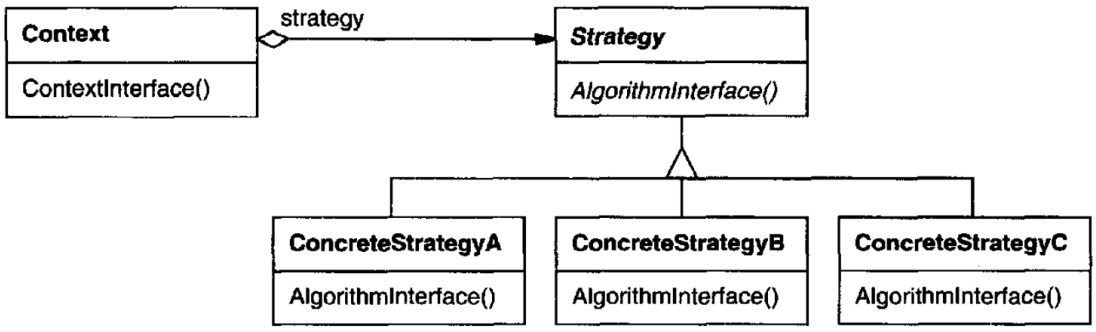
Pada dasarnya, *design pattern* memiliki empat elemen utama. Pertama, nama pola yang memberikan cara singkat untuk menyebut masalah desain tertentu, solusinya, dan konsekuensi dari penerapannya. Kedua, masalah, yaitu deskripsi konteks atau situasi di mana pola desain ini relevan. Ketiga, solusi, berupa abstraksi dari elemen-elemen desain dan kolaborasinya tanpa menyebutkan implementasi konkret. Keempat, konsekuensi, yang mencakup hasil dari penerapan pola, termasuk dampak pada fleksibilitas, efisiensi, dan pengelolaan sistem.

Penggunaan *design pattern* juga memungkinkan sistem menjadi lebih adaptif terhadap perubahan kebutuhan. Pola seperti *Strategy* mempermudah pergantian algoritma di runtime, sedangkan *Factory Method* membantu mengurangi ketergantungan pada implementasi spesifik dengan menyediakan cara fleksibel untuk membuat objek. Dengan demikian, design pattern mempermudah kolaborasi dan komunikasi antar tim pengembang.

Strategy Pattern merupakan salah satu pola desain perilaku yang dirancang untuk mendefinisikan serangkaian algoritma, mengenkapsulasi setiap algoritma, dan memungkinkan algoritma-algoritma tersebut untuk saling dipertukarkan. Pola ini memungkinkan algoritma untuk bervariasi. Dengan demikian, pengguna tidak perlu mengetahui detail implementasi dari algoritma yang digunakan, melainkan cukup berinteraksi melalui antarmuka umum yang disediakan oleh objek strategi.

Pola ini sangat berguna ketika terdapat kebutuhan untuk mendukung berbagai varian algoritma dalam menyelesaikan tugas yang sama. *Strategy Pattern* memindahkan setiap algoritma ke dalam kelas terpisah, yang disebut sebagai *ConcreteStrategy.* Klien dapat memilih dan menyuntikkan strategi yang sesuai ke dalam konteks pada waktu eksekusi, sehingga memberikan fleksibilitas yang tinggi dalam proses pengembangan perangkat lunak.

Manfaat utama dari *Strategy Pattern* adalah kemampuannya untuk menghilangkan kompleksitas yang diakibatkan oleh penggunaan pernyataan kondisional yang rumit dalam kode, serta kemudahan dalam menambahkan atau mengganti algoritma tanpa perlu memodifikasi kode klien atau konteks. Namun, penerapan pola ini juga memiliki kelemahan, seperti meningkatnya jumlah kelas dalam sistem dan potensi timbulnya overhead komunikasi antara konteks dan strategi. Oleh karena itu, penerapan *Strategy Pattern* sebaiknya dipertimbangkan dengan cermat, terutama dalam situasi di mana variasi algoritma memang diperlukan untuk memenuhi kebutuhan sistem.



Gambar 2.3: Struktur Strategy Pattern

2.2.1 Contoh Kode Program

2.3 MySQL

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

- ***CREATE DATABASE***

```
1  CREATE DATABASE database_name;
```

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

- ***DROP DATABASE***

```
1  DROP DATABASE database_name;
```

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

- ***CREATE TABLE***

```
1  CREATE TABLE table_name (
2      column1 datatype,
3      column2 datatype,
4      column3 datatype,
5      ....
6  );
```

  Sintaks tersebut digunakan untuk membuat atau memasukan tabel baru kedalam sebuah database. `table_name` diisi nama dari tabel yang akan dibuat, `column` diisi dengan nama kolom didalam tabel yang akan dibuat, dan `datatype` diisi dengan tipe data dari kolom yang akan dibuat, seperti `varchar`, `integer`, `date`, dan lain-lain.

- ***DROP TABLE***

```
1  DROP TABLE table_name;;
```

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

- ***SELECT***

```
1  SELECT column1, column2, ...
2  FROM table_name;
```

Sintaks tersebut digunakan untuk memilih atau mengambil data dari sebuah tabel dalam database. `column` diisi dengan nama kolom dari sebuah tabel yang datanya akan diambil dan `table_name` diisi dengan nama tabel dimana kolom tersebut berada.

- ***WHERE***

```
SELECT column1, column2, ...
FROM table_name
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. `column` 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***

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

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

- ***DELETE***

```
DELETE FROM table_name
WHERE condition;
```

Sintaks tersebut digunakan untuk menghapus data baru kedalam kolom dari sebuah tabel yang telah ada. `tabel_name` diisi nama tabel yang akan ditambahkan data baru, `column` diisi dengan nama kolom yang akan ditambahkan data baru, dan `value` diisi dengan nilai atau value dari data baru yang akan ditambahkan.

2.3.1 LineString

[2] *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:

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

- ST_IsClosed(ls)
  Mengecek apakah *LineString ls* membentuk lintasan tertutup (titik awal dan akhir sama). Contoh:

```
SET @ls = 'LineString(1 1, 2 2, 3 3, 1 1)';
SELECT ST_IsClosed(ST_GeomFromText(@ls));
-- Hasil: 1 (TRUE)
```

- ST_Length(ls)
  Menghitung panjang total *LineString ls*. Contoh:

```
SET @ls = 'LineString(1 1, 2 2, 3 3)';
SELECT ST_Length(ST_GeomFromText(@ls));
-- Hasil: 2.828427
```

- ST_NumPoints(ls)
  Mengembalikan jumlah titik yang membentuk *LineString ls*. Contoh:

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

- ST_Point(ls, N)
  Mengembalikan titik ke-N pada *LineString ls*. Contoh:

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

- ST_StartPoint(ls)
  Mengembalikan titik awal dari *LineString ls*. Contoh:

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

2.4 Graf

[3] Graf adalah struktur yang terdiri dari simpul (*vertex*) dan sisi (*edge*), di mana sisi menghubungkan pasangan simpul. Sebuah graf direpresentasikan sebagai pasangan $G = (V, E)$, dengan V sebagai himpunan simpul dan E sebagai himpunan sisi. Sisi diwakili oleh pasangan simpul yang terhubung. Graf dapat bersifat terarah (sisi memiliki arah) atau tidak terarah (sisi tidak memiliki arah). Simpul-simpul dalam graf dapat memiliki derajat tertentu, yaitu jumlah sisi yang menghubunginya. Sebuah graf disebut terhubung jika terdapat jalur antara setiap pasangan simpul. Jalur ini adalah urutan simpul yang dihubungkan oleh sisi. Selain itu, siklus adalah jalur tertutup di mana simpul awal dan akhir adalah sama. Teori graf juga mencakup konsep seperti pohon (graf terhubung tanpa siklus), graf bipartit (simpul dibagi menjadi dua himpunan yang saling bebas sisi), dan subgraf (bagian dari graf yang tetap mempertahankan struktur graf).
Graf digunakan dalam berbagai hal, seperti jaringan komputer, rute transportasi, dan analisis hubungan sosial. Teori graf menyediakan dasar matematis untuk mempelajari struktur ini dan memberikan cara untuk memodelkan dan menyelesaikan masalah kompleks di berbagai bidang.

2.5 Algoritma Shortest Path

Ada berbagai jenis algoritma *shortest path* yang dirancang untuk menemukan lintasan terpendek antara dua titik dalam sebuah graf. Algoritma ini memainkan peran penting dalam berbagai aplikasi, seperti sistem navigasi, perencanaan jaringan, analisis data geografis, dan pemecahan masalah rute optimal.

Setiap algoritma memiliki pendekatan, kelebihan, dan keterbatasan masing-masing, yang menjadikannya lebih sesuai untuk situasi tertentu. Sebagai contoh, algoritma seperti *Dijkstra* sangat cocok untuk graf dengan bobot positif, sementara *Floyd-Warshall* lebih sesuai untuk menemukan lintasan terpendek pada semua pasangan titik dalam graf kecil. Di sisi lain, algoritma $A*$ dirancang khusus untuk mempercepat pencarian lintasan dengan memanfaatkan heuristik. Berikut pembahasan secara detail tiga algoritma *shortest path* yang digunakan, yaitu *Dijkstra*, *Floyd-Warshall*, dan $A*$.

2.5.1 Algoritma Dijkstra

[4] Algoritma Dijkstra merupakan sebuah algoritma untuk menyelesaikan masalah *single-source shortest path*, yaitu menemukan jalur terpendek dari satu titik asal ke semua titik lainnya dalam sebuah graf berarah dengan bobot tepi non-negatif. Proses ini dimulai dengan menginisialisasi perkiraan jarak terpendek dari titik asal s ke semua titik lain. Algoritma ini menggunakan sebuah struktur *min-priority queue* (antrean prioritas minimum) yang menyimpan titik-titik dengan prioritas sesuai dengan perkiraan jarak terpendek mereka dari titik asal.

Selama eksekusi, algoritma Dijkstra akan secara bertahap memindahkan titik dengan estimasi jarak terpendek dari antrean ke dalam satu set S, yang menampung titik-titik dengan jarak terpendek yang sudah final. Untuk setiap titik u yang baru dipindahkan ke dalam set S, algoritma akan memeriksa setiap tetangganya v dan memperbarui perkiraan jarak terpendek ke v jika melalui u memberikan jarak yang lebih pendek. Proses ini dikenal sebagai "relaksasi" tepi, yaitu memperbarui perkiraan jarak dan menunjuk u sebagai pendahulu v bila ditemukan jalur yang lebih optimal.

Proses algoritma berlanjut hingga semua titik di graf telah diproses, sehingga jarak terpendek dari titik asal s ke setiap titik yang dapat dijangkau sudah final. Kompleksitas waktu dari algoritma ini bergantung pada implementasi antrean prioritas yang digunakan, dengan menggunakan *Fibonacci heap*, algoritma Dijkstra dapat mencapai kompleksitas $O(VlogV + E)$, yang efisien untuk graf yang jarang (sparse). Algoritma ini sangat bermanfaat dalam berbagai aplikasi yang melibatkan pencarian jalur terpendek, seperti sistem navigasi dan perutean jaringan.

2.5.2 Algoritma Floyd-Warshall

[4] 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.

Langkah pertama dalam algoritma ini adalah mempersiapkan matriks bobot jalur terpendek yang akan terus diperbarui. Algoritma memulai dengan menganggap setiap titik memiliki jalur langsung

ke dirinya sendiri dengan bobot nol, sementara bobot antar titik lain mengikuti nilai bobot sisi pada graf. Secara rekursif, algoritma memperbarui jalur terpendek dengan menambahkan titik perantara secara bertahap, yaitu jika titik k menjadi perantara dari titik i ke j, maka bobot jalur terpendek d_{ij} akan diperbarui menjadi minimum dari d_{ij} atau $d_{ik} + d_{kj}$. Proses ini mengoptimalkan semua jalur antara pasangan titik dengan menambahkan satu titik perantara setiap kali iterasi dilakukan. Algoritma Floyd-Warshall memiliki kompleksitas waktu $O(V^3)$ karena terdiri dari tiga lapisan perulangan untuk semua titik dalam graf, dengan V sebagai jumlah titik. Meskipun kompleksitasnya tinggi, algoritma ini cukup praktis untuk graf ukuran sedang dan memiliki struktur yang sederhana sehingga dapat diimplementasikan secara efisien. Selain itu, hasil algoritma ini dapat digunakan untuk mendeteksi siklus dengan bobot negatif dalam graf, jika ada nilai negatif pada diagonal utama dari matriks akhir, maka graf tersebut memiliki siklus negatif.

Algoritma ini juga memungkinkan pencarian jalur terpendek melalui matriks pendahulu yang mencatat titik sebelumnya pada jalur terpendek untuk setiap pasangan titik. Dengan matriks ini, jalur terpendek antara titik manapun dapat direkonstruksi secara efisien.

2.5.3 Algoritma A*

[5] 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.

Kondisi utama yang diperlukan agar algortima 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.

Terdapat kendala utama dari algoritma A*, yaitu penggunaan memori yang besar karena algoritma ini perlu menyimpan semua simpul yang telah dihasilkan. Meskipun waktu komputasi dapat diatasi dengan baik, kebutuhan memori yang tinggi sering kali menjadi tantangan. 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.

BAB 3

ANALISIS

3.1 Analisis Sistem Kini

Analisis akan dilakukan terhadap sistem backend bernama NewMenjangan, yang merupakan bagian dari KIRI. 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. 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 dari NewMenjangan. Berikut adalah struktur kelas NewMenjangan, dengan mengamati seluruh kode program yang ada.

3.1.1 Main.java

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

- **Konstanta**
  - `TRACKS_CONF`, `MYSQL_PROPERTIES`, dan `MJNSERVE_PROPERTIES`
    Konstanta-konstanta tersebut digunakan untuk mengarahkan pada file konfigurasi yang diperlukan.
  - `LOGGING_PROPERTIES` dan `NEWMJNSERVE_LOG`
    Konstanta-konstanta tersebut digunakan untuk mengatur lokasi konfigurasi logging dan file log.
- **Atribut**
  - `server`, `puller`, dan `timer`
    Atribut-atribut tersebut digunakan untuk mengelola server, menarik data, dan menjalankan tugas terjadwal.
  - `portNumber`
    Atribut ini berfungsi untuk menetapkan *port default* yang digunakan oleh server.
  - `homeDirectory`
    Atribut ini digunakan untuk menjadi direktori utama yang diambil dari variabel lingkungan `NEWMJNSERVE_HOME`, yang diperlukan agar aplikasi berjalan.

- **Method**
  - `main(String[] args)`

    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.
  - `sendCheckStatus(int portNumber)` dan `sendShutdown(int portNumber)`

    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 ditimpa dengan data baru yang diperbarui. Selain itu, proses pembaruan ini akan dicatat dalam log untuk pemantauan perubahan data yang terjadi.
  - `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.

3.1.2 AdminListener

Kelas ini berfungsi sebagai *handler* HTTP khusus yang menerima perintah-perintah administratif untuk mengelola server *backend* KIRI. Kelas ini memungkinkan aplikasi *backend* menerima dan menjalankan perintah administrasi dari localhost melalui HTTP. Pada kelas ini, terdapat 1 atribut 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.

Metode 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.

3.1.3 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`
    Berfungsi untuk menyimpan port yang digunakan server.
  - `homeDirectory`
    Berfungsi 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 ke dalam *log* global untuk memastikan bahwa kegagalan tersebut tercatat.
  - `start()` dan `stop()` Kedua metode ini berfungsi untuk mengelola server HTTP. Metode start digunakan untuk menjalankan server sehingga dapat mulai mendengarkan dan memproses permintaan yang masuk. Sebaliknya, metode stop bertugas menghentikan server HTTP, memastikan bahwa semua aktivitas server dihentikan dengan aman.

3.1.4 ServiceListener

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

- **Konstanta**
  - `PARAMETER_START`, `PARAMETER_FINISH`, `PARAMETER_MAXIMUM_WALKING`, `PARAMETER_WALKING_MULTIPLIER`,`PARAMETER_PENALTY_TRANSFER`, dan `PARAMETER_TRACKTYPEID_BLACKLIST`
    Semua konstanta tersebut digunakan untuk menentukan parameter permintaan.
- **Atribut**
  - `worker`
    Merupakan *instance* dari *class* `Worker`, yang bertanggung jawab untuk menemukan rute dan transportasi
- **Method**
  - `handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)`
    Method ini bertanggung jawab untuk menangani permintaan HTTP dan menghasilkan respons yang sesuai. Dalam prosesnya, variabel query digunakan untuk menyimpan string parameter permintaan, sementara params adalah objek Map yang berisi parameter permintaan yang telah diurai menggunakan *method* `parseQuery(query)`. Untuk menentukan hasil yang akan dikirimkan, *method* ini menggunakan variabel `responseText`

untuk menyimpan teks respons dan `responseCode` untuk menyimpan status HTTP yang akan dikembalikan kepada klien.

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

3.1.5 Worker

Kelas ini bertanggung jawab untuk menangani permintaan routing (pencarian rute) menggunakan algoritma pencarian jalur terpendek. Fungsinya adalah untuk memproses permintaan rute berdasarkan data graf, dengan mempertimbangkan parameter jarak berjalan kaki, penalti transfer, dan lainnya. Pada kelas ini, terdapat 8 atribut. Selain itu, terdapat konstruktor dan method - method diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**
  - `globalMaximumWalkingDistance`
    Merepresentasikan jarak maksimal untuk berjalan kaki.
  - `global_maximum_transfer_distance`
    Merepresentasikan arak maksimal untuk *transfer node*.
  - `globalMultiplierWalking`
    Berfungsi sebagai faktor pengali jarak berjalan kaki.
  - `globalPenaltyTransfer`
    Merepresentasikan nilai penalti untuk *transfer node*.
  - `numberOfRequests`
    Merepresentasikan jumlah permintaan yang diproses.
  - `totalProcessTime`
    Merepresentasikan total waktu proses (dalam milidetik).
  - `tracks`
    Merepresentasikan daftar rute (jalur transportasi) yang tersedia.
  - `nodes`
    Representasi graf dari seluruh *node*.
- **Konstraktor**
  - `public Worker(String homeDirectory)`
    Konstruktor ini membaca file konfigurasi utama yang disebut `MJNSERVE_PROPERTIES` untuk memuat pengaturan yang dibutuhkan. Selanjutnya, data graf jalur diambil dari file konfigurasi tambahan, yaitu `TRACKS_CONF`, untuk membangun struktur jalur yang akan digunakan. Setelah itu, *method* `linkAngkots()` dijalankan untuk menghubungkan *node-node* angkot, memastikan keterhubungan jalur transportasi dalam graf. Terakhir, metode `cleanUpMemory()` dipanggil untuk membersihkan memori sementara, sehingga efisiensi dan stabilitas sistem tetap terjaga.

- **Method**
  - `cleanUpMemory()`
    Method ini berfungsi untuk membersihkan memori yang digunakan selama proses komputasi.
  - `readConfiguration(String filename)`
    Method ini berfungsi untuk membaca konfigurasi dari file properti dan menyimpan nilai ke dalam variabel global.
  - `printTracksInfo()`
    Method ini berfungsi untuk membuat ringkasan informasi tentang rute (*track*) dan *node* yang dimuat dalam aplikasi.
  - `findRoute(LatLon start, LatLon finish, Double customMaximumWalkingDistance, Double customMultiplierWalking, Double customPenaltyTransfer, Set<String> trackTypeIdBlacklist)`
    Method ini dirancang untuk membangun graf virtual sebagai bagian dari proses pencarian rute. Proses dimulai dengan menambahkan *node start* dan *end* ke dalam graf, yang mewakili titik awal dan tujuan perjalanan. Setelah itu, *node-node* dalam graf dihubungkan menggunakan jarak berjalan kaki untuk mencerminkan kemungkinan pergerakan pejalan kaki. Selanjutnya, algoritma Dijkstra dijalankan untuk menghitung dan menemukan rute terpendek antara *node start* dan *end*. Sebagai hasil akhirnya, langkah-langkah rute yang ditemukan disusun dalam format protokol Kalapa-Dago untuk memberikan panduan perjalanan yang terstruktur dan dapat diinterpretasikan dengan mudah.
  - `resetStatistics()`
    Method ini bertujuan untuk mereset statistik pemrosesan server. Dalam prosesnya, jumlah permintaan (numberOfRequests) diatur ulang menjadi nol untuk menghapus data historis mengenai jumlah permintaan yang telah diterima. Selain itu, waktu pemrosesan total (totalProcessTime) juga diatur ulang menjadi nol untuk menghapus catatan akumulasi waktu yang digunakan dalam memproses permintaan.
  - `getNumberOfRequests()`
    Method ini mengembalikan jumlah permintaan yang telah diproses sejak statistik terakhir diatur ulang.
  - `getTotalProcessTime()`
    Method ini mengembalikan total waktu pemrosesan semua permintaan dalam detik.
  - `readGraph(String filename)`
    Method ini dirancang untuk membangun graf dengan membaca data dari file yang berisi informasi lintasan, node, dan koneksi antar node. Proses dimulai dengan membaca file untuk mengambil detail lintasan, node, serta hubungan koneksi di antara keduanya. Graf kemudian dibuat untuk merepresentasikan struktur jaringan yang akan digunakan dalam proses pencarian rute.

- `linkAngkots()`
  Method membangun *K-D Tree* yang digunakan untuk mempermudah pencarian node terdekat. Metode ini juga menghubungkan node transfer yang berada dalam batas jarak transfer maksimum, sehingga memastikan konektivitas antar node sesuai dengan aturan jarak yang telah ditentukan.
- `toString()`
  Method ini mengembalikan representasi string dari semua *track* yang dimuat.
- `findNearbyTransports(LatLon start, Double customMaximumWalkingDistance)`
  Method ini bertujuan untuk menemukan transportasi terdekat dari lokasi tertentu dengan menghitung jarak dari lokasi awal ke setiap *track* dalam graf. Setelah semua jarak dihitung, *method* ini akan menentukan lintasan dengan jarak minimum yang masih berada dalam batas yang dapat dijangkau dengan berjalan kaki.

3.1.6 DataPuller

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

- **Konstanta**
  - `EARTH_RADIUS`
    Konstanta tersebut merupakan radius Bumi dalam kilometer dan digunakan untuk menghitung jarak antar titik koordinat.
  - `MAX_DISTANCE`
    Konstanta tersebut merupakan jarak maksimum antar titik yang diizinkan, jika jarak antar dua titik melebihi nilai ini, titik-titik virtual akan ditambahkan di antaranya.
- **Method**
  - `pull(File sqlPropertiesFile, PrintStream output)`
    Berfungsi untuk mengambil data dari tabel tracks di basis data, kemudian menuliskan hasil format jalur dalam format yang ditentukan. *Method* ini memuat data dari file properti, terhubung ke basis data, dan melakukan query untuk mengambil data yang diperlukan. Hasil query diolah dan ditulis ke *output*.
  - `lineStringToLngLatArray(String wktText)`
    Mengubah data koordinat dalam format `LINESTRING` menjadi array `LngLatAlt`. Ini menghilangkan teks `LINESTRING` dan tanda kurung, kemudian memecah data menjadi objek koordinat textttLngLatAlt.
  - `computeDistance(LngLatAlt p1, LngLatAlt p2)`
    Menghitung jarak antara dua titik koordinat. Metode ini mempertimbangkan kelengkungan bumi dalam perhitungan jaraknya.

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

3.1.6.1 RouteResult

Merupakan sebuah *inner class* yang menyimpan hasil akhir dalam format konfigurasi sebagai string `trackInConfFormat`, yang dapat diambil dengan *method* `getTrackInConfFormat()`.

3.1.7 DataPullerException

Kelas ini adalah kelas *custom exception* yang dibuat untuk menangani kesalahan khusus yang terjadi saat pemrosesan data dalam kelas DataPuller. Kelas ini memperluas `RuntimeException`, sehingga DataPullerException adalah *unchecked exception* dan tidak memerlukan penanganan eksplisit dengan blok *try-catch* di tempat pemanggilannya. Pada kelas ini, terdapat sebuah konstanta serta konstruktor yang memiliki penjelasan sebagai berikut:

- **Konstanta**
  - `serialVersionUID`
    Menyimpan ID *serial*. ID ini memastikan data yang disimpan atau dikirimkan tetap cocok dengan versi kelas yang digunakan saat objek tersebut dibaca kembali. Hal ini penting, terutama jika kelas ini mengalami perubahan struktur, agar versi yang berbeda tetap dapat dikenali atau mencegah kesalahan jika struktur sudah tidak cocok.
- **Konstruktor**
  - DataPullerException(String message)
    Konstruktor ini menerima pesan kesalahan dalam bentuk String, yang kemudian diteruskan ke konstruktor *superclass* `RuntimeException` untuk disimpan dan nantinya dapat diambil dengan metode `getMessage()`. Pesan ini bertujuan untuk memberikan informasi yang lebih rinci tentang kesalahan yang terjadi.

3.1.8 LatLon

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

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

- **Atribut**
  - `lat`
    Merepresentasikan nilai lintang suatu titik.
  - `lon`
    Merepresentasikan nilai bujur suatu titik.
- **Konstruktor**
  - *LatLon(float lat, float lon)*
    Konstruktor ini menerima dua parameter, `lat` (*latitude*) dan `lon` (*longitude*), yang disimpan langsung dalam atribut publik kelas.
  - *LatLon(String latlon)*
    Konstruktor ini menerima parameter tunggal berupa String dengan format *latitude*,*longitude*. String ini kemudian dipisah berdasarkan tanda koma (","), lalu nilai-nilai yang diperoleh diubah menjadi float dan disimpan dalam atribut `lat` dan `lon`.
- **Method**
  - `toString()`
    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).

3.1.9 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`.
- `cleanUpMemory()`
  Memangkas ukuran `internalArray` ke jumlah elemen yang sebenarnya untuk menghemat memori.

3.1.9.1 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 diimpelentasikan.

- **Atribut**
  - `currentList`
    Menunjuk pada UnrolledLinkedList saat ini yang sedang diiterasi.
  - `currentIndex`
    Posisi indeks di `internalArray` pada `currentList`.
  - `globalIndex`
    Posisi indeks global yang melacak elemen secara keseluruhan di seluruh UnrolledLinkedList.
- **Method**
  - `hasNext()`
    Mengecek apakah masih ada elemen berikutnya di dalam `internalArray` atau di `nextList`.
  - `next()`
    Mengembalikan elemen berikutnya dalam iterasi. Jika sudah mencapai akhir `internalArray`, beralih ke `nextList`.
  - `hasPrevious()` dan `previous()`
    Melempar `UnsupportedOperationException`.
  - `nextIndex()` dan `previousIndex()`
    Mengembalikan indeks global berikutnya atau sebelumnya.
  - `remove()`
    melempar `UnsupportedOperationException`.
  - `set(E e)`
    Mengganti elemen di `currentIndex` dengan elemen baru `e`.
  - `add(E e)`
    Menambahkan elemen `e` ke dalam `UnrolledLinkedList`.

3.1.10 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.
- **Method**
  - `int getNode()`
    Metode ini mengembalikan nilai *node* yang dituju oleh *edge* ini. Berguna untuk mendapatkan informasi tentang *node* tujuan.
  - `double getWeight()`
    Metode ini mengembalikan bobot dari *edge* saat ini, yang dapat digunakan dalam perhitungan atau penelusuran jalur dalam graf.

3.1.11 GraphNode

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

- **Atribut**
  - `location`
    Atribut ini menyimpan lokasi *node* dalam bentuk koordinat lintang dan bujur. `location` merupakan objek dari `LatLon`, yang menyimpan posisi geografis dalam satuan derajat.
  - `edges`
    Atribut ini menyimpan daftar dari semua *edge* (sisi) atau jalur keluar dari node ini. Jalur keluar ini mengarah ke node lain, menciptakan koneksi dalam graf. `edges` diimplementasikan menggunakan UnrolledLinkedList.

- `track`
  Atribut ini berfungsi sebagai referensi balik ke informasi rute (track) dari node. Dengan atribut ini, sistem dapat mengetahui rute atau *track* mana yang terkait dengan node tersebut.
- `isTransferNode`
  Atribut ini menunjukkan apakah *node* tersebut adalah *node transfer*. Dalam konteks transportasi umum, sebuah *node transfer* memungkinkan pengguna untuk turun atau naik kendaraan umum dari node tersebut. Jika bernilai *true*, berarti *node* ini adalah *node transfer*.

- **Kontruktor**
  - `GraphNode(LatLon location, Track track)`
    Konstruktor ini membuat instance baru dari kelas `GraphNode` dengan lokasi (`LatLon`) dan referensi rute (`track`). Saat diinisialisasi, atribut `isTransferNode` diatur ke *false* secara *default*, dan `edges` diinisialisasi sebagai daftar kosong dari objek `GraphEdge`.
- **Method**
  - `getEdges()`
    Mengembalikan daftar *edges*, yaitu daftar sisi keluar dari *node* ini. Daftar ini dapat digunakan untuk mengetahui semua koneksi dari *node* ke *node* lain dalam graf.
  - `push_back(int node, float weight)`
    Method ini menambahkan sisi baru ke daftar *edges* dengan memasukkan informasi *node* tujuan dan bobotnya. Bobot (*weight*) mencerminkan jarak atau biaya perjalanan ke *node* lain.
  - `link(GraphNode nextNode)`
    Method ini menghubungkan *node* ini dengan *node* lain (`nextNode`) dengan menambahkan semua sisi (*edges*) dari *node* berikut ke daftar sisi (*edges*) *node* ini.
  - `getLocation()`
    Mengembalikan lokasi geografis dari *node* ini dalam bentuk objek `LatLon`
  - `getTrack()`
    Mengembalikan referensi rute (*track*) yang terkait dengan *node* ini. Ini memungkinkan akses ke informasi rute dari *node*.
  - `toString()`
    Mengembalikan representasi teks dari *node* ini, yang mencakup informasi lokasi dan status apakah *node* ini adalah *node transfer* atau bukan.
  - `setTransferNode(boolean b)`
    Mengatur apakah *node* ini merupakan *node transfer* berdasarkan parameter . Jika `b` bernilai *true*, maka *node* akan dianggap sebagai *node transfer*.
  - `isTransferNode()`
    Mengembalikan nilai *boolean* yang menunjukkan apakah *node* ini adalah *node transfer* (*true*) atau bukan (*false*).

3.1.12 Graph

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

- **Konstruktor**
  - `Graph()` Konstruktor ini membuat sebuah objek `Graph` tanpa menentukan kapasitas awal. Dengan kata lain, kapasitas akan ditentukan berdasarkan elemen yang ditambahkan.
  - `Graph(int capacity)`
    Konstruktor ini membuat objek `Graph` dengan kapasitas awal tertentu, sesuai dengan nilai `capacity` yang diberikan.
- **Method**
  - `rangeSearch(LatLon center, double distance)`
    Method ini berfungsi untuk mencari semua node yang berada dalam jangkauan atau radius tertentu dari suatu titik pusat. Parameter `center` adalah objek `LatLon` yang menyatakan titik pusat dari area pencarian, dan `distance` adalah radius pencarian dalam satuan kilometer.
    Pada implementasi awal, *method* ini menggunakan perulangan sederhana melalui setiap `GraphNode` dalam graf (`for(GraphNode node : this)`). Untuk setiap *node* dalam graf, *method* ini menghitung jarak antara `center` dan lokasi *node* (`node.location`) dengan menggunakan metode `distanceTo` pada objek `LatLon`. Jika jarak antara `center` dan *node* tersebut lebih kecil atau sama dengan `distance`, maka *node* tersebut dianggap berada dalam jangkauan, dan dimasukkan ke dalam *list*. Metode ini mengembalikan *list*, yaitu daftar `GraphNode` yang berada dalam jangkauan dari `center`.

3.1.13 Dijkstra

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

- **Konstanta**
  - `DIJKSTRA_NULLNODE`
    Nilai konstan -1 untuk menandakan node yang tidak valid.
- **Atribut**
  - `graph`
    Daftar *node* dalam graf yang merepresentasikan jalur.
  - `startNode` dan `finishNode`
    Menyimpan indeks *node* awal dan akhir dari pencarian.

- `nodeInfoLinks`
  Array NodeInfo yang menyimpan informasi jarak terdekat untuk setiap node.
- `nodesMinHeap`
  Array NodeInfo yang menyimpan node dalam urutan jarak terpendek untuk mendukung operasi heap dalam algoritma Dijkstra.
- `heapsize`
  Ukuran heap, menandakan jumlah node yang ada dalam heap.
- `numOfNodes`
  Jumlah total node dalam graf.
- `memorySize`
  Ukuran memori yang digunakan.
- `multiplierWalking` dan `penaltyTransfer`
  Faktor pengali dan penalti yang digunakan untuk menghitung bobot tambahan pada node yang terkait dengan jalur pejalan kaki atau transfer angkot.

- **Method**
  - `runAlgorithm(Set<String> trackTypeIdBlacklist)`
    Merupakan *method* utama untuk menjalankan algoritma Dijkstra yang dirancang untuk menemukan jalur terpendek dalam sebuah graf. Langkah pertama adalah menginisialisasi setiap objek `NodeInfo` dengan jarak awal bernilai `POSITIVE_INFINITY` dan menetapkan jarak awal *node* sumber (`startNode`) menjadi 0. Setelah itu, struktur `nodesMinHeap` diatur ulang menggunakan metode `heapify` untuk mengurutkan *node* berdasarkan jarak terpendek. Proses pencarian dimulai dengan mengambil *node* saat ini (`currentNode`) dari heap, dan pencarian dihentikan jika *node* tersebut adalah *node* tujuan (`finishNode`). Selama iterasi, untuk setiap objek `GraphEdge` yang terhubung dengan `currentNode`, jarak ke *node* tujuan dihitung menggunakan *method* `calculateWeight`. Jika jarak baru yang dihitung lebih pendek daripada jarak yang tercatat sebelumnya, dan tidak termasuk dalam `trackTypeIdBlacklist`, maka jarak tersebut diperbarui. Selanjutnya, node dengan jarak yang lebih kecil dipindahkan ke posisi yang sesuai dalam heap menggunakan *method* `heapPercolateUp`. Setelah semua langkah selesai, *method* ini akan mengembalikan jarak terpendek ke *node* tujuan. Jika tidak ada jalur yang tersedia, nilai `POSITIVE_INFINITY` akan dikembalikan untuk menandakan kegagalan menemukan jalur.
  - `getParent(int node)`
    Mengembalikan *parent* (*node* asal) dari *node* yang dimaksud dalam rute terpendek.
  - `getDistance(int node)`
    Mengembalikan jarak dari *node* awal ke *node* yang diminta.
  - `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 menerapkan 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.

- `heapPercolateDown(int index)`
  Menjaga agar heap tetap terurut setelah penghapusan node dengan jarak terpendek.
- `heapPercolateUp(int index)`
  Memperbarui posisi node dalam heap setelah perubahan jarak.
- `heapDeleteMin()`
  Menghapus node dengan jarak terpendek dari heap, dan menyesuaikan urutan heap.
- `getString(int node)`
  Mengembalikan string dari `NodeInfo`.

3.1.13.1 NodeInfo

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

DAFTAR REFERENSI

[1] Gamma, E., Helm, R., Johnson, R., dan Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

[2] Version 8.4 (2024) *MySQL 8.4 Reference Manual - Including MySQL NDB Cluster 8.4*. Oracle Corporation. Austin, USA.

[3] Diestel, R. (2017) *Graph Theory*, 5th edition. Springer, Berlin, Heidelberg.

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., dan Stein, C. (2009) *Introduction to Algorithms*, 3rd edition. The MIT Press, Cambridge, MA.

[5] Russell, S. dan Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, Upper Saddle River, NJ.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

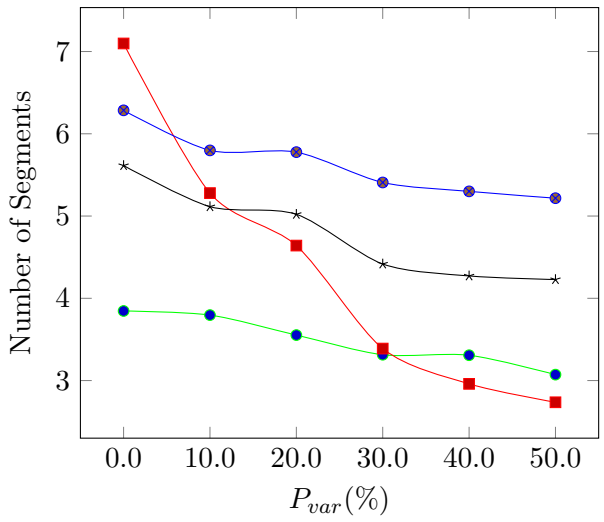
Kode A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

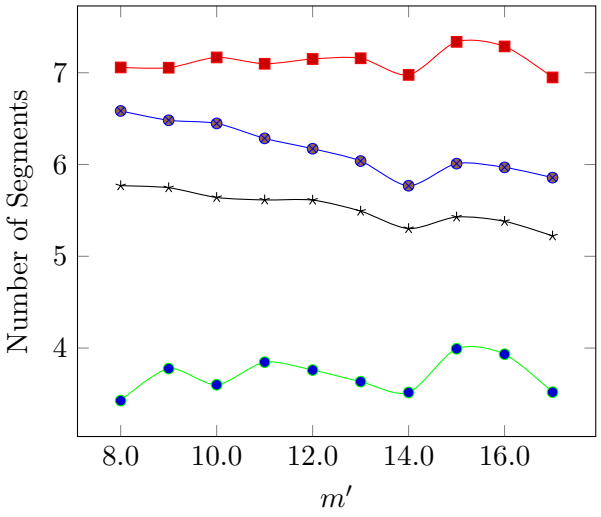
LAMPIRAN B

HASIL EKSPERIMEN

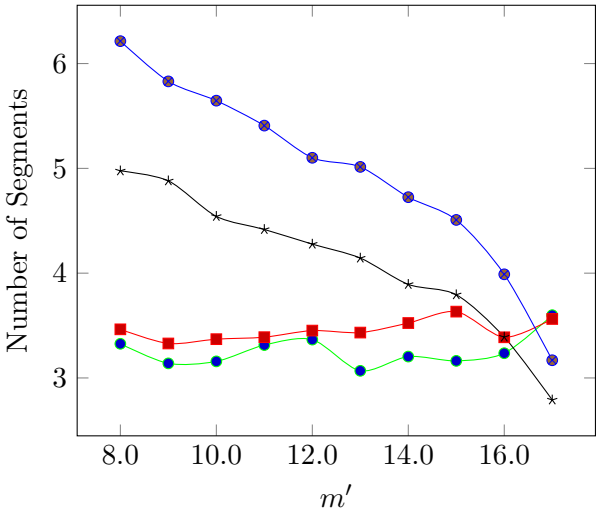
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



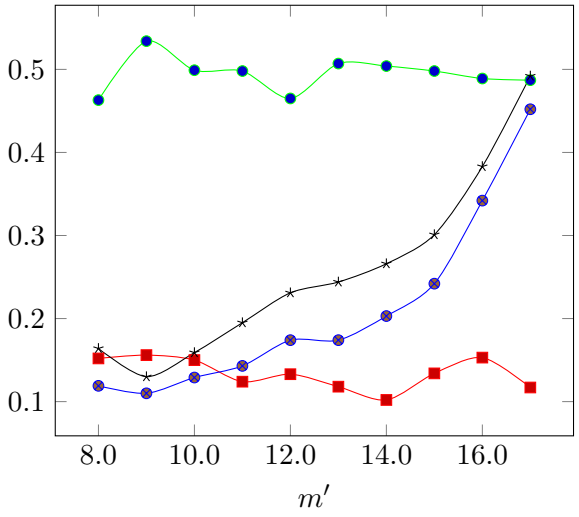
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4