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

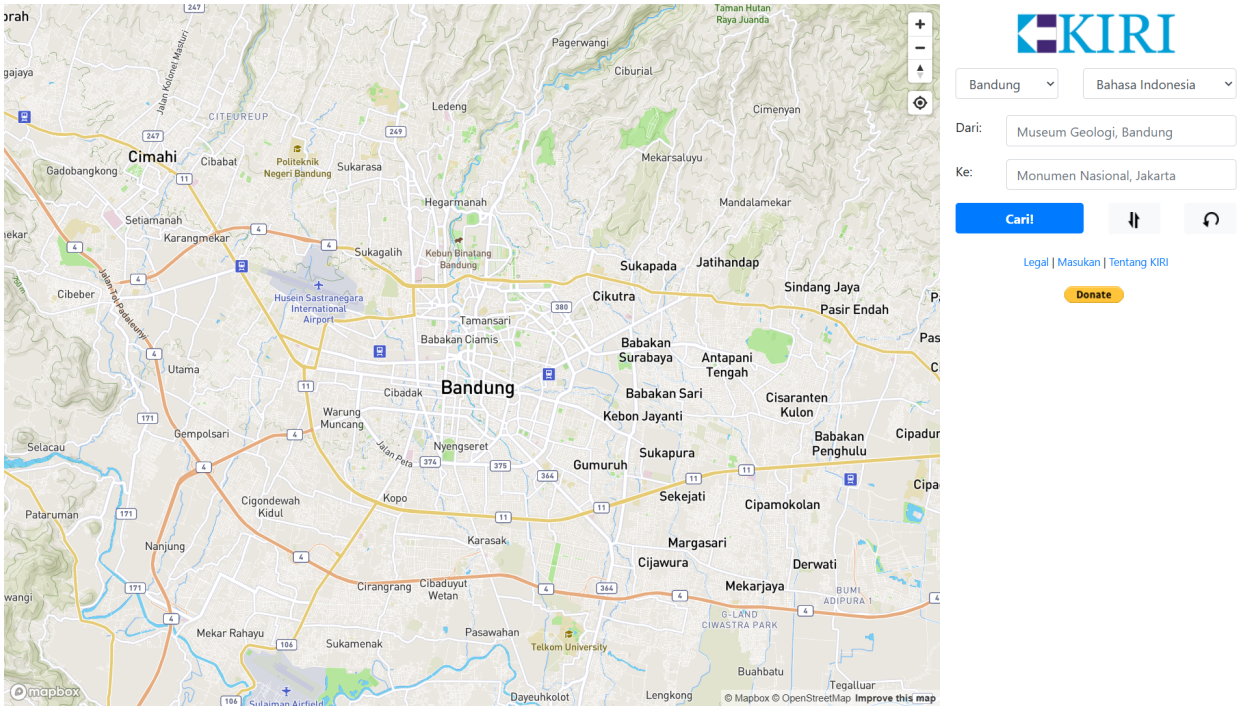
DAFTAR GAMBAR

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 tipe data geometri yang digunakan dalam basis data, untuk merepresentasikan garis atau rangkaian segmen garis[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

2.2 Design 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.

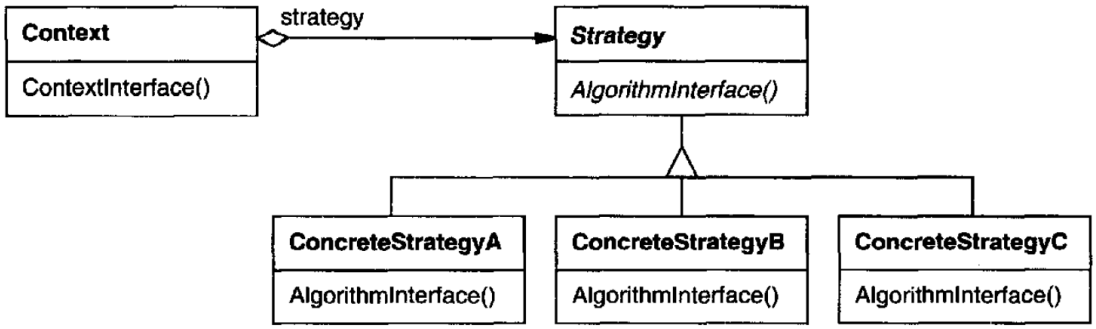
Penggunaan *design pattern* juga memungkinkan sistem menjadi lebih adaptif terhadap perubahan kebutuhan. Pola seperti *Strategy* mempermudah pergantian algoritma di runtime, sedangkan *Factory Method* membantu mengurangi ketergantungan pada implementasi spesifik dengan menyediakan cara fleksibel untuk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

2.2.1 Strategy Pattern

[1] *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 secara independen dari pengguna. 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 2.1: Struktur Strategy Pattern

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;
```

- ***DROP DATABASE***

```
DROP DATABASE database_name;
```

- ***CREATE TABLE***

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ....
);
```

- ***DROP TABLE***

```
DROP TABLE table_name;;
```

- ***SELECT***

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

- ***WHERE***

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- ***INSERT INTO***

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

- ***DELETE***

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

2.3.1 LineString

[2] *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. Setiap titik pada *LineString* memiliki koordinat (x, y), dan minimal terdiri dari dua titik untuk membentuk garis. *LineString* digunakan dalam Sistem Informasi Geografis (GIS) untuk merepresentasikan lintasan seperti jalan, sungai, atau rute perjalanan.

Terdapat fungsi untuk memanipulasi dan menganalisis *LineString*. Fungsi-fungsi ini memungkinkan pengguna untuk mengambil informasi spesifik dari objek *LineString*, seperti titik awal, titik akhir, jumlah titik, atau panjang lintasan. Berikut adalah beberapa fungsi penting yang berkaitan dengan *LineString*:

- ST_EndPoint(ls)

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

Berfungsi untuk mengembalikan titik akhir dari objek *LineString ls*.

- ST_IsClosed(ls)

```
SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';
SELECT ST_IsClosed(ST_GeomFromText(@ls1));
```

Berfungsi untuk mengembalikan nilai TRUE/1 jika *LineString ls* adalah lintasan tertutup (titik awal dan akhir sama).

- ST_Length(ls)

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

Berfungsi untuk menghitung panjang total *LineString ls*.

- ST_NumPoints(ls)

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

Berfungsi untuk mengembalikan jumlah total titik yang membentuk *LineString ls*.

- ST_Point(ls, N)

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

Berfungsi untuk mengembalikan titik ke-N pada *LineString ls*.

- ST_StartPoint(ls)

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

Berfungsi untuk mengembalikan titik awal dari objek *LineString ls*.

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 dapat 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 aplikasi, 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-masalah kompleks di berbagai bidang.

2.5 Algoritma Dijkstra

[4] Algoritma Dijkstra 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. Jarak titik asal sendiri diatur ke nol, sedangkan titik lainnya diatur sebagai tak hingga. 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.

2.6 Algoritma Floyd-Warshall

[4] Algoritma Floyd-Warshall 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.

2.7 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 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, 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

3.1.1 Main.java

Kelas Main 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 beberapa variabel diinisialisasikan serta *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Variabel**
  - `TRACKS_CONF`, `MYSQL_PROPERTIES`, dan `MJNSERVE_PROPERTIES`
    Variabel - variabel ini digunakan untuk mengarahkan pada file konfigurasi yang diperlukan.
  - `LOGGING_PROPERTIES` dan `NEWMJNSERVE_LOG`
    Variabel - variabel ini digunakan untuk mengatur lokasi konfigurasi logging dan file log.
  - Variabel `server`, `puller`, dan `timer`
    Variabel - variabel ini digunakan untuk mengelola server, menarik data, dan menjalankan tugas terjadwal.
  - `portNumber`
    Variabel ini berfungsi untuk menetapkan *port default* yang digunakan oleh server.
  - `homeDirectory`
    Variabel ini digunakan untuk menjadi direktori utama yang diambil dari variabel lingkungan `NEWMJNSERVE_HOME`, yang diperlukan agar aplikasi berjalan.
- **Method**
  - `main`
    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` dan `sendShutdown`
  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`
  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 AdminListener 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 beberapa variabel diinisialisasikan serta *method* - *method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Variabel**
  - `worker`
    Variabel ini bertipe Worker yang merupakan sebuah kelas. worker ini diperlukan untuk menjalankan perintah-perintah tertentu, seperti tracksinfo.
- **Method**
  - `handle`
    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. Dengan pengaturan ini, metode memastikan respons yang sesuai terhadap setiap jenis permintaan yang diterima.

3.1.3 NewMenjanganServer

Kelas NewMenjanganServer 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 konstruktor yang berfungsi untuk menginisialisasi variabel yang digunakan serta *handler* yang berfungsi untuk menangani permintaan - permintaan serta perintah administratif. Selain itu, terdapat beberapa variabel lainnya diinisialisasikan serta *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Variabel**
  - `DEFAULT_PORT_NUMBER`
    Merupakan nomor port *default* yang digunakan jika tidak ada port yang ditentukan.
  - `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.
- **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 ServiceListener 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 beberapa variabel diinisialisasikan serta *method* - *method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Variabel**
  - `PARAMETER_START`, `PARAMETER_FINISH`, `PARAMETER_MAXIMUM_WALKING`, `PARAMETER_WALKING_MULTIPLIER`,`PARAMETER_PENALTY_TRANSFER`, dan `PARAMETER_TRACKTYPEID_BLACKLIST`
    Semua parameter tersebut digunakan untuk menentukan parameter permintaan.
  - `worker`
    Merupakan instance dari class Worker, yang bertanggung jawab untuk menemukan rute dan transportasi.
- **Method**
  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.
  Dalam menangani permintaan HTTP, *method* ini juga mengatur logika pemrosesan permintaan berdasarkan parameter yang diterima. Metode ini mencoba mendapatkan parameter `start` dan `finish`, yang mewakili koordinat awal dan akhir untuk menentukan rute perjalanan. Selain itu, parameter tambahan seperti `maximumWalking`, `walkingMultiplier`, `penaltyTransfer`, dan `trackTypeIdBlacklist` digunakan untuk menyesuaikan batas jarak berjalan kaki, faktor pengali untuk jalan kaki, penalti transfer, serta daftar jenis jalur yang diblokir. Jika parameter textttstart dan `finish` tersedia, pencarian rute dilakukan menggunakan *method* `worker.findRoute`. Sebaliknya, jika hanya start yang tersedia, pencarian transportasi terdekat dilakukan melalui *method* `worker.findNearbyTransports`. Namun, jika kedua parameter tidak tersedia, *method* akan mengembalikan pesan kesalahan berupa "*Please provide start and finish location.*"
  Setelah proses permintaan selesai, pengaturan respons dilakukan untuk memastikan hasil dikirimkan dengan benar. Status respons HTTP diatur menggunakan `response.setStatus(responseCode)`, sementara *method* `baseRequest.setHandled(true)` digunakan untuk menandakan bahwa permintaan telah berhasil ditangani. Akhirnya, teks respons dikirimkan ke klien melalui `response.getWriter().println(responseText)`. Dengan pendekatan ini, *method* ini memastikan bahwa setiap permintaan ditangani secara terstruktur dan memberikan tanggapan yang sesuai.

3.1.5 Worker

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 beberapa variabel serta *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**
  - `EARTH_RADIUS`
    Bertipe `double` yang merupakan radius Bumi dalam kilometer dan digunakan untuk menghitung jarak antar titik koordinat.
  - `MAX_DISTANCE`
    Bertipe `double` yang 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.
  - `RouteResult`
    Merupakan *inner class*, kelas ini menyimpan hasil akhir dalam format konfigurasi sebagai string `trackInConfFormat`, yang dapat diambil dengan metode `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 atribut serta konstruktor yang memiliki penjelasan sebagai berikut:

- **Atribut**
  - `serialVersionUID`

    Bertipe data long yang 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.

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
```

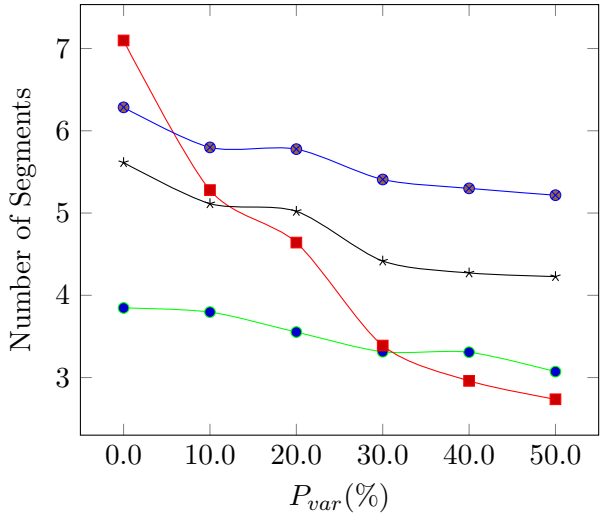
Kode A.2: MyCode.java

```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);
        }
    }

}
```

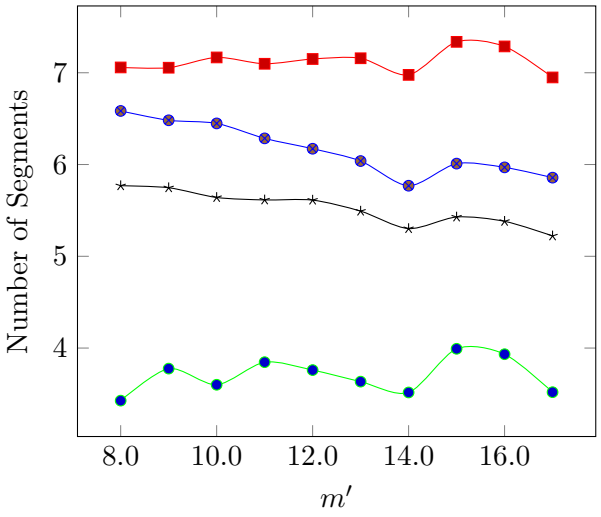
LAMPIRAN B

HASIL EKSPERIMEN

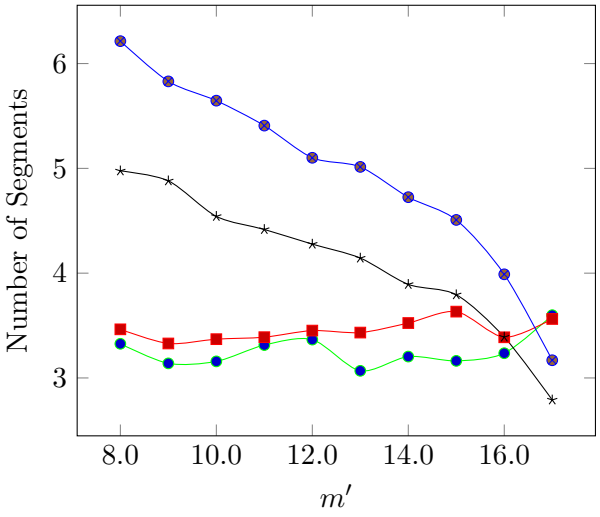
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



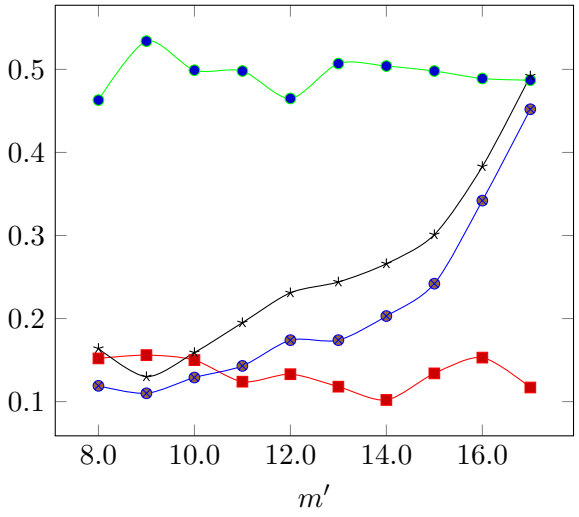
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4