

LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA
MENENTUKAN LINTASAN TERPENDEK
MENGGUNAKAN ALGORITMA UCS DAN A*



Disusun Oleh
Akmal Mahardika Nurwahyu P. (13521070)
Michael Leon Putra Widhi (13521108)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023

BAB I

DESKRIPSI MASALAH DAN SPESIFIKASI

A. Deskripsi Masalah

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antar dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

B. Spesifikasi Program

Berikut merupakan spesifikasi program pada Tugas Kecil 3 IF2211

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

BAB II

ALGORITMA UCS DAN A* SERTA IMPLEMENTASINYA

A. Algoritma UCS dalam Penentuan Lintasan Terpendek

Dalam pencarian solusi dari masalah lintasan terpendek atau *Shortest-Path Problems*, salah satu algoritma yang sangkil digunakan adalah Algoritma *Uniform-Cost Search* (UCS). Algoritma UCS mengutamakan simpul yang memiliki kumulatif biaya terendah untuk dieksplorasi terlebih dahulu.

Algoritma UCS dikategorikan sebagai algoritma pencarian *Uninformed Search* atau *Blind-Search*. Hal tersebut berarti algoritma ini hanya mengandalkan informasi mengenai graf yang sudah dan akan dieksplorasi tanpa adanya informasi tambahan. Algoritma UCS melakukan pencarian dari simpul awal kemudian mengeksplorasi simpul-simpul yang bertetangga dengan simpul awal tersebut. Untuk setiap simpul yang dijelajahi, algoritma ini akan menghitung biaya yang telah dilalui. Biaya ini akan dijumlahkan untuk setiap jalur yang diambil dari simpul awal dan disimpan kedalam variabel atau fungsi $g(n)$. Setelah itu, algoritma UCS akan memilih simpul jelajah berikutnya dengan memilih biaya terkecil, yaitu simpul dengan nilai $g(n)$ terendah. Proses ini akan diulang hingga simpul tujuan ditemukan.

Algoritma UCS cocok digunakan pada persoalan *Shortest-Path Problems* yang memiliki graf yang sangat besar atau kompleks, karena algoritma UCS memungkinkan untuk mengeksplorasi seluruh simpul pada graf tersebut. Namun, penggunaan algoritma UCS dapat menjadi tidak sangkil jika persoalan tersebut memiliki biaya yang tidak seragam atau jika simpul tujuan terletak jauh dari simpul awal, sehingga memerlukan banyak waktu dan sumber daya untuk mengeksplorasi seluruh simpul pada graf. Distribusi biaya simpul yang tidak merata dan lokasi simpul tujuan yang jauh dari simpul awal berakibat pada prioritas simpul jelajah yang dikunjungi, misal jika simpul awal dengan simpul akhir memiliki nilai bobot $g(n)$ yang lebih besar dari semua nilai $g(n)$ yang lain, prioritas pengecekan menuju simpul akhir akan terus tergeser dengan simpul jelajah yang lebih dekat.

B. Algoritma A* dalam Penentuan Lintasan Terpendek

Persoalan penentuan lintasan terpendek atau *Shortest-Path Problems* juga dapat diselesaikan dengan Algoritma A*. Algoritma ini merupakan algoritma pencarian yang termasuk dalam golongan pencarian dengan informasi (*informed search*), sehingga jika didesain dengan baik, algoritma ini akan memperoleh hasil pencarian yang tidak hanya sangkil, tetapi juga mangkus.

Secara umum algoritma ini mirip seperti algoritma UCS. Hal yang membuatnya berbeda adalah selain dilakukan perhitungan total jarak yang telah dilalui, penentuan simpul jelajah berikutnya juga dilakukan dengan mempertimbangkan nilai jarak heuristik dari posisi simpul yang sedang dianalisis. Misalkan terdapat sebuah simpul n yang akan dianalisis, maka akan terdapat sebuah nilai $g(n)$ yang menyatakan total nilai dari simpul itu saat ini yang diperoleh dari jarak tempuh simpul dari simpul awal pencarian hingga saat ini dan nilai $h(n)$ yang menyatakan nilai heuristik jarak terdekat antara simpul saat ini dengan simpul akhir. Total nilai yang dimiliki oleh simpul ini dinyatakan dengan $f(n)$ yang merupakan penjumlahan dari kedua faktor sebelumnya, $g(n) + h(n)$. Bobot ini yang kemudian akan di konsiderasi dalam pemilihan simpul jelajah yang dilakukan oleh simpul orangtuanya (dalam hal ini simpul yang akan menjadikan simpul n sebagai simpul jelajah).

Lebih lanjut, nilai heuristik diperoleh dari perhitungan jarak garis lurus (*straight-line distance*) dari simpul tersebut hingga simpul akhir pencarian. Misalkan nilai jarak sebenarnya dari simpul n hingga simpul akhir dinyatakan sebagai $h'(n)$, maka nilai dari heuristik harus selalu lebih kecil dari atau sama dengan nilai sebenarnya ($h(n) \leq h'(n)$) untuk menghasilkan nilai estimasi yang bisa diterima (*admissible*) dengan nilai $h(n)$ yang harus konsisten. Metode penentuan nilai heuristik menggunakan jarak garis lurus yang telah dibahas sebelumnya sudah pasti menjamin kedua kondisi ini sehingga menjadi metode estimasi yang paling sering digunakan.

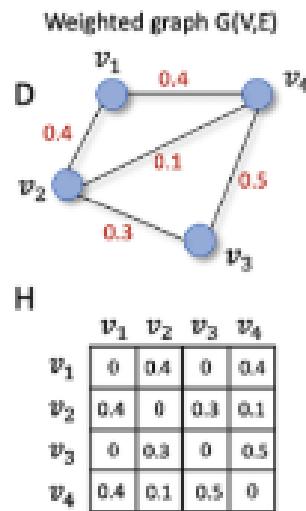
C. Penjelasan Algoritma Program

1. Penjelasan dasar terkait graf dan matriks ketetanggaan

Graf merupakan sebuah struktur diskrit yang terdiri atas simpul dan sisi. Struktur ini kemudian dapat digunakan untuk merepresentasikan sebuah permasalahan terkait pencarian lintasan terpendek dengan membuat sebuah graf berbobot (setiap sisi yang

menghubungkan dua buah simpul memiliki sebuah nilai yang dinyatakan sebagai “bobot”) yang memenuhi kondisi setiap bobot pada sisi memiliki makna jarak yang ditempuh dari simpul “asal” dan simpul “tujuan” yang dihubungkan oleh sisi pada graf tersebut. Perwujudan graf yang bercabang dan runtut membuat proses analisis jalur secara manusiawi menjadi lebih mudah, tetapi lain hal dengan implementasi secara komputasi. Oleh sebab itu, terdapat beberapa metode yang digunakan untuk merepresentasikan struktur graf, salah satunya dengan menggunakan matriks ketetanggaan berbobot.

Misalkan terdapat sebuah senarai multidimensi yang merupakan sebuah matriks ketetanggaan berbobot berukuran $m \times n$ dengan m menyatakan jumlah baris dan n menyatakan jumlah kolom. Untuk memenuhi kaidah dasar matriks ketetanggaan, maka haruslah nilai $m = n$. Lebih lanjut, karena kasus berada pada lingkup penelusuran jalur, maka dapat diasumsikan bahwa graf yang direpresentasikan dalam matriks ketetanggaan ini merupakan graf tidak berarah. Hal ini membuat dalam matriks ketetanggaan, nilai bobot pada baris ke- i dan kolom ke- j harus sama dengan nilai bobot pada baris ke- j dan kolom ke- i dengan $\{i, j\} \in m$ dan $\{i, j\} \in n$.



Gambar 1. Graf dalam bentuk matriks ketetanggaan berbobot
Sumber : Tautan QnA Spesifikasi Tugas Strategi Algoritma

Terdapat pula beberapa aturan dasar yang harus dipenuhi agar sebuah matriks ketetanggaan berbobot dikatakan merupakan representasi graf tidak berarah yang valid, diantaranya :

- a. Untuk dua buah simpul yang tidak dihubungkan dengan sisi, maka nilai bobot dari matriks pada baris dan kolom terkait adalah nilai yang khusus, dapat berupa nol dengan asumsi tidak ada dua buah simpul berbeda yang berhimpit, nilai negatif, atau nilai yang sangat besar untuk menyatakan “tak hingga”.
- b. Simpul yang direpresentasikan dalam matriks ketetanggaan pada baris ke- i dan kolom ke- j yang memenuhi $i = j$ juga harus memiliki nilai khusus yang bernilai sama untuk kasus bobot pada baris ke- j dan kolom ke- i .

Pada program yang telah diimplementasikan, graf tidak berarah berbobot yang dinyatakan dalam matriks ketetanggaan berbobot telah memenuhi semua aturan yang dijelaskan diatas. Proses implementasi menggunakan nilai -1 untuk kedua simpul yang tidak saling terhubung dan nilai 0 untuk simpul yang sama dengan alasan bahwa tidak diperlukannya nilai bobot yang menyatakan jarak dari simpul menyatakan sebuah titik yang berada pada posisi yang sama.

2. Penjelasan dasar terkait masukan *file* kasus uji

Secara umum, masukan yang diterima setidaknya mengandung elemen-elemen dasar proses pencarian jalur pencarian jalur terpendek, yaitu matriks ketetanggaan berbobot. Pada program yang telah diimplementasikan, sistem akan menerima masukan berupa sebuah *file* masukan dengan tipe ekstensi *.json*. Adapun alasan pemilihan ekstensi adalah karena ekstensi ini akan membuat proses pembacaan masukan kasus uji menjadi lebih mangkus karena tidak memerlukan skema pembacaan *file* yang relatif rumit.

File yang menjadi masukan kasus uji harus terdiri atas dua buah komponen utama, yaitu :

- a. posList, merupakan sebuah larik berisi himpunan informasi terkait simpul masukan yang akan memudahkan proses penggambaran dan penjelasan pada peta. Himpunan informasi tersebut terdiri atas :
 - 1) id, sebuah *string* numerik yang merupakan identifikator dari titik masukan.
 - 2) nama, sebuah *string* yang menyatakan nama simpul masukan pada id yang bersangkutan. Penambahan atribut ini lebih lanjut merupakan tidak lanjut dari jawaban pada laman QnA yang menyatakan bahwa setiap simpul perlu untuk diberi nama.

- 3) lintang, sebuah *float* yang menyatakan posisi lintang dari sebuah simpul masukan dengan id terkait, dan
 - 4) bujur, sebuah *float* yang menyatakan posisi bujur dari sebuah simpul masukan dengan id terkait.
- b. adjMatrix, merupakan sebuah senarai multidimensi yang merupakan matriks ketetanggaan yang valid (memenuhi semua kriteria yang dijelaskan pada poin sebelumnya). Gunakan nilai -1 untuk kedua simpul yang tidak saling terhubung dan nilai 0 untuk simpul yang sama dengan alasan bahwa tidak diperlukannya nilai bobot yang menyatakan jarak dari simpul menyatakan sebuah titik yang berada pada posisi yang sama.

Pada lampiran tautan repositori, khususnya pada pada *folder test*, telah dicantumkan beberapa contoh kasus uji valid yang dapat digunakan untuk melakukan pengujian lintasan terpendek. Lebih lanjut, Program juga telah dilengkapi dengan berbagai skema validasi masukan untuk menjamin semua kriteria masukan yang telah dijabarkan terpenuhi dan program dapat berjalan dengan baik.

3. Implementasi struktur data

Pada program yang dibuat telah diimplementasikan dua buah struktur data yang harapannya dapat membantu proses menjalankan algoritma dengan lebih baik dan lebih natural. Berikut adalah penjelasannya.

a. Struktur data jalur (*Path*)

Sebuah jalur merupakan kumpulan dari beberapa simpul terurut yang menyatakan daftar simpul yang telah dikunjungi dalam rute tersebut mulai dari simpul yang paling awal dikunjungi hingga simpul yang saat ini atau terakhir kali dikunjungi. Struktur data ini memanfaatkan sebuah senarai sebagai penampung jalur, sebuah bilangan untuk menyimpan posisi simpul paling depan (yang akan ditelusuri simpul jelajahnya) saat ini, total jarak yang telah dilewati sebelumnya dari simpul awal, sebut saja $g(n)$, dan nilai prioritas jalur tersebut yang didasarkan pada penjumlahan nilai simpul yang telah dilalui dan nilai heuristik jarak ke simpul yang menjadi simpul tujuan, sebut saja $h(n)$, dengan persamaan $f(n) = g(n) + h(n)$. Struktur data ini yang kemudian menjadi sebuah objek

yang diantrikan dalam antrian prioritas untuk menentukan pilihan jalur yang akan dieksekusi terlebih dahulu.

b. Struktur data antrian prioritas (*PrioQueue*)

Struktur data ini secara fungsional mirip seperti struktur data antrian (*Queue*) sehingga bisa tetap menambahkan elemen pada antrian (*enqueue*) dan menghapus elemen dari antrian (*dequeue*). Akan tetapi proses penambahan dan penghapusan elemen mempertimbangkan yang disebut sebagai nilai prioritas. Nilai prioritas sendiri dapat diperoleh secara eksplisit (disebutkan pada elemen antrian suatu nilai untuk dibandingkan) atau implisit (antrian akan membandingkan menggunakan sebuah skema khusus).

Proses penambahan elemen pada antrian prioritas (misalkan elemen X) tidak ditambahkan pada bagian belakang, tetapi pada posisi yang memenuhi karakteristik prioritas elemen di “depan” posisi X lebih besar dari prioritas X dan semua elemen di “belakang” posisi tersebut memiliki prioritas yang lebih kecil dari X . Sedangkan proses penghapusan memiliki mekanisme yang mirip dengan struktur data antrian, yaitu mengambil elemen paling depan yang sekaligus merupakan elemen dengan prioritas paling tinggi. Pada program yang diimplementasikan, elemen dari antrian adalah struktur data jalur yang dijelaskan sebelumnya. Proses penentuan prioritas didasarkan pada nilai prioritas yang telah menjadi bagian dari struktur data setiap jalur yang ada pada antrian prioritas sehingga mekanisme pemilihan simpul atau jalur yang akan dijelajahi selanjutnya menjadi lebih mangkus dan natural.

4. Implementasi solusi dalam Algoritma UCS

Alur algoritma UCS merupakan pengembangan dari *Breadth-First Search* dan *Iterative Deepening Search* (BFS dan IDS tidak menjadi bagian dari tugas kecil 3). Berikut merupakan prosedurnya.

- Misalkan terdapat sebuah matriks ketetapan berukuran $m \times n$ dengan m menyatakan jumlah baris dan n menyatakan jumlah kolom. Matriks ini juga menyimpan informasi bobot terhadap tetangganya. Kemudian dipilih dua buah simpul, misalkan simpul X sebagai simpul awal dan simpul Y sebagai simpul

tujuan yang merupakan salah satu simpul yang terbaca pada data kasus uji masukan.

- b. Inisiasi pencarian dilakukan dengan menyatakan simpul awal sebagai simpul aktif. Simpan simpul tersebut dalam sebuah senarai yang menampung simpul yang pernah dikunjungi (tahap ini serupa dengan mengubah status simpul dari “belum dikunjungi” menjadi “telah dikunjungi”).
- c. Inisiasi selanjutnya, yaitu membuat sebuah objek jalur (*Path*) yang menampung simpul awal. Berilah prioritas pada jalur tersebut sebesar $g(n) = 0$ karena belum mengunjungi simpul apapun selain simpul awal. Kemudian inisiasi sebuah objek antrian prioritas (*PrioQueue*) dan antrikan jalur tersebut dengan prioritas $f(n) = g(n) = 0$. Objek antrian prioritas ini akan mengurutkan $f(n) = g(n)$ (kumulatif jarak yang telah ditempuh dari simpul awal) dari paling kecil hingga paling besar
- d. Ambil jalur pada antrian paling depan dan nyatakan sebagai jalur yang sedang diperiksa, kemudian nyatakan simpul terakhir pada jalur tersebut sebagai simpul yang sedang diperiksa. Pengambilan jalur dalam antrian dilakukan dengan cara dihapus dari antrian (*dequeue*)
- e. Tentukan semua simpul jelajah yang valid terhadap simpul yang sedang diperiksa. Simpul jelajah yang valid adalah simpul yang bertetangga dengan simpul yang sedang diperiksa dan tidak ada dalam senarai simpul yang telah dikunjungi (simpul belum pernah dikunjungi). Daftar simpul jelajah ini akan disimpan dalam sebuah senarai.
- f. Untuk setiap simpul dalam daftar simpul jelajah :
 - 1) Hitung nilai $f(n)$ dengan menjumlahkan $g(n)$ yang terdapat pada jalur dengan jarak dari simpul yang sedang diperiksa menuju simpul dalam daftar jelajah (informasi jarak ini disimpan pada matriks berbobot-ketetanggaan pada poin a).
 - 2) Buat sebuah jalur baru dengan menyalin jalur yang sedang diperiksa ditambah simpul jelajah, kemudian tentukan prioritas, yaitu sebesar $f(n)$ pada poin f.1.
 - 3) Antrikan jalur tersebut ke dalam antrian prioritas.

- g. Ulangi proses pada d, e, dan f hingga mencapai salah satu dari dua kondisi:
- 1) Jika simpul yang sedang diperiksa (simpul aktif) merupakan simpul tujuan, maka masukan jalur yang sedang diperiksa (jalur ini mengandung nilai $f(n)$ terkecil dan mengandung simpul tujuan) ke dalam senarai penampung jalur solusi.
 - 2) Jika antrian prioritas kosong sebelum melakukan *dequeue* dan belum ada simpul jelajah yang merupakan simpul solusi, maka terbukti tidak ada jalur yang akan mencapai simpul tujuan.

5. Implementasi solusi dengan Algoritma A*

Secara umum alur jalannya Algoritma A* merupakan pengembangan dari Algoritma UCS yang ditambahkan dengan metode aproksimasi dengan heuristik. Solusi pemecahan ini dibangun menggunakan konsep graf dinamis sehingga perlu dilakukan penyimpanan jalur untuk mempermudah melakukan peruntutan langkah pencarian simpul jelajah selanjutnya. Berikut adalah prosedurnya.

- a. Misalkan terdapat sebuah matriks ketetapan berukuran $m \times n$ dengan m menyatakan jumlah baris dan n menyatakan jumlah kolom dan sebuah larik himpunan nilai informasi sebuah simpul dari kasus uji masukan. Kemudian dipilih dua buah simpul, misalkan simpul X sebagai simpul awal dan simpul Y sebagai simpul tujuan yang merupakan salah satu simpul yang terbaca pada data kasus uji masukan.
- b. Proses dimulai dengan melakukan inisiasi pencarian. Nyatakan posisi awal sebagai simpul aktif (simpul posisi saat ini) dan simpan dalam sebuah senarai yang menampung daftar simpul aktif yang sudah pernah dikunjungi untuk menghindari pengunjungan berulang. Buatlah sebuah objek jalur (*Path*) yang menampung simpul posisi awal X . Beri nilai prioritas pada jalur tersebut sebesar nilai heuristik $h(n)$ menggunakan metode jarak garis lurus (*straight-line distance*) ke simpul tujuan Y dan nilai $g(n) = 0$ karena belum mengunjungi simpul apapun selain simpul awal.
- c. Inisiasi sebuah objek antrian prioritas (*PrioQueue*) dan antrikan jalur yang dibuat pada tahap sebelumnya ke dalam antrian prioritas ini. Prioritas didasarkan pada

nilai prioritas yang dinyatakan pula pada tahap sebelumnya ($f(n) = h(n)$ karena nilai $g(n) = 0$)

- d. Tentukan simpul jelajah yang dapat dikunjungi dengan Algoritma A*. Simpul jelajah yang dapat dikunjungi merupakan simpul yang “bertetangga” dengan simpul aktif yang sedang dikunjungi (dalam hal ini simpul awal pencarian). Pada proses lanjutan, simpul jelajah yang dianalisis adalah simpul terakhir pada jalur pertama – yang sekaligus merupakan jalur dengan nilai bobot tempuh paling kecil – antrian prioritas yang telah dideklarasikan sebelumnya.

Pengambilan jalur pertama dalam antrian dilakukan dengan cara dihapus dari antrian (*dequeue*). Adapun pencarian simpul jelajah dilakukan dengan mempertimbangkan simpul jelajah yang valid (terhubung dengan simpul yang diperiksa) dan belum pernah dikunjungi sebelumnya (tidak ada dalam senarai penampung simpul yang sudah pernah dikunjungi).

- e. Setelah menentukan daftar simpul jelajah yang kemudian dinyatakan dalam sebuah senarai, dilakukan analisis sebagai berikut :

- 1) Jika ternyata jumlah simpul jelajah dari simpul tersebut ternyata tidak ada dan antrian prioritas yang dideklarasikan sebelumnya sudah kosong di saat belum ada simpul jelajah yang merupakan simpul solusi, maka terbukti tidak ada jalur yang akan mencapai simpul tujuan. Pencarian dihentikan dan disampaikan bahwa simpul tujuan tidak dapat dicapai dari simpul asal.
- 2) Jika ternyata jumlah simpul jelajah dari simpul tersebut tidak ada tetapi antrian prioritas belum kosong, maka proses dilanjutkan dengan mengambil jalur terdepan berikutnya dari antrian prioritas dan lakukan proses d.
- 3) Jika ternyata terdapat simpul jelajah, maka untuk setiap simpul jelajah lakukan hal berikut :
 - a) Inisiasi sebuah jalur baru yang merupakan salinan dari jalur yang sebelumnya dihapus dari antrian prioritas pada poin d.
 - b) Tentukan bobot dari jalur baru yang ditambahkan dengan setiap simpul jelajah dengan melakukan kalkulasi nilai $g(n)$ dan $h(n)$. Nyatakan nilai $g(n)$ sebagai nilai jarak antara simpul terakhir pada jalur yang dihapus dari antrian prioritas ke simpul jelajah yang berkoresponden,

kemudian nyatakan nilai $h(n)$ sebagai nilai heuristik jarak garis lurus antara setiap simpul jelajah dengan simpul tujuan.

- c) Berikan nilai prioritas pada simpul baru tersebut dengan $f(n)$ yang merupakan nilai penjumlahan dari kedua nilai yang didefinisikan sebelumnya ($f(n) = g(n) + h(n)$).
- d) Antrikan setiap simpul baru hasil bentukan ke dalam antrian prioritas. Karena struktur data antrian prioritas menambahkan antrian berdasarkan prioritas, maka proses penambahan simpul tidak akan ditambahkan ke belakang, tetapi sesuai dengan nilai prioritas yang ditetapkan pada setiap jalur.
- f. Lakukan proses d dan e secara berulang hingga ditemukan sebuah jalur dari antrian prioritas yang mencapai simpul tujuan (simpul terakhir dari jalur sudah sama dengan simpul tujuan). Jika simpul simpul aktif merupakan simpul tujuan, maka masukan jalur yang sedang diperiksa ke dalam senarai penampung jalur solusi.

Adapun solusi pencarian rute terpendek dengan algoritma A* yang diimplementasikan sudah pasti merupakan lintasan yang paling pendek untuk mencapai tujuan karena menggunakan metode heuristik yang tidak berlebihan (*overestimate*) sehingga pasti akan memiliki nilai hasil estimasi yang masuk akal dan bisa diterima (*admissible*).

D. Implementasi Bonus

Visualisasi peta menggunakan API *Google Maps*

Adapun program yang dibuat berhasil menggunakan API berupa modul yang dapat menggunakan fitur peta untuk melakukan penggambaran pencarian jalur terpendek dari dua buah simpul yang dipilih. Adapun implementasi dilakukan menggunakan modul leaflet yang menggunakan OpenStreetMap sebagai penampil peta. Berbagai mekanisme dan fitur tambahan diluar spesifikasi diimplementasikan disini, salah satunya adalah melakukan penyimpanan *file* jika terdapat perubahan pada kondisi peta. Cara penggunaan dan fitur telah dibahas lebih lanjut pada bagian [README.md](#).

BAB III

SOURCE CODE PROGRAM

Program yang digunakan untuk mengimplementasikan program dibuat berbasis *website* sehingga mudah untuk diakses pengguna dari mana saja dan kapan saja. Berikut adalah kode sumber dari proses implementasi program.

A. File index.html

Pada file ini didefinisikan semua fungsi dan prosedur yang berhubungan dengan pemrosesan terhadap tampilan dalam bentuk *hypertext*. Semua penggayaan dan fungsionalitas program dituliskan disini. Berikut adalah kode sumbernya.

```
index.html

<!DOCTYPE html>
<html lang="en">
    <!-- Header, impor beberapa script dan stylesheet yang diperlukan -->
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <!-- GoogleMap API -->
        <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
            integrity="sha256-kLaT2GOSpHechhszzB+fInD+zUyjE2LlfWPgU04xyI="
            crossorigin="" />
        <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
            integrity="sha256-WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM="
            crossorigin=""></script>
        <!-- Menambahkan stylesheet CSS dan script Javascript -->
        <link rel="stylesheet" href="style.css">
        <script src="/main.js"></script>
        <script src="modules/Algorithm.js"></script>
        <script src="modules/Path.js"></script>
        <script src="modules/PrioQueue.js"></script>
        <!-- Judul Popup -->
        <title>Tugas Kecil 3 - Strategi Algoritma</title>
    </head>

    <!-- Navbar, pilihan menu -->
    <nav>
        <a href="/" class="pop">Path Finder</a>
        <ul>
            <li><a href="https://github.com/mikeleo03/Tucil3_13521070_13521108/blob/main/README.md">About</a></li>
            <li><a href="https://github.com/mikeleo03/Tucil3_13521070_13521108">Explore</a></li>
        </ul>
    </nav>
```

```

<!-- Body -->
<body>
    <div class="row">
        <!-- Bagian kolom kiri -->
        <div class="column left">
            <!-- Tampilan peta -->
            <div id="map" class="map"></div>
            <!-- Tampilan hasil algoritma -->
            <div class="path"></div>
        </div>

        <!-- Bagian kolom tengah -->
        <div class="column mid">
            <!-- Tambahkan masukan -->
            <h4>Insert File</h4>
            <!-- Tempat menambahkan kasus uji masukan -->
            <div class="label">
                <label>Insert Test Case here!</label>
            </div>
            <div class="inputs">
                <input type="file" id="inputFile">
            </div>

            <!-- Pembacaan terhadap file, menjalankan fungsi readFile -->
            <div class="inputs">
                <input type="button" class="button" value="Load Map" onclick="readFile();">
            </div>

            <!-- Penyimpanan terhadap perubahan peta, menjalankan fungsi saveFile -->
            <div class="label">
                <label>Made a change on the map?</label>
            </div>
            <div class="inputs">
                <button class="button-down" onclick="saveFile();"><span>Save Current Map </span></button>
            </div>

            <!-- Pemilihan simpul yang akan dianalisis -->
            <h4>Shortest Path Searching</h4>
            <div>
                <!-- Simpul awal -->
                <div class="label">
                    <label for="instantiateInitial">Initial node</label>
                </div>
                <div class="inputs">
                    <select id="instantiateInitial" class="lengthinput init-pos"></select>
                </div>
                <!-- Simpul tujuan -->
                <div class="label">
                    <label for="instantiateFinal">Final node</label>
                </div>
                <div class="inputs">
                    <select id="instantiateFinal" class="lengthinput final-pos"></select>
                </div>
            </div>

```

```

<!-- Melakukan pemrosesan algoritma sesuai button, menjalankan fungsi doAlgo() -->
<div class="inputs">
    <input type="button" class="button" value="Process UCS" onclick="doAlgo(2);">
    <input type="button" class="button" value="Process A*" onclick="doAlgo(1);">
</div>
</div>

<!-- Bagian kolom kanan -->
<div class="column right">
    <!-- Menambah relasi antar simpul masukan -->
    <h4>Insert Relations</h4>
    <div class="row">
        <!-- Simpul awal -->
        <div class="label column">
            <label for="relationInitial">Start node</label>
        </div>
        <div class="inputs column right">
            <select id="relationInitial" class="lengthinput2 init-rels"></select>
        </div>
    </div>

    <!-- Simpul tujuan -->
    <div class="label column">
        <label for="relationFinal">Final node</label>
    </div>
    <div class="inputs column right">
        <select id="relationFinal" class="lengthinput2 final-rels"></select>
    </div>
    </div>

    <!-- Proses penambahan relasi simpul, menjalankan fungsi addRelation -->
    <div class="inputs">
        <input type="button" class="button" value="Insert Relation" onclick="addRelation();">
    </div>

    <!-- Menampilkan data simpul yang berhasil di-load -->
    <h4>List of Nodes</h4>
    <table class="table">
        <!-- Table header -->
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">Nodes</th>
                <th scope="col">Distance</th>
            </tr>
        </thead>

        <!-- Table content, isi dirubah dari javascript sesuai jumlah masukan -->
        <tbody id="pathContent">
        </tbody>
    </table>
    </div>
</div>
</body>
</html>

```

B. File style.css

Pada file ini didefinisikan semua penggayaan terhadap komponen yang ada pada bagian HTML. Berikut adalah kode sumbernya.

```
style.css

/* Perubahan style pada elemen di HTML */
/* Style utama */
:root {
    font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;
    line-height: 1.5;
    font-weight: 400;

    color-scheme: light dark;
    color: rgba(255, 255, 255, 0.87);
    background: rgb(2,0,36);
    background: linear-gradient(90deg, rgba(0,0,0,1) 0%, rgba(2,0,36,1) 45%, rgba(18,53,60,1) 100%);

    font-synthesis: none;
    text-rendering: optimizeLegibility;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    -webkit-text-size-adjust: 100%;
}

/* Style bagian navbar */
nav {
    z-index: 2;
    position: relative;
    padding: 0.5rem 2rem;
    display: flex;
    justify-content: space-between;
    font-size: 1.5rem;
    padding-bottom: 0.5rem;
}

nav a {
    text-decoration: none;
    padding: 1rem;
    color: white;
    padding-bottom: 0.5rem;
}

nav a:hover {
    font-weight: bold;
}

nav ul {
    list-style: none;
    display: flex;
    gap: 2rem;
    padding: 0.5rem;
}
```

```

.pop {
    text-decoration: none;
    color: white;
    font-weight: bold;
    font-size: 1.5rem;
    padding: 1.5rem;
    margin-bottom: 0.1rem;
}

/* Style bagian body */
/* Separasi baris dan kolom */

.row {
    display: table;
    clear: both;
}

.column {
    float:left;
}

.left {
    width: 60%;
}

.mid {
    width: 25%;
}

.right {
    width: 15%;
}

/* Penanganan penggayaan pada masukan */

.label {
    padding: 0.3rem 1rem;
}

.inputs {
    padding: 0.1rem 1rem;
}

/* Penanganan penggayaan peta */

#map {
    height: 25rem;
    width: 36rem;
}

.map {
    padding: 0.1rem 0.5rem;
    margin-left: 3.5rem;
    margin-right: -20rem;
}

/* Penggayaan terhadap luaran dan tata letaknya */

```

```

.path {
  padding: 0.1rem 1rem;
  margin-left: 3rem;
  width: 18rem;
  overflow: auto;
}

.right h4 {
  margin-left: 1rem;
}

h4 {
  margin-bottom: -0.1rem;
}

.column h4 {
  margin-bottom: 0.5rem;
}

/* Penanganan penggayaan terhadap komponen button */

.button {
  background-color: #4CAF50; /* Green */
  border: none;
  border-radius: 4px;
  color: white;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  padding: 1rem;
  padding-top: 0.5rem;
  padding-bottom: 0.5rem;
  margin-top: 0.5rem;
}

.button:hover {
  background-color: #29762b; /* Green */
  cursor: pointer;
}

.delete-button {
  background-color: #D11A2A; /* Green */
  border: none;
  border-radius: 4px;
  color: white;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  padding: 0.5rem;
  padding-top: 0.2rem;
  padding-bottom: 0.2rem;
}

.delete-button:hover {
  background-color: #9b1822; /* Green */
  cursor: pointer;
}

```

```

}

.button-down {
    border-radius: 4px;
    background-color: #4CAF50; /* Green */
    border: none;
    color: #FFFFFF;
    text-align: center;
    padding: 1rem;
    padding-top: 0.5rem;
    padding-bottom: 0.5rem;
    transition: all 0.5s;
    cursor: pointer;
}

.button-down span {
    cursor: pointer;
    display: inline-block;
    position: relative;
    transition: 0.5s;
}

.button-down span:after {
    content: '\00bb';
    position: absolute;
    opacity: 0;
    top: 0;
    right: -20px;
    transition: 0.5s;
}

.button-down:hover span {
    padding-right: 25px;
    cursor: pointer;
}

.button-down:hover span:after {
    opacity: 1;
    right: 0;
}

.lengthinput {
    width: 10rem;
    height: 2rem;
    padding-left: 0.5rem;
}

.lengthinput2 {
    width: 7rem;
    height: 2rem;
    padding-left: 0.5rem;
}

/* Penanganan penggayaan pada komponen tabel */
table {

```

```

border-collapse: collapse;
text-align: center;
vertical-align: middle;
margin-left: 1rem;
margin-top: 1rem;
max-height: 2rem;
}

thead {
background-color: #333;
color: white;
font-size: 0.875rem;
letter-spacing: 2%;
}

tbody tr:nth-child(odd) {
background-color: #fffff5d;
}

tbody tr:nth-child(even) {
background-color: #eeeeee39;
}

th, td {
padding-left: 15px;
padding-right: 15px;
padding-top: 10px;
padding-bottom: 10px;
}

/* Penanganan penggayaan terhadap konten pop-up */
.popup-content {
width: 7rem;
margin-top: -0.5rem;
margin-bottom: 1rem;
overflow-y: auto;
}

```

C. File Path.js

Pada file ini, dideklarasikan struktur data jalur (*Path*) yang akan digunakan untuk menjalankan fungsionalitas algoritma.

Path.js

```

// File javascript tempat untuk mendeklarasikan tipe data Path
/**
 * Kelas Path, untuk melakukan pemrosesan terkait data simpul analisis
 *
 * @class Path
 * @property {Array} listPath - Array yang berisi simpul yang telah dilewati
 * @property {Number} currentPos - Simpul yang sedang diperiksa
 * @property {Number} passedpath - Jarak yang telah ditempuh

```

```

* @property {Number} priority - Bobot dari simpul yang sedang diperiksa
*/
class Path {
    /**
     * @constructor Konstruktor kelas Path
     * @param {Number} currentPos
     * @param {Number} passedpath
     * @param {Number} priority
     */
    constructor(currentPos, passedpath, priority) {
        this.listPath = [];
        this.currentPos = currentPos;
        this.passedpath = passedpath;
        this.priority = priority;
    }

    /**
     * @method copyPath - Melakukan penyalinan isi dari jalur lain ke jalur ini
     * @param {Path} paths
     */
    copyPath(paths) {
        for (var i = 0; i < paths.listPath.length; i++) {
            this.listPath.push(paths.listPath[i]);
        }
    }

    /**
     * @method getPrio - Getter bobot dari simpul yang sedang diperiksa
     * @returns {Number}
     */
    getPrio() {
        return this.priority;
    }

    /**
     * @method printPath
     * @returns {String} - Mengembalikan string yang berisi jalur yang menjadi masukan
     */
    printPath() {
        let Path = "";
        // Melakukan iterasi terhadap semua komponen penyusun jalur
        for (var i = 0; i < this.listPath.length; i++) {
            if (i != this.listPath.length - 1) {
                Path = Path + this.listPath[i] + " > ";
            } else {
                Path = Path + this.listPath[i];
            }
        }

        // Mengembalikan string hasil konversi jalur masukan
        return Path;
    }

    /**
     * @method addPosition - Menambahkan simpul yang telah dilewati
     */
}

```

```

    * @param {Number} pos
    */
addPosition(pos) {
    this.listPath.push(pos);
    this.currentPos = pos;
}

```

D. File PriorityQueue.js

Pada file ini, dideklarasikan struktur data antrian prioritas (*PriorityQueue*) yang akan digunakan untuk menjalankan fungsionalitas algoritma.

PriorityQueue.js

```

// File javascript tempat untuk mendeklarasikan tipe data PrioQueue
/**
 * Kelas PQ, untuk melakukan pemilihan simpul dengan bobot terkecil
 * Direpresentasikan sebagai antrian prioritas (PrioQueue)
 *
 * @class PQ
 * @property {Array} queue - Array yang berisi simpul yang telah dilewati
 */
class PQ {
    /**
     * @constructor
     */
    constructor() {
        this.queue = [];
    }

    /**
     * @method getLength - Getter panjang dari queue
     * @returns {Number}
     */
    getLength() {
        return this.queue.length;
    }

    /**
     * @method getElmt - Getter elemen queue pada indeks idx
     * @param {Number} idx
     * @returns {Path}
     */
    getElmt(idx) {
        return this.queue[idx];
    }

    /**
     * @method setElmt - Setter elemen queue pada indeks idx
     * @param {Number} idx
     * @param {Path} val
     */
}
```

```

/*
setElmt(idx, val) {
    this.queue[idx] = val;
}

/***
* @method swap - Menukar posisi elemen pada indeks pos1 dan pos2
* @param {Path} pos1
* @param {Path} pos2
* @returns {PQ}
*/
swap(pos1, pos2) {
    let val = this.queue[pos1];
    this.setElmt(pos1, this.getElmt(pos2));
    this.setElmt(pos2, val);
    // Mengembalikan isi queue setelah ditukar
    return this.queue;
}

/***
* @returns {Boolean} - Mengembalikan true jika queue kosong
*/
isEmpty() {
    return (this.getLength() == 0);
}

/***
* @method enqueue - Menambahkan elemen ke dalam queue
* Ingat perlu mempertimbangkan prioritas pada saat menambahkan elemen
*
* @param {Path} newPath
*/
enqueue(newPath) {
    var check = false; // Boolean validasi
    // Menemukan lokasi yang tepat untuk insert
    for (var i = 0; i < this.getLength(); i++) {
        if (this.getElmt(i).getPrio() > newPath.getPrio()) {
            // Menemukan lokasi yang tepat, masukkan elemen
            this.queue.splice(i, 0, newPath);
            check = true;
            break;
        }
    }

    // Kalo ternyata semua elemen pada antrian lebih kecil dari masukan
    // Maka tambahkan di akhir
    if (!check) {
        this.queue.push(newPath);
    }
}

/***
* @method dequeue - Menghapus elemen pertama dengan bobot terkecil dalam antrian
*
* @returns {Path} - Mengembalikan simpul dengan bobot terkecil
*/

```

```

*/
dequeue() {
    // Melakukan pengecekan apakah antrian kosong
    if (this.isEmpty()) {
        return "Priority Queue is empty";
    } else {
        return this.queue.shift();
    }
}

```

E. File Algorithm.js

Pada file ini, diimplementasikan hasil penggunaan modul dari dua struktur data yang dijelaskan sebelumnya. Beberapa fungsi yang didefinisikan disini dijadikan sebagai fungsi pembantu dan penggerak fungsionalitas dari masing-masing algoritma.

Algorithm.js

```

// File javascript yang menjalankan fungsionalitas algoritma
/**
 * Fungsi heuristics, untuk melakukan kalkulasi nilai h(n)
 * Yaitu straight line distance dari simpul n ke finish
 * Proses perhitungan dilakukan menggunakan Persamaan Haversine
 *
 * @function heuristics
 * @param {Array} posList - Array yang berisi informasi posisi simpul
 * @param {Number} final - Indeks simpul tujuan
 * @param {Number} initial - Indeks simpul awal
 */
function heuristics (posList, final, initial) {
    // Mengambil informasi dari masukan
    // Simpul awal
    let lintang1 = posList[initial - 1].lintang * Math.PI / 180;
    let bujur1 = posList[initial - 1].bujur * Math.PI / 180;
    // Simpul tujuan
    let lintang2 = posList[final - 1].lintang * Math.PI / 180;
    let bujur2 = posList[final - 1].bujur * Math.PI / 180;

    // Perhitungan inisiasi
    let r = 6371 // Radius bumi
    let dLintang = lintang2 - lintang1
    let dBujur = bujur2 - bujur1

    // Perhitungan dengan Persamaan Haversine
    let a = Math.pow(Math.sin(dLintang / 2), 2)
        + Math.cos(lintang1) * Math.cos(lintang2)
        * Math.pow(Math.sin(dBujur / 2), 2);
    let c = 2 * Math.asin(Math.sqrt(a));

    return r * c;
}

```

```

/**
 * Fungsi isAStarDone, untuk mengecek apakah proses pencarian A* sudah selesai dilaksanakan
 * Simpul tujuan yang sedang dianalisis sudah ada di listActiveNode
 *
 * @function isAStarDone
 * @param {PQ} listActiveNode - List simpul yang masih aktif
 * @param {Number} finish - Simpul tujuan
 * @returns {Boolean} - Mengembalikan true jika simpul finish sudah ada di listActiveNode
 */
function isAStarDone (listActiveNode, finish) {
    // Inisiasi nilai
    let temp = false;
    // Melakukan traversal terhadap isi antrian
    for (var i = 0; i < listActiveNode.getLength(); i++) {
        // Jika sudah ada yang membuat simpul tujuan
        if (listActiveNode.getElmt(i).currentPos == finish) {
            temp = true; // Ubah nilai kiriman
            finalPath = listActiveNode.getElmt(i); // Masukan dalam jalur solusi akhir
            break; // Selesaikan pencarian
        }
    }
    // Kembalikan kondisi akhir pencarian kondisi pada antrian
    return temp;
}

/**
 * Fungsi getExpand, untuk mendapatkan simpul jelajah berdasarkan titik saat ini
 *
 * @function getExpand
 * @param {Number} position - Simpul saat ini yang akan dicek
 * @param {Number[][]} adjMatrix - Matriks ketetanggaan yang telah terdefinisi
 * @param {Number[]} cek - Senarai simpul yang sudah pernah dianalisis
 * @returns {Number[]} - ID dari simpul jelajah
 */
function getExpand (position, adjMatrix, cek) {
    // Inisiasi senarai penampung simpul jelajah
    let expandNode = [];
    // Melakukan traversal terhadap isi matriks ketetanggaan
    for (var i = 0; i < adjMatrix[0].length; i++) {
        // Jika terdapat relasi simpul dan belum pernah dijelajahi
        if (adjMatrix[position - 1][i] != -1 && !cek.includes(i + 1)) {
            // Tambahkan dalam daftar simpul jelajah
            expandNode.push(i);
        }
    }
    // Kembalikan daftar simpul jelajah
    return expandNode;
}

/**
 * Fungsi AStar, menjalankan algoritma A*.
 * prioritas berdasarkan f(n) = g(n) + h(n)
 * dengan g(n) adalah jarak dari simpul awal ke simpul n,
 * h(n) adalah jarak garis lurus dari simpul n ke ke simpul tujuan
*/

```

```

/*
* @function AStar
* @param {Number} start simpul awal
* @param {Number} finish simpul tujuan
* @param {Number[][]} adjMatrix matriks ketetanggaan
* @param {Number[]} posList daftar posisi
*/
function AStar (start, finish, adjMatrix, posList) {
    // Instansiasi simpul yang udah pernah dijelajahi
    let sudahdicek = [];
    sudahdicek.push(start);
    // Inisiasi posisi awal
    let current = start;
    // Membuat sebuah jalur awal, inisiasi peta
    let initialPath = new Path(current, 0, 0 + heuristics(posList, finish, start));
    initialPath.listPath.push(current);
    // Memasukkan jalur awal ke antrian simpul aktif
    let listActiveNode = new PQ();
    listActiveNode.enqueue(initialPath);

    // Selama belum ada rute yang mencapai finish
    while (!isAStarDone(listActiveNode, finish)) {
        // hapus dan ambil rute paling depan dari antrian prioritas
        let Paths = listActiveNode.dequeue();
        // Ubah posisi titik analisis saat ini
        current = Paths.currentPos;
        if (!sudahdicek.includes(current)) {
            sudahdicek.push(current);
        }

        // Cari semua simpul jelajah dari titik saat ini
        let expandNode = getExpand(current, adjMatrix, sudahdicek);
        // Jika simpul tidak lagi memiliki ekspan karena tidak terhubung dengan tujuan
        if (expandNode.length == 0 && listActiveNode.isEmpty()) {
            // Keluarkan tanggapan dan keluar dari kalang
            alert("There's no path to that node");
            break;
        } else {
            // Jika tidak, lakukan iterasi terhadap semua simpul jelajah
            for (var i = 0; i < expandNode.length; i++) {
                // Inisiasi path baru yang ditambahkan rute sebelumnya
                // Ambil nilai gn dan hn
                gn = Paths.passedpath + adjMatrix[current - 1][expandNode[i]];
                hn = heuristics(posList, finish, expandNode[i] + 1);

                // Buat sebuah jalur baru
                let newPath = new Path(expandNode[i] + 1, gn, gn + hn);
                // Salin nilai jalur sebelumnya
                newPath.copyPath(Paths);
                // Tambahkan simpul ekspan dalam jalur baru
                newPath.addPosition(expandNode[i] + 1);
                // Masukkan jalur baru dalam antrian prioritas
                listActiveNode.enqueue(newPath);
            }
        }
    }
}

```

```

        }

    /**
     * Fungsi UCS, menjalankan algoritma UCS.
     * prioritas berdasarkan g(n)
     * dengan g(n) adalah jarak dari simpul awal ke simpul n
     *
     * @function UCS
     * @param {Number} start - simpul awal
     * @param {Number} finish - simpul tujuan
     * @param {Number[][]} adjMatrix - matriks ketetanggaan
     */
    function UCS(start, finish, adjMatrix) {
        // Instansiasi simpul yang udah pernah dijelajahi
        let sudahdicek = [];
        sudahdicek.push(start);
        // Inisiasi posisi awal
        let current = start;
        // Membuat sebuah jalur awal, inisiasi peta
        let initialPath = new Path(current, 0, 0);
        initialPath.listPath.push(current);
        // Memasukkan jalur awal ke antrian simpul aktif
        let listActiveNode = new PQ();
        listActiveNode.enqueue(initialPath);

        // Selama belum ada rute yang mencapai finish
        while (true) {
            // Jika tidak ada jalur yang tersedia || semua sudah di cek
            if (listActiveNode.isEmpty()) {
                alert("There's no path to that node");
                break;
            }

            // hapus dan ambil rute paling depan dari antrian prioritas
            let Paths = listActiveNode.dequeue();
            // Ubah posisi titik analisis saat ini
            current = Paths.currentPos;
            if (!sudahdicek.includes(current)) {
                sudahdicek.push(current);
            }

            // Jika sudah mencapai tujuan, keluarkan jalur
            if (current == finish) {
                finalPath = Paths;
                break;
            }

            // Cari semua simpul jelajah dari titik saat ini
            let expandNode = getExpand(current, adjMatrix, sudahdicek);

            // lakukan iterasi terhadap semua simpul jelajah
            for (var i = 0; i < expandNode.length; i++) {
                // Inisiasi path baru yang ditambahkan rute sebelumnya
                // Ambil nilai gn dan

```

```

gn = Paths.passedpath + adjMatrix[current - 1][expandNode[i]];

    // Buat sebuah jalur baru
    let newPath = new Path(expandNode[i] + 1, gn, gn)
    // Salin nilai jalur sebelumnya
    newPath.copyPath(Paths);
    // Tambahkan simpul ekspan dalam jalur baru
    newPath.addPosition(expandNode[i] + 1);
    // Masukkan jalur baru dalam antrian prioritas
    listActiveNode.enqueue(newPath);
}
}
}

```

F. File main.js

Pada file ini, dijalankan semua fungsionalitas yang langsung terhubung ke pengguna dengan menjalankan semua fungsionalitas yang telah dideklarasikan sebelumnya.

main.js

```

// File javascript yang menjalankan fungsionalitas web
// Deklarasi variabel global yang akan digunakan pada saat eksekusi program
let maps;
let newNodeCount = 0;
let finalPath = [];
let adjMatrix = [];
let posList = [];
let markers = [];
let lines = [];

// Berkaitan dengan setup awal web
window.onload = function() {
    // Instansiasi tidak menampilkan peta pada HTML
    document.getElementsByClassName('map')[0].style.display = 'block';
    maps = L.map('map').setView([0, 0], 1);

    // Memberikan instansiasi peta kosong
    L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(maps);

    // Penanganan terhadap penambahan simpul peta dengan event double klik
    // Melakukan disable pada double click dengan zoom
    maps.doubleClickZoom.disable();

    // Melakukan penambahan simpul dengan melakukan double click
    maps.on('dblclick', function(e) {
        // Melakukan penanganan terhadap marker yang digunakan pada peta
    });
}

```

```

let marker = L.marker(e.latlng, {draggable: 'true'});
// Meminta nama simpul
var nodeName = prompt("What is the node name?");

while (nodeName === "") {
    var nodeName = prompt("What is the node name? (name cannot be empty)");
}

// Pengguna mengklik tombol cancel
if (nodeName === null) return;

// Membuat sebuah simpul baru dalam JSON dengan data baru
let newNode = {
    "id": (posList.length + 1).toString(),
    "nama": nodeName,
    "lintang": e.latlng.lat,
    "bujur": e.latlng.lng};
marker.id = newNode.id;
// Memperbaharui ID Marker
const popUpContent =
    <div class="popup-content">
        <p>${newNode.id} - ${newNode.nama}</p>
        <input type="button" class="delete-button" value="Delete Node"
onclick="deleteNode(${newNode.id});">
    </div>';
marker.bindPopup(popUpContent); // Melakukan penanganan popup menampilkan simpul
maps.addLayer(marker);

// Penanganan terhadap event dragend pada HTML
marker.on('dragend', function() {
    // Ambil nilai lintang dan bujur, perbaharui
    posList[marker.id - 1].lintang = marker.getLatLng().lat;
    posList[marker.id - 1].bujur = marker.getLatLng().lng;
    // Mengganti nilai matriks ketetangan
    for (var i = 0; i < adjMatrix.length; i++) {
        if (adjMatrix[i][marker.id - 1] != -1) {
            let distance = heuristics(posList, i + 1, marker.id).toFixed(3)
            // Perbaharui nilai jaraknya
            adjMatrix[i][marker.id - 1] = Number(distance);
            adjMatrix[marker.id - 1][i] = Number(distance);
        }
    }
    drawPathLine(); // Gambar ulang petak yang baru
    tablePathControl(); // Perbaharui nilai data simpul
});

// Memasukkan elemen baru pada variabel global
markers.push(marker);
posList.push(newNode);

```

```

// Penanganan terhadap matriks ketetanggaan
// Instansiasi matriks elemen baru, baris matriks ketetanggaan
let newRow = [];
// Menambahkan elemen secara row-wise
for (var i = 0; i < adjMatrix.length; i++) {
    adjMatrix[i].push(-1); // instansiasi tidak terhubung
    newRow.push(-1); // Menambah row baru dengan elemen -1 semua sebanyak kolom
}
// Menambah elemen column-wise
newRow.push(0); // Ingat, elemen baru pasti bertemu elemen baru
adjMatrix.push(newRow); // Menambahkan instansiasi baris baru matriks ketetanggaan

// Tangani untuk masukan
chooseNode();

// Tambah jumlah simpul baru
newNodecount += 1;
return e;
})
}

/**
* Fungsi readFile, menjalankan mekanisme pembacaan file dengan masukan JSON
*/
* @function readFile
*/
function readFile () {
// Menerima elemen yang dikirimkan dari id inputFile pada HTML
let input = document.getElementById('inputFile');

// Penanganan terhadap kasus ujung yang mungkin dimiliki masukan
// Jika masukan belum terdefinisi, keluarkan alert
if (!input.files[0]) {
    alert("Please select a file!");
}
// Jika tidak, maka ambil nilai, baca file
else {
    let file = input.files[0];
    let reader = new FileReader();
    reader.onload = initiateSearch; // Lakukan pembacaan dengan mengirim event
    reader.readAsText(file); // Baca sebagai teks
}

// Instansiasi pembacaan file dengan JSON
function initiateSearch (e) {
    let value = e.target.result;
    // Melakukan berbagai skema validasi
    try {

```

```

// Validasi apakah file masukan kosong
if (value === "") {
    throw "Your file input is empty!";
}

// Proses informasi berdasar JSON
infoParsed = JSON.parse(value.toString());
// Menangkap nilai dari masukan JSON
posList = infoParsed.posList;
adjMatrix = infoParsed.adjMatrix;

// Validasi apakah masukan sesuai dengan format
if (posList === null) {
    throw "posList` value is not defined!";
} else if (adjMatrix === null) {
    throw "adjMatrix` value is not defined!";
}

// Validasi terhadap isi file
// Jumlah elemen posList dan adjMatrix beda
if (posList.length != adjMatrix.length) {
    throw "There might be inconsistency between the `posList` and `adjMatrix` data!";
}

// Isi posList
for (var i = 0; i < posList.length; i++) {
    if (posList[i].id === "" || posList[i].id === null) {
        throw "There's an ID value in `posList` that is not well defined!";
    } else if (typeof posList[i].lintang !== "number" || typeof posList[i].bujur !== "number") {
        throw "There's lintang or bujur value in `posList` that is not well defined!";
    }
}

// Isi adjMatrix
for (var i = 0; i < adjMatrix.length; i++) {
    // Cek apakah jumlah kolom dari setiap baris sama
    if (adjMatrix.length != adjMatrix[i].length) {
        throw "The number of rows and columns in `AdjMatrix` is different!";
    }
    // Uji isi matriks ketetanggaan
    for (var j = 0; j < adjMatrix.length; j++) {
        if (i === j && adjMatrix[i][j] !== 0) {
            throw "The same node must be connected to one another!";
        } else if (adjMatrix[i][j] !== adjMatrix[j][i]) {
            throw "There must be inconsistency value for distance in `adjMatrix`!";
        }
    }
}

// Jika sampai pada tahap ini, maka peta yang terbaca sudah pasti valid

```

```

startMapSearch(); // Inisiasi penandaan posisi koordinat pada peta
drawPathLine(); // Menggambar petak path
tablePathControl(); // Menangani tabel daftar simpul dan jaraknya
chooseNode(); // Melakukan penanganan terhadap pengisian simpul yang akan dijelajahi

} catch (err) {
    if (err instanceof SyntaxError) {
        alert("There's syntax error in your input, Please check your file!");
    } else if (err instanceof TypeError) {
        alert("Your input might not in array, Please check your file!");
    } else {
        alert(err + "Please check your file");
    }
}
}

/**
 * Fungsi startMapSearch, inisiasi melakukan ilustrasi
 * Dilakukan dengan "terbang" menuju lokasi lintang dan bujur koordinat pertama masukan
 *
 * @function startMapSearch
 */
function startMapSearch () {
    // Tahap inisialisasi
    // Melakukan pembersihan terhadap layer peta yang mungkin sebelumnya di load
    for (var j = 0; j < markers.length; j++) {
        maps.removeLayer(markers[j]);
    }
    // Mengosongkan isi marker yang mungkin terisi dari sebelumnya
    markers = [];

    // Penampilan
    // Melakukan zoom-in ke lokasi koordinat pertama masukan, nilai perbesaran = 16
    maps.flyTo([parseFloat(posList[0].lintang), parseFloat(posList[0].bujur)], 16);

    // Melakukan pemasian terhadap kondisi peta setiap koordinat masukan
    let i = 1;
    // Melakukan parsing terhadap semua koordinat
    posList.forEach (function(info) {
        // Mendefinisikan isi marker dan id
        let marker = L.marker([parseFloat(info.lintang), parseFloat(info.bujur)], {draggable: 'true'});
        marker.id = i;
        i++;

        // Mendefinisikan isi pop-up peta dan menambah layer ke peta
        const popUpContent =
            <div class="popup-content">
                <p>${info.id} - ${info.nama}</p>

```

```

        <input type="button" class="delete-button" value="Delete Node"
        onclick="deleteNode(${info.id});">
    </div>';
    marker.bindPopup(popUpContent); // Melakukan penanganan popup menampilkan simpul
    maps.addLayer(marker);

    // Melakukan penanganan terhadap event dragend dari HTML
    marker.on('dragend', function() {
        // Ambil nilai lintang dan bujur, perbarui
        posList[marker.id - 1].lintang = marker.getLatLang().lat;
        posList[marker.id - 1].bujur = marker.getLatLang().lng;
        // Mengganti nilai matriks ketetanggaan
        for (var i = 0; i < adjMatrix.length; i++) {
            if (adjMatrix[i][marker.id - 1] != -1) {
                let distance = heuristics(posList, i + 1, marker.id).toFixed(3)
                // Perbarui nilai jaraknya
                adjMatrix[i][marker.id - 1] = Number(distance);
                adjMatrix[marker.id - 1][i] = Number(distance);
            }
        }
        drawPathLine();
        tablePathControl();
    });

    // Menambah marker yang telah didefinisikan
    markers.push(marker);
});
}

/**
 * Fungsi drawPathLine, menggambar "graf" pada peta
 * Penggambaran dilakukan dengan nilai marker yang menjadi masukan sebelumnya
 *
 * @function drawPathLine
 */
function drawPathLine () {
    // Tahap inisialisasi
    // Melakukan pembersihan terhadap layer peta yang mungkin sebelumnya di load
    for(var i = 0; i < lines.length; i++) {
        maps.removeLayer(lines[i]);
    }
    // Mengosongkan isi lines yang mungkin terisi dari sebelumnya
    lines = [];

    // Melakukan traversal terhadap isi matriks ketetanggaan
    for (var i = 0; i < adjMatrix.length; i++) {
        for (var j = 0; j < i; j++) {
            // Jika simpul memiliki relasi (tidak bernilai -1)
            if (adjMatrix[i][j] != -1) {

```

```

// Mengambil nilai lintang dan bujur dari marker
let latlons = [markers[i].getLatLang(), markers[j].getLatLang()];

// Menggambar polyline untuk membuat graf pada peta (berwarna hijau)
let line = L.polyline(latlons, {color: 'green'});
// Melempar nilai ke peta untuk ditampilkan
lines.push(line);
maps.addLayer(line);
}

}

}

/**
* Fungsi tablePathControl, melakukan pengendalian terhadap nilai yang terdapat
* pada tabel jarak antar simpul, melakukan penanganan juga terhadap event pergeseran simpul
*
* @function tablePathControl
*/
function tablePathControl () {
// Mengambil elemen id pada HTML untuk disimpan, inisiasi penampilan
let tableContents = document.getElementById('pathContent');
tableContents.innerHTML = "";

// Membuat tabel list simpul dari masukan
let count = 1;
// Melakukan traversal terhadap isi matriks ketetanggaan
for (var i = 0; i < adjMatrix.length; i++) {
    for (var j = 0; j < i; j++) {
        // Jika simpul memiliki relasi (tidak bernilai -1)
        if (adjMatrix[i][j] != -1) {
            // Tambahkan data setiap simpul pada elemen id HTML yang ditangkap
            tableContents.innerHTML +=
                '<tr>
                    <th scope="row">$ {count}</th>
                    <td>$ {j+1} - $ {i+1}</td>
                    <td>$ {adjMatrix[i][j]}</td>
                </tr>';
            count++;
        }
    }
}

/**
* Fungsi chooseNode, melakukan penanganan terhadap pilihan simpul yang akan dianalisis
* dilakukan dengan melakukan traversal terhadap isi matriks ketetanggaan
*
* @function chooseNode

```

```

/*
function chooseNode () {
    // Instansiasi konstanta dengan mengambil nilai elemen pada id HTML
    let pilAwal = document.getElementsByClassName('init-pos')[0];
    let pilAkhir = document.getElementsByClassName('final-pos')[0];
    let relAwal = document.getElementsByClassName('init-rels')[0];
    let relAkhir = document.getElementsByClassName('final-rels')[0];

    // Instansiasi elemen awal kosong tanpa pemilihan
    pilAwal.innerHTML = "";
    pilAkhir.innerHTML = "";
    relAwal.innerHTML = "";
    relAkhir.innerHTML = "";

    // Menambahkan nilai opsi untuk setiap id pada matriks ketetanggaan
    for (var i = 0; i < adjMatrix.length; i++) {
        pilAwal.innerHTML += <option value="$i">$ {posList[i].id} - $ {posList[i].nama}</option>
        pilAkhir.innerHTML += <option value="$i">$ {posList[i].id} - $ {posList[i].nama}</option>
        relAwal.innerHTML += <option value="$i">$ {posList[i].id} - $ {posList[i].nama}</option>
        relAkhir.innerHTML += <option value="$i">$ {posList[i].id} - $ {posList[i].nama}</option>
    }
}

/**
 * Fungsi addRelation, melakukan penambahan relasi antar simpul
 * Melakukan penanganan terhadap semua kasus penambahan simpul yang mungkin
 *
 * @function addRelation
*/
function addRelation () {
    // Penanganan jika peta belum terload, sehingga belum ada data yang terbaca
    if (adjMatrix.length === 0 && posList.length === 0) {
        alert("You haven't load any map yet!");
    } else {
        // Ambil nilai dari masukan pengguna melalui elemen pada kelas di HTML
        let init_rels = document.getElementsByClassName('init-rels')[0].value;
        let final_rels = document.getElementsByClassName('final-rels')[0].value;

        // Penanganan kasus penambahan relasi
        // Tidak dapat menambah relasi pada simpul itu sendiri
        if (init_rels === final_rels) {
            alert("You can't add a relation to node itself");
        }
        // Tidak bisa menambah relasi pada simpul yang sudah terhubung sebelumnya
        else if (adjMatrix[init_rels][final_rels] != -1) {
            alert("The relation already exist");
        }
        // Jika kasus sudah valid
        else {
}
}

```

```

// Gunakan fungsi heuristics untuk mengambil nilai jarak garis lurus
let distance = heuristics(posList, parseInt(final_rels) + 1, parseInt(init_rels) + 1).toFixed(3);

// Isi matriks ketetanggan dengan angka jarak
adjMatrix[init_rels][final_rels] = Number(distance);
adjMatrix[final_rels][init_rels] = Number(distance);

// Melakukan penanganan terhadap nilai jarak simpul pada tabel dan tampilan peta
tablePathControl();
drawPathLine();
}

}

}

/***
* Fungsi deleteNode, melakukan penghapusan simpul dengan id tertentu
* Melakukan pembaharuan terhadap nilai posList dan adjMatrix
*
* @function deleteNode
*/
function deleteNode (NodeID) {
    var index = -1; // Inisiasi kosong
    // Melakukan pencarian index melalui data ID
    posList.find (function (item, i) {
        if (item.id === NodeID.toString()) {
            index = i;
            return i;
        }
    });
}

// Menghapus isi poslist dengan id terkait
posList.splice(index, 1);
// Menghapus baris pada adjMatrix
adjMatrix.splice(index, 1);
// Menghapus semua baris yang berhubungan dengan simpul ini
for (var i = 0; i < adjMatrix.length; i++) {
    adjMatrix[i].splice(index, 1);
}

// Layering pada peta
maps.removeLayer(markers[index]);
markers.splice(index, 1);

// Melakukan update pada kondisi peta
drawPathLine(); // Menggambar petak path
tablePathControl(); // Menangani tabel daftar simpul dan jaraknya
chooseNode(); // Melakukan penanganan terhadap pengisian simpul yang akan dijelajah
}

```

```

/**
 * Fungsi saveFile, melakukan penanganan terhadap opsi penyimpanan hasil
 * perubahan terhadap peta pada sebuah file JSON
 *
 * @function saveFile
 */
function saveFile() {
    // Penanganan jika peta belum terload, sehingga belum ada data yang terbaca
    if (adjMatrix.length === 0 && posList.length === 0) {
        alert("You haven't load any map yet!");
    } else {
        // Instansiasi objek yang akan disimpan dalam JSON
        let saveObject = {posList: posList, adjMatrix: adjMatrix};

        // Melakukan konversi balik masukan ke JSON
        let dataHref = "data:text/json;charset=utf-8," + encodeURIComponent(JSON.stringify(saveObject));
        let downloader = document.createElement('a'); // Membuat elemen instansiasi

        // Menyimpan file dalam default map.JSON
        downloader.setAttribute("href", dataHref);
        downloader.setAttribute("download", "map.json");
        downloader.click();
        downloader.remove();
    }
}

/**
 * Fungsi doAlgo, melakukan pemrosesan algoritma berdasarkan nilai flag masukan
 *
 * @function doAlgo
 * @param {Number} flag - 1 untuk A*, 2 untuk UCS
 */
function doAlgo (flag) {
    // Inisialisasi
    // Jika sebelumnya telah terdefinisi sebuah peta solusi, ubah kembali menjadi warna hijau
    // Ilustrasikan dalam peta masukan
    // Kosongkan dulu isi finalPath sebelum proses selanjutnya
    finalPath = [];

    // Pemrosesan algoritma
    // Penanganan jika peta belum terload, sehingga belum ada data yang terbaca
    if (adjMatrix.length === 0 && posList.length === 0) {
        alert("You haven't load any map yet!");
    } else {
        // Inisiasi posisi, mengambil nilai posisi awal dan akhir dari HTML
        initialPosition = document.getElementsByClassName('init-pos')[0].value;
        finalPosition = document.getElementsByClassName('final-pos')[0].value;

        // Pemrosesan berdasarkan flag masukan
    }
}

```

```

if(flag === 1) {
    // Melakukan pemrosesan menggunakan A*
    AStar(parseInt(initialPosition) + 1, parseInt(finalPosition) + 1, adjMatrix, posList);
} else {
    // Melakukan pemrosesan menggunakan UCS
    UCS(parseInt(initialPosition) + 1, parseInt(finalPosition) + 1, adjMatrix);
}

// Mencetak hasil pada layar, lakukan pemrosesan pada kelas tertentu di HTML
elmtPath = document.getElementsByClassName('path')[0];

// Penanganan terhadap seluruh kemungkinan penampilan grafik
if(finalPath.length === 0) {
    // Jika panjang path kosong, maka tidak ada jalur
    elmtPath.innerHTML = '<p>Path not found!</p>';
} else {
    // Jika ada, cetak path
    if(flag === 1) {
        // Cetak sebagai hasil algoritma A*
        elmtPath.innerHTML = '<h4>Result using A* Algorithm</h4>';
        elmtPath.innerHTML += '<p>Path : ${finalPath.printPath()} | Distance : ${finalPath.getPriority().toFixed(3)} km</p>';
    } else {
        // Cetak sebagai hasil algoritma UCS
        elmtPath.innerHTML = '<h4>Result using UCS Algorithm</h4>';
        elmtPath.innerHTML += '<p>Path : ${finalPath.printPath()} | Distance : ${finalPath.getPriority().toFixed(3)} km</p>';
    }

    // Ilustrasikan dalam peta masukan
    let pointPos = [];
    // Lakukan redraw untuk setiap peta masukan
    drawPathLine();
    // Instansiasi objek perubahan nilai
    for(let i = 0; i < finalPath.listPath.length; i++) {
        pointPos.push(markers[finalPath.listPath[i] - 1].getLatLng());
    }

    // Buat sebuah polyline berdasarkan hasil pemrosesan algoritma dengan marker merah
    // Menandakan solusi pencarian
    let line = L.polyline(pointPos, {color: 'red'});
    // Lempar hasil ke peta untuk ditampilkan
    lines.push(line);
    maps.addLayer(line);
}
}

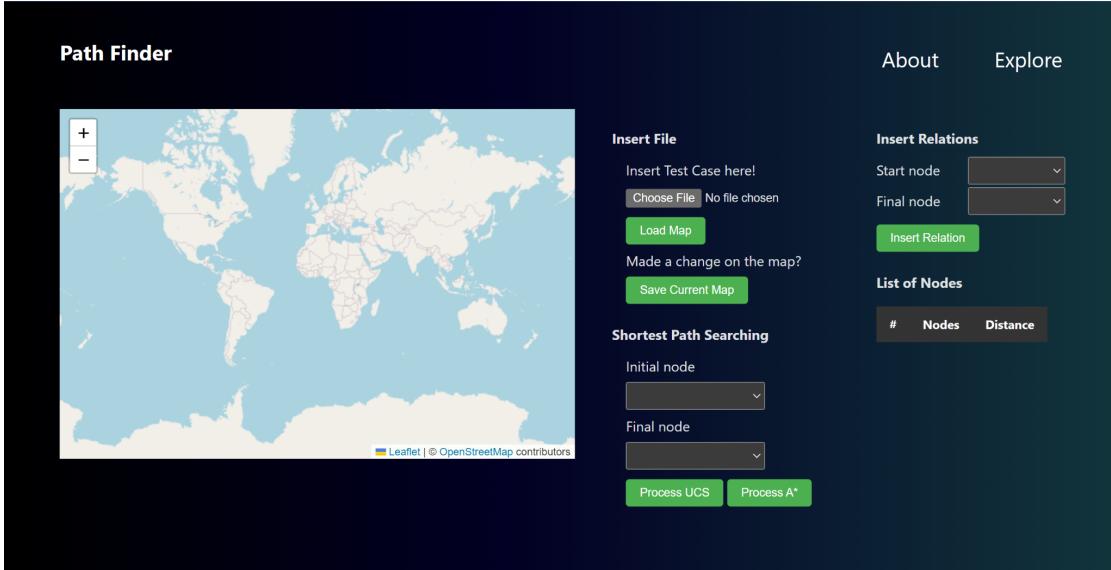
```

BAB IV

PENGUJIAN PROGRAM

A. Tampilan Awal

Berikut adalah tampilan awal website yang telah dibuat.



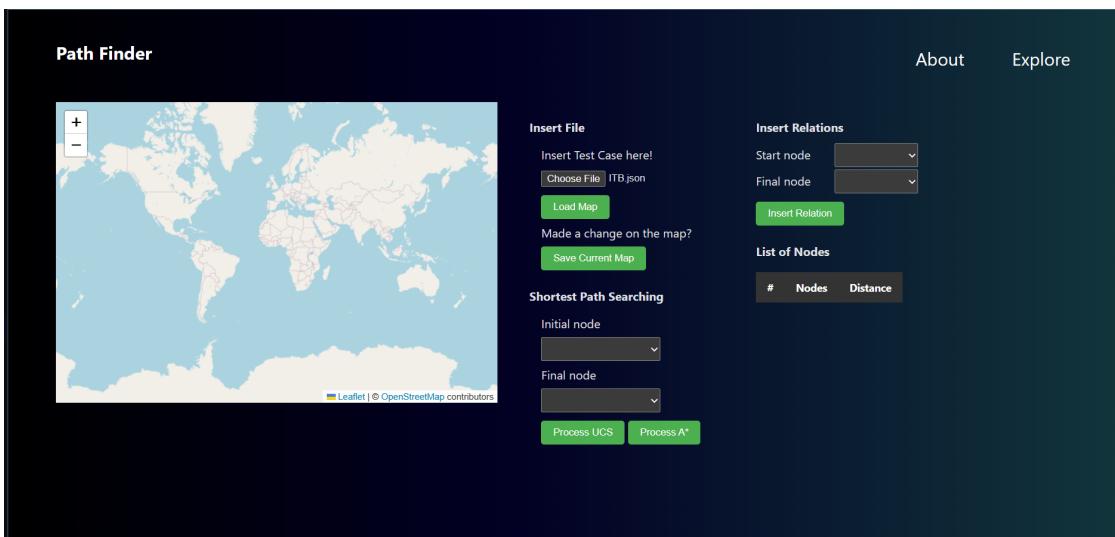
Gambar 4.1. Tampilan awal website yang dibuat.

B. Skema Penerimaan Masukan

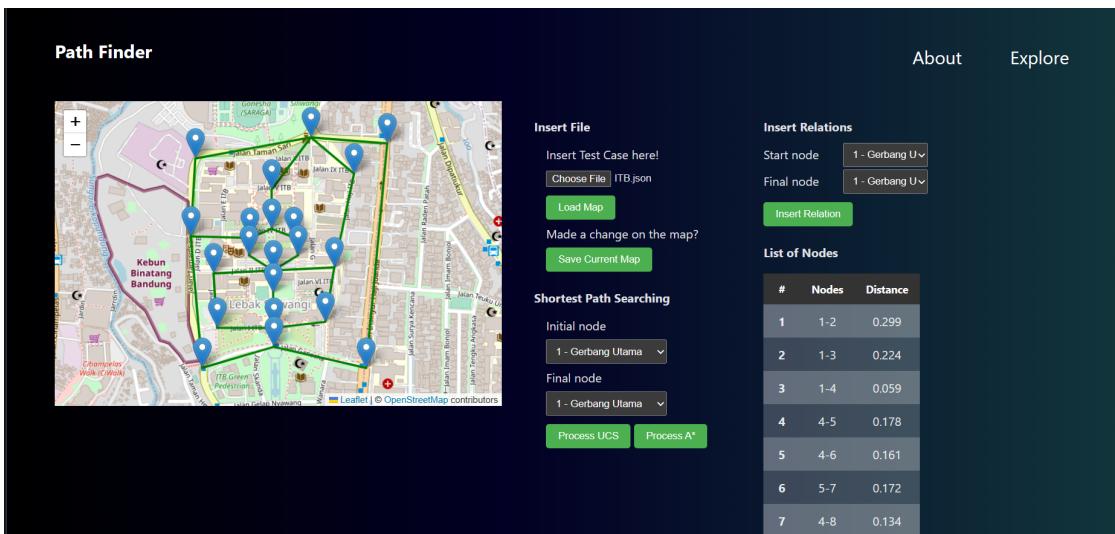
Berikut adalah konfigurasi peta masukan yang digunakan.

Gambar 4.2.1. Contoh input untuk skema penerimaan masukan (file ITB.json).

1. Menerima peta masukan

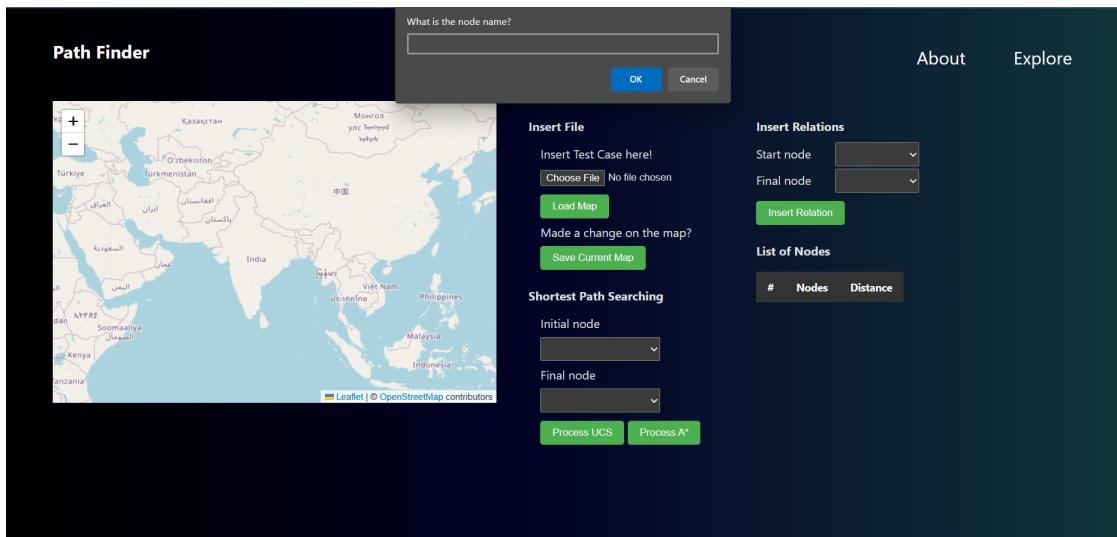


Gambar 4.2.2. Tampilan setelah memilih file masukan.

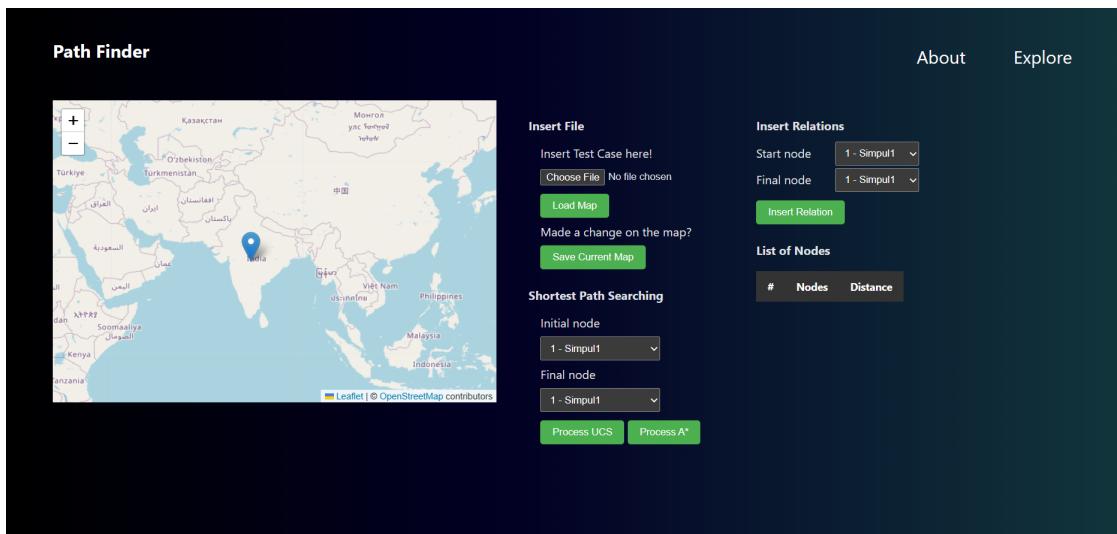


Gambar 4.2.3. Tampilan setelah mengklik "Load Map".

2. Menerima simpul masukan dari peta

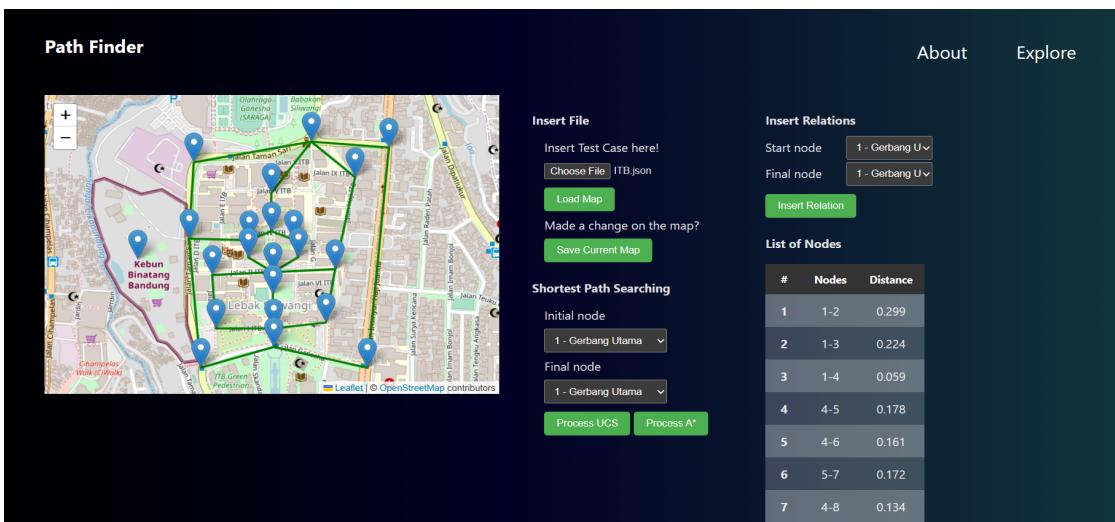


Gambar 4.2.4. Tampilan setelah melakukan aksi *Double click* pada peta.



Gambar 4.2.5. Tampilan setelah memasukan simpul baru.

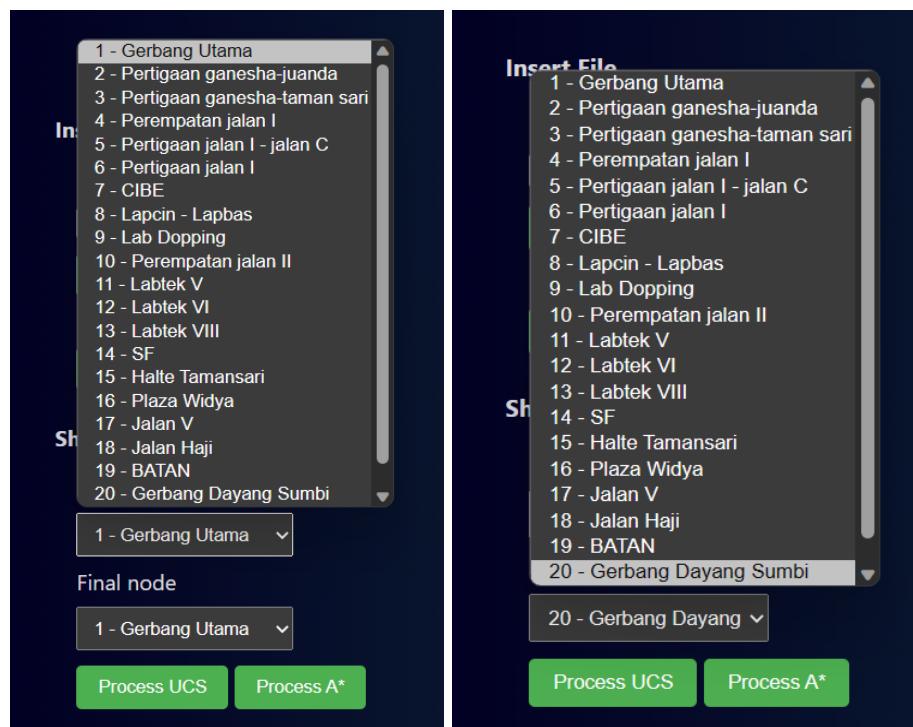
3. Kombinasi kedua metode



Gambar 4.2.6. Tampilan setelah memasukan file dan membuat simpul baru di Kebun Binatang Bandung.

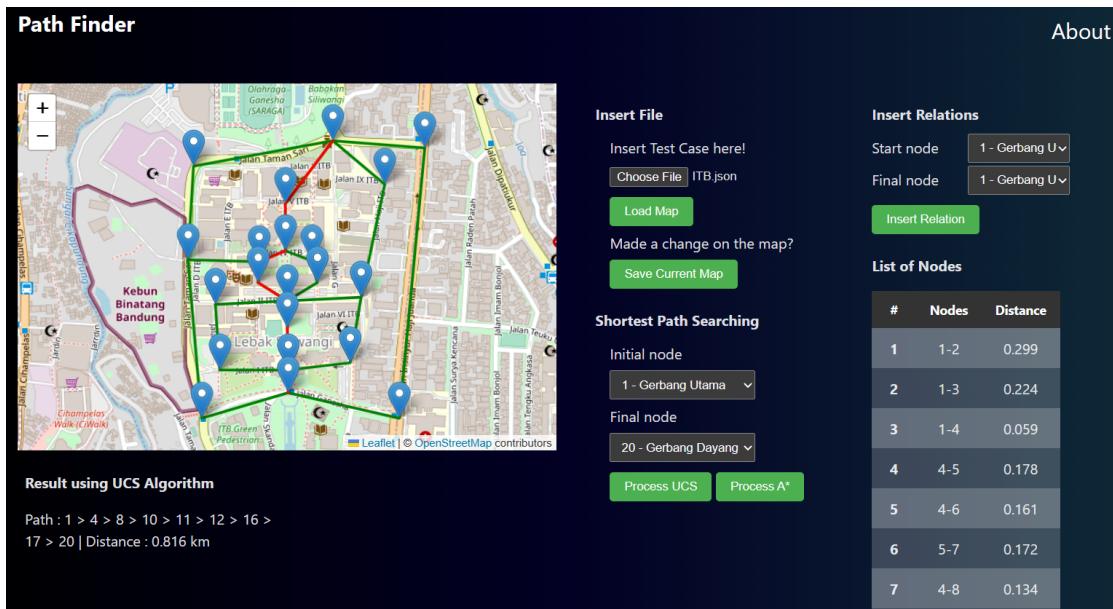
C. Skema Pemrosesan Solusi

1. Pemilihan simpul



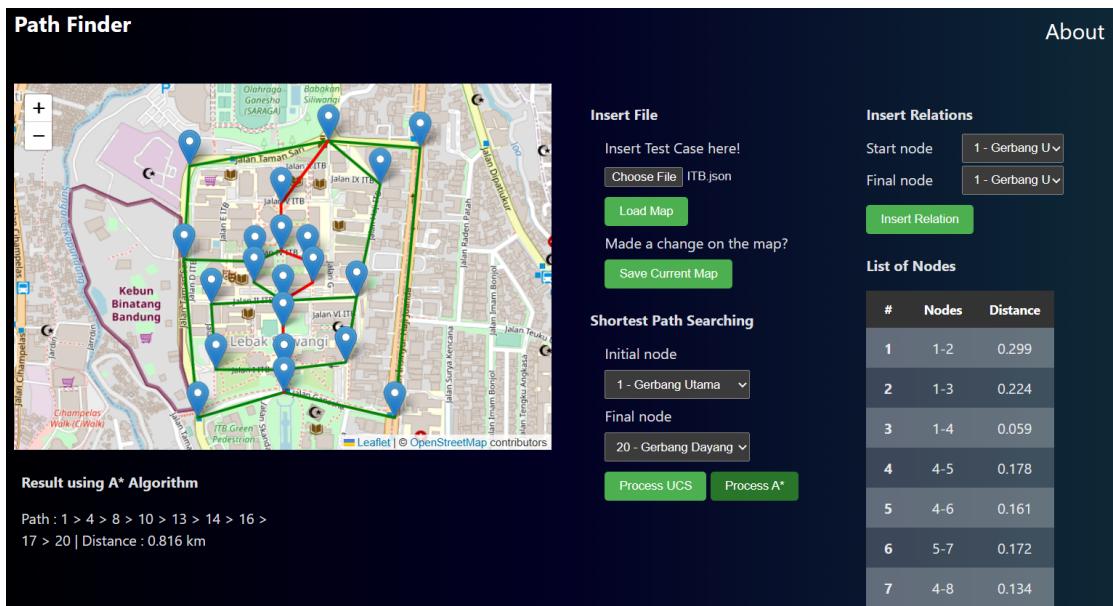
Gambar 4.3.1. Pemilihan simpul awal (kiri) dan simpul tujuan (kanan).

2. Pemrosesan menggunakan Algoritma UCS



Gambar 4.3.2. Pemrosesan pencarian jalur dengan algoritma UCS.

3. Pemrosesan menggunakan Algoritma A*



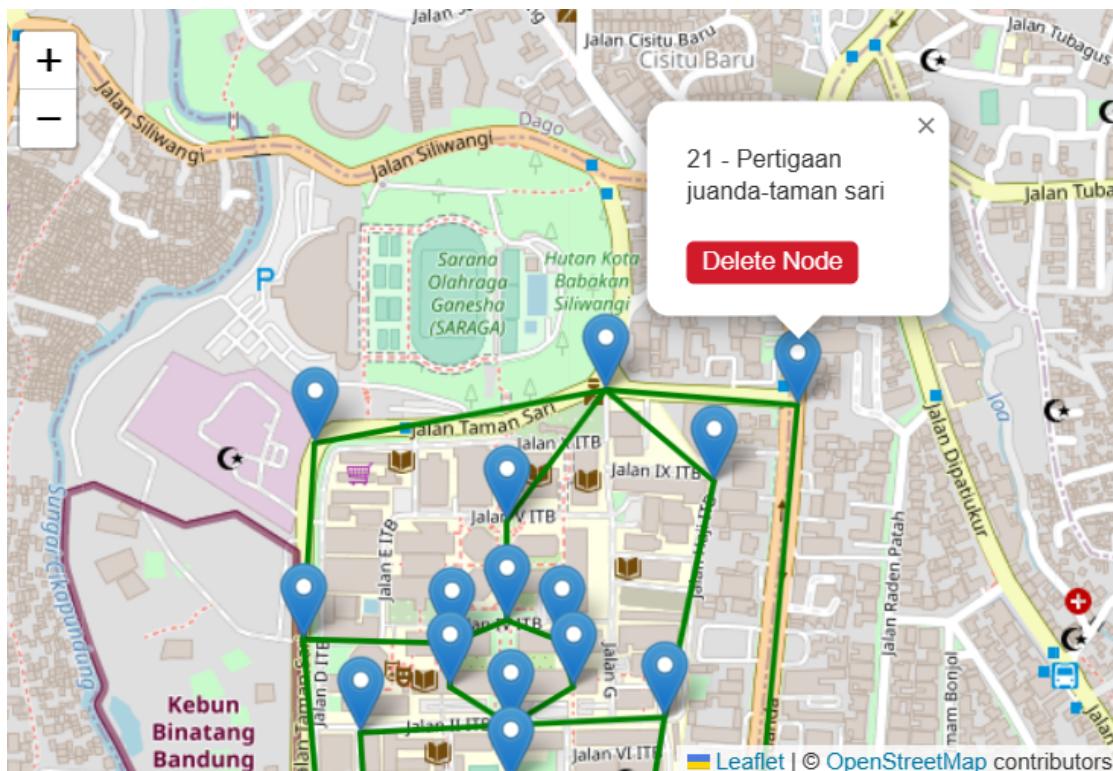
Gambar 4.3.3. Pemrosesan pencarian jalur dengan algoritma A*.

Jika diperhatikan dengan baik, ada kemungkinan hasil jalur yang dihasilkan dari algoritma UCS berbeda dengan yang dihasilkan oleh algoritma A*, contohnya pada kasus diatas. Akan tetapi, jika disimak lebih baik, jarak tempuh yang dihasilkan juga sama, artinya bahwa keduanya pasti menghasilkan hasil pencarian jalur terpendek yang sangkil meskipun

mungkin memiliki urutan jalur yang berbeda dengan catatan bahwa heuristik yang digunakan pada algoritma A* dapat diterima (*admissible*).

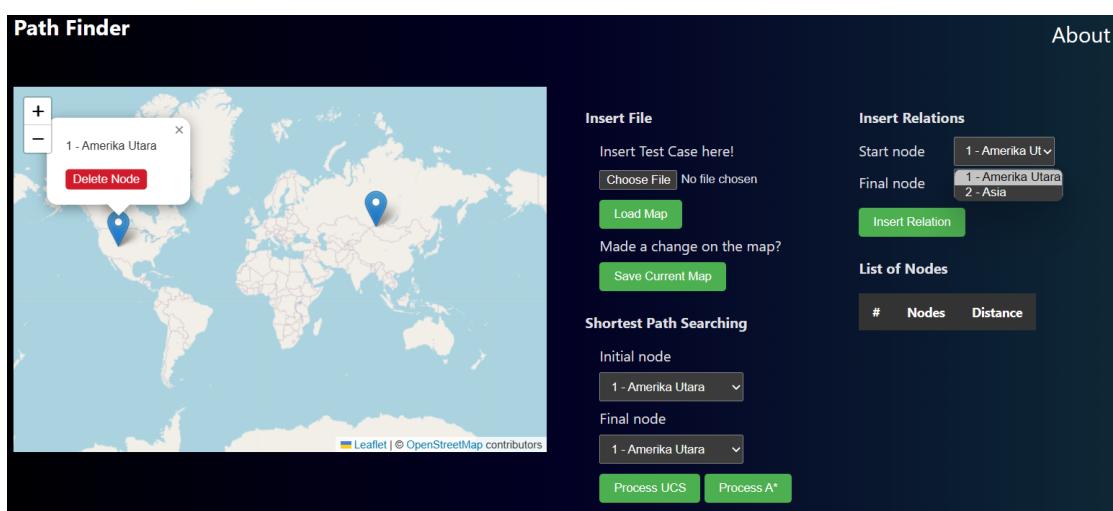
D. Beberapa Fungsionalitas Tambahan

1. Tampilan informasi simpul

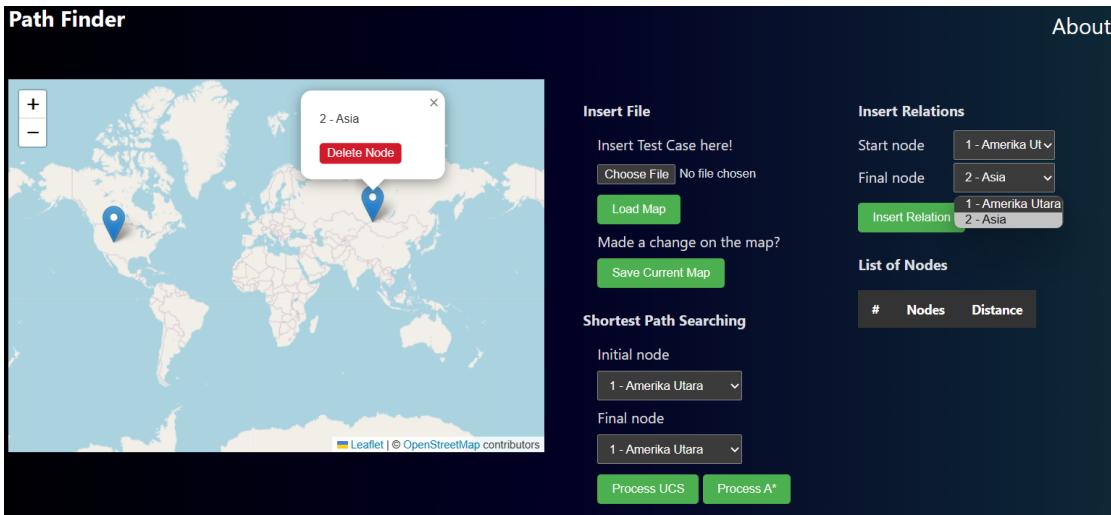


Gambar 4.4.1. Tampilan informasi simpul setelah menekan salah satu simpul.

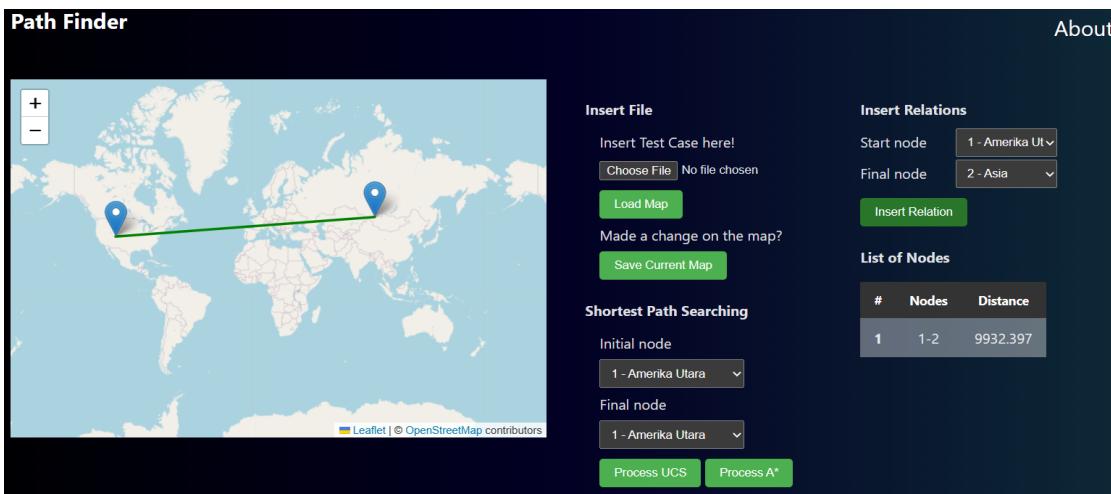
2. Penambahan relasi



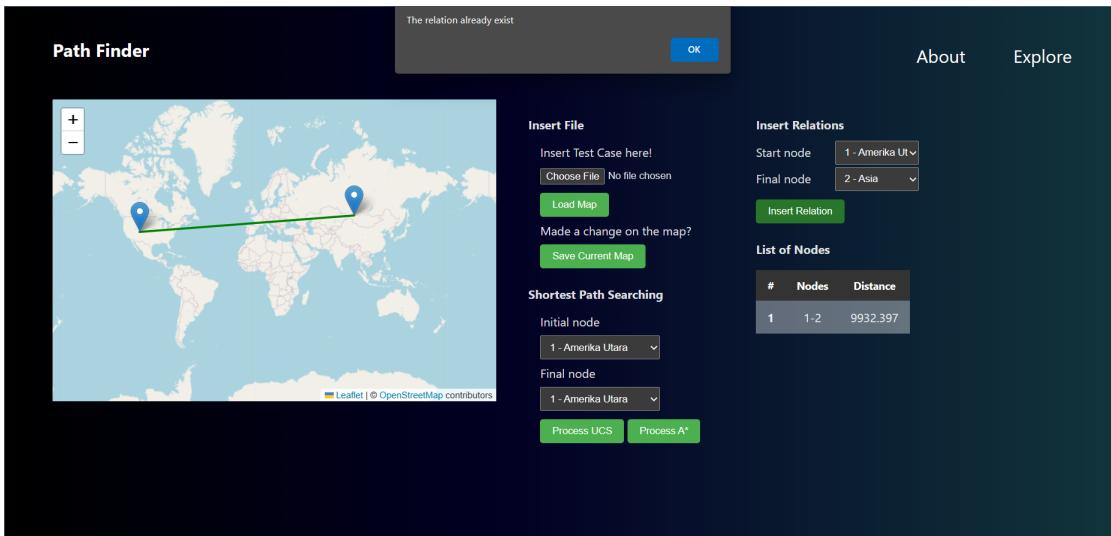
Gambar 4.4.2. Pemilihan simpul awal (*Start node*).



Gambar 4.4.3. Pemilihan simpul akhir (*Final node*).

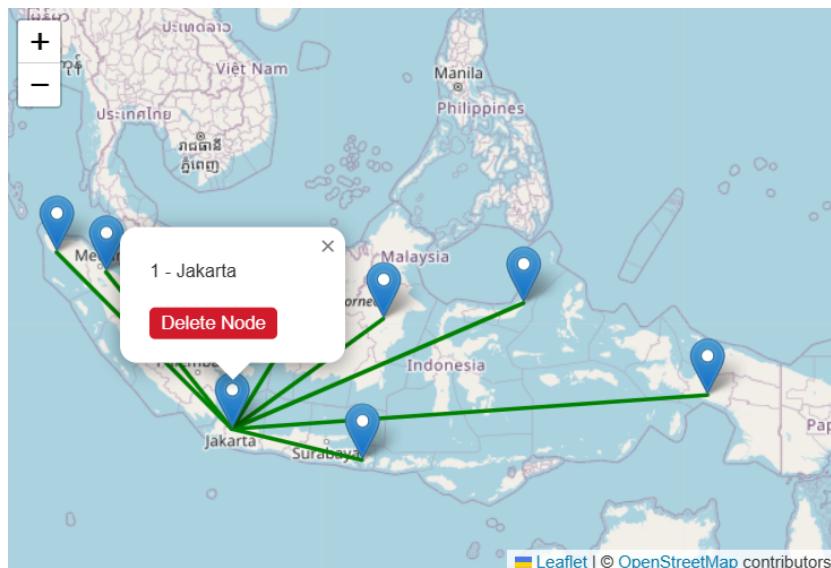


Gambar 4.4.4. Tampilan setelah mengklik tombol “*Insert Relation*”.

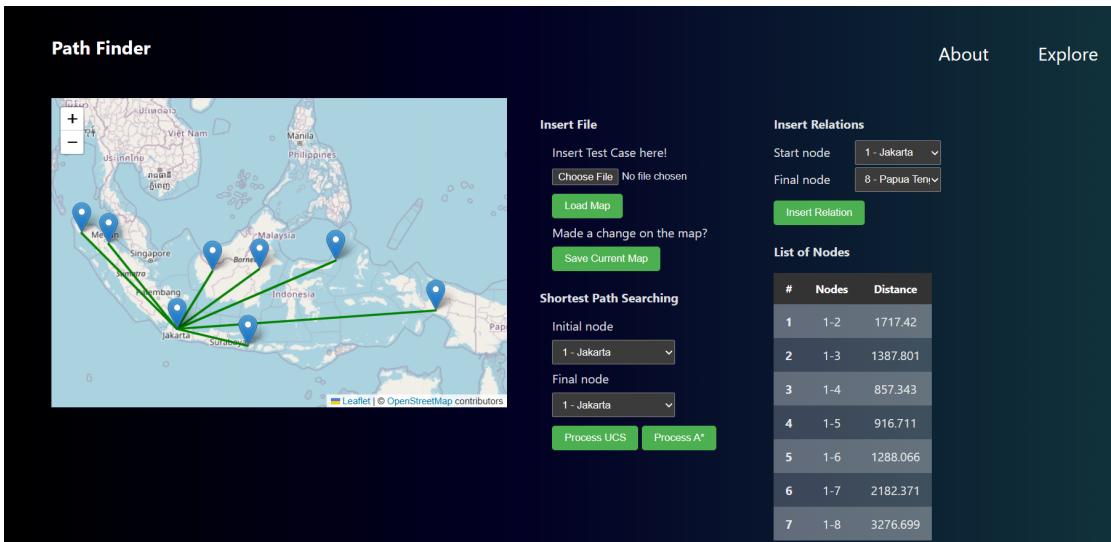


Gambar 4.4.5. Tampilan jika menekan tombol “*Insert Relation*” pada relasi yang sudah ada sebelumnya.

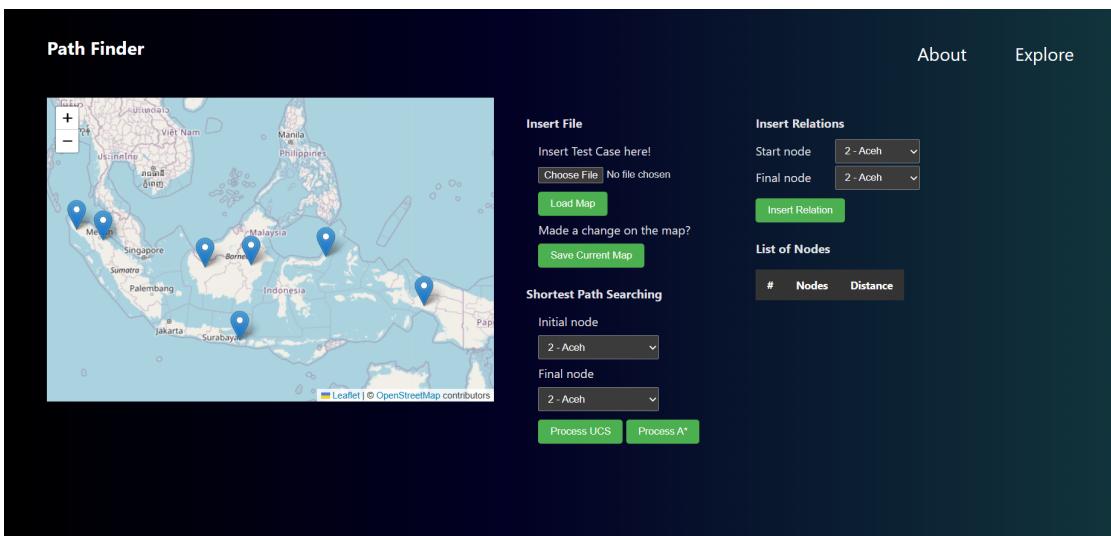
3. Penghapusan simpul



Gambar 4.4.6. Posisi simpul bernama Jakarta.

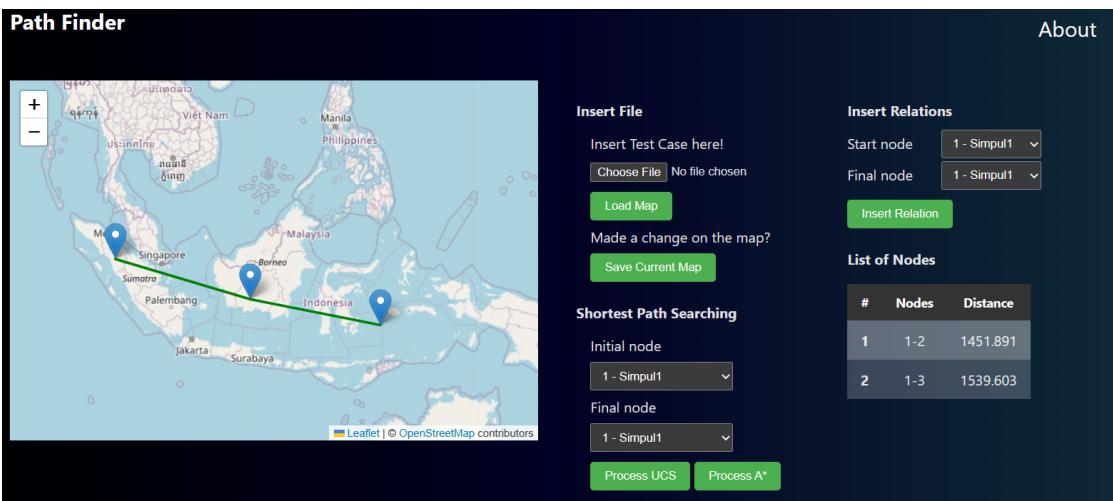


Gambar 4.4.7. Tampilan sebelum simpul bernama Jakarta dihapus.

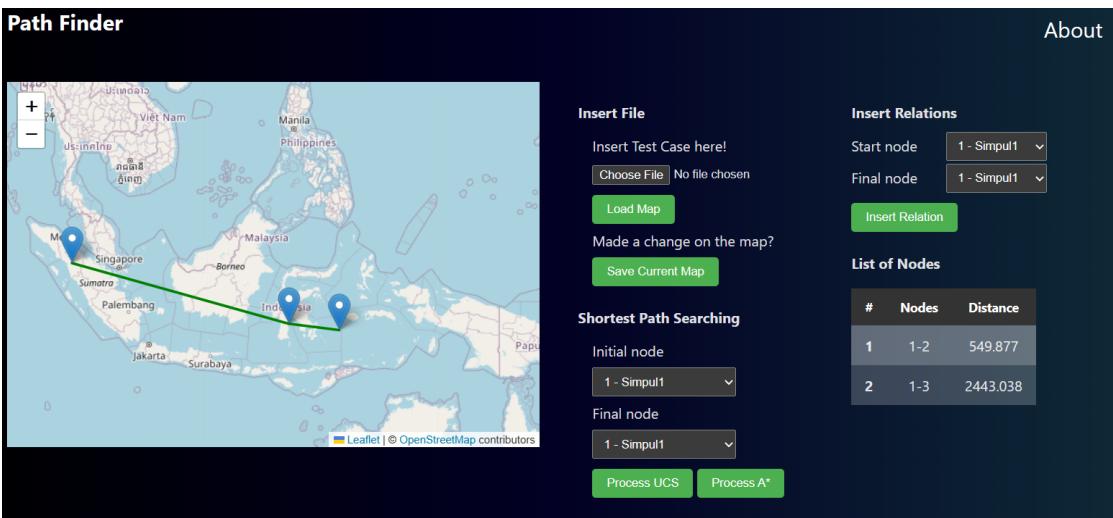


Gambar 4.4.8. Tampilan setelah simpul bernama Jakarta dihapus.

4. Pergeseran Simpul



Gambar 4.4.9. Tampilan sebelum Simpul1 digeser.



Gambar 4.4.10. Tampilan setelah menggeser Simpul1. Terlihat banyak perubahan data jarak pada tabel “List of Nodes”.

5. Menyimpan konfigurasi file

The screenshot shows the Path Finder application interface. On the left is a world map with four blue location markers. The sidebar on the right contains several configuration sections:

- Insert File:** A section for "Insert Test Case here!" with a "Choose File" button and a message "No file chosen". Below it are "Load Map" and "Save Current Map" buttons.
- Insert Relations:** A section with "Start node" set to "1 - Amerika Utara" and "Final node" also set to "1 - Amerika Utara". There is a green "Insert Relation" button.
- List of Nodes:** A table with three rows:

#	Nodes	Distance
1	1 - Amerika Utara	
2	1 - Amerika Utara	
3	1 - Amerika Utara	
- Shortest Path Searching:** A section with "Initial node" set to "1 - Amerika Utara" and "Final node" set to "1 - Amerika Utara". It includes "Process UCS" and "Process A*" buttons.

Below the sidebar is a code editor window showing a JSON configuration file:

```

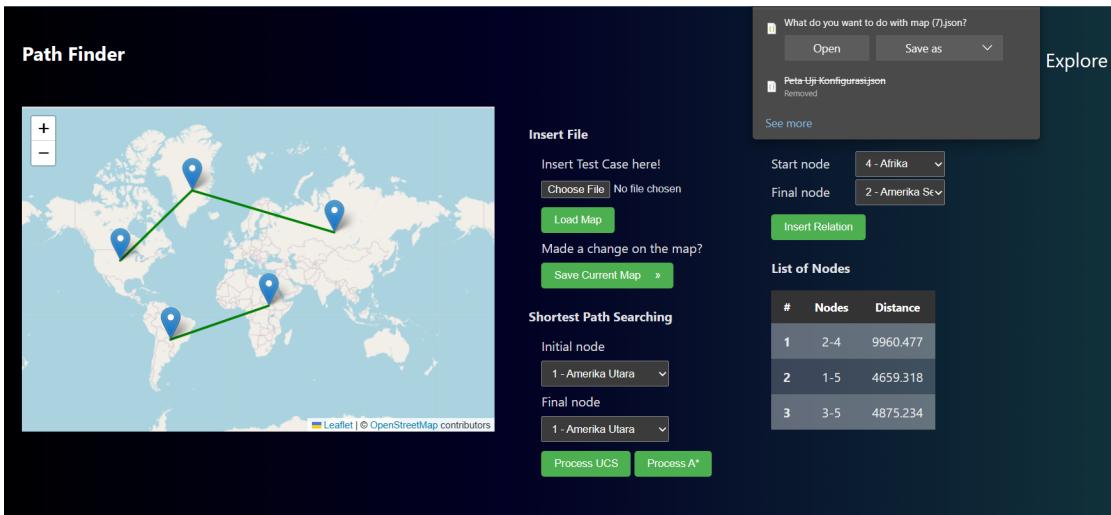
test > {} Uji Konfigurasi.json > ...
1
2  {
3    "posList": [
4      {"id": "1", "nama": "Amerika Utara", "lintang": 42.58750064468325, "bujur": -101.25000000000001},
5      {"id": "2", "nama": "Amerika Selatan", "lintang": -21.283849397812993, "bujur": -56.953125},
6      {"id": "3", "nama": "Asia", "lintang": 58.09613019166115, "bujur": 87.18750000000001},
7      {"id": "4", "nama": "Afrika", "lintang": 7.793591434281613, "bujur": 29.531250000000004}],
8
9  "adjMatrix": [
10   [0,-1,-1,-1],
11   [-1,0,-1,-1],
12   [-1,-1,0,-1],
13   [-1,-1,-1,0]]
14 }

```

Gambar 4.4.11. Kondisi awal tampilan (atas) dan konfigurasi file masukan (bawah).

The screenshot shows the Path Finder application interface after processing. The world map now has a green line connecting the node in North America (top-left) to the node in Africa (bottom-right). The sidebar on the right remains the same as in the previous screenshot.

Gambar 4.4.12. Kondisi akhir tampilan setelah konfigurasi peta masukan diterima dan ditambahkan simpul baru bernama “Greenland” dengan konfigurasi diatas.



Gambar 4.4.13. Tampilan setelah mengklik “Save Current Map”.

```
test > {} Uji Konfigurasi Hasil.json > ...
1   {"posList":[{"id": "1", "nama": "Amerika Utara", "lintang": 42.58750064468325, "bujur": -101.25000000000001}, {"id": "2", "nama": "Amerika Selatan", "lintang": -21.283849397812993, "bujur": -56.953125}, {"id": "3", "nama": "Asia", "lintang": 58.09613019166115, "bujur": 87.18750000000001}, {"id": "4", "nama": "Afrika", "lintang": 7.793591434281613, "bujur": 29.53125000000004}, {"id": "5", "nama": "Greenland", "lintang": 72.82596352689488, "bujur": -37.96875000000001}]};
```

```
test > {} Uji Konfigurasi Hasil.json > ...
1
2   {
3     "posList": [
4       {"id": "1", "nama": "Amerika Utara", "lintang": 42.58750064468325, "bujur": -101.25000000000001},
5       {"id": "2", "nama": "Amerika Selatan", "lintang": -21.283849397812993, "bujur": -56.953125},
6       {"id": "3", "nama": "Asia", "lintang": 58.09613019166115, "bujur": 87.18750000000001},
7       {"id": "4", "nama": "Afrika", "lintang": 7.793591434281613, "bujur": 29.53125000000004},
8       {"id": "5", "nama": "Greenland", "lintang": 72.82596352689488, "bujur": -37.96875000000001}],
9
10    "adjMatrix": [
11      [0, -1, -1, -1, 4659.318],
12      [-1, 0, -1, 9960.477, -1],
13      [-1, -1, 0, -1, 4875.234],
14      [-1, 9960.477, -1, 0, -1],
15      [4659.318, -1, 4875.234, -1, 0]]]
```

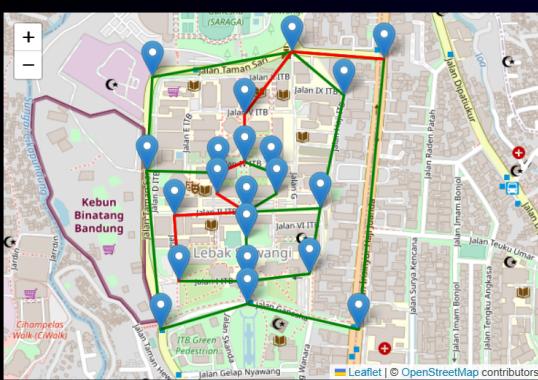
Gambar 4.4.14 Kondisi akhir file masukan setelah disimpan (atas) dan setelah disunting (bawah).

E. Kasus Uji (*Test Case*)

Berikut adalah kasus uji yang diujikan pada *website* yang telah dibuat.

1. Uji Fungsionalitas

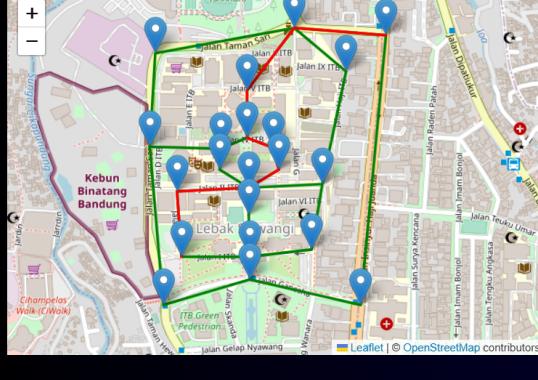
Path Finder



Result using UCS Algorithm

Path : 5 > 7 > 10 > 11 > 12 > 16 > 17 >
20 > 21 | Distance : 1.150 km

Path Finder



Result using A* Algorithm

Path : 5 > 7 > 10 > 13 > 14 > 16 > 17 >
20 > 21 | Distance : 1.150 km

Insert File

Choose File ITB.json

Load Map

Made a change on the map?

Save Current Map
Insert Relation

Shortest Path Searching

#	Nodes	Distance
1	1-2	0.299
2	1-3	0.224
3	1-4	0.059
4	4-5	0.178
5	4-6	0.161
6	5-7	0.172
7	4-8	0.134

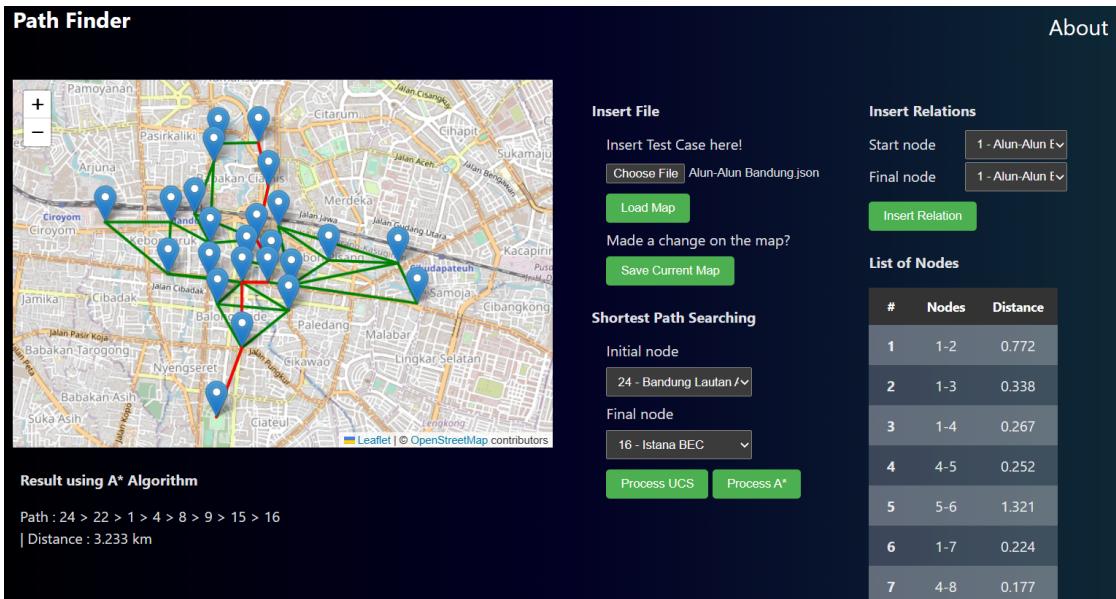
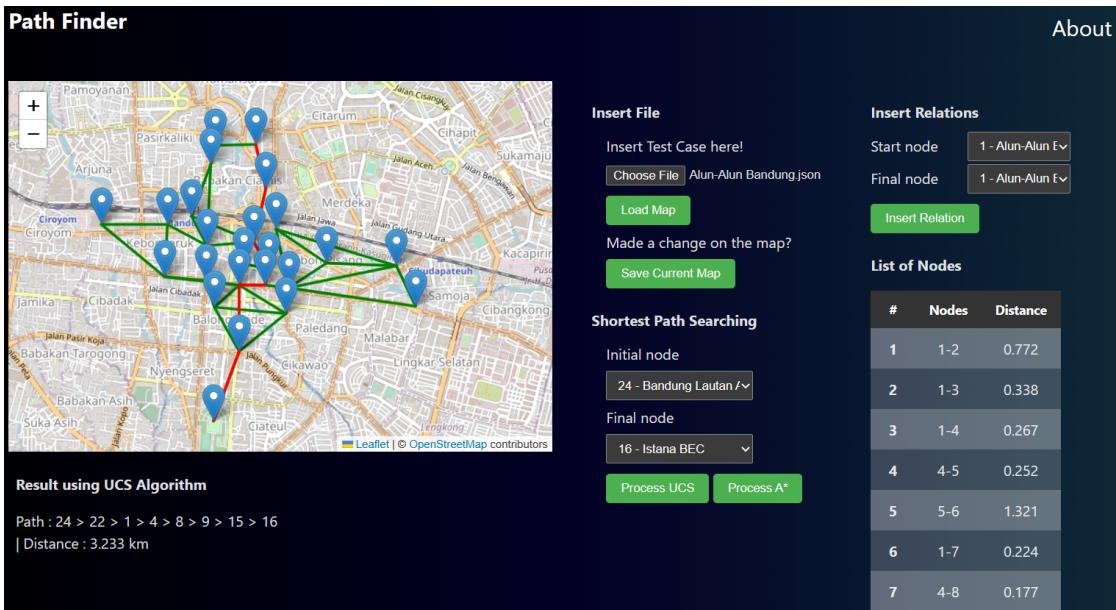
Initial node

5 - Pertigaan jalan I - j...

Final node

21 - Pertigaan juanda -...

Process UCS
Process A*



Gambar 4.5.2 Uji kasus file Alun-alun Bandung.json dengan algoritma UCS (atas) dan A* (bawah).

Path Finder

Result using UCS Algorithm

Path : 4 > 3 > 15 > 17 > 19 > 20 |
Distance : 3.486 km

Insert File
Insert Test Case here!
 Buahbatu.json

Insert Relations
Start node
Final node

List of Nodes

#	Nodes	Distance
1	1-2	0.801
2	1-3	1.158
3	1-4	0.608
4	2-4	0.449
5	3-4	0.58
6	2-5	0.535
7	3-5	0.329

Shortest Path Searching
Initial node
Final node

Path Finder

Result using A* Algorithm

Path : 4 > 3 > 15 > 17 > 19 > 20 |
Distance : 3.486 km

Insert File
Insert Test Case here!
 Buahbatu.json

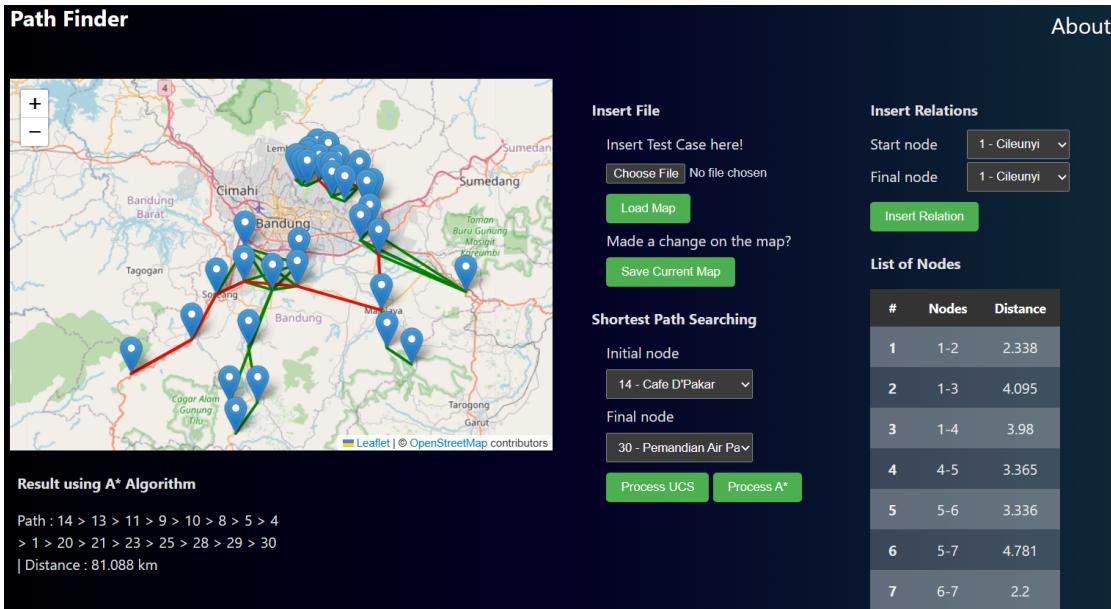
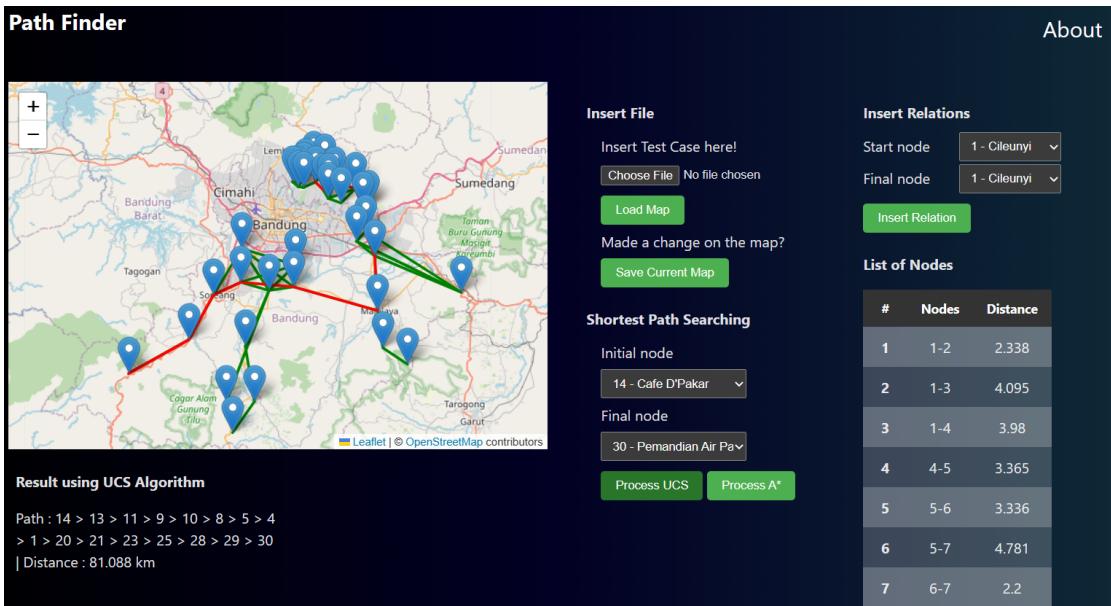
Insert Relations
Start node
Final node

List of Nodes

#	Nodes	Distance
1	1-2	0.801
2	1-3	1.158
3	1-4	0.608
4	2-4	0.449
5	3-4	0.58
6	2-5	0.535
7	3-5	0.329

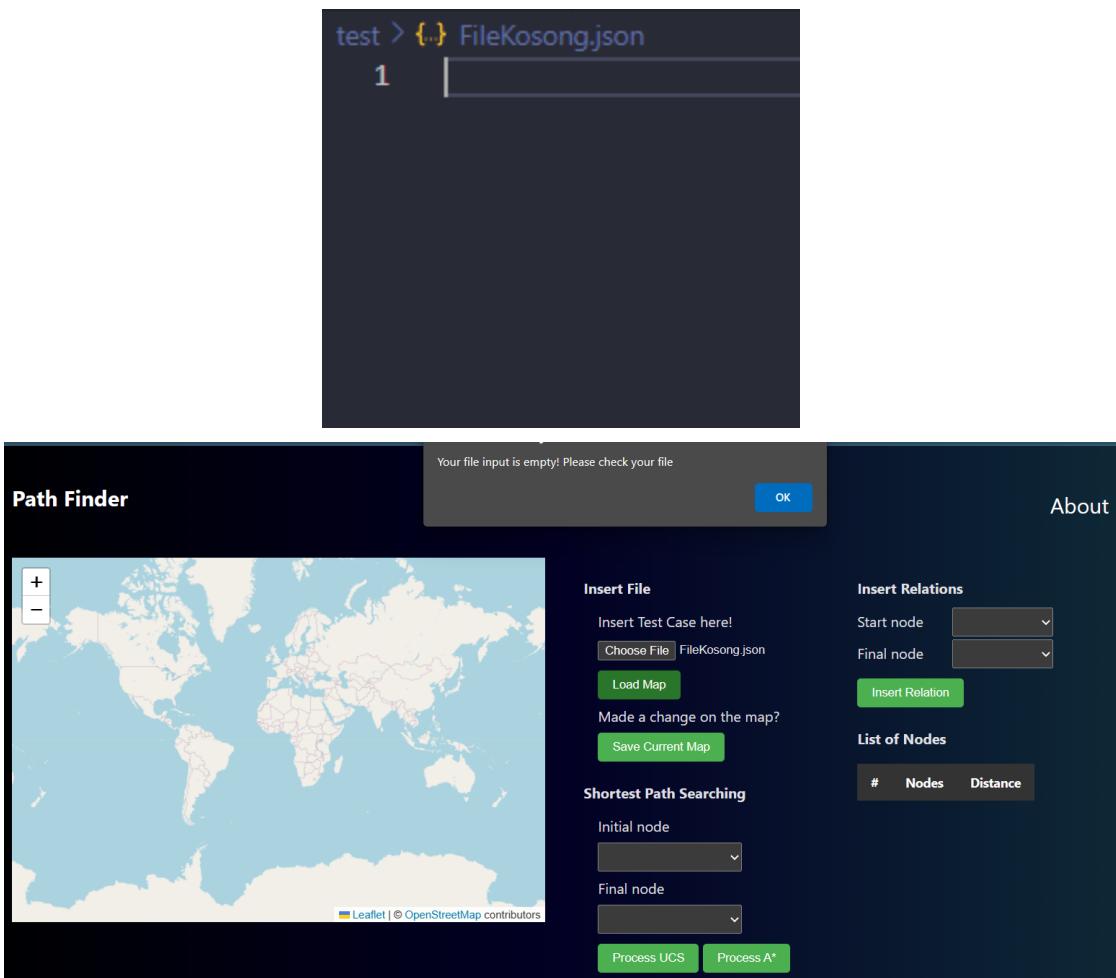
Shortest Path Searching
Initial node
Final node

Gambar 4.5.3 Uji kasus file Buahbatu.json dengan algoritma UCS (atas) dan A* (bawah).

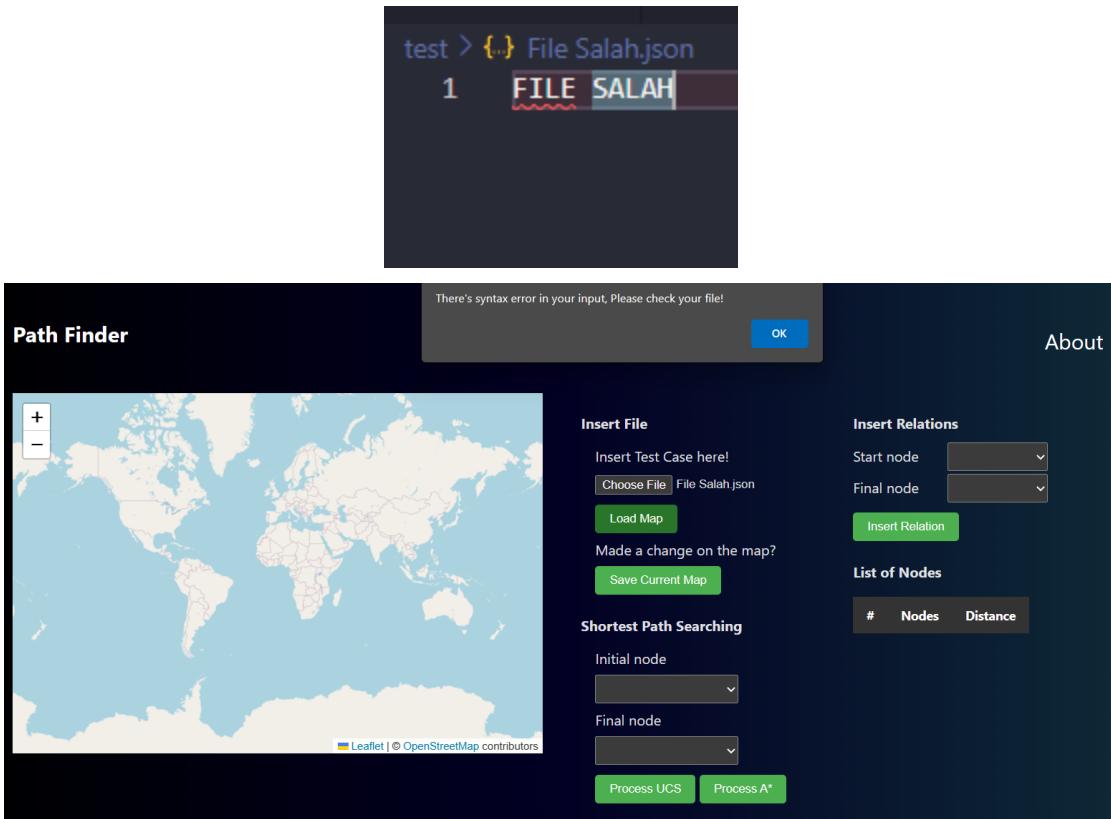


Gambar 4.5.4 Uji kasus file Kabupaten Bandung.json dengan algoritma UCS (atas) dan A* (bawah).

2. Uji Kasus Ujung (*edge cases*)



Gambar 4.5.5 Uji kasus isi *file* kosong.

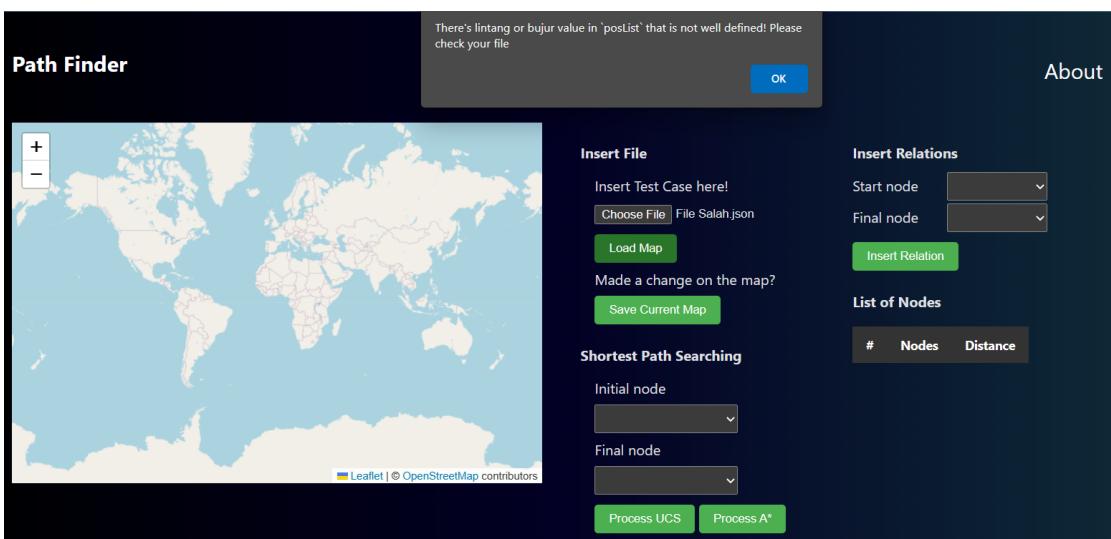


Gambar 4.5.6. Uji kasus isi file tidak sesuai.

```

test > {} File Salah.json > [ ] posList > () 4 > lintang
1
2 { "posList": [
3   {"id":"1","nama":"Amerika Utara","lintang":42.58750064468325,"bujur":-101.25000000000001},
4   {"id":"2","nama":"Amerika Selatan","lintang":-21.283849397812993,"bujur":-56.953125},
5   {"id":"3","nama":"Asia","lintang":58.09613019166115,"bujur":87.18750000000001},
6   {"id":"4","nama":"Afrika","lintang":7.793591434281613,"bujur":29.531250000000004},
7   {"id":"5","nama":"Greenland","lintang":"72.82596352689488","bujur":-37.96875000000001}],
8
9   "adjMatrix": [
10    [0,-1,-1,-1,4659.318],
11    [-1,0,-1,9960.477,-1],
12    [-1,-1,0,-1,4875.234],
13    [-1,9960.477,-1,0,-1],
14    [4659.318,-1,4875.234,-1,0]]
15 }

```



Gambar 4.5.7. Uji kasus nilai pada posList salah (Lintang pada id 5 memiliki tipe *string*).

test > {} File Salah.json > ...

```

1
2   {
3     "posList": [
4       {"id": "1", "nama": "Amerika Utara", "lintang": 42.58750064468325, "bujur": -101.25000000000001},
5       {"id": "2", "nama": "Amerika Selatan", "lintang": -21.283849397812993, "bujur": -56.953125},
6       {"id": "3", "nama": "Asia", "lintang": 58.09613019166115, "bujur": 87.18750000000001},
7       {"id": "4", "nama": "Afrika", "lintang": 7.793591434281613, "bujur": 29.53125000000004}],
8
9     "adjMatrix": [
10      [0,-1,-1,-1,4659.318],
11      [-1,0,-1,9960.477,-1],
12      [-1,-1,0,-1,4875.234],
13      [-1,9960.477,-1,0,-1],
14      [4659.318,-1,4875.234,-1,0]
15    ]
16  }

```

There might be inconsistency between the 'posList' and 'adjMatrix' data! Please check your file.

Path Finder

Insert File
Insert Relations
Shortest Path Searching
List of Nodes

Insert Test Case here!
Choose File File Salah.json
Load Map
Made a change on the map?
Save Current Map
Initial node
Final node
Process UCS Process A*

Start node
Final node
Insert Relation

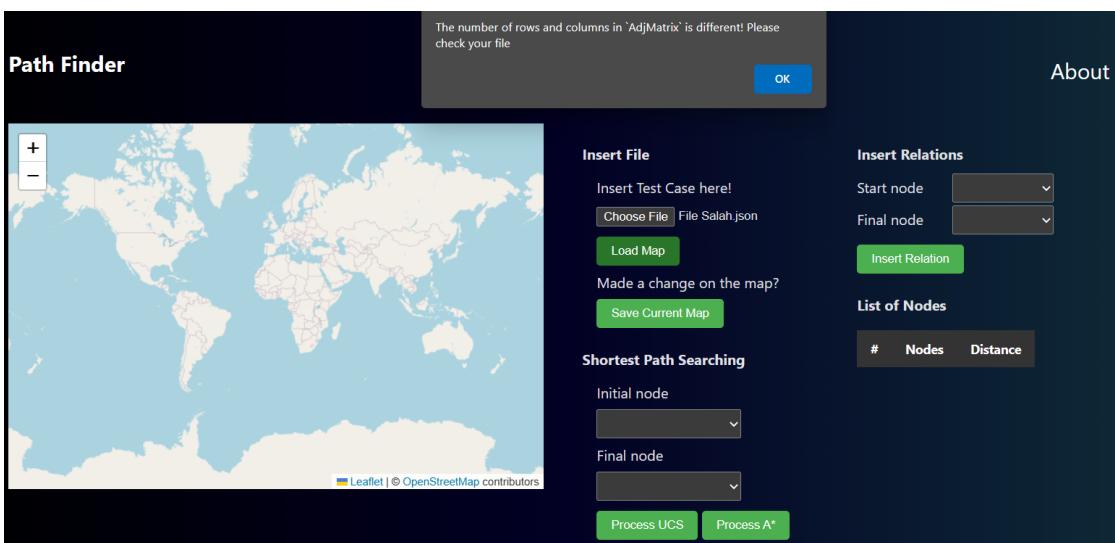
About

Gambar 4.5.8. Uji kasus jumlah simpul pada posList berbeda dengan adjMatrix.

```

test > {} File Salah.json > ...
1
2   {
3     "posList": [
4       {"id": "1", "nama": "Amerika Utara", "lintang": 42.58750064468325, "bujur": -101.25000000000001},
5       {"id": "2", "nama": "Amerika Selatan", "lintang": -21.283849397812993, "bujur": -56.953125},
6       {"id": "3", "nama": "Asia", "lintang": 58.09613019166115, "bujur": 87.18750000000001},
7       {"id": "4", "nama": "Afrika", "lintang": 7.793591434281613, "bujur": 29.531250000000004},
8       {"id": "5", "nama": "Greenland", "lintang": 72.82596352689488, "bujur": -37.96875000000001}],
9
10   "adjMatrix": [
11     [0, -1, -1, -1, 4659.318],
12     [-1, 0, -1, 9960.477, -1],
13     [-1, -1, 0, -1, 4875.234],
14     [-1, 9960.477, -1, 0, -1],
15     [4659.318]]
16 }

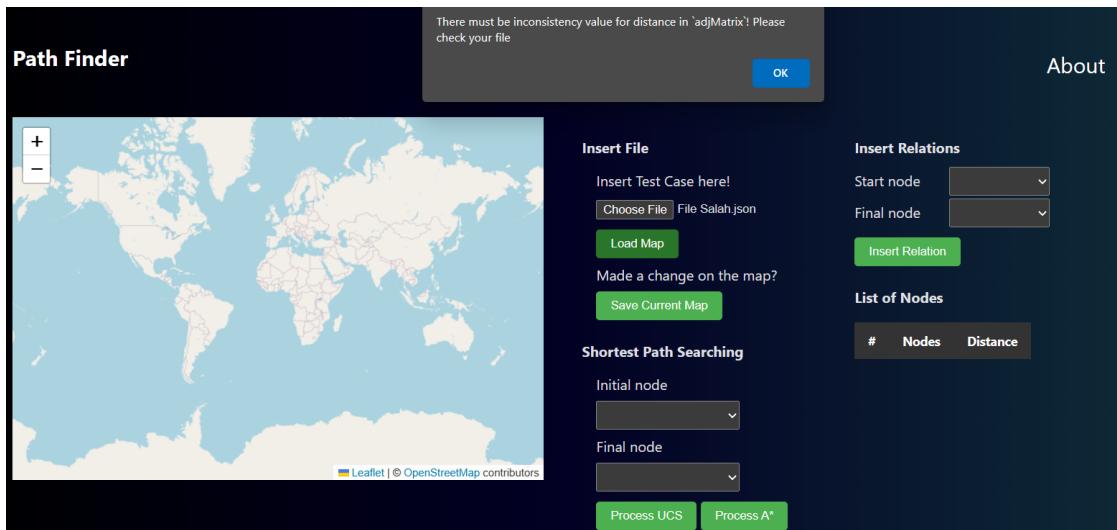
```



Gambar 4.5.9. Uji kasus jumlah baris dan kolom pada adjMatrix berbeda.

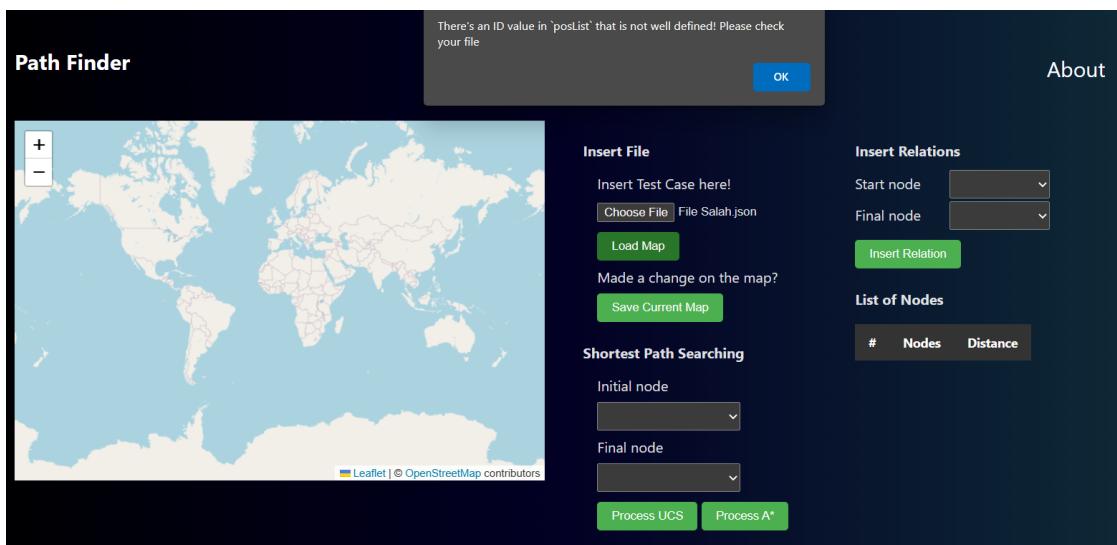
Program dibuat dengan membangun graf tidak berarah sehingga kasus tersebut dapat tertangani sebagai berikut.

```
test > {} File Salah.json > ...
1
2   {
3     "posList": [
4       {"id": "1", "nama": "Amerika Utara", "lat": 4659.318, "lon": -1, "order": 1},
5       {"id": "2", "nama": "Amerika Selatan", "lat": 9960.477, "lon": -1, "order": 2},
6       {"id": "3", "nama": "Asia", "lat": 4875.234, "lon": 0, "order": 3},
7       {"id": "4", "nama": "Afrika", "lat": 999999999, "lon": -1, "order": 4},
8       {"id": "5", "nama": "Greenland", "lat": 4875.234, "lon": 0, "order": 5}
9
10    "adjMatrix": [
11      [0, -1, -1, -1, 4659.318],
12      [-1, 0, -1, 9960.477, -1],
13      [-1, -1, 0, -1, 4875.234],
14      [-1, 999999999, -1, 0, -1],
15      [999999999, -1, 4875.234, -1, 0]
16    ]
17 }
```



Gambar 4.5.10. Uji kasus bobot pada adjMatrix tidak memenuhi graf tidak berarah.

```
test > {} File Salah.json > ...
1
2   {
3     "posList": [
4       {"id": "1", "nama": "Amerika Utara", "lintang": 42.3456789, "longt": -71.0589234, "order": 1},
5       {"id": "2", "nama": "Amerika Selatan", "lintang": -15.0, "longt": -52.0, "order": 2},
6       {"id": "3", "nama": "Asia", "lintang": 58.09613019, "longt": 95.72287654, "order": 3},
7       {"nama": "Afrika", "lintang": 7.793591434281613, "longt": 13.75623544821492, "order": 4},
8       {"id": "5", "nama": "Greenland", "lintang": 72.8259, "longt": -52.3687, "order": 5}
9   }
10  "adjMatrix": [
11    [0, -1, -1, -1, 4659.318],
12    [-1, 0, -1, 9960.477, -1],
13    [-1, -1, 0, -1, 4875.234],
14    [-1, 9960.477, -1, 0, -1],
15    [4659.318, -1, 4875.234, -1, 0]
16  ]
17}
```



Gambar 4.5.11. Uji kasus atribut pada posList tidak lengkap.

BAB V

LAMPIRAN

Repository Github:

https://github.com/mikeleo03/Tucil3_13521070_13521108

Tautan Pengunggahan Aplikasi Web :

Telah dilakukan *deployment* pada *website* yang telah dibangun pada tautan berikut.

<https://shortestpath-finder.netlify.app/>

Status Penyelesaian : *Completed*

No.	Poin	Ya	Tidak
1.	Program dapat menerima input graf	✓	
2.	Program dapat menghitung lintasan terpendek dengan UCS	✓	
3.	Program dapat menghitung lintasan terpendek dengan A*	✓	
4.	Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
5.	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓	